



*electronics*

Special Issue Reprint

---

# System-on-Chip (SoC) and Field-Programmable Gate Array (FPGA) Design

---

Edited by  
Nicola Lusardi

[mdpi.com/journal/electronics](https://mdpi.com/journal/electronics)



# **System-on-Chip (SoC) and Field-Programmable Gate Array (FPGA) Design**





# System-on-Chip (SoC) and Field-Programmable Gate Array (FPGA) Design

Guest Editor

**Nicola Lusardi**



Basel • Beijing • Wuhan • Barcelona • Belgrade • Novi Sad • Cluj • Manchester

*Guest Editor*

Nicola Lusardi  
Department of Electronics,  
Information and  
Bioengineering  
Politecnico di Milano  
Milano  
Italy

*Editorial Office*

MDPI AG  
Grosspeteranlage 5  
4052 Basel, Switzerland

This is a reprint of the Special Issue, published open access by the journal *Electronics* (ISSN 2079-9292), freely accessible at: [https://www.mdpi.com/journal/electronics/special\\_issues/GAH33B33WE](https://www.mdpi.com/journal/electronics/special_issues/GAH33B33WE).

For citation purposes, cite each article independently as indicated on the article page online and as indicated below:

Lastname, A.A.; Lastname, B.B. Article Title. <i>Journal Name</i> <b>Year</b> , Volume Number, Page Range.
--

**ISBN 978-3-7258-6101-9 (Hbk)**

**ISBN 978-3-7258-6102-6 (PDF)**

**<https://doi.org/10.3390/books978-3-7258-6102-6>**

© 2025 by the authors. Articles in this book are Open Access and distributed under the Creative Commons Attribution (CC BY) license. The book as a whole is distributed by MDPI under the terms and conditions of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) license (<https://creativecommons.org/licenses/by-nc-nd/4.0/>).

# Contents

About the Editor . . . . . vii

**Stefano Ricci, Stefano Caputo and Lorenzo Mucchi**

FPGA-Based Manchester Decoder for IEEE 802.15.7 Visible Light Communications

Reprinted from: *Electronics* **2025**, *14*, 96, <https://doi.org/10.3390/electronics14010096> . . . . . 1

**Mattia Morabito, Nicola Lusardi, Fabio Garzetti, Gabriele Fiumicelli, Gabriele Bonanno, Enrico Ronconi, et al.**

Optimal Implementation of Tapped Delay Line Time-to-Digital Converters in 20 nm Xilinx

UltraScale FPGAs

Reprinted from: *Electronics* **2024**, *13*, 4888, <https://doi.org/10.3390/electronics13244888> . . . . . 23

**Sérgio N. Silva, Mateus A. S. de S. Goldbarg, Lucileide M. D. da Silva and Marcelo A. C. Fernandes**

Real-Time Simulator for Dynamic Systems on FPGA

Reprinted from: *Electronics* **2024**, *13*, 4056, <https://doi.org/10.3390/electronics13204056> . . . . . 56

**Haobo Lv and Qiongzhi Wu**

An Energy-Efficient Field-Programmable Gate Array (FPGA) Implementation of a Real-Time Perspective-n-Point Solver

Reprinted from: *Electronics* **2024**, *13*, 3815, <https://doi.org/10.3390/electronics13193815> . . . . . 84

**Paulino Ruiz-de-Clavijo, German Cano-Quiveu, Jorge Juan, Manuel Jesus Bellidoo, Julian Viejo-Cortes, David Guerrero and Enrique Ostua**

NanoBoot: A Field-Programmable Gate Array/System-on-Chip Hardware Boot Loader for IoT Devices

Reprinted from: *Electronics* **2024**, *13*, 3731, <https://doi.org/10.3390/electronics13183731> . . . . . 101

**Jingwen Huang, Chaiyi Kuo, Sihuang Liu and Tao Su**

An Area-Efficient and Configurable Number Theoretic Transform Accelerator for Homomorphic Encryption

Reprinted from: *Electronics* **2024**, *13*, 3382, <https://doi.org/10.3390/electronics13173382> . . . . . 120

**Chaewoon Park, Seongjoo Lee and Yunho Jung**

FPGA Implementation of Pillar-Based Object Classification for Autonomous Mobile Robot

Reprinted from: *Electronics* **2024**, *13*, 3035, <https://doi.org/10.3390/electronics13153035> . . . . . 142

**Binh Kieu-Do-Nguyen, Khai-Duy Nguyen, Tuan-Kiet Dang, Nguyen The Binh, Cuong Pham-Quoc, Ngoc-Thinh Tran, et al.**

A Trusted Execution Environment RISC-V System-on-Chip Compatible with Transport Layer Security 1.3

Reprinted from: *Electronics* **2024**, *13*, 2508, <https://doi.org/10.3390/electronics13132508> . . . . . 158

**Dongmin Jeong, Myeongjin Lee, Wookyoung Lee and Yunho Jung**

FPGA-Based Acceleration of Polar-Format Algorithm for Video Synthetic-Aperture Radar Imaging

Reprinted from: *Electronics* **2024**, *13*, 2401, <https://doi.org/10.3390/electronics13122401> . . . . . 182

**Seongmo An, Jongwon Oh, Sangho Lee, Jinyeol Kim, Youngwoo Jeong, Jeongeun Kim and Seung Eun Lee**

Lightweight and Error-Tolerant Stereo Matching with a Stochastic Computing Processor

Reprinted from: *Electronics* **2024**, *13*, 2024, <https://doi.org/10.3390/electronics13112024> . . . . . 202

<b>Hsin-Chen Lu, Liang-Ying Su and Shih-Hsu Huang</b> Highly Fault-Tolerant Systolic-Array-Based Matrix Multiplication Reprinted from: <i>Electronics</i> <b>2024</b> , <i>13</i> , 1780, <a href="https://doi.org/10.3390/electronics13091780">https://doi.org/10.3390/electronics13091780</a> . . . . .	<b>219</b>
<b>Fabio Garzetti, Gabriele Bonanno, Nicola Lusardi, Enrico Ronconi, Andrea Costa and Angelo Geraci</b> New High-Rate Timestamp Management with Real-Time Configurable Virtual Delay and Dead Time for FPGA-Based Time-to-Digital Converters Reprinted from: <i>Electronics</i> <b>2024</b> , <i>13</i> , 1124, <a href="https://doi.org/10.3390/electronics13061124">https://doi.org/10.3390/electronics13061124</a> . . . . .	<b>236</b>
<b>Nikolaos Bartzoudis, José Rubio Fernández, David López-Bueno, Antonio Román Villarroel and Angelos Antonopoulos</b> Agile FPGA Computing at the 5G Edge: Joint Management of Accelerated and Software Functions for Open Radio Access Technologies Reprinted from: <i>Electronics</i> <b>2024</b> , <i>13</i> , 701, <a href="https://doi.org/10.3390/electronics13040701">https://doi.org/10.3390/electronics13040701</a> . . . . .	<b>257</b>

# About the Editor

## Nicola Lusardi

Nicola Lusardi was born in Piacenza on November 25th, 1990. He received his BSc and MSc degrees with honors in Electronic Engineering, as well as a PhD in Information Technology, from Politecnico di Milano. His research focuses on digital architectures on configurable devices, with particular emphasis on state-of-the-art systems for time measurement. He is affiliated with the Italian National Institute for Nuclear Physics (INFN), and he is a member of the IEEE Nuclear and Plasma Sciences Society (NPSS). He is also responsible for FPGA-based electronics in the DSSC experiment at the European XFEL. In 2020, he was awarded the IEEE NPSS Glenn F. Knoll Postdoctoral Education Grant. He serves as a referee for several international journals and is the author of more than one hundred peer-reviewed scientific publications. Additionally, he has contributed as an electronics engineer to various collaborative experiments with the Delft University of Technology (TU Delft, Netherlands), the LHCb experiment at CERN (Geneva), the École Polytechnique Fédérale de Lausanne (EPFL, Switzerland), and the Elettra Synchrotron Group in Trieste (Basovizza, TS). In 2019, he co-founded TEDIEL Srl, an innovative start-up and spin-off of Politecnico di Milano, based on a novel Time-to-Digital Converter (TDC) architecture entirely developed on latest-generation FPGA and SoC devices. The company operates in high-tech application areas including Smart Mobility, Industry 4.0, Health, and Environmental Monitoring. He has led several R&D projects focused on product innovation through digital electronics, in collaboration with both private and public sector partners (e.g., RFI, Cordon Electronics Italia, CAEN). He is currently an Associate Professor in the Electronics Section of Politecnico di Milano.





## Article

# FPGA-Based Manchester Decoder for IEEE 802.15.7 Visible Light Communications

Stefano Ricci \*, Stefano Caputo and Lorenzo Mucchi

Information Engineering Department, University of Florence, 50121 Florence, Italy; stefano.caputo@unifi.it (S.C.); lorenzo.mucchi@unifi.it (L.M.)

\* Correspondence: stefano.ricci@unifi.it

**Abstract:** Visible Light Communication (VLC) is a cutting-edge transmission technique where data is sent by modulating light intensity. Manchester On–Off Keying (OOK) is among the most used modulation techniques in VLC and is normed by IEEE 802.15.7 standard for wireless networks. Various Manchester decoder schemes are documented in the literature, often leveraging minimal two-level analog-to-digital converters followed by straightforward digital logic. These methods often compromise performance for simplicity. However, the VLC applications in fields like automotive and/or aerospace require the maximum performance in terms of bit error rate (BER) with respect to Signal-to-Noise Ratio (SNR), together with a real-time low-latency implementation. In this work, we introduce a high-performance Manchester decoder and detail its implementation in a Field Programmable Gate Array (FPGA). The decoder operates by acquiring a fully resolved signal (12-bit resolution) and by calculating the phase of the transmitted bit. Additionally, the proposed decoder achieves and maintains synchronization with the incoming signal, tolerating frequency shifts and jitter up to 1%. The Manchester decoder was tested in a VLC system with automotive-certified headlamps, realizing an IEEE 802.15.7-compliant link at 100 kb/s. The proposed decoder ensures a BER below  $10^{-2}$  for  $\text{SNR} > -12$  dB and, compared to a standard decoder, achieves the same BER when the input signal has an SNR of 10 dB lower.

**Keywords:** Manchester coding; IEEE 802.15.7; Visible Light Communication (VLC); Field Programmable Gate Array (FPGA); bit error rate (BER); automotive VLC applications; real-time VLC

## 1. Introduction

Visible Light Communications (VLCs) represents nowadays a novel and promising wireless technique for sending and receiving information in short-range links [1–3]. VLC works by modulating the light intensity of a transmitting source, which typically is a Light Emitting Diode (LED). This technology has gained momentum by the recent diffusion on the market of white LED lamps designed for ambient lighting, which have proven effective as VLC transmitters [4].

The performance of a VLC link is limited by the Signal-to-Noise Ratio (SNR) present at the receiver, which imposes a tradeoff between the achievable data rate and the distance between the transmitter (TX) and the receiver (RX). Research showed that it is possible to establish communication in the Gb/s range (for example Cossu et al. showed a 3.4 Gb/s link [5]), or in a vehicle-to-vehicle (V2V) communication link, the maximum transmission distance is approximately 72 m in clear weather and 26 m in fog [6].

The information needs to be modulated before being transmitted through light. The highest performance in terms of efficiency in the use of bandwidth is obtained by applying complex modulations such as Orthogonal Frequency-Division Multiplexing (OFDM) plus Quadrature Amplitude Modulation (QAM) [7]. However, the implementation of these techniques requires electronics systems capable of a relatively high calculation power and TX/RX front-ends with elevated analog performance. On the other hand, in the On–Off Key (OOK) modulation [8], the bits are coded in only two different intensities of the light. This approach is very simple, since in TX the LED can be directly driven by digital devices [9], and in RX even a simple threshold receiver can work [10]. Despite the simplicity of this method, OOK modulation can achieve a relatively high performance [11].

An important variation of the OOK modulation is the Manchester coding. In Manchester OOK coding, the bits ‘0’ and ‘1’ are transmitted through the symbols couples ‘01’ and ‘10’, respectively (or the opposite) [12]. This choice grants a transition for every bit, which is very useful to recover the clock at the receiver and permits a direct-current (DC) balance of the channel. The positive features of this approach are confirmed by its inclusion in important standards such as the IEEE 802.15.7 [13] for wireless communications, which applies, for example, to automotive VLC applications [14,15].

### 1.1. Manchester Receivers in the Literature

While the Manchester OOK transmitter is easy to implement, the receiver is less straightforward and requires a bit more of electronics. The Manchester receivers present in the literature (see Table 1) typically share the same front-end based on a two-level analog-to-digital (AD) converter, realized by comparing the input signal to a low-pass filtered replica [16]. The works present in the literature are mainly focused on the Clock Data Recover (CDR) circuit that follows the two-level AD converter. The most used approaches are based on a Phase-Locked Loop (PLL) or, alternatively, exploit the so called “blind oversampling”.

In the first approach, a PLL is synchronized to the incoming signal and used to generate a clock with a phase suitable to sample the data [16,17]. This approach can achieve very high frequencies, and it is used in Application-Specific Integrated Circuits (ASICs) (for example, Ethernet receivers). On the other hand, it is not compatible to implementation in the general fabric architecture of a FPGA.

On the other hand, the “blind oversampling” method is often the preferred approach for FPGA implementation. An unsynchronized clock, whose frequency is at least two times the bit rate [18], samples the data with different phases. A following digital logic locates the edges and reconstruct the bits [19–25]. The performance of this method depends on the oversampling ratio and/or the number of phases of the clock, and is quite sensitive to jitter [26].

The aforementioned methods can be used for rates in the order of Gb/s. For lower rates, such as those at the reach of microcontrollers ( $\mu\text{C}$ ), the Manchester code is typically decoded by counting the time intervals between edges of the incoming signal by the means of  $\mu\text{C}$  peripherals timers [10]. This approach is quite simple, works for rates up to some hundreds of kb/s, but has limited immunity to noise.

**Table 1.** Main Manchester receiver methods.

Method	AD Conv.	Rates	Device	Resources	BER	Ref.
PLL	1-bit	>1 Gb/s	ASIC	Low	Medium	[16,17]
blind ‘over-sampling’	1-bit	>1 Gb/s	FPGA, ASIC	Medium	Medium	[19–25]

Table 1. Cont.

Method	AD Conv.	Rates	Device	Resources	BER	Ref.
uC-based	1-bit	<200 kb/s	uC	Low	Medium	[10,27]
proposed	12-bit	<1 Gb/s	FPGA, ASIC	High	Low	This Study

### 1.2. Our Contribution

The weak point of the Manchester decoders described above is the very simple two-level front-end, which results in limited noise immunity. Input noise can easily surpass the threshold, leading to false commutations and data errors. Additionally, implementing delays through digital gates is not advisable, and overall tolerance to frequency errors is limited. In case of rates of Gb/s, this choice is related to the excessive complication and cost of a high-resolution and high velocity AD converter, together with the following digital data processing.

On the other hand, for lower data rates, the improvement of the front-end will result in a performance gain, although at the expense of complexity. Data produced by high-resolution AD converters, even in case of a relatively low rate, cannot be processed by a  $\mu\text{C}$  [27], and a more complicated FPGA system would be required. However, applications exist where the performance of a VLC link is more important than simplicity or cost. These include, for example, aerospace and automotive fields. It is feasible to imagine a situation placed in a near future where a reliable, low-latency data exchange between two cars equipped for VLC communication, one blocked and the other fast approaching in the same lane, can automatically trigger the breaks of the second car and be crucial in saving lives [28].

In this work we propose a high-performance Manchester decoder implemented in a Field Programmable Gate Array (FPGA), specifically dedicated to demanding VLC applications that implements the IEEE 802.15.7 standard in real time. It accepts input data from a 12-bit analog-to-digital converter at 10 Msps, and it processes these data by performing a phase analysis over the samples acquired in a bit-time. In other words, the Manchester code is decoded like a special case of a Binary Phase-Shift Keying (BPSK) modulation [29]. The decoder calculates the phase by applying the discrete Fourier transform (DFT) to the single frequency component corresponding to the bit period. As will be clarified in the following, it compares to the existing methods such as those reported in the last row of Table 1: it achieves high noise immunity at the cost of a more demanding architecture.

The architecture of the proposed decoder is detailed in Section 2. In Section 3 the proposed architecture is validated and shown more robust to noise with respect to the simple Manchester decoder typically employed. Then, in Section 4, the decoder is implemented in the FPGA of a system designed for VLC research [30], and tailored for the 100 kb/s rate provided in the IEEE 802.15.7 standard.

In the experiments, reported in Section 5, the FPGA-based VLC system was connected to a 16 W white LED headlamp certified for automotive lighting. The tests showed that the decoder receives with no error when the signal amplitude is higher than  $-30$  dB with respect to the input dynamics, with a bit error rate (BER) [31] lower than  $10^{-3}$  as long as the signal is higher than  $-47$  dB.

## 2. Architecture of the Receiver

### 2.1. The Method

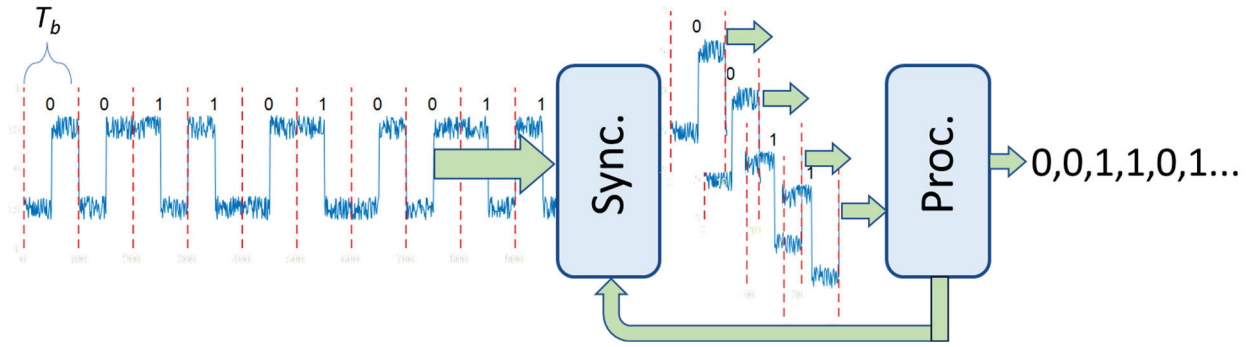
The signal modulated according to the Manchester code is a 2-level signal like that represented in the example of Figure 1. In the example, a white noise is added (with a SNR

of 30 dB) to the signal to give a more realistic representation. Each bit is transmitted for a time  $T_b$ . Formally, the signal can be expressed as:

$$s(t) = \sum_{n_b=0}^{N_b-1} b_{n_b} \cdot \text{one}(t - n_b T_b) + (1 - b_{n_b}) \cdot \text{zero}(t - n_b T_b) \quad (1)$$

where  $N_b$  is the number of transmitted bits,  $b_{n_b}$  is the value (0 or 1) of the bit in position  $n_b$ , and the functions  $\text{one}(t)$  and  $\text{zero}(t)$  represent the single bits of amplitude  $A$ :

$$\text{one}(t) = A \cdot \text{rect}\left(\frac{t - 3T_b/4}{T_b/2}\right); \quad \text{zero}(t) = A \cdot \text{rect}\left(\frac{t - T_b/4}{T_b/2}\right); \quad (2)$$



**Figure 1.** The “Sync.” block cuts the signal in slices of  $N$  samples; the “Proc.” block processes the slices for detecting the bits, and produces a feedback to Sync. which maintains the correct synchronism with respect to the bit boundaries.

The signal is AD converted with a rate  $f_c$ , generating the sequence  $s_i = s(i/f_c)$ . Each bit is composed by  $N = T_b \cdot f_c$  samples. It is convenient—but not mandatory—that the sampling frequency  $f_c$  is a multiple of the transmission rate  $f_s = 1/T_b$ . The input signal is subdivided into slices  $sl$  of  $N$  sample each, so that each slice of index  $h$  lasts for the duration of a bit:

$$sl_h = [s_{h \cdot N + of}, s_{h \cdot N + 1 + of}, \dots, s_{N(h+1) - 1 + of}]. \quad (3)$$

Please note that in the bit boundaries (i.e., the first sample of each bit) are located in the samples  $s_{i \cdot N}$  for each  $i = 0, 1, 2, 3$ , etc., while the slices  $sl_h$  in (3) are aligned to the sample  $s_{h \cdot N + \xi of}$ , i.e., the slice, in general, is not aligned to the bit boundaries, but is affected by an offset of  $\xi of$  samples.

In the representation of Figure 1, the subdivision of the sequence  $s_i$  in the slices  $sl_h$  is performed by the “Sync” block. The “Sync” block has no a priori knowledge on where the bit boundaries are located along the signal, and cutting the slices so that they are centered on the true bit boundaries ( $\xi of = 0$ ) is not a trivial task. This is a general problem that most of the receivers must face, and it is known as “synchronization”. A wide literature is present that investigates possible solutions that apply to different modulations [32]. We will return to this later; for now, let us imagine that the slices are perfectly centered over the bits, i.e., the receiver is synchronized and  $\xi of = 0$ .

## 2.2. The Bit Decision

The slices are processed to detect the bit value and to extract the information necessary to maintain the synchronization. In the proposed approach, the signal is processed like it was modulated according to a BPSK modulation, and Manchester code is indeed a specific case of BPSK. Thus, the phase  $\varphi$  of the signal actually present in each slice  $sl_h$  is calculated.

The slice represents a square wave (see Figure 1) at frequency  $f_s$ , when the communication rate is  $f_s$  bit/s. The square wave, in the frequency domain, is characterized by the fundamental frequency at  $f_s$  with harmonics at multiple frequencies. The amplitude of the harmonics decreases with the increasing frequency, being the component at  $f_s$  characterized by the highest amplitude. Thus, it is reasonable to suppose that the component at  $f_s$  features the highest SNR. We will extract the phase  $\varphi$  from this component (see Equation (7)).

Let us start by calculating the complex spectrum of the slice  $sl_h$  composed by the  $N$  samples as given in (3), by exploiting the discrete Fourier transform (DFT) [33]:

$$I_k = \sum_{i=0}^{N-1} s_{h \cdot N + of + i} \cdot \cos\left(2\pi \frac{k}{N} i\right); \quad Q_k = - \sum_{i=0}^{N-1} s_{h \cdot N + of + i} \cdot \sin\left(2\pi \frac{k}{N} i\right); \quad (4)$$

$$0 \leq k \leq N - 1$$

In (4),  $I_k$  and  $Q_k$  represent the In-phase and Quadrature-phase (IQ) components of the spectrum for the generic frequency  $f_k = k \cdot f_c / N$ , where  $f_c$  is the sampling frequency. Since  $N = T_b \cdot f_c = f_c / f_s$ , the component of our interest at frequency  $f_s$  has the index:

$$k = \frac{f_s}{f_c} N = \frac{f_s}{f_c} T_b f_c = \frac{f_s}{f_c} \frac{f_c}{f_s} = 1. \quad (5)$$

Equation (4), rewritten for  $k = 1$ , is:

$$I_1 = \sum_{i=0}^{N-1} s_{h \cdot N + of + i} \cdot \cos\left(2\pi \frac{i}{N}\right); \quad Q_1 = - \sum_{i=0}^{N-1} s_{h \cdot N + of + i} \cdot \sin\left(2\pi \frac{i}{N}\right) \quad (6)$$

From here on  $I_1$  and  $Q_1$  are named  $I$  and  $Q$  for brevity. In the next step, the phase  $\varphi$  is calculated through the four-quadrant arctangent function  $\text{atan2}(Q, I)$  that can be defined as:

$$\varphi = \text{atan2}(Q, I) = \begin{cases} \text{atan}\left(\frac{Q}{I}\right) & I > 0 \\ \pi - \text{atan}\left(-\frac{Q}{I}\right) & I < 0 \\ \frac{\pi}{2} & I = 0; Q > 0 \\ -\frac{\pi}{2} & I = 0; Q < 0 \end{cases} \quad (7)$$

where  $\text{atan}(x)$  is the inverse tangent of  $x$ . In case of an ideal signal coded according to Manchester modulation of amplitude  $\pm A$ , the slice holding bit '0' (transition from symbol 0 to symbol 1) has  $I = 0$ ,  $Q = 2A$ , and  $\varphi = \frac{\pi}{2}$ ; and the slice holding the bit '1' (transition from symbol 1 to symbol 0) has  $I = 0$ ,  $Q = -2A$ , and  $\varphi = -\frac{\pi}{2}$ , as summarized in Table 1.

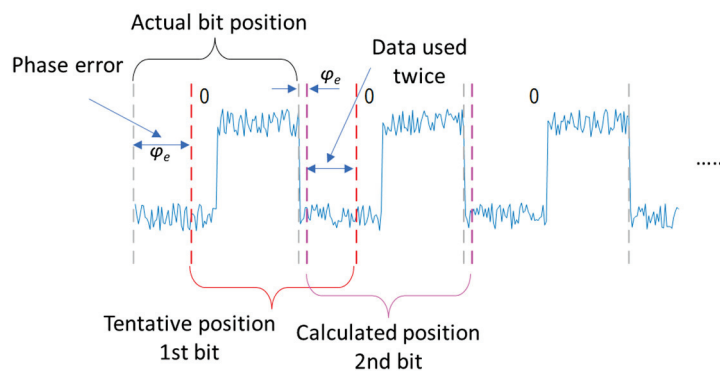
The decision on the value of the bit can be easily taken by considering the sign of the phase  $\varphi$ . However, it is apparent that the same decision can be obtained by looking at the sign of  $Q$ , without the need of calculating  $I$  or the phase  $\varphi$ . However, the calculation of the phase is required for the synchronization of the receiver, as it will be discovered in the next section.

### 2.3. The Synchronization of the Receiver

The problem of transmitter–receiver synchronization is well known in the field, and several techniques have been proposed in the literature [32]. As anticipated, the receiver has no a priori knowledge on where the borders of the bits are located along the incoming data sequence. It is necessary to implement some mechanism that finds the positions of the borders and maintains the sequence dynamically synchronized. In fact, even if the transmitter and receiver share the same nominal communication rate, unavoidable differences on their local oscillators make the calculation of the timings in the two apparatuses not perfectly equal. The result is that, without correction, the time-markers of the

receiver are likely to accumulate a phase error with respect to the data produced by the transmitter. Moreover, this phase error increases in time and eventually prevents a correct communication [15].

In this application, we use the phase  $\varphi$ , calculated by (7), to locate the bit boundaries at the start of the transmission, and then to dynamically maintain the right position during the remaining part of the communication. To better understand the process, let us imagine that the receiver places the tentative borders of the bit like depicted in Figure 2 (red-dashed lines). In this example the bit transmitted has a '0' value and the tentative bit-slice is placed with a phase error  $\varphi_e$ . The error can be due to noise on the signal or simply because this is the first transmitted bit, and thus the receiver has no history to help in locating the right position.



**Figure 2.** The receiver tentatively places the borders of the first bit in the position delimited by the red-dashed segments. The phase error with respect to the true bit is calculated and used to correct the border positions (magenta dashed lines) for the 2nd bit.

The data-slice delimited by the red interval is processed for phase calculation as detailed in the previous sections. In this condition, from (7), the resulting phase  $\varphi$  is not  $90^\circ$  like it should be ideally for the '0' value of the bit (see Table 2), but it results around  $\varphi = -150^\circ$ . This info is precious for the receiver that can calculate the phase error:  $\varphi_e = -150^\circ - 90^\circ + 180^\circ = -60^\circ$ . Now the receiver knows that the borders were placed with a delay of  $60^\circ$ . Thanks to this knowledge, the error is corrected, and the next bit can be located with much higher accuracy, as shown in Figure 2. From now on the receiver is synchronized and dynamically maintains the lock by correcting the error  $\varphi_e$  at every step. At this point the error is due only to the noise on the signal, and the receiver maintains the synchronization as long as the SNR is sufficiently high.

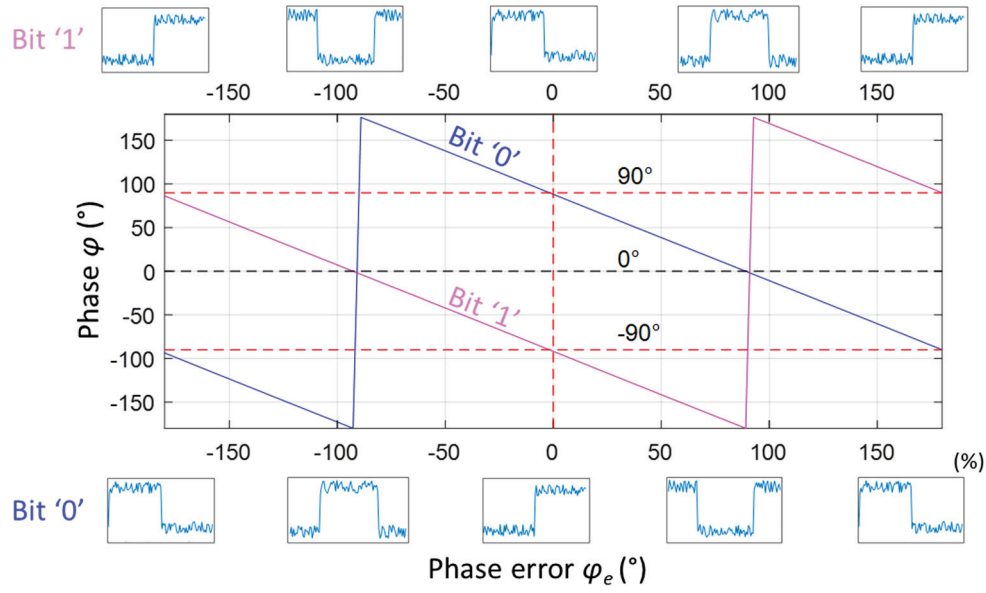
**Table 2.** Values of  $I$ ,  $Q$ , and phase for an ideal Manchester signal of  $\pm A$  levels.

BIT	Symbols	I	Q	Phase
0	01	0	$+2A$	$+90^\circ$
1	10	0	$-2A$	$-90^\circ$

It should be noted that in the example of Figure 2, the last samples of the red slice (1st tentative bit) are used in the calculation of the phase (7) for the 1st bit, and, after the correction, are used again in the calculation of the phase for the 2nd bit. Similarly, it happens that when the error is positive (advance tentative position), some samples are not used at all.

The example of Figure 2 can be extended to analyze the general condition. Figure 3 shows how the phase  $\varphi$  calculated by (7) changes according to the phase error  $\varphi_e$ . The graph reports the cases for bit of value '0' and '1'.





**Figure 3.** Relation between the position of the rising edge with respect to the tentative bit start (phase error  $\varphi_e$ ) and the phase  $\varphi$  calculated through (7). Blue and magenta curves refer to the bit values of '0' and '1', respectively. A selection of waveforms are reported on top (bit '1') and bottom (bit '0') of the graph in the position of the phase error they represent.

Given a phase  $\varphi$ , Table 3 summarizes the decision about the bit value and the estimation of the error  $\varphi_e$  to be used in the positioning of the next bit-slice. Given that  $s_i$  is the first sample of the current slice, affected by the phase error  $\varphi_e$ , the next positioning is achieved by starting the reading of the slice from the sample  $s_{i+l}$ , where  $l$  is obtained as:

$$l = \text{int} \left[ T_b f_c \left( 1 + \frac{\varphi_e}{360^\circ} \right) \right] = N + \text{int} \left( \frac{\varphi_e}{360^\circ} N \right) = N + z \quad (8)$$

**Table 3.** Bit decision and phase error derived from calculated phase.

Calculated Phase $\varphi$	Bit Decision	Error Estimation $\varphi_e$
$-180^\circ \leq \varphi < 0^\circ$	1	$\varphi + 90^\circ$
$0^\circ \leq \varphi < 180^\circ$	0	$90^\circ - \varphi$

In (8)  $\varphi_e$  is expressed in deg ( $^\circ$ ),  $\text{int}(x)$  is the nearest integer to  $x$ , and  $N$  represents the number of samples per bit. The increment  $l$  is composed by  $N$ , i.e., the bit length, plus  $z$  that can be positive or negative depending on the sign of  $\varphi_e$ , and ranges in  $\pm N/2$ .

In a practical implementation, it can be beneficial to reduce the variability of  $z$  among subsequent bits. This precaution can avoid, for example, that a sudden noise in a single bit could compromise the synchronization. A simple approach, whose efficacy will be discussed in Section 3, consists in allowing for the next bit a phase correction  $z$  not above a given percentage of  $N$ .

From Figure 3 we see that, as long as the phase error  $\varphi_e$  is between  $-90^\circ$  and  $+90^\circ$ , the decision about the bit value is correct; on the other hand, a higher phase error results in a wrong bit value, an incorrect calculation of the phase error  $\varphi_e$ , and thus in an incorrect positioning of the next bit. In practice, a phase error higher than  $\pm 90^\circ$  causes the receiver to lock over the second commutation edge of the bit, which is an unwanted condition. However, once the correct synchronization is achieved, it is easily maintained as long as the SNR is sufficiently high. Unfortunately, at the beginning of the sequence the synchronization



has still to be achieved: the receiver has to “guess” the position of the first bit without any other info (see Figure 2), and a phase error higher than  $\pm 90^\circ$  can happen.

The issue is solved by transmitting a known “synchronization” sequence of some bits at the beginning of the data packet (for example, eight zeroes). With the knowledge of the bit value, the receiver can correctly calculate the phase error  $\varphi_e$  (see Figure 3) over the full  $\pm 180^\circ$  range, thus achieving the correct synchronization.

#### 2.4. Approximation of the Inverse Tangent

The calculation of the phase (7) requires the inverse tangent. In general, the real-time calculation of the inverse tangent is not a trivial task. Several algorithms have been reported in the literature that produce approximations with different accuracies and calculation efforts. Readers interested in the subject can find some different implementing techniques for example in [34–37], but several others are present in the literature.

In order to optimize the effort, let us evaluate the accuracy required for the calculation of the inverse tangent in this application. As detailed in the previous section, the phase error  $\varphi_e$  is used by the receiver to correct the slice position on the next bit by applying (8). The quantized nature of  $l$ , which must be integer (8), imposes a limitation to the requirements of resolution in the calculation of  $\varphi_e$ , corresponding to the minimum variation  $\Delta\varphi_e$  of the phase  $\varphi_e$  that occurs for a unity variation of  $l$ . The same applies for  $\Delta\varphi$  in the calculation of  $\varphi$ . In other words, it is a waste to calculate the inverse tangent with a grade of resolution higher with respect to the minimum that changes the result of the rounding in (8), namely  $\text{int}(\frac{\varphi_e}{360^\circ} N)$ . This reasoning ends in the following reference for evaluating the resolution  $\Delta\varphi$ :

$$\frac{\Delta\varphi_e}{360^\circ} N = 1 \Rightarrow \Delta\varphi_e = \frac{360^\circ}{N} \Rightarrow \Delta\varphi = \frac{360^\circ}{N} \quad (9)$$

Another useful constraint is that the approximation produces no error when the system is synchronized, so the lock condition is effectively maintained.

Following the reasons discussed so far, we approximated the four-quadrant inverse tangent through a very simple linear relation that crosses the origin. As will be clearer in the following, the “origin crossing” grants the condition of no error in lock condition. Formally we have:

$$\varphi = \text{itanpp}(x) = m \cdot x \quad (10)$$

where  $\text{itanpp}(x)$  is the approximate inverse tangent function, and  $m$  is the value that minimizes the maximum error in the octant:

$$\min_m \left\{ \max_x \left| m \cdot x - \frac{180^\circ}{\pi} \text{atan}(x) \right| \right\} \quad 0 \leq x < 1 \quad (11)$$

After some basic steps of mathematical analysis, here omitted for brevity, we obtain the following closed expression for  $m$  that minimizes (11):

$$\frac{180^\circ}{\pi} \text{atan} \left( \sqrt{\frac{\frac{180^\circ}{\pi} - m}{m}} \right) - m \left( \sqrt{\frac{\frac{180^\circ}{\pi} - m}{m}} \right) - m + 45^\circ = 0 \quad (12)$$

The numerical solution of Equation (12) is  $m \approx 47.7434^\circ$ .

In addition to the approximation of Equation (10) with  $m$  given by Equation (12), it is worth of considering also the further approximation of the solution of Equation (12) for  $m = 45^\circ$ . In fact, being  $45^\circ = 360^\circ/2^3$ , the hardware implementation of this second approximation is particularly efficient, as will be clarified in Section 4.3.

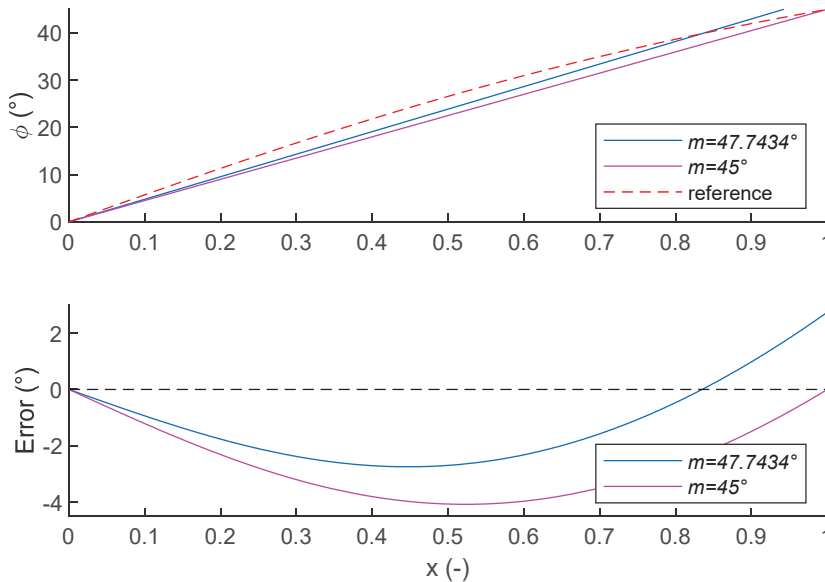
Once the  $\text{atan}(x)$  is approximated inside an octant, it is trivial to extend the result to the quadrant by applying  $\text{atan}(1/x) = 90^\circ - \text{atan}(x)$ , and then to the full four-quadrant co-domain by evaluating the signs of the I, Q components. The steps are the following:

- (1) The octant is determined from the absolute values of I and Q (see Table 3, where octants are numbered in anti-clockwise order starting from  $0^\circ$ );
- (2) We calculate (10) with  $x = \min(|I|, |Q|)/\max(|I|, |Q|)$ . This way we approximate the inverse tangent or cotangent depending on the angle behaving to an even or odd octant; referring the result to the zero octant ( $0 \leq \varphi < 45^\circ$ );
- (3) The result is rotated back to the octant calculated in point 1 with the simple operation reported in the last column of Table 4. Like the ideal four-quadrant extension of  $\text{atan}(x)$ , the approximation of Table 4 is undefined when both I and Q are null.

**Table 4.** Calculation of the approximated 4-quadrant atan.

Sign(I)	Sign(Q)	I   Q	Octant	Angle	$\varphi$
+	+	I  >  Q	0	$0 \leq \varphi < 45^\circ$	$0^\circ + m \cdot  Q / I $
+	+	I  ≤  Q	1	$45^\circ \leq \varphi < 90^\circ$	$90^\circ - m \cdot  I / Q $
-	+	I  <  Q	2	$90^\circ \leq \varphi < 135^\circ$	$90^\circ + m \cdot  I / Q $
-	+	I  ≥  Q	3	$135^\circ \leq \varphi < 180^\circ$	$180^\circ - m \cdot  Q / I $
-	-	I  <  Q	4	$-180^\circ \leq \varphi < -135^\circ$	$-180^\circ + m \cdot  I / Q $
-	-	I  ≥  Q	5	$-135^\circ \leq \varphi < -90^\circ$	$-90^\circ - m \cdot  Q / I $
+	-	I  ≤  Q	6	$-90^\circ \leq \varphi < -45^\circ$	$-90^\circ + m \cdot  I / Q $
+	-	I  >  Q	7	$-45^\circ \leq \varphi < 0^\circ$	$0^\circ - m \cdot  Q / I $

Figure 4 compares the  $\text{atan}(x)$  approximations with  $m = 47.7434$  (blue curve) and  $m = 45^\circ$  (magenta curve) as calculated in Equation (9), and the reference function (red-dashed curve). On top of the picture, the three trends are reported; on the bottom panel, the phase errors  $\varphi_e$ , calculated with respect to the reference inverse tangent, are highlighted. The graph reports the data for the first octant ( $0 \leq x < 1$ ) only, since they repeat unchanged in the remaining octants.



**Figure 4.** Top: Comparison between inverse tangent (red-dashed curve) and proposed approximations (blue and magenta curves) in the first octant, where the atan argument  $x$  ranges from 0 to 1. Bottom: Error of proposed approximations.

We can see that with  $m = 47.74^\circ$  the error ranges between  $\pm 2.74^\circ$ , while with  $m = 45^\circ$  the error is between  $-4.07^\circ$  and  $0^\circ$ . In addition to the maximum error, it should be noted

that once the receiver is synchronized, the phase will be around  $\pm 90^\circ$ , which corresponds to the origin of the octant in both cases (see Table 4). In the origin, the error is null, and remains reasonably low even near the origin. As discussed before, this behavior is important for the system to correctly maintain the lock condition.

As we anticipated, the error calculated here should be related through Equation (9) to the number of samples per bit  $N$ . From Equation (9), we deduce that the accuracy granted by the proposed approximations, considering the maximum errors, corresponds to values of  $N \approx 130$  and  $N \approx 90$  for  $m = 47.74^\circ$  and  $m = 45^\circ$ , respectively. However, this is a quite conservative estimation, and from a practical point of view,  $N \approx 100$  with  $m = 45^\circ$  represents a good compromise.

### 3. Evaluation of the Receiver Performance in Simulations

#### 3.1. Evaluation of How the Saturation of $z$ Affects the Receiver Performance

In Section 2.3, when we discussed Equation (8), we mentioned the convenience to limit the value of  $z$  in order to reduce the possible jitter in phase jumps among subsequent bits. The value  $z$  in Equation (8) ranges in  $\pm N/2$ , thus we can limit the ‘jumps’ to a percentage  $Z\%$  of the maximum range  $N/2$ . Formally:

$$z = \begin{cases} \text{sign}(z) \cdot Z\% N/2 & |z| > Z\% N/2 \\ z & |z| \leq Z\% N/2 \end{cases} \quad (13)$$

In this test we investigate how  $Z\%$  affects the receive performance. The algorithm described so far was implemented in MATLAB R2024a (The Mathworks, Natick, MA, USA). In input to the receiver, a signal was prepared that included a synchronization sequence of 100 bit of zeros (01 symbols), followed by a payload of  $10^6$  bits generated randomly. At the very beginning of the sequence, a period of noise was added. It lasted for a random length between 0 and  $T_b$ . The aim was generating an initial phase difference between the transmission and the reception timings. According to IEEE 802.15.7 standard, we simulated  $T_b = 10 \mu\text{s}$ , corresponding to 100 kbit/s. The signal was sampled at  $f_c = 10 \text{ Msps}$ , thus we had  $N = 100$ .

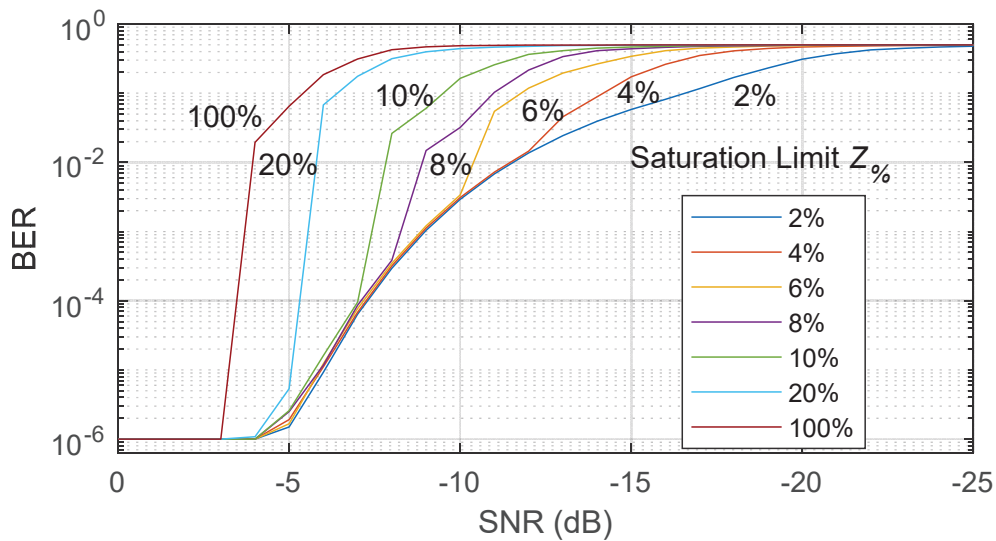
The relatively long extent of the synchronization sequence used here was necessary to test low values of  $Z\%$ : in fact, with  $Z\% = 2\%$  the receiver can adjust for a maximum of one sample per bit, and thus, in the worst case, needs 100 bits to synchronize.

A white noise was added to the signal to achieve a SNR ranging from 0 dB to  $-25 \text{ dB}$ , with a 1 dB step. The value of  $Z\%$  evaluated were: 2%, 4%, 6%, 8%, 10%, 20%, and 100%. Parameters are summarized in Table 5. We performed 4 simulations for each couple of  $Z\%$ , SNR values, for a total of  $4 \times 7 \times 25 = 700$  simulations. The performance of the receiver was evaluated through the BER, calculated as the ratio between the number of the wrong received bits, and the number of bits in the payload ( $10^6$  in these experiments). In case of no error, the BER was considered  $1/10^6$ .

**Table 5.** Parameters employed in simulations.

Parameter	Value
Sync seq.	100 bits
Payload	$10^6$ bits
$T_b$	$10 \mu\text{s}$
$f_c$	10 Msps
$N$	100
SNR	0; $-25 \text{ dB}$ ; step 1 dB
$Z\%$	2%, 4%, 6%, 8%, 10%, 20%, 100%

The BER resulting from the 4 simulations for each couple of  $Z_{\%}$  and SNR values were averaged and reported in Figure 5. The picture shows 7 curves in different colors: one for each value of  $Z_{\%}$  tested (see the legend). The curves represent the BERs over the simulated range of SNR. It is interesting noting that the receiver makes no error as long as the SNR is higher than about  $-3$  dB, and the output is only noise when  $\text{SNR} < -20$  dB, regardless of  $Z_{\%}$ . For SNR between  $-3$  and  $-20$  dB  $Z_{\%}$  plays an important role, and in general the lower  $Z_{\%}$ , the better. However, the results show a limit in the improvement of the BER with  $Z_{\%}$ . For example, given an acceptable error rate of  $\text{BER} < 10^{-4}$ , we observe a gain in the performance when  $Z_{\%}$  decreases until 10%, but no gain for  $Z_{\%} < 10\%$ . Similarly, the lower limit of  $Z_{\%}$  to achieve an improvement in the performance is about  $Z_{\%} = 6\%$  when we require  $\text{BER} < 10^{-2}$ ; and so on.



**Figure 5.** BER measured when a data packet of 1 Mbit affected by white noise from 0 to  $-25$  dB is received by applying different  $Z_{\%}$ .

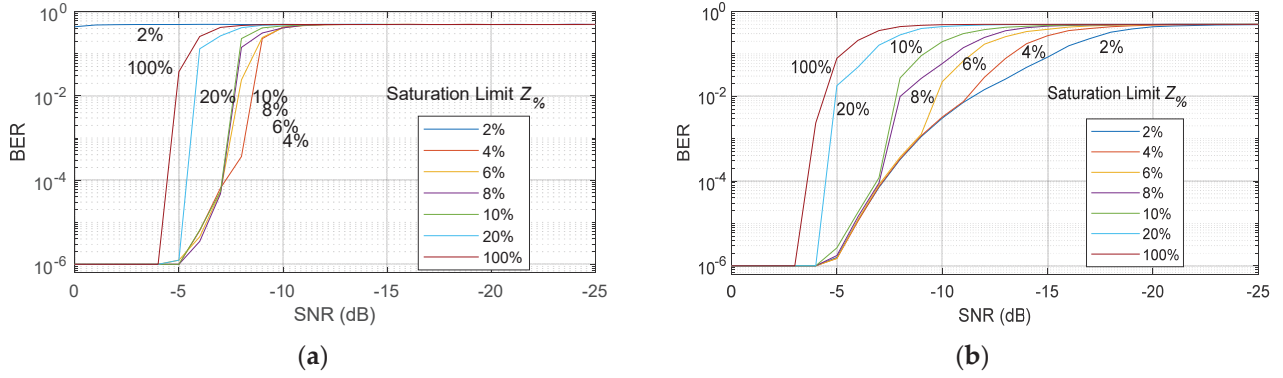
### 3.2. Evaluation of the Synchronization on Sequences Affected by Timing Errors

The timings of the transmitter and the receiver are based on independent local oscillators, which, although they share the same nominal frequency, generate clocks affected by light differences. Typical oscillators have accuracy in the order of some tens of part per million (ppm) that translate in relative frequency differences between transmitter and receiver below 0.1%. Another possible source of frequency shift is the Doppler effect generated when the transmitter and the receiver move one with respect to the other. In addition, the oscillators are affected by jitter noise [38], i.e., rapid variation of the clock phase due to the electrical noise, and frame jitter noise [39], i.e., temporal variation of the start of the bit frames.

Here, we tested how the proposed algorithm tolerates the frequency shift and jitter noise, starting with the first. We duplicated the experiment proposed in the previous section by changing the bit length  $T_b$  by 1%, which, being a tenfold variation with respect to what expected in a real application, represents a quite unfavorable condition.

The results are presented in Figure 6a. By comparing the curves to those of Figure 5, we note a slight worsening of the performance. In particular, the curve for  $Z_{\%} = 2\%$  stands out, indicating that in this condition the reception is never possible regardless of the SNR, while in the previous test  $Z_{\%} = 2\%$  granted the best performance (see Figure 5). This result is explained by considering that the limitation of  $Z_{\%} = 2\%$  corresponds to a correction of 1 sample per bit with the experimental parameters of Table 5, that is, exactly the out-of-frequency condition we imposed in the transmission sequence. In other words, the

maximum correction rate is not enough to catch with the bit temporal variation produced by the simulated frequency error. Apart from  $Z_{\%} = 2\%$ , like in the previous test, the performance improves with decreasing  $Z_{\%}$ , but there is no significant gain for  $Z_{\%} < 10\%$ . No reception is achieved for  $\text{SNR} < -10$  dB.

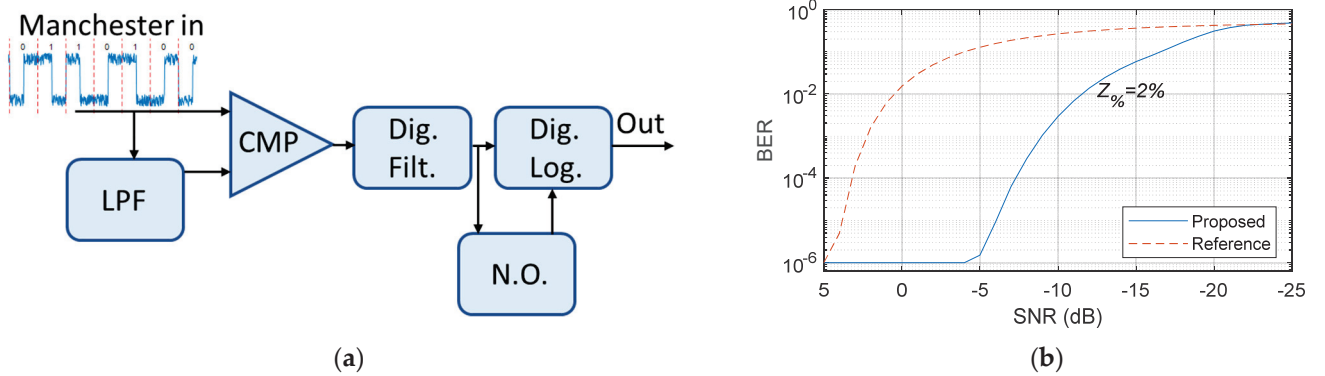


**Figure 6.** BER measured by receiving a data packet of 1 Mbit when frequency error of 1% (a) and noise jitter (b) are present. SNR is reported in  $x$ -axis, different curves refer to receptions obtained with different saturation levels  $Z_{\%}$ .

The last experiment was designed to simulate the presence of jitter: the bits were generated by varying their duration. If  $T_b$  and  $T_{bn}$  are the actual and the nominal bit duration, we varied randomly  $T_b$  in the range  $T_{bn}(1 \pm 0.01)$ , i.e., a 1% variation. Once again, the other parameters were the same as reported in Table 5. The measurements reported in Figure 6b, are pretty similar to those obtained without jitter shown in Figure 5. A light worsening is present for all the saturation levels  $Z_{\%}$  tested, but the receiver easily tolerates this level of jitter.

### 3.3. Comparison of the Proposed Decoder to Reference

The Manchester decoder proposed has so far been compared in performance to a standard decoder, similar to those described in some papers [24,25], but also present in several books and application notes. The standard decoder here considered (see Figure 7a) was based on a 2-level analog-to-digital converter realized by a tracking comparator, followed by a digital filter and logic for the decoding. A Numerical Oscillator (NO) reconstructed the data clock. Similarly to what we did to test the proposed decoder, Manchester-coded signals with added noise at increasing power were generated and processed through the decoder. The BER was obtained by comparing the output and ground-truth data.

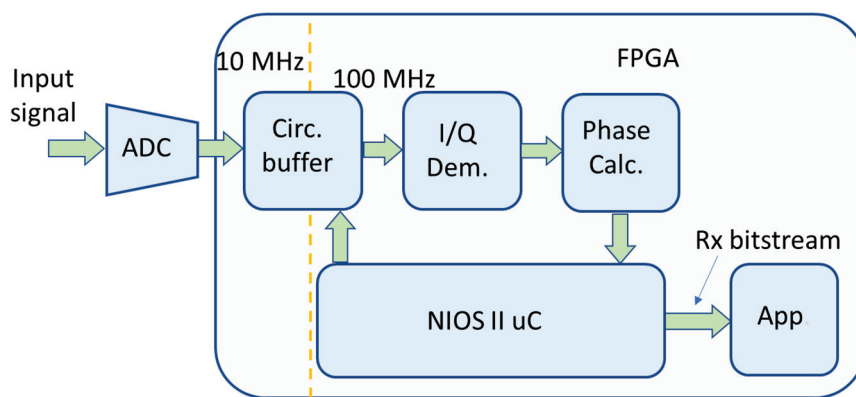


**Figure 7.** (a) Manchester reference decoder. (b) BER comparison between the proposed (blue continuous curve) and the reference decoder (red-dashed curve).

Figure 7b reports the outcome of the comparison for BER in the range 5–25 dB. The proposed decoder was set to  $Z_{\%} = 2\%$  to compare better with the low bandwidth of the NO of the reference decoder. Being equal the BER, the proposed decoder gains about 10 dB over the SNR. For BER higher than  $10^{-2}$  the advantage reduces, but this error range finds scarce interest in practical applications.

#### 4. FPGA Implementation

This section describes in detail how the decoder, whose general architecture is elaborated and characterized in the previous section of the paper, is implemented in the FPGA. The data flow in input consists of 10 M words per second at 12-bit, as sourced by the ADC. The tasks necessary to carry out in real time the calculations on such a data flow are subdivided, as visible in Figure 8, between the NIOS II soft-processor and other blocks that act like efficient mathematical specialized co-processors. This subdivision, quite common in FPGA projects, allows the exploitation of the advantages belonging to both the microcontroller ( $\mu C$ ) and the dedicated logic. The former is programmed in ‘C’ language, making it quite easy to apply changes, make corrections, investigate new possible approaches; the latter grants the required calculation power to process the data flow that the  $\mu C$  alone would never grant. Nevertheless, as it would be clear in the following, the  $\mu C$  must complete its tasks inside timing windows strictly correlated to the bit-length.



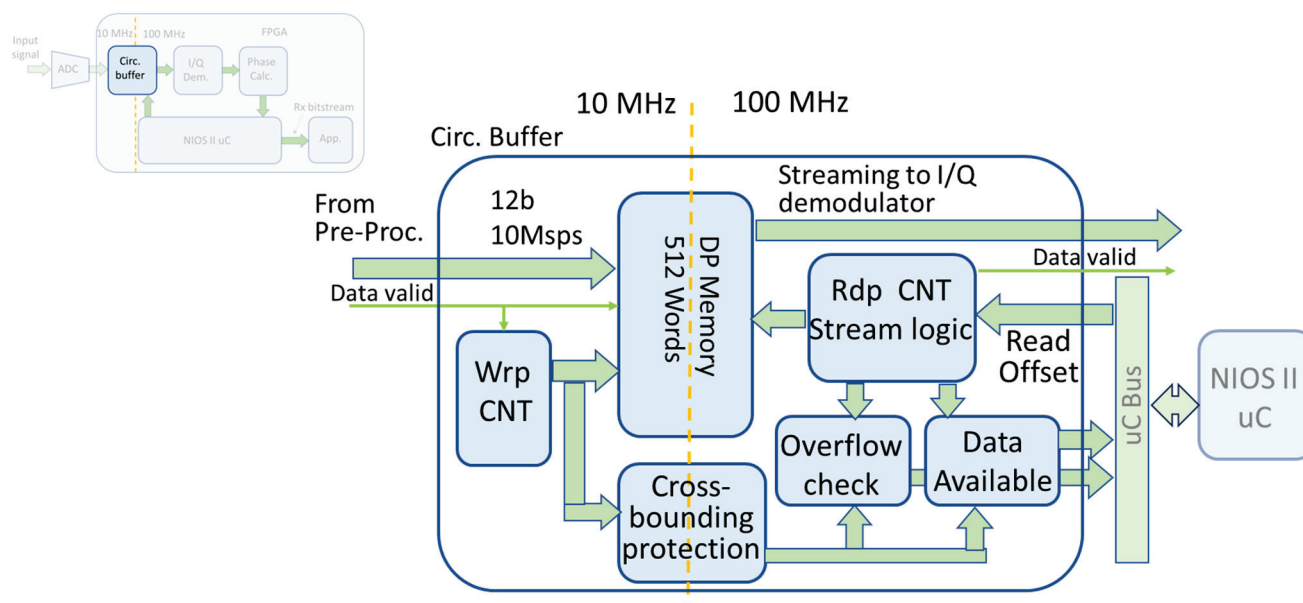
**Figure 8.** “Bird’s eye” view of the decoder architecture implemented in FPGA.

It should be noted that the architecture spans two separated clock domains: data is sourced from the ADC at 10 Msps, synchronized to a 10 MHz clock; while the  $\mu C$ , the I/Q demodulator and the phase calculation run at the higher 100 MHz clock. This choice leaves more execution power for the  $\mu C$  and the processing logics. The circular buffer that, as clarified later, is based on dual-port memory, represents the ideal bridge to cross the two clock domains. A description of the functional blocks visible in Figure 8 follows, organized per single block.

##### 4.1. Circular Buffer

The acquired samples reach the circular buffer, where they are temporarily stored (see Figure 9). The heart of the buffer is a dual-port memory capable of holding 512 data. The memory is used like a clock-domain bridge as well: it is written from the 10 MHz side and read from the 100 MHz side. The write side is managed as a typical circular buffer: the writing address pointer (Wrp) is generated by a counter modulo 512 that increments every input sample. The position of the writing pointer is necessary also in the read-domain. A cross-bounding logic protects against possible metastable events that can occur when the pointer crosses the clock-domain.





**Figure 9.** Acquired data are temporarily stored in the circular buffer. They are written from the 10 MHz side, and read from the 100 MHz domain in blocks of 100 samples that represent the data covering a  $T_h$ .

In the read-side of the buffer, the read-pointer (Rdp) is generated by a counter modulo 512 managed by a specific logic, detailed in this section. The values of write and read pointers are monitored to detect possible overflow of the buffer, which occurs when input samples are written in positions where previous data have not been read yet. In addition, the values of the pointers are used to calculate the number of samples available in the buffer, i.e., the number of sample not yet read, but ready to be read. The number of available samples and the information about overflow are mapped in the address space of the  $\mu\text{C}$  that reaches them through its bus.

Let us go back to the logic that manages the read-pointer. The  $\mu\text{C}$ , again through its bus, writes a Read Offset. When this operation occurs, the Rdp is updated with the operation:  $\text{Rdp} = \text{Rdp} - \text{ROff}$ , where ROff is the Read Offset set by the  $\mu\text{C}$ . In addition, the streaming of 100-samples data towards the next stage is automatically triggered. At the sample time of 1/10 MHz, the 100-sample corresponds to  $T_b = 10 \mu\text{s}$ . In other words, the data corresponding to a bit with the starting point set by the  $\mu\text{C}$ , is moved at the maximum velocity of one sample per clock cycle towards the demodulator. During the streaming, Rrp is incremented, and the data valid to the next block is generated as needed.

It should be noted that, although the logic that governs this circular buffer resembles that of a First-In-First-Out (FIFO) memory, it is not exactly the same. In particular, while the write-side corresponds to the logic of a FIFO, the read-side does not. In fact, in FIFO memory, each sample must necessarily be read, and it should be read only once. In the proposed architecture, according to the Read Offset set by the  $\mu\text{C}$ , it can happen that some sample is not read at all, or some sample is read multiple times, as required for the decoder and detailed in Section 2.3.

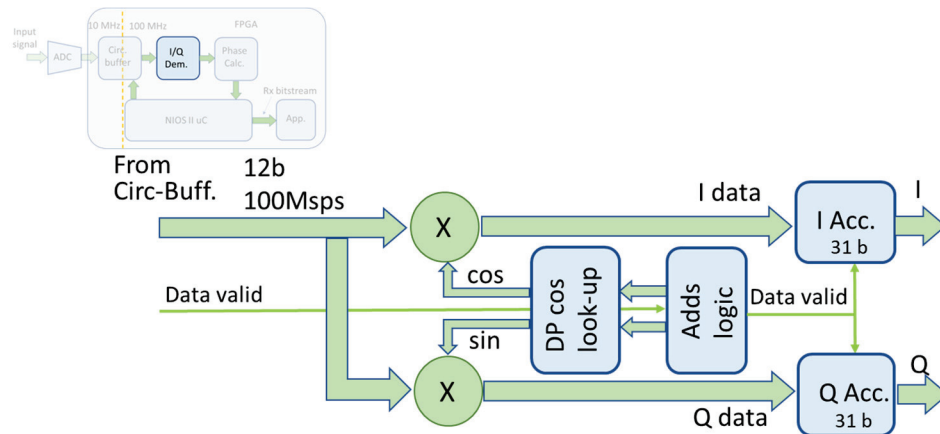
The memory is composed by 512-sample, and thus it can buffer about  $5 \cdot T_b$  of data.

#### 4.2. I/Q Demodulator

The data-bursts coming from the circular buffer, composed by 100-sample, reach two parallel multipliers in the I/Q demodulator, designed based on [40], (see Figure 10). The bursts are supported by the data-valid flag. The aim of this block is calculating the I and Q values according to Equation (6). A look-up dual port memory holds 100-sample of a



period of cosine signal, with 12-bit resolution. The address logic generates two parallel read addresses so that a cosine and sine signal feed the corresponding multipliers. The cosine signal is obtained by reading the look-up from address 0, while the sine signal is obtained by starting the reading from address 75, i.e., with a  $270^\circ$  phase-delay. The full 24-bit dynamics of the multipliers output is maintained. These signals feed two parallel accumulators working at 31-bit. The data-valid signal is used to reset the address logics and to zero the I/Q accumulators before the beginning of each data burst.



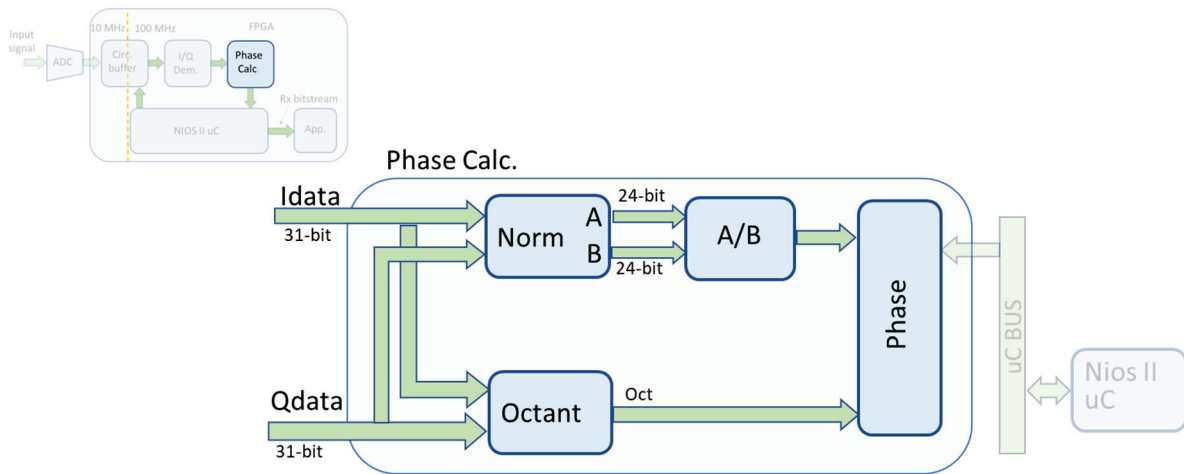
**Figure 10.** I/Q demodulator.

#### 4.3. Phase Calculation and Approximation of the Inverse Tangent

This part of the FPGA implements the algorithm for phase calculation through the inverse tangent approximation described in Section 2.4 like reported in Figure 11. The I and Q data present in the accumulators are the starting point that feed the “Octant” and “Norm” blocks. The previous block, like the name suggests, compares I and Q to find the octant like described in Table 4. The “Norm” block prepares the numerator, A, and denominator, B, for the 24-bit fixed-point divisor that follows. I and Q are left-shifted of  $j$  positions so that the maximum of them in absolute value fills the 24-bit dynamic. Then, the lower value is cut to the 12 MSB, further scaled by  $2^{12}$ , and placed in the numerator A. The higher value is cut to 12 MSB, filled with 12 zeros on the left, and placed in the denominator B. This way leads to the calculation of the inverse tangent for even quadrants and inverse cotangent for odd quadrants, which always results in an angle less than  $45^\circ$ . The shift has no effect on  $A/B$  but allows the exploitation of the full dynamics of the divisor; the  $2^{12}$  scaling applied to the numerator A is necessary for achieving an integer division. Table 6 summarizes the procedure.

**Table 6.** Preparation of numerator and denominator of division.

Sign (I)	Sign (Q)	I ,  Q	Numerator (A)	Denominator (B)
+	+	$ I  \leq  Q $	$2^j \cdot  I  \cdot 2^{12}$	$2^j \cdot  Q $
-	+	$ I  <  Q $	$2^j \cdot  I  \cdot 2^{12}$	$2^j \cdot  Q $
-	+	$ I  \geq  Q $	$2^j \cdot  Q  \cdot 2^{12}$	$2^j \cdot  I $
-	-	$ I  <  Q $	$2^j \cdot  I  \cdot 2^{12}$	$2^j \cdot  Q $
-	-	$ I  \geq  Q $	$2^j \cdot  Q  \cdot 2^{12}$	$2^j \cdot  I $
+	-	$ I  \leq  Q $	$2^j \cdot  I  \cdot 2^{12}$	$2^j \cdot  Q $
+	-	$ I  >  Q $	$2^j \cdot  Q  \cdot 2^{12}$	$2^j \cdot  I $
+	+	$ I  >  Q $	$2^j \cdot  Q  \cdot 2^{12}$	$2^j \cdot  I $



**Figure 11.** Phase calculation in FPGA.

The output of the 24-bit divisor is processed by the “Ph” block that applies the simple operations reported in the last column of Table 4. Finally, the phase is ready to be read by the  $\mu$ C.

#### 4.4. Bit Decision, Synchronization, and Managing

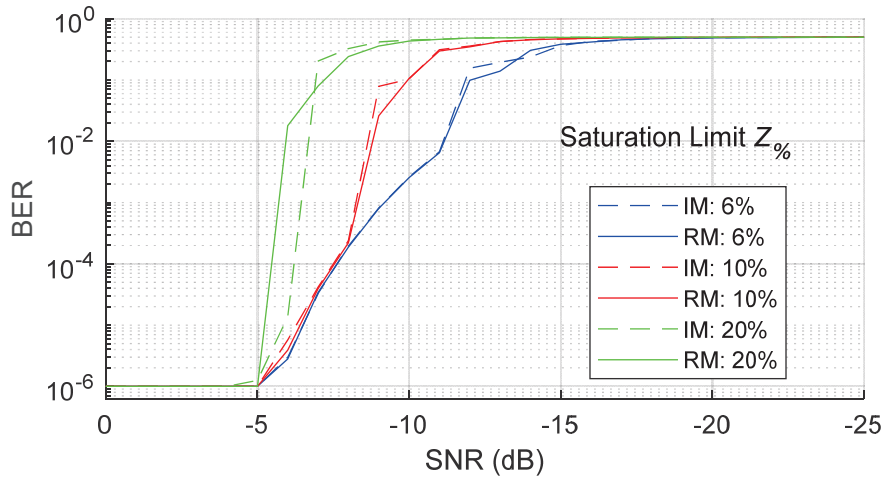
The  $\mu$ C reads the phase. From its value, it decides the value of the bit (see Table 3) and calculates the offset correction  $l$  according to (8) to maintain the synchronization. The  $\mu$ C finally writes the offset  $l$  in the circular buffer triggering the starting of the evaluation of the next bit. The detect bit is moved in memory and is available for further processing or directly passed to the final application. The  $\mu$ C can use the remaining time (see Section 5.1) for other tasks, like searching for data preambles, data unpackaging, CRC checks, etc.

#### 4.5. Mathematical Noise

The mathematical processing in the receiver is implemented in fixed-point representation. This representation, very convenient in FPGA, inevitably involves approximations. In addition, the inverse tangent function was further approximated through linear functions. In this section we describe the tests planned to investigate if these approximations impact the performance of the receiver.

The real mathematics (RM), like implemented in FPGA, was duplicated in MATLAB® R2024a. The same test reported in Section 3 was repeated with RM and compared to the reference results obtained with ideal mathematics (IM), i.e., the 64-bit floating point format.

BERs are reported for  $Z_{\%}$  of 6, 10, 20% in Figure 12, where the dashed and continuous curves describe the results calculated with IM and RM, respectively. In all cases, the curves are very similar. In some parts of the graph, the results obtained with RM seems even better, but this is attributed to statistical fluctuations related to the random nature of the payloads and the noise added into the generation of every signal.



**Figure 12.** Comparison of BERs measured by a receiver working with ideal mathematic (IM, dashed curves) and real mathematic (RM, continuous curves). Tests are performed with SNR from 0 to  $-25$  dB and  $Z_{\%}$  of 6%, 10%, and 20%.

## 5. Experiments and Results

### 5.1. Resources and Timings

The algorithm described so far was coded in VHDL [41] and implemented in a VLC research system realized in-house [30]. The system includes a 10M50DAF486C6 FPGA from Intel (Santa Clara, CA, USA) together with all the electronics for transmitting and receiving VLC data. The system is managed through a MATLAB® user interface that runs on a host PC. A framework present in the FPGA supports the user in the implementation of new applications, like the one here proposed. More details about the VLC system and its use can be found in [30,42].

The resources employed by the proposed decoder once integrated in the FPGA are listed in Table 7. The receiver requires about 7% of Advanced Logic Modules (ALMs), 3.5% of the Digital Signal Processors (DSPs) and 1.3% of the memory available in the target FPGA. The logic that requires most of the resources is the NIOSII soft processor. However, it should be noted that the processor can be used for other tasks as well.

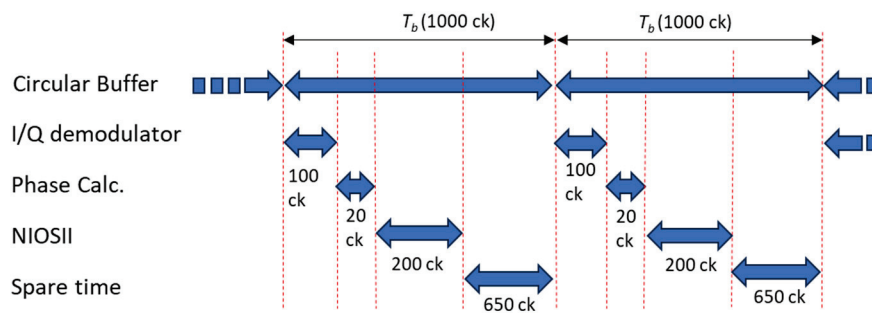
**Table 7.** FPGA resources required by receiver.

Entity	ALM	DSP (18 × 18)	Memory (bit)
Circular Buffer	143	0	8192
I/Q Demodulator	172	2	3072
Phase Calc.	686	0	0
NIOSII Processor	2133	3	10,762
Interconnect	321	0	0
TOT	3455	5	22,026

The receiver was compiled with a target clock of 100 MHz, and the compilation software confirmed the timing closure with a large margin.

Before proceeding to the field experiments, we checked with the on-board emulator that the timings of the system tasks were consistent. Every bit in air lasts for  $T_b$ , which corresponds to 1000 clock cycles when the clock is 100 MHz. Thus, every bit must be processed in 1000 clock cycles at most on average. Thanks to the 5-bit capacity of the input circular buffer, a few bits every now and again can take more time, as long as the 1000 cycles limit is maintained on average.

Figure 13 shows the timing of the processing, checked in the FPGA working in real time. As long as a bit is present in the circular buffer, the NIOSII commands the start of the I/Q demodulation, which terminates in 100 clock cycles: a cycle per sample. At this time, I and Q are present in the accumulators and ready for the phase calculation, which occurs in 20 cycles. The NIOSII gets the phase, makes the decision on the bit, and calculates the phase correction for reading the next bit. This info is set in the circular buffer that in the meantime continued to store input samples. Now the NIOSII waits for the next 100 samples from the new offset to be ready in the buffer, and the processing starts again. On average the processor has about 650 cycles free every 1000, that can be used for managing the data packaging, searching for the data preambles, CRC checking, etc. The  $\mu C$  is apparently unloaded; however, its calculation power would never be enough to support the calculation of the phase in real time, which, for this reason, is performed by the specialized hardware described in the previous parts of this work.



**Figure 13.** Temporal budget of the processing in the receiver. Every bit must be processed in 1000 clock cycles on average. Small fluctuations are tolerated thanks to the circular buffer in input.

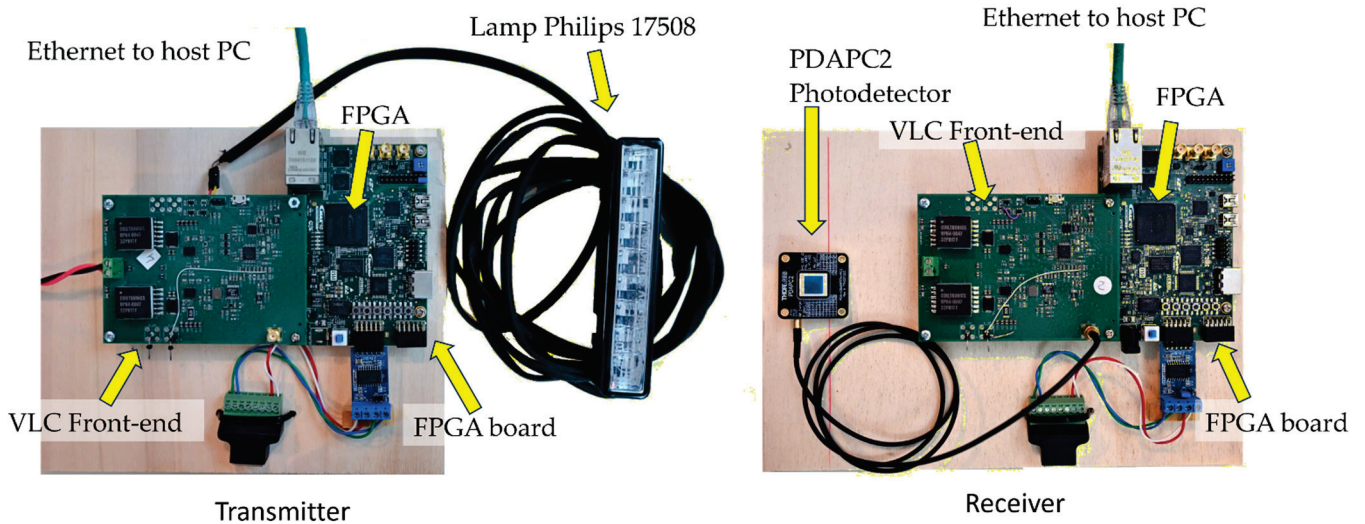
Finally, we note that the latency of the processing is very low, less than  $T_b$ .

## 5.2. Experimental Set-Up

The experiments were performed by employing two of the VLC systems composed by a house-made frontend connected to a commercial FPGA developing boards, described in [30]: one system acted like the transmitter; the second as the receiver. The transmitter included in the FPGA was a Manchester coder designed according to the IEEE 802.15.7 standard. It was connected to the lamp model 17508 produced by Philips N.V. (Amsterdam, The Netherlands): a 16 W, 9 LED lamp certified for automotive applications. The receiver, whose FPGA was completed with the described Manchester decoder, was connected to the photodetector PDAPC2 produced by Thorlabs Inc. (Newton, NJ, USA). The 2 systems run on independent clock oscillators, which produced an unavoidable frequency shift between the TX and the RX. A photo of the 2 VLC systems connected to the lamp and the photodetector is reported in Figure 14.

The lamp and the receiver were assembled on tripods and moved 4 m apart. The transmitter was loaded with data packets of 2 Mb random bits. The transmission of a sequence of '0' bits preceded the transmission of the 2 Mb data packet. The sequence was used by the receiver to achieve synchronization. The receiver saved in its memory the decoded bits and the IQ data, used later for debugging and analysis. These data were downloaded to the PC after the end of communication.

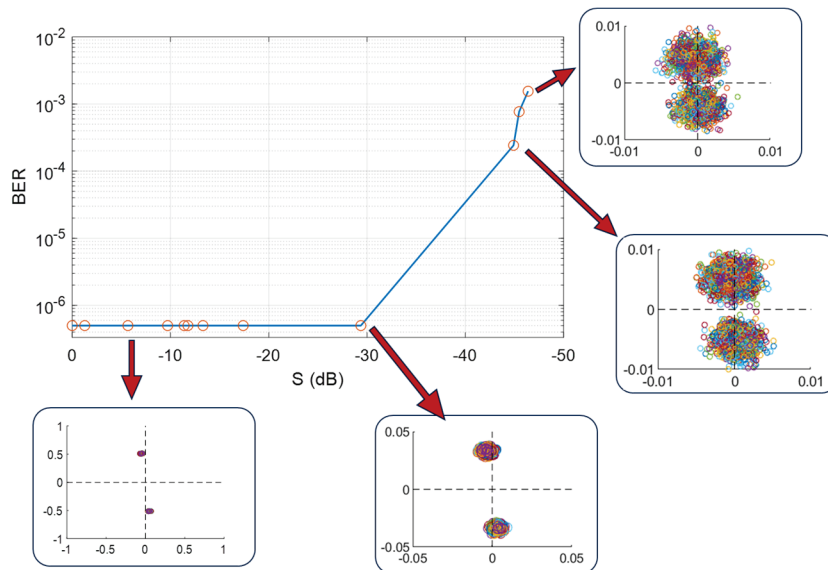
Several tests were conducted by varying the SNR at the receiver. At the receiver, the main noise source is constituted by the electronics noise of the photodetector and the first stage of amplification, which are independent from the input signal and constant over the experiments. Given this constant input noise, the variation of SNR was obtained by reducing the level of the signal captured from the transmitter.



**Figure 14.** In the experiments, we employed one VLC systems: one acts like TX (left) and is connected to the automotive lamp, the second like RX (right) and is connected the PDAPC2 photodetector.

### 5.3. Experimental Results

Twelve transmissions were carried out with different SNRs at the receiver. The received bits and IQ data were saved in every test. The BER was calculated by comparing the received packed to the transmitted data. Results are shown in Figure 15. On the abscissa, the received signal power  $S$  is reported in dB after being normalized with respect to the maximum dynamics of the analog-to-digital converter of the system. We measured no errors until the input signal was higher than  $-30$  dB. For lower intensities, the BER rose and reached the  $10^{-3}$  threshold for  $S$  of  $-47$  dB.



**Figure 15.** BER measured at different levels of input signal  $S$ . Input signal is normalized with respect to receiver dynamics. Red circles report experimental measurements. Received constellations are shown in four selected cases.

For a better understanding of the results, Figure 15 reports the constellations measured in four selected cases. The constellations are shown in the IQ plane by overlapping 200 IQ values. Please note that the axes scale changes among the four panels. The unity in the axes again represents the maximum dynamics of the receiver. As noted before, the



noise is mainly due to the electrical noisy of the analog section of the receiver, and remains constant. Thus, the diameter of the circular clouds, formed by the distribution of the received IQ points, remains basically unchanged. When the power of the signal, which in the constellation is related to the distance of each point from the origin, reduces below  $S < -30$  dB, the decision given by the sign of Q starts to generate errors.

## 6. Discussion and Conclusions

In this work the architecture of a high-performance Manchester receiver, specifically designed for FPGA implementation, was described and tested. The receiver has been implanted in a research VLC system and the expected results have been verified through simulations and experiments. Thanks to the 12-bit AD in the front-end and the IQ demodulation the receiver achieves a high noise immunity, boosting the performance of the application it is embedded in.

This approach is relatively calculation-demanding: just the demodulation for the IQ calculation requires 200 multiplications and as many additions in every  $T_b$ , i.e., 4G operations per s. A real-life deployment of the algorithm, for example in the automotive context, requests addition security features for the implementation of the Automotive Safety Integrity Level (ASIL), which require even further calculations and hardware complication. Even if the required calculation intensity is hardly achievable through a processor and far from the reach of a  $\mu\text{C}$ , it can be easily supported by only two DSP blocks of the FPGA (see Table 7), leaving much room for the implementation of security protocols or other algorithms.

On the other hand, the calculation efforts of the proposed approach are similar to other complex algorithms successfully employed in widespread commercial applications, where the use of FPGA can be problematic. A notable example is the Global Position System (GPS) receiver [43]. The challenge is solved by the design of Application-Specific Integrated Circuits (ASICs) which include all the dedicated processing and, in high volume applications, are produced for few cents (see Table 1). In other high-end fields of application, FPGAs are already present. For example, where Manchester coded avionics buses like the MIL-STD-1553 [44] or the ARINC 659 [45] are used. These buses are already managed by FPGAs, and the transition to VLC is being considered [46].

**Author Contributions:** Conceptualization and methodology, S.R. and S.C.; firmware, S.R.; validation, S.C.; formal analysis, L.M.; investigation, S.R. and S.C.; resources, L.M.; writing—original draft preparation, S.R.; writing—review and editing, L.M. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was funded by the Italian Ministry of University and Research (MIUR) and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001—program “RESTART”).

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors on request.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Khan, L.U. Visible Light Communication: Applications, Architecture, Standardization and Research Challenges. *Digit. Commun. Netw.* **2017**, *3*, 78–88. [CrossRef]
2. Rehman, S.; Ullah, S.; Chong, P.; Yongchareon, S.; Komosny, D. Visible Light Communication: A System Perspective—Overview and Challenges. *Sensors* **2019**, *19*, 1153. [CrossRef]
3. Shen, W.H.; Tsai, H.M. Testing vehicle-to-vehicle visible light communications in real-world driving scenarios. In Proceedings of the 2017 IEEE Vehicular Networking Conference (VNC), Torino, Italy, 27–29 November 2017. [CrossRef]

4. Wang, Y.; Chen, X.; Xu, Y. Transmitter for 1.9 Gbps phosphor white light visible light communication without a blue filter based on OOK-NRZ modulation. *Opt. Express* **2023**, *31*, 7933–7946. [CrossRef] [PubMed]
5. Cossu, G.; Khalid, A.M.; Choudhury, P.; Corsini, R.; Ciaramella, E. 3.4 Gbit/s Visible Optical Wireless Transmission Based on RGB LED. *Opt. Express* **2012**, *20*, B501. [CrossRef] [PubMed]
6. Elamassie, M.; Karbalayghareh, M.; Miramirkhani, F.; Kizilirmak, R.C.; Uysal, M. Effect of fog and rain on the performance of vehicular visible light communications. In Proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), Porto, Portugal, 3–6 June 2018. [CrossRef]
7. Fuada, S.; Pradana, A.; Adiono, T.; Popoola, W.O. Demonstrating a real-time QAM-16 visible light communications utilizing off-the-shelf hardware. *Results Opt.* **2023**, *10*, 100348. [CrossRef]
8. Gagliardi, R.M.; Karp, S. Optical Communications. In *Wiley Series in Telecommunications and Signal Processing*, 2nd ed.; Wiley: New York, NY, USA, 1995; ISBN 978-0-471-54287-2.
9. Ricci, S.; Caputo, S. Transmitter for Visible Light Communications Based on FPGA's Output Buffers. *IEEE Commun. Lett.* **2024**, *28*, 2116–2120. [CrossRef]
10. Nawaz, T.; Seminara, M.; Caputo, S.; Mucchi, L.; Cataliotti, F.S.; Catani, J. IEEE 802.15.7-Compliant Ultra-Low Latency Relaying VLC System for Safety-Critical ITS. *IEEE Trans. Veh. Technol.* **2019**, *68*, 12040–12051. [CrossRef]
11. Li, H.; Chen, X.; Guo, J.; Chen, H. A 550 Mbit/s Real-Time Visible Light Communication System Based on Phosphorescent White Light LED for Practical High-Speed Low-Complexity Application. *Opt. Express* **2014**, *22*, 27203. [CrossRef] [PubMed]
12. Forster, R. Manchester encoding: Opposing definitions resolved. *Eng. Sci. Educ. J.* **2000**, *9*, 278–280. [CrossRef]
13. IEEE 802.15.7-2018; IEEE Standard for Local and Metropolitan Area Networks—Part 15.7: Short-Range Optical Wireless Communications. IEEE Standards Association: Piscataway, NJ, USA, 2019. Available online: <https://standards.ieee.org/ieee/802.15.7/6820/> (accessed on 6 December 2024).
14. Cui, Z.; Yue, P.; Yi, X.; Li, J. Research on non-uniform dynamic vehicle-mounted VLC with receiver spatial and angular diversity. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019. [CrossRef]
15. Caputo, S.; Ricci, S.; Mucchi, L. IEEE 802.15.7-Compliant Full Duplex Visible Light Communication: Interference Analysis and Experimentation. *IEEE Open J. Veh. Technol.* **2024**, *5*, 1242–1255. [CrossRef]
16. Razavi, B. Challenges in the design high-speed clock and data recovery circuits. *IEEE Commun. Mag.* **2002**, *40*, 94–101. [CrossRef]
17. Ahmed, S.I.; Kwasniewski, T.A. Overview of oversampling clock and data recovery circuits. In Proceedings of the Canadian Conference on Electrical and Computer Engineering, Saskatoon, SK, Canada, 1–4 May 2005; pp. 1876–1881. [CrossRef]
18. Moon, Y.H.; Kang, J.K.  $2\times$  oversampling 2.5 Gbps clock and data recovery with phase picking method. *Curr. Appl. Phys.* **2004**, *4*, 75–81. [CrossRef]
19. Bartley, T.; Tanaka, S.; Nonomura, Y.; Nakayama, T.; Muroyama, M. Delay window blind oversampling clock and data recovery algorithm with wide tracking range. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, 24–27 May 2015; pp. 1598–1601. [CrossRef]
20. Kubicek, M.; Kolka, Z. Blind Oversampling Data Recovery with Low Hardware Complexity. *Radioengineering* **2010**, *19*, 74–78.
21. Kolka, Z.; Kubicek, M.; Biolkova, V. Optimization of oversampling Data Recovery. In Proceedings of the 52nd IEEE International Midwest Symposium on Circuits and Systems, Cancun, Mexico, 2–5 August 2009; pp. 467–470. [CrossRef]
22. Wang, C.C.; Lee, C.L.; Hsiao, C.Y.; Huang, J.F. Clock-and-Data Recovery Design for LVDS Transceiver Used in LCD Panels. *IEEE Trans. Circuits Syst. II Express Briefs* **2006**, *53*, 1318–1322. [CrossRef]
23. Vijayalakshmi, S.; Paramasivam, A.; Nagarajan, V.; Kudiyarasan, S.; Kamatchi, S.; Hasheetha, J. Design and performance analysis of manchester coder-based body channel communication using FPGA. *E-Prime—Adv. Electr. Eng. Electron. Energy* **2024**, *9*, 100660. [CrossRef]
24. Shi, J.; Xu, Y.; Shi, J. Manchester encoder and decoder based on CPLD. In Proceedings of the IEEE International Conference on Industrial Technology, Chengdu, China, 21–24 April 2008; pp. 1–3. [CrossRef]
25. Zuo, Y.; Yang, J.; Cheng, X. Design and implementation of Manchester CODEC based on FPGA. *Appl. Mech. Mater.* **2013**, *273*, 805–809. [CrossRef]
26. Kim, J.; Jeong, D.K. Multi-gigabit-rate clock and data recovery based on blind oversampling. *IEEE Commun. Mag.* **2003**, *41*, 68–74. [CrossRef]
27. Yoo, J.H.; Jang, J.S.; Kwon, J.K.; Kim, H.C.; Song, D.W.; Jung, S.Y. Demonstration of vehicular visible light communication based on LED headlamp. *Int. J. Automot. Technol.* **2016**, *17*, 347–352. [CrossRef]
28. Eldeeb, H.B.; Elamassie, M.; Sait, S.M.; Uysal, M. Infrastructure-to-Vehicle Visible Light Communications: Channel Modelling and Performance Analysis. *IEEE Trans. Veh. Technol.* **2022**, *71*, 2240–2250. [CrossRef]
29. Berber, S. Discrete Bandpass Modulation Methods. In *Discrete Communication Systems*; Oxford University Press: Oxford, UK, 2021; pp. 305–385. [CrossRef]



30. Ricci, S.; Caputo, S.; Mucchi, L. FPGA-based visible light communications instrument for implementation and testing of ultralow latency applications. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 2004811. [CrossRef]
31. Almeida, A.J.; Silva, N.A.; Muga, N.J.; André, P.S.; Pinto, A.N. Calculation of the number of bits required for the estimation of the bit error ratio. In Proceedings of the Second International Conference on Applications of Optics And Photonics, Aveiro, Portugal, 26–30 May 2014. [CrossRef]
32. Mengali, U.; D’Andrea, A.N. *Synchronization Techniques for Digital Receivers*; Springer Science + Business Media: New York, NY, USA, 1997. [CrossRef]
33. Cooley, J.; Lewis, P.; Welch, P. The finite Fourier transform. *IEEE Trans. Audio Electroacoust.* **1969**, *17*, 77–85. [CrossRef]
34. Rajan, S.; Wang, S.; Inkol, R.; Joyal, A. Efficient approximations for the arctangent function. *IEEE Signal Process. Mag.* **2006**, *23*, 108–111. [CrossRef]
35. Pilato, L.; Fanucci, L.; Saponara, S. Real-Time and High-Accuracy Arctangent Computation Using CORDIC and Fast Magnitude Estimation. *Electronics* **2017**, *6*, 22. [CrossRef]
36. Benammar, M.; Alassi, A.; Gastli, A.; Ben-Brahim, L.; Touati, F. New Fast Arctangent Approximation Algorithm for Generic Real-Time Embedded Applications. *Sensors* **2019**, *19*, 5148. [CrossRef] [PubMed]
37. Gutierrez, R.; Torres, V.; Valls, J. FPGA-implementation of atan(Y/X) based on logarithmic transformation and LUT-based techniques. *J. Syst. Archit.* **2010**, *56*, 588–596. [CrossRef]
38. Zeng, Z.; Zhang, L.; Gong, L.; Zhang, N. A Fast Lock-In Time, Capacitive FIR-Filter-Based Clock Multiplier with Input Clock Jitter Reduction. *Electronics* **2023**, *12*, 1439. [CrossRef]
39. Russo, D.; Ricci, S. FPGA Implementation of a Synchronization Circuit for Arbitrary Trigger Sequences. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 5251–5259. [CrossRef]
40. Ricci, S.; Meacci, V. Data-Adaptive Coherent Demodulator for High Dynamics Pulse-Wave Ultrasound Applications. *Electronics* **2018**, *7*, 434. [CrossRef]
41. Dally, W.J.; Harting, R.C.; Aamodt, T.M. *Digital Design Using VHDL: A Systems Approach*; Cambridge University Press: Cambridge, UK, 2015; ISBN 978-1107098862.
42. Ricci, S.; Caputo, S.; Mucchi, L. FPGA-Based Pulse Compressor for Ultra Low Latency Visible Light Communications. *Electronics* **2023**, *12*, 364. [CrossRef]
43. Kaplan, E.; Hegart, C. *Understanding GPS/GNSS: Principles and Applications*, 3rd ed.; Artech House Publishers: Norwood, MA, USA, 2017; ISBN 978-1630810580.
44. Jiang, S.; Liu, S.; Guo, C.; Fan, X.; Ma, T.; Tiwari, P. Implementation of ARINC 659 bus controller for space-borne computers. *Electronics* **2019**, *8*, 435. [CrossRef]
45. Pendyala, P.; Pasupureddi, V.S.R. 100-Mb/s enhanced data rate MIL-STD-1553B controller in 65-nm CMOS technology. *IEEE Trans. Aerosp. Electron. Syst.* **2016**, *52*, 2917–2929. [CrossRef]
46. Karafolas, N.; Armengol, J.M.P.; McKenzie, I. Introducing photonics in spacecraft engineering: ESA’s strategic approach. In Proceedings of the IEEE Aerospace Conference 2009, Big Sky, MT, USA, 7–14 March 2009; pp. 1–15. [CrossRef]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# Optimal Implementation of Tapped Delay Line Time-to-Digital Converters in 20 nm Xilinx UltraScale FPGAs

Mattia Morabito, Nicola Lusardi <sup>\*,†</sup>, Fabio Garzetti <sup>†</sup>, Gabriele Fiumicelli, Gabriele Bonanno, Enrico Ronconi, Andrea Costa and Angelo Geraci

DEIB (Dipartimento di Elettronica, Informazione e Bioingegneria), Politecnico di Milano, Via Golgi 40, 20133 Milano, Italy; mattia.morabito@mail.polimi.it (M.M.); fabio.garzetti@polimi.it (F.G.); gabriele.fiumicelli@mail.polimi.it (G.F.); gabriele.bonanno@polimi.it (G.B.); enrico.ronconi@polimi.it (E.R.); andrea1.costa@polimi.it (A.C.); angelo.geraci@polimi.it (A.G.)

\* Correspondence: nicola.lusardi@polimi.it

<sup>†</sup> These authors contributed equally to this work.

**Abstract:** This study investigated implementation strategies to optimize the precision of Tapped Delay Line (TDL) Time-to-Digital Converters (TDCs) designed for Xilinx 20 nm UltraScale Field-Programmable Gate Arrays (FPGAs). This optimization process aims to bridge the performance gap between FPGA-based TDCs, which are more flexible and suitable for fast prototyping, and the better-performing Application-Specific Integrated Circuit (ASIC) solutions, making FPGA-based TDCs viable for cutting-edge applications. Our key areas of focus included the optimal design of the decoder, the degree of sub-interpolation, and the placement of TDLs, with particular emphasis on the clocking distribution scheme within the Configurable Logic Block (CLB) to minimize the effects of Bubble Errors (BEs) and quantization error. The research led to the development and comparison of multiple TDL TDC solutions implemented on a Kintex UltraScale device (i.e., XCKU040-2FFVA1156E) housed on a KCU105 general-purpose Evaluation Board (EVB). From these, two main solutions emerged: one with high precision and one with low area. The first one was characterized by a Single-Shot Precision (SSP) of 2.64 ps r.m.s., and by Differential and Integral Non-Linearity (DNL/INL) Errors of 0.523 ps and 16.939 ps, respectively, occupying 883 CLBs and 126 kb of Block RAM (BRAM). The second one had an SSP of 3.75 ps r.m.s., a DNL of 0.599 ps, and an INL of 7.151 ps, and it occupies only 259 CLBs and 72 kb of BRAM.

**Keywords:** bubble errors; decoding; Tapped Delay Line (TDL); Time-to-Digital Converter (TDC); Field-Programmable Gate Array (FPGA)

## 1. Introduction

Within the field of electronics engineering, Time-to-Digital Converters (TDCs) are integral parts of time-resolved systems and hold a prominent place [1]. Their functions encompass multiple fields, including quantum technologies [2,3], life sciences [4,5], and nuclear physics [6,7].

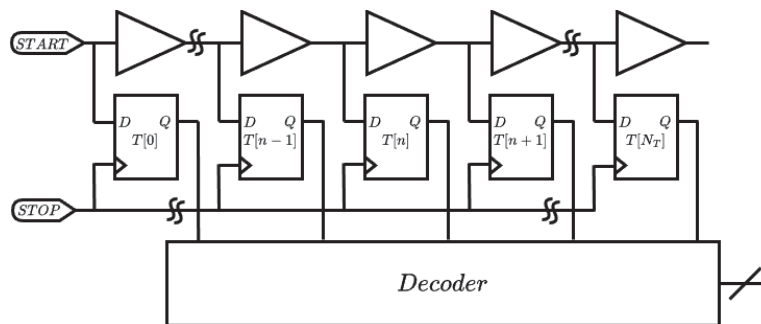
In digital electronics, including TDC circuits, Field-Programmable Gate Arrays (FPGAs) provide unmatched customization, adaptability, and off-the-shelf fast prototyping solutions compared to Application-Specific Integrated Circuit (ASIC) solutions, thanks to their rapid prototyping capabilities, reconfigurability, shorter time-to-market, and lower Non-Recurring Engineering (NRE) costs [8]. In this way, especially in R&D and academia, the use of FPGA-based TDCs has had a profound impact on the development of multiple novel TDC architectures [9,10]. The difficulty of developing competitive FPGA-based TDC implementations that excel in both performance (i.e., number of channels in a single device, resolution (i.e., LSB), Single-Shot Precision (SSP), linearity, dead time, and acquisition rate [5,11–16]) and efficiency (i.e., area occupied and power consumed by a single channel [5]) is addressed in the literature.

### 1.1. FPGA-Based Tapped Delay Line TDC

The FPGA-based TDC architecture that best balances the performance and efficiency trade-off is the Tapped Delay Line (TDL) [17], also known as TDL TDCs [9,10]. In FPGAs, which are subdivided into predefined Configurable Logic Blocks (CLBs), each containing a limited set of resources, TDLs are usually implemented by using carry chains available either in Digital Signal Processing (DSP) blocks [18] or in the FPGA fabric as carry logic primitives (i.e., CARRY4 for Xilinx 28 nm 7-Series and CARRY8 for Xilinx 16/20 nm UltraScale FPGAs) [12,13]. According to the scientific literature, at the same technological node, DSP-based TDLs are less linear compared to carry-based TDLs, due to the presence of ultra-bins (i.e., taps with a propagation time much slower than the average) placed between two consecutive DSP blocks [18,19]. On the other hand, the average propagation delay offered by the DSP-based TDL is faster compared to the carry-based one; this forces the use of more DSP blocks in series, resulting in a high count of ultra-bins. To further improve linearity at the expense of resolution, the scientific literature also presents net-based TDLs implemented using routing logic (i.e., net) [20]. In addition to offering a lower average propagation delay per tap compared to carry-based and DSP-based TDLs at the same technological node, these structures are more complex, in terms of placement.

### 1.2. Tapped Delay Line TDC Working Principle

As Figure 1 shows, the TDL TDC uses a digital low-to-high step signal (referred to as START) that passes through a series of buffers, called taps or bins, characterized by a specific propagation delay. The output of each tap of the TDL is connected to the D input of D-type Flip-Flops (DFFs). A digital low-to-high step signal (called STOP) is used as the clock for the DFFs. In this way, the propagation of the START signal through the TDL, with a quantization error proportional to the propagation delays, is captured at the Q outputs. This method yields a sequence of consecutive high-logic values known as a thermometer code, which is proportional to the duration of the time interval delimited by the START and STOP signals. For easier management of the time interval (i.e., numerical representation of the elapsed time between the START and STOP events) captured by the DFFs, the thermometer code is compressed into pure binary, using a thermometer-to-binary converter, referred to as a decoder or encoder in the literature. The two main architectures used are the “sum1s” approach [21,22], which counts the number of 1s in the thermometer code, and the “Log2” method [23], also known as “one-hot” [24], which detects the transition from 1 to 0 (the “one-hot” decoder presented in [24] searches for the most significant 0–1 transition, effectively performing the base-2 logarithm “Log2” of the thermometer code as presented in [23]).



**Figure 1.** Schematic view of TDL TDC implementation, where each buffer output is sampled by a DFF followed by a decoder.

To be more precise, if all the  $N_T$  taps that compose the TDL have the same propagation delay  $\overline{\tau_p}$  then the time interval  $\Delta T$  can be calculated simply by using (1), where  $n$  is the output of the thermometric-to-binary converter:

$$\Delta T = T_{STOP} - T_{START} = n \times \overline{\tau_p} \quad (1)$$

Referring to Equation (1), it is evident that the resolution (LSB) of the TDL TDC is  $\overline{\tau}_p$ , the quantization noise  $\overline{\tau}_p/\sqrt{12}$  (i.e.,  $\varepsilon_Q^2 = LSB^2/12$ ) [25,26] (the digitization process introduces a quantization error ranging between  $-\overline{\tau}_p/2$  and  $+\overline{\tau}_p/2$ , with a uniform distribution characterized by a variance of  $\overline{\tau}_p^2/12$ , thereby defining the measurement precision), and the Full-Scale Range (FSR)  $N_T \times \overline{\tau}_p$ . Therefore, in the presence of jitter (i.e.,  $\sigma_j$ ) between the START and STOP signals, due to electronic system noise, the measurement precision (i.e.,  $\sigma_{\Delta T}$ ) of the measured time interval (i.e.,  $\Delta T$ ) results [27]:

$$\sigma_{\Delta T}^2 = \varepsilon_Q^2 + \sigma_j^2 = \frac{\overline{\tau}_p^2}{12} + \sigma_j^2 \quad (2)$$

### 1.3. Tapped Delay Line Calibration

However, due to strong Process, Voltage, and Temperature (PVT) fluctuations, this linear approach does not properly fit FPGA technologies. In fact, each tap has a dispersed propagation delay (i.e.,  $\tau_p[k]$  with  $k \in [0; N_T - 1]$ ) that strongly differs from the average  $\overline{\tau}_p$ . PVT fluctuations, if high resolution, precision, and linearity are required, necessitate calibrated operation. The most efficient algorithm is known in the scientific literature as “bin-by-bin” calibration, in which, thanks to a Code Density Test (CDT), the propagation delay of each tap is estimated with a negligible error and stored in a so-called Calibration Table (CT) [14,21,28], i.e.,  $CT[k] \simeq \tau_p[k]$  with  $k \in [0; N]$ . Thus, with proper integration of the CT, the Characteristic Curve (CC) is generated, which assigns a timestamp correcting PVT fluctuations to each decoded thermometric code  $n \in [0; N_T - 1]$ :

$$\Delta T = T_{STOP} - T_{START} = CC[n] \quad (3)$$

Under this condition, the resolution (i.e., LSB) is better-represented by the distribution of the CT and, roughly, by the average propagation delay  $\overline{\tau}_p$  (i.e.,  $\overline{\tau}_p = \sum_k \tau_p[k]/N_T \simeq \sum_k CT[k]/N_T$ ), while the precision is represented by the so-called Equivalent LSB ( $LSB_{EQ}$ ) [26], whose mathematical expression is expressed in Equation (4). Thus, due to the propagation delays distortion, the quantization error is proportional to the  $LSB_{EQ}$  (i.e.,  $\varepsilon_Q^2 = LSB_{EQ}^2/12$ ) rather than the resolution (i.e.,  $\varepsilon_Q^2 \neq \overline{\tau}_p^2/12$ ):

$$LSB_{EQ}^2 = \frac{\sum_k \tau_p^3[k]}{\sum_k \tau_p[k]} \simeq \frac{\sum_k CT^3[k]}{\sum_k CT[k]} \quad (4)$$

### 1.4. Tapped Delay Line Decoding

Considering the fact that PVT fluctuations can be compensated by calibration, the main criticality in TDL TDC is the Bubble Error (BE) [29–32], which is a switching effect in the thermometer code’s uniform pattern. Instead of the output being a continuous string of high levels followed by zeros, which is typical of a proper thermometer code, the DFFs and their connections have non-idealities and mismatches that could cause irregularities (i.e., one or more zeros that show up as bubbles in the continuity of ones). For example, in an 8-tap-long TDL, the output might be “11111010” rather than “11111110”. BEs result from deterministic non-linearities in TDL propagation (e.g., skews) [33] or from stochastic processes (e.g., sampling mistakes due to setup and hold time violations) [34]. The presence of BEs strongly impacts the output of the thermometer-to-binary converter, reducing the resolution, precision, and linearity. Moreover, the behavior of the pure binary output is also influenced by the architecture of the decoder. If the “sum1s” approach is used, the BEs are compressed (e.g., “11111010” is interpreted as “11111100”), while the “Log2” method neglects the presence of bubbles (e.g., “11111010” is interpreted as “11111110”). Due to these opposite behaviors, “sum1s” and “Log2” are also known as “bubble compression” or “ones-counter” [21] and “one-hot” algorithms [24], respectively.

### 1.5. Tapped Delay Line Sub-Interpolation

In order to achieve TDL TDCs characterized by better resolution compared to that offered by the average propagation delay of the FPGA technology node  $\overline{\tau_p}$  (Table 1), sub-interpolation techniques [20,35,36] and other exotic TDL-based structures [37] have been introduced in the scientific literature. These techniques involve either repeating the measurement on multiple TDLs (i.e.,  $N_{TDL}$ ) placed in parallel [21] (i.e., spatial sub-interpolation), using only one TDL with each tap sampled twice (i.e.,  $N_{TDL} = 2$ ) by doubling the DFFs (i.e., dual sampling) [38], or cycling through multiple times (i.e.,  $N_{TDL}$ ) via feedback on the same TDL (i.e., temporal sub-interpolation) [39], in order to obtain a Virtual TDL (VTDL) characterized by propagation delays of  $N_{TDL}$  being faster, thus providing an improvement in resolution. Mathematically, a VTDL of  $N_{TDL}$  sub-interpolation order provides an improvement in  $LSB_{EQ}$  by a factor between  $1/\sqrt{N_{TDL}}$  and  $1/N_{TDL}$ , but it also results in an increase in jitter (i.e.,  $\sigma_j$ ) compared to the non-sub-interpolated one [36]. This increase in jitter is low in spatial sub-interpolation and dramatically high in temporal sub-interpolation, especially in scaled FPGAs (i.e., 28 nm or less). In this scenario, a trade-off arises between resolution (i.e., sub-interpolation order, the number of TDLs in parallel in spatial sub-interpolation for modern FPGAs), precision (i.e., the increase in jitter), and efficiency (i.e., area and power used by the sub-interpolation). One of the most common techniques of temporal sub-interpolation is Wave Union A (WUA), while among the temporal techniques Wave Union B (WUB) stands out [40]. In WUA, initially two [40] and later even more [41,42] edges are propagated on the same TDL for each event for which a timestamp is desired. Meanwhile, in WUB, the TDL is closed in feedback [40] or connected to a ripple generator [39,43], constructing a sort of multivibrator that allows the event for which a timestamp is desired to be recirculated multiple times in the TDL; the number of re-circulations thus corresponds to the order of sub-interpolation. In this sense, with WUA and WUB using a single TDL an order of sub-interpolation equal to the number of propagated edges (WUA) or the number of re-circulations (WUB) is obtained at the cost of greater decoding complexity, which increases with the number of edges/re-circulations. Moreover, even if we focus solely on precision, as the order of sub-interpolation increases, the jitter associated with each edge/re-circulation also increases, making these techniques less effective at high orders of sub-interpolation.

Research has thus been directed towards spatial sub-interpolation, first by aligning multiple TDLs in parallel [44,45], each propagating a measurement edge (a.k.a. merged TDL) [46,47], and then moving towards the Super Wave Union (SuperWU), where a WUA is performed on each TDL to double the order of sub-interpolation [36]. Among these two techniques, due to a good trade-off between resolution, precision (i.e., reduction of quantization error), and implementation complexity (i.e., complexity of the decoding mechanism for the various edges), the merged TDL has become increasingly popular.

In the scientific literature, on the other hand, it is possible to find other and more efficient TDL-based architectures for increasing resolution, if viewed solely from the perspective of quantization error; among the main ones are the multisampling TDL [48], the Pseudo-Segmented Delay Line (PSDL) [37], and Multiple Time Coding Lines (MTCL) [33]. In multi-sampling TDL [48], a WUA with multiple edges is present not only in the START signal (i.e., the one propagated in the TDL) but also in the STOP signal (i.e., clock-off DFFs), thus allowing the order of sub-interpolation to be amplified but also increasing the decoding complexity. While the PSDL [37] and MTCL [33] resemble multi-TDL techniques, as several TDLs operating in parallel are appropriately temporally offset to minimize quantization error, the dimensioning of the offset implies, compared to the classic multi-TDL, an increase in complexity from the point of view of the placing algorithm.

In the tuned delay line [27], instead, the goal is not only to reduce the quantization error but to create an extremely linear TDL by appropriately choosing the taps among the available carry logic outputs. Whereas, in [49] the Multi-Segment digital TDL is presented, where the same goal is achieved by arranging the taps from multiple TDLs placed in parallel, at the cost of significant computational effort, to identify the optimal combination.



Another matter is the best placement of TDLs within the FPGA to mitigate signal jitter proportional to device occupation density [50].

**Table 1.** Average tap delay per device family.

FPGA Family	Node (nm)	$\overline{\tau_P}$ (ps) <sup>1</sup>	Tested Device	Ref.
Cyclone I	130	70	EP1C20F400C6N	[36]
Cyclone II	90	45	EP2C35F672C6N	[36]
Virtex 5	65	34	XC5V50T	[36]
Spartan 6	45	25	XC6SLX9	[36]
Spartan 6	45	20	XC6SLX75	[26]
Kintex 7	28	11	XC7K160T	[33]
Kintex 7	28	17	XC7K325T	[36]
Artix 7	28	15	XC7A200T	[36]
Kintex UltraScale	20	4.6	XCKU040	[33]
Kintex UltraScale	20	5	XCKU040	[12]

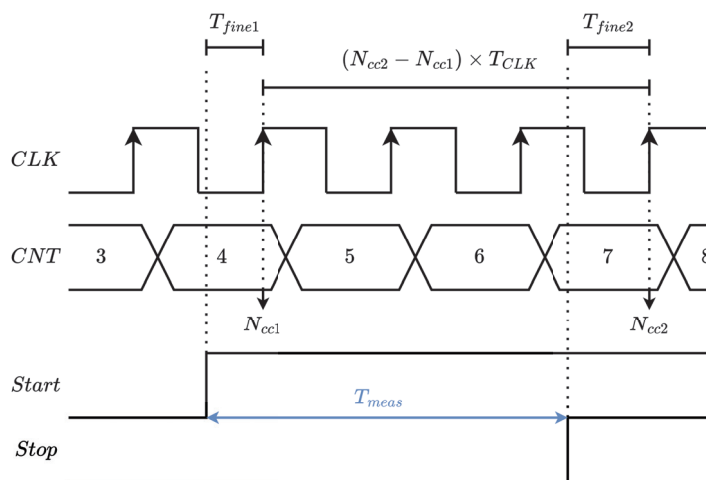
<sup>1</sup> Differences values of  $\overline{\tau_P}$  for the same FPGA family could depend on the speed grade (not specified in the papers) as well as on different experimental setup conditions (e.g., varying measurement temperatures) and the positioning of the TDL within the FPGA.

### 1.6. Nutt Interpolation

As mentioned previously, while TDL TDCs are able to provide good precision and resolution, they suffer from a trade-off between the FSR and area occupancy [51]. This is quite limiting, since the tap propagation delay in modern nodes is in the few-picoseconds range. To break this trade-off, a common method is to employ Nutt interpolation [52]. This technique, as Figure 2 shows, for all channels (e.g., START and STOP) splits the measurement into two parts: the Coarse part measured by an  $N_{CC}$ -bit width Coarse Counter clocked at  $T_{CLK}$  (i.e.,  $N_{cc1}$  for the START and  $N_{cc2}$  for the STOP), and the Fine part performed by the TDL (i.e.,  $T_{fine1}$  for the START and  $T_{fine2}$  for the STOP). Given that the START signal of the event may happen at any time and is not synchronous with the system clock, the measured time in this particular configuration will be

$$T_{meas} = T_{fine1} + (N_{cc2} - N_{cc1}) \times T_{CLK} - T_{fine2} \quad (5)$$

In this way, if the dynamic range of the TDL exceeds  $T_{CLK}$ , the FSR of the integral system is extended up to  $2^{N_{CC}} \times T_{CLK}$ :



**Figure 2.** Timing diagram of system featuring the Nutt interpolation technique.

### 1.7. Purpose of This Contribution

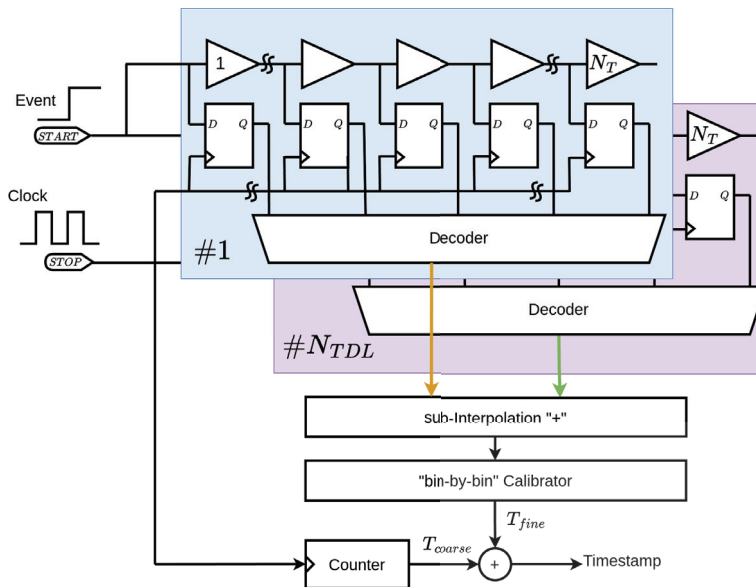
The purpose of these work was to optimize from an LSB ( $LSB_{EQ}$ )-and-precision point of view the structure reported in Figure 3, considering Xilinx 20 nm UltraScale

FPGA devices as targets. This optimization process aims to reduce the performance gap between FPGA-based and ASIC-based TDCs, with ASICs being better-performing and often preferred in cutting-edge applications (e.g., quantum technologies, life sciences, and nuclear physics) that require tens of channels within a square centimeter footprint and picosecond precision. The ultimate goal was to make FPGA-based TDCs usable in cutting-edge applications, as they are more suitable for R&D and rapid prototyping. They offer greater flexibility, lower costs for small production volumes, and a shorter time-to-market compared to ASIC-based solutions.

The research led to the development and comparison of multiple TDL TDC solutions implemented on a Kintex UltraScale device (i.e., XCKU040-2FFVA1156E) housed on a KCU105 general-purpose evaluation board (EVB).

We aimed to implement a TDC in a Xilinx 20 nm Kintex UltraScale FPGA with moderate area consumption capable of handling dozens of channels in a single device and equipped with a real-time decoding-and-calibration algorithm executed directly on the FPGA; thus, we selected the carry-based TDL topology with merged TDL. This choice optimized the trade-offs between linearity, average propagation delay, minimization of quantization error, area occupancy, and place-and-route complexity.

Figure 3 summarizes the architecture of a modern FPGA-based spatial sub-interpolated and Nutt-interpolated TDL TDC.



**Figure 3.** Modern FPGA-based spatial sub-interpolated and Nutt-interpolated TDL TDC.

From this optimization process, two main solutions emerged: one with high precision and another with low area utilization, both offering more than a 4% improvement in precision over the state of the art for this technology node. The first solution achieved an SSP of 2.64 ps r.m.s., with Differential and Integral Non-Linearity (DNL/INL) errors of 0.523 ps and 16.939 ps, respectively. This design occupied 883 CLBs and 126 kb of BRAM (i.e., 3.5 blocks), allowing up to 24 channels to be implemented on the selected  $35 \times 35$  mm FPGA. The second solution, with an SSP of 3.75 ps r.m.s., a DNL of 0.599 ps, and an INL of 7.151 ps, occupied only 259 CLBs and 72 kb of BRAM (i.e., 2.5 blocks), which enabled up to 64 channels to be implemented on the target FPGA.

### 1.8. Paper Organization

This paper is organized as follows: Section 2 discusses all the relevant UltraScale device features from a TDL TDC point of view. As shown in Section 3, multiple TDL TDC solutions were implemented and tested. A comparison with a state-of-the-art FPGA-based TDL TDC implemented in Xilinx 20 nm UltraScale FPGA is conducted in Section 4.

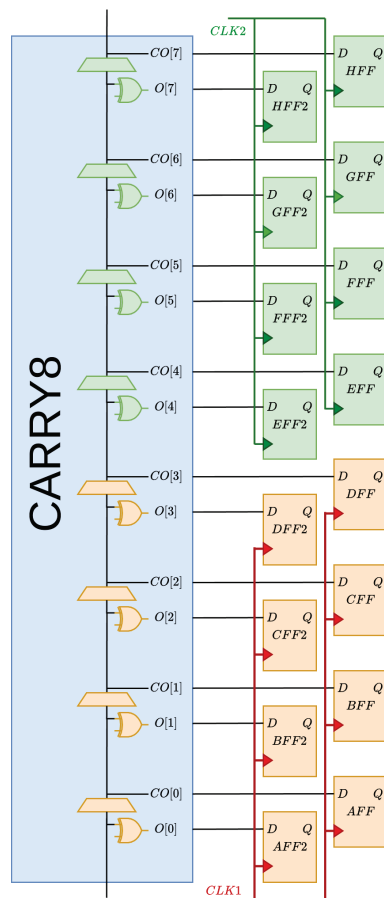
## 2. TDL TDC Architecture in 20 nm Xilinx UltraScale FPGA

In this Section, to discuss both the technological and architectural properties of the 20 nm Xilinx UltraScale FPGAs, we must take a bottom-up approach, starting from the CLB, the carry logic primitive (i.e., CARRY8), the clock distribution scheme in Section 2.2, and then addressing the CARRY8-based TDL itself in Section 2.3, as well as its specific placements inside the device. Lastly, the decoding policies are summarized in Section 2.4.

### 2.1. Configurable Logic Block Primitives

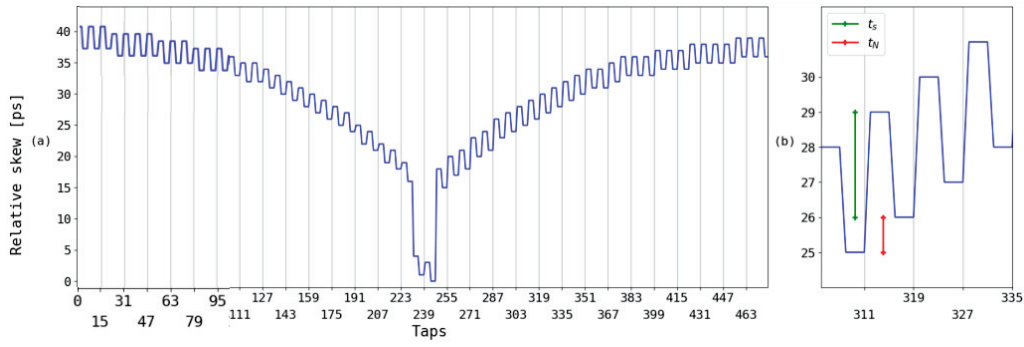
In UltraScale, each CLB has a CARRY8 primitive with 8 outputs (numbered 0 to 7): for each output, two distinct signals are available, the CO and the O, which, for our purpose, were each the negation of the other. Each of these, now, 16 outputs is sampled by the corresponding lettered DFF (i.e., CO[0] to AFF, O[0] to AFF2). The CLB is split into two parts: the bottom part containing 8 DFFs (A to D and A1 to D1), and the top part containing 8 DFFs (E to H and E1 to H1).

The CARRY8 thus functions as an 8-tap TDL that propagates the input signal from the bottom to the top of the CLB (i.e., from CO[0] and O[0] to CO[7] and O[7]). Similar to previous Xilinx technology nodes (i.e., 28 nm, 40 nm, and 45 nm) [27], if taken independently then all CO and O outputs have a similar delay between each tap [12,53]; as such, to build a TDL we only have to sample one of them; however, in the same CARRY8, the first 4 taps and the last 4 taps may have different skews. This can be observed in UG574 [54] and in Figure 4. By running a Vivado Post-Implementation timing analysis, we can see how this independent clocking structure has an effect on the clock skew inside the CLB, as shown in Figure 5:



**Figure 4.** Independent clocking scheme of the two halves (i.e., red and green) of the CLB with DFFs and CARRY8 [54].



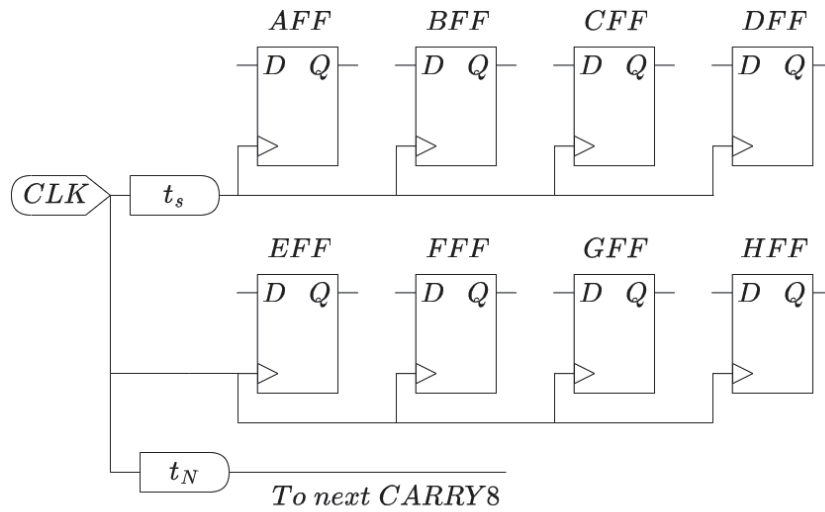


**Figure 5.** Clock skew pattern among taps, where the horizontal lines identify each CLB, with (a) increasing values starting from insertion point (roughly in correspondence to tap 247) and (b) different propagation times of the clock into the CLB. In (b), we can also see the local CLB skew ( $t_s$ ) and the skew between the CLBs ( $t_N$ ).

Observing Figure 5, we can see that the clock skew exhibits two main patterns:

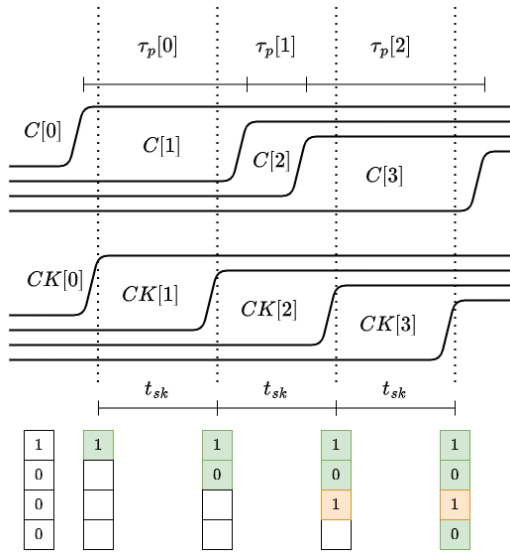
- an inter-CLB trend, which is controlled by the clock distribution network, where we see a delay accumulation (i.e.,  $t_N$ ) CLB by CLB from the clock insertion point (roughly in correspondence to tap 240 in Figure 5);
- an intra-CLB trend, a local skew pattern (i.e.,  $t_s$ ), with the clock arriving at the second half of the primitive earlier than the first.

This allows us to model the clock distribution network of the CARRY8 primitive (Figure 6). To put it numerically, as also shown previously in Figure 5, the  $t_s$  and  $t_N$  values are below a few ps, e.g.,  $t_s \simeq 3$  ps and  $t_N \simeq 1$  ps.



**Figure 6.** Model of the clock skew from the second half to the first one, represented by  $t_s$ , and from the second half to the second one, represented by  $t_N$ .

These figures must, however, be taken into account in light of the TDL, particularly with regard to the tap (C or O) propagation delay (i.e.,  $\tau_p[k]$  with  $k \in [0;7]$ ). Indeed, in cases when the tap propagation delay falls short of  $t_N$ , we can run into skew-induced BEs. More logically, because there is a non-negligible amount of skew, the output code is compiled “progressively” rather than “simultaneously”. In general, if the tap propagation delay is less than the skew between two sampling DFFs, it is possible that the values, which propagate asynchronously, will skip a few taps, as illustrated graphically in Figure 7, where  $C[k]$  represents the CARRY8 outputs (i.e., the  $D$  pins of the DFFs),  $CK[n]$  represents the  $C$  pins of the DFFs, and the tap propagation delays are represented by  $\tau_p[k]$ :

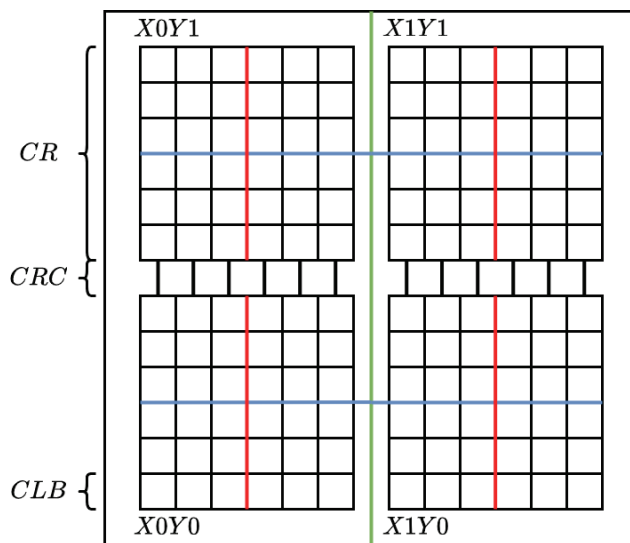


**Figure 7.** Generation mechanism of the BEs. In particular, showing the situation in which a single tap has a smaller propagation delay ( $\tau_p$ ) than the clock skew between the sampling FFs ( $t_{sk}$ ).

In our case, this is the situation where a signal is traveling in the second half of the CLB, and after being sampled in that area it keeps propagating in the next one, having a skew of  $t_s + t_N$ .

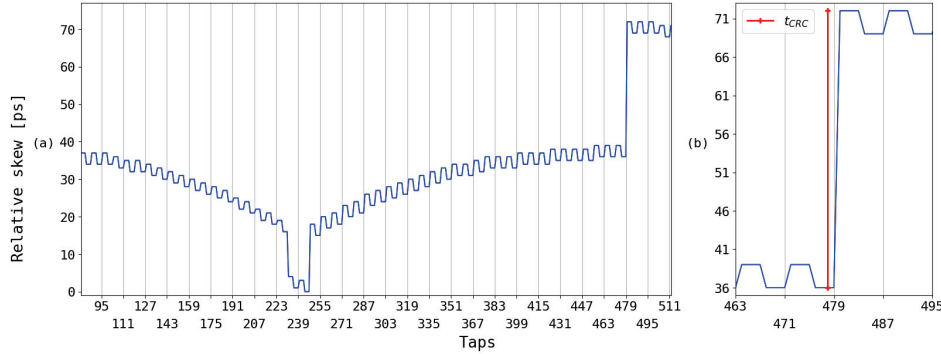
## 2.2. Clock Regions and the Clock Region Crossing Issue

Xilinx FPGAs are organized in Clock Regions (CRs), that are groups of CLBs, characterized by internal negligible skews; CRs are spaced to adjacent ones by means of Clock Region Crossing (CRC), and they are connected by proper clocks lines characterized, across the CRC, with a clock skew that is more than an order-of-magnitude higher, with respect to  $t_s$  and  $t_N$ . As illustrated in Figure 8, the clock signal is distributed among different CRs vertically via the clock distribution network from the backbone (green) and horizontally via horizontal lines (violet). Meanwhile, as described in Section 2.1, within the CR, the inter-clock region routing (red) is used. Further details are available in UG94 [55].



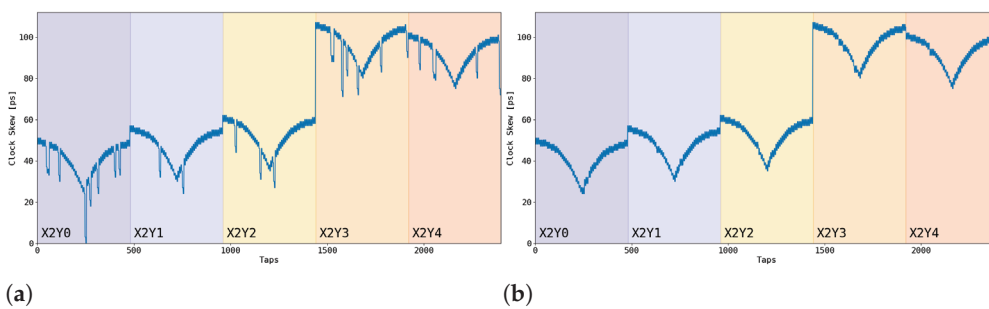
**Figure 8.** Schematic view of a generic Xilinx FPGA, subdivided into CRs (i.e., X0Y0, X0Y1, X1Y0, and X1Y1) spaced by CRC. The connection in between the clock region is shown, as well as the clock distribution network from the backbone (green) to the horizontal lines (violet) to the inter-clock region routing (red).

The skew diagram from Figure 5 has been expanded across the CR, and it is shown in Figure 9, to better represent this issue. This is especially critical, because at the CRC the clock skew (i.e.,  $t_{CRC}$ ) is  $\approx 35$  ps, which is guaranteed to result in BEs, if negative, and ultra-bin, if positive.



**Figure 9.** Clock skew pattern along two CRs (i.e., the CRC is between tap 479 and 480), with (a) showing the overall trend and (b) showing the sudden increase in clock skew ( $t_{CRC}$ ).

Another noteworthy aspect to investigate was the absence of continuity in the clock skew diagram within the CLB (taps 223–255 in Figures 5 and 9) compared to the 28 nm 7-Series [33]. We repeated the tests in different CLBs (Figure 10), consistently observing the absence of continuity. Considering only data collected in post-implementation firmware, we hypothesize that this was due to the non-ideal nature of the FPGA fabric. This discontinuity contributes to the creation of an ultra-bin in the TDL. The discontinuities observed within the CLB, unlike what happened with the TDC in the 7-Series [33], were detected only when the FPGA was programmed with the TDC firmware. Moreover, such discontinuities, when varying the implementation of the TDC (i.e., order of sub-interpolation and type of decoder), remain almost identical in magnitude although they may manifest with slightly different numbers and positions. Another important point is that, from an architectural standpoint, not only the division of the CLBs underwent changes between the 28 nm 7-Series [56] and the 20 nm UltraScale [54] but also the clock distribution network [57]. Consequently, the Xilinx 20 nm technology is not simply a scaled-down version of the 28 nm but has its own architecture, both in terms of clock distribution and fabric organization.



**Figure 10.** Clock skew pattern along many CRs with TDC firmware (a) and without (b).

### 2.3. Tapped Delay Line

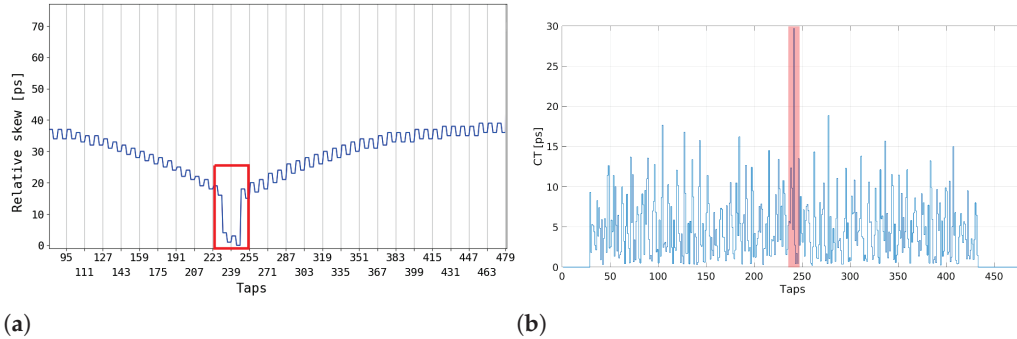
A complete TDL is created by concatenating the CARRY8 primitives. Therefore, the mean tap of delay  $\overline{\tau_P}$  multiplied by the total number of taps provides a first-order estimate of the FSR of the TDL. Additionally, to overcome the CRC issue exposed in Section 2.2, it is necessary to fit the TDL inside the CR. In this context, the longest TDL segment that can fit inside a CR is a Figure of Merit (FoM), in relation to the FPGA family. However, since the CARRY8 primitive (i.e., that offers only  $N_C = 8$  taps) can only be concatenated vertically (i.e., from the bottom to the top), the maximum length depends on the number of CLBs that are present vertically in a CR (i.e.,  $N_{CLB} = 60$ ) [54]. This restricts the span of the TDL

to up to 480 taps (i.e.,  $N_{CLB} \times N_C$ ) in Xilinx 20/16 nm UltraScale and UltraScale+ FPGA. In our system, considering that the FSR of the TDL is constrained by the Nutt interpolation, we can estimate the maximum clock period  $T_{CLK}^{MAX}$ , as shown in Equation (6), assuming, in the first approximation, that the tap propagation delay is uniform and equal to the average value  $\bar{\tau}_p$  (i.e., 5 ps from Table 1):

$$T_{CLK}^{MAX} \leq N_{CLB} \times N_C \times \bar{\tau}_p = 60 \times 8 \times 5 \text{ ps} = 2.4 \text{ ns} \quad (6)$$

We require at least  $T_{CLK}^{MAX} < 2.4 \text{ ns}$ . Adding an engineering margin factor, we finally settle on  $T_{CLK}^{MAX} = 2 \text{ ns}$  (i.e., 500 MHz, in terms of frequency) and a TDL composed of 512 (since the decoding block is modular to powers of 2, we approximate to the next-higher power) taps (of which the last 32 will never be reached), because it is well within the device's capability, which is defined between 630 MHz and 850 MHz, depending on the speed grade of the device [58]. The high clock frequency available made it possible to implement a TDL sampled from the same clock source without requiring the use of multi-clock techniques, thus avoiding architectural complications [26].

Once the vertical length and placing has been addressed, there may be some factors to take into account while determining its precise horizontal placement. The FPGA does, in fact, display a non-uniform structure, cycling between CLB, RAM, and DSP columns. In this regard, referring to the discontinuities at taps 223–255 in Figures 5 and 9, we can observe the presence of an ultra-bin in the same position (i.e., tap 241) of the CT (Figure 11) (for this reason, we attributed the skew to the interaction between the firmware and the non-uniform structure of the FPGA, and we hypothesize that the highlighted skew is the cause of the ultra-bin). Since such discontinuities are too temporally close, any TDL that excludes them would result in a  $T_{CLK}^{MAX}$  that would be too small, and, therefore, physically could not be implemented.



**Figure 11.** CR's clock skew pattern with discontinuity at taps 223–255 highlighted (a) and the relative CT with an ultra-bin at tap 241 highlighted (b).

#### 2.4. Decoder

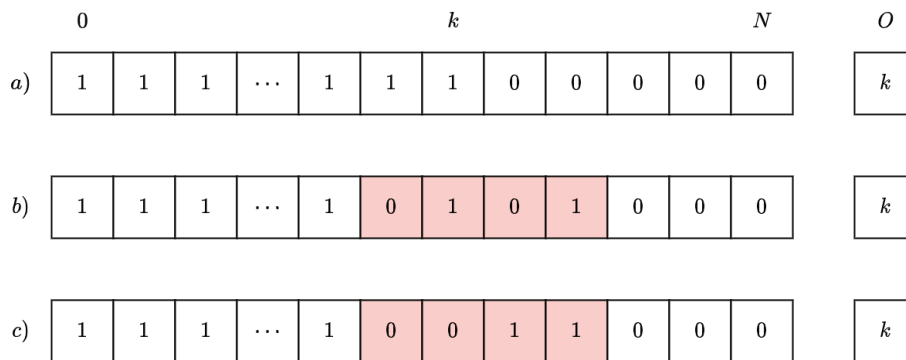
In light of the TDL TDC architecture that we previously discussed, the thermometric code spreads down the TDL before being sampled by the DFFs. According to this theory, if there are no BEs, the code's one-to-one ratio corresponds exactly to the event's timestamp. As a result, the thermometric code sampled by a  $N = 2^{N_B}$ -long TDL can be remapped into a  $N_B$ -long unsigned integer that indicates which has been hit. We will now compare the two main decoder architectures.

From the scientific literature, considering only one TDL, whether it be Log2 or sum1s, the decoders are always  $N_B$ -stage pipeline structures that proceed dichotomously. The input stage has a number of single-bit inputs  $N$  equal to or bigger than the size of the TDL, which is 480. For this reason, in this paper, considering that 480 is not a power of two, the decoders are designed with 9 pipeline stages (i.e.,  $N_B = 9$ ) and 512 single-bit inputs (i.e.,  $N = 512$ ), where the last unused 32 single-bit inputs (i.e.,  $N - N_T = 512 - 480$ ) are connected to "0".

To perform the sub-interpolation over more TDLs (i.e.,  $N_{TDL}$ ),  $N_{TDL}$  identical decoders (i.e., all sum1s or all Log2) are required. The  $N_{TDL}$  outputs are summed together, using a tree adder with  $\lceil \log_2(N_{TDL}) \rceil$  pipeline stages (i.e., 1 for  $N_{TDL} = 2$ , 2 for  $N_{TDL} \in \{3; 4\}$ , 3 for  $N_{TDL} \in \{5; 6; 7; 8\}$ ).

#### 2.4.1. The sum1s Decoder

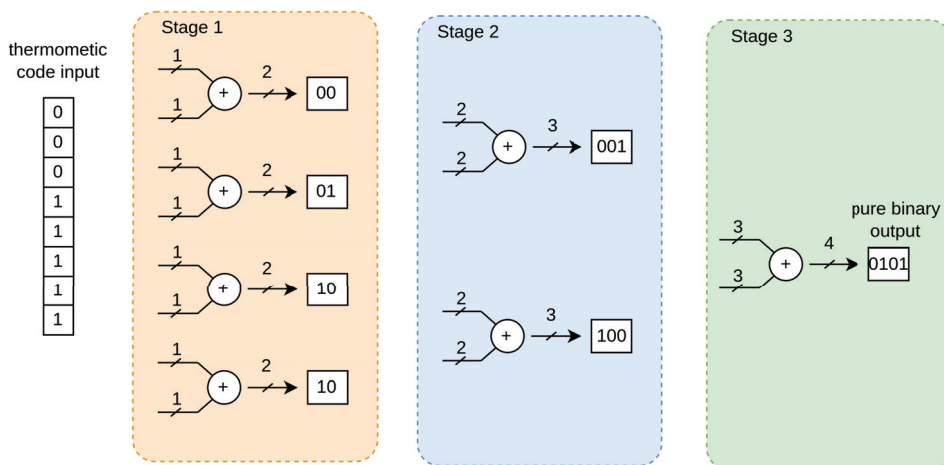
This decoding scheme is the easiest and most direct, especially if there are no BEs, and it consists in counting the instances of “ones” inside the thermometric code. Nevertheless, if BEs are present, the sum1s approach applies bubble compression, meaning that several codes with the same number of “ones” may correlate to the same output. Figure 12 is an illustration of this:



**Figure 12.** Example of bubble compression where codes a, b, and c return the same output (red highlights indicate BEs).

The sum1s decoder is usually organized as tree adders (unlike [21], in this paper this module is described in VHDL behavioral modeling style, and it is left to Vivado to assign the LUTs and DFFs appropriately), where the stage  $i$ -th (with  $i \in [1; 9]$ ) has  $2^{N_B - (i-1)}$  inputs of  $i$  bits and  $2^{N_B - (i-1)} / 2$  outputs of  $i + 1$  bits (i.e.,  $2^{9 - (i-1)}$  inputs of  $i$  bits and  $2^{9 - (i-1)} / 2$  outputs of  $i + 1$  bits in this implementation). In Figure 13, an example is shown, with  $N_B = 3$  (i.e.,  $N = 8$ ).

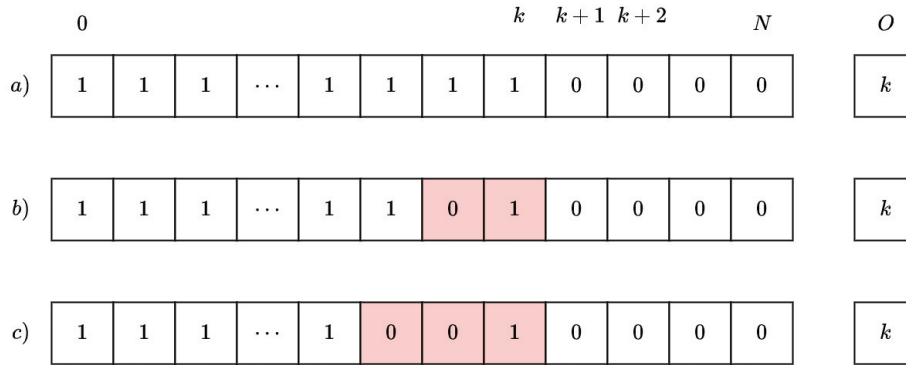
With an input thermometer code of  $N$  bits where the LSB is always “1” the output code is an  $N_B + 1$ -bit number ranging from 1 to  $N$ . Therefore, typically, to save area, thermometer codes are remapped, such that their conversion into pure binary results in an  $N_B$  bit number ranging from 0 to  $N$ . In the example shown in Figure 13, we thus obtain “100” (i.e., decimal 4) as the output binary instead of “0101” (e.g., decimal 5):



**Figure 13.** Example of a sum1s decoder with  $N_B = 3$  (i.e.,  $N = 8$ ) and its corresponding data flow for decoding the thermometer code “0001111” into pure binary “0101” (i.e., decimal 5).

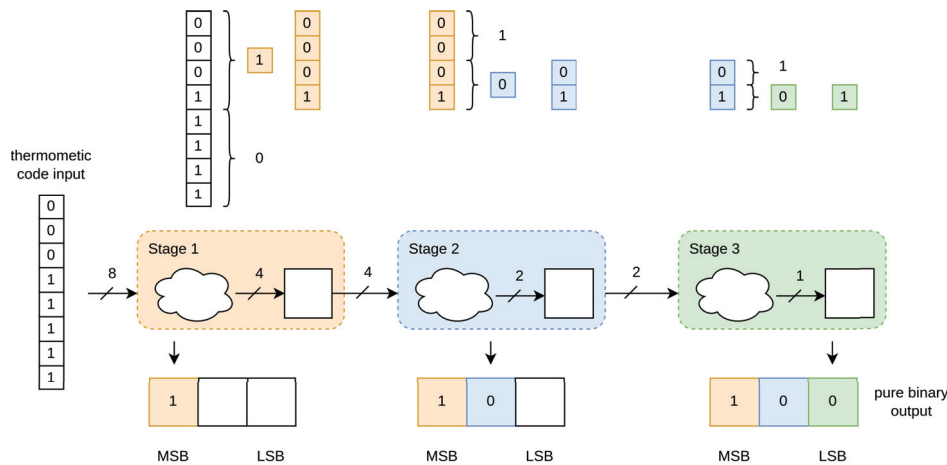
### 2.4.2. The Log2 Decoder

Building on the presumption shared with the sum1s decoder, it is clear that the population of the most significant bit (MSB) in the TDL is correlated with the total number of “ones” in the thermometric code. With this knowledge, a more effective binary search strategy can be used to identify the MSB, as opposed to thoroughly going over the full code sequence. In this way, BEs are ignored. Figure 14 shows how the Log2 Decoder operates:



**Figure 14.** Example of insensitivity to BEs in Log2 decoder where a, b, and c will have the same output (red highlights indicate BEs).

The stages, in the case of Log2 decoders, are generally characterized by  $2^{N_B-(i-1)}$  inputs and  $2^{N_B-(i-1)}/2$  outputs (i.e.,  $2^{9-(i-1)}$  inputs and  $2^{9-(i-1)}/2$  outputs in this implementation), which propagate the portion of the TDL where the most significant 1–0 transition occurs in the next stage. So, during the propagation the pure binary output is compiled from the most significant bit to the least significant bit, resembling a successive-approximation register. In Figure 15, an example is shown, with  $N_B = 3$  (i.e.,  $N = 8$ ):



**Figure 15.** Example of a Log2 decoder with  $N_B = 3$  (i.e.,  $N = 8$ ) and its corresponding data flow for decoding the thermometer code “0001111” into pure binary “100” (i.e., decimal 4).

### 2.4.3. A Brief Comparison

If we consider multi-channel systems, which are nowadays the most requested in cutting-edge applications, efficient resource utilization is a critical FoM for the design. In the FPGA context, resource usage is typically measured in terms of the number of CLBs consumed by the design. Each CLB contains a limited number of Look-Up Tables (LUTs) (eight, in this case), which are basic combinational elements capable of implementing any arbitrary binary function, DFFs (16 per CLB), and a CARRY8 element (1 per CLB). To provide a clear comparison, we evaluated the resource utilization of an  $N_B = 8$  Log2 and sum1s decoders for 480-tap-long TDLs (i.e.,  $N = 512$ ). This comparison is summarized



in Table 2, which presents the maximum clock frequencies ensuring timing enclosure (i.e., Max. Freq.) [59], as well as the area utilization computed post-implementation by Vivado [60]. The area utilization is detailed not only by counting the number of occupied CLBs but also by reporting the number of individual LUTs and DFFs [61]. It is evident that if spatial sub-interpolation is required ( $N_{TDL} > 1$ ) then the area occupancy scales linearly for both solutions. Although the Log2 decoder demanded fewer resources compared to the sum1s decoder, it performed less well, in terms of timing.

**Table 2.** Comparison of resources necessary for sum1s and Log2 decoders.

$N_{TDL}$	Type	LUTs	DFFs <sup>1</sup>	CARRY8 <sup>2</sup>	CLB	Max Freq. [MHz] <sup>3</sup>
1	sum1s	808	1021	18	160	664
	Log2	602	593	24	98	527
2	sum1s	1620	2042	38	288	704
	Log2	1209	1185	50	183	500
4	sum1s	3247	4084	78	568	593
	Log2	2423	2366	102	450	541
8	sum1s	6502	8168	158	979	560
	Log2	4885	4719	206	839	500

<sup>1</sup> These DFFs do not take into account the 512 DFFs used to sample the 512-taps-long TDL. <sup>2</sup> These CARRY8 primitives do not count the 64 used to implement the 512-taps-long TDL (i.e.,  $64 \times 8$ ), but refer to the CARRY8 blocks used to perform addition operations within the decoder. <sup>3</sup> This parameter is calculated as the reciprocal of the slowest path.

### 3. Experimental Results

Several 2-channel (i.e., START vs. STOP) TDL TDCs characterized by different orders of sub-interpolation with different decoders (e.g., sum1s vs. Log2) were implemented on a Kintex UltraScale device (i.e., XCKU040-2FFVA1156E) hosted on a KCU105 general-purpose evaluation board, to identify the best-performing solutions, in terms of precision. The experimental setup used is described in Section 3.1, while the measurements to determine the optimal sub-interpolation order were carried out as described in Section 3.2. The precision and linearity measurements are reported in Sections 3.3, 3.4, and 3.5, respectively. Section 3.6 contains the final considerations.

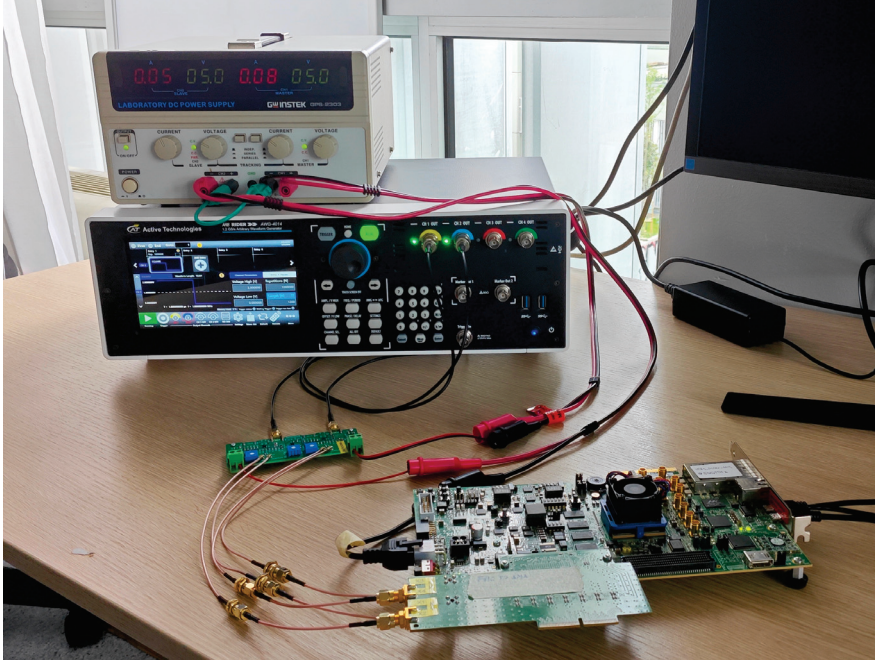
#### 3.1. Experimental Setup

Figure 16 shows the experimental setup, where the KCU105 (the larger green EVB at the bottom-right) is depicted, connected via Low-Voltage Differential Signaling (LVDS) standard and SMA cables to START and STOP signals generated by an AWG-4014 Arbitrary Function Generator [62] from Active Technologies (black instrument in the center). Between the AWG-4041 and the KCU105 there was a board (the small green PCB in the center) composed of two HMC674LP3E comparators (with nominal jitter, i.e.,  $<0.2$  ps r.m.s.) tasked with converting the single-ended signal from the AWG-4041 into LVDS. As schematically depicted in Figure 16, the 2-channel (i.e., START vs. STOP) TDL TDCs could measure external signals coming from the AWG-4041 or internal signals generated by a simple logic hosted inside the FPGA, clocked by a clock (CLK1 in Figure 17) uncorrelated to that of the TDC (CLK0 in Figure 17).

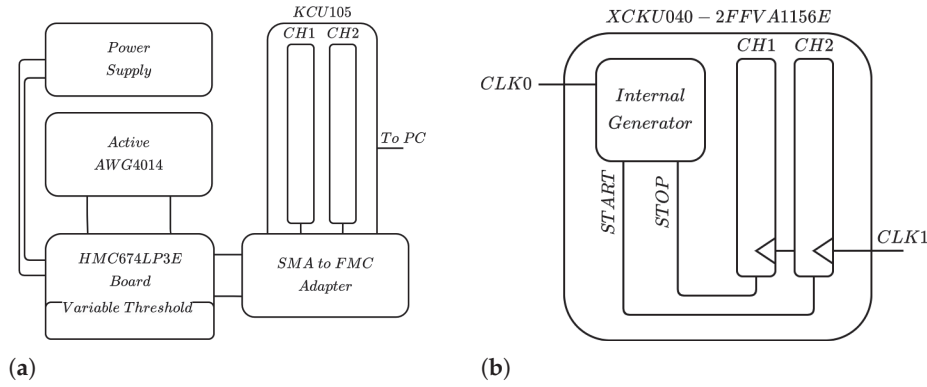
Internal signals were used to evaluate quantization error and, thus, estimate the optimal order of sub-interpolation (Section 3.2), while external signals were used for the final validation (Sections 3.3–3.5), i.e., characterizing precision as a function of measured delay, single-shot channel precision, and linearity.

The timestamps used to perform the measurements were then acquired, and a histogram of the time difference between START and STOP was computed directly in real time inside the FPGA, using a proper hardware histogram module [63].





**Figure 16.** Experimental setup: the board had two independent oscillators (CLK0 and CLK1), ensuring there was no correlation between CLK0 and CLK1.



**Figure 17.** Schematic view of the experimental setup for the external signals (a) and for the internal ones (b).

### 3.2. Quantization Error and Sub-Interpolation Order

In order to represent the quantization error that, as can be seen in (2), represented the best theoretical achievable precision, our first analysis used the computation of the  $LSB_{EQ}$  from the CT, as shown in Equation (4). When we had 2 channels in our TDC, with START being the channel that received the first event and STOP being the channel that received the last event, then we had to consider both the quantization errors, as shown in Equation (7), i.e.,  $LSB_{EQ,START}/\sqrt{12}$  for START and  $LSB_{EQ,STOP}/\sqrt{12}$  for STOP, respectively:

$$\epsilon_Q^2 = \frac{LSB_{START}^2 + LSB_{STOP}^2}{12} \quad (7)$$

The quantization error (i.e.,  $\epsilon_Q$ ) could not be directly measured, due to jitters; however, it could be estimated as the mean real precision of the measurements (i.e.,  $\overline{\sigma_{REAL}}$ ) performed with the internal START and STOP signals that offered only the internal jitter offered by the FPGA (i.e.,  $\sigma_j$ ):

$$\overline{\sigma_{REAL}^2} = \sigma_j^2 + \epsilon_Q^2 \quad (8)$$

The findings of this analysis are provided in Table 3 for the sum1s decoder and in Table 4 for the Log2 decoder (considering the UltraScale, the average propagation delay on the TDL was about 5 ps, while due to the non-uniformity of the taps the calculation of the equivalent LSB fluctuated between 8.5 and 9.5 ps). The experiments were conducted with both decoders, as the BEs were handled differently at various spatial sub-interpolation orders (i.e.,  $N_{TDL}$ ). This clearly showed that the sum1s algorithm provided slightly better resolution performance for  $N_{TDL} \geq 2$ , indicating a slightly better handling of BEs compared to the Log2 algorithm in this family of FPGAs, when sub-interpolation was addressed. Therefore, the choice of decoder did not significantly impact the quantization error.

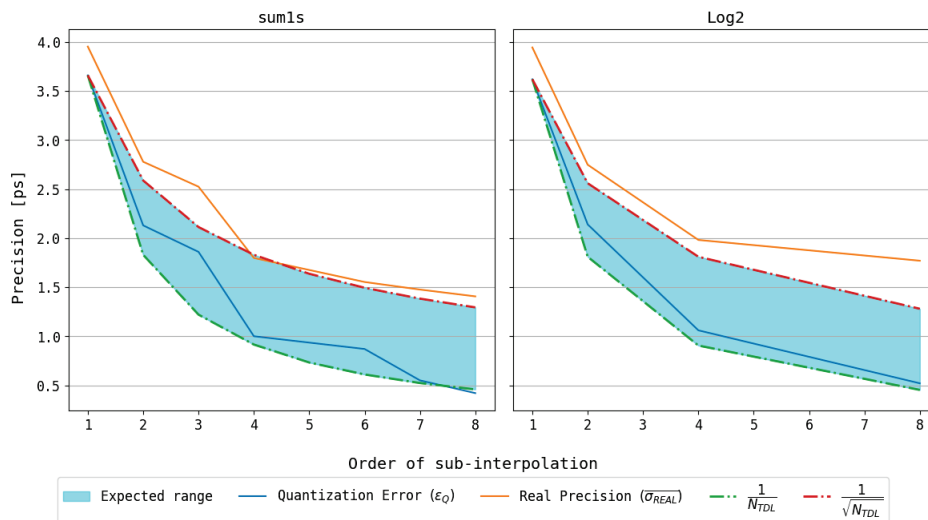
**Table 3.** Sum1s decoder quantization error overview.

$N_{TDL}$	$LSB_{START}^2$ [ps <sup>2</sup> ]	$LSB_{STOP}^2$ [ps <sup>2</sup> ]	$\varepsilon_Q^2$ [ps <sup>2</sup> ]	$\varepsilon_Q$ [ps]	$\overline{\sigma}_{REAL}$ [ps]
1	89.65	71.48	13.43	3.66	3.95
2	28.92	25.41	4.53	2.13	2.78
3	23.30	18.22	3.46	1.86	2.52
4	5.78	6.12	0.99	1	1.8
6	4.37	4.81	0.76	0.87	1.68
7	1.96	1.63	0.3	0.55	1.55
8	1.11	0.99	0.18	0.42	1.48

**Table 4.** Log2 decoder quantization error overview.

$N_{TDL}$	$LSB_{START}^2$ [ps <sup>2</sup> ]	$LSB_{STOP}^2$ [ps <sup>2</sup> ]	$\varepsilon_Q^2$ [ps <sup>2</sup> ]	$\varepsilon_Q$ [ps]	$\overline{\sigma}_{REAL}$ [ps]
1	78.59	78.26	13.07	3.62	3.94
2	25.85	29.09	4.58	2.14	2.75
4	6.16	7.33	1.12	1.06	1.98
8	1.28	2.01	0.27	0.52	1.77

Regarding the sub-interpolation order, as reported in Figure 18, both decoding architectures (sum1s on the left and Log2 on the right) exhibited the expected trend of the quantization error (blue) falling between a  $1/\sqrt{N_{TDL}}$  error (red) and a  $1/N_{TDL}$  error (light green) factor of improvement, and the real precision (orange).



**Figure 18.** Behavior of quantization error (i.e.,  $\varepsilon_Q$  in blue) and real precision of measurements (i.e.,  $\overline{\sigma}_{REAL}$  in orange) as function of the sub-interpolation order  $N_{TDL}$  considering sum1s (left) and Log2 (right) decoders, as reported in Tables 3 and 4, respectively.

### 3.3. Time Sweep

Using the hardware described in Section 3.1, with external signals, we conducted a delay sweep in the interval  $[-2\text{ ns}; 10\text{ ns}]$  to accurately characterize the precision of our implementations considering different decoders and sub-interpolation orders, taking into account both the quantization error (i.e.,  $\varepsilon_Q$ ) and the jitter (i.e.,  $\sigma_j$ ) as a function of the measured delay. These measurements are reported in Figures 19–22, where four different orders of sub-interpolation (i.e.,  $N_{TDL} = 1$  none,  $N_{TDL} = 2$ ,  $N_{TDL} = 4$ , and  $N_{TDL} = 8$ ) are depicted. In the top part of the graph, the precision obtained with the sum1s decoder is shown in red, while that with the Log2 decoder is in blue; the bottom part shows the spread between the two approaches.

Referring to Equation (8), considering that AWG-4014 generated delay with START and STOP with a jitter  $\sigma_{AWG}$ , and considering that HMC674LP3E introduced jitter  $\sigma_{HMC}$  both on the STOP and START signals, then we could derive the total precision ( $\sigma_{SWEEP}$ );

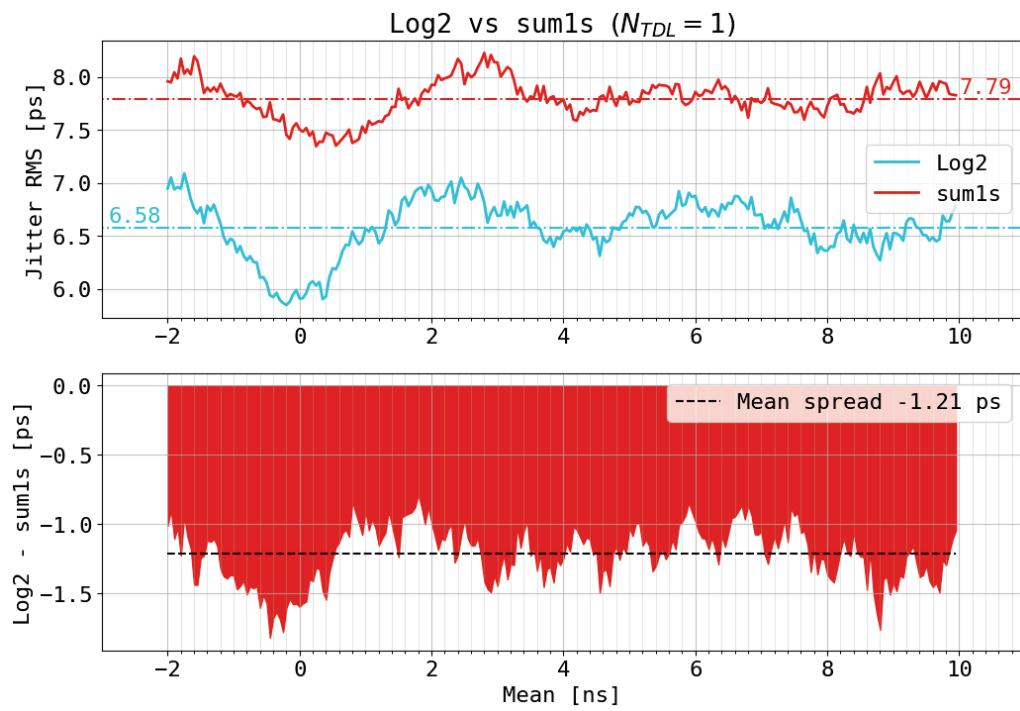
$$\sigma_{SWEEP}^2 = \sigma_{AWG}^2 + 2\sigma_{HMC}^2 + \sigma_j^2 + \varepsilon_Q^2 \quad (9)$$

Moreover, to better highlight the trend of the precision over the sub-interpolation order, the results of Figures 19–22 (i.e., minimum, maximum, average precision, and the amplitude of the amplitude fluctuation of the precision) are also listed in Table 5. Observing Table 5, we demonstrate that the best precision was achieved using  $N_{TDL} = 4$  and sum1s. However, when the minimum area occupancy was the target, observing Figure 20, we argue that the Log2 decoder was the best choice.

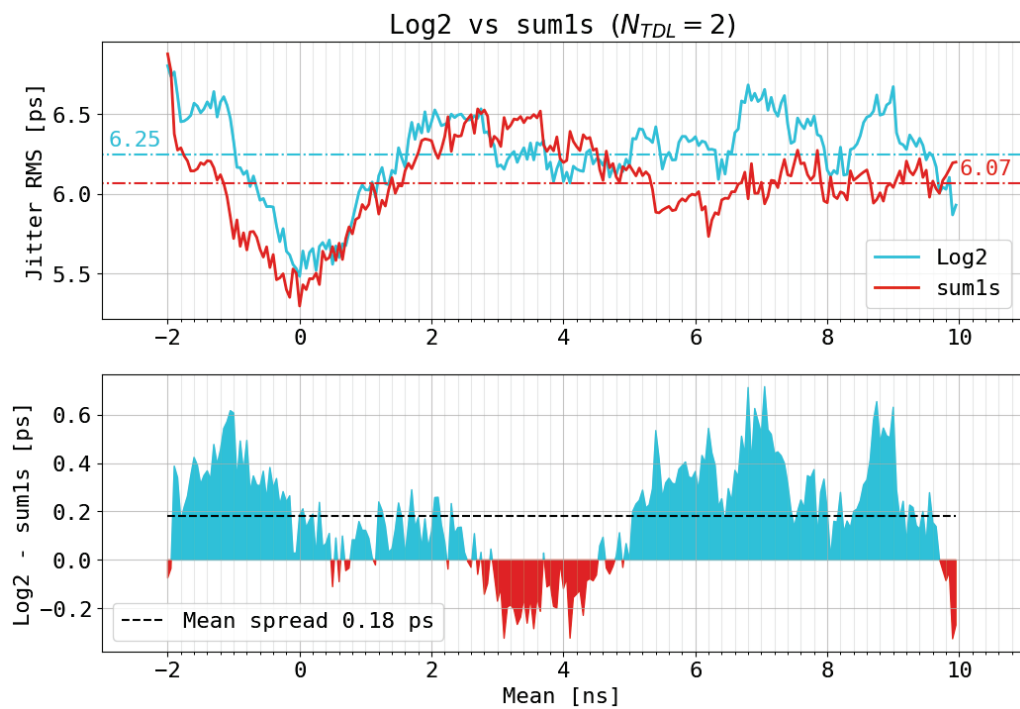
**Table 5.** Summary of performance: (a) sum1s; (b) Log2.

(a)				
$N_{TDL}$	Mean	Measured Precision $\sigma_{SWEEP}$ [ps]		Amplitude
		Max	Min	
1	7.79	8.23	7.35	0.88
2	6.07	6.87	5.3	1.57
3	6.39	7.85	5.42	2.43
4	6.01	7.06	4.54	2.52
6	6.64	7.59	4.86	2.73
7	5.95	7.78	4.35	3.43
8	7.49	11.11	4.61	6.5
(b)				
$N_{TDL}$	Mean	Measured Precision $\sigma_{SWEEP}$ [ps]		Amplitude
		Max	Min	
1	6.58	7.09	5.85	1.24
2	6.25	6.8	5.48	1.32
4	6.03	7.24	4.6	2.64
8	7.95	9.57	6.04	3.53

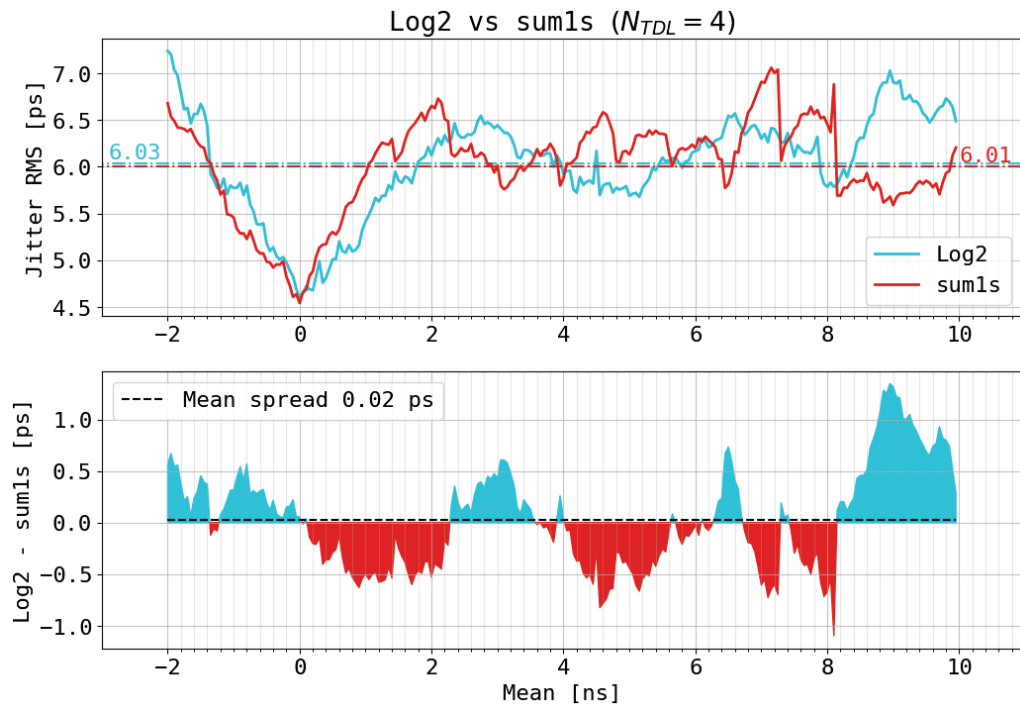
When sub-interpolation was adopted, as shown by Figures 20–22, from a precision perspective, the Log2 and sum1s became effectively comparable, i.e., this means that both decoding policies have the same amount of quantization error. Moreover, we observed an increase in the amplitude of the fluctuation of the precision with the sub-interpolation order. This was justified by an increase in the jitter ( $\sigma_j$ ) due to the sub-interpolation process. Mostly, adding more TDLs caused the decoder and all subsequent blocks to grow in size as well. Congestion and increased electronic noise were thus brought about, mostly as a result of excessive switching activity close to the TDL, which increased the jitter. When we plotted the precision against the sub-interpolation order, as shown in Figure 23, independently of the decoder solutions, we observed a fluctuation of the precision induced by the increasing jitter. In conclusion, we can observe that the system achieved a fair trade-off between maximal precision and stability between two and four TDLs.



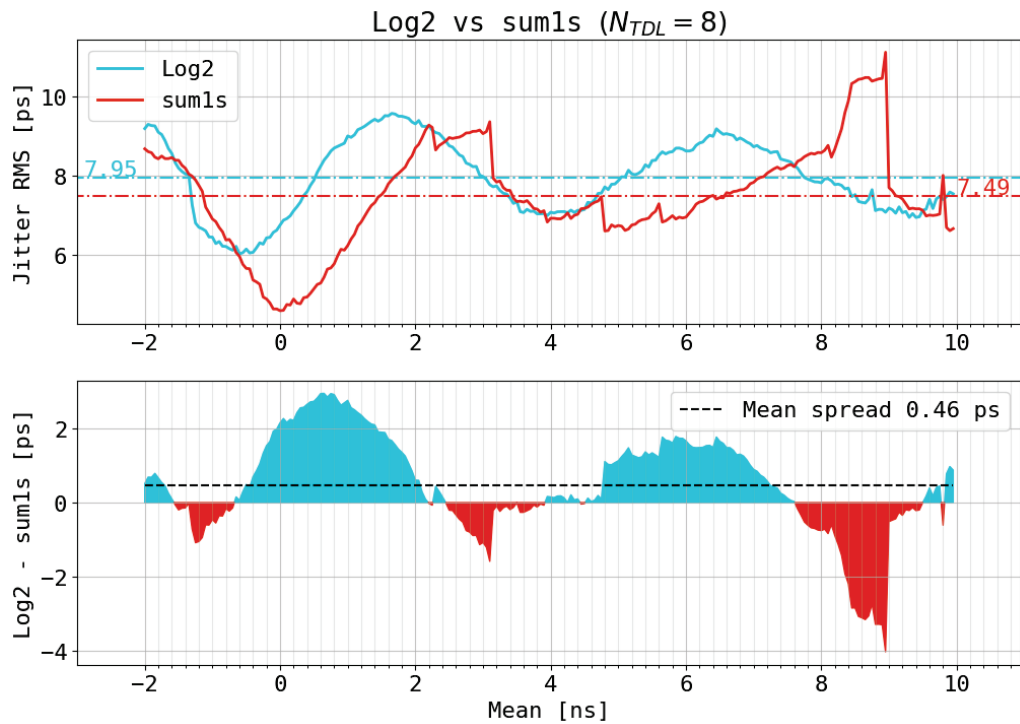
**Figure 19.** Single TDL precision (**top**) and spread (**bottom**) of the sum1 (red) and the Log2 (blue) implementations, with respect to the measured delay.



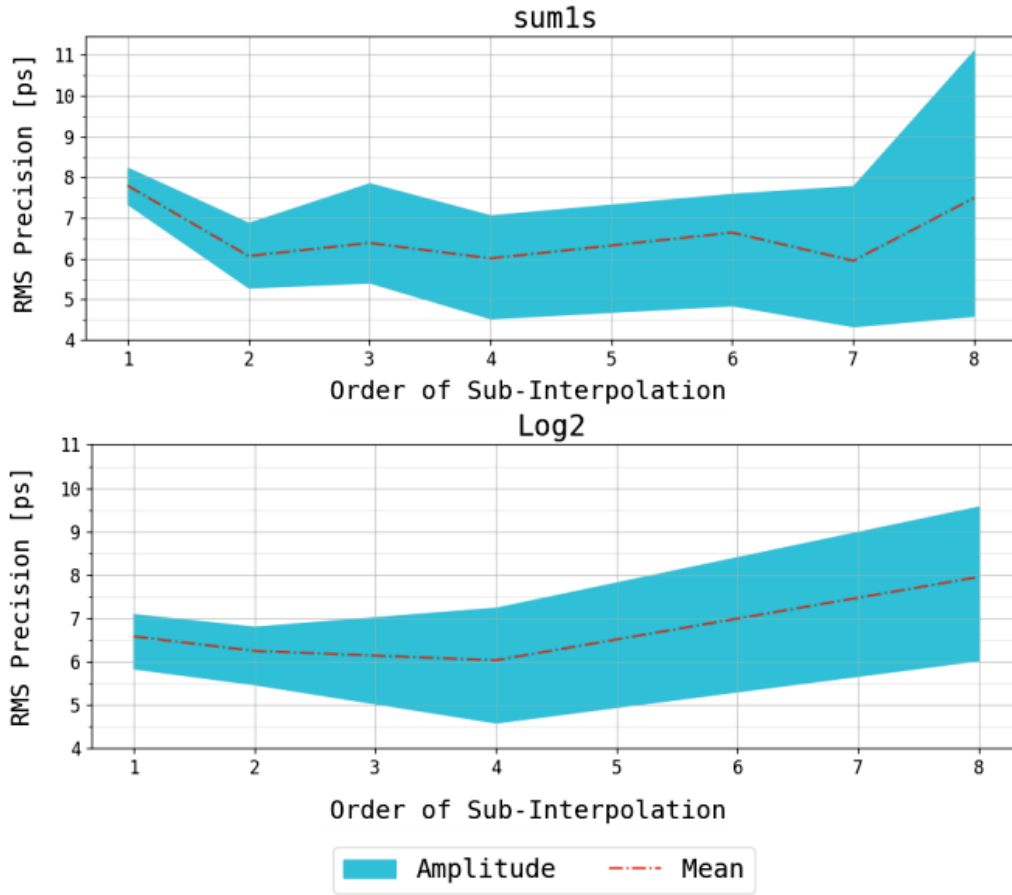
**Figure 20.**  $N_{TDL} = 2$  sub-interpolated TDL precision (**top**) and spread (**bottom**) of the sum1 (red) and the Log2 (blue) implementations, with respect to the measured delay.



**Figure 21.**  $N_{TDL} = 4$  sub-interpolated TDL precision (**top**) and spread (**bottom**) of the sum1 (red) and the Log2 (blue) implementations, with respect to the measured delay.



**Figure 22.**  $N_{TDL} = 8$  sub-interpolated TDL precision (**top**) and spread (**bottom**) of the sum1 (red) and the Log2 (blue) implementations, with respect to the measured delay.



**Figure 23.** Precision of the sum1s (top) and Log2 (bottom) implementations, with respect to the order of sub-interpolation, reporting both the mean (red) and the amplitude (blue), defined as the difference between the maximum and minimum precision measured.

### 3.4. Single-Shot Precision Characterization

Section 3.3 took into account a series of noise contributions not directly related to the TDC itself, as shown in Equation (9), in particular the jitter between the channels of the AWG4014 (i.e.,  $\sigma_{AWG}$ ). To correctly characterize the precision of the proposed TDCs, the jitter of the AWG4014 had to be estimated and removed; to achieve this, the precision was measured in three different scenarios:

- As  $\overline{\sigma_{REAL}}$ , Section 3.2 and Equation (8); using the internal signal, to consider only the internal jitter (i.e.,  $\sigma_j$ ) and the quantization error (i.e.,  $\epsilon_Q$ ).
- As  $\sigma_{SWEEP}$ , Section 3.3 and Equation (9); using 2 distinct channels of the AWG4014, to consider both the jitter of the AWG4014 (i.e.,  $\sigma_{AWG}$ ) and the jitter of the HMC674LP3E (i.e.,  $\sigma_{HMC}$ ).
- As  $\sigma_{split}$ , this paragraph and Equation (10); using only 1 channel of the AWG4014 split into two (i.e., START and STOP), to consider only the jitter of of the HMC674LP3E (i.e.,  $\sigma_{HMC}$ ):

$$\sigma_{split}^2 = 2\sigma_{HMC}^2 + \sigma_j^2 + \epsilon_Q^2 \quad (10)$$

In these terms, and considering Equations (8)–(10) we could ascertain the following:

1. The difference between  $\overline{\sigma_{REAL}}$  and the quantization error (i.e.,  $\epsilon_Q$ ) was the internal jitter (i.e.,  $\sigma_j$ ), i.e.,  $\overline{\sigma_{REAL}}^2 - \epsilon_Q^2 = \sigma_j^2$ ;
2. The difference between  $\sigma_{split}$  and  $\overline{\sigma_{REAL}}$  was the jitter of of the two HMC674LP3E (i.e.,  $\sqrt{2}\sigma_{HMC}$ ), i.e.,  $\sigma_{split}^2 - \overline{\sigma_{REAL}}^2 = 2\sigma_{HMC}^2$ ;



3. The difference between  $\sigma_{SWEEP}$  and  $\sigma_{split}$  was the jitter between the 2 channels of the AWG4014 (i.e.,  $\sigma_{AWG}$ ) that interacted with the HMC674LP3E comparators, i.e.,  $\sigma_{SWEEP}^2 - \sigma_{split}^2 = \sigma_{AWG}^2$ .

These values are reported in Tables 6 and 7.

**Table 6.** Summary of the measured precision for different signal modes: (a) sum1s; (b) Log2.

(a)				
$N_{TDL}$	$\sigma_{SWEEP}$ [ps r.m.s]	$\sigma_{split}$ [ps r.m.s]	$\overline{\sigma_{REAL}}$ [ps r.m.s]	$\varepsilon_Q$ [ps r.m.s]
1	7.79	6.68	3.95	3.66
2	6.07	4.7	2.77	2.12
4	6.01	3.73	1.79	1
8	7.49	4.01	1.40	0.42
(b)				
$N_{TDL}$	$\sigma_{SWEEP}$ [ps r.m.s]	$\sigma_{split}$ [ps r.m.s]	$\overline{\sigma_{REAL}}$ [ps r.m.s]	$\varepsilon_Q$ [ps r.m.s]
1	6.58	5.31	3.94	3.62
2	6.25	4.93	2.74	2.14
4	6.03	3.77	1.98	1.06
8	7.95	5.44	1.76	0.52

**Table 7.** Summary of the measured jitters for different signal modes: (a) sum1s; (b) Log2.

(a)			
$N_{TDL}$	$\sigma_{AWG}$ [ps r.m.s]	$\sqrt{2}\sigma_{HMC}$ [ps r.m.s]	$\sigma_j$ [ps r.m.s]
1	4.01	5.39	1.49
2	3.84	3.80	1.78
4	4.71	3.27	1.48
8	6.33	3.76	1.34
(b)			
$N_{TDL}$	$\sigma_{AWG}$ [ps r.m.s]	$\sqrt{2}\sigma_{HMC}$ [ps r.m.s]	$\sigma_j$ [ps r.m.s]
1	3.89	3.56	1.56
2	3.84	2.96	1.71
4	4.71	3.21	1.67
8	5.80	5.15	1.68

From Table 7, it can be observed that the  $\sigma_{HMC}$  contribution was greater than the nominal value of 0.2 ps r.m.s. This is attributable to the threshold jitter (i.e.,  $Sl/\sigma_{Th}$ ) caused by the signal's slew rate (i.e.,  $Sl$ ), in our case 2 V/0.833 ps (i.e., 2.4 V/ns), and the electronic noise present at the threshold (i.e.,  $\sigma_{Th}$ ). Since the entire TDC (i.e., TDL, Nutt interpolation, decoder, calibrator, measurement histograms) was integrated into a single firmware, the level of threshold noise varied slightly from firmware to firmware. Thus, a noise level of a few millivolts r.m.s., compatible with what was measured experimentally, was calculated. All of this is reported in Table 8.

As for  $\sigma_{AWG}$ , we took into consideration that the jitter contribution AWG-4014 should have been independent of the TDC firmware, and that this contribution also involved jitters due to the interaction of the two HMC674LP3E. However, in practice, we observed small fluctuations of a few picoseconds that varied from firmware to firmware; so, we hypothesized that the fluctuations were principally due to the interaction between the AWG-4014 and the AWG-4014 (more suggestions will be listed in Section 4).



As expected, the additional jitter contribution of the AWG4014 was quite relevant, and the real precision of the TDC was correctly represented by  $\sigma_{split}$ . In this scenario, considering that the START and STOP signals and channels were equal, we defined the Single-Shot Precision (SSP) as

$$\sigma_{SSP}^2 = \frac{\sigma_{split}^2}{2} = \frac{\sigma_j^2 + \epsilon_Q^2}{2} + \sigma_{HMC}^2, \quad (11)$$

which is reported in Table 9.

**Table 8.** Summary of the measured ( $\sigma_{Th}$ ) and computed (i.e.,  $SI/\sigma_{HMC}$ ) threshold jitters for different TDC implementations: (a) sum1s; (b) Log2.

(a)			
$N_{TDL}$	$\sigma_{HMC}$ [ps r.m.s]	$SI/\sigma_{HMC}$ [mV r.m.s]	$\sigma_{Th}$ [mV r.m.s]
1	3.81	6.35	6.23
2	2.68	4.47	4.38
4	2.31	3.86	3.91
8	2.66	4.43	4.22
(b)			
$N_{TDL}$	$\sigma_{HMC}$ [ps r.m.s]	$SI/\sigma_{HMC}$ [mV r.m.s]	$\sigma_{Th}$ [mV r.m.s]
1	2.52	4.20	4.19
2	2.90	4.83	4.25
4	2.27	3.78	3.82
8	3.64	6.07	5.63

**Table 9.** Summary of the SSP for different signal modes: (a) sum1s; (b) Log2.

(a)		
$N_{TDL}$	$\sigma_{split}$ [ps r.m.s]	$\sigma_{SSP}$ [ps r.m.s]
1	6.68	4.72
2	4.7	3.32
4	3.73	2.64
8	4.01	2.84
(b)		
$N_{TDL}$	$\sigma_{split}$ [ps r.m.s]	$\sigma_{SSP}$ [ps r.m.s]
1	5.31	3.75
2	4.93	3.49
4	3.77	2.67
8	5.44	3.85

### 3.5. Linearity

Two linearity tests were performed; the first test focused on the linearity of the timestamps generated by the VTDL (Section 3.5.1), like the tests conducted in [28,53], while the second test concentrated on the linearity of the time intervals, considering the Nutt interpolation (Section 3.5.2), like the tests conducted in [64,65]. Both tests were conducted using CDT [50].

### 3.5.1. Virtual Tapped Delay Line Linearity

The DNL and INL were derived from the CT and CC of each TDC. Specifically, a histogram (i.e., CDT) of the hits per tap for each TDC implementation (i.e.,  $h[i]$  with  $i$  was the tap) was created. Considering that the FSR, thanks to the Nutt interpolation, corresponded to the clock period (i.e.,  $T_{CLK}$ ), the CT was derived, and, thus, the absolute (i.e., value in ps) tap-by-tap DNL (i.e.,  $dnl[i]$ ) was calculated, as reported in Equations (12) and (13):

$$CT[i] = \frac{h[i]}{\sum_i h[i]} \cdot T_{CLK} \quad (12)$$

$$dnl[i] = CT[i] - LSB \quad (13)$$

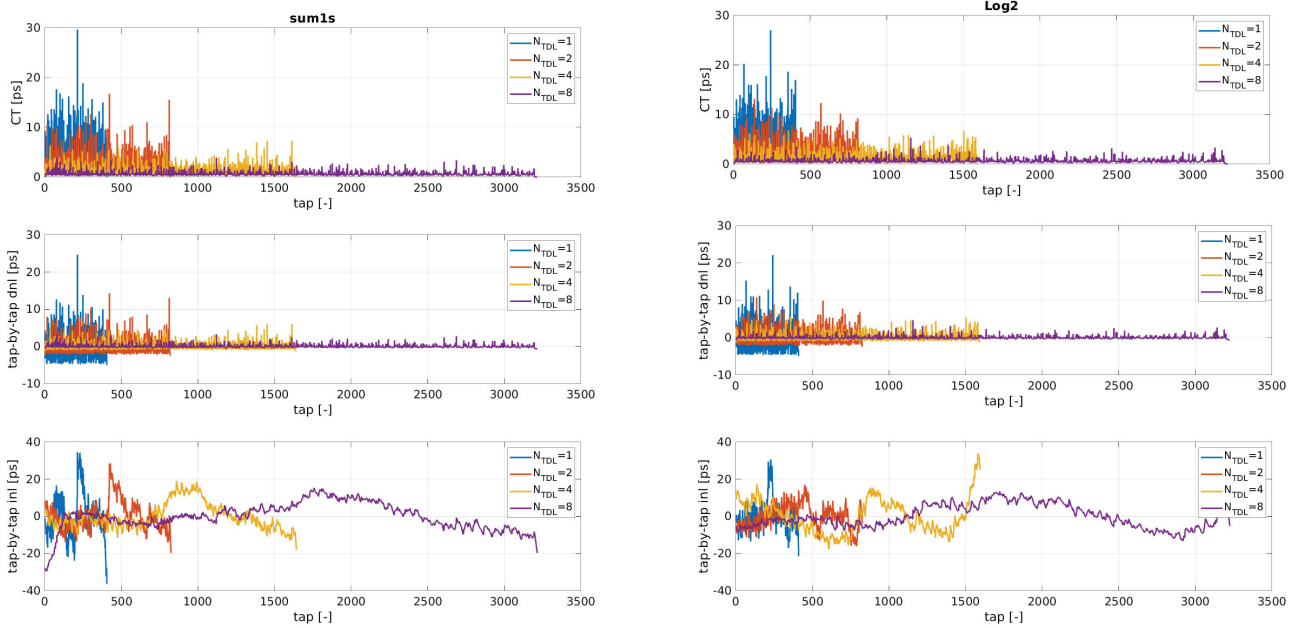
The absolute (i.e., value in ps) tap-by-tap INL (i.e.,  $inl[i]$ ) was derived like the tap-by-tap difference between the CC (i.e.,  $CC[i] = \sum_{k=0}^{k=1} CT[k]$ ) and its linear fitting (i.e.,  $r[i]$ ), as shown in Equation (14):

$$inl[i] = CC[i] - r[i] \quad (14)$$

The DNL and INL figures reported in Table 10, shown here in absolute terms (i.e., in picoseconds), were, therefore, extracted as the maximum amplitude value of the  $dnl[i]$  and  $inl[i]$  curves, i.e.,  $DNL = \max\{|dnl[i]|\}$ , and  $INL = \max\{|inl[i]|\}$ .

The LSB was derived as the average value of the CT, while the ultra-bin was determined as the maximum value. Figure 24 shows the CTs, as well as the tap-by-tap DNLs and INLs for the different TDC implementations. Table 10 summarizes the ultra-bins, LSBs, and the absolute peak-to-peak values of the DNLs and INLs.

Moreover, observing Table 10, we can see that DNL, LSB, and ultra-bin, whether using sum1 or Log2, benefited from sub-interpolation, unlike INL, which showed only slight improvements for  $N_{TDL} = 1$ ,  $N_{TDL} = 2$ ,  $N_{TDL} = 4$ , and a worsening at  $N_{TDL} = 8$ . We can also note that for low orders of sub-interpolation (i.e., none,  $N_{TDL} = 1$ , and  $N_{TDL} = 2$ ) the Log2 implementation was better, compared to sum1. For  $N_{TDL} = 1$ , both in the sum1s case and in the Log2 case, the presence of an ultra-bin at position 241 of 29.672 ps and 27.221 ps, respectively, was observed, as hypothesized in Section 2.3.



**Figure 24.** CT (top), tap-by-tap DNL (center), and tap-by-tap INL (bottom) of the VTDL for sum1s (left) and Log2 (right) TDC implementations at different sub-interpolated orders.

**Table 10.** DNL, INL, LSB, and ultra-bin for sum1s and Log2 TDC implementations at different sub-interpolated orders: (a) sum1s; (b) Log2.

(a)						
$N_{TDL}$	DNL [ps]		INL [ps]		LSB [ps]	Ultra-Bin [ps]
1	[−5.010; 24.657]	24.657	[−36.347; 34.557]	36.347	4.938	29.672
2	[−2.465; 14.306]	14.306	[−19.631; 28.481]	28.481	2.430	16.771
4	[−1.231; 6.033]	6.033	[−17.926; 19.060]	19.060	1.218	7.264
8	[−0.620; 3.272]	3.272	[−29.463; 15.351]	29.463	0.622	3.892

(b)						
$N_{TDL}$	DNL [ps]		INL [ps]		LSB [ps]	Ultra-Bin [ps]
1	[−4.911; 22.343]	22.343	[−21.434; 30.606]	30.606	4.878	27.221
2	[−2.420; 10.748]	10.748	[−16.075; 17.018]	17.018	2.424	13.169
4	[−1.246; 5.580]	5.580	[−17.557; 33.934]	33.934	1.255	6.826
8	[−0.620; 4.6883]	4.689	[−13.153; 13.406]	13.406	0.628	6.645

### 3.5.2. Nutt-Interpolated Linearity

For this section, the complete linearity test was considered, also taking into account the Nutt interpolation performed in a range between 0 and 100 ns (i.e.,  $\Delta T = 100$  ns). The CDT was performed by providing 2 channels of the various TDC implementations with uncorrelated events, thus collecting the CDT with a binning of 15.6 ps (i.e.,  $BIN = 15.6$  ps); the histogram was entirely acquired in the FPGA [63].

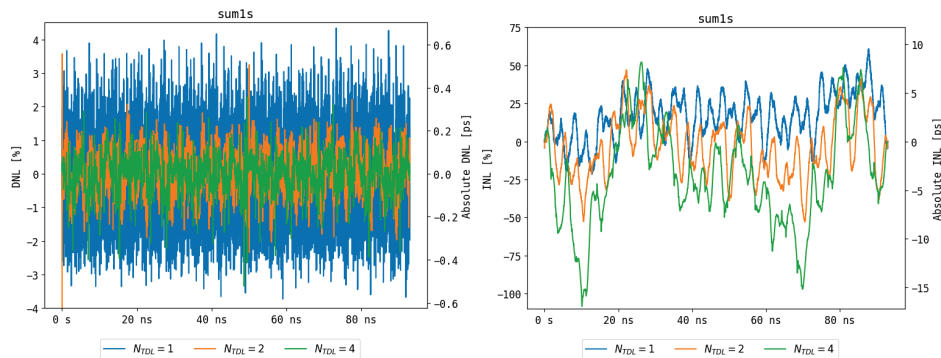
In this context, given the histogram  $H[b]$  where  $b$  denoted the bin, and in a manner completely analogous to Equations (12)–(14), considering that the measurement range was no longer  $T_{CLK}$  but rather  $\Delta T$ , the absolute (value in ps) bin-by-bin DNL and INL were calculated as reported in Equations (15) and (16):

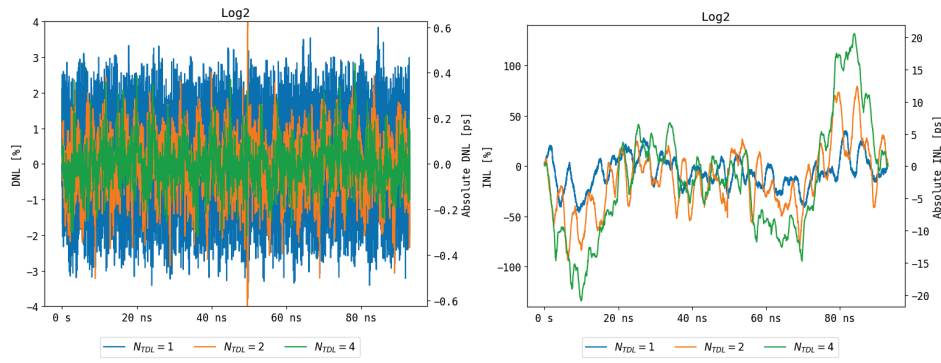
$$dnl[b] = \frac{H[b]}{\sum_b H[b]} \cdot \Delta T - BIN \quad (15)$$

$$inl[b] = \sum_{k=0}^{k=b} dnl[k] \quad (16)$$

The DNL and INL shown in Table 11, shown here in absolute terms (i.e., in picoseconds), were, therefore, extracted as the maximum amplitude value of the  $dnl[b]$  and  $inl[b]$  curves, i.e.,  $DNL = \max\{|dnl[b]|\}$  and  $INL = \max\{|inl[b]|\}$ .

Figure 25, for the sum1s decoder, and Figure 26, for the Log2 implementations, show the bin-by-bin DNL and bin-by-bin INL, considering only the sub-interpolation orders (i.e.,  $N_{TDL}$ ) of 1 (i.e., none), 2, and 4, which were the most interesting ones, as discussed in the previous paragraph (i.e., lower jitter). The INL and DNL values referred to the 15.6 ps bin are also provided in Table 11. As we can see, the order of sub-interpolation did not yield a significant effect on the DNL while the INL increased with it.

**Figure 25.** DNL (left) and INL (right) in the range 0–100 ns binned at 15.6 ps for the sum1s TDC implementations at different sub-interpolation orders.



**Figure 26.** DNL (left) and INL (right) in the range 0–100 ns binned at 15.6 ps for the Log2 TDC implementations at different sub-interpolation orders.

**Table 11.** Linearity test (i.e., DNL and INL) in the range 0–100 ns result for the TDC implementations: (a) sum1s; (b) Log2.

(a)				
$N_{TDL}$	DNL [ps]		INL [ps]	
1	[−0.583; 0.678]	0.678	[−3.373; 9.507]	9.507
2	[−0.633; 0.558]	0.663	[−8.255; 7.333]	8.255
4	[−0.523; 0.357]	0.523	[−16.939; 8.157]	16.939
(b)				
$N_{TDL}$	DNL [ps]		INL [ps]	
1	[−0.534; 0.599]	0.599	[−7.151; 5.494]	7.151
2	[−0.601; 0.605]	0.605	[−14.69; 12.381]	14.69
4	[−0.32; 0.441]	0.441	[−20.925; 20.611]	20.925

### 3.6. Final Results

From Tables 3–6, it is evident that the sub-interpolated TDL TDCs using the sum1s decoder achieved better precision compared to those using Log2; hence, we deduce that the sum1s decoder is preferred for sub-interpolated TDL TDCs, while Log2 is more suitable for non-sub-interpolated ones. As a conclusion, the optimal order of sub-interpolation was identified as 4 for the optimization of the precision of the TDC, in conjunction with the sum1s decoder reaching an SSP of 2.64 ps rms, a quantization error of 1 ps r.m.s., an LSB of 1.218 ps, a peak DNL over the V-TDL of 6.033 ps, and a peak DNL on the Nutt-interpolated TDC of 0.523 ps. In reference to precision vs. sub-interpolation order, observing Figure 23, a “bathtub curve”—similar to that obtained in [36] (in detail,  $N_{TDL}$  TDLs were implemented in parallel, each executing a Wave Union A (WUA) algorithm for a total of  $2 \times N_{TDL}$  orders of sub-interpolation) by the same research group—was observed on a 28 nm Xilinx 7-Series Artix-7 200T. Obviously, since both the technology node and the sub-interpolation algorithm were different, the optimum in [36] was  $N_{TDL} = 6$ , different from the 4 obtained in this paper.

However, such a “bathtub curve” was not observed in [33] (28 nm Xilinx 7-Series Kintex-7 160T) and [46] (40 nm Xilinx 6-Series Virtex-6 200T), where a trend of continuous improvement was observed with increasing  $N_{TDL}$ . This was hypothesized to be due to the absence of processing blocks in [33,46] (i.e., decoder and calibrator), as the thermometer codes of the TDLs were acquired directly and subsequently processed offline on a PC. In fact, in this paper (and also in [36], although not specified in the text), all processing (i.e., decoding, calibration, and measurement) was performed on the FPGA, which significantly worsened the signal integrity of the FPGA (i.e., higher  $\sigma_j$ ) as  $N_{TDL}$  increased.

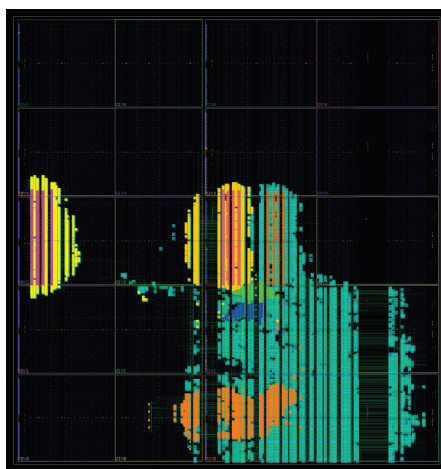
Another interesting implementation was the single TDL with the Log2 decoder providing great linearity and acceptable precision, reaching an SSP of 3.75 ps r.m.s., a quantization

error of 3.62 ps r.m.s., an LSB of 4.878 ps, a peak DNL over the TDL of 15.308 ps, and a peak DNL on the Nutt-interpolated TDC of 0.599 ps. On the other hand, a single TDL with the sum1s provided an SSP of 4.72 ps r.m.s., a quantization error of 3.66 ps r.m.s., an LSB of 4.938 ps, a peak DNL over the TDL of 24.657 ps, and a peak DNL on the Nutt-interpolated TDC of 0.678 ps. Comparing the quantization errors, LSBs, and DNLs of these two implementations, we deduced that ignoring BEs, even if they were present in the thermometer code, was more effective in the 20 nm Xilinx UltraScale FPGAs if a single TDL (without sub-interpolation) was implemented. Therefore, even though the TDL was affected by BEs, in our opinion the main informational content resided in the Most Signification Bit (MSB) of the thermometric code, and the bubbles did not help to increase the precision. In contrast, increasing the order of sub-interpolation, the bubbles helped to improve the quantization errors, LSBs, and DNLs, probably because they somehow helped to decorrelate the measurements coming from the different TDLs, making the sub-interpolation more effective.

The single TDL with the Log2 can be also exploited for implementations where a high number of channels is needed (64 in the proposed XCKU040-2FFVA1156E FPGA), since it utilizes less than 1% of each type of the resources in the FPGA per channel, while the four sub-interpolated TDLs with sum1s can be used for up to 24 channels in the proposed XCKU040-2FFVA1156E FPGA. Table 12 summarizes the area occupation, in terms of the raw number of blocks, considering the TDL, the decoder, and the calibrator. For comparison, the best-performing implementation occupied as much as 4x the area, which greatly limited the maximum number of channels. Table 13 shows the percentage area occupancy of the entire firmware and of a single TDC channel, demonstrating that the TDC is a minimal part of the overall firmware in considering a real measurement setup (e.g., with processing end transmission). To provide a pictorial overview, Figure 27 shows the floorplan of the FPGA for the sum1s implementation with 2 channels and four TDLs, highlighting the TDLs (violet for channel 1 and pink for channel 2), the decoder (yellow for channel 1 and ocher for channel 2), the decoder (green for channel 1 and blue for channel 2), the histogram engine (orange), and the auxiliary logic (blue).

**Table 12.** Summary of area occupation.

$N_{TDL}$	Decoder	LUTs	DFFs	CARRY8	CLB	Block RAM
1	Log2	955	1529	103	259	72 kb (2 blocks)
4	sum1s	3607	6564	337	883	126 kb (3.5 blocks)



**Figure 27.** Floorplan for the sum1s implementation with 2 channels and four TDLs, highlighting the TDLs (violet for channel 1 and pink for channel 2), the decoder (yellow for channel 1 and ocher for channel 2), the decoder (green for channel 1 and blue for channel 2), the histogram engine (orange), and the auxiliary logic (blue).

**Table 13.** Percentage area occupancy of the entire firmware (a) and a single channel (b) of the TDC in the various implementations.

(a)					
$N_{TDL}$	Type	LUTs	DFFs	BRAM	DSP
1	sum1s	8.88	4.05	17.2	0.75
	Log2	8.63	3.79	17.2	0.75
2	sum1s	9.89	4.69	17.2	0.75
	Log2	9.38	4.16	17.2	0.75
4	sum1s	11.9	10.35	17.2	0.75
	Log2	10.88	9.29	17.2	0.75
8	sum1s	15.93	10.85	17.2	0.75
	Log2	13.93	10.74	17.2	0.75
(b)					
$N_{TDL}$	Type	LUTs	DFFs	BRAM	
1	sum1s	0.48	0.72	0.33	
	Log2	0.39	0.32	0.33	
2	sum1s	0.82	0.93	0.50	
	Log2	0.65	0.32	0.50	
4	sum1s	1.49	1.35	0.58	
	Log2	1.15	0.56	0.58	
8	sum1s	2.83	1.35	0.67	
	Log2	2.16	1.48	0.67	

#### 4. State of the Art and Future Development

##### 4.1. State of the Art

It is important to review the current state of the art on the subject matter, to understand where our contribution stands. In particular, in the TDL TDC space, research has moved towards the implementation of two main techniques to improve measurement precision. The first technique is Dual Sampling (DS) [7,14,38], which samples both the C and O outputs of the TDL, effectively halving the propagation delay of the taps, at the cost of linearity. The second technique is Sub-TDL [12,14], where taps of the same TDL are grouped into smaller chains and then sub-interpolated, to reduce the impact of BEs. While this last approach is valid for reducing BEs by elongating the tap delay, our focus was on finding a general decoding policy to address these errors globally. By implementing placement optimizations, we achieved similar results. An interesting future development, since nothing similar has been found in the literature, would be to verify the tuned delay line technique [27] and compare it accordingly. A comparison between this work and other relevant UltraScale implementations is provided in Table 14. Unlike all the works reported in Table 14, in this paper not only the TDC but also the entire histogramming and measurement mechanism were implemented directly in the FPGA, thereby contributing to the increases in disturbance and noise that were converted in an increase of jitter (from Figure 1 of [38], Figure 6 and Table I of [7], Figures 1 and 2 of [66], Figure 12 of [12], Figure 5 of [13], and Figure 14 of [14], it is possible to see that only the TDC was implemented in the FPGA).



**Table 14.** Summary of the state of the art.

Ref.	Mode	Precision [ps r.m.s.]
[38]-2016	DS	3.9
[7]-2016	DS, Bin Decimation	4.2
[66]-2017	Bin Decimation	4.7–5.6
[12]-2019	Sub-TDL	5.37
[13]-2021	LSPM	5.55
[14]-2022	WUA, Sub-TDL, DS	3.63
This work	One-TDL Log2	5.31
	Four-TDL sum1s	3.73

#### 4.2. Potential Applications Enabled

As mentioned in the Introduction, time measurements are crucial in quantum technologies, life sciences, and nuclear physics. State-of-the-art experiments demand very high precision (i.e., a few picoseconds) and a significant number of channels in a single chip (i.e., tens), which has led, until now, to the use of Time-to-Amplitude Converters (TACs), analog time converters on ASICs inherently more precise (i.e., sub-picosecond) than a TDC [67], and ASIC-based TDCs, such as CERN’s PicoTDC [68,69], PETsys’s TOFPET2 [70,71], and Weeroc’s Temporoc [72].

However, the use of ASICs for time measurement imposes significant limitations, in terms of flexibility and time-to-market, characteristics that are increasingly required in both academic and industrial research. As a result, various sectors where FPGA-based TDC technology is already mature, such as Cross Delay Line (CDL) detection systems [73], are transitioning toward FPGA-based TDCs, moving away from ASIC-based solutions like ACAM’s TDCs [74].

In emerging research fields, such as high-energy physics [71], TOF-PET [75] and particle therapy [76,77], and quantum technologies [78,79], FPGA-based TDCs are progressively reaching ASIC performance levels, in terms of both precision and channel count, thanks to continuous optimization efforts. Therefore, optimization processes and analyses, such as those discussed in this paper, are essential to opening the doors for FPGA-based TDCs in these advanced fields. In this context, the single TDL with Log2 decoder achieved an SSP of 3.75 ps r.m.s. with an area occupation per channel of only 259 CLBs and 72 kb of BRAM, enabling the implementation up to 64 channels on the proposed FPGA (i.e., XCKU040-2FFVA1156E). This system is, thus, comparable in terms of both precision and channel count to the PicoTDC, and superior in precision to the TOFPET2 and Temporoc ASICs. On the other hand, the four-TDL sub-interpolated TDC with sum1s decoder, although limited to a maximum of only 24 channels on the proposed FPGA (i.e., 883 CLBs and 126 kb per channel), achieved better precision (i.e., SSP of 2.64 ps r.m.s.) than the PicoTDC. A summary is reported in Table 15.

**Table 15.** Comparative table between proposed FPGA-based TDCs and state-of-the-art ASIC-based TDCs used in advanced applications, such as high-energy physics, medical diagnostics, and quantum technologies.

Ref.	Name	LSB <sup>1</sup>	SSP <sup>2</sup>	Channels
[68,69]	PicoTDC	3 ps	3.7 ps r.m.s.	64
[70]	TOFPET2	30 ps	-	64 in 36 × 25 mm chip
[72]	Temporoc	50 ps	20 ps r.m.s.	64 in 20 × 20 mm chip
This work	One-TDL Log2	<8.9 ps	3.75 ps r.m.s.	<64 in 35 × 35 mm chip
	Four-TDL sum1s	<2.47 ps	2.64 ps r.m.s.	<24 in 35 × 35 mm chip

<sup>1</sup>  $LSB_{EQ}$  was considered for “This work”. <sup>2</sup> Also, the jitter due to the comparator was considered in “This work”.

#### 4.3. Open Issues and Pending Experiments

In the following sub-paragraphs, all open issues and pending experiments identified in this manuscript are listed, which are, therefore, left for future developments.



#### 4.3.1. AWG4014 Jitter Fluctuation

In Section 3.4 and in Table 7, it is possible to observe fluctuations of  $\sigma_{AWG}$  that represent the jitter between the channels of the AWG-4014 when connected to the two HMC674 comparators in order to perform Time Sweep tests (Section 3.3). Obviously, the state of the TDC should not have influenced in any way the jitter that existed only between the 2 channels of the AWG-4014, but the TDC firmware could modify the state of the KCU105 EVB (e.g., noise on the power lines, ground lines, and common mode fluctuations). Therefore, we hypothesize that the jitters between the AWG-4014, the two HMC674 comparators, and the KCU105 varied slightly from firmware to firmware. An estimate of the jitter existing between the 2 AWG-4014 channels using, for instance, an oscilloscope was not performed, as it would not only be difficult to distinguish the jitter component of the AWG-4014 alone from the measurement precision and jitter of the oscilloscope, but also because oscilloscopes have an analog front end that conditions the signals, thereby practically altering (e.g., amplifying and filtering) the input signals and possibly also the jitter.

Consequently, as found in the scientific literature, the figure of merit for precision used was the SSP, evaluated as indicated in Equation (11).

#### 4.3.2. Most Significant Information in the MSB of the Thermometric Code

Another interesting open issue would be to mathematically demonstrate, not just experimentally, that the Log2 decoding is more effective in terms of precision than the sum1s decoding in the case of non-sub-interpolated TDLs, whereas the bubble compression offered by the sum1s algorithm helps to improve sub-interpolation. This would validate or refute the hypothesis proposed in Section 3.6, where we deduced that the most significant information is contained in the MSB of the thermometer code and that the BEs only help improve sub-interpolation, likely because they somehow help to decorrelate measurements from different TDLs, thereby making sub-interpolation more effective.

#### 4.3.3. Absence of the “Bathtub Curve”

From a metrological point of view, it would also be interesting to conduct a mathematical analysis to demonstrate the absence of the “bathtub curve” shape in the relationship between precision and sub-interpolation, depending on the size of the firmware, which induces an increase in noise and disturbances resulting in greater jitter (Section 3.3, Figure 23).

#### 4.3.4. Clock Skew Pattern Behavior

Another interesting point would to investigate—and, thus, formulate a more accurate hypothesis—the phenomenon reported in Section 2.2, Figure 10. Specifically, to reassess the clock skew pattern in the presence and absence of the TDC firmware observed experimentally in post-implementation on the XCKU040-2FFVA1156E FPGA and not observed in the scientific literature Xilinx 28 nm 7-Series FPGA (e.g., [33]).

### 5. Conclusions

In conclusion, within the context of 20 nm FPGA technology, we have established design rules to mitigate BEs, which are the most significant error sources in TDL TDC implementations in FPGAs. This is particularly relevant when spatial sub-interpolation is used to maximize resolution. We identified that the optimal number of parallel TDLs, or sub-interpolation order, is four. Additionally, the optimal decoding policy was determined both in the presence and absence of sub-interpolation. When sub-interpolation was employed, the sum1s decoder achieved an SSP of 2.64 ps r.m.s., with a DNL of 0.523 ps and an INL of 16.939 ps. This configuration occupied 883 CLBs and 126 kb of BRAM, and it could fit up to a 24-channel implementation in the proposed FPGA (i.e., XCKU040-2FFVA1156E). In contrast, for a single TDL, the Log2 decoder achieved a single-shot precision of 3.75 ps r.m.s., with a DNL of 0.599 ps and an INL of 7.151 ps, while occupying only 259 CLBs and 72 kb of BRAM, and it could fit up to a 64-channel implementation in the proposed FPGA (i.e., XCKU040-2FFVA1156E). These two distinct implementations provide different benefits:

the former offers very high precision, while the latter provides adequate precision in 1/4 of the area.

Moreover, this study helps to further narrow the gap between FPGA-based TDCs and ASIC solutions, which, being better-performing, are often used in cutting-edge applications (e.g., quantum technologies, life sciences, and nuclear physics). However, ASIC-based TDCs are less suitable for R&D and fast prototyping, as they are much less flexible, more expensive, and have long time-to-market, with respect to FPGA-based solutions.

**Author Contributions:** Methodology, N.L.; Software, G.F.; Validation, F.G.; Formal analysis, E.R.; Investigation, M.M.; Data curation, G.B.; Writing—original draft, A.G.; Visualization, A.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding

**Data Availability Statement:** Data are contained within the articles.

**Acknowledgments:** A special thanks to TEDIEL S.r.l., a spin-off of Politecnico di Milano.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Yuan, F. *CMOS Time-Mode Circuits and Systems: Fundamentals and Applications*; CRC Press: Boca Raton, FL, USA, 2020.
2. Agüero, M.; Hnilo, A.; Kovalsky, M.; Nonaka, M. Testing a hypothetical transient deviation from quantum mechanics: Preliminary results. *J. Opt. Soc. Am. B* **2023**, *40*, C28–C34. [CrossRef]
3. Lusardi, N.; Garzetti, F.; Bulgarini, G.; Gourgues, R.; Los, J.; Geraci, A. Single photon counting through multi-channel TDC in programmable logic. In Proceedings of the 2016 IEEE Nuclear Science Symposium, Medical Imaging Conference and Room-Temperature Semiconductor Detector Workshop (NSS/MIC/RTSD), Strasbourg, France, 29 October–6 November 2016; pp. 1–4. [CrossRef]
4. Garzetti, F.; Salgado, S.; Venialgo, E.; Lusardi, N.; Corna, N.; Geraci, A.; Charbon, E. Plug-and-play TOF-PET Module Readout Based on TDC-on-FPGA and Gigabit Optical Fiber Network. In Proceedings of the 2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Manchester, UK, 26 October–2 November 2019; pp. 1–4. [CrossRef]
5. Parsakordasibi, M.; Vornicu, I.; Rodríguez-Vázquez, A.; Carmona-Galán, R. A Low-Resources TDC for Multi-Channel Direct ToF Readout Based on a 28-nm FPGA. *Sensors* **2021**, *21*, 308. [CrossRef] [PubMed]
6. Lusardi, N.; Garzetti, F.; Corna, N.; Reale, A.; Geraci, A.; Dobovcnik, E.; Cautero, G.; Dri, C.; Sergio, R.; Stebel, L. Advanced System in FPGA for 3D (X, Y, t) Imaging with Cross Delay-Lines. In Proceedings of the 2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Manchester, UK, 26 October–2 November 2019; pp. 1–4. [CrossRef]
7. Wang, Y.; Liu, C. A 4.2 ps Time-Interval RMS Resolution Time-to-Digital Converter Using a Bin Decimation Method in an UltraScale FPGA. *IEEE Trans. Nucl. Sci.* **2016**, *63*, 2632–2638. [CrossRef]
8. Garzetti, F.; Lusardi, N.; Geraci, A.; Dobovcnik, E.; Cautero, G.; Dri, C.; Sergio, R.; Stebel, L. Fully FPGA-based and all-reconfigurable TDC for 3D (X, Y, t) Cross Delay-Line detectors. In Proceedings of the 2018 IEEE Nuclear Science Symposium and Medical Imaging Conference Proceedings (NSS/MIC), Sydney, NSW, Australia, 10–17 November 2018; pp. 1–3. [CrossRef]
9. Machado, R.; Cabral, J.; Alves, F.S. Recent Developments and Challenges in FPGA-Based Time-to-Digital Converters. *IEEE Trans. Instrum. Meas.* **2019**, *68*, 4205–4221. [CrossRef]
10. Szyduczyński, J.; Kościelnik, D.; Miśkiewicz, M. Time-to-digital conversion techniques: A survey of recent developments. *Measurement* **2023**, *214*, 112762. [CrossRef]
11. Wang, Y.; Xie, W.; Chen, H.; Pei, C.; Li, D. A two-stage interpolation time-to-digital converter implemented in 20 nm and 28 nm FPGAs. *IEEE Trans. Ind. Electron.* **2024**, *71*, 15200–15210. [CrossRef]
12. Chen, H.; Li, D.D.U. Multichannel, Low Nonlinearity Time-to-Digital Converters Based on 20 and 28 nm FPGAs. *IEEE Trans. Ind. Electron.* **2019**, *66*, 3265–3274. [CrossRef]
13. Zhang, M.; Yang, K.; Chai, Z.; Wang, H.; Ding, Z.; Bao, W. High-Resolution Time-to-Digital Converters Implemented on 40-, 28-, and 20-nm FPGAs. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 2002310. [CrossRef]
14. Xie, W.; Chen, H.; Li, D.D.U. Efficient Time-to-Digital Converters in 20 nm FPGAs With Wave Union Methods. *IEEE Trans. Ind. Electron.* **2022**, *69*, 1021–1031. [CrossRef]
15. Xie, W.; Wang, Y.; Chen, H.; Li, D.D.U. 128-Channel High-Linearity Resolution-Adjustable Time-to-Digital Converters for LiDAR Applications: Software Predictions and Hardware Implementations. *IEEE Trans. Ind. Electron.* **2022**, *69*, 4264–4274. [CrossRef]
16. Kim, J.; Jung, J.H.; Choi, Y.; Jung, J.; Lee, S. Linearity improvement of UltraScale+ FPGA-based time-to-digital converter. *Nucl. Eng. Technol.* **2023**, *55*, 484–492. [CrossRef]
17. Dadouche, F.; Turko, T.; Uhring, W.; Malass, I.; Dumas, N.; Normand, J.P.L. New Design-methodology of High-performance TDC on a Low Cost FPGA Targets. *Sens. Transducers* **2015**, *193*, 123.

18. Wang, Z.; Lu, J.; Nunez-Yanez, J. A Low-complexity FPGA TDC based on a DSP Delay Line and a Wave Union Launcher. In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 31 August–2 September 2022; pp. 101–108. [CrossRef]
19. Kwiatkowski, P. Employing FPGA DSP blocks for time-to-digital conversion. *Metrol. Meas. Syst.* **2019**, *26*, 631–643. [CrossRef]
20. Chaberski, D.; Frankowski, R.; Zieliński, M.; Zaworski, Ł. Multiple-tapped-delay-line hardware-linearisation technique based on wire load regulation. *Measurement* **2016**, *92*, 103–113. [CrossRef]
21. Wang, Y.; Kuang, J.; Liu, C.; Cao, Q. A 3.9-ps RMS Precision Time-to-Digital Converter Using Ones-Counter Encoding Scheme in a Kintex-7 FPGA. *IEEE Trans. Nucl. Sci.* **2017**, *64*, 2713–2718. [CrossRef]
22. Carra, P.; Bertazzoni, M.; Bisogni, M.G.; Cela Ruiz, J.M.; Del Guerra, A.; Gascon, D.; Gomez, S.; Morrocchi, M.; Pazzi, G.; Sanchez, D.; et al. Auto-Calibrating TDC for an SoC-FPGA Data Acquisition System. *IEEE Trans. Radiat. Plasma Med. Sci.* **2019**, *3*, 549–556. [CrossRef]
23. Garzetti, F.; Corna, N.; Lusardi, N.; Geraci, A. Time-to-Digital Converter IP-Core for FPGA at State of the Art. *IEEE Access* **2021**, *9*, 85515–85528. [CrossRef]
24. Wang, Y.; Liu, C. A Nonlinearity Minimization-Oriented Resource-Saving Time-to-Digital Converter Implemented in a 28 nm Xilinx FPGA. *IEEE Trans. Nucl. Sci.* **2015**, *62*, 2003–2009. [CrossRef]
25. Randall, R.; Tordon, M. Data Acquisition. In *Encyclopedia of Vibration*; Braun, S., Ed.; Elsevier: Oxford, UK, 2001; pp. 364–376. [CrossRef]
26. Szplet, R.; Jachna, Z.; Kwiatkowski, P.; Rozyc, K. A 2.9 ps equivalent resolution interpolating time counter based on multiple independent coding lines. *Meas. Sci. Technol.* **2013**, *24*, 035904. [CrossRef]
27. Won, J.Y.; Lee, J.S. Time-to-Digital Converter Using a Tuned-Delay Line Evaluated in 28-, 40-, and 45-nm FPGAs. *IEEE Trans. Instrum. Meas.* **2016**, *65*, 1678–1689. [CrossRef]
28. Won, J.Y.; Kwon, S.I.; Yoon, H.S.; Ko, G.B.; Son, J.W.; Lee, J.S. Dual-Phase Tapped-Delay-Line Time-to-Digital Converter With On-the-Fly Calibration Implemented in 40 nm FPGA. *IEEE Trans. Biomed. Circuits Syst.* **2016**, *10*, 231–242. [CrossRef] [PubMed]
29. Zhang, S.; Wang, S.; Lin, X.; Ren, G. A 6-bit low power flash ADC with a novel bubble error correction used in UWB communication systems. In Proceedings of the 2014 IEEE International Conference on Electron Devices and Solid-State Circuits, Chengdu, China, 18–20 June 2014; pp. 1–2. [CrossRef]
30. Ghoshal, P.; Sen, S.K. A bit swap logic (BSL) based bubble error correction (BEC) method for flash ADCs. In Proceedings of the 2016 2nd International Conference on Control, Instrumentation, Energy & Communication (CIEC), Kolkata, India, 28–30 January 2016; pp. 111–115. [CrossRef]
31. Jaworski, Z. Verilog HDL model based thermometer-to-binary encoder with bubble error correction. In Proceedings of the 2016 MIXDES—23rd International Conference Mixed Design of Integrated Circuits and Systems, Lodz, Poland, 23–25 June 2016; pp. 249–254. [CrossRef]
32. Kwiatkowski, P.; Sondej, D.; Szplet, R. Bubble-Proof Algorithm for Wave Union TDCs. *Electronics* **2022**, *11*, 30. [CrossRef]
33. Kwiatkowski, P.; Szplet, R. Efficient Implementation of Multiple Time Coding Lines-Based TDC in an FPGA Device. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 7353–7364. [CrossRef]
34. Lusardi, N.; Garzetti, F.; Corna, N.; Marco, R.D.; Geraci, A. Very High-Performance 24-Channels Time-to-Digital Converter in Xilinx 20-nm Kintex UltraScale FPGA. In Proceedings of the 2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Manchester, UK, 26 October–2 November 2019; pp. 1–4. [CrossRef]
35. Chaberski, D.; Frankowski, R.; Gurski, M.; Zielinski, M. Comparison of Interpolators Used for Time-Interval Measurement Systems Based on Multiple-Tapped Delay Line. *Metrol. Meas. Syst.* **2017**, *24*, 401–412. [CrossRef]
36. Lusardi, N.; Garzetti, F.; Geraci, A. The role of sub-interpolation for Delay-Line Time-to-Digital Converters in FPGA devices. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2019**, *916*, 204–214. [CrossRef]
37. Kwiatkowski, P.; Szplet, R. Time-to-Digital Converter with Pseudo-Segmented Delay Line. In Proceedings of the 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Auckland, New Zealand, 20–23 May 2019; pp. 1–6. [CrossRef]
38. Liu, C.; Wang, Y.; Kuang, P.; Li, D.; Cheng, X. A 3.9 ps RMS resolution time-to-digital converter using dual-sampling method on Kintex UltraScale FPGA. In Proceedings of the 2016 IEEE-NPSS Real Time Conference (RT), Padua, Italy, 6–10 June 2016; pp. 1–3. [CrossRef]
39. Wang, Y.; Zhou, X.; Song, Z.; Kuang, J.; Cao, Q. A 3.0-ps rms Precision 277-MSamples/s Throughput Time-to-Digital Converter Using Multi-Edge Encoding Scheme in a Kintex-7 FPGA. *IEEE Trans. Nucl. Sci.* **2019**, *66*, 2275–2281. [CrossRef]
40. Wu, J.; Shi, Z. The 10-ps wave union TDC: Improving FPGA TDC resolution beyond its cell delay. In Proceedings of the 2008 IEEE Nuclear Science Symposium Conference Record, Dresden, Germany, 19–25 October 2008; pp. 3440–3446. [CrossRef]
41. Qi, J.; Gong, H.; Liu, Y. On-Chip Real-Time Correction for a 20-ps Wave Union Time-To-Digital Converter (TDC) in a Field-Programmable Gate Array (FPGA). *IEEE Trans. Nucl. Sci.* **2012**, *59*, 1605–1610. [CrossRef]
42. Szplet, R.; Sondej, D.; Grzęda, G. Interpolating time counter with multi-edge coding. In Proceedings of the 2013 Joint European Frequency and Time Forum and International Frequency Control Symposium (EFTF/IFC), Prague, Czech Republic, 21–25 July 2013; pp. 321–324. [CrossRef]

43. Bayer, E.; Zipf, P.; Traxler, M. A multichannel high-resolution (<5 ps RMS between two channels) Time-to-Digital Converter (TDC) implemented in a field programmable gate array (FPGA). In Proceedings of the 2011 IEEE Nuclear Science Symposium Conference Record, Valencia, Spain, 23–29 October 2011; pp. 876–879. [CrossRef]
44. Daigneault, M.A.; David, J.P. A High-Resolution Time-to-Digital Converter on FPGA Using Dynamic Reconfiguration. *IEEE Trans. Instrum. Meas.* **2011**, *60*, 2070–2079. [CrossRef]
45. Daigneault, M.A.; David, J.P. A novel 10 ps resolution TDC architecture implemented in a 130nm process FPGA. In Proceedings of the 8th IEEE International NEWCAS Conference 2010, Montreal, QC, Canada, 20–23 June 2010; pp. 281–284. [CrossRef]
46. Shen, Q.; Liu, S.; Qi, B.; An, Q.; Liao, S.; Shang, P.; Peng, C.; Liu, W. A 1.7 ps Equivalent Bin Size and 4.2 ps RMS FPGA TDC Based on Multichain Measurements Averaging Method. *IEEE Trans. Nucl. Sci.* **2015**, *62*, 947–954. [CrossRef]
47. Wang, Y.; Cao, Q.; Liu, C. A Multi-Chain Merged Tapped Delay Line for High Precision Time-to-Digital Converters in FPGAs. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 96–100. [CrossRef]
48. Kwiatkowski, P.; Szplet, R. Multisampling wave union time-to-digital converter. In Proceedings of the 2020 6th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP), Krakow, Poland, 23–25 September 2020; pp. 1–5. [CrossRef]
49. Frankowski, R.; Gurski, M.; Szplet, R. Kintex UltraScale’s multi-segment digital tapped delay lines with controlled characteristics for precise time-to-digital conversion. *Metrol. Meas. Syst.* **2024**, *31*. [CrossRef]
50. Lusardi, N.; Corna, N.; Garzetti, F.; Salgaro, S.; Geraci, A. Cross-Talk Issues in Time Measurements. *IEEE Access* **2021**, *9*, 129303–129318. [CrossRef]
51. Kuang, J.; Wang, Y.; Cao, Q.; Liu, C. Implementation of a high precision multi-measurement time-to-digital convertor on a Kintex-7 FPGA. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2018**, *891*, 37–41. [CrossRef]
52. Nutt, R. Digital Time Intervalometer. *Rev. Sci. Instrum.* **1968**, *39*, 1342–1345. [CrossRef]
53. Zhu, M.; Qi, X.; Cui, T.; Gao, Q. Tapped delay line for compact time-to-digital converter on UltraScale FPGA and its coding method. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2023**, *1056*, 168639. [CrossRef]
54. UG574, UltraScale Architecture Configurable Logic Block. Available online: <https://users.ece.utexas.edu/~mcdermot/arch/articles/Zynq/ug574-ultrascale-clb.pdf> (accessed on 11 November 2024).
55. Xilinx. UG949 UltraScale Device Clocking. Available online: <https://docs.amd.com/r/2021.1-English/ug949-vivado-design-methodology/UltraScale-Device-Clocking> (accessed on 11 November 2024).
56. UG474, 7 Series FPGAs Configurable Logic Block. Available online: [https://www.eng.auburn.edu/~nelson/courses/elec4200/FPGA/ug474\\_7Series\\_CLB.pdf](https://www.eng.auburn.edu/~nelson/courses/elec4200/FPGA/ug474_7Series_CLB.pdf) (accessed on 11 November 2024).
57. UG572, UltraScale Architecture Clocking Resources User Guide. Available online: <https://docs.amd.com/r/en-US/ug572-ultrascale-clocking/UltraScale-Architecture-Clocking-Resources-User-Guide> (accessed on 11 November 2024).
58. Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics 2020. Available online: <https://docs.amd.com/v/u/en-US/ds892-kintex-ultrascale-data-sheet> (accessed on 11 November 2024).
59. Xilinx. UG906 Timing Analysis. Available online: <https://docs.amd.com/r/en-US/ug906-vivado-design-analysis/Category-1-Timing> (accessed on 11 November 2024).
60. Xilinx. Vivado Overview. Available online: [https://www.xilinx.com/support/documents/sw\\_manuals/xilinx2022\\_1/ug892-vivado-design-flows-overview.pdf](https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug892-vivado-design-flows-overview.pdf) (accessed on 11 November 2024).
61. Xilinx. UG906 Report Utilization. Available online: <https://docs.amd.com/r/en-US/ug906-vivado-design-analysis/Report-Utilization> (accessed on 11 November 2024).
62. AWG4000 Series Arbitrary Waveform Generator. Available online: <https://www.tek.com/en/datasheet/arbitrary-waveform-generators-awg4000-series-datasheet> (accessed on 11 November 2024).
63. Costa, A.; Corna, N.; Garzetti, F.; Lusardi, N.; Ronconi, E.; Geraci, A. High-Performance Computing of Real-Time and Multichannel Histograms: A Full FPGA Approach. *IEEE Access* **2022**, *10*, 47524–47540. [CrossRef]
64. Tamborini, D.; Portaluppi, D.; Villa, F.; Zappa, F. Eight-Channel 21 ps Precision 10 $\mu$ s Range Time-to-Digital Converter Module. *IEEE Trans. Instrum. Meas.* **2016**, *65*, 423–430. [CrossRef]
65. Hari Prasad, K.; Chandratre, V.; Sukhwani, M. A high-density, 129-channel time-to-digital converter in FPGA for trigger-less data acquisition systems. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2023**, *1056*, 168657. [CrossRef]
66. Kuang, J.; Wang, Y.; Liu, C. A 128-Channel High Performance Time-to-Digital Converter Implemented in an UltraScale FPGA. In Proceedings of the 2017 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Atlanta, GA, USA, 21–28 October 2017; pp. 1–4. [CrossRef]
67. Acconcia, G.; Malanga, F.; Farina, S.; Ghioni, M.; Rech, I. A 1.9 ps-rms Precision Time-to-Amplitude Converter With 782 fs LSB and 0.79%-rms DNL. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 2003711. [CrossRef]
68. CERN. PicoTDC. Available online: <https://kt.cern/technologies/picotdc> (accessed on 11 November 2024).
69. Altruda, S.; Christiansen, J.; Horstmann, M.; Perktold, L.; Porret, D.; Prinzie, J. PicoTDC: A flexible 64 channel TDC with picosecond resolution. *J. Instrum.* **2023**, *18*, P07012. [CrossRef]
70. PETsys. TOFPET2 ASIC. Available online: <https://www.petsyselectronics.com/web/public/products/1> (accessed on 11 November 2024).



71. Nadig, V.; Hornisch, M.; Oehm, J.; Herweg, K.; Schulz, V.; Gundacker, S. 16-channel SiPM high-frequency readout with time-over-threshold discrimination for ultrafast time-of-flight applications. *EJNMMI Phys.* **2023**, *10*, 76. [CrossRef] [PubMed]
72. Weeroc. Temporoc. Available online: [https://www.weeroc.com/read\\_out\\_chips/temporoc/](https://www.weeroc.com/read_out_chips/temporoc/) (accessed on 11 November 2024).
73. Lusardi, N.; Garzetti, F.; Costa, A.; Cautero, M.; Corna, N.; Ronconi, E.; Brajnik, G.; Stebel, L.; Sergo, R.; Cautero, G.; et al. High-Resolution Imager Based on Time-to-Space Conversion. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 2004811. [CrossRef]
74. ACAM. ACAM TDC. Available online: <https://www.sciosense.com/wp-content/uploads/2023/12/TDC-GP22-Datasheet.pdf> (accessed on 11 November 2024).
75. Lapington, J.S.; Leach, S.A.; Sudjai, T.; Milnes, J.S.; Conneely, T.; Duran, A.; Hink, P. Investigating microchannel plate PMTs with TOFPET2 multichannel picosecond timing electronics. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2020**, *958*, 162758. [CrossRef]
76. Venturini, Y.; Maggio, C.; Tintori, C.; Ninci, D.; Mati, A.; Picchi, A.; Abba, A.; Locatelli, M.; Williams, C.; Williams, T.; et al. Characterization of a compact TDC unit with picosecond timing capabilities. In Proceedings of the 2024 IEEE Nuclear Science Symposium (NSS), Medical Imaging Conference (MIC) and Room Temperature Semiconductor Detector Conference (RTSD), Tampa, FL, USA, 26 October–2 November 2024. [CrossRef]
77. Milian, F.M.; Data, E.M.; Bersani, D.; Cirio, R.; Donetti, M.; Mazinani, M.F.; Ferrero, V.; Fiorina, E.; Hosseini, M.A.; Olivares, D.M.; et al. Use of the CERN picoTDC for timing application in particle therapy. In Proceedings of the 2024 IEEE Nuclear Science Symposium (NSS), Medical Imaging Conference (MIC) and Room Temperature Semiconductor Detector Conference (RTSD), Tampa, FL, USA, 26 October–2 November 2024. [CrossRef]
78. Bouchard, F.; England, D.; Bustard, P.J.; Heshami, K.; Sussman, B. Quantum Communication with Ultrafast Time-Bin Qubits. *PRX Quantum* **2022**, *3*, 010332. [CrossRef]
79. Joshi, C.; Sparkes, B.M.; Farsi, A.; Gerrits, T.; Verma, V.; Ramelow, S.; Nam, S.W.; Gaeta, A.L. Picosecond-resolution single-photon time lens for temporal mode quantum processing. *Optica* **2022**, *9*, 364–373. [CrossRef]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

# Real-Time Simulator for Dynamic Systems on FPGA

Sérgio N. Silva <sup>1,2,†</sup>, Mateus A. S. de S. Goldberg <sup>1,2</sup>, Lucileide M. D. da Silva <sup>1,2,3</sup> and Marcelo A. C. Fernandes <sup>1,2,4,\*,†</sup>

<sup>1</sup> InovAI Lab, nPITI/IMD, Federal University of Rio Grande do Norte (UFRN), Natal 59078-970, RN, Brazil; sergionatan@dca.ufrn.br (S.N.S.); mateus.goldbarg@dca.ufrn.br (M.A.S.d.S.G.); lucileide.dantas@ifrn.edu.br (L.M.D.d.S.)

<sup>2</sup> Leading Advanced Technologies Center of Excellence (LANCE), nPITI/IMD, UFRN, Natal 59078-970, RN, Brazil

<sup>3</sup> Federal Institute of Education, Science and Technology of Rio Grande do Norte, Paraíso, Santa Cruz 59200-000, RN, Brazil

<sup>4</sup> Department of Computer and Automation Engineering, Federal University of Rio Grande do Norte, Natal 59078-970, RN, Brazil

\* Correspondence: mfernandes@dca.ufrn.br

† These authors contributed equally to this work.

**Abstract:** This work presents the development of an embedded platform using Field Programmable Gate Arrays (FPGAs) for real-time simulation of dynamic systems in industrial plants. The platform, Real-Time Simulator for Dynamic Systems in FPGA (RTSDS-FPGA), is designed for industrial and academic applications. In industrial contexts, the RTSDS-FPGA facilitates the optimization and tuning of embedded control algorithms, while in academia, it supports research on new embedded solutions in automation and control. It is also an educational tool for undergraduate and postgraduate students developing embedded control system projects. Additionally, the simulator accelerates the simulation of slow dynamic systems, significantly reducing overall simulation time. Experimental results demonstrate the platform's capability to perform real-time simulations effectively, validating its accuracy and performance through comparative analyses with established software tools such as Matlab/Simulink.

**Keywords:** real-time; FPGA; embedded systems; dynamic systems

## 1. Introduction

The implementation of new control algorithms for embedded systems on engines, industrial processes, vehicles, and aircraft requires tests typically using the device that needs to be controlled. These tests are usually expensive and are time-consuming to develop and improve. One example is the control of an anti-lock braking system (ABS) for an automobile, where field testing requires a team of different professionals, and adjustments of the algorithm can take hours or even days. Hence, the implementation of procedures using real systems can be a complicated and costly task. In order to avoid this direct approach, testing can employ prototypes constructed at smaller scales or use simulations, as in the case of the algorithms employed for aircraft and ships. Some companies use off-line (non-real-time) simulation software such as Matlab/Simulink (license 596681) to check the validity of algorithms, reducing the need for experiments in the real environment. A negative aspect of these tests is the possibility of ignoring specific data, so the values may not be coherent with reality. For this reason, tests using real systems should never be eliminated.

Over the last decade, there have been many developmental studies of real-time simulation platforms. These studies include the implementation of real-time systems using micro-controllers ( $\mu C$ ), micro-processors ( $\mu P$ ) or general-purpose processors (GPPs) [1–3], as well as specific solutions using real-time Field Programmable Gate Arrays (FPGAs) [4–11], systems that are easy to implement [12], proposals for the teaching of engineering [13,14], and studies aimed at improving the performance of real-time systems [15,16]. Research in



this area has made rapid progress in terms of the speed and ease of development associated with new hardware platforms. Real-time simulation techniques have included the use of hardware such as digital signal processors, general-purpose processors, and reconfigurable computational solutions that employ FPGAs. The choice of FPGAs as the main elements in studies is easily justified by their better performance and flexibility, compared to other hardware [17].

The use of FPGA platforms for real-time simulation has gained prominence across various application areas, mainly due to their parallel processing capabilities and flexibility. Recent studies, such as that by [18], propose methodologies for assessing the credibility of simulations in complex electronic control systems using FPGA-based Hardware-in-the-Loop (HIL), highlighting the importance of accuracy in replicating actual operating conditions. One study presents another significant advancement [19], in which the authors developed a real-time simulator for power converters with optimized switching model parameters using FPGA. Ref. [20] explore the implementation of real-time simulators with floating-point arithmetic, demonstrating that low-cost solutions can achieve performance comparable to high-cost commercial simulators. Additionally, ref. [21]’s research on stability simulation in power systems and the application of digital control techniques in single-phase PFC converters by [22] highlight the potential of FPGAs in applications requiring high precision and efficiency. These studies underscore the versatility and power of FPGAs as a solution for complex real-time simulations, making them an attractive choice for various industrial and academic applications.

Recent advancements in FPGA-based systems have expanded the potential of real-time simulation and Hardware-in-the-Loop (HIL) applications. Ref. [23] demonstrated a novel real-time digital simulation platform for differential drive mobile robots, showcasing the integration of electrical and mechanical models within an FPGA environment, significantly improving robotic simulations’ fidelity and speed. Additionally, ref. [24] addressed the complexities of transforming SIMULINK models into FPGA-in-the-Loop systems, proposing a systematic approach that enhances the development of digital twins for industrial applications. Moreover, the work by [25] presents a universal equivalent model for real-time CPU-FPGA co-simulation, specifically targeting hybrid cascaded multi-level converters, which marks a significant step toward more efficient power electronics simulations. These contributions underscore the evolving capabilities of FPGAs in handling increasingly complex simulation tasks, making them indispensable tools in modern engineering and research.

This work presents four main contributions to the field of real-time simulation using FPGAs:

- Development of a real-time simulation platform based on FPGA: The proposed platform offers a low-cost and high-efficiency solution capable of reproducing the performance of commercial simulators across various applications, from electrical to mechanical systems.
- Integration of floating-point arithmetic for enhanced precision: Including floating-point arithmetic in the FPGA enables high-precision simulations, overcoming common challenges in modeling complex dynamic systems.
- Methodology for credibility assessment: This work proposes a novel approach to assess the credibility of simulations, providing a quantitative tool for validating real-time models.
- Flexibility and broad applicability: The platform’s modular and flexible architecture allows its application across a variety of dynamic systems, whether electrical, mechanical, or hybrid.

## 2. Related Work

In the work of [12], a low-cost RTS was proposed, which could be constructed in the laboratory for use in electronic power systems. The system could be implemented on a computer possessing a multi-core processor together with data acquisition and signal conditioning systems. In another similar approach, but using a general-purpose proces-

sor, a low-cost RTS was described that could easily be developed in the laboratory using  $\mu$ Ps and  $\mu$ Cs [1]. This proposal involved a distributed architecture composed of different hardware platforms, enabling it to be constructed with limited resources. As an example, a case study was presented where the RTSDS was composed of a personal computer (Intel DN2800MT Mini-ITX) associated with low-cost and easily obtained hardware platforms (with embedded software), such as the Arduino and CY8CKIT-001 kits [26–30]. RTS proposals using general-purpose processors have been described by [2,3], offering distributed software solutions for RTS, in which many computers are used in an internet network in order to simulate real-time systems.

The benefits of using RTS in undergraduate and postgraduate engineering courses are described in detail in [13,14], which describes several commercial RTS tools. The signal processing techniques presented in [15] aim to reduce the sampling frequency and computational cost associated with RTS. It was demonstrated that the computational cost could be reduced up to 100-fold, without compromising the precision of the simulation. In [16], a methodology consisting of seven steps is described for the validation of real-time simulation models. Real-time simulators have been proposed for power systems that use FPGAs associated with computers, where the main advantage is parallel implementation of the simulated dynamic systems [6,8,9].

Hardware-in-the-loop (HIL) systems have been described by [4,5,10,31]. In [4], an FPGA platform was used for digital hardware implementation of a real-time simulator for a multi-phase system. According to the author, this could be developed using an ordinary computer, but this would not allow the real-time simulations that are essential for control using HIL. The implementations employed a Xilinx SP605 kit that included a Spartan-6 FPGA to facilitate the implementation of the system inputs and outputs. The kit has a 200 MHz clock, and the PWM signals use 50 and 100 MHz clocks derived from the main clock, resulting in the model having a simulation step of 40.96  $\mu$ s. The performance of the real-time system was evaluated using a configuration with an induction motor model (DT-PIM) in Matlab/Simulink and the system step used the fourth-order Runge–Kutta method. The experimental results and statistical comparisons showed satisfactory performance of the Matlab/System Generator environment, in the absence of bench tests. Already, in [10], a simulation framework with HIL has been described to estimate the state of Li-ion batteries in electric vehicles. In this work, the framework was implemented on FPGA and it shows that this is a fast simulation strategy. The high-performance real-time simulator on FPGA is showed in [31]. In this work, the simulator was implemented on Altera's FPGA using the DE3-150 development kit and a low fixed sample time was achieved, at about 500 ns.

In [11], an implementation is presented of a dynamic system molecular model. In order to improve the execution time, this model is accelerated using the FPGA. In [5], a model is proposed for a permanent magnet synchronous motor. The system uses the HIL approach, where a Matlab/Simulink vehicle model is executed in conjunction with an embedded engine model in a Virtex-6 (XC6VLX240T) FPGA of the ML605 development kit. This system performs its actions with a cycle of 2  $\mu$ s, and in [7], the real-time Hardware-in-the-Loop (HIL) simulation of vehicle systems is presented. The acceleration is based on FPGA technology using Runge–Kutta methods to solve the Ordinary Differential Equation (ODE), and the results are compared with GPP (2.83 GHz Intel Core 2 Quad CPU).

In the work of [4], a simulation was made of a photo-voltaic system using the ML605 kit to develop a simulation environment that enabled a step of approximately 1  $\mu$ s. The use of the FPGA was determined by the maximum step size that the system supported, approximately 5  $\mu$ s, which was not viable for studies using platforms such as  $\times 86$  CPUs or digital signal processors (DSPs). This system was also implemented in Matlab/Simulink, generating off-line (non-real-time) simulation data for comparison and validation purposes. Auxiliary hardware items including a PC and DSP were used for data acquisition and conditioning of the final data. For communication, connection of the FPGA with the PC employed a Peripheral Component Interconnect Express (PCIe) bus, enabling a communication rate of up to 5 Gbps. Successful implementation was achieved with a time step of 1  $\mu$ s.

An example of FPGA for real-time simulations on dynamic systems is shown in [32]. This version works with a time step between 150  $\mu$ s and 200  $\mu$ s. The main target of this product is the simulation of power systems and High-Voltage Direct Current (HVDC). An overview of a real-time simulations approach to a smart-grid is presented in [33]. This work shows the current context of real-time simulators and the new demands from smart-grid and similar systems, such as machine-to-machine (M2M) systems, the Internet of Things (IoT), and others.

The work presented in [18] presents a methodology for assessing the credibility of simulations in electronic control systems using an FPGA-based HIL platform. The work addresses the need for efficient testing methods for complex control systems by proposing a HIL platform replicating the actual operating conditions of these systems. The evaluation methodology considers performance, time-domain response, and frequency-domain response, enabling the quantification of simulation credibility. The results show that the platform achieves credibility indices above 60%, offering a quantitative approach for validating simulations in electronic control systems. Similarly, ref. [19] developed and implemented a real-time simulator based on ADC, focusing on the optimal selection of switch model parameters. The work deals with modeling power converters using the Associated Discrete Circuit (ADC) method and Modified Nodal Analysis (MNA). The authors employed the Particle Swarm Optimization (PSO) algorithm to optimize the simulator's behavior to adjust conductance and damping resistance values, minimizing numerical errors and improving simulation accuracy. The results demonstrate that the proposed simulator accurately reproduces the voltage and current waveforms of the modeled converters, validating the method's effectiveness in real-time simulation.

The manuscript presented in [20] proposes the implementation of a real-time simulator for electrical systems using floating-point arithmetic on an FPGA. The study aims to develop a low-cost simulation platform that offers performance comparable to high-cost commercial simulators. The authors use Hardware Description Language (HDL) to ensure flexibility in FPGA architecture selection and avoid dependence on commercial packages. The results demonstrate that the developed simulator can operate in real time, allowing for precise analysis of interactions between the simulated electrical system and its control, with efficiency similar to commercial solutions. Similarly, ref. [22] explores the application of HIL techniques and digital control in single-phase PFC converters. The study presents an approach that allows for testing digital controllers under realistic conditions without the need for physical prototypes, using FPGAs to implement the mathematical models of the converters in real time. The results indicate that the proposed HIL methodology provides a practical and cost-efficient verification of controller performance, accelerating development and reducing risks associated with designing new PFC converters. Additionally, ref. [34] presents a high-performance and cost-effective FPGA-based motor drive emulator. The emulator is designed to replicate the dynamic behavior of permanent magnet synchronous machines (PMSMs), including the voltage source inverter and control strategy, using a digital description in Verilog. The study explores two design methods, parallel and sequential, to optimize FPGA resource usage, resulting in a solution that accurately emulates motor behavior while saving resources and costs. The results show that the emulator is efficient for many applications, from developing new control strategies to early fault detection and maintaining the portability and flexibility of FPGA platforms. Furthermore, ref. [25] presents a universal equivalent model for real-time CPU/FPGA co-simulation applied to hybrid cascaded multilevel converters. The study focuses on creating a model that can be used for simulating complex power converters, combining CPU processing with the parallelism capabilities of FPGAs. The proposed model was validated through real-time simulations, demonstrating its effectiveness in improving the accuracy and efficiency of co-simulation systems, particularly in power electronics applications that require high temporal resolution and computational performance.

The research presented in [35] provides an overview of interface algorithms, communication signals, and delay management in real-time co-simulation of distributed power

systems. The study addresses the challenges of maintaining stability and accuracy in geographically distributed simulations, where communication latency between simulators can compromise results. The authors review various interface techniques and discuss methods to mitigate the effects of communication delays, ensuring that distributed simulators remain synchronized and stable, thereby improving the fidelity of simulations in complex power systems. Complementing this discussion, ref. [24] address the automatic transformation of SIMULINK models into FPGA-in-the-Loop (FiL) systems for real-time simulation, identifying the main challenges in the C-to-HDL code synthesis process and proposing a detailed roadmap to overcome them. The research demonstrates, step by step, how to convert SIMULINK models into FiL systems using tools like Vitis HLS and Vivado, validating the resulting FiL system to ensure its real-time functionality. This approach offers a practical solution for engineers seeking to automate the design process of FiL systems, facilitating the development of digital twins and other high-performance industrial applications.

Table 1 compares the proposed methodology with recent studies on real-time simulations using FPGAs and other platforms. The criteria selected for comparison include the platform used, the simulation methods employed, the precision achieved, the cost, the flexibility and applicability, the real-time efficiency, and the type of dynamic system simulated. These criteria were chosen because they are the most relevant for evaluating the performance and innovation of the proposed methods. Table 1 shows that besides offering simulation precision and reduced cost, the approach presented in this paper stands out for its flexibility and applicability to a wide range of dynamic systems. Compared to the studies analyzed, the proposed methodology achieves comparable or superior precision, providing a versatile platform that can be applied in different contexts, from electrical to mechanical systems. This reflects the contributions of this work to the field, offering an efficient solution for real-time simulations adaptable to various needs and applications.

**Table 1.** Comparison of RTSDS-FPGA with recent studies.

Criterion	RTSDS-FPGA (Our Work)	Dai et al. (2020) [18]	Queiroz et al. (2021) [20]	Lamo et al. (2021) [22]	Bieber et al. (2023) [25]
Platform used	FPGA Virtex-6 XC6VLX240T	FPGA Spartan-6	FPGA		CPU/FPGA
Simulation methods	Euler, Runge–Kutta (3rd, 4th)	Runge–Kutta	HDL	Digital Control	Parallelism
Simulation precision	High (up to 6 decimal places)	Medium	High	High	High
Flexibility	High (various systems)	Specific	Specific	Specific	High (Power Systems)
Real-Time efficiency	High (cycle time 210 ns)	High	High	High	High
Type of dynamic system simulated	General dynamic systems	Electronic control	Electrical systems	PFC converters	Power converters

The studies presented above are individually focused, in that they are aimed at specific energy or automotive applications. In contrast, the RTSDS-FPGA system presented in this article has the ability to implement several dynamic systems. Another important point is that the system developed here not only performs tests simulated using software, but also performs bench tests. The main purpose of the RTSDS-FPGA simulator is to provide a simulation platform to assist in the development of tools for embedded control systems, focusing on reducing the simulation step in order to be able to work with highly dynamic or more complex systems. Novel trends, such as real-time trends with data-intensive computing (digital twins in Industry 4.0, for example) [36,37], can be solved with the

RTSDS-FPGA simulator proposed here. The RTSDS-FPGA also allows the possibility of accelerating the simulation of systems that have slow dynamics, with execution in non-real time, and it can also be used in the academic environment as a programmable teaching platform.

The main contribution of this paper is the proposal of a real-time simulator for dynamic systems, using an FPGA. The advantage of this new FPGA development environment is that parallel operation is intrinsic to its architecture. This enables the use of smaller sampling granularity, allowing application to highly dynamic systems, or faster simulation of systems with suitable dynamics, since there are substantial gains regarding execution time. Furthermore, the FPGA provides flexibility in working with hardware developed for specific applications. The ability to deliver faster speed is a critical aspect, given that simulations that generally last for hours can be performed in a few minutes using the FPGA. All the devices external to the FPGA should be able to accompany the decrease in the sampling period, or reduction techniques should be used. A large number of academic applications is an important consideration. Research institutions such as universities and institutes offer courses in the areas of embedded systems, digital systems, automation, and control. All make use of practical experiments in laboratories, involving teaching equipment and test systems. In many cases, the availability of the equipment required is minimal or nonexistent, due to the high purchasing costs, encouraging the use of real-time simulations by students and teachers. However, although simulations can be used to test algorithms, they do not replace testing using real systems. Another possible use of simulations is to reduce the wear caused by the constant use of teaching equipment. For example, in a control application, the platform could be controlled with a PID. Another contribution of this work is the creation of a library of IP cores for projects modeling hardware, such as vehicles and tank/pump systems, as well as for Runge–Kutta methods and other circuits that could be constructed.

### 3. Architecture of the RTSDS-FPGA

An example of a first-order dynamic system for the proposed platform can be expressed as

$$\eta \frac{dv(t)}{dt} = \sum_{i=0}^P \delta_i(\gamma_i(t)) + \sum_{\kappa=0}^H \psi_{\kappa}(v(t)), \quad (1)$$

where  $P$  is the number of independent variables,  $\gamma_i(t)$  is the  $i$ -th independent variable,  $\delta_i(\cdot)$  is the  $i$ -th function associated with the  $i$ -th independent variable,  $\gamma_i(t)$ ,  $\eta$  is a constant,  $H$  is the number of dependent variables,  $v(t)$  is the dependent variable, and  $\psi_{\kappa}(\cdot)$  is the  $\kappa$ -th function associated with the dependent variable,  $v(t)$ , which can be expressed by

$$v(t) = \begin{cases} w(t) & \text{for } w(t) < v_{max} \\ v_{max} & \text{for } w(t) \geq v_{max} \end{cases}, \quad (2)$$

where  $v_{max}$  is the maximum value of the  $v(t)$ , and  $w(t)$  is expressed as

$$w(t) = v_0 + \int_{t_0}^t L_s(s, v(s)) ds, \quad t > t_0 \text{ e } v_0 \geq 0, \quad (3)$$

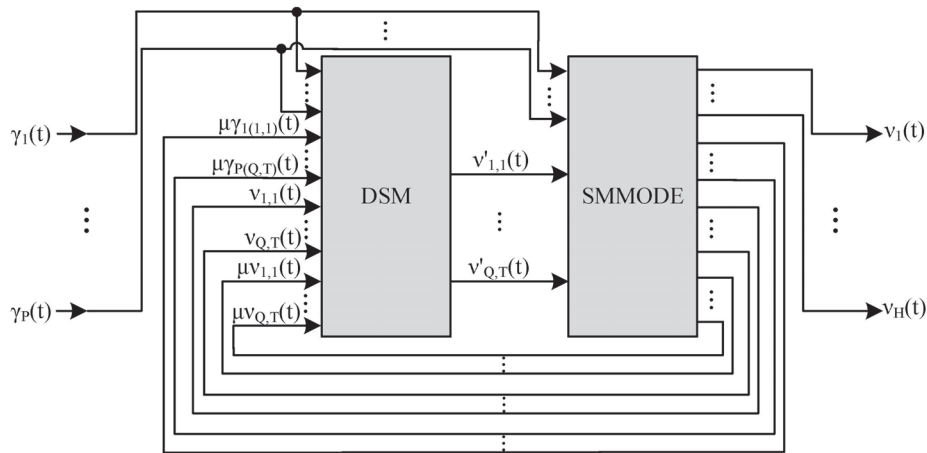
where  $v_0$  is the initial condition of the variable  $v$ , and  $L_s$  is expressed by

$$L(s, v(s)) = \frac{1}{\eta} \left( \sum_{i=0}^P \delta_i(\gamma_i(s)) + \sum_{\kappa=0}^H \psi_{\kappa}(v(s)) \right). \quad (4)$$

The RTSDS-FPGA architecture, for Equation (4), is presented in Figure 1, where it is composed of two modules, here called the Dynamic System Module (DSM) and Solution Methods Module for ODEs (SMMODE). The DSM implements the dynamic system model and SMMODE is responsible for the implementation of the solution methods module for



ODEs. The SMMODE executes  $O$  different ODE methods in parallel, and for each  $o$ -th method,  $Q$  copies of the dynamic model are necessary, implemented on DSM. DSM and SMMODE are detailed in following subsections.

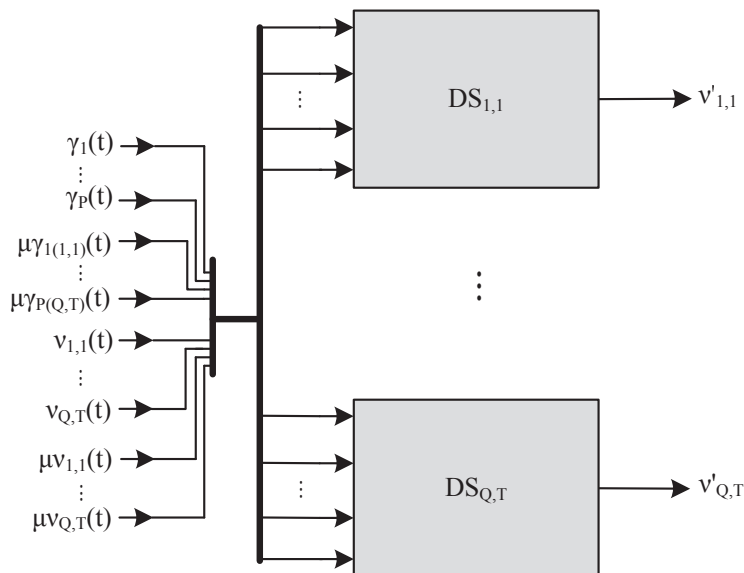


**Figure 1.** The ODS architecture.

### 3.1. Dynamic System Module (DSM)

The DSM is the responsible for implementing the set of equations which define the dynamic system. Using the example with the first-order Equation (1), the DSM is composed of  $P + Q \times O$  inputs (see Figures 1 and 2).

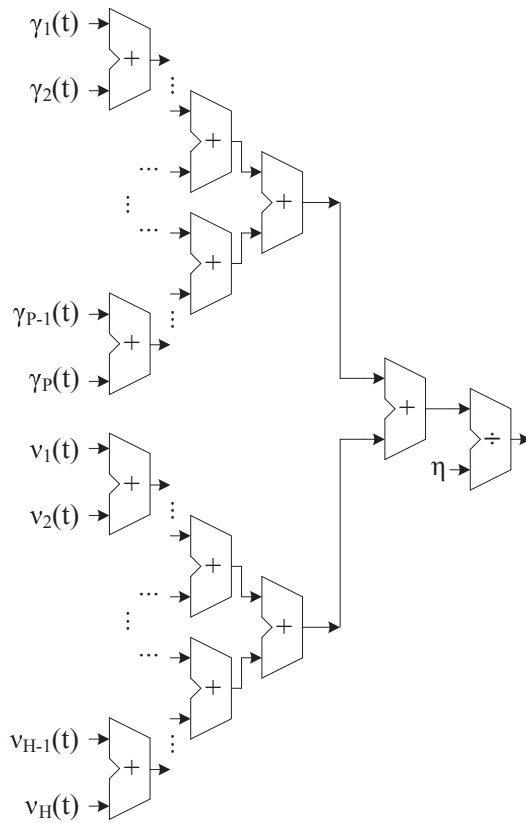
Furthermore, this system may still require addition inputs, since the solution methods for ODEs require one or more calculation stages to estimate the next value, as can be seen in [38]. Thus, Figure 2 represents the architecture of the DSM to Equation (4), where  $DS_{Q,T}$  is the  $Q$ -th reply of the system for a given solution method for ODE.



**Figure 2.** Details of the DSM architecture.

Figure 3 shows the implementation of the dynamic system in more detail. This circuit is the representation of Equation (4).

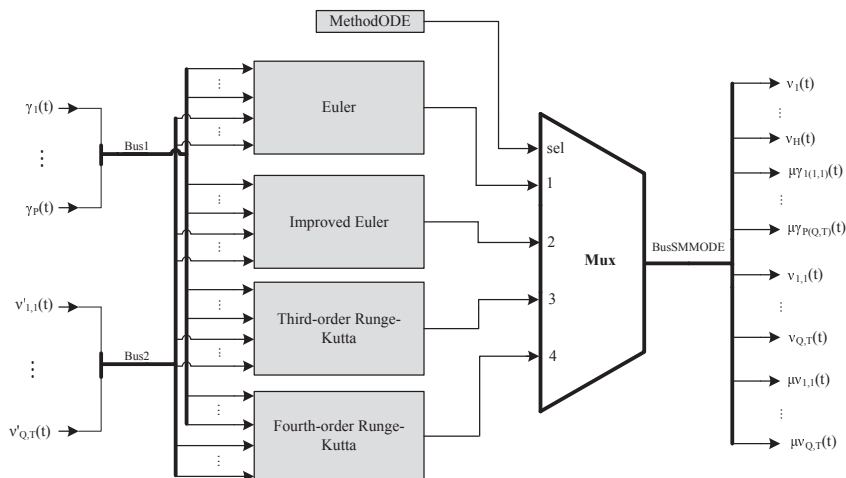




**Figure 3.** Details of the DS circuit.

### 3.2. Solution Methods Module for ODE (SMMODE)

This module has the role of collecting the value of the differentiation generated by the dynamical system and calculating the integral value. For this, we use methods such as Euler, Euler Improved, the third-order Runge–Kutta method, or the fourth-order Runge–Kutta method as a way to perform this calculation. This module implements all methods in parallel, as users have the option to choose the method by the variable *MethodODE* present in the description of the logical architecture. Figure 4 illustrates the architecture of the solution methods module for ODE.



**Figure 4.** Details of the SMMODE architecture.

#### 3.2.1. Euler Method

The Euler method approximating the integral  $v'(t)$ , given a range  $[t_n, t_{n+1}]$ , multiplying the function value, in the given moment, for system step value, by the step value of the

system, describes  $t_s$  in the logical architecture. Thus, the equation which express the Euler method is given by

$$v(n+1) = v(n) + t_s f(\gamma_P(n), v(n)), \quad (5)$$

where  $v(n+1)$  is the state to be estimated,  $v(n)$  is the actual state,  $t_s$  is the method step, and  $f(\gamma_P(n), v(n))$  is the value of the differential function for the actual values of  $\gamma_P(n)$  and  $v(n)$ , where  $\gamma_P(n)$  is the  $P$ -th variable input of the system in instant  $n$ . Thus, one can represent the Euler method as a circuit, as shown in Figure 5.

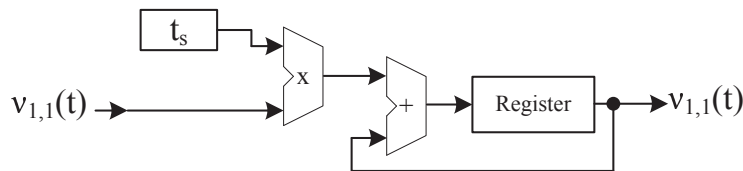


Figure 5. Euler method circuit.

### 3.2.2. Improved Euler Method

Analogous to the Euler method is the improved Euler method. This method uses a second step to adjust the output value of the original Euler method and finally average between the two points found. Thus, the improved Euler method can be expressed by

$$v(n+1)^* = v(n) + t_s f(\gamma_P(n), v(n)), \quad (6)$$

$$v(n+1) = v(n) + \frac{t_s (f(\gamma_P(n), v(n)) + f(\gamma_P(n+1), v(n+1)^*))}{2}, \quad (7)$$

where  $v(n+1)^*$  is the output value of the Euler method and  $f(\gamma_P(n+1), v(n+1)^*)$  is the function value estimate through the Euler method. Thus, the improved Euler method can be represented by Figure 6.

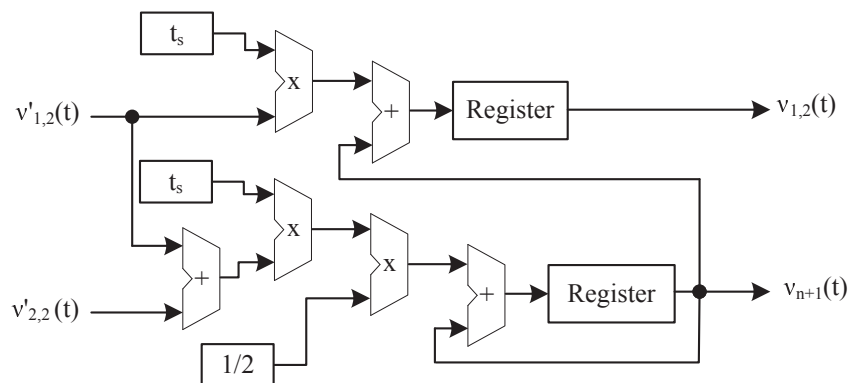


Figure 6. Improved Euler method circuit.

### 3.2.3. Third-Order Runge–Kutta Method

The third-order Runge–Kutta method makes use of a point in the middle of the range as a way to refine the final value. Thus, the third-order Runge–Kutta method can be expressed by

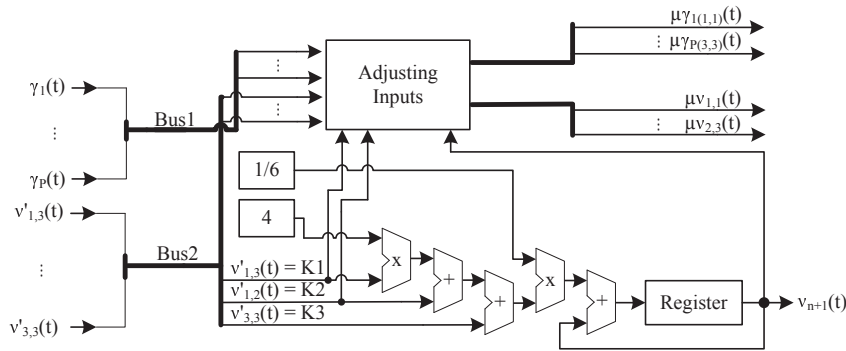
$$v(n+1) = v(n) + \frac{t_s (K_1 + 4K_2 + K_3)}{6}, \quad (8)$$

$$K_1 = f(\gamma_P(n), v(n)), \quad (9)$$

$$K_2 = f(\gamma_P(n) + \frac{t_s}{2}, v(n) + \frac{t_s}{2} K_1), \quad (10)$$

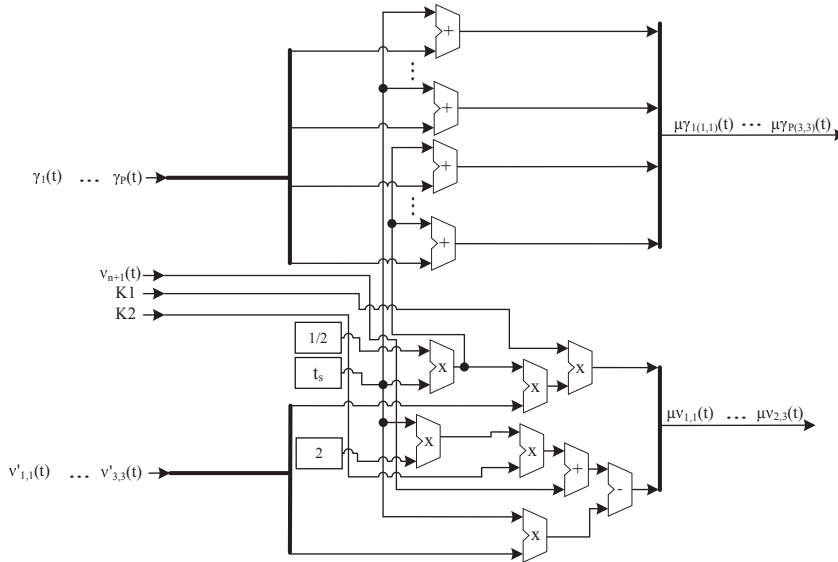
$$K_3 = f(\gamma_P(n) + t_s, v(n) + 2t_s K_2 - t_s K_1), \quad (11)$$

where  $K_1$ ,  $K_2$ , and  $K_3$  are partial estimates for a certain  $v(n)$ . Thus, the third-order Runge–Kutta method can be depicted by Figure 7.



**Figure 7.** Third-order Runge–Kutta method circuit.

Figure 8 shows the implementation of the Adjusting Inputs block for the third-order Runge–Kutta method.



**Figure 8.** Adjusting inputs circuit for the RK3.

### 3.2.4. Fourth-Order Runge–Kutta Method

The fourth-order Runge–Kutta method is the most accurate method. This method use two estimated points in the middle of the range as a way to refine the final value. Thus, the fourth-order Runge–Kutta method can be expressed by

$$v(n+1) = v(n) + \frac{(K_1 + 2K_2 + 2K_3 + K_4)}{6}, \quad (12)$$

$$K_1 = t_s f(\gamma_P(n), v(n)), \quad (13)$$

$$K_2 = t_s f(\gamma_P(n), v(n) + \frac{t_s}{2} K_1), \quad (14)$$

$$K_3 = t_s f(\gamma_P(n) + \frac{t_s}{2}, v(n) + \frac{t_s}{2} K_2), \quad (15)$$

$$K_4 = t_s f(\gamma_P(n) + t_s, v(n) + K_3), \quad (16)$$

where  $K_1$ ,  $K_2$ ,  $K_3$ , and  $K_4$  are partial estimates for a certain  $v(n)$ . Thus, the fourth-order Runge–Kutta method is depicted in Figure 9.

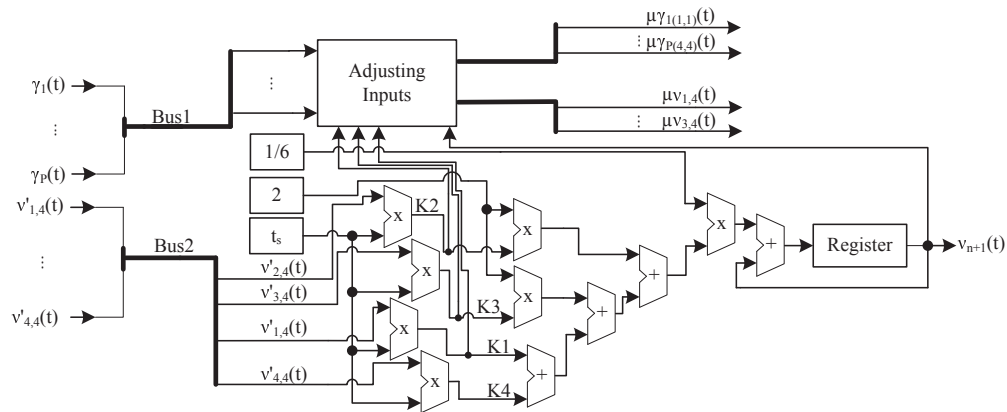


Figure 9. Fourth-order Runge–Kutta method circuit.

Figure 10 shows the implementation of the Adjusting Inputs block for the fourth-order Runge–Kutta method.

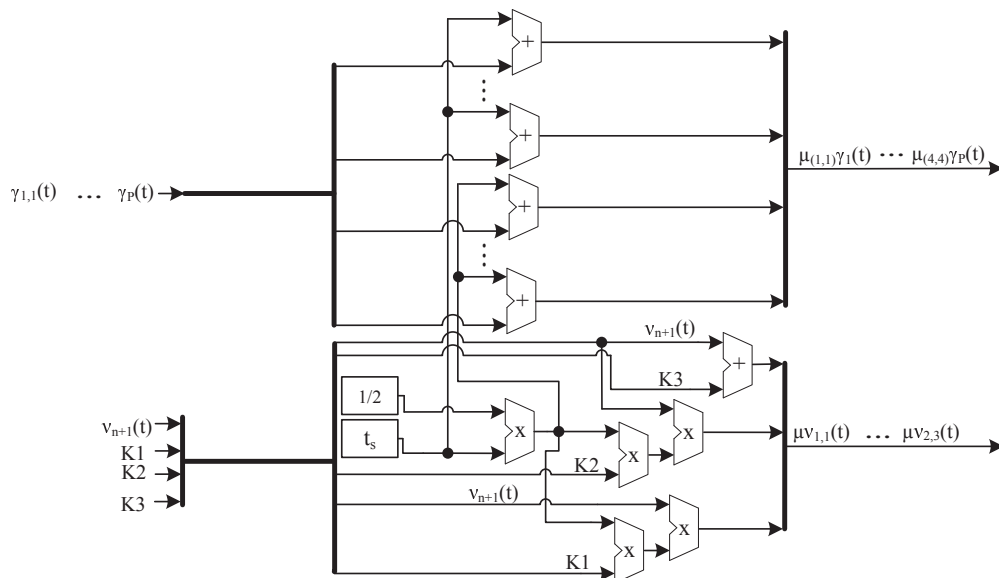


Figure 10. Adjusting inputs circuit for the RK4.

## 4. Development of the Prototype and Results

### 4.1. Prototype Description

The simulated model presented in this work follows the same test model presented in [1], aiming for continuity of reading. Thus, they developed a model of a longitudinal vehicle, shown in Figure 11. It was used as the basis of the model defined in [39]. In this model, the input parameters were the torque delivered to the wheels and the longitudinal inclination angle of the track. The output parameter was the speed of the vehicle.

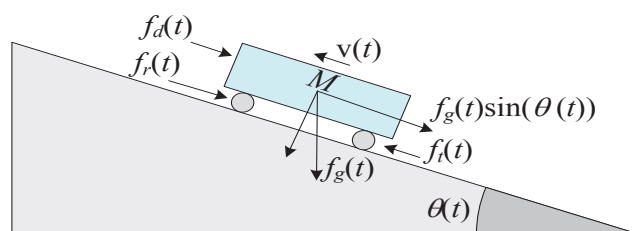


Figure 11. Scheme of the longitudinal model of the vehicle.

The longitudinal model of the vehicle [39], shown in Figure 11, can be described for the following expression:

$$M \frac{dv(t)}{dt} = f_t(t) - f_a(t), \quad (17)$$

where  $M$  is the mass of the vehicle (in Kg),  $v(t)$  is the linear velocity of the vehicle (in m/s),  $f_t(t)$  is the traction force exerted by the vehicle (in N), and  $f_a(t)$  is the friction force (in N). The traction force can be expressed by

$$f_t(t) = \frac{\tau_m(t)}{r}, \quad (18)$$

where  $r$  is the radius of the wheels of the vehicle (in m) and  $\tau_m(t)$  is the torque (in Nm) generated by the motor, which can be expressed by

$$\tau_m(t) = V_e(t) \tau_m^{max}, \quad (19)$$

where  $\tau_m^{max}$  is the maximum torque (Nm) and  $V_e(t)$  is the first signal of continuous actuation.

As shown in [39], the friction force,  $f_a(t)$ , can be expressed by

$$f_a(t) = f_d(t) + f_r(t) + f_g(t) \sin(\theta(t)), \quad (20)$$

where  $f_d(t)$  is the aerodynamic friction force (in N),  $f_r(t)$  is the rolling resistance force (in N),  $f_g(t)$  is the force due to gravity (in N), and  $\theta(t)$  is the inclination angle of the plane on which the vehicle is located. The variable  $\theta(t)$  can be expressed by

$$\theta(t) = V_o(t) \theta^{max}, \quad (21)$$

where  $\theta^{max}$  is the maximum inclination angle, and  $V_o(t)$  is the second signal of continuous actuation.

The aerodynamic friction is given by

$$f_d(t) = \frac{1}{2} \rho C_d A_{fr} x^2(t), \quad (22)$$

where  $\rho$  is the density of air,  $C_d$  is the aerodynamic drag coefficient, and  $A_{fr}$  is the frontal area of the vehicle (in m<sup>2</sup>). The rolling resistance force is given by

$$f_r(t) = Mg (C_0 + C_1 x^2(t)), \quad (23)$$

where  $C_0$  and  $C_1$  are the rolling coefficients, and  $g$  is the acceleration due to gravity (in m/s<sup>2</sup>). Finally, the gravitational force is given by

$$f_g(t) = Mg. \quad (24)$$

The dynamic system presented in Equation (17) was simulated in real time, using methods to solve the ODEs (Euler, improved Euler, and third- and fourth-order Runge-Kutta), selected through the variable *MethodODE*. Equation (25) can be expressed by

$$v(t) = \begin{cases} u(t) & \text{para } u(t) < v_{max} \\ v_{max} & \text{para } u(t) \geq v_{max} \end{cases}, \quad (25)$$

where  $v_{max}$  is the maximum velocity of the vehicle, and  $u(t)$  is given by

$$u(t) = v_0 + \int_{t_0}^t h_{vs}(s, v(s)) ds, \quad t > t_0 \text{ e } v_0 \geq 0, \quad (26)$$

where  $v_0$  is the initial condition of the velocity of the vehicle, and  $h_{vs}(s, v(s))$  can be expressed as

$$h_{vs}(s, v(s)) = \frac{1}{M} \left( V_e(s) \frac{\tau_m^{max}}{r} - \frac{1}{2} \rho C_d A_{fr} v^2(s) - Mg \left( C_0 + C_1 v^2(s) \right) - Mg \sin(V_o(s) \theta^{max}) \right). \quad (27)$$

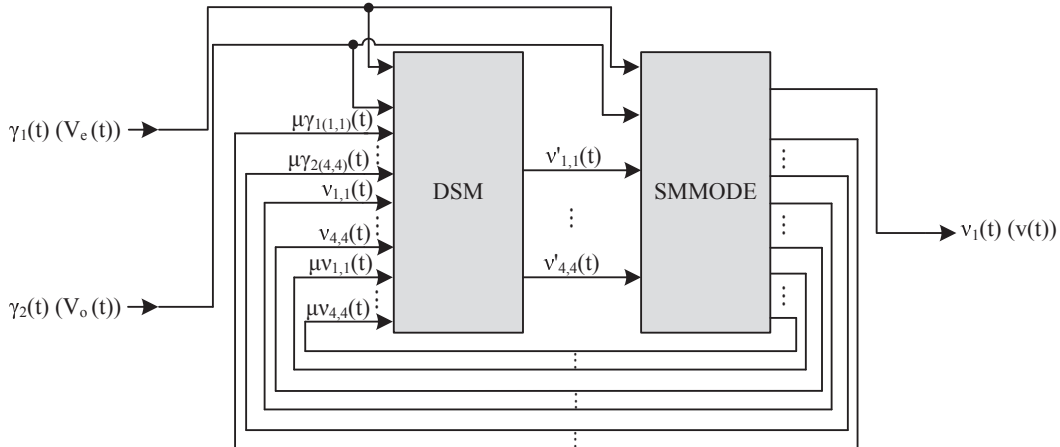
#### 4.2. Prototype Implementation

The implementation of the ODE described in the previous section shows the ideal realization of the ODS prototype, as shown in Figure 12, which has two generic inputs  $P = 2$ ,  $V_e(t)$  and  $V_o(t)$ , and one output  $H = 1$ ,  $v(t)$ . Considering Equation (27), the inputs described refer to torque and angle values and the output refers to the velocity.



**Figure 12.** ODS with two inputs,  $P = 2$  (torque,  $V_e(t)$ , and inclination angle  $V_o(t)$ ), and one output,  $H = 1$  (vehicle speed in a longitudinal direction,  $v(t)$ ).

The prototype was developed using two DAMs (DAM-1 and DAM-2) and one SGM (SGM-1), translating the signals from the world to the platform and from the platform to the world. DAM-1 and DAM-2, illustrated in Figure 13, are A/D converters responsible for receiving the input signals,  $V_e(t)$  and  $V_o(t)$ , and SGM-1, also shown in Figure 13, is a D/A converter responsible for system output,  $v(t)$ . The physical architecture of this system is shown in Figure 13, constituted, as described in Section 3, by one DSM and one SMMODE.



**Figure 13.** ODS implemented for the prototype.

Thus, conforming to Equation (27), the dynamic system was implemented in FPGA, based on the generic circuit illustrated in Figure 14. This circuit is constituted of sub-circuits, corresponding to parts of Equation (27). The sub-circuit responsible for the aerodynamic friction force, described in Equation (22), is shown in Figure 15, called ADF. The sub-circuit responsible for the rolling resistance force, described in Equation (23), is shown in Figure 16, called RRF.



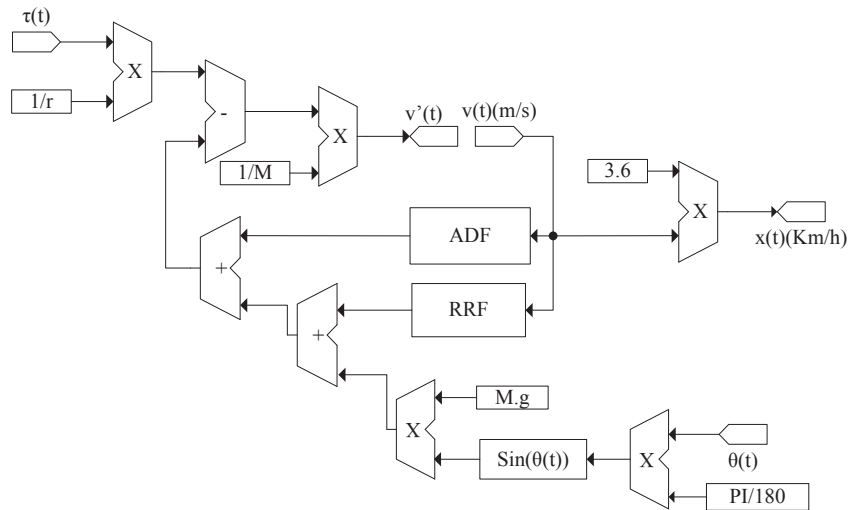


Figure 14. Prototype of the circuit for the DS of the vehicle.

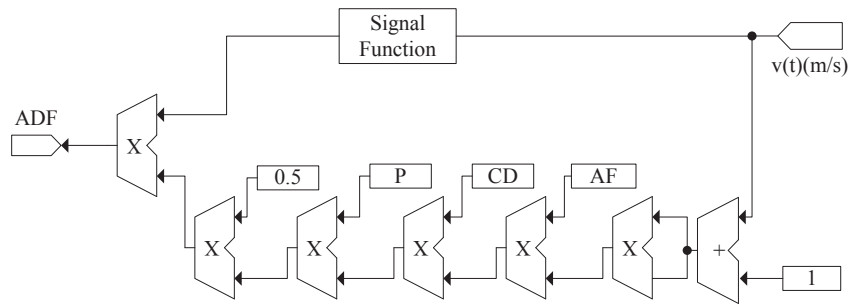


Figure 15. In detail, the circuit responsible for the aerodynamic friction force.

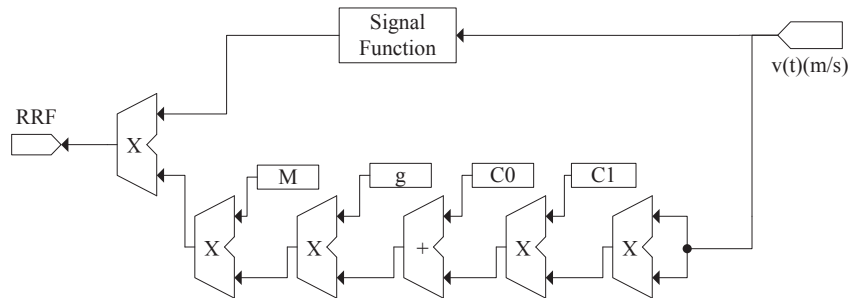


Figure 16. In detail, the circuit responsible for the rolling resistance force.

#### 4.3. Methodology

In the process of obtaining the results, some parameterizations were adopted as a way to gain greater control. Two sets of results were obtained:

- Simulation of the dynamic system using Matlab/Simulink in comparison with the dynamic system using Matlab/System Generator;
- Simulation of the dynamic system using RTSDS-FPGA.

The test sets have a difference because their numerical configurations are distinct, but they do not make the results unfeasible. The output value for Matlab/Simulink is represented in a 64-bit floating-point format, and the same output value in Matlab/System Generator is represented with a signed 32-bit fixed-point format, 16 of which are used for the representation of the fractional part.

Thus, the two scenarios were configured as follows:

- Scenario 1: The input of the dynamic system with constant values;

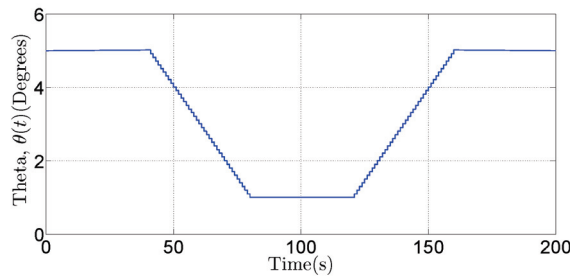
- Scenario 2: The input of the dynamic system with variable values.

The dynamic system has set solution methods for ODE. Thus, for each of the cases discussed above, tests were performed with each of the methods described below:

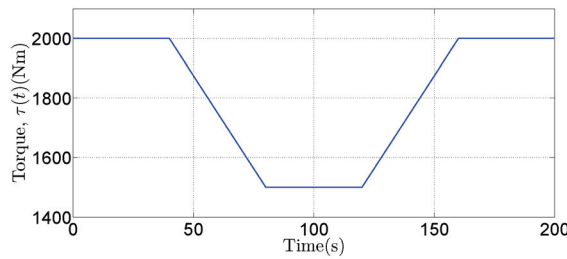
- Euler;
- Improved Euler;
- Third-order Runge–Kutta;
- Fourth-order Runge–Kutta.

The implementation was carried out in a completely parallel way, allowing the choice of methods and differentiating from the models implemented in the literature.

The angle of inclination varies according to the graph shown in Figure 17, and the torque value varies according to the graph shown in Figure 18.



**Figure 17.** Theta inclination angle variation signal.



**Figure 18.** Torque variation signal.

In order to analyze the reliability of the obtained results, we used error measures among the obtained data. The analyses are based on mean error, mean square error, and error variance. The statistical calculations have as reference the value of floating-point simulation acquired with the simulation performed in Matlab/Simulink. The calculation of the mean error is shown below.

$$M_e(X) = \frac{1}{N} \sum_{i=0}^{N-1} (X(i) - Y(i)), \quad (28)$$

where  $M_e(X)$  is the value of the mean error,  $X(i)$  is the value of the  $i$ -th collected sample, and  $Y(i)$  is the value of the simulation in the software for the same instant. The calculation for the mean square error is defined as

$$M_{se}(X) = \frac{1}{N} \sum_{i=0}^{N-1} (X(i) - Y(i))^2, \quad (29)$$

where  $M_{se}(X)$  is the value of the mean square error,  $X(i)$  is the value of the  $i$ -th collected sample, and  $Y(i)$  is the value of the simulation in the software for the same instant. For the error variance, the statistical variable that aims to demonstrate how much the samples vary around their mean value is demonstrated by

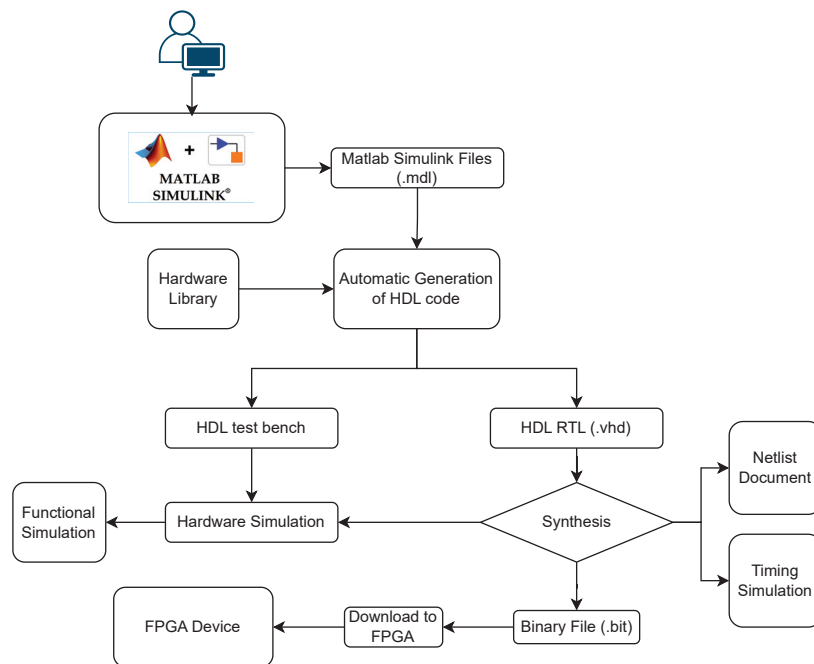
$$\sigma^2(M_{se}) = \frac{1}{N} \sum_{i=0}^{N-1} (M_{se}(i) - \overline{M_{se}})^2, \quad (30)$$

where  $\sigma^2(M_{se})$  is the mean error variance,  $M_{se}(i)$  is the value of the  $i$ -th collected sample, and  $\overline{M}_{se}$  is the mean of the analyzed variable.

Comparative analyses were performed, among the results obtained in each of the methods presented for each of the exposed scenarios. At the end of the tests and simulation results, an analysis of occupation and use of the platform is performed, highlighting points of use, such as Look up Tables (LUTs), memories, registers, and multipliers, in addition to revealing the maximum frequencies obtained in each of the implementations.

In order to perform the real-time tests, it was necessary to divide the clock to a value less than 1 ms (1 KHz), in order to work with the system in time, since the platform works with an internal clock of approximately 200 MHz (5 ns). Otherwise this would result in an acceleration for the current working system (vehicle).

Figure 19 illustrates the design workflow for FPGA project development using MATLAB Simulink. This design utilizes hardware libraries and automatic HDL code generation. The process begins with the creation of Simulink files (.mdl), in which the desired system is modeled using blocks from a specialized hardware library. From this model, the automatic generation of HDL code is performed. This HDL code is divided into two paths: one for the creation of an HDL test bench, which allows functional simulation, and the other for the generation of RTL code in HDL (.vhd). In the HDL test bench block, a thorough functional simulation analysis is performed, providing reassurance about the integrity of the generated code. On the other hand, the result obtained from the HDL RTL block is sent to the synthesis flow, where the feasibility of synthesis is determined (indicated as 'Synthesis'). In this synthesis flow, documents such as the Netlist are generated, describing the usage and interconnection of components, and the Timing Simulation is performed to validate the temporal performance of the project. If the synthesis is successful, a binary file (.bit) is generated, which can be transferred to the FPGA, allowing implementation on the hardware device. After downloading the .bit file to the FPGA, the device can be tested directly in the physical environment, where the input and output signals can be monitored to ensure that the FPGA's behavior aligns with the original project expectations. Real-time tests can be conducted using debugging and signal analysis tools, such as oscilloscopes and logic analyzers, to ensure that the system functions correctly under real operating conditions.



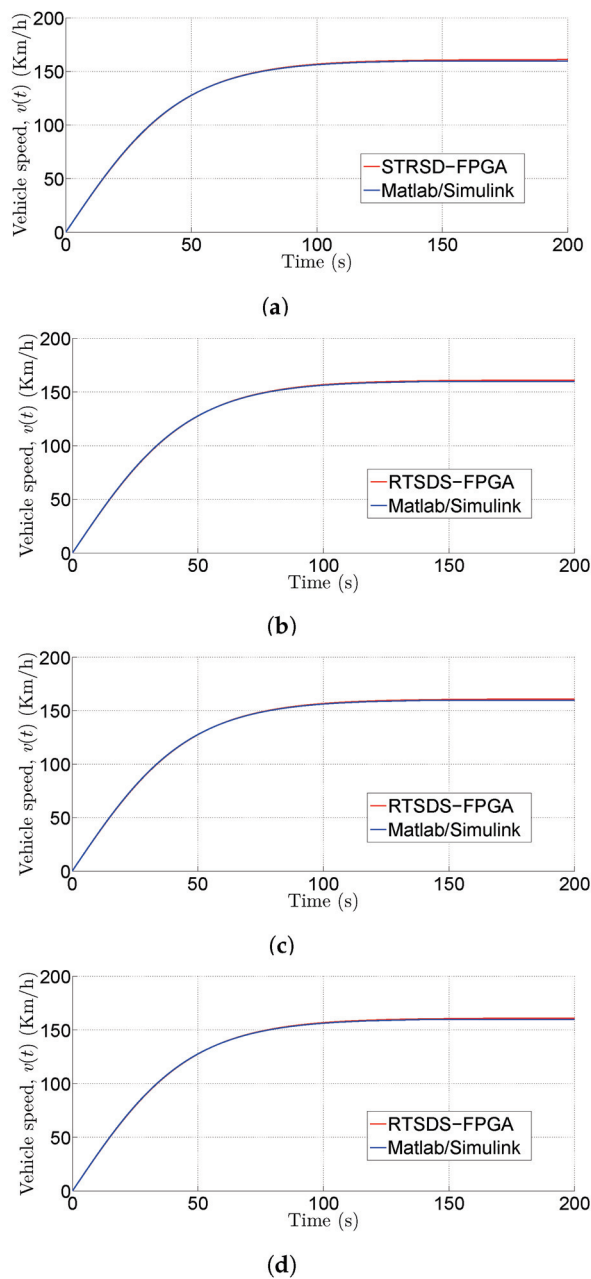
**Figure 19.** Design workflow for FPGA project development using MATLAB Simulink.

#### 4.4. Simulations in Matlab

##### 4.4.1. Scenario 1

In this scenario, the system inputs were kept constant, maintaining the torque at 500 Nm and the tilt angle at 1 degree. The ODE values were determined for each of the solution methods.

Using an implementation in which all the solution methods were loaded and executed in parallel, results were obtained for the Euler method (Figure 20a), the improved Euler method (Figure 20b), the third-order Runge–Kutta method (Figure 20c), and the fourth-order Runge–Kutta method (Figure 20d). The images show the results for the simulations performed using Simulink (floating point) and System Generator (fixed point). It can be seen from the figures that the same behavior was obtained for both solution methods.



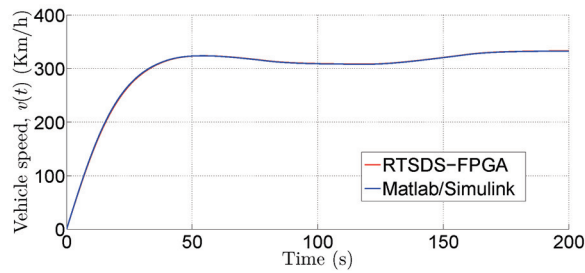
**Figure 20.** Result obtained in simulations for scenario 1. The signal represents the vehicle's speed as a function of time,  $v(t)$ , and also corresponds to the differential equation presented in Equation (17).

(a) Result obtained in the simulations with the Euler method. (b) Result obtained in the simulations with the improved Euler method. (c) Result obtained in the simulations with the third-order Runge–Kutta method. (d) Result obtained in the simulations with the fourth-order Runge–Kutta method.

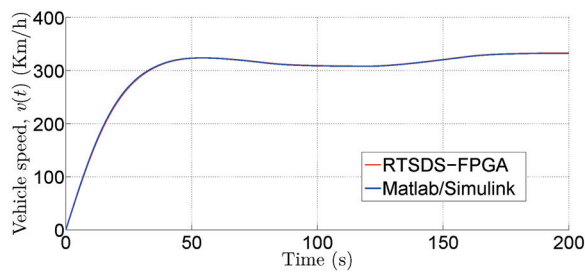
#### 4.4.2. Scenario 2

In this scenario, the values of inputs 1 and 2 were altered over time, with the same behavior shown previously in Figures 17 and 18.

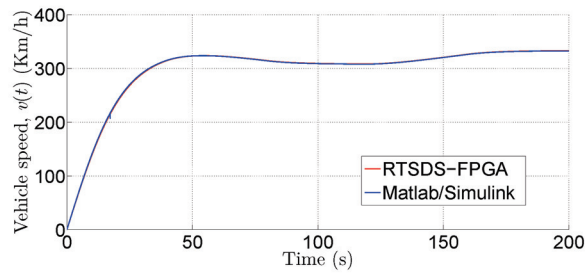
Results were obtained for the Euler method (Figure 21a), the improved Euler method (Figure 21b), the third-order Runge–Kutta method (Figure 21c), and the fourth-order Runge–Kutta method (Figure 21d).



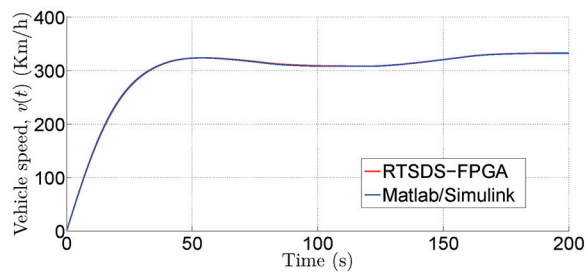
(a)



(b)



(c)



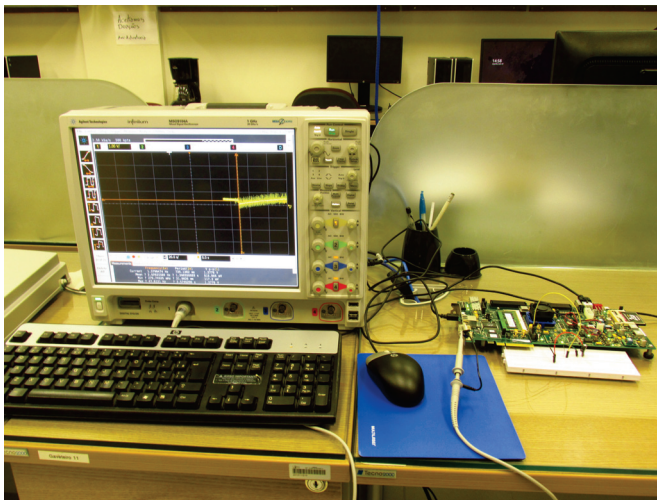
(d)

**Figure 21.** Result obtained in simulations for scenario 2. The signal represents the vehicle's speed as a function of time,  $v(t)$ , and also corresponds to the differential equation presented in Equation (17).

(a) Result obtained in simulations with the Euler method. (b) Result obtained in simulations with the Improved Euler method. (c) Result obtained in simulations with the third-order Runge–Kutta method. (d) Result obtained in simulations with the fourth-order Runge–Kutta method.

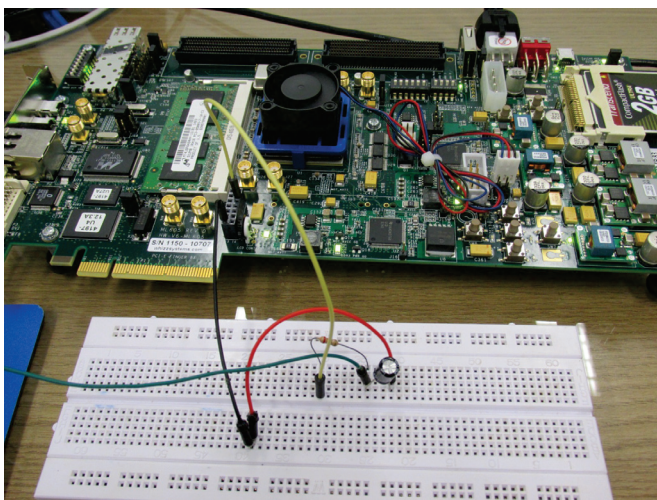
#### 4.5. Simulations in RTSDS-FPGA

For the second set of results, simulations were performed with the embedded RTSDS-FPGA. For this implementation, bench tests were performed using the equipment shown in Figure 22, which included the ML605 kit, an MSO9104A oscilloscope (Agilent Instruments) for observing the data, and an RC circuit to extract the analog data from the PWM signal generated by the platform. The analog values obtained by the oscilloscope (Figure 22) were processed for better viewing by reducing the number of signals sampled.



**Figure 22.** Test bench (Oscilloscope, ML605 and RC circuit).

Figure 23 details the circuit used as a DAC, consisting of a digital-to-analog converter integrated with a low-pass filter. This circuit is responsible for converting the digital signals generated by the FPGA into analog signals suitable for analysis and measurement. The DAC setup includes resistors and capacitors selected to ensure appropriate response and minimize noise, allowing the analog signals to reflect the simulated values accurately. This circuit was essential for obtaining the output data analyzed and presented in the results.



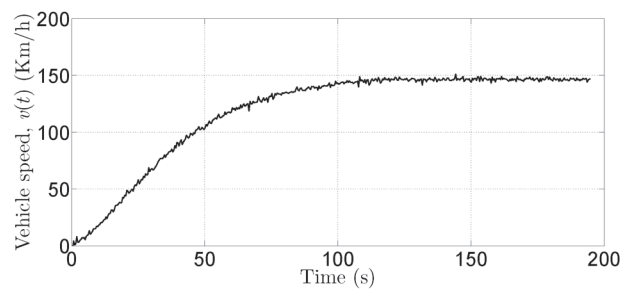
**Figure 23.** Circuit configuration for the digital-to-analog converter (DAC) used in the RTSDS-FPGA platform.



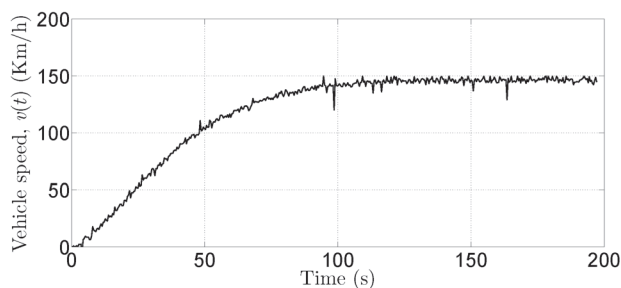
The observed noise in the output signal during the practical implementation of the system is primarily attributed to the digital-to-analog converter (DAC) used in the setup. The DAC was implemented using an RC circuit in conjunction with a PWM signal generated by the simulator circuit within the FPGA. While this method is functional, it inherently introduces noise due to the limitations of PWM and the filtering provided by the RC circuit. This noise can affect the fidelity of the output signal, particularly in applications that require precise signal differentiation. However, it is essential to note that the internal processing within the FPGA remains highly accurate, as demonstrated by the low error rates in Tables 2 and 3. The noise issue is primarily tied to the quality of the DAC, and using a higher-quality DAC could significantly mitigate this problem, ensuring that the high precision of the internal simulation is reflected more accurately in the output signal.

#### 4.5.1. Scenario 1

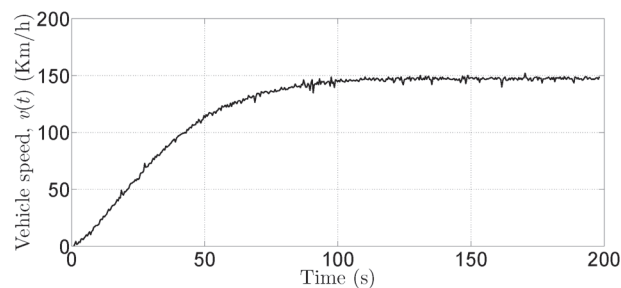
For scenario 1, the simulation result on the platform is shown in Figure 24a for the Euler method, Figure 24b for the Improved Euler method, Figure 24c for the third-order Runge–Kutta method, and Figure 24d for the fourth-order Runge–Kutta method. The results demonstrate that the platform was able to simulate the expected behavior, as well as the behavior produced in software.



(a)

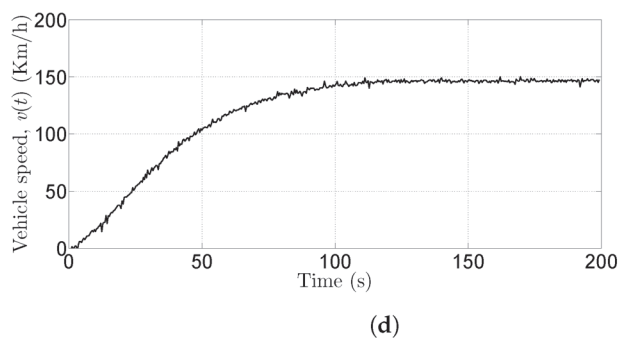


(b)



(c)

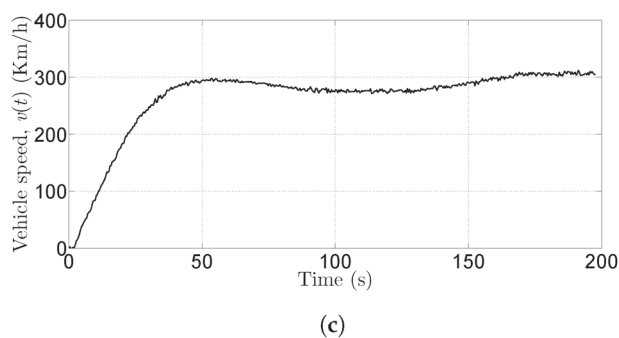
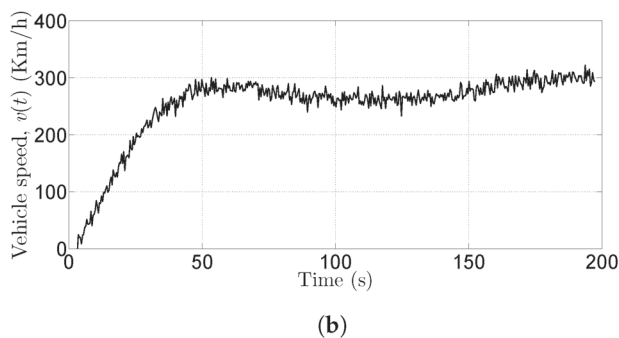
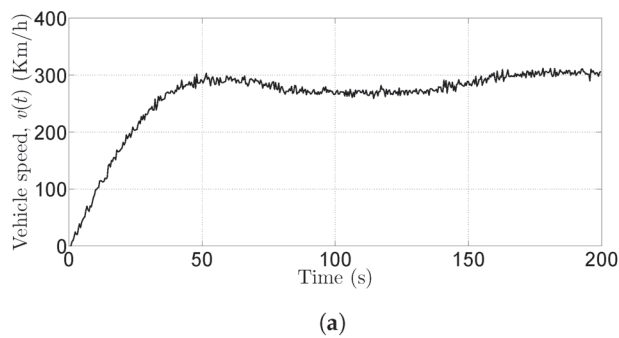
Figure 24. Cont.



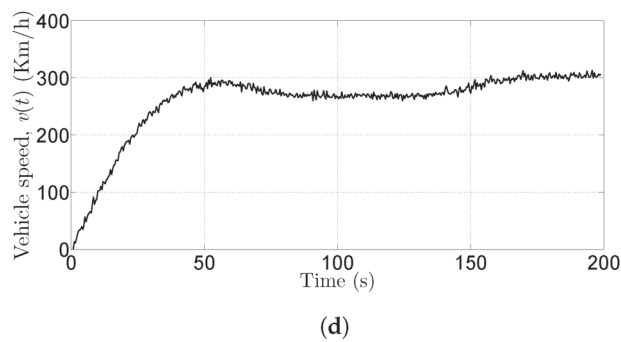
**Figure 24.** Result obtained in the RTSDS-FPGA for scenario 1. The signal represents the vehicle's speed as a function of time,  $v(t)$ , and also corresponds to the differential equation presented in Equation (17). (a) Result obtained in the RTSDS-FPGA with the Euler method. (b) Result obtained in the RTSDS-FPGA with the improved Euler method. (c) Result obtained in the RTSDS-FPGA with the third-order Runge–Kutta method. (d) Result obtained in the RTSDS-FPGA with the fourth-order Runge–Kutta method.

#### 4.5.2. Scenario 2

For scenario 2, the simulation result on the platform is shown in Figure 25a for the Euler method, Figure 25b for the improved Euler method, Figure 25c for the third-order Runge–Kutta method, and Figure 25d for the fourth-order Runge–Kutta method.



**Figure 25.** Cont.



**Figure 25.** Result obtained in the RTSDS-FPGA for scenario 2. The signal represents the vehicle's speed as a function of time,  $v(t)$ , and also corresponds to the differential equation presented in Equation (17). (a) Result obtained in the RTSDS-FPGA with the Euler method. (b) Result obtained in the RTSDS-FPGA with the improved Euler method. (c) Result obtained in the RTSDS-FPGA with the third-order Runge–Kutta method. (d) Result obtained in the RTSDS-FPGA with the fourth-order Runge–Kutta method.

Tables 2 and 3 explain the mean error, mean square error, and variance values for each of the scenarios, based on the values obtained in Simulink (64-bit floating point). With the analysis, one can see that the results were satisfactory, with error values smaller than one unit and with low variances around the mean, in the order of the sixth decimal place.

**Table 2.** Data analyses for scenario 1.

Method	Mean Error	Mean Square Error	Error Variance
Euler	0.0477	0.4531	$6.553 \times 10^{-7}$
Improved Euler	−0.1442	0.7013	$1.8618 \times 10^{-6}$
Third-order Runge–Kutta	−0.1472	0.5983	$2.8418 \times 10^{-7}$
Fourth-order Runge–Kutta	−0.4462	0.4435	$6.3756 \times 10^{-6}$

**Table 3.** Data analyses for scenario 2.

Method	Mean Error	Mean Square Error	Error Variance
Euler	0.0473	0.4521	$6.5167 \times 10^{-7}$
Improved Euler	0.0477	0.4531	$6.553 \times 10^{-7}$
Third-order Runge–Kutta	−0.1442	0.7013	$1.8618 \times 10^{-6}$
Fourth-order Runge–Kutta	−0.1472	0.5983	$2.8418 \times 10^{-7}$

#### 4.6. Netlist Analysis

In the approach of scenario 1, the shortest possible implementation time was 218.283 ns (maximum frequency of 4.581 MHz). For scenario 2, the shortest time was 196.622 ns (maximum frequency of 5.086 MHz), which was the shortest time among all the implementations. These values contrasted to those reported in the literature, where the time periods were in the region of microseconds.

The time periods found for the system in question were more than sufficient for performing all the computations of the simulated model. However, the simulation could be accelerated, since there was a delay between the time for computing the next value and the data deadline. This became much clearer with analysis of the worst-case slack value, which represents the period of time that the system remains idle. Tables 4 and 5 provide the values obtained for the implementations in each of the scenarios. It can be seen that there was a large excess of time for implementations with click time of 1 ms. The negative values in Tables 4 and 5 indicate that the system was unable to achieve the set time, informing the time still required. This was precisely the inverse of the period described as the idle time.

**Table 4.** Analysis of the time slack in scenario 1.

Value of the Clock	Time Slack
5 ns	−3.747 ns
1 ms	999,791.419 ns

**Table 5.** Analysis of the time slack in scenario 2.

Value of the Clock	Time Slack
5 ns	−3.977 ns
1 ms	999,805.378 ns

A study of occupation was also undertaken (Netlist). Tables 6 and 7 show the values for the use of LUTs, registers, memories, and multipliers, obtained by analysis of the reports generated by the ISE (the development environment used by Xilinx) for each scenario. The amount of area occupied by the entire system was approximately 53%, taking into account the multiple system replies required to implement the various ODE solution methods. This area could be reduced if time was invested to research simpler solutions (for example, using only one solution method) or to avoid creating multiple replies. Nevertheless, the fully parallel implementation was an important differentiating feature of the proposed method. It is useful at this point to provide a summary of the results.

**Table 6.** Platform Netlist analysis for scenario 1.

Item	Using	Percentage Relative to Absolute Value
Number of Slices Registers	1838	1%
Number of Slices LUTS	82,964	55%
Number of Memories	0	0%
Number of Multipliers	120	15%

**Table 7.** Platform Netlist analysis for scenario 2.

Item	Using	Percentage Relative to Absolute Value
Number of Slices Registers	1958	1%
Number of Slices LUTS	80,604	53%
Number of Memories	144	34%
Number of Multipliers	120	15%

The first observation that can be made concerns the first point set out at the start of the paper; execution of the simulation in real time was achieved. This is clear from the graphs presented in Sections 4.4.1 and 4.4.2, where it can be seen that the behavior of the system implemented using the RTSDS-FPGA was in agreement with the validation data obtained using Matlab/Simulink and Matlab/System Generator. The graphs show that the signal increases and decreases were executed in very similar time periods.

Comparison of the Matlab/Simulink and Matlab/System Generator methods showed that the results obtained were satisfactory, as expected, with good statistical parameter values, maximum errors of one unit, and variances of around six decimal places.

The second point to consider concerns the ability to execute tasks in shorter periods of time than those required for real-time evaluations, reducing the total simulation time to even lower levels, for example, enabling training and testing of control algorithms in shorter times. This would have positive impacts in industrial processes that have severe time restrictions. The platform could be executed in times shorter than required for such processes, freeing the system to perform other tasks in the remaining time.

The use of the system with hardware enables it to be employed in education and learning environments, where the hardware could be used to perform tests in the area of

control. For example, students could undertake development tasks such as controlling the speed of a vehicle at a set value.

The proposed methodology offers significant potential as a teaching tool in undergraduate and graduate courses, particularly in electrical engineering, control systems, and computer engineering. For undergraduate courses, the method can provide students with hands-on experience in real-time simulation and FPGA programming. For instance, students could use the platform to simulate basic electrical circuits or control systems, allowing them to visualize and understand complex theoretical concepts practically and interactively. This approach would enhance their comprehension of course material and equip them with valuable skills in hardware description languages (HDLs) and real-time systems, which are increasingly important in the industry. The methodology could be employed at the graduate level for more advanced applications, such as the design and simulation of complex systems like power converters, robotic controllers, or signal processing algorithms. Graduate students could use the platform to conduct research projects that involve modeling and simulating sophisticated systems, offering a low-cost and flexible alternative to commercial simulation tools. Furthermore, the platform's modularity allows for the easy integration of custom components, enabling students to experiment with new ideas and develop innovative solutions to engineering problems. This hands-on approach to learning and research would deepen their understanding of theoretical concepts and foster critical thinking and problem-solving skills, preparing them for careers in academia or industry.

## 5. Other Examples of Dynamic Systems

As described in Section 3, the proposed methodology allows for the simulation of generic systems, not limited to the specific example presented in Section 4.1. To further clarify the use of the RTSDES-FPGA, this section will detail its application in two other dynamic systems: one associated with a flow system in a tank and another with a series RL electrical circuit.

### 5.1. Flow System in a Tank

This dynamic system represents a water tank connected to a pump, as presented in [40]. The pump, operating at a constant flow rate  $q_m$ , feeds into an inlet valve controlled continuously by  $V_e(t)$ , which in turn determines the input flow  $q_e(t)$  into the tank. The outlet valve, also with continuous control  $V_o(t)$ , governs the outflow  $q_s(t)$  from the tank. The water level in the tank is represented by the variable  $n(t)$ . The system has two inputs—the inlet valve control  $V_e(t)$  and the outlet valve control  $V_o(t)$ —and one output, the water level  $n(t)$  in the tank. The system is modeled by the following differential equation:

$$A \frac{dn(t)}{dt} = q_e(t) - V_o(t)q_s(t) \quad (31)$$

where

$$q_e(t) = q_m V_e(t) \quad (32)$$

and

$$q_s(t) = a \sqrt{2gn(t)}, \quad (33)$$

In these equations,  $q_m$ ,  $A$ ,  $g$ , and  $a$  represent the pump flow rate ( $\text{m}^3/\text{s}$ ), the cross-sectional area of the tank ( $\text{m}^2$ ), gravitational acceleration ( $\text{m}/\text{s}^2$ ), and the cross-sectional area of the tank's outlet tube ( $\text{m}^2$ ), respectively [40,41].

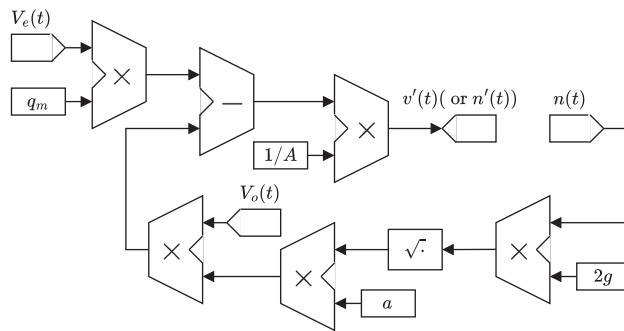
Equation (31) can be expressed by

$$n(t) = \begin{cases} u(t) & \text{para } u(t) < n_{\max} \\ n_{\max} & \text{para } u(t) \geq n_{\max} \end{cases} \quad (34)$$

where  $n_{max}$  is the maximum water level of the tank, and  $u(t)$  is given by Equation (26), where  $v_0$  is the initial condition of the water level of the tank, and Equation (27) can be expressed as

$$h_{vs}(s, v(s)) = \frac{1}{A} \left( q_m V_e(t) - V_o(t) a \sqrt{2gn(t)} \right). \quad (35)$$

As described in Section 3, the proposed methodology is designed to simulate generic systems and is not limited to the specific example presented in Section 4. To clarify the use of RTSDS-FPGA, the simulation of the tank system in hardware is similar to the example discussed in Section 4. The primary modification required is to change the DSM hardware, which will now be replaced by the configuration presented in Figure 26. This further emphasizes the generic nature of the proposed methodology, as outlined in Section 3 and illustrated in Figure 1, demonstrating its adaptability to various dynamic systems.



**Figure 26.** Hardware configuration for tank system simulation using RTSDS-FPGA.

## 5.2. Series RL Electrical Circuit

This dynamic system represents a series RL circuit [40,41]. The circuit consists of a resistor with resistance  $R$  and an inductor with inductance  $L$ , connected in series to a voltage source  $V(t)$ . The current  $i(t)$  flowing through the circuit is the main variable of interest. The system has one input, which is the applied voltage  $V(t)$ , and one output, which is the current  $i(t)$ . The behavior of the series RL circuit is governed by the following differential equation:

$$L \frac{di(t)}{dt} = V(t) - Ri(t) \quad (36)$$

where  $R$  represents the resistance ( $\Omega$ ) and  $L$  represents the inductance (H) [40,41]. The applied voltage  $V(t)$  can be expressed by

$$V(t) = V_m V_e(t) \quad (37)$$

where  $V_m$  is the maximum applied voltage in the circuit.

Equation (36) can be expressed by

$$n(t) = \begin{cases} u(t) & \text{para } u(t) < i_{max} \\ i_{max} & \text{para } u(t) \geq i_{max} \end{cases}, \quad (38)$$

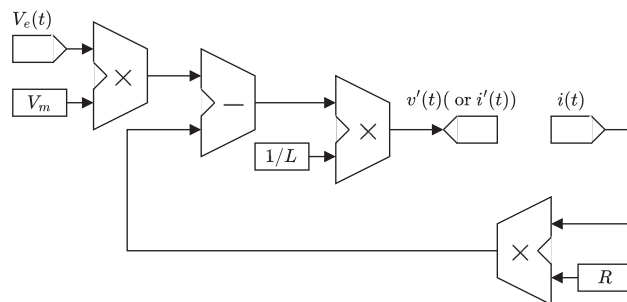
where  $i_{max}$  is the maximum current value, and  $u(t)$  is given by Equation (26), where  $v_0$  is the initial condition of the current value, and Equation (38) can be expressed as

$$h_{vs}(s, v(s)) = \frac{1}{L} (V_m V_e(t) - Ri(t)). \quad (39)$$

Similar to the approach taken in Section 5.1, the simulation of the series RL circuit in hardware follows the example provided in Section 4. The key adjustment involves replacing the DSM's hardware configuration with the setup depicted in Figure 27. This change aligns with the proposed methodology's generic framework, as discussed in Section 3 and



illustrated in Figure 1, further demonstrating its flexibility and applicability to a wide range of dynamic systems, including electrical circuits like the series RL system.



**Figure 27.** Hardware configuration for series RL circuit simulation using RTSDS-FPGA.

## 6. Conclusions

The work in this article proposes an embedded solution in FPGA, for real auditory simulations for dynamic systems, here called RTSDS-FPGA. A pallet comprised a main control module (MCM) and a set of auxiliary hardware modules, called DAM and SGM. A prototype was developed for a model of a longitudinal vehicle, unable to perform real-time simulations in various scenarios. The prototype was implemented using Matlab/System Generator, with the kit Virtex-6 (XC6VLX240T) ML605, as MCM. The results obtained for all scenarios explored showed that the RTSDS-FPGA can be used as a tool for the development of real-time control systems. With the developed prototype, the RTSDS-FPGA performed well, since, for a period of 1 ms, its actions were executed at approximately 210 ns, remaining idle the rest of the time. With this analysis, it is possible to observe the simulation of the system in real time and to execute simulations with models more complex than those in the literature. On the other hand, it can be seen that the RTSDS-FPGA can also be used as a tool to accelerate simulations of systems with slow dynamics. The RTSDS-FPGA was developed with four differential equation resolution methods (Euler, improved Euler, and third- and fourth-order Runge–Kutta methods), which operate in a completely parallel way, using dynamic system reply and allowing a choice of methods, which distinguishes this implementation from presentations in the literature. Precisely for this reason, one approach utilized approximately 53% of the total structures of the development platform. Last but not least, one can make use of RTSDS-FPGA in the academic field, as a teaching tool for research and teaching, both for undergraduate and graduate levels. The primary limitation of the proposed methodology lies in the need for a high degree of customization for very specific or non-standard use cases, even with the modular design that covers a wide range of applications. This can increase development time and complexity. Therefore, while the methodology is robust for many real-time simulation tasks, it may require additional refinements to meet the specific requirements of advanced applications.

**Author Contributions:** All authors have contributed in various degrees to ensure the quality of this work (e.g., S.N.S. and M.A.C.F. conceived the idea and experiments; S.N.S. and M.A.C.F. designed and performed the experiments; S.N.S., M.A.S.d.S.G., L.M.D.d.S. and M.A.C.F. analyzed the data; S.N.S., M.A.S.d.S.G., L.M.D.d.S. and M.A.C.F. wrote the paper; M.A.C.F. coordinated the project). All authors have read and agreed to the published version of the manuscript.

**Funding:** Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES)-Finance Code 001.

**Data Availability Statement:** No new data were created or analyzed in this study. Data sharing is not applicable to this article.

**Acknowledgments:** The authors would like to express their gratitude to the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for providing financial support.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

- de Souza, D.T.; Silva, S.N.; Teles, R.; Fernandes, M.A.C. Platform for Real-Time Simulation of Dynamic Systems and Hardware-in-the-Loop for Control Algorithms. *Sensors* **2014**, *14*, 19176–19199. [CrossRef] [PubMed]
- Boukerche, A.; Lu, K. A novel approach to real-time RTI based distributed simulation system. In Proceedings of the 38th Annual Simulation Symposium, San Diego, CA, USA, 4–6 April 2005; pp. 267–274. [CrossRef]
- You, T.; Zhu, Y.A.; Zhang, D.P. Applied Research of Delaminated Real-Time Network Framework Based on RTX in Simulation. In Proceedings of the 2009 Second International Conference on Information and Computing Science, Manchester, UK, 21–22 May 2009; Volume 1, pp. 389–392. [CrossRef]
- Handong, B.; Zhiguo, Z.; Zhiwen, L. Design and Implementation of an FPGA-based Real Time Simulation System for Photovoltaic Power Generation. In Proceedings of the 2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), Beijing, China, 31 August–3 September 2014; pp. 1–5. [CrossRef]
- Dufour, C.; Cense, S.; Yamada, T.; Imamura, R.; Bélanger, J. FPGA permanent magnet synchronous motor floating-point models with variable-DQ and spatial harmonic Finite-Element Analysis solvers. In Proceedings of the 2012 15th International Power Electronics and Motion Control Conference (EPE/PEMC), Novi Sad, Serbia, 4–6 September 2012; pp. LS6b.2-1–LS6b.2-10. [CrossRef]
- Matar, M.; Iravani, R. FPGA Implementation of the Power Electronic Converter Model for Real-Time Simulation of Electromagnetic Transients. *IEEE Trans. Power Deliv.* **2010**, *25*, 852–860. [CrossRef]
- Monga, M.; Karkke, M.; Tondehal, L.K.; Steward, B.; Kelkar, A.; Zambreno, J. Real-Time Simulation of Dynamic Vehicle Models using a High-performance Reconfigurable Platform. In Proceedings of the International Conference on Computational Science (ICCS) 2012, Omaha, NE, USA, 4–6 June 2012.
- Hernández, F.A.I.; Canesin, C.A. Electrical Power Distribution System modeling with VHDL-AMS for the construction of a Real-Time Digital Simulator using FPGAS devices. In Proceedings of the 201210th IEEE/IAS International Conference on Industry Applications (INDUSCON), Fortaleza, Brazil, 5–7 November 2012; pp. 1–7.
- Liu, C.; Guo, X.; Gao, F.; Breaz, E.; Damien, P.; Gechter, F. FPGA based real-time simulation of high frequency soft-switching circuit using time-domain analysis. In Proceedings of the IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, Italy, 23–26 October 2016; pp. 5832–5837.
- Morello, R.; Baronti, F.; Tian, X.; Chau, T.; Di Rienzo, R.; Roncella, R.; Jeppesen, B.; Lin, W.; Ikushima, T.; Saletti, R. Hardware-in-the-loop simulation of FPGA-based state estimators for electric vehicle batteries. In Proceedings of the 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, USA, 8–10 June 2016; pp. 280–285.
- Guo, H.; Su, L.; Wang, Y.; Long, Z. FPGA-accelerated molecular dynamics simulations system. In Proceedings of the 2009. SCALCOM-EMBEDDED COM'09. International Conference on Scalable Computing and Communications, Eighth International Conference on Embedded Computing, Dalian, China, 25–27 September 2009; pp. 360–365.
- Dixit, V.; Patil, M.; Chandorkar, M. Real time simulation of power electronic systems on multi-core processors. In Proceedings of the 2009 International Conference on Power Electronics and Drive Systems PEDS, Taipei, Taiwan, 2–5 November 2009; pp. 1524–1529. [CrossRef]
- Menghal, P.M.; Laxmi, A. Real time simulation: A novel approach in engineering education. In Proceedings of the 2011 3rd International Conference on Electronics Computer Technology (ICECT), Kanyakumari, India, 8–10 April 2011; Volume 1, pp. 215–219. [CrossRef]
- Dufour, C.; Andrade, C.; Bélanger, J. Real-Time Simulation Technologies in Education: A Link to Modern Engineering Methods and Practices. In Proceedings of the 2010 11th International Conference on Engineering and Technology Education (INTERTECH), Ilhéus, Brazil, 7–10 March 2010; pp. 1–5.
- Vityaz, O.; Zimmermann, G. Real-time simulation using graceful degradation of accuracy. *Energy Build.* **2005**, *37*, 795–806. [CrossRef]
- Anagnostopoulos, D. A methodological approach for model validation in faster than real-time simulation. *Simul. Model. Pract. Theory* **2002**, *10*, 121–139. [CrossRef]
- Laplante, P.A.; Ovaska, S.J. *Real-Time Systems Design and Analysis: Tools for the Practitioner*, 4th ed.; Wiley: Hoboken, NJ, USA, 2012.
- Dai, X.; Ke, C.; Quan, Q.; Cai, K.Y. Simulation credibility assessment methodology with FPGA-based hardware-in-the-loop platform. *IEEE Trans. Ind. Electron.* **2020**, *68*, 3282–3291. [CrossRef]
- Rezaei Larijani, M.; Zolghadri, M.R. Design and implementation of an ADC-based real-time simulator along with an optimal selection of the switch model parameters. *Electr. Eng.* **2021**, *103*, 2315–2325. [CrossRef]
- Queiroz, J.; Carvalho, S.; Barros, C.; Barros, L.; Barbosa, D. Embedding an electrical system real-time simulator with floating-point arithmetic in a field programmable gate array. *Energies* **2021**, *14*, 8404. [CrossRef]
- Mesgena, D.W. Future Operation and Control of Power Systems-Laboratory Models and Real-Time Simulation. Master's Thesis, Norwegian University of Science and Technology NTNU, Trondheim, Norway, 2022.
- Lamo, P.; De Castro, A.; Sanchez, A.; Ruiz, G.A.; Azcondo, F.J.; Pigazo, A. Hardware-in-the-loop and digital control techniques applied to single-phase PFC converters. *Electronics* **2021**, *10*, 1563. [CrossRef]

23. Sarac, M.; Aydoğmuş, Ö. Real-Time Digital Simulator Design for Differential Drive Mobile Robot using FPGA. *Balk. J. Electr. Comput. Eng.* **2024**, *12*, 84–89. [CrossRef]
24. Galkin, N.; Yang, C.W.; Vyatkin, V. Towards the Automatic Transformation of the SIMULINK Model into an FPGA-in-Loop System and its Real-Time Simulation. *IEEE Access* **2024**. [CrossRef]
25. Bieber, L.; Wang, L.; Jatskevich, J.; Li, W. Universal Equivalent Model for Real-Time CPU/FPGA Co-Simulation of Hybrid Cascaded Multilevel Converters. *IEEE Access* **2023**, *11*, 4228–4241. [CrossRef]
26. Arduino-Mega. Arduino-Arduino Mega2560. 2024. Available online: <http://arduino.cc/en/Main/ArduinoBoardMega2560> (accessed on 12 September 2024).
27. ATmega328p. Atmel-AVR-ATmega328p. 2024. Available online: <https://www.microchip.com/en-us/product/ATMEGA328P> (accessed on 12 September 2024).
28. Arduino-Uno. Arduino-ArduinoUno. 2024. Available online: <http://arduino.cc/en/Main/ArduinoBoardUno> (accessed on 12 September 2024).
29. Cypress-PSoC3-CY8C38. Cypress-PSoC 3: CY8C38 Family Data Sheet. 2016. Available online: <http://www.cypress.com/?rID=35178> (accessed on 11 September 2016).
30. CY8CKIT-001, C. Cypress-CY8CKIT-001 PSoC Development Kit. 2016. Available online: <http://www.cypress.com/?rID=37464> (accessed on 11 September 2016).
31. Matar, M.; Karimi, H.; Etemadi, A.; Iravani, R. A high performance real-time simulator for controllers hardware-in-the-loop testing. *Energies* **2012**, *5*, 1713–1733. [CrossRef]
32. RTDS, RSCAD.v5. Available online: <https://www.rtds.com> (accessed on 12 September 2024).
33. Ibarra, L.; Rosales, A.; Ponce, P.; Molina, A.; Ayyanar, R. Overview of Real-Time Simulation as a Supporting Effort to Smart-Grid Attainment. *Energies* **2017**, *10*, 817. [CrossRef]
34. Hernandez, J.; Rangel-Magdaleno, J.d.J.; Morales-Caporal, R. A High-Performance and Cost-Effective Field Programmable Gate Array-Based Motor Drive Emulator. *Micromachines* **2023**, *14*, 1864. [CrossRef] [PubMed]
35. Buraimoh, E.; Ozkan, G.; Timilsina, L.; Chamathi, P.K.; Papari, B.; Edrington, C.S. Overview of interface algorithms, interface signals, communication and delay in real-time co-simulation of distributed power systems. *IEEE Access* **2023**, *11*, 103925–103955. [CrossRef]
36. Parkinson, D.Y.; Beattie, K.; Chen, X.; Correa, J.; Dart, E.; Daurer, B.J.; Deslippe, J.R.; Hexemer, A.; Krishnan, H.; MacDowell, A.A.; et al. Real-time data-intensive computing. *AIP Conf. Proc.* **2016**, *1741*, 050001. [CrossRef]
37. Gilchrist, A. Introducing Industry 4.0. In *Industry 4.0: The Industrial Internet of Things*; Apress: Berkeley, CA, USA, 2016; pp. 195–215. [CrossRef]
38. Butcher, J. *Numerical Methods for Ordinary Differential Equations*; Wiley: Hoboken, NJ, USA, 2008.
39. Pacejka, H.B. *Tire and Vehicle Dynamics*; Elsevier: Amsterdam, The Netherlands, 2002.
40. Silva, S.N.; Torquato, M.F.; Fernandes, M.A. Comparison of binary and fuzzy logic in feedback control of dynamic systems. *Int. J. Dyn. Control.* **2019**, *7*, 1056–1064. [CrossRef]
41. Ogata, K. *Modern Control Engineering*, 4th ed.; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2001.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# An Energy-Efficient Field-Programmable Gate Array (FPGA) Implementation of a Real-Time Perspective-n-Point Solver

Haobo Lv and Qiongzhi Wu \*

School of Integrated Circuit and Electronics, Beijing Institute of Technology, Beijing 100081, China;  
bobl@bit.edu.cn

\* Correspondence: wqz\_bit@bit.edu.cn

**Abstract:** Solving the Perspective-n-Point (PnP) problem is difficult in low-power systems due to the high computing workload. To handle this challenge, we present an originally designed FPGA implementation of a PnP solver based on Vivado HLS. A matrix operation library and a matrix decomposition library based on QR decomposition have been developed, upon which the EPnP algorithm has been implemented. To enhance the operational speed of the system, we employed pipeline optimization techniques and adjusted the computational process to shorten the calculation time. The experimental results show that when the number of input data points is 300, the proposed system achieves a processing speed of 45.2 fps with a power consumption of 1.7 W and reaches a peak-signal-to-noise ratio of over 70 dB. Our system consumes only 3.9% of the power consumption per calculation compared to desktop-level processors. The proposed system significantly reduces the power consumption required for the PnP solution and is suitable for application in low-power systems.

**Keywords:** FPGA; hardware implementation; high-level synthesis (HLS); Perspective-n-Point (PnP); QR decomposition

## 1. Introduction

The Perspective-n-Point (PnP) problem is the problem of determining the pose of a calibrated camera from  $n$  correspondences between 3D reference points and their 2D projections [1]. It is widely used in visual SLAM [2,3], robotics [4], and Augmented Reality [5]. Typically, the PnP algorithm is implemented by the CPU. However, in embedded systems, the processing speed of the processor is relatively slow, and the high computational complexity of the PnP algorithm leads to the inability of real-time calculations. Although PC processors bring considerable computational speed, their high-power consumption makes it difficult to apply them in low-power systems, such as battery-powered robots and VR glasses.

To reduce power consumption and latency, hardware acceleration is often considered a feasible method of algorithmic acceleration. It provides faster computing speed while using little power. Despite the potential benefits of hardware acceleration, currently, there is no published hardware implementation of the PnP algorithm. The lack of a hardware implementation is primarily attributed to algorithm complexity.

The difficulties of hardware design for PnP solvers mainly stem from the following three aspects. Firstly, during the calculation process, the data exhibit strong dependencies, and some data are used multiple times, making pipelining difficult to achieve and increasing the difficulty of data management.

Secondly, the diversity of input data leads to a wide dynamic range of numbers during the calculation process, posing a significant challenge in maintaining reasonable hardware costs while ensuring calculation accuracy by using fixed-point numbers. It is also difficult to use floating-point numbers in hardware design due to poor support offered by hardware description language.

Thirdly, during the computation process, various math functions are required, such as the computation of square roots and the solution for matrix eigenvalues, which cannot be simply computed by calling libraries as easily as in software. For this reason, most current PnP-solving processes are based on processors.

In this paper, we propose an FPGA accelerator for solving the PnP problem, which enables real-time computational performance with high energy efficiency. To overcome the challenges of implementing the PnP solving algorithm on hardware, we utilize high-level synthesis (HLS) languages to ease the developing process. HLS languages are capable of translating high-level languages such as C/C++ into register transfer level (RTL) languages [6,7], allowing hardware implementation to be directly realized with high-level languages, which simplifies the process of hardware implementation design. The results obtained show that the proposed system achieves real-time processing speed, while its energy consumption is significantly lower than that of systems based on processors.

The contributions of this paper can be summarized as follows.

- (1) Solving the PnP problem is useful in many applications, but so far there is no hardware solver due to complexity. To the best of our knowledge, the proposed accelerator in this paper makes the first attempt to implement a hardware PnP solver on FPGA.
- (2) To achieve a complete PnP solution, we developed a functional module for solving matrix eigenvalues, eigenvectors, and linear least squares equations using QR decomposition. The module can be used as a reference for other designs.
- (3) We carefully fine-tuned the system to achieve maximum performance. To compare the performance of the proposed system with others, experiments were carried out on different platforms. The results indicate that the proposed system exhibits superior performance in terms of energy efficiency.

The rest of the paper is organized as follows: Section 2 introduces related work on solving the PnP problem. Section 3 describes the hardware architecture of the proposed system. Section 4 describes the obtained optimization methods. Section 5 shows the experimental results of the system. Section 6 summarizes the paper.

## 2. Related Work

Efforts have been made to optimize the speed and accuracy of solving the PnP problem. Most of them focus on the improvement of software algorithms.

The core of solving the PnP problem is to determine the camera pose based on 3D reference points and their 2D projected points. The minimum number of points required to solve this problem is three. In this case, the PnP problem is transformed into a P3P [8] problem. Accordingly, when the number of points is 4 or 5, the PnP problem transforms into P4P and P5P problems, respectively. Although solving the PnP problem with a small number of points is possible, the results can become inaccurate under noise [9]. By using more points, the accuracy of the calculation can be significantly improved.

Most PnP solution algorithms support input of an arbitrary number of points and can be broadly classified into two categories: iterative and non-iterative methods [10]. Iterative methods approach the PnP problem as an optimization problem between 3D reference points and 2D projected points. They represent the error through a nonlinear cost function [11,12] and then employ iterative optimization techniques, such as the Gauss–Newton method and the Levenberg–Marquardt method, to minimize this error. When the algorithm converges correctly, iterative algorithms tend to achieve higher accuracy, but their computational process is more complex, resulting in a heavier computational burden. Iterative methods may fall into local minima, leading to incorrect calculation results [11].

Non-iterative algorithms avoid the multiple calculations and adjustments that may be required during the iterative process. These algorithms have a lower dependence on initial values because they do not rely on the gradual approximation in the iterative process. The Direct Linear Transformation (DLT) [13] is a viable method for solving the PnP problem. However, it ignores some important inherent constraints, such as the orthogonality and unitarity of the rotation matrix, which may affect the stability and accuracy of the results.



To improve accuracy, various methods have been proposed, but some of these algorithms have a relatively high computational complexity. For example, [14] with  $O(n^2)$ , [15] with  $O(n^5)$ , or even [16] with  $O(n^8)$ . EPnP [1] is an efficient non-iterative algorithm with linear complexity, it utilizes four sets of virtual “control points” to represent all actual points, and by solving for the positions of these control points in the camera coordinate system, it realizes high-precision pose estimation.

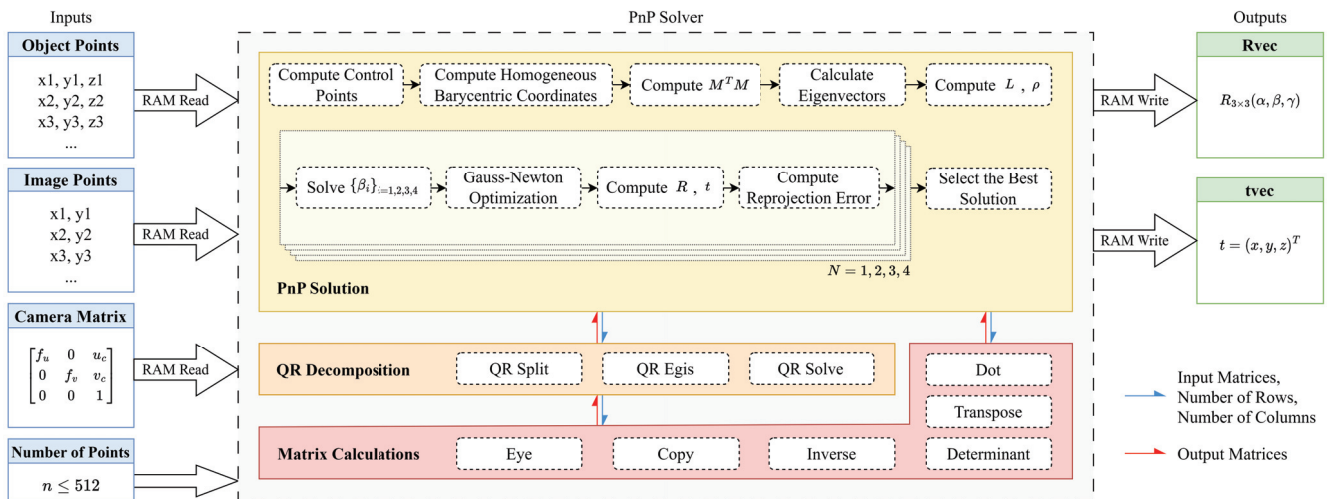
Although previous work has improved the speed and accuracy of solving the PnP problem algorithmically, they have not explored implementation methods other than using CPUs. Furthermore, these algorithmic improvements rarely take into account the power consumption during the computational process of the algorithms. In this work, we improve the PnP solver from the hardware level, focusing on reducing power consumption during the algorithm execution process, while ensuring accuracy and speed.

In the proposed system, we utilize the EPnP algorithm proposed in [1]. The choice of the EPnP algorithm is primarily based on the following considerations: Firstly, the EPnP algorithm is a non-iterative method with linear complexity, which implies that its efficiency advantage is evident regardless of whether the data comprises four points or hundreds of points. The EPnP algorithm is significantly faster than iterative algorithms and algorithms with high complexity, which is crucial for real-time PnP computation. Furthermore, the EPnP algorithm not only ensures computational efficiency but also achieves high positioning accuracy. The EPnP algorithm can further enhance the pose estimation accuracy through post-optimization. Therefore, by using the EPnP algorithm to construct a hardware accelerator, there will be minimal loss of precision during the computation process. In addition, the EPnP algorithm allows for parallel computation, such as in the process of matrix multiplication, which can fully utilize the parallel computing capabilities of FPGAs, thereby accelerating the calculation speed.

### 3. Hardware Architecture

#### 3.1. Overview of the System

A high-level block diagram of the proposed system is shown in Figure 1. The proposed PnP solving accelerator consists of three main components: matrix calculation, QR decomposition (QRD), and PnP solution.



**Figure 1.** High-level block diagram of the proposed PnP solving accelerator.

The input data of the proposed system are 3D reference points and their 2D projections, the camera matrix, as well as the number of point pairs used in the calculation process. The output data are the camera pose. The number of input point pairs for PnP calculation varies with different applications. Depending on the number of input point pairs, the amount of storage space required during the calculation process also varies. Given the limitation of



FPGA devices in facilitating dynamic memory allocation, it is necessary to allocate memory space in advance. After weighing the trade-off between the practicality of the design and the consumption of hardware resources, the proposed system has been designed to support the calculation of a maximum of 512 points.

During the solution process of the PnP algorithm, there are significant differences in the dynamic range of data based on different input data. When using fixed-point numbers to store and compute data, in order to avoid both overflow and loss of precision, a higher bit width is required. To reduce the trouble of fixed-point conversion during the computation process, all decimal data are stored and computed as 32-bit floating-point numbers. Thanks to the HLS tool, the computation of floating-point arithmetic becomes straightforward, which is challenging to accomplish with RTL code.

Except for the number of point pairs, the input and output interfaces of the solver core are implemented as block RAM interfaces. In each clock cycle, a floating-point number can be retrieved from or stored in the block RAM. Additionally, users are allowed to specify the number of points involved in the calculation by changing the parameter  $n$ , which is implemented as a vector. During the computation process, the first  $n$  points of data stored in the block RAM are utilized, while the subsequent data are ignored, thus enabling the adjustment of the number of point pairs involved in the computation.

In addition to the input and output data signals, the solver core also carries control signals that initiate the computation process and indicate the completion of the computation. To use the solver core, users need to implement the external logic for initializing the solver core on the FPGA. The coordinates of the reference point, the projection point, and the camera matrix are first written into the block RAM. Then, the number of points to be used in the calculation is specified, and finally, the solver core is initiated to start the computation. After the computation is completed, the camera pose can be read from the block RAM. The solver core independently completes the PnP calculation without the involvement of other modules, achieving hardware acceleration for solving the PnP problem. During this process, the solver core is not responsible for data interaction with external circuits, and additional logic circuits need to be implemented to realize data transmission.

### 3.2. Basic Matrix Calculations

In the process of solving the PnP problem, certain matrix operations are required. Performing matrix operations on hardware is not as simple as on a processor. Despite the powerful capabilities of HLS tools, designing efficient hardware still requires extensive experience and design skills. This is mainly due to the limitation of FPGA storage methods, which cannot dynamically allocate memory during the computation process, making it impossible to directly use existing C++ libraries like Eigen [17] in HLS designs. Works such as [18] have attempted to design matrix operation libraries based on HLS, but due to their non-opensource nature, we have developed a library containing a set of basic matrix operations that can be synthesized in the HLS environment on our own. The functionalities of the library are shown in Table 1. By encapsulating matrix operations into modular functions, we facilitate a unified approach to configuring parallel optimization strategies for matrix computations within the HLS tools, while also enhancing the reusability of the code.

**Table 1.** Supported matrix operations.

HLS Function	Description	Equation <sup>1</sup>
eye	Generating identity matrix	$A = I$
copy	Copy a matrix to another one	$B = A$
transpose	Find the transpose of a matrix	$B = A^T$
inverse	Find the inverse of a matrix	$B = A^{-1}$
dot	Calculate dot product	$C = A \cdot B$
determinant	Calculate determinant	$det =  A $

<sup>1</sup> A, B, and C are all matrices.

In order to reduce the complexity of implementation, thereby reducing the hardware resources occupied by the FPGA implementation and optimizing the computation speed, this matrix library only implements the necessary matrix operations and dimensions in the EPnP solution. Since the determinant is only required when calculating the rotation matrix  $R$ , while  $R$  is a  $3 \times 3$  matrix, the determinant function is only implemented for  $3 \times 3$  matrices. For other matrix operations, arbitrary dimensions within a certain range are supported. Due to the limitation that FPGAs cannot dynamically allocate memory, these functions pre-allocate storage space based on the maximum matrix dimension during the computation process. In this way, there is enough space to save temporary data during the calculation process.

The parameters of the functions include pointers to input and output arrays, as well as the dimensions of the matrices. The HLS tool synthesizes the matrices into block RAMs, where it reads data from the input block RAMs based on the matrix size during the computation process, performs the calculations, and then writes the results back to the output block RAMs. The implementation is identical to C++ in other respects. For example, the matrix inversion operation can be expressed in Algorithm 1.

---

**Algorithm 1: Matrix Inversion.**


---

```

Input: A, n(Matrix dimension)
Output: AInv
// Using the elementary transformation method
// Transform A into an upper triangular matrix
1  AInv ← A
2  for i in 0 : n − 2 do
3      for j in i + 1 : n − 1 do
4          temp = A[j*n + i]/A[i*n + i]
5          for k in i + 1 : n − 1 do
6              A[j*n + k] -= temp * A[i*n + k]
7          end
8          for k in 0 : n − 1 do
9              AInv[j*n + k] -= temp * AInv[i*n + k]
10         end
11     end
12 end
    // Transform A into an diagonal matrix
13 for i in n − 1 : 1 do
14     for j in 0 : i − 1 do
15         temp = A[j*n + i]/A[i*n + i]
16         A[j*n + i] -= temp * A[i*n + i]
17         for k in 0 : n − 1 do
18             AInv[j*n + k] -= temp * AInv[i*n + k]
19         end
20     end
21 end
    // Transform A into an identity matrix
22 for i in 0 : n − 1 do
23     for j in 0 : n − 1 do
24         AInv[i*n + j] /= A[i*n + i]
25     end
26 end

```

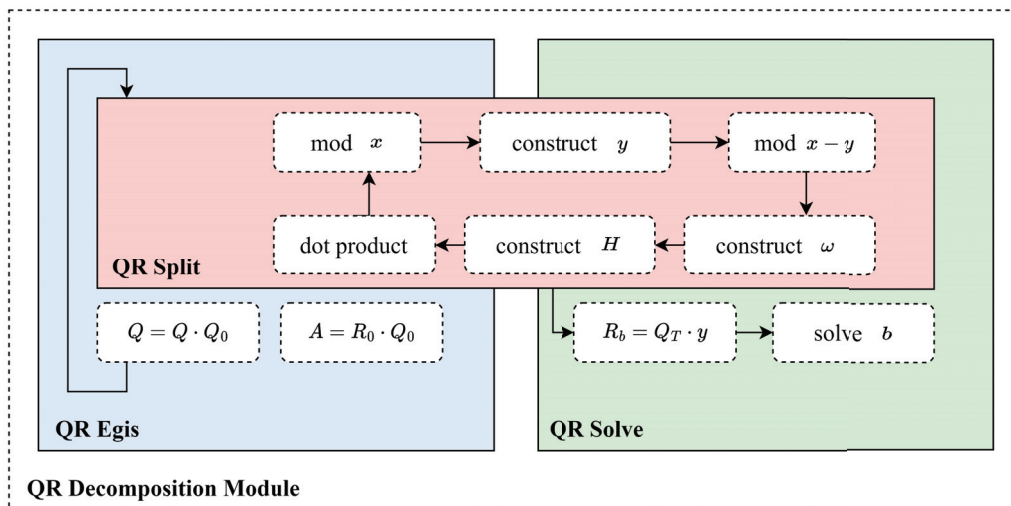
---

Compared to implementing Verilog, HLS places more emphasis on the computational process of data, eliminating the trouble of manually controlling the flow through state machines. Users can focus their attention on the implementation of the algorithm.

### 3.3. QR Decomposition Module

QR decomposition is used to solve matrix eigenvalues, eigenvectors, and solutions to homogeneous linear equations. QR decomposition breaks down a matrix into an orthogonal matrix  $Q$  and an upper triangular matrix  $R$ . Through iterative methods, the original matrix can be gradually transformed into an approximately upper triangular matrix, thereby facilitating the solution of eigenvalues. This decomposition method is computationally more efficient, especially for large matrices, as it significantly reduces the amount of computation. Additionally, due to the properties of orthogonal matrices, QR decomposition exhibits high numerical stability, contributing to more accurate eigenvalue solutions. The QR decomposition method for solving eigenvalues can be applied to various types of matrices, including real symmetric matrices and complex matrices. In the process of solving eigenvalues and eigenvectors, the QR algorithm typically demonstrates fast convergence rates. This implies that, with the same number of iterations, the QR algorithm can approximate the true eigenvalues and eigenvectors more quickly, thereby enhancing the efficiency of the solution. Apart from QR decomposition, there exist other methods for solving eigenvalues and eigenvectors, but they come with their own limitations. For instance, the inverse matrix method requires the matrix to be invertible, the Jacobi iteration method necessitates the matrix to be symmetric, and the method based on the definition involves solving a high-degree polynomial equation. In terms of practical applicability and convergence speed, these methods often pale in comparison to the QR decomposition approach.

The hardware architecture of the QRD module consists of three parts: QR split, QR egis, and QR solve. A functional block diagram of the QRD module is shown in Figure 2.



**Figure 2.** Hardware architecture of the QR decomposition module.

The QR egis module and the QR solution module share the same QR split module. When a matrix is decomposed, the result can be utilized to compute eigenvalues and eigenvectors by the QR Egis module, or for solving linear least squares problems by the QR Solve module.

#### 3.3.1. QR Split

QR split module factors an  $m \times n$  matrix  $A$  into the product of an  $m \times m$  orthogonal matrix  $Q$  and an  $m \times n$  upper triangular matrix  $R$ . QR split can be implemented through various algorithms, including the Gram–Schmidt algorithm [19], the Householder

algorithm [20], the Givens algorithm [21,22], the Modified Gram-Schmidt algorithm [23,24] and so on. Among them, the Householder algorithm is widely used due to its numerical stability. In the proposed system, a QR split based on Householder transformation is implemented, which can be expressed in Algorithm 2.

---

**Algorithm 2:** Householder QR.

---

**Input:** A  
**Output:** Q (orthogonal matrix), R (upper triangular matrix)

```

1   $Q \leftarrow I, R \leftarrow A$ 
2  for  $i$  in  $1 : n$  do
3       $x = A(1 : m, i)$ 
4       $mod = norm(x)$ 
5      for  $j$  in  $1 : m$  do
6          if  $i == j$  then
7               $y[j] = mod$ 
8          else
9               $y[j] = 0$ 
10         end
11     end
12      $\omega = (x - y) / norm(x - y)$ 
13      $H = I - 2\omega\omega^T$ 
14      $Q = dot(H, Q), R = dot(H, R)$ 
15 end

```

---

In Algorithm 2, A is the input matrix, and Q and R are the output matrices. After initializing matrix Q as the identity matrix and matrix R as the input matrix, the modulus of the first column vectors of A is first calculated, which is used to construct the unit vector  $\omega$ . Next, a Householder matrix H is constructed and multiplied by matrices Q and R. At this point, all elements in the first column of matrix R, except the first element, are reduced to 0. The same operation is performed on the remaining columns of the input matrix in sequence, since the Householder matrix H is orthogonal, its cumulative product result Q is also an orthogonal matrix, and R is reduced to an upper triangular matrix.

For FPGA implementation, the norm function is replaced by first computing the sum of squares of the vector elements and then taking the square root of that sum. The square root can be calculated using the sqrt function provided by HLS tools, eliminating the need to use the CORDIC core. Note that in Algorithm 2, there are two loops. The sub-loop is used to initialize the vector y, and since the computations within the loop are independent of each other, they can be unrolled to take advantage of parallel computing on an FPGA. However, in the parent loop, the Q and R matrices are generated iteratively, so the loop cannot be unrolled. It must wait for one iteration to complete before starting the next iteration.

### 3.3.2. QR Egis

The QR Egis module implements an iterative method for computing eigenvalues and eigenvectors, using Algorithm 3.

At the beginning, matrix Q is initialized as the identity matrix. By repeatedly performing QR decomposition on matrix A and multiplying the obtained orthogonal matrices together, the matrix Q gradually approaches the eigenvector matrix. When matrix A converges approximately to an upper triangular matrix, the iteration is complete, and the main diagonal elements are then the eigenvalues.

**Algorithm 3:** QR Egis.

---

**Input:** A  
**Output:** Q (Eigenvectors), e (Eigenvalues)

```

1  $Q \leftarrow I$ 
2 for  $i$  in 1 :  $iteration\_times$  do
3    $Q_0, R_0 = QR\_Split(A)$ 
4    $Q = dot(Q, Q_0)$ 
5    $A = dot(R_0, Q_0)$ 
6 end
7  $e = trace(A)$ 

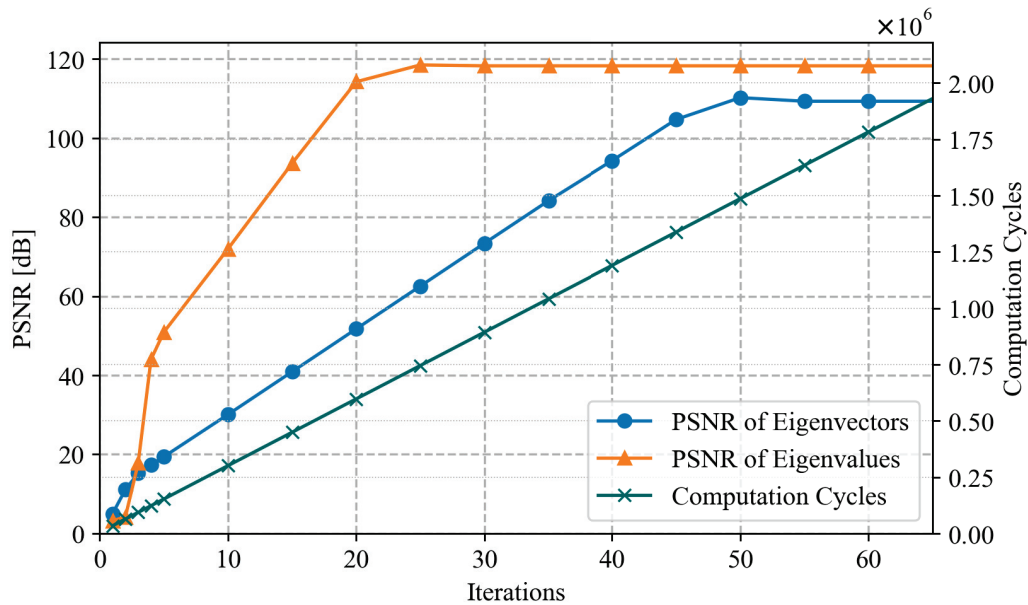
```

---

The number of iteration times determines the accuracy and speed of the calculation. As the number of iterations increases, the result becomes more precise, but the computation time also increases. Figure 3 demonstrates the influence of the number of iterations on the accuracy and computational time of QRD. The peak signal-to-noise ratio (PSNR) is defined as

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right) \quad (1)$$

where  $MAX$  is the value of the largest element in the result matrix, and  $MSE$  stands for Mean Squared Error, which is the minimum squared error between the output matrix and the true value.



**Figure 3.** The relationship between iterations and PSNR as well as computational speed.

As the number of iterations increases, the computational time grows linearly, while the errors of eigenvalues and eigenvectors continue to decrease and finally stabilize. When the number of iterations is small, the results exhibit significant errors due to non-convergence. However, once the number of iterations increases to a certain value, the errors are primarily influenced by floating-point precision, rendering further increases in iteration count ineffective in improving the results. Since the computation time for each iteration is the same, the number of computation cycles grows linearly with the number of iterations.

The proposed system selects the number of iterations as 10 because the data accuracy can already fulfill the precision demands of the PnP problem at this point. Selecting an excessively large number of iterations can lead to a protracted computation time.

### 3.3.3. QR Solve

The QR solution module is used for solving linear least squares problems. Its core objective is to find a set of solutions that minimize the sum of squared errors between the model's predicted values and the actual observed values when an exact solution to a set of linear equations does not exist. The form of the system of equations is given as

$$Ab = y \quad (2)$$

When the matrix  $A$  can be factored into the product of an orthogonal matrix  $Q$  and an upper triangular matrix  $R$  through QR decomposition, the equation transforms into

$$QRb = y \quad (3)$$

i.e.,

$$Rb = Q^{-1}y = Q^T y \quad (4)$$

Since  $R$  is an upper triangular matrix, the solution to the equation can be obtained using Algorithm 4:

---

**Algorithm 4:** QR Split.

---

**Input:**  $A, y$   
**Output:**  $b$  (Solve for  $Ab = y$ )

```

1  $Q, R = QR\_Split(A)$ 
2  $Rb = dot(Q^T, y)$ 
3 for  $i$  in  $n : 1$  do
4   for  $j$  in  $1 : n$  do
5      $Rb[i] = Rb[i] - R[i * n + j] * b[j]$ 
6   end
7    $b[i] = Rb[i] / R[i * n + i]$ 
8 end
```

---

### 3.4. PnP Solution

The PnP solution is the core function of the system. During the PnP solution process, the same set of data is used multiple times during the computation, which poses difficulties in synchronizing the functional modules. In this design, when a portion of the hardware circuitry is performing a computation, the remaining circuitry remains in an inactive mode, and only after the computation is completed will the subsequent computation be initiated, thereby ensuring the correctness of the results. Since there were no major modifications made to the EPnP algorithm, the computational process of the PnP solver remains the same as that of the original algorithm. First, four control points are determined based on the world coordinates. For each world coordinate, homogeneous barycentric coordinates are calculated based on the control points. Next, matrix  $M$  is constructed with each reference point, and the product  $M^T M$  is calculated simultaneously. Upon the completion of  $M^T M$  computation, the eigenvectors of this matrix are calculated. Then, the matrix  $L$  is formed with eigenvectors, and  $\rho$  is derived from control points.

We compute solutions for all four distinct values of  $N$ , which is the effective dimension of the null space, and subsequently select the optimal solution that minimizes the reprojection error, thereby ensuring the highest level of accuracy and precision. During the computation process, some data will be used multiple times, and these data are stored in BRAM. When these data are being read and written, they cannot be accessed simultaneously by other modules, so the entire PnP solution process is executed sequentially in the order of the algorithm.

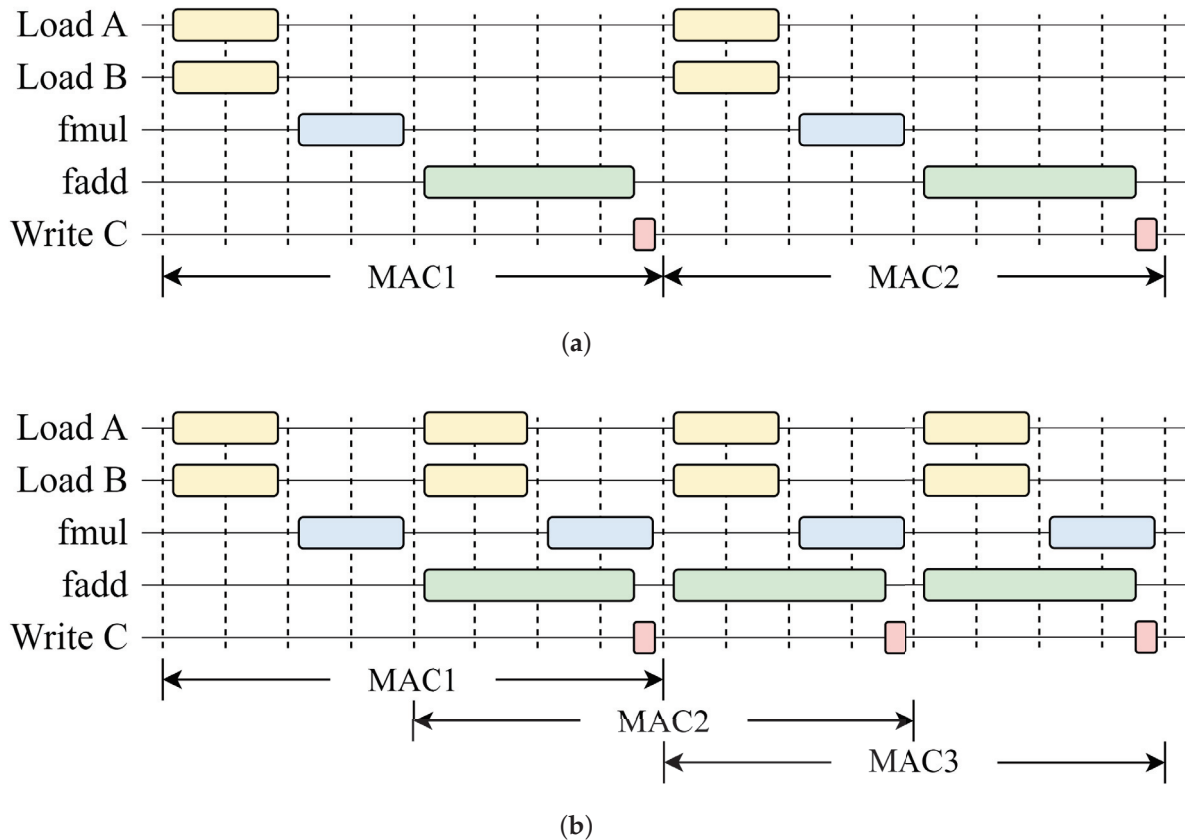


## 4. Optimization

### 4.1. Pipelining

Pipelining is a frequently employed optimization strategy in FPGA design that enables concurrent hardware execution, thereby significantly elevating computational performance. In the traditional RTL design method, designers need to manually control the data flow, dividing the computing process into multiple steps and allocating them to several clock cycles. In HLS implementation, designers only need to set the pipeline parameters, and the HLS tool will automatically implement the pipeline.

In the process of solving the PnP problem, matrix multiplication is a time-consuming process. The core of matrix multiplication is multiplication–accumulation, which consists of four stages: data loading, floating-point multiplication, floating-point addition, and data saving. A single multiplication–accumulation process requires eight clock cycles, as shown in Figure 4. Before pipelining is implemented, the next operation can only be executed after the previous one is completed, resulting in low hardware utilization. After applying pipelining, when the floating-point addition operation is being performed, the loading of the next set of data and the floating-point multiplication can be calculated simultaneously. The interval between each two multiplication–accumulation operations is reduced to four clock cycles.



**Figure 4.** The timing of multiply–accumulate (MAC) operations before (a) and after (b) applying pipelining.

### 4.2. Optimization for Constructing the $M$ Matrix

In the original process of estimating pose using the EPnP algorithm, we need to construct a matrix  $M$  using 3D world coordinates and their 2D projections and store the matrix in RAM. After completing the process, the matrix  $M$  is then read from RAM and the product  $M^T M$  is calculated. The product is used to calculate eigenvectors in order to solve the PnP problem. This calculation process is very efficient for software. Unfortunately, for FPGA, the calculation of matrix  $M$  is a process that demands considerable time and

resources. Matrix  $M$  is a  $2n \times 12$  matrix computed from each reference point. Due to the storage of reference points in floating-point format, with each number occupying 4 bytes of space, the matrix  $M$  occupies a total space of  $2n \times 12 \times 4$  bytes. As the number of points increases, the size of the  $M$  matrix expands at a rate of two times, leading to an increase in the amount of RAM consumed. The proposed design supports a maximum of 512 reference points, thus requiring a reserved space of 48 KB for the  $M$  matrix. However, the total amount of RAM in FPGA is typically limited. This represents a significant resource overhead for FPGA.

There are two types of RAMs in FPGA: block RAM and distributed RAM. Distributed RAM typically has a smaller capacity but can achieve a larger bit width; block RAM has a larger capacity but a limited bit width. For the  $M$  matrix, which occupies a large amount of space, it is stored in block RAM. At this point, the read and write speed of the block RAM becomes a performance bottleneck for the system. Although the hardware circuits implemented by FPGA can be executed in parallel, the entire computation process is forced to be serial due to the necessity of serially writing data into BRAM, and then serially reading the data out of BRAM after the computation is completed. The calculation process hinders the effective implementation of pipelining, thereby diminishing the overall execution efficiency. Due to the significant time and hardware resource consumption required for computing the  $M$  matrix, optimizations targeting the calculation of the  $M$  matrix are necessary.

In this design, to optimize the computation process of matrix  $M$ , the construction and dot product calculations are considered as a whole, and the computation process is split up to bypass block RAM read and write operations as much as possible. As shown in Figure 5, instead of constructing the whole matrix  $M$  and then computing the product  $M^T M$ , we compute only one row of the  $M$  matrix each time. Multiplying this row vector by its transpose results in an intermediate matrix  $M_n^T M_n$ . Finally, by accumulating the intermediate matrices, the final result  $M^T M$  is obtained.

The result obtained through this calculation process is the same as that obtained from the original calculation process. On the one hand, the proposed calculation process eliminates the need to store the entire  $M$  matrix, which reduces the block RAM usage. On the other hand, since the intermediate matrix in the calculation process is stored in distributed RAM, which has a larger bit width than block RAM, the FPGA can perform pipelined computation more efficiently, thereby improving the calculation speed.

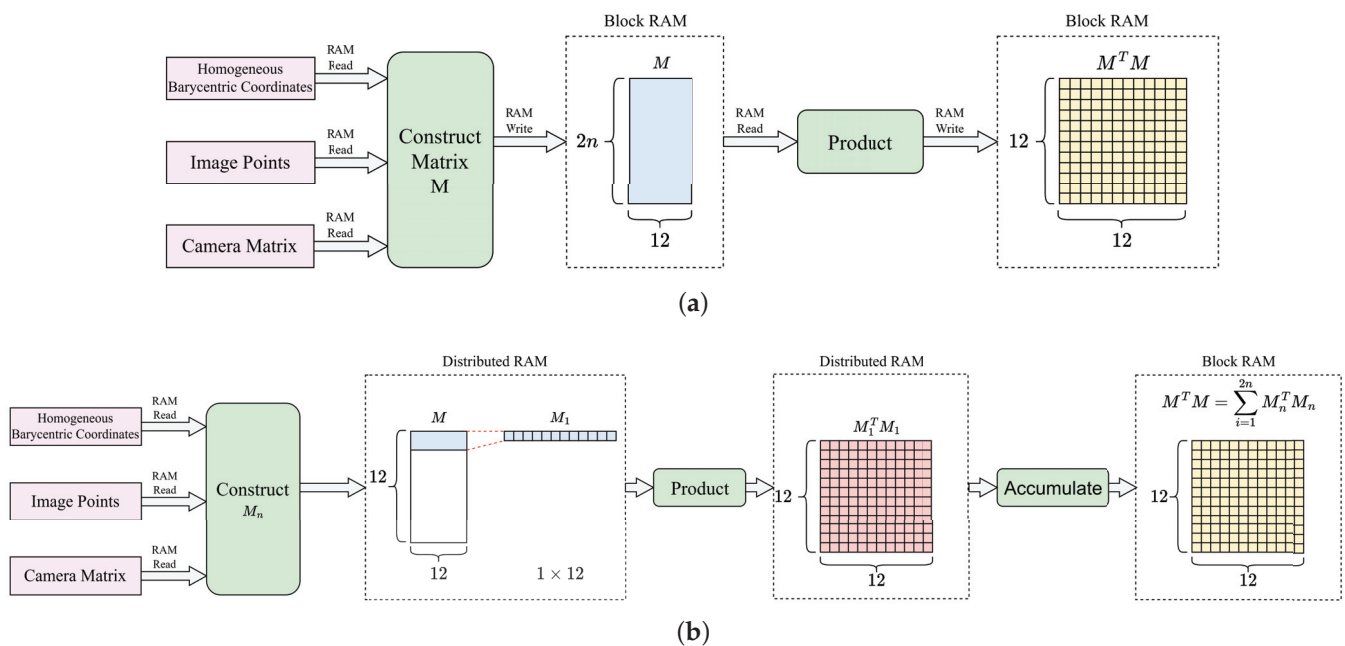


Figure 5. The original (a) and optimized (b) calculation process of  $M^T M$ .

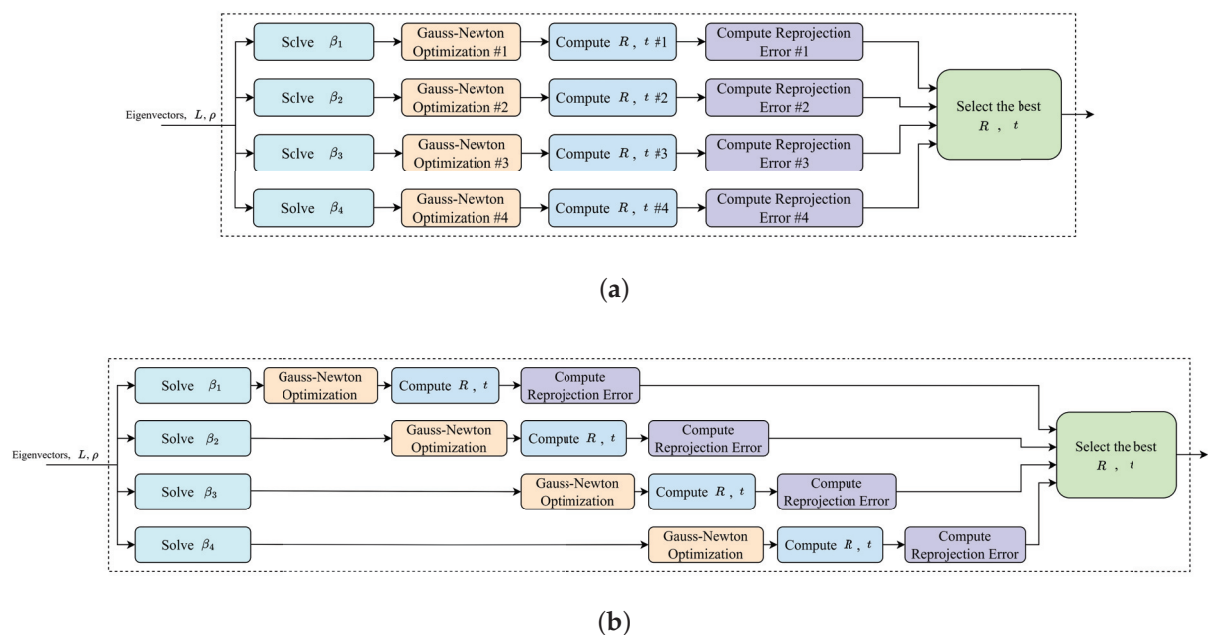
To validate the performance of the proposed computational method, we implemented two computation processes on FPGA and a comparative analysis of performance disparities was conducted between the two methods under identical input data conditions. Table 2 presents the experimental results. In the original computational process, it is necessary to retain the entire  $M$  matrix, resulting in the utilization of 32 block RAMs, which significantly impact storage requirements. Conversely, the proposed computational method saves storage space by eliminating the requirement to store the entire  $M$  matrix, marking a notable advancement in resource optimization. Due to the utilization of distributed RAM, the proposed computational process employs more Flip-Flops (FFs) and Lookup Tables (LUTs). When the number of points in the computational data is 512, the proposed computational method takes 57,391 clock cycles, whereas the original method consumes 595,120 clock cycles. By refining the computational process, a 10-fold acceleration in computation speed is achieved.

**Table 2.** Resource utilization and speed comparison between the proposed calculation process and the original process.

	Proposed	Original	Ratio
BRAM	0	32	0%
DSP	7	8	87.5%
FF	5582	1182	472.25%
LUT	3826	1672	228.83%
cycles	57,391	595,120	9.64%

#### 4.3. Trade-Off between Speed and Area

The EPnP algorithm computes solutions for all four values of  $N$ . In theory, all four solutions can be calculated simultaneously to shorten the overall computing time, as shown in Figure 6a. While the computational speed has been accelerated, the implementation of four identical modules for Gauss–Newton optimization, pose calculation, and reprojection error computation has led to an increase in the consumption of hardware resources. Since the computing process takes a relatively short time, the design still adopts the serial computing method shown in Figure 6b. This allows for the reuse of hardware circuits, thereby reducing the consumed hardware resources.



**Figure 6.** Comparison between parallel computing (a) and pipeline computing (b).

## 5. Experimental Results

### 5.1. Accuracy and Speed Evaluation

In our experimental setup, we employ a predefined camera pose to project randomly generated world coordinate points onto a 2D plane, thereby obtaining the corresponding 2D projection points. In reality, due to the influence of noise, the 2D projection points will deviate within a certain range. In order to verify the system performance under the influence of noise, we applied Gaussian noise to each 2D projection point. We utilized the proposed PnP solution module to estimate the camera pose from these projections. At the same time, we used the solvePnP function in OpenCV as a benchmark to compare the differences in results between hardware and software implementations. OpenCV's solvePnP method is a general approach used in computer vision projects to solve the PnP problem, and its default solver is also EPnP.

Figure 7 shows the accuracy performance of the proposed system. When no noise is introduced, the PSNR of the rotation matrix  $R$  exceeds 100 dB, while the displacement  $t$  achieves over 70 dB. At this point, the primary source of precision error lies in the accuracy of QR iteration for eigenvalue decomposition. In the presence of noise, the accuracy of the results decreases, and the PSNR value increases as the number of input points increases. At this point, the PSNR of the rotation matrix  $R$  can still be maintained at around 50 dB, with its accuracy slightly lower than that of OpenCV. This is due to the fewer iterations when solving for the eigenvectors. However, its accuracy can already satisfy most engineering requirements.

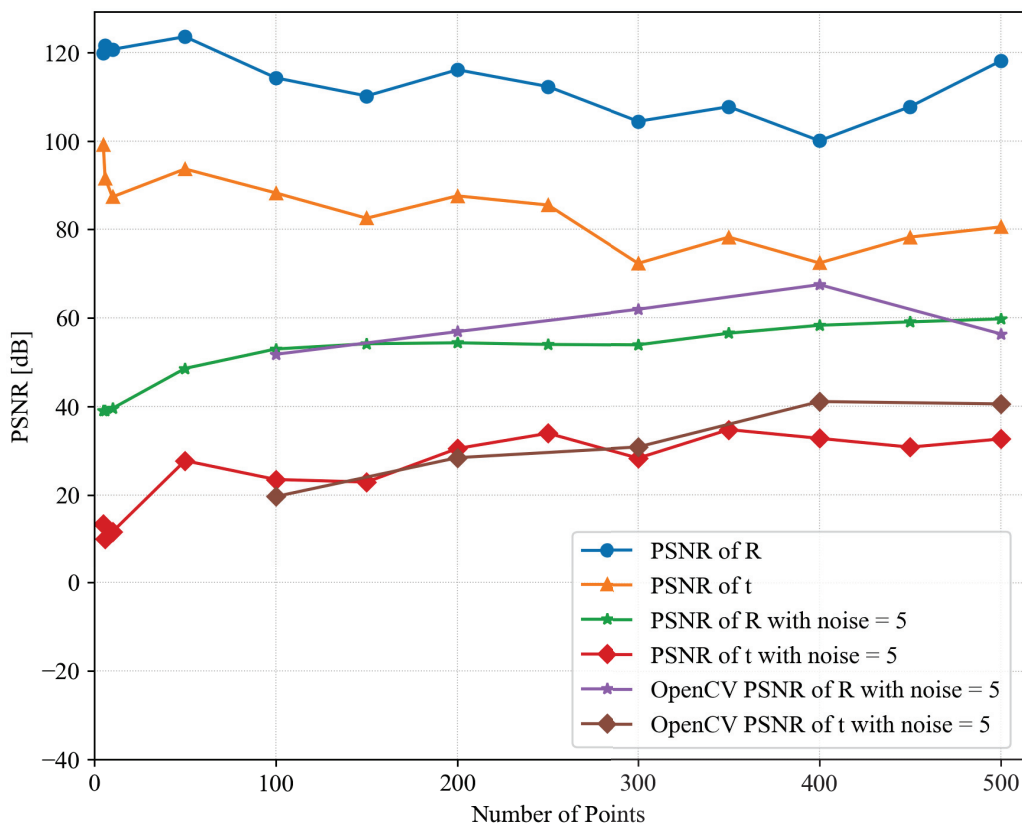
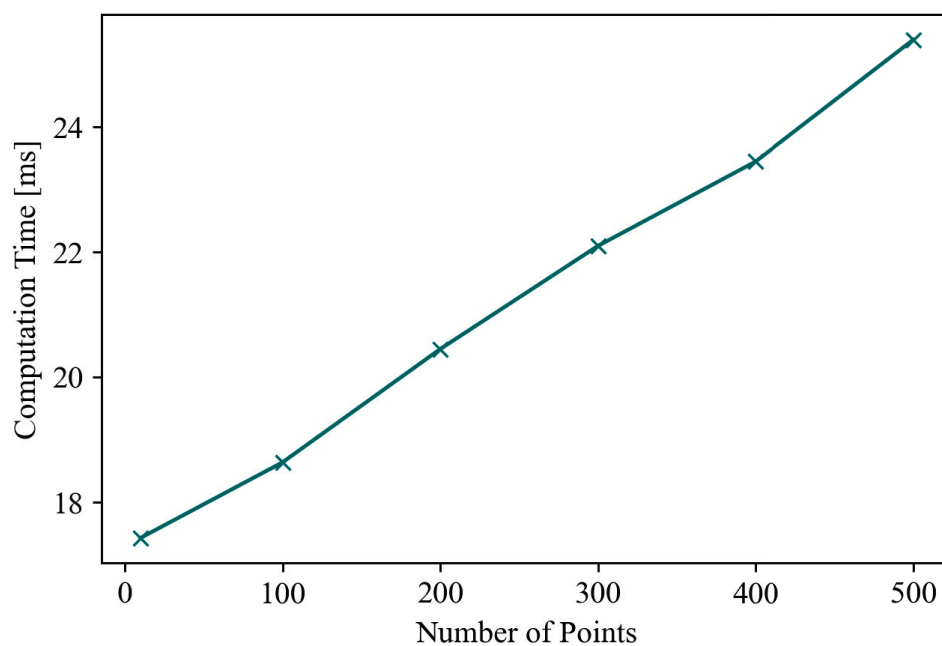


Figure 7. PSNR performance of the proposed system and OpenCV with noise = 0, 5.

Figure 8 demonstrates the relationship between the number of calculation points and the computation time. The computation time grows linearly with the number of points, with a base value of 17.5 milliseconds.



**Figure 8.** The relationship between number of points and computation time.

### 5.2. Hardware Implementation

The PnP solver core is synthesized and implemented for Xilinx XC7K325T with Vivado HLS. The proposed system takes 1,473,295,500 clock cycles to complete computations for an input number of points of 300. In this system, the PnP algorithm is calculated in series, and the HLS tool will distribute the entire calculation process evenly to each clock cycle. While meeting the timing requirements, it tries to shorten the number of clock cycles as much as possible. Therefore, the clock frequency of FPGA has little impact on the total time of a single calculation. After actual testing, a higher clock frequency cannot bring a shorter total calculation time. Upon comprehensive consideration, a clock frequency of 67 MHz has been deliberately selected for the FPGA implementation. Therefore, the total calculation time is 22.1 ms. The hardware resource utilization of the prototype system is reported in Table 3.

**Table 3.** FPGA resource utilization of the proposed system.

	C Synthesis	Implementation	Total	Utilization
LUT	120,238	84,120	203,800	41.28%
FF	79,451	52,069	407,600	12.77%
BRAM_18K	135	77	890	17.30%
DSP	487	550	840	65.48%

### 5.3. Power Efficiency Comparison

We implemented the same algorithm on several computing platforms and measured the computational time consumed. To measure the actual operational power consumption of the system, we downloaded the FPGA implementation mentioned above onto a Kintex-7 Base board. The Kintex-7 Base board is equipped with a Xilinx XC7K325T-FFG676 FPGA. During the testing process, the FPGA cyclically performs PnP calculations for 300 input data points, which allows us to calculate the average power consumption of the system based on its total power consumption over a period of time. Since the Kintex-7 Base board is powered by USB, we inserted a USB power meter between the power adapter and the board to record the electricity consumed by the board over one minute. The same method was used to test the power consumption of the ZYNQ processor. A program that



repeatedly executes the PnP solution algorithm is downloaded onto a XC7Z010 board, and the average power consumption is measured using the USB power meter. To measure the execution time of the PnP algorithm on the Zynq-7000 processor, another experiment was conducted. The system timer of the Zynq-7000 was used to measure the time required for the program to execute from start to finish, and the results were sent to the host computer via a serial port.

For desktop-level processors, their power consumption can be measured by sensors on the motherboard. We used CoreTemp software to read the overall package power consumption of the processor. Due to the presence of the operating system, the processor consumes power even when it is not executing the PnP solution task, so we measured the power consumption of the processor during both idle and execution states and took the difference as the power consumption of the PnP solution.

Table 4 shows the comparison of calculation time and speed of the PnP solution algorithm on different platforms. Despite the relatively high resource utilization rate of this design, the power is still only 1.7 W. There are two main reasons for achieving such low power consumption: Firstly, the operation of an algorithm requires the implementation of both computational and control circuits. The operation process of FPGA is a state machine, which consumes most of its energy on computation, while control only consumes a small portion of the power. Unlike FPGA, CPUs consume a large amount of power on fetching and decoding instructions during computation, with only a small part of the power consumption used for actual computation. Therefore, its energy efficiency is much lower than that of FPGA. Secondly, the proposed PnP calculation process on FPGA runs in series. When a part of the FPGA circuit is calculating, the rest of the parts are inactive. This allows FPGA to have lower power consumption per unit time. However, the disadvantage of this approach is that FPGA does not fully utilize its hardware in terms of time, resulting in the consumption of more resources and also leading to slower computation speeds.

**Table 4.** Speed and power comparison.

	Ours	Core i7-12700H	Zynq-7000
Power	1.7 W	32.5 W	1.13 W
Computation Time	22.1 ms	20.8 ms	54.0 ms
Energy Consumption	26.52 W·ms	676.0 W·ms	61.02 W·ms

Although the FPGA implementation is slightly slower than the X86 processor in terms of speed, each computational iteration consumes merely 3.9% of the electric energy consumed by the aforementioned method, demonstrating substantial energy efficiency. When solving the PnP problem on battery-powered computing devices, higher energy efficiency leads to longer working hours, making it suitable to be applied to robotics and VR headsets. At the same time, the FPGA implementation exhibits significantly enhanced processing speeds in comparison to the ARM-based application processor. FPGA achieves faster computing speed while consuming only 43.4% of the energy of the latter. FPGA can achieve real-time data processing at 45.2 fps, which is important for computer vision tasks.

## 6. Conclusions

This paper proposes an energy-efficient Perspective-n-Point accelerator implemented on FPGA. The solver core implements a basic matrix operation library, as well as eigenvalue and eigenvector solvers and linear least squares equation solvers based on QR decomposition. On this basis, the computational process of the EPnP algorithm is implemented. By using hardware pipelining and adjusting the calculation order of the algorithm, the computation speed is improved while reducing hardware resource consumption. The proposed design not only meets the requirement for real-time computation, but it is also more efficient compared to other computing hardware. The advantages of this design make it suitable to be applied to mobile robots and other situations that require low power consumption.



Future research can be carried out in the following areas. (1) In the hardware PnP solver proposed in this design, all data are calculated and stored using single-precision floating-point numbers. Fixed-point numbers offer faster computation speeds compared to floating-point numbers while also occupying fewer FPGA resources. Therefore, replacing floating-point numbers with fixed-point numbers for PnP solution can be explored, and the impact of this operation should be evaluated. (2) In the EPnP algorithm, due to the interdependence of data, the same set of data may be used at different stages of the computation. When the computation of the current sequence is completed, the data still need to be stored in RAM, occupying storage space and preventing the next set of data from starting its computation. In this design, only after a round of PnP solution is completed can the next set of data begin computation, which reduces the utilization rate of hardware resources. To address this, pipelining optimization can be applied throughout the design to reduce the initiation interval and improve throughput. (3) Parallel QR decomposition algorithms can be studied to reduce the latency of the design.

**Author Contributions:** Conceptualization, methodology, developing, testing and writing—original draft preparation, H.L.; writing—review and editing, supervision, Q.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

FPGA	Field-Programmable Gate Array
PnP	Perspective-n-Point
HLS	High-level Synthesis
SLAM	Simultaneous Localization and Mapping
VR	Virtual Reality
RTL	Register Transfer Level
QRD	QR Decomposition
DLT	Direct Linear Transformation
RAM	Random Access Memory
BRAM	Block Random Access Memory
CORDIC	Coordinate Rotation Digital Computer
PSNR	Peak Signal To Noise Ratio
MSE	Mean Squared Error
DSP	Digital Signal Processor
FF	Flip-Flop
LUT	Look-Up Table
MAC	multiply-accumulate

## References

1. Lepetit, V.; Moreno-Noguer, F.; Fua, P. EPnP: An Accurate  $O(n)$  Solution to the PnP Problem. *Int. J. Comput. Vis.* **2009**, *81*, 155–166. [CrossRef]
2. Cui, T.; Guo, C.; Liu, Y.; Tian, Z. Precise Landing Control of UAV Based on Binocular Visual SLAM. In Proceedings of the 2021 4th International Conference on Intelligent Autonomous Systems (ICoIAS), Wuhan, China, 14–16 May 2021; pp. 312–317.
3. Li, W.; Li, D.; Shao, H.; Xu, Y. An RGBD-SLAM with Bi-Directional PnP Method and Fuzzy Frame Detection Module. In Proceedings of the 2019 12th Asian Control Conference (ASCC), Kitakyushu, Japan, 9–12 June 2019; pp. 1530–1535.
4. Zhou, M.; Li, S.; Lu, W. An Improved SLAM Based On The Indoor Mobile Robot. In Proceedings of the 2022 34th Chinese Control and Decision Conference (CCDC), Hefei, China, 15–17 August 2022; pp. 5594–5601.
5. Zhou, H.; Zhang, T.; Jagadeesan, J. Re-Weighting and 1-Point RANSAC-Based PnP Solution to Handle Outliers. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *41*, 3022–3033. [CrossRef] [PubMed]
6. Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2011**, *30*, 473–491. [CrossRef]

7. Nane, R.; Sima, V.-M.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.T.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [CrossRef]
8. Fischler, M.A.; Bolles, R.C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Readings in Computer Vision*; Fischler, M.A., Firschein, O., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 1987; pp. 726–740, ISBN 978-0-08-051581-6.
9. Zeng, G.; Chen, S.; Mu, B.; Shi, G.; Wu, J. CPnP: Consistent Pose Estimator for Perspective-n-Point Problem with Bias Elimination. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 1940–1946.
10. Pan, S.; Wang, X. A Survey on Perspective-n-Point Problem. In Proceedings of the 2021 40th Chinese Control Conference (CCC), Shanghai, China, 26–28 July 2021; pp. 2396–2401.
11. Lu, C.-P.; Hager, G.D.; Mjolsness, E. Fast and Globally Convergent Pose Estimation from Video Images. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 610–622. [CrossRef]
12. Horaud, R.; Dornaika, F.; Lamiroy, B. Object Pose: The Link between Weak Perspective, Paraperspective, and Full Perspective. *Int. J. Comput. Vis.* **1997**, *22*, 173–189. [CrossRef]
13. Abdel-Aziz, Y.I.; Karara, H.M.; Hauck, M. Direct Linear Transformation from Comparator Coordinates into Object Space Coordinates in Close-Range Photogrammetry. *Photogramm. Eng. Remote Sens.* **2015**, *81*, 103–107. [CrossRef]
14. Fiore, P.D. Efficient Linear Solution of Exterior Orientation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2001**, *23*, 140–148. [CrossRef]
15. Quan, L.; Lan, Z. Linear N-Point Camera Pose Determination. *IEEE Trans. Pattern Anal. Mach. Intell.* **1999**, *21*, 774–780. [CrossRef]
16. Ansar, A.; Daniilidis, K. Linear Pose Estimation from Points or Lines. *IEEE Trans. Pattern Anal. Mach.-Intel-Ligence* **2003**, *25*, 578–589. [CrossRef]
17. Eigen. Available online: [https://eigen.tuxfamily.org/index.php?title=Main\\_Page](https://eigen.tuxfamily.org/index.php?title=Main_Page) (accessed on 24 September 2024).
18. Zhao, W.; Li, C.; Ji, Z.; Guo, Z.; Chen, X.; You, Y.; Huang, Y.; You, X.; Zhang, C. Flexible High-Level Synthesis Library for Linear Transformations. *IEEE Trans. Circuits Syst. II Express Briefs* **2024**, *71*, 3348–3352. [CrossRef]
19. Venkata Siva Kumar, K.; Kopparthi, V.R.; Sabat, S.L.; Varma, K.T.; Peesapati, R. System on Chip Implementation of Floating Point Matrix Inversion Using Modified Gram–Schmidt Based QR Decomposition on PYNQ FPGA. In Proceedings of the 2021 IEEE International Symposium on Smart Electronic Systems (iSES), Jaipur, India, 18–22 December 2021; pp. 84–88.
20. Golub, G.H.; Loan, C.F.V. *Matrix Computations*; JHU Press: Baltimore, MD, USA, 2013; ISBN 978-1-4214-0859-0.
21. Omran, S.S.; Abdul-abbas, A.K. Design and Implementation of 32-Bits MIPS Processor to Perform QRD Based on FPGA. In Proceedings of the 2018 International Conference on Engineering Technology and their Applications (IICETA), Al-Najaf, Iraq, 8–9 May 2018; pp. 36–41.
22. Aslan, S.; Niu, S.; Saniie, J. FPGA Implementation of Fast QR Decomposition Based on Givens Rotation. In Proceedings of the 2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS), Boise, ID, USA, 5–8 August 2012; pp. 470–473.
23. Tan, C.Y.; Ooi, C.Y.; Ismail, N. Loop Optimizations of MGS-QRD Algorithm for FPGA High-Level Synthesis. In Proceedings of the 2019 32nd IEEE International System-on-Chip Conference (SOCC), Singapore, 3–6 September 2019; pp. 138–143.
24. Lee, D.; Hagiescu, A.; Pritsker, D. Large-Scale and High-Throughput QR Decomposition on an FPGA. In Proceedings of the 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, 28 April–1 May 2019; p. 337.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# NanoBoot: A Field-Programmable Gate Array/System-on-Chip Hardware Boot Loader for IoT Devices

Paulino Ruiz-de-Clavijo, German Cano-Quiveu \*, Jorge Juan, Manuel Jesus Bellido, Julian Viejo-Cortes, David Guerrero and Enrique Ostua

Electronics Technology Department, E.T.S. Ingeniería Informática, University of Seville, Avda. Reina Mercedes s/n, 41012 Seville, Spain; pruíz@us.es (P.R.-d.-C.); jjchico@dte.us.es (J.J.); bellido@dte.us.es (M.J.B.); julian@us.es (J.V.-C.); guerre@dte.us.es (D.G.); ostua@dte.us.es (E.O.)

\* Correspondence: germancq@dte.us.es

**Abstract:** This paper presents a new boot loader scheme for embedded devices with file system support built as a hardware module. The work focuses on improving the boot loader hardware and the possibility of carrying out a full boot-up process from the dedicated on-chip hardware, using a light file system to store an operating system kernel. To do so, the new full-hardware boot loader is integrated into two Field-Programmable Gate Array (FPGA) System-on-Chip (SoC), capable of launching a Linux kernel from a formatted removable media.

**Keywords:** boot loader; field-programmable gate array; Internet of Things; System-on-Chip; hardware; embedded device

## 1. Introduction

Today, some embedded systems are commonly used as Internet of Things (IoT) devices. These devices usually have limited hardware and software resources. Therefore, it is often necessary to create custom designs that optimize performance and/or area.

A significant part of these devices are based on System-on-Chip designs, which are complete computer systems capable of performing complex tasks. Each SoC includes a power-on reset device, which ensures that the core and peripherals reach a known state before starting the pre-programmed firmware.

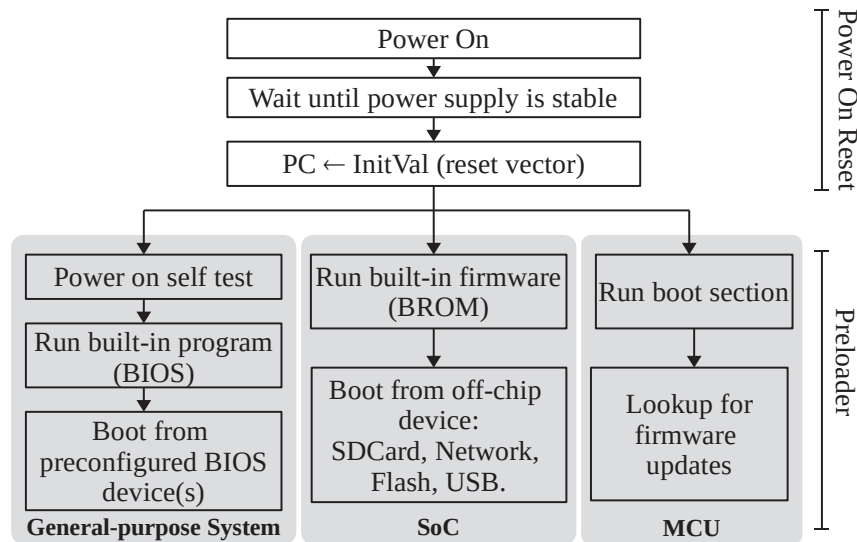
According to [1], then firmware can be defined as the first code executed in an embedded system, which is stored in a Non-Volatile Memory (NVM) such as Flash or Read-Only Memory (ROM). Among the objectives of the firmware may be to provide the final standalone application software of the system (like in simple Micro-Controller Units (MCUs)), but in most cases, one of the main tasks of the firmware is to retrieve the application code from an external peripheral.

In embedded systems that run a full operating system, the firmware typically works as a boot loader or fetches the boot loader from some pre-defined peripheral device. According to [1], the boot loader is an application to fetch the OS or a standalone application from an external device. Boot loaders are also used in simple embedded systems that do not run an OS. For instance, MCUs include a boot section in their non-volatile Flash memory, whose main purpose is to support firmware updates.

Boot loader technology is constantly evolving to improve features such as performance [2,3], reliability [4], development effort [5] and portability [6]. In general, the scheme of the boot process in complex SoCs is similar to general-purpose computer systems. Initially, designs lacked the ability to update firmware, as it was stored in ROM. In such systems, only the secondary software loaded by the firmware could be customized.

In this situation, the boot loader code is divided into *stages*, where the role of each stage is the initialization of some peripherals that are necessary to obtain the next stage from another peripheral. The firmware only implements the first stage of the boot-up

process, which is commonly referred to as the *preloader*. A similar procedure is used in general-purpose computer systems, but currently the ROM-BIOS is upgradeable. The main differences in the boot-up process for MCUs, SoCs, and general-purpose computer systems are summarized in Figure 1.



**Figure 1.** Comparison of preloader boot-up steps.

The majority of embedded systems based on System-on-Chip use Linux as their operating system due to its flexibility and support for a wide range of architectures. The Linux system exposes a set of well-defined system-callable subroutines to manage peripherals. These subroutines are implemented in the OS as specific device drivers for each peripheral. Over these drivers and on peripherals that can work as storage elements, file systems are built. File systems are a fundamental part of the OS and an almost mandatory requirement to run applications in most SoC solutions. The file systems store both the application programs and also the OS components in the form of executable files, libraries, configuration files, etc. Storing these resources in a file system greatly simplifies application and OS updates including most of the boot loader stages.

In order for the boot loader to be able to fetch the kernel, it needs to include support to read the file system type used in the underlying device. Due to the limited resources available to the first stage of the boot loader, only simple and lightweight file systems may be read from this stage. For this reason, the old and limited File Allocation Table (FAT) file system is widely used in embedded computing systems. But FAT has important deficiencies, the most important being as follows: (1) when used on Flash memory devices, the Flash is quickly degraded, since the file metadata table is frequently updated and the same Flash pages are recurrently written; (2) FAT consistency is compromised for unfinished or failed write operations. To overcome the limitations of FAT more capable later stages of the boot loader have to be added that support more robust and complex file systems.

The current performance of Flash memory has driven SoC designs to adopt it as the main mass-storage technology. Traditional limitations of Flash memory have been overcome by the addition of the so-called Flash Translation Layer (FTL). FTL is a hardware/software driver that makes Flash memory linear, appearing to the system like a disk drive that emulates a block device [7]. FTLs are present in most commercial Flash mass-storage devices [8].

This paper presents NanoBoot, a hardware boot loader which is capable of booting an OS (Linux) or a standalone application from Flash removable media (SD card) using a lightweight file system for resource-restricted IoT devices, which are modeled in this paper as SoCs implemented in Field-Programmable Gate Array chips (FPGA/SOCs). The

main goal to achieve with NanoBoot is to develop a fast, full hardware, generic and open source boot loader with a small footprint capable of booting an SO or standalone application. Linux was chosen because it has become the most used OS in SoC systems, and SD cards are an adequate solution to hold the system software in an SoC because of its low cost, high availability, and easiness to provide updates and fixes along the lifetime of the product. Nevertheless, many ideas and contributions in this paper may be extended to other operating systems or storage devices. The core of NanoBoot is the *NanoFS* file system introduced in [9] and now extended in this work. NanoFS is a lightweight file system that is suitable for implementation in hardware due to its small footprint.

The paper is organized as follows: Section 2 includes some background about booting Linux on SoCs and the role of file systems in the boot-up process; Section 3 describes the design and internal layout of the NanoFS file system in detail; the design of NanoBoot is developed in Section 4; Section 5 details the main results of the proposed approach and compares it to the standard solutions. Conclusions are summarized in Section 6.

## 2. Background

This paper focuses on booting a Linux system from removable Flash media. The following subsections outline some background about the different technologies involved, such as Flash media storage, FPGA SoCs, and the role of file systems in the Linux boot process.

### 2.1. SoC Booting from Flash Media

In modern SoCs, most firmware preloaders include support for a removable storage setup mode that permits obtaining the first boot loader stage from sources such as NAND Flash, SD/TF Card, eMMC, NOR Flash, USB, etc. The most common type of removable Flash storage is NAND-based, which has certain limitations. These include large page data, asymmetric read/write times, and low write-cycle endurance. Despite the limitations, NAND Flash mass storage reduces the cost of the embedded system. Currently, most NAND Flash is manufactured with an FTL that assumes the use of a file system. Therefore, most devices include FTL features such as garbage collection and a fine-grained address translation layer to efficiently manage the Flash memory [8,10,11]. Thus, today, standard consumer-grade Flash devices are ready to drive file systems with acceptable endurance and have adequate performance to boot up an OS and act as the main storage for the OS at run-time. These devices are optimized to be used with existing file systems (e.g., FAT, ext4, NTFS, exFAT, etc.) without any modifications. In this way, the Secure Digital (SD) standard [12] of the SD Association is one of the most widely used storage devices in general consumer electronics. SD cards are supported by many SoC boot loaders and have great compatibility with desktop computers, allowing the user to update or modify the system's software and/or data in a friendly way.

For testing purposes, FPGA devices are a good choice because they are flexible and allow custom hardware and software development. Most FPGA vendors include ARM cores in their high-end products, offering a powerful and very flexible SoC platform for the development of embedded systems. There is also a range of processors that can be used on FPGA chips as soft cores: RISC-V [13], Nios [14], Microblaze [15], OpenRisc [16], Leon, OpenSparc, LowRisc, etc. Some of these processors are distributed under an open hardware license and can be synthesized on FPGA chips from different manufacturers. Since custom hardware can be included in FPGA/SOCs, these platforms offer new options for boot loader design and optimization, especially when the SoC is based on a soft-core processor. There exists the possibility to include boot-loading-specific hardware or to develop a full-hardware boot-loading solution, which is proposed later in this paper.

For this work, two soft-core processors have been selected to exemplify the design and implementation of a full-hardware boot-loading solution, OpenRisc and Microblaze, since both are 32-bit microprocessors capable of running Linux. OpenRISC is an open source microprocessor developed by the OpenCores community [17], implemented using a GCC-based toolchain and written in Verilog HDL synthesizable code. Its main characteristics



are as follows: 32/64-bit load/store Reduced Instruction Set Computer (RISC) architecture with 32-bit uniform-length instruction format; virtual memory support through a Memory Management Unit (MMU); internal Wishbone B3 bus [18], which is an open specification. Instead, MicroBlaze is a soft-core processor designed by AMD Inc. and distributed as Intellectual Property Core (IP Core) under a proprietary license. It is a 32-bit RISC Harvard architecture processor, and the main difference compared to OpenRISC is the internal Advanced eXtensible Interface 4 (AXI4) [19] bus used. Both microprocessors can be synthesized on mid-range AMD devices, leaving plenty of spare FPGA resources to add extra internal peripherals needed to support the Linux boot-up process from removable media. FPGA/SOC booting will be discussed in detail in Section 4.

## 2.2. Embedded Linux

Despite the available alternatives, Linux is the preferred OS for SoC development. Although Linux was originally created to run on general-purpose computers, it is extremely flexible and configurable and supports a very wide range of computer architectures [20], making it ideal for custom embedded computing.

Linux for embedded systems has some specific features compared to Linux for general-purpose computers, especially related to the boot-up process. Embedded Linux covers a wide range of devices starting at small embedded systems with about 4 MB of ROM and 8MB of RAM [20] and a variety of custom boot loaders are available for these kinds of device.

The Linux boot-up process for both general-purpose computers and embedded systems is divided into three general stages [21]: *boot loader*, *kernel startup*, *user space initialization*. The *boot loader* stage is fully dependent on the platform, and although a custom boot loader is needed for each type of hardware, once the process reaches the *kernel startup* stage, the boot-up process is similar on all platforms. The Linux *boot loader* stage for current SoCs is thoroughly described in [22] and, despite some differences, its final target always consists of launching an executable Linux kernel image that was previously fetched from an external peripheral and copied to RAM. The differences among different SoC boot loaders are in firmware capabilities and limitations. For instance, the Linux kernel image is compressed in some configurations to reduce the fetch time, and other configurations add an extra RAM disk image that is also loaded before the kernel is launched. Both decompression and RAM disk loading are extra tasks carried out by the firmware. Moreover, in a removable media setup mode, the boot loader must also include the driver to access the external massive storage, increasing the firmware footprint.

Linux boot performance is highly platform-dependent, but the Linux boot process is flexible and can be optimized with multiple techniques, as introduced in [23–25]. In these works, the boot time is reduced by testing multiple platforms and scenarios with different configurations, but the optimization heavily depends on the scheme used, with or without RAM disk. For both boot-up schemes, the *boot loader* stage is divided into multiple sub-stages heavily dependent on the hardware platform, but the Linux community tries to reuse boot loader code pieces while making boot loaders compatible with multiple SoCs. Despite the variety of SoCs and custom boot loaders, Linux has achieved compatibility with a variety of platforms, continuously decreasing the number of custom boot loaders and taking preference for generic and open solutions such as U-Boot and Redboot [20], which are the preferred solutions for embedded systems because they minimize hardware dependency by reusing a generic first stage where it is only necessary to adapt a small piece of code for each platform.

Currently, *Das U-Boot* is the most widely used solution, since, like Linux, it is open software supported by a community and is compatible with a variety of ASIC-based SoCs, Single-Board Computers (SBCs) and FPGA/SOCs. Also, it supports development boards with microprocessors like ARM, AVR32, Blackfin, x86, Motorola 68K, AMD Microblaze, MIPS, Altera Nios, Nios2, PowerPC, Super-H, etc., and it is a good starting point for new boot loader developments [26].



All Linux boot loaders support file systems in some sub-stage of the process, since the Linux kernel image is size-variable and is frequently updated along the device's lifetime. In addition, the availability of the file system improves the update capabilities of the system. *Das U-Boot* follows a standard Linux boot-up process that expects a kernel stored as a regular file in a file system. Thus, *Das U-boot* supports all modern file systems included in current Linux kernels.

### 2.3. File Systems in Embedded Linux

The Linux user space initialization stage is started when a large file system with the user data and applications is mounted, though sometimes it is more complex since multiple file systems may be mounted across the file system hierarchy. In Linux distributions, the kernel image and the RAM disk, if present, are stored in a file system. In this way, the kernel can be replaced or updated by writing or overwriting these files during the Linux run-time. Moreover, in removable setup mode, the kernel can be easily changed from an external desktop computer, but the file system makes the boot loader code complex since it must search for and read the files from a file system at boot time as has already been mentioned. Thus, the performance of the boot loader is heavily dependent on the ability to manage the file system efficiently, and for this reason, the file system selected for the boot process is of the utmost importance to improve the performance of the Linux boot process [24]. However, it is not always possible to change the initial file system used by the boot loader. Some SoC manufacturers integrate file system support into the on-chip firmware. This increases the firmware size since two levels of software drivers must be included: storage medium drivers (SD card, eMMC, etc.) and file system drivers. The size limitations of the firmware make it not possible to support complex file systems like ext4, btrfs and others, or use advanced features like journaling, snapshots, compression, etc. Other SoCs, choose to include only the medium access drivers, accessing the storage medium in raw mode, and leaving file system support for a secondary customizable stage.

In boot loaders, file systems designed without complex data structures are the best option. Therefore, there are some file systems with a simple layout whose implementation has a low software footprint aimed to be used by Linux in the boot-up process, like Read-Only File System (ROMFS) or Compressed ROM File System (CRAMFS). They are valid for embedded systems, but they have some limitations, such as the size of the file system and the size of individual files. In fact, ROMFS is very limited and can only be used for very small kernels and small initial RAM disks. Unlike, CRAMFS (developed by Linus Torvald) is more advanced, including features like EXIP for executable files, while keeping it simple. However, these types of file systems are not applicable to all kinds of embedded systems because any change in the contents, such as a driver or the kernel, requires a full file system image regeneration. With these types of file systems, an upgrade to the kernel or some boot loader stage requires two complex steps: (1) a full file system image regeneration and (2) a replacement of the old image safely. The second step is critical since any failure makes the system unbootable. Thus, it is very common to find firmware with FAT16/FAT32 support mainly for two reasons: FAT has a simple internal layout despite it being an old design and, due to its ubiquity in most computer systems, facilitates the sharing of data across their devices.

As already explained, the removable setup mode is the preferred configuration for general-purpose SBCs and for SoC-based designs in general, with SD cards being the current preferred medium. Most SBC Linux distributions store the kernel in a file system that can be accessed in read/write mode by Linux itself, mounted in the */boot* directory. The main reason for this configuration is to make it easy to change or upgrade the OS using a desktop computer, even auto-update it from the running OS. Custom designs such as those implemented in FPGA/SOCs can be fully customized to support different booting strategies, like booting Linux from SD cards in a single boot stage.

In any scenario, the boot process will benefit from having the kernel and a possible RAM disk image in a file system that is easy to handle from a resource-limited firmware code, for this task, the *NanoFS* has been chosen due to being specifically designed for that.

### 3. Nano File System

NanoFS is a light file system aimed at being used with reduced software or directly embedded in hardware due to its small footprint. It is designed with simple internal data structures to be handled with low-memory resources and has an optimized internal layout to reduce file lookup and file read algorithms. NanoFS is a good choice for the first boot loader stage, even for the kernel fetch stage, since only read operations are used. The internal structure of this file system facilitates the construction of a low-footprint hardware module that can replace the internal firmware for reading files from a NanoFS. Hence, it is a suitable option for use in IoT devices where minimizing area and resources to implement a hardware boot loader is crucial.

At the user level, it is similar to other file systems where the files are organized in a hierarchical structure building a directory tree. Also, at the physical level, it is quite similar to other file systems that use the *logical block addressing* scheme, which consists of dividing the available space into fixed-size blocks called logical block (*l-block*), but in the internal data structures, NanoFS is quite different from other file systems like FAT or Extended series. The main design differences are summarized as follows:

1. Small l-block size: 1 byte and 512 bytes.
2. Reduced types of l-blocks named: *dir node*, *data node* and *metadata node*.
3. Decentralized file system data structure to minimize implementation resources.
4. Internal layout structured around forward lists to simplify file system algorithms implementations.

The following subsections detail the internal layout of NanoFS that is necessary to understand the simplified algorithms for NanoFS operations, such as file lookup and file reading.

#### 3.1. NanoFS Layout

Like other file systems, NanoFS divides the storage device into an array of l-blocks. All l-blocks are referenced by a 32-bit unsigned integer number; thus, the file system is limited to 4G l-blocks. The size of the l-block is set when the file system is created, taking the values shown in Table 1. Valid block size values were chosen to achieve the best performance according to the storage technology used. The relationship between *physical block* and *logical block* size directly impacts file system performance; thus, a primary rule for most file systems is to match their sizes or at least to use a multiple number.

**Table 1.** NanoFS valid block sizes.

s_Blocksize	Block Size	File System Size Limit
0	1 byte	4 G bytes
1	512 bytes	2 T bytes
2	1024 bytes	4 T bytes
3	4096 bytes	16 T bytes

The size of the l-block also impacts the complexity of the software required to manage the file system. In this sense, NanoFS allows small l-block sizes like 1 byte or 512 bytes, designed mainly to be used in devices with limited resources. The 1-byte size is proposed to be used for on-board ROM memories due to minimize the wasted space by the file system's internal structures. The second fits with the SD card protocol block size and is intended to optimize the temporal data used in the file system driver for this storage media.

To reduce the resources to manage the file system, the NanoFS specification includes a small amount of data structures stored in Little-Endian format: *superblock*, *dir node*, *data node*

and *metadata node*. In block 0, the *super-block* record is located that contains key information to manage the file system. The later l-blocks contain data structures called *node* with the following characteristics:

- Each *node* is referenced by a unique 32-bit unsigned integer called *node number*. It corresponds to the l-block where the node starts.
- All l-blocks are part of some *node*. This is required to preserve the file system integrity.
- The file system *nodes* can be of three types: *dir node*, *data node* and *metadata node*.
- *Dir nodes* are intended to compose the file system hierarchy.
- *Data nodes* stores the user data of a file and a header with file system internal data. In this way, the *data node* header is used to navigate to the next allocated *data node* for the current file.
- The *data node* is size-variable and consists of single or multiple consecutive l-blocks. It can be extended to consecutive l-blocks to improve the sequential file read/write performance and to reduce the file system fragmentation.
- *Metadata nodes* are optional, reducing the file system weight if they are not included.

A NanoFS formatted device is built over two lists of *nodes* starting at the superblock. Both are forward lists referenced by the *node numbers* called the *data allocated nodes* and *free nodes*. The whole file system is a forward list that can be navigated keeping only a single file system handle, the current *node number*. Each *node* always contains a record with a *node number* referencing the next *node* in the file system structure. In addition, the *dir nodes* and *data nodes* structures contain the records detailed in Tables 2 and 3, respectively.

**Table 2.** Records in the *dir node* data structure.

Offset (Bytes)	Size (Bytes)	Record	Description
0	1	d_flags	Dir node flags
1	4	d_next_ptr	Node of next dir node
5	4	d_data_ptr	Node of first child node
9	4	d_meta_ptr	Node of first metadata node
13	1	d_fname_len	Length in bytes of filename
14	256	f_name	Name of file
270	34	f_meta	Standard metadata

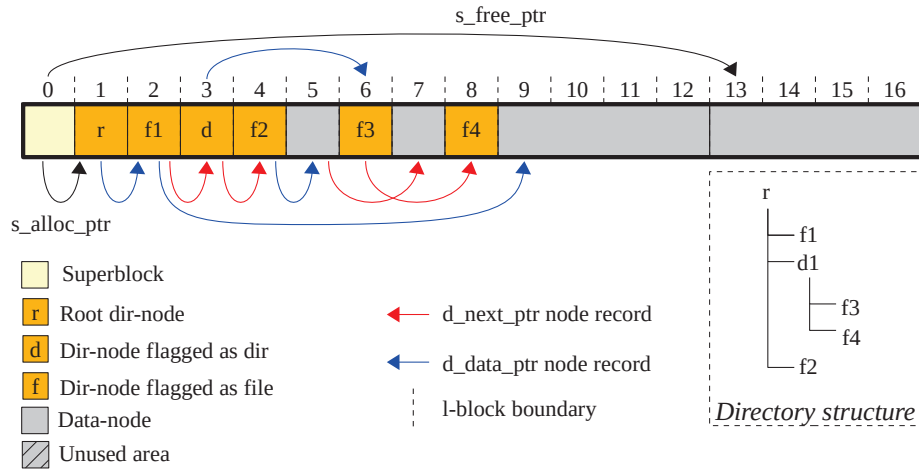
**Table 3.** Records in the *data node* data structure.

Offset (Bytes)	Size (Bytes)	Record	Description
0	4	d_next_ptr	Node of next data node
4	4	d_len	Data length
8	d_len	d_data	Data

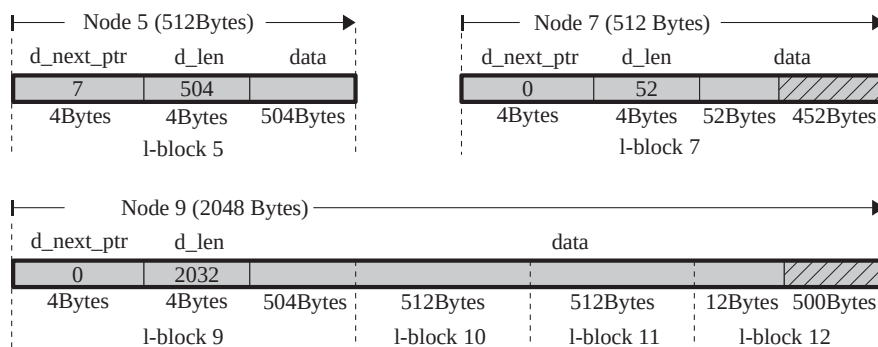
Figure 2 contains an example of a NanoFS formatted device using a 512-byte l-block size. The example has a directory structure that starts in the root directory with one subdirectory and four files (Figure 2a). Additionally, it enumerates the sequence of l-blocks along the device highlighting the *dir nodes*. Starting at *superblock* (l-block 0), the file system has the two lists referenced by its *node numbers*: *s\_alloc\_ptr* references the list of data allocated nodes (l-block 1) and *s\_free\_ptr* references the free nodes list (l-block 13). The file system hierarchy begins at the *root dir node* (l-block 1) and is a list of *dir nodes* linked by two records: *d\_next\_ptr* and *d\_data\_ptr*. The *d\_next\_ptr* record always addresses the next *dir node* in the current directory until the last *dir node* is reached; in the last case, the value is 0 (end mark).

The *d\_data\_ptr* record can point to another *dir node* or to a *data node*. To distinguish it, each *dir node* is flagged in the *d\_flags* record as a file or directory. When a *dir node* is a file, *d\_data\_ptr* addresses the first *data node* of the file. Instead, whether it is flagged as

a directory, it represents a subdirectory and  $d\_data\_ptr$  points to the  $dir$  node where the subdirectory starts. The files are stored in  $data$  nodes that contains a small header followed by the file data, as is detailed in Table 3. A file consists of a single or multiple  $data$  nodes organized in a forward list and a  $data$  node can be extended to multiple consecutive l-blocks.



(a)



(b)

**Figure 2.** NanoFS layout example: (a) directory structure and (b) nodes detail.

Figure 2a has an example of two files with allocated data:  $f1$  and  $f2$ . The  $d\_data\_ptr$  records point to node 9 (l-block 9) and node 5 (l-block 5), respectively. Figure 2b details the nodes for both files, where  $f2$  is a fragmented file in two nodes (5 and 7) and each node fits with a single l-block. Instead,  $f1$  is an unfragmented file with a single node (9) that is expanded through four consecutive l-blocks (from 9 to 12). For large files, read/write operations can be sped up because data are allocated in consecutive physical blocks, especially for storage technologies with accelerated sequential access modes support, such as CMD8 protocol command on SD cards.

NanoFS manages the free nodes in another forward list of  $data$  nodes referenced from the superblock. In Figure 2, the free nodes start at node 13 (l-block 13), referenced by the  $s\_free\_ptr$  record of the superblock, and it has only one expanded  $data$  node (l-blocks 13 to 16). Finally, the intended purpose of the  $metadata$  nodes list is to extend the file system with custom features if a specific application requires them. Consequently, they are not mandatory and are not covered in this work.

### 3.2. NanoFS Implementations

The complexity and resources required for the implementation of file system operations depend on the internal layout, internal data structures, and the storage media used. The relationship between the file system l-block size not only affects the performance, but there is also a direct relationship between the resources required for the read/write algorithms. There are two recommended rules that are generally accepted to achieve optimal behavior: (1) set the l-block size to a multiple of the physical block size, (2) set the start offset of the file system to be aligned with a physical block, especially for devices with a large physical block size, such as Flash memory.

Referring to SD cards, they work with a serial bit stream protocol in 1-bit or 4-bit transference mode. The SD protocol also includes a 1-bit SPI mode, which is a reduced subset of the protocol aimed at supporting limited host devices. In both modes, the protocol assumes a 512-byte block size for data transference; therefore, the SD optimal l-block size for a partition formatted with NanoFS is this size. When matching the l-block size, the procedures to manage NanoFS are optimal, with the result of a simplified algorithm that is able to run with very-low-memory resources.

In this way, Algorithms 1 and 2 depict a possible implementation for file lookup and file read procedures. The file lookup procedure can be implemented using a single handle that is the current *node number* (a 32-bit unsigned integer). It can be seen in Algorithm 1, which describes a non-recursive file lookup starting at the root directory. The file lookup ends once the file is located; after that, the *d\_data\_ptr* record has the *dir node* number of the file, which will be used later as a handle to read the file data in the read procedure.

---

#### Algorithm 1 Lookup file procedure

---

```

procedure LOOKUP(file_name)
  handle  $\leftarrow$  root_dir_node.d_data_ptr
  while handle  $\neq$  0 do
    d_flags  $\leftarrow$  get_bytes(handle,1)
    if (d_flags  $\wedge$  0x07) = 0x01 then
      ▷ dir_node is a file
      if match(file_name) then
        break
      end if
    else
      ▷ navigate to next dir node
      handle  $\leftarrow$  get_bytes(handle,4)
    end if
  end while
  return handle
end procedure

```

---



---

#### Algorithm 2 Read file procedure

---

```

procedure READ_FILE(dir_node)
  handle  $\leftarrow$  dir_node.d_data_ptr
  while handle  $\neq$  0 do
    d_next_ptr  $\leftarrow$  get_bytes(handle,4)
    d_len  $\leftarrow$  get_bytes(handle,4)
    read_data(handle,d_len)
    handle  $\leftarrow$  d_next_ptr
  end while
end procedure

```

---

Hence, the Algorithm 2 depicts the read procedure taking the previous handle (*d\_data\_ptr*) and needs two additional 32-bit numbers to read the file content. Both procedures call the

*get\_bytes* function taking as the second argument  $N$ , returning only  $N$  bytes from the storage media. This function is optimized for the SD protocol since the read operation captures the data from the input stream, and a memory buffer of 512 bytes is not necessary.

Though write support is not required for boot loaders, it is used to prepare NanoFS partitions on external storage media. For Linux, the available NanoFS version is implemented using the *Filesystem in Userspace* (FUSE) [27] interface and supports read/write operations. The NanoFS fuse driver is intended to configure the SD cards used in the boot loaders tested in the following sections.

#### 4. NanoBoot Implementation

This section describes the implementation of NanoBoot and analyzes how the use of NanoFS simplifies the complexity of the boot loader when applied to booting FPGA/SOCs. The objective is to minimize the number of stages and the complexity of booting a Linux kernel from a NanoFS partition. Since the hardware in FPGA/SOCs can be customized, it can be used to implement the boot loader in hardware. It will boot a Linux kernel in a single stage that will run on a soft-core processor. The SD card removable media is used as a convenient storage device as explained in previous sections.

The selected testing environments are two full FPGA/SOCs: OpenRISC 1200 and Microblaze 9.6. Both are intended to be implemented into AMD FPGA chips and deployed on prototype boards with at least 128MB of DDR2 off-chip memory, which is enough to run an OS such as Linux. The implementation results are presented later in Section 5.1, while the NanoBoot design is fully detailed in this section. The hardware is developed in SystemVerilog Hardware Description Language (HDL), and the source code is available on the authors' Github [28].

The OpenRISC SoC in Figure 3 shows a block diagram with the complete proposed SoC to test the new boot loader procedure. It includes the processor and some controllers for off-chip peripherals that connect to a Wishbone bus. Wishbone defines a master-slave interface for connected devices. In this configuration, the OpenRISC processor and the *Boot Loader Unit* work as masters and the DDR controller has a slave role. When the boot process is triggered, the *Boot Loader Unit* halts the microprocessor (asserting its reset signal) and becomes the master of the bus in order to use the DDR controller in exclusive mode. Then, the boot loader reads the Linux kernel from the SD card and copies it to a specific address in the DDR memory. After that, the *Boot loader Unit* de-asserts the reset signal of the microprocessor and goes into an idle state while the processor starts executing the OS kernel.

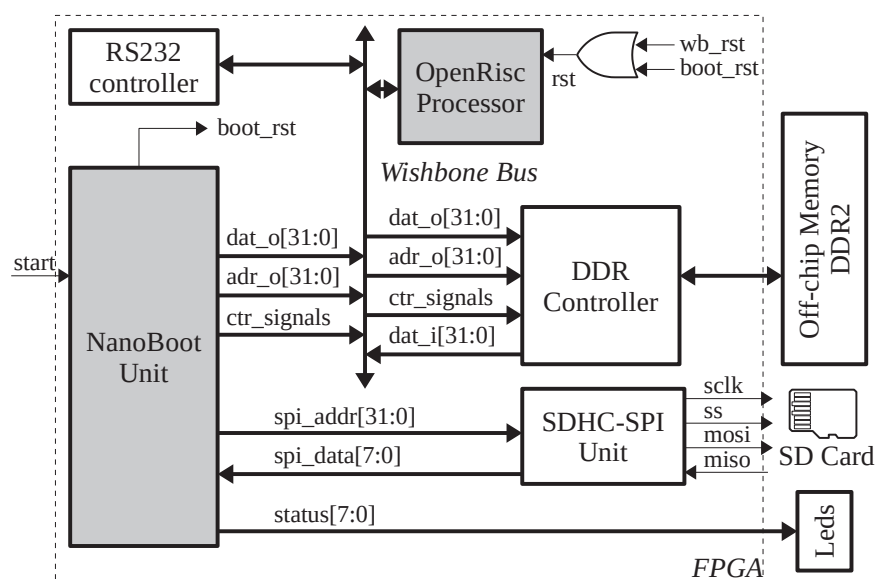


Figure 3. OpenRISC boot loader architecture.



The boot loader process depends on the hardware architecture, the included peripherals, and the behavior of the processor after a reset. Despite this, the proposed SoC with Microblaze is very similar to the one presented above, and a minimal set of changes is needed to support the new processors. Although both processors have the same peripherals, some changes are necessary: the internal bus used with Microblaze is AXI4, and both microprocessors must be reconfigured to set the *program counter register* (PC) to point to DDR memory after reset. Concerning the DDR controller, it was generated by the *Memory Interface Generator* (MIG) tool included in the Vivado 2019.1 suite and is compatible with the AXI4 bus protocol used in the MicroBlaze architecture. The same DDR controller is used in OpenRISC with a minor change that consists of adding a bridge with some glue logic.

Another peripheral added to the SoCs is an SD controller (*SDHC-SPI Unit*) [29] which is used by the *Boot Loader Unit* to boot up from a removable media. This unit is a light SD card host controller that implements part of the SD-SPI protocol specified in [12] since the boot loader only performs read operations. The *Boot Loader Unit* assumes that the SD card is prepared using a classic Master Boot Record (MBR) with the first partition formatted with NanoFS.

Also, for debugging purposes, some LEDs on the development board were connected to the boot loader status register and an RS232 controller was included to obtain a Linux terminal and check whether the boot process was successful.

Figure 4 shows the block diagram of the *Boot Loader Unit*. Internally, it is designed using multiple Finite State Machine (FSM) described using the Verilog HDL. The Boot Loader Unit divides the process into four steps:

1. Check the MBR and obtain the address of the first partition.
2. Check the first partition for a valid NanoFS file system.
3. Look up the file *vmlinuz*.
4. Copy *vmlinuz* to DDR memory.

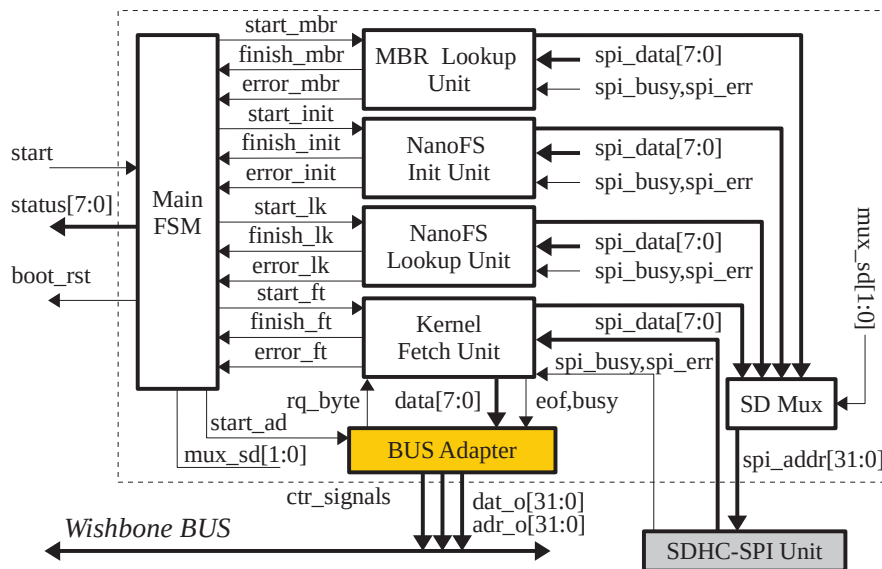


Figure 4. OpenRISC NanoBoot unit detailed.

In this way, the design is split into the four FSMs depicted in the figure: *MBR Lookup Unit*, *NanoFS Init Unit*, *NanoFS Lookup Unit* and *Kernel Fetch Unit*. The four FSMs are governed by the *Main FSM* which starts each one sequentially and covers the full boot loader process. FSMs are controlled by a handshake protocol with three signals connected to the *Main FSM* unit. Each step is triggered when its signal *start\_\** is asserted and ends when its output signals are asserted: the *finish\_\** signal on success; or the *error\_\** signal when error. The *error\_\** signals are captured in a status register to check for possible errors during the process. Internally, the FSMs only need a reduced set of resources: a few 32-bit

registers for file system handlers and a small additional ROM to store the file name looked up (vmlinuz).

Figure 5 presents a flow diagram of the *Main FSM* including the main signals that govern the other four FSMs. Additionally, a flow diagrams for the *MBR Lookup Unit*, *NanoFS Init Unit*, *NanoFS Lookup Unit* and *Kernel Fetch Unit* are depicted in Figures 6 and 7. As well, in the next paragraphs, the design and internal operation of the *Boot Loader Unit* for OpenRISC are described in detail, followed by the necessary modifications to use it with Microblaze.

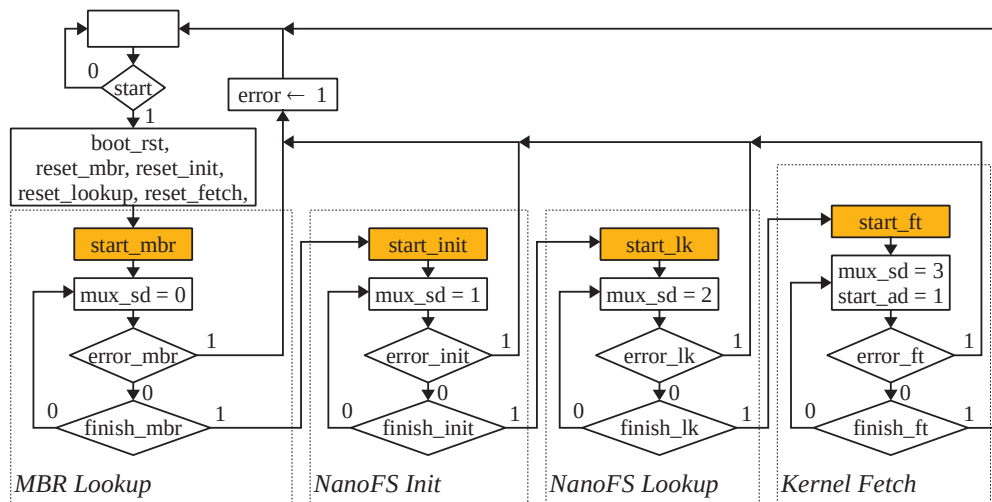
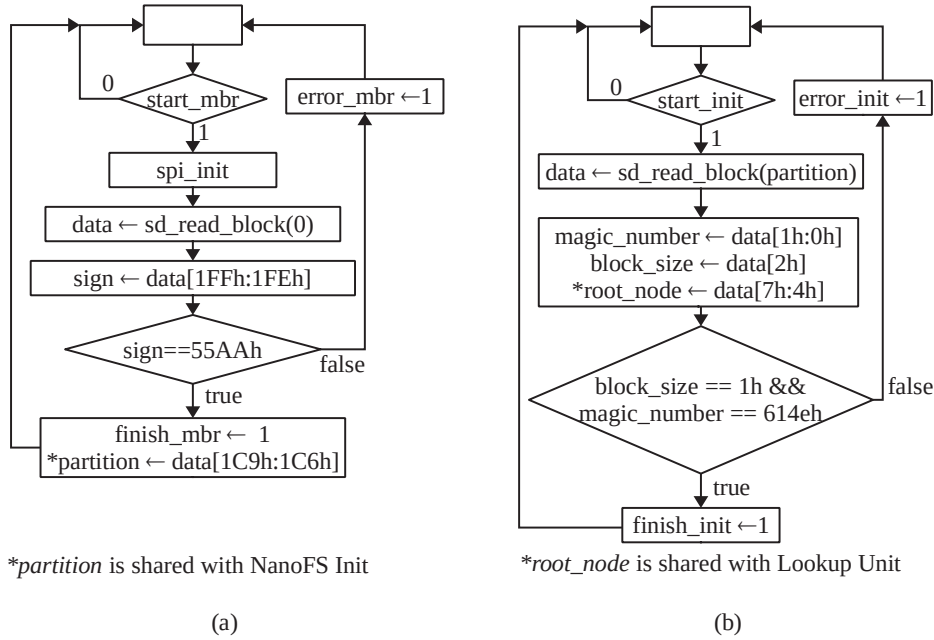


Figure 5. Flow diagram of the Main FSM.

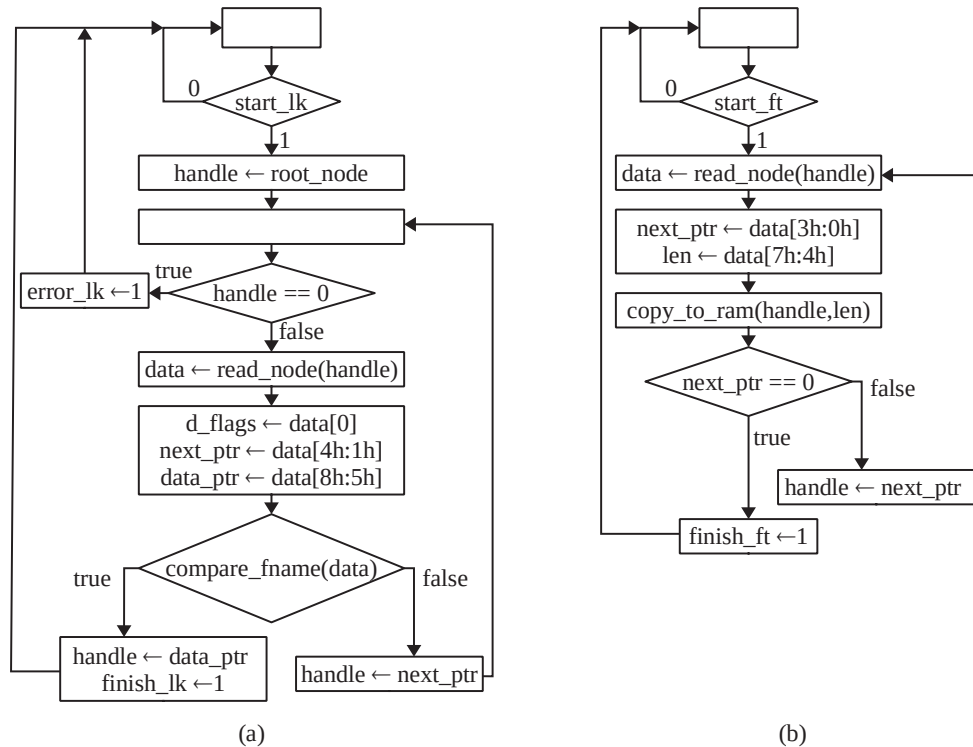
The OpenRISC boot process is detailed as follows:

1. As depicted in Figure 5, the boot process starts after the external signal *start* is asserted. Then the *Boot Loader Unit* asserts the *boot\_rst* and *start\_mbr* signals. The first signal stops the microprocessor which remains halted until the last step; hence, the *Main FSM* handles the boot process. The second starts the FSM *MBR Lookup Unit*.
2. The *MBR Lookup Unit* performs two tasks: it initializes the SD card controller and reads block 0 of the SD card to analyze the MBR. The MBR is valid when it has the right 16-bit signature value at address 1FEh. In that case, the first partition offset is the 16-bit value located at offset 1C6h. This last value is saved in a register to be used in the next stage to access a NanoFS-formatted partition. Figure 6a shows the FSM functional flow diagram for this unit which saves the first partition offset in a register called *partition*.
3. The next step is implemented in the *NanoFS Init Unit* and described in Figure 6b. This FSM checks the first partition for a valid NanoFS file system. It reads the NanoFS superblock at a partition offset of 0 to compare the magic number (614Eh) and, when successful, it obtains the file system handler of the root directory. The file system handler is a 32-bit value stored at offset 04h, which will be used in the next stage.
4. The third FSM is the *NanoFS Lookup Unit* and it internally implements the Algorithm 1 that looks up the file *vmlinuz* along the root directory of the file system. As can be seen in the flow diagram of Figure 7a, this unit starts at the root directory and navigates through the directory structure until the file is located. When this stage is successfully completed, the file system handler points to the start of the file and it will be used in the next step.
5. The last step is the *Kernel Fetch Unit* FSM that implements the Algorithm 2 to transfer the *vmlinuz* file directly to DDR memory through the *Memory Bus Adapter*. This adapter acts as a Wishbone bus master device, taking control of the DDR controller until the kernel is fully transferred. The flow diagram of this unit is depicted in Figure 7b.

6. Finally, once the *vmlinuz* file has been copied to DDR memory, the microprocessor's reset signal is de-asserted and the execution starts at the reset interruption vector, launching the kernel.



**Figure 6.** Flow diagram of (a) MBR Lookup Unit and (b) NanoFS Init Unit.



**Figure 7.** Flow diagram of (a) NanoFS Lookup Unit and (b) Kernel Fetch Unit.

The last FSM (*Kernel Fetch Unit*) takes control of the microprocessor bus to write to the DDR memory through the *Bus Adapter* component highlighted in Figure 4. In general, for each SoC, only the *Bus Adapter* module must be adapted because the differences between the architectures are mainly in the internal bus. For Microblaze, the *Boot Loader Unit* is

reused with minimal changes, which consists mainly of defining the bus control signals and conversion of the byte order from Big-Endian to Little-Endian due to the AXI4/AXI4-Lite [19] bus used by Microblaze.

## 5. Results

This section includes the resource utilization for the OpenRISC and Microblaze FPGA/SOC implementations described in the previous section. It also presents a functional and performance test of the NanoBoot implemented in both FPGA/SOCs.

### 5.1. NanoBoot Resources Implementation Results

The implementation of the two FPGA/SOCs presented in Section 4 has been tested on the prototype boards shown in Table 4. All boards include an AMD FPGA chip, off-chip DDR memory, and several peripherals. Nexys4 DDR [30] and Arty [31] are boards from Digilent Inc. (Pullman, WA, USA) with a mid-grade FPGA chip: XC7A35T and XC7A100T [32], respectively. Both are intended to prove the feasibility of the proposed design in a scenario with limited resources. Instead, the Genesys2 prototype board [33], also from Digilent Inc., includes a high-grade AMD Kintex-7 (XC7K325T) [32] chip, intended to demonstrate the small footprint of the hardware boot loader when implemented in state-of-the-art devices.

**Table 4.** Prototype boards used to test PSoCs.

Name	FPGA Chip	RAM Type	RAM Size
Arty	Artix-7 (XC7A35T)	DDR3	256 MB
Nexys4 DDR	Artix-7 (XC7A100T)	DDR2	128 MB
Genesys 2	Kintex-7 (XC7K325T)	DDR3	1 GB

Tables 5 and 6 summarize the resource utilization of the entire PSoCs and its main components for the XC7A35T, XC7A100T, and XC7K325T chips. The peripherals and the glue logic at the top level are not included in the tables, hence the difference between the full SoC and the sum of the parts. Microblaze and OpenRISC SoCs have been synthesized for these chips using the AMD Vivado 2019.1 design suite with the following options: *Synthesis strategy = Area Optimized-high* and *Implementation strategy = Flow RunPhysOpt*.

**Table 5.** OpenRISC 1200 SoC resource utilization by FPGA.

FPGA	Parts	Slices	Slices Reg.	LUTs	Slices (%)
XC7A35T	Full SoC	3317	8200	10,682	40.70
	NanoBoot	330	837	649	4.05
	DDR Controller	1837	4996	5479	22.54
	OpenRISC	1177	2020	4135	14.44
XC7A100T	Full SoC	3593	7937	9508	22.67
	NanoBoot	315	855	593	1.99
	DDR Controller	1974	4767	5154	12.45
	OpenRISC	1186	1939	3387	7.48
XC7K325T	Full SoC	3871	8252	10,579	7.60
	NanoBoot	320	857	613	0.63
	DDR Controller	2320	5090	6165	4.55
	OpenRISC	1116	1950	3398	2.19

It can be seen that the NanoBoot unit takes 4.05% and 3.93% of the FPGA resources in an OpenRISC SoC and a Microblaze SoC, respectively, on a mid-grade chip (XC7A35T), making it adequate for systems with limited hardware resources. On a high-grade chip (XC7K325T), resource utilization has a nearly negligible impact ( $\approx 0.7\%$ ).

It is interesting to note that the significant resource difference for the DDR controller in Tables 5 and 6 is because Vivado uses different IP cores for each FPGA chip.

**Table 6.** MicroBlaze 9.6 SoC resource utilization by FPGA.

FPGA	Parts	Slices	Slices Reg.	LUTs	Slices (%)
XC7A35T	Full SoC	3712	10,279	10,482	45.55
	NanoBoot	320	871	604	3.93
	DDR Controller	1555	4049	4711	19.08
	Microblaze	1098	2904	3099	13.47
XC7A100T	Full SoC	5406	14,691	14,587	34.11
	NanoBoot	353	837	752	1.19
	DDR Controller	1346	3051	3859	5.18
	MicroBlaze	1264	3090	3245	4.67
XC7K325T	Full SoC	6084	15,952	16,645	11.94
	NanoBoot	367	858	771	0.72
	DDR Controller	2811	6525	8591	5.52
	Microblaze	1137	2829	3071	2.23

## 5.2. Boot Loader Verification on OpenRISC and Microblaze FPGA/SOCs

Functional verification consists of booting a Linux kernel in both FPGA/SOCs on the three prototype boards introduced previously in this section. For this purpose, a Linux kernel has been compiled and stored in an SD card partition formatted with NanoFS. The kernel used is Linux 4.4.0 released on 10 January 2016 for OpenRISC and Linux 4.19.0 released on 22 October 2018 for Microblaze. Both have been compiled with default options using a cross-compiler, obtaining a size of 6.2 MB for OpenRISC and 8.3 MB for Microblaze.

The compiled kernel has been copied to a file named *vmlinuz* in the root directory of a NanoFS partition, where the *Boot Loader Unit* expects to find it. After the boot process is finished, a Linux console is opened in the serial port included in both SoCs that serves to check the correct system function. The functional verification has been carried out on all chips listed in Table 4 with a wide variety of SD cards.

The time required by the *Boot Loader Unit* to load the kernel into DDR memory (fetch time) is shown in Table 7. Time measurements were taken using a logic analyzer connected to two General Purpose InputOutput (GPIO) pins. The first GPIO pin is asserted after a hardware reset and the second when the *Boot Loader Unit* has copied the kernel to DDR memory. This fetch time adds to the platform's configuration time, that is, the time elapsed to load the FPGA bitstream from ROM, but this time is approximately two orders of magnitude smaller than the fetch time ( $\approx 90$  ms for Arty,  $\approx 150$  ms for Nexys4 and  $\approx 200$  ms for Genesys2) and has not been considered in Table 7. The fetch time has been measured with Class 10 SD cards, but the results shown in Table 7 are only for the *Sandisk Ultra UHS-I Class 10* model. Table 7 contains the results with the clock frequencies set to the same value on all SoCs. It makes the fetch time to be similar for the three platforms, showing that it is not dependent on the platform. The fetch time difference between processors is mainly due to the different sizes of the kernel to be loaded.

The fetch time can be improved when the platforms are configured with their optimal clock frequencies. These results are summarized in Table 8. This configuration improves the performance of the boot loader on the XC7K325T chip: 34% for MicroBlaze and 48% for OpenRISC. Although the DDR speeds are different in different boards, the fetch time depends mainly on the SD card read speed, since the SD-SPI protocol is a serial protocol limited to 25 MHz.

**Table 7.** Linux kernel fetch time (initial configuration).

FPGA	Processor	System Clk (MHz)	SD Clk (MHz)	DDR Clk (MHz)	Fetch Time (s)
XC7A35T	OpenRisc	83	20.75	333.33	3.87
	MicroBlaze	83	20.75	333.33	5.01
XC7A100T	OpenRisc	83	20.75	333.33	3.88
	MicroBlaze	83	20.75	333.33	5.00
XC7K325T	OpenRisc	83	20.75	333.33	3.86
	Microblaze	83	20.75	333.33	4.96

**Table 8.** Linux kernel fetch time (optimal configuration).

FPGA	Processor	System Clk (MHz)	SD Clk (MHz)	DDR Clk (MHz)	Fetch Time (s)
XC7A35T	OpenRisc	83	20.75	333.33	3.87
	MicroBlaze	83	20.75	333.33	5.01
XC7A100T	OpenRisc	100	25	200	3.22
	MicroBlaze	100	25	200	4.17
XC7K325T	OpenRISC	200	25	800	2.60
	Microblaze	200	25	800	3.70

The complete *Boot time* is the time elapsed until the Linux kernel starts the *init* process (process 0). It has been measured for the three platforms and the results are shown in Table 9. This time is the sum of *Fetch time* and *Kernel boot* and is measured by using a logic analyzer that monitors the GPIO pin that is asserted when the *init* process starts.

**Table 9.** Linux kernel boot time.

FPGA	Processor	System Clk (MHz)	Fetch Time (s)	Kernel Boot (s)	Boot Time (s)
XC7A35T	OpenRISC	83	3.9	0.06	3.96
	MicroBlaze	83	5.0	1.1	6.10
XC7A100T	OpenRISC	100	3.2	0.05	3.25
	MicroBlaze	100	4.2	1.0	5.20
XC7K325T	OpenRISC	200	2.6	0.04	2.64
	MicroBlaze	200	3.7	0.8	4.50

Most factory solutions for FPGA/SOCs do not support booting from an SD card, since they are intended to boot from an internal Block RAM (BRAM) or using Joint Test Action Group (JTAG) to transfer the kernel. Thus, the closest to a standard solution is *PetaLinux* [34] from AMD, which is used as a reference to compare with NanoBoot. Petalinux has been designed to boot a Linux system on a Microblaze processor from an SPI Flash memory. Petalinux is a non-free system, but it uses a conventional boot loader based on U-Boot which is preprogrammed as internal firmware (FPGA BRAM) that acts as a preloader. The Petalinux boot process has multiple stages, starting at the BRAM preloader, which obtains the next stage from an off-chip Flash memory.

A Microblaze system booted using Petalinux with the kernel stored in the on-board SPI Flash memory has been deployed on the three prototype boards. The Petalinux preloader is not able to use any file system; therefore, the kernel is stored in raw mode in the off-chip SPI Flash. The Petalinux kernel used is a Linux 5.15.36 with 10.4 MB size, similar to



the one used in the OpenRISC and Microblaze SoCs are used as examples in this paper. Table 10 compares boot time with the measures taken using the same GPIO-based method described above, consisting of asserting a GPIO when the kernel launches the *init* process. The *OpenRISC* and *Microblaze* rows correspond to results using the proposed hardware boot loader using NanoFS, while the *Petalinux* rows correspond to the factory solution using Petalinux.

The results obtained in Table 10 clearly show that the use of a full hardware boot loader with a file system does not have a negative impact on the OS boot time compared to the standard Petalinux AMD solution. Moreover, the boot times of the hardware solutions are much better with the added benefits of booting from a file system.

**Table 10.** Power-on time.

SoC	Boot Time (s)
Arty—OpenRISC	4.0
Arty—Microblaze	6.1
Arty—Petalinux	42.6
Nexys4—OpenRISC	3.3
Nexys4—MicroBlaze	5.2
Nexys4—Petalinux	36.4
Genesys2—OpenRISC	2.6
Genesys2—Microblaze	4.5
Genesys2—Petalinux	51.2

## 6. Conclusions

This paper introduces NanoBoot, a novel boot loader for embedded devices that uses the NanoFS light file system on removable SD card media. The simplicity of NanoFS makes it possible to boot a Linux system using a full-hardware boot loader in FPGA/SOCs and offers a convenient and easy procedure for system OS updates.

Full-hardware boot-loading capabilities are demonstrated with OpenRISC and Microblaze FPGA/SOCs. The hardware resources needed by the boot loader are low (about 7% of the whole system) and the boot time is greatly reduced compared to the standard solution using Petalinux on SPI Flash.

**Author Contributions:** Conceptualization, P.R.-d.-C., G.C.-Q. and J.J.; methodology, P.R.-d.-C., G.C.-Q. and J.J.; software, P.R.-d.-C. and G.C.-Q.; validation, P.R.-d.-C., G.C.-Q., J.J., M.J.B., J.V.-C., D.G. and E.O.; formal analysis, P.R.-d.-C. and G.C.-Q.; investigation, P.R.-d.-C., G.C.-Q. and J.J.; resources, M.J.B., J.V.-C., D.G. and E.O.; data curation, P.R.-d.-C. and G.C.-Q.; writing—original draft preparation, P.R.-d.-C., G.C.-Q. and J.J.; writing—review and editing, P.R.-d.-C., G.C.-Q., J.J., M.J.B., J.V.-C., D.G. and E.O.; visualization, P.R.-d.-C., G.C.-Q. and J.J.; supervision, P.R.-d.-C., J.J. and M.J.B.; project administration, J.J., P.R.-d.-C. and M.J.B. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been funded by the Ministry for Digital Transformation and Public Function through grant USECHIP (TSI-069100-2023-001) of PERTE Chip Chair program, funded by European Union—Next Generation EU.

**Data Availability Statement:** The data presented in this study are openly available in [NanoBoot] [<https://github.com/germancq/NanoBoot>] (accessed on 20 July 2024).

**Conflicts of Interest:** The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## Abbreviations

IoT	Internet of Things
SoC	System-on-Chip
NVM	Non-Volatile Memory
ROM	Read-Only Memory
MCUs	Micro-Controller Units
OS	Operating System
FAT	File Allocation Table
FLT	Flash Translation Layer
FPGA	Field-Programmable Gate Array
SD	Secure Digital
RISC	Reduced Instruction Set Computer
MMU	Memory Management Unit
IP Core	Intellectual Property Core
AXI4	Advanced eXtensible Interface 4
ROMFS	Read-Only File System
CRAMFS	Compressed ROM File System
<i>l-block</i>	logical block
HDL	Hardware Description Language
MBR	Master Boot Record
FSM	Finite State Machine
GPIO	General Purpose InputOutput
BRAM	Block RAM
JTAG	Joint Test Action Group

## References

1. Sloss, A.; Symes, D.; Wright, C. *ARM System Developer's Guide: Designing and Optimizing System Software*; Elsevier Science: Amsterdam, The Netherlands, 2004.
2. Lewandowski, M.; Orczyk, T.; Porwik, P. Dedicated AVR bootloader for performance improvement of prototyping process. In Proceedings of the 2017 MIXDES—24th International Conference Mixed Design of Integrated Circuits and Systems, Bydgoszcz, Poland, 22–24 June 2017; pp. 553–557.
3. Bogdan, D.; Bogdan, R.; Popa, M. Design and implementation of a bootloader in the context of intelligent vehicle systems. In Proceedings of the 2017 IEEE Conference on Technologies for Sustainability (SusTech), Phoenix, AZ, USA, 12–14 November 2017; pp. 1–5.
4. Kang, Y.; Chen, J.; Li, B. Generic Bootloader Architecture Based on Automatic Update Mechanism. In Proceedings of the 2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP), Shenzhen, China, 13–15 July 2018; pp. 586–590.
5. Nuratch, S. A serial Bootloader with IDE extension tools design and implementation technique based on rapid embedded firmware development for developers. In Proceedings of the 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), Siem Reap, Cambodia, 18–20 June 2017; pp. 1865–1869.
6. Sha, C.; Lin, Z.Y. Design Optimization and Implementation of Bootloader in Embedded System Development. In Proceedings of the 2015 International Conference on Computer Science and Applications (CSA), Wuhan, China, 20–22 November 2015; pp. 151–156.
7. Intel, C. Understanding the Flash Translation Layer (FTL) Specification. 1998. Available online: [http://dooroo.org/download\\_documents/FTL\\_INTEL.pdf](http://dooroo.org/download_documents/FTL_INTEL.pdf) (accessed on 10 October 2023).
8. Boukhobza, J.; Olivier, P. Chapter 7—Flash Translation Layer. In *Flash Memory Integration*; Elsevier: Amsterdam, The Netherlands, 2017; pp. 129–147. [CrossRef]
9. de Clavijo, P.R.; Ostua, E.; Juan, J.; Bellido, M.J.; Viejo, J.; Guerrero, D. NanoFS: A hardware-oriented file system. *Electron. Lett.* **2013**, *49*, 1216–1218. [CrossRef]
10. Wu, C.H.; Kuo, T.W. An adaptive two-level management for the Flash translation layer in embedded systems. In Proceedings of the 2006 IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, USA, 5–9 November 2006; pp. 601–606.
11. Liu, Y.; Zhang, Z.; Xu, J.; Xie, G.; Li, R. Coded worn block mechanism to reduce garbage collection in SSD. *J. Syst. Archit.* **2022**, *126*, 102487. [CrossRef]
12. *SD Specifications Part 1 Physical Layer Simplified Specification Version 5.0*; Technical Report; Technical Committee SD Card Association: San Ramon, CA, USA, 2016.
13. RISC-V International. Available online: <https://riscv.org/> (accessed on 20 October 2023).
14. Intel Corp. *Nios II Processor Reference Guide, NII-PRG—2020.04.24*; Intel Corp.: Santa Clara, CA, USA, 2020.
15. Xilinx; IARR. *Microblaze Processor Reference Guide*; Xilinx: San Jose, CA, USA, 2006.

16. OpenRISC Community. Available online: <https://openrisc.io> (accessed on 8 August 2023).
17. OpenCores. Available online: <http://opencores.org> (accessed on 8 August 2023).
18. Silicore, T.M. *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*; Revision B3; Silicore TM: Shaoxing, China, 2002.
19. Arm Limited. *AMBA AXI and ACE Protocol Specification*; Arm Limited: Cambridge, UK, 2011.
20. Yaghmour, K.; Masters, J.; Ben, G. *Building Embedded Linux Systems*, 2nd ed.; O'Reilly & Associates, Inc.: Sebastopol, CA, USA, 2008.
21. PramodKumar Singh, P.S.K. Startup Time Optimization Techniques for Embedded Linux. *Int. J. Sci. Eng. Res.* **2016**, *7*, 742–745.
22. Gu, C. Power On and Bootloader. In *Building Embedded Systems: Programmable Hardware*; Apress: Berkeley CA, USA, 2016; pp. 5–25.
23. Lee, D.; Won, Y. Booting Linux faster. In Proceedings of the 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content, Beijing, China, 21–23 September 2012; pp. 665–668. [CrossRef]
24. Ho Chung, K.; Sil Choi, M.; Seon Ahn, K. A Study on the Packaging for Fast Boot-up Time in the Embedded Linux. In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, Republic of Korea, 21–24 August 2007; pp. 89–94. [CrossRef]
25. Park, C.; Kim, K.; Jang, Y.; Hyun, K. Linux bootup time reduction for digital still camera. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 19–22 July 2006; p. 231.
26. Denk, W. Das U-Boot—The Universal Boot Loader, rev r1.30—30 Mar 2020. 2020. Available online: <https://github.com/u-boot/u-boot> (accessed on 20 July 2023).
27. Rath, N. FUSE—Filesystem in Userspace. Available online: <https://github.com/libfuse/libfuse> (accessed on 6 July 2023).
28. Sources for NanoBoot. Available online: <https://github.com/germancq/NanoBoot-systemverilog> (accessed on 20 July 2024).
29. de Clavijo, P.R.; Ostua, E.; Bellido, M.J.; Juan, J.; Viejo, J.; Guerrero, D. Minimalistic SDHC-SPI hardware reader module for boot loader applications. *Microelectron. J.* **2017**, *67*, 32–37. [CrossRef]
30. Digilent, Inc. *Nexys4 DDR FPGA Board Reference Manual*; Digilent, Inc.: Pullman, WA, USA, 2016.
31. Digilent, Inc. *Arty TM FPGA Board Reference Manual, Rev. C*; Digilent, Inc.: Pullman, WA, USA, 2017.
32. Xilinx Inc. *7 Series FPGAs Data Sheet: Overview. DS180 (v2.6) Product Specification*; Xilinx Inc.: San Jose, CA, USA, 2018.
33. Digilent, Inc. *Genesys 2 FPGA Board Reference Manual, Rev. F*; Digilent, Inc.: Pullman, WA, USA, 2017.
34. Xilinx Inc. *PetaLinux Tools Documentation Reference Guide, UG1144 v2018.2*; Xilinx Inc.: San Jose, CA, USA, 2018.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# An Area-Efficient and Configurable Number Theoretic Transform Accelerator for Homomorphic Encryption

Jingwen Huang, Chiayi Kuo, Sihuang Liu and Tao Su \*

School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510006, China; huangjw86@mail2.sysu.edu.cn (J.H.); kuocy@mail2.sysu.edu.cn (C.K.); liush233@mail2.sysu.edu.cn (S.L.)

\* Correspondence: sutao@mail.sysu.edu.cn

**Abstract:** Homomorphic Encryption (HE) allows for arbitrary computation of encrypted data, offering a method for preserving privacy in cloud computations. However, efficiency remains a significant obstacle, particularly with the polynomial multiplication of large parameter sets, which occupies substantial computing and memory overhead. Prior studies proposed the use of Number Theoretic Transform (NTT) to accelerate polynomial multiplication, which proved efficient, owing to its low computational complexity. However, these efforts primarily focused on NTT designs for small parameter sets, and configurability and memory efficiency were not considered carefully. This paper focuses on designing a unified NTT/Inverse NTT (INTT) architecture with high area efficiency and configurability, which is more suitable for HE schemes. We adopt the Constant-Geometry (CG) NTT algorithm and propose a conflict-free access pattern, demonstrating a 16.7% reduction in coefficients of storage capacity compared to the state-of-the-art CG NTT design. Additionally, we propose a twiddle factor generation strategy to minimize storage for Twiddle Factors (TFs). The proposed architecture offers configurability of both compile time and runtime, allowing for the deployment of varying parallelism and parameter sets during compilation while accommodating a wide range of polynomial degrees and moduli after compilation. Experimental results on the Xilinx FPGA show that our design can achieve higher area efficiency and configurability compared with previous works. Furthermore, we explore the performance difference between precomputed TFs and online-generated TFs for the NTT architecture, aiming to show the importance of online generation-based NTT architecture in HE applications.

**Keywords:** homomorphic encryption; number theoretic transform; constant-geometry; memory access pattern; twiddle factor generation strategy; configurability

## 1. Introduction

Homomorphic Encryption (HE), enabling the computation of encrypted data, is an optimal privacy-preserving solution for various scenarios, including cloud computing [1] and machine learning [2]. Unlike cryptographic methods [3,4] based on chaos theory, HE relies primarily on algebraic structures such as latticed-based mathematical theories. Over the past few decades, multiple HE schemes based on the Ring Learning With Error (RLWE) problem [5] have been proposed, among which the Brakerski–Gentry–Vaikuntanathan (BGV) scheme [6], Brakerski–Fan–Vercauteren (BFV) scheme [7], and Cheon–Kim–Kim–Song (CKKS) scheme [8] are popular. Despite the improvements in theoretical cryptography, intensive computing and high storage requirements remain serious, i.e., thousands of times slower on unencrypted data [9] and occupying up to 512 MB of on-chip memory [10], limiting its widespread adoption.

Polynomial multiplication, as a fundamental operation in HE schemes, has become a performance bottleneck for HE applications. Using NTT can reduce the complexity of polynomial multiplication from traditional  $\mathcal{O}(N^2)$  to  $\mathcal{O}(N \log_2 N)$ , where  $N$  is the degree of the polynomial. Consequently, the effective implementation of NTT significantly impacts

the performance of HE applications. Unlike low-degree polynomials (ranging from  $2^8$  to  $2^{12}$ ) with small moduli (i.e., 24 bit) needed in some traditional post-quantum cryptography schemes [11,12], HE schemes [13–16] require a higher polynomial degree (ranging from  $2^{13}$  to  $2^{17}$ ) and larger moduli (i.e., 1240-bit) to support multiplicative depth. Introducing a Residue Number System (RNS) to HE enables the decomposition of a large modulus into several smaller moduli with varying bit widths (e.g., 54 bit). Therefore, an NTT architecture suitable for RNS-based HE schemes is required to support a set of high-degree polynomials with diverse moduli. This implies that when accelerated in hardware, such an architecture consumes a significant amount of on-chip resources. Consequently, the building of a hardware-efficient NTT architecture for practical applications of HE is in great demand.

Area efficiency and configurability are two key metrics used to evaluate the performance of NTT designs. Area efficiency shows a negative correlation with resource and time costs. Configurability emphasizes both Compile-Time Configurability (CTC) [17] and Run-Time Configurability (RTC) [18]. CTC enables the adjustment of parameter sets at the compilation time to satisfy different schemes with a wide range of parameters. It also means that the computational parallelism of the NTT architecture can be compiled to realize the trade-off between area and speed, RTC corresponds to the ability to support various parameter sets at runtime.

Existing NTT designs [19,20] for HE parameters tend to employ a large number of Processing Elements (PEs) to achieve high throughput, albeit at the cost of notable resource overhead and low area efficiency. Öztürk et al. [19] introduced an NTT architecture with up to 256 PEs to accelerate the LopezAlt–Tromer–Vaikuntanathan (LTV) scheme, ensuring low latency. However, the proposed architecture stores all TFs for multiple moduli in on-chip memory, leading to high memory overhead, which is challenging for polynomial degrees greater than  $2^{15}$  on the target FPGA device. Similarly, Su et al. [20] deployed 168 PEs across 41 RNS channels and stored all TFs for 41 32-bit moduli in internal memory, occupying a significant amount of hardware resources. Consequently, supporting moduli with a larger bit width on the target FPGA device became impossible. Kurniawan et al. [21] reported a memory-based NTT design, presenting remarkable advantages of its low-complexity modular multiplier and optimized memory access pattern. However, it is still unsuitable for RNS-based HE schemes with multiple moduli due to the considerable storage overhead for TFs. To solve this issue, Kim et al. [22] and Duong-Ngoc et al. [23] used the twiddle factor generation strategy to avoid the storage of TFs. Furthermore, they reduced the memory bandwidth requirement during high-computational parallelism by employing the mixed-radix NTT algorithm, enhancing area efficiency. However, flexibility is limited because both designs only aim at the specific RNS-based HE parameter sets. Moreover, the number of PEs was fixed at design time, while the lack of unified data flows between NTT and INTT constrained the architectures' adaptability.

On the other hand, some recent NTT designs [17,24–27] with configurability have restrictions on performance and area efficiency. Mert et al. [17] proposed the first NTT hardware generator that takes the desired parameters and the number of PEs as inputs. A multiplier using the word-level Montgomery algorithm could adapt to moduli of diverse bit widths without recompilation. Hu et al. [24] developed an NTT-based high-efficiency polynomial multiplier that permits the reuse of twiddle factors across different polynomial degrees. However, the two designs based on the in-place algorithm suffer from increased access complexity and lower hardware efficiency as PEs increase. To solve this problem, Banerjee et al. [28] proposed the Constant-Geometry (CG) algorithm, which exhibits superior scalability for multiple PEs, benefiting from its consistent memory access structure in each stage. However, the CG algorithm operates out of place, resulting in twice the memory overhead for coefficients compared to the in-place algorithm. Su et al. [25] and Liu et al. [26] optimized the coefficient access pattern, reducing the coefficient storage requirement from  $2N$  to  $1.5N$ , and exploited algorithm-level optimization techniques. However, the memory bandwidth requirement remains high, which hinders its implementation on FPGA. Geng et al. [27] introduced a novel high–low iterative memory access approach to reduce



the storage bandwidth required per PE, mitigating storage restrictions. However, it is worth noting that in the mentioned works with configurability, all precomputed constants are stored in the internal memory. Therefore, the memory footprint remains a major obstacle when these NTT designs are embedded into HE schemes.

Consequently, developing a unified NTT/INTT architecture with flexibility while ensuring high area efficiency for HE-based parameters remains challenging. This paper proposes an area-efficient and highly configurable architecture to accelerate CG-based NTT/INTT operations on FPGA. Reducing the storage of coefficients and twiddle factors is the core focus of this study. More specifically, our contributions are summarized as follows:

- (1) We propose a novel memory access pattern based on the CG algorithm, reducing the total coefficient capacity requirement from  $1.5N$  to  $1.25N$ . Moreover, we minimize the required memory bandwidth per PE as much as that of the in-place-based radix-2 algorithm. The reduction in overall capacity and bandwidth significantly mitigates memory demands when compared to prior works.
- (2) We develop a flexible Twiddle Factor Generator (TFG), with a Step Generator (SG) supplying multipliers for the TFG. With this approach, only a few TF bases need to be stored, while the remaining TFs are generated on the fly. Implementation results show a 99.17% reduction of TF storage in our proposed NTT architecture with 32 PEs when the maximum polynomial degree is set to  $2^{16}$ .
- (3) We present a hardware-efficient and configurable unified architecture to accelerate NTT/INTT without additional modifications. The proposed design not only allows for a balanced design in terms of area and throughput based on the compiled parameters, including the number of PEs, the supported maximum number of polynomial degrees, and the maximum size of polynomial coefficients, but it also supports polynomials with varying degrees and coefficients of different sizes after compilation. In addition, although our architecture is designed for HE parameter sets, it also accommodates small parameter sets without compromising computational complexity.
- (4) We also implement a memory-based NTT/INTT architecture adopting our proposed coefficient access pattern, aiming to fairly compare it with previous NTT designs with all TFs stored in memory. The experimental results demonstrate the improved performance of the proposed memory access pattern. In the end, we conduct another comparison to investigate the performance difference between precomputed TFs and online-generated TFs for the NTT architecture. The results show the importance of the online generation-based NTT architecture in RNS-based HE applications.

The organization of this paper is outlined as follows. Section 2 provides basic knowledge about polynomial multiplication and the CG NTT algorithm. The hardware architecture is detailed in Section 3. In Section 4, the implementation results are analyzed and compared with prior works. Finally, Section 5 concludes this article.

## 2. Background

### 2.1. Polynomial Multiplication

Polynomial multiplication defined in a finite domain is considered a fundamental operation in cryptographic algorithms. It is a computational bottleneck when the polynomial degree is high, especially in HE applications. Typically, mathematical operations of polynomials are performed over a quotient ring ( $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ , where  $N$  is a power of 2 and  $q$  is an NTT-friendly prime satisfying the condition of  $q \equiv 1 \pmod{2N}$ ). The polynomial over  $R_q$  has a degree of  $N$ , with coefficients constrained to the interval of  $[0, q)$ . Consider two polynomials ( $a(x) = \sum_i a_i x^i$  and  $b(x) = \sum_i b_i x^i$ ), with  $c(x) = a(x) \cdot b(x)$  denoting the multiplication result over  $R_q$ . The traditional computational method requires expensive matrix-vector multiplication, resulting in computational complexity of  $\mathcal{O}(N^2)$ . The NTT algorithm is utilized to accelerate the process, with a complexity of  $\mathcal{O}(N \log_2 N)$ , while still requiring a modulo operation of  $(x^N + 1)$  in the end. Employing the Negative Wrapped Convolution (NWC) [29] method can circumvent the need for zero-padding and dimension extension by introducing additional preprocessing and post-processing. Ac-



cordingly, for determined  $q$  and  $N$ , we define  $w$  and  $\psi$  as the  $N$ -th and  $2N$ -th unit primitive roots on  $\mathbb{Z}_q$ , respectively satisfying the relation of  $\psi^2 \equiv w \pmod{q}$ . The symbol  $\odot$  signifies element-wise multiplication. Preprocessing operations for  $a(x)$  and  $b(x)$  are outlined in Equation (1) [26] and Equation (2) [26], respectively.

$$\tilde{a}(x) = a(x) \odot \{\psi^0, \psi^1, \dots, \psi^{N-1}\} \quad (1)$$

$$\tilde{b}(x) = b(x) \odot \{\psi^0, \psi^1, \dots, \psi^{N-1}\} \quad (2)$$

Then, polynomial multiplication is performed as follows [26]:

$$\tilde{c}(x) = INTT(NTT(\tilde{a}(x)) \odot NTT(\tilde{b}(x))) \quad (3)$$

The final multiplication results over  $R_q$  are obtained by executing the post-processing operation for  $\tilde{c}(x)$  as Equation (4) [26].

$$c(x) = \tilde{c}(x) \odot \{\psi^{-0}, \psi^{-1}, \dots, \psi^{-(N-1)}\} \quad (4)$$

## 2.2. Constant-Geometry NTT/INTT Algorithm

The CG algorithm is an out-of-place NTT algorithm, where access patterns at every stage remain consistent, rendering it suitable for the extension of multi-core array architectures. The CG NTT module utilizes the Cooley–Tukey (CT) algorithm [30], whereas the CG INTT module relies on the Gentleman–Sande (GS) algorithm [31]. The CT algorithm and GS algorithm are two divide-and-conquer strategies for NTT/INTT computation. Additional optimization efforts involve merging preprocessing and post-processing steps [32], along with the incorporation of the  $N^{-1}$  operation into each stage of the INTT [25]. Based on the simplified CG algorithm proposed by Su et al. [25], we make some adjustments to make it applicable to scenarios with  $L$  PEs, as shown in Algorithms 1 and 2. The BitReverse function reorders elements by reversing the binary representation of their original indices, effectively rearranging the input vector.

---

### Algorithm 1 Constant-Geometry NTT [25]

---

**Require:**  $a(x)$ ,  $q$ ,  $\psi^i$ , where  $i = 0, 1, \dots, N - 1$ .

**Ensure:**  $A(x) = NTT(a(x))$

```

1:  $a \leftarrow \text{BitReverse}(a)$ 
2: for  $s = 1$  to  $\log_2 N$  do
3:   for  $j = 0$  to  $\frac{N}{2^s} - 1$  do
4:     for  $l = 0$  to  $L - 1$  do
5:        $e = jL + l$ 
6:        $k = \frac{N}{2^s}$ 
7:        $A[e] = a[2e] + a[2e + 1] \cdot \psi^{2\lfloor \frac{e}{k} \rfloor \cdot k + k} \pmod{q}$ 
8:        $A[e + \frac{N}{2}] = a[2e] - a[2e + 1] \cdot \psi^{2\lfloor \frac{e}{k} \rfloor \cdot k + k} \pmod{q}$ 
9:     end for
10:   end for
11:   if  $s \neq \log_2 N$  then
12:      $a = A$ 
13:   end if
14: end for
15: return  $A(x)$ 
```

---

**Algorithm 2** Constant-Geometry INTT [25]

---

**Require:**  $A(x), q, \psi^{-i}$ , where  $i = 0, 1, \dots, N - 1$ .  
**Ensure:**  $a(x) = INTT(A(x))$

```

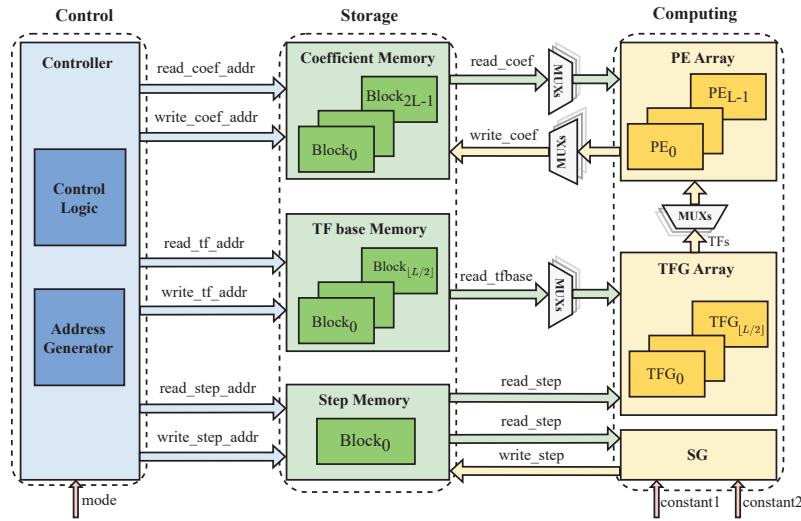
1: for  $s = 1$  to  $\log_2 N$  do
2:   for  $j = 0$  to  $\frac{N}{2^s} - 1$  do
3:     for  $l = 0$  to  $L - 1$  do
4:        $e = jL + l$ 
5:        $k = 2^{s-1}$ 
6:        $a[2e] = (A[e] + A[e + \frac{N}{2}]) \cdot 2^{-1} \bmod q$ 
7:        $a[2e + 1] = (A[e] - A[e + \frac{N}{2}]) \cdot \psi^{-(2\lfloor \frac{L}{2} \rfloor \cdot k + k)} \cdot 2^{-1} \bmod q$ 
8:     end for
9:   end for
10:  if  $s \neq \log_2 N$  then
11:     $A = a$ 
12:  end if
13: end for
14:  $a \leftarrow \text{BitReverse}(a)$ 
15: return  $a(x)$ 

```

---

**3. The Proposed Accelerator****3.1. Overall Architecture**

The overall architecture of our design is illustrated in Figure 1, consisting of the following three types of components: computing, storage, and control components. The computing components comprise the PE array, TFG array, and SG. Meanwhile, the storage components include the coefficient memory, TF base memory, and step memory. The control unit receives a two-bit working mode signal as input to transition the circuit among the following three states: IDLE, NTT, and INTT. Further elaboration on the specific functionalities of each submodule follows.



**Figure 1.** Overall architecture with  $L$  PEs and  $\lceil L/2 \rceil$  TFGs.

- (1) **PE array:** The PE array is the computational component in this architecture, primarily handling butterfly operations. It consists of  $L$  PEs, where the parameter ( $L$ ) is typically set to a power of two. Each PE executes either NTT or INTT operations on input polynomials, depending on the working state of the architecture.
- (2) **Coefficient memory:** The coefficient memory stores all polynomial coefficients during calculation. In this design, a total storage capacity of  $1.25N$  for coefficients is necessary, which is 16.7% less than the prior CG storage requirement of  $1.5N$ . The total storage space is divided into  $2L$  memory blocks to accommodate the computational bandwidth of the PE array.

- (3) TF base memory: The TF base memory holds TF bases, supplying inputs to the TFG array for the dynamic generation of remaining TFs. The overall storage capacity is contingent upon computational parallelism and the polynomial degree. When  $L \geq 2$ ,  $((\log_2 N + 1) \times L/2) \times 2$  storage space is required; otherwise,  $\log_2 N \times 2$  space suffices. The total storage space is divided into  $\lceil L/2 \rceil$  blocks to meet the computational bandwidth of the TFG array, where the symbol  $\lceil \cdot \rceil$  represents rounding up.
- (4) Step memory: The step memory stores the inputs and outputs of the SG. The storage capacity is adjusted to match the pipeline of the SG, preventing pipeline stalls caused by mismatched data processing speeds.
- (5) SG: The SG is responsible for cyclically generating steps to supply the TFG array, thereby avoiding the need for expensive step storage.
- (6) TFG array: The TFG array consists of  $\lceil L/2 \rceil$  twiddle factor generators (TFGs), which take steps and TF bases as inputs and output TFs to the PE array for computations.
- (7) Control unit: The controller provides the correct read address and write address for storage components and coordinates the work of all components.

In this architecture, all memory modules are configured in simple dual-port mode, allowing for external initialization after compilation. Modular Multipliers (MMs) utilize the word-level Montgomery algorithm. With specified iterations and parameterized word size, they can accommodate moduli within a range of bit widths after compilation. The compile parameters of  $N$  and  $\lceil \log_2 q_{max} \rceil$  are the maximum supported polynomial degree and the maximum supported modulus bit width, respectively. The compiled circuit can support different polynomial degrees ( $n$ ;  $2L \leq n \leq N$ ). Moreover, the architecture realizes the unification of NTT and INTT in the design of all components.

### 3.2. Unified PE

The proposed PE architecture is shown in Figure 2, consisting of a Modular Adder (MA), Modular Subtractor (MS), Modular Multiplier (MM), Modular Divider (MD), Multiplexers (Muxs), and registers. The inputs are  $a$ ,  $b$ , and  $c$ , and the outputs are  $Res0$  and  $Res1$ . The signal *mode* determines whether the PE performs NTT or INTT. The PE executes the CG NTT operations when *mode* = 0. The results are shown as follows [26]:

$$Res0 = a + b \times c \mod q \quad (5)$$

$$Res1 = a - b \times c \mod q \quad (6)$$

When *mode* is set to 1, the PE executes the CG INTT operations. The results are shown as follows [26]:

$$Res0 = (a + b) \times \frac{1}{2} \mod q \quad (7)$$

$$Res1 = (a - b) \times c \times \frac{1}{2} \mod q \quad (8)$$

In addition, a total of  $(2m + 4)$  registers are inserted to balance the pipeline latency, where  $m$  is the pipeline level of the MM.

The MM applies the word-level Montgomery algorithm proposed in [33] as shown in Algorithm 3. It utilizes the equivalent expression of an NTT-friendly prime ( $q = q_H \times 2^W + 1$ ) to decompose the modular reduction operation into smaller steps. The compiled parameter ( $W$ ) is usually set to  $\log_2 2N$ , denoting the word size. The iteration number of the “for” loop is represented by the parameter ( $T$ ), where  $T = \lceil \log_2 q_{max} / W \rceil$ . This means splitting large-bit-width modulo reduction steps into  $T$  iterations of smaller-bit-width modular operations; therefore, fewer Digital Signal Processing (DSP) resources are needed. After compilation, the supported modulus bit width ( $K$ ) is configurable at runtime within the range of  $(W, \lceil \log_2 q_{max} \rceil]$ , while the traditional Barrett modular reduction algorithm [34] only supports a fixed-modulus bit width of  $K = \lceil \log_2 q_{max} \rceil$  after compilation and lacks runtime configurability. For  $n < N$ , setting  $W$  to  $\log_2 2N$  restricts the range of  $q$  for

dimension  $n$ . Hence,  $W$  should be small enough to accommodate a wide range of  $n$ , yet this entails more iterations for specific  $\lceil \log_2 q_{max} \rceil$ . Therefore,  $W$  and  $T$  should be carefully selected to obtain the optimal trade-off between generality and complexity. In our work, we set  $T$  to 4, and  $W$  can be compiled as  $\{15, 16, 17\}$  to match the common HE-based parameter set at  $(N = 2^{16}, K = 60)$  [23]. It is important to note that the constraint on  $q$  varies with  $n < N$ . Specifically, as  $n$  decreases, the range of values for  $q$  shrinks compared to the initially supported range, while it remains unchanged as  $n$  increases. For example, when  $N = 2^{16}$  and  $W = 17$ , for  $n = 2^{15}$  or  $n = 2^{14}$ ,  $q$  theoretically only needs to satisfy the expression of NTT-friendly primes ( $q = q_H \times 2^{16} + 1$  or  $q = q_H \times 2^{15} + 1$ ). However, the compiled parameter ( $W$ ) determines that  $q$  must adhere to the condition of  $q = q_H \times 2^{17} + 1$ , thereby imposing a constraint on  $q$ . Even so, compared to previous work [22,23], which utilized specialized forms of modulus to reduce complexity, the proposed MM is more general for HE applications.

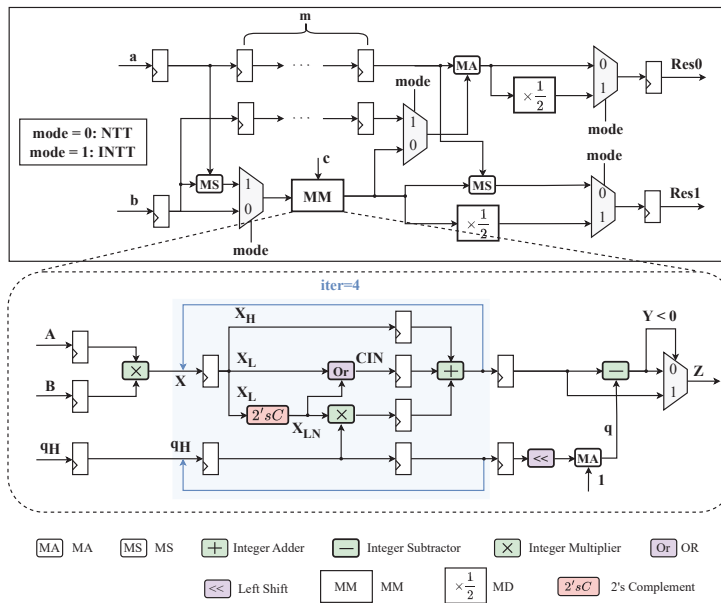


Figure 2. Architectures of PE and MM.

### Algorithm 3 Word-level Montgomery Modular Multiplication [33]

**Require:**  $A, B, q = q_H \cdot 2^W + 1$  (three  $K$ -bit positive integers,  $W < K \leq \lceil \log_2 q_{max} \rceil$ )

**Ensure:**  $A \cdot B / R \bmod q$ , where  $R = 2^{4 \times W} \bmod q$

```

1:  $X = A \cdot B$ 
2: for  $i = 0$  to 3 do
3:    $X_H = X \gg W$ 
4:    $X_L = X \bmod 2^W$ 
5:    $X_{LN} = -X_L \bmod 2^W$  // 2's Complement of  $X_L$ 
6:    $CIN = X_{LN}[W-1] \mid X_L[W-1]$ 
7:    $X = X_H + CIN + q_H \cdot X_{LN}$ 
8: end for
9:  $Y = X - q$ 
10: if  $(Y < 0)$  then
11:    $Z = X$ 
12: else
13:    $Z = Y$ 
14: end if
15: return  $Z$ 

```

Modular division can be viewed as an inverse element multiplier. For  $x/2 \bmod q$ , it can be implemented using a shifter, adder, and multiplexers without multipliers. Similar to prior works [25,26], MD is designed based on Equation (9).

$$\frac{x}{2} \bmod q = \begin{cases} (x \gg 1), & \text{if } x \text{ is even;} \\ (x \gg 1) + \frac{q+1}{2}, & \text{if } x \text{ is odd.} \end{cases} \quad (9)$$

### 3.3. Coefficient Memory Access Pattern

The storage overhead for coefficients on FPGA mainly depends on the following two factors: the total storage capacity and the number of required memory blocks per PE. This paper proposes a novel coefficient memory access pattern based on the CG algorithm, effectively reducing the storage demand from  $1.5N$  to  $1.25N$  (where  $N$  is the polynomial degree). At the same time, each PE necessitates just two memory blocks. In prior CG-based NTT studies [25–27], a storage capacity of  $2N$  or  $1.5N$  was typically allocated for coefficient storage. Su et al.'s work [25] and Liu et al.'s work [26] required 12 and 4 memory blocks per PE, respectively. Although Geng et al.'s work [27] required just two memory blocks per PE, their total storage requirement remained at  $2N$ . In contrast, our design minimizes the total storage capacity and bandwidth, leading to a notable decrease in Block Random Access Memory (BRAM) resource usage.

The proposed coefficient storage arrangement comprises  $2L$  memory blocks to match the computational needs of  $L$  PEs. Assuming that  $N$  original coefficients are cyclically distributed across  $2L$  memory blocks, the coefficient pairs  $(a[e], a[e + N/2])$  occupy the same block, hindering the simultaneous retrieval of both source data during INTT. Consequently, we reorganize coefficients  $a[N/2] \sim a[N - 1]$  in reverse order within  $2L$  memory blocks, while coefficients  $a[0] \sim a[N/2 - 1]$  are still stored sequentially. This strategy effectively segregates  $a[e]$  and  $a[e + N/2]$  into distinct blocks. Moreover, it is important to acknowledge that this arrangement necessitates at least  $0.5N$  of additional space to ensure sufficient time for data retrieval before updates. To minimize the extra storage requirement, we further refine the layout, grouping coefficients in sets of  $N/4$ .

The improved coefficient arrangement is depicted in Figure 3, in which each block has a depth of  $1.25N/2L$ . Within each block,  $N/2L$  of space is designated for storage of the original coefficients, while the remaining  $N/8L$  of space is allocated for newly generated data. To facilitate explanation, we divide  $2L$  blocks into five regions from top to bottom, labeled as Regions 1~5. The arrangement of the original coefficients proceeds as follows:

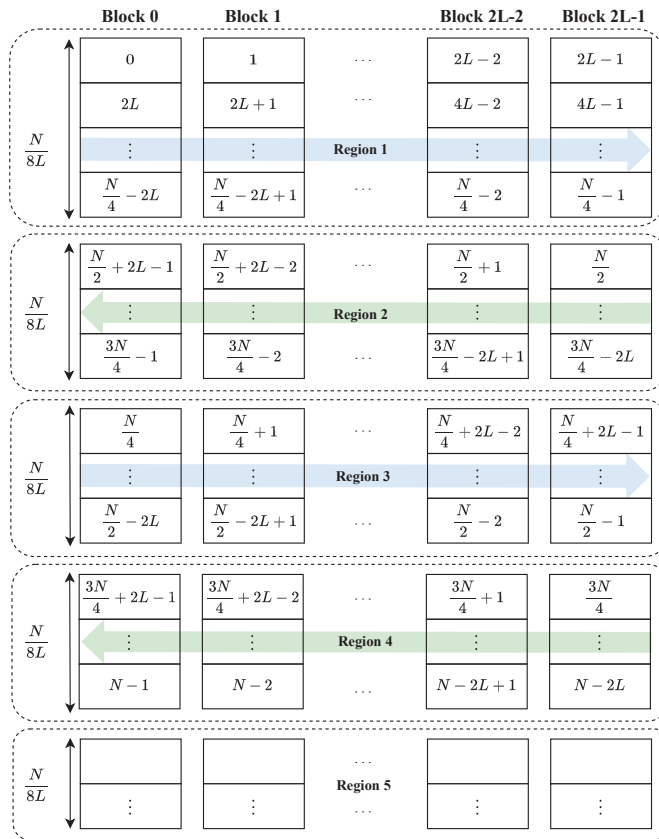
- $a[0] \sim a[N/4 - 1]$  are placed in Region 1 sequentially;
- $a[N/4] \sim a[N/2 - 1]$  are placed in Region 3 sequentially;
- $a[N/2] \sim a[3N/4 - 1]$  are stored in reverse order at the block level, filling up Region 2;
- $a[3N/4] \sim a[N - 1]$  are stored in reverse order at the block level, filling up Region 4.

More details are provided below for the loading and storing patterns of data during NTT. As shown in Algorithm 1, during the  $j$ -th iteration, coefficient pairs  $(a[2(jL + l)], a[2(jL + l) + 1])$  are loaded from blocks and transmitted to  $PE_l$  for computation. Then, the updated results  $(A[jL + l], A[jL + l + N/2])$  are stored in blocks. The  $l$  parameter ranges from 0 to  $L - 1$ , indicating a total of  $2L$  data being simultaneously read and stored in each computation cycle of the NTT. In the following, we describe the two initial executions of PEs in stage 1 in detail.

- At the first step in stage 1, i.e.,  $j = 0$ ,  $PE_l$  retrieves  $a[2l]$  from Block  $2l$  and  $a[2l + 1]$  from Block  $2l + 1$  to execute butterfly computation. The result  $(A[l])$  is stored in the first available slot in Block  $l$ , and another result  $(A[N/2 + l])$  is stored in the first row of Block  $2L - 1 - l$ . When  $L$  PEs operate simultaneously, coefficients in the first row of Region 1 are loaded simultaneously.  $2L$  results are stored in the first vacant position of Block  $0 \sim L - 1$  in Region 5 and the first row of Block  $L \sim 2L - 1$  in Region 1 (the original coefficients in the first row of Block  $L \sim 2L - 1$  are utilized).

- At the second step in stage 1, i.e.,  $j = 1$ ,  $PE_l$  fetches  $a[2(L + l)]$  from Block  $2l$  and  $a[2(L + l) + 1]$  from Block  $2l + 1$ . The output ( $A[L + l]$ ) is stored in the first available position in Block  $L + l$ , and  $A[L + l + N/2]$  is stored in the first row of Block  $L - 1 - l$ . When  $L$  PEs work simultaneously, coefficients in the second row of Region 1 are loaded, and the computed results are stored separately in the first vacant positions of Block  $L \sim 2L - 1$  in Region 5 and the first row of Block  $0 \sim L - 1$  in Region 1 (the original coefficients in the first row of Block  $0 \sim L - 1$  are utilized).

In the next steps of stage 1, the coefficients are loaded and stored according to the above steps. After  $N/4L$  clock cycles,  $a[0] \sim a[N/2 - 1]$  from Regions 1 and 3 are read. The newly generated  $N/2$  data fill Regions 5 and 1. PEs then handle the data from Regions 2 and 4, with the results filling in Regions 2 and 3. Upon computing stage 1, Region 4 becomes available for storage in stage 2. The described operations are replicated in successive stages. After each stage's execution, the remaining space is utilized for storage in the next stage. Subsequently, the coefficient storage arrangement reverts to its initial state at stage 6.

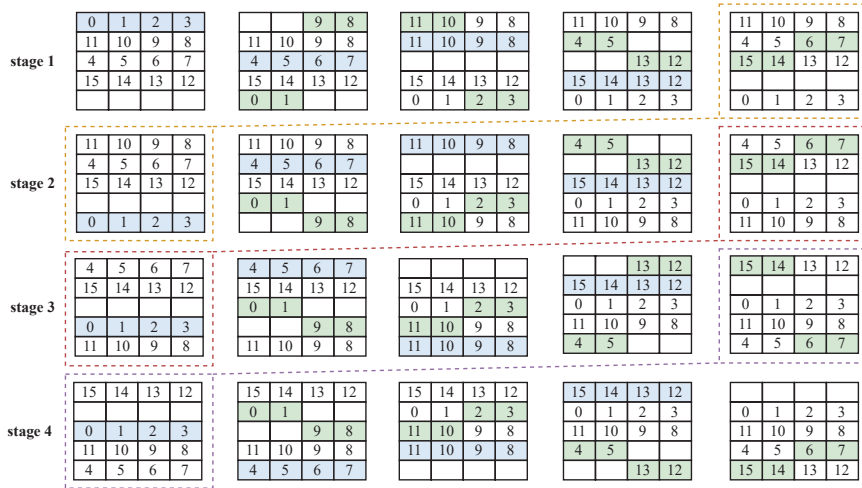


**Figure 3.** The arrangement of original coefficients across  $2L$  memory blocks.

For a more detailed depiction of the coefficient storage access pattern, let us derive the process for a 16-point NTT employing two PEs. The load and stored data flow are illustrated in Figure 4. It is assumed that one clock cycle is required from data retrieval to the completion of computation. The left–right direction represents the progression of clock cycles within a stage. Blue denotes the read data to be transmitted to PEs, and green denotes the newly generated data to be written into memory in that cycle.

The coefficient access logic for INTT is the reverse of that used in NTT. For example, in the first step of stage 1,  $L$  PEs read coefficients  $A[0] \sim A[L - 1]$  from Region 1 and  $A[N/2] \sim A[L - 1 + N/2]$  from Region 2. The newly generated data ( $a[0] \sim a[2L - 1]$ ) are written into the first row in Region 5. Subsequent steps follow a similar pattern, which is not repeated here.





**Figure 4.** The 16-point NTT employs two PEs. It is assumed that it takes one clock cycle from read to finish computation. Blue signifies the read data to be sent to PEs. Green denotes the newly generated data to be stored in memory.

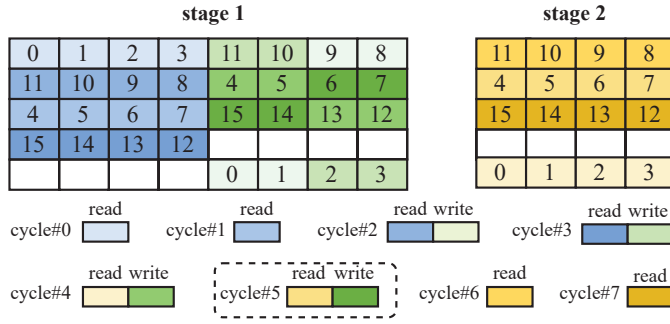
It is worth noting that the preceding description is rooted in the compilation parameter ( $N$ ), but the memory access structure specifically supports polynomials of  $n$  degree, which is much smaller than  $N$ . In this scenario, only the initial  $5n/8L$  of space of each block is accessed, leaving the remaining  $5N/8L - 5n/8L$  of space unused. Moreover, our design allows for external access to arbitrary addresses in all memory blocks, enabling users to dynamically update coefficients of  $n$ -dimensional polynomials in memory blocks post compilation. This feature ensures the proposed memory access pattern with runtime configurability.

### 3.4. Read-after-Write Conflict Analysis

Due to the delay in accessing RAM and PE computation, data from a particular address may be needed by the next stage before it is updated by the computation of the current stage, leading to access conflict. The critical point of conflict lies in the simultaneous reading and updating of data in two consecutive stages, known as a read-after-write conflict. The following conflict analysis process refers to Geng et al.'s work [27].

We define  $d$  as the total number of clock cycles, involving the latency of RAM access and the pipeline levels of PE. Figure 5 illustrates the occurrence of conflict when  $N = 16$ ,  $L = 2$ , and  $d = 2$ . The notation “cycle# $x$ ” indicates the  $x$ -th clock cycle starting from the NTT computation. Cycle#0 and cycle#2 are the first read and first write cycles of stage 1, respectively. If the coefficient access operations between stages are contiguous, then at cycle#5, coefficients indexed as 6 and 7 are to be read in stage 2, while both are also updated with results from stage 1, resulting in conflict. We can observe that conflicts always arise during the final write-back operation in every stage.

In general, each stage spans  $N/2L$  cycles for the retrieval of  $N$  coefficients. Coefficients with indices ranging from  $N/2 - L$  to  $N/2 - 1$  are written in the final cycle of the current stage's operation and read in the  $\#(N/4L - 1)$ -th cycle of the next stage. If  $N/2L - 1 + d = N/2L + N/4L - 1$ , then coefficients with indices of  $N/2 - L$  to  $N/2 - 1$  are written and read at the same time. Therefore, the condition of  $N/2L - 1 + d < N/4L + N/2L - 1$  is necessary to prevent such conflicts. The simplified condition of  $L < N/4d$  indicates that the maximum level of parallelism ( $L$ ) is decided by the polynomial degree ( $N$ ) and the delay ( $d$ ), thereby constraining the architectural flexibility. Furthermore, inserting idle periods between stages decelerates the read operation at the next stage; therefore, there are sufficient cycles to update coefficients within the current stage, effectively avoiding conflict.



**Figure 5.** Conflict in a 16-point NTT when  $L = 2$  and  $d = 2$ .

Assuming that the number of inserted idle cycles is  $g$ , the next stage waits for the additional  $g$  cycles to start reading after the current stage completes reading. Therefore, we have

$$\frac{N}{2L} - 1 + d < \frac{N}{2L} + \frac{N}{4L} - 1 + g \quad (10)$$

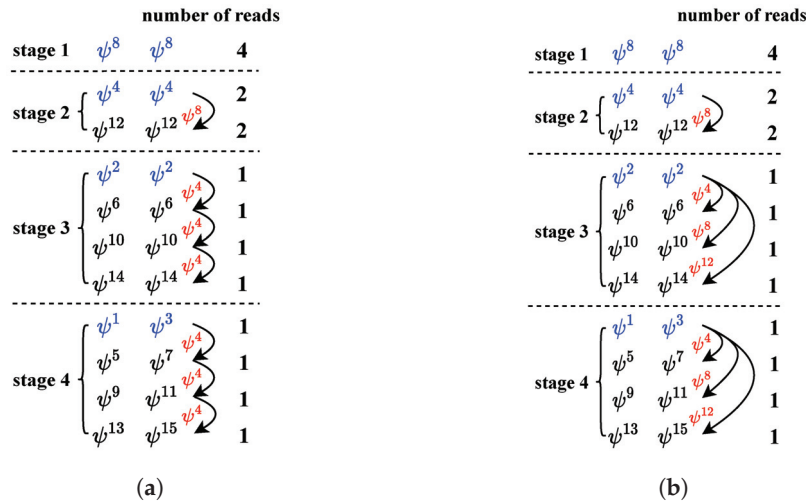
The minimum number of inserted idle cycles ( $g_{min}$ ) is expressed as follows:

$$g_{min} = \begin{cases} 0, & \text{if } L < \frac{N}{4d}, \\ d + 1 - \frac{N}{4L}, & \text{otherwise.} \end{cases} \quad (11)$$

The NTT operation takes an additional  $g_{min} \times (\log_2 N - 1)$  clock cycles, in total, to avoid conflict. In the end, through the strategic insertion of idle cycles between stages, we achieve a conflict-free memory access architecture.

### 3.5. TF Generation Strategy

The TF online generation strategy can be classified into two categories, i.e., data-dependent and data-independent, based on whether the generated TFs are utilized in the next generation. We take Figure 6 as an example to illustrate the difference between these generation methods. In Figure 6a,b, the required TFs for each stage are listed according to Algorithm 1. Columns 2 and 3 represent the factors allocated to two PEs, and column 4 indicates the frequency of repeated reading by PEs for each row's factors. For example, in stage 1,  $\psi^8$  is used by PEs and lasts for four clock cycles.



**Figure 6.** Twiddle factor generation methods in a 16-point NTT with 2 PEs. (a) Data-dependent strategy; (b) data-independent strategy.

For the case of data dependence, TFs labeled in black are obtained through the modular multiplication of TFs generated in the preceding operation and the corresponding step labeled in red. Therefore, the pre-stored constants include the TF bases marked in blue and the steps marked in red. Considering that MMs usually take several cycles to perform calculations, it is necessary to cache more TF bases for each stage to avoid pauses of the MM. The total number of stored TF bases is directly proportional to the pipeline levels of the MM. Kim et al. [22] and Duong-Ngoc et al. [23] employed this method to generate TFs on the fly. In the scenario of data independence, the update of TFs relies on TF bases (labeled in blue) and steps, as depicted in Figure 6b. The total number of pre-stored TF bases is determined by compilation parameters ( $N$  and  $L$ ), regardless of the pipeline levels in the MM. However, the varying steps within a stage pose a new challenge in terms of storage. In this paper, we focus on the data-independent TF generation strategy and propose a step generation method to avoid storing steps.

### 3.5.1. Step Generation

As shown in Algorithm 1,  $L$  PEs must execute  $N/2L$  iterations to complete coefficient transformation for one stage. For a new stage, assume that the  $j$ -th parallel execution of  $L$  PEs occurs in cycle# $j$ . Then, according to line 7 in Algorithm 1, the required TFs for PEs are  $\{\psi^{2\lfloor \frac{j}{k} \rfloor \cdot k + k} | l \in [0, L-1]\}$  at cycle#0 and  $\{\psi^{2\lfloor \frac{jL+l}{k} \rfloor \cdot k + k} | l \in [0, L-1]\}$  at cycle# $j$ , where  $l$  represents the index of PE. So, for PE $_l$ , the required step is computed as follows:

$$\frac{\psi^{2\lfloor \frac{jL+l}{k} \rfloor \cdot k + k}}{\psi^{2\lfloor \frac{j}{k} \rfloor \cdot k + k}} = \psi^{2k(\lfloor \frac{jL+l}{k} \rfloor - \lfloor \frac{j}{k} \rfloor)} \quad (12)$$

We can represent  $jL + l$  and  $l$  in binary form as Equation (13) and Equation (14), respectively.

$$(jL + l)_2 = \{j_{(\log_2 \frac{N}{2L}-1)}, \dots, j_0, l_{(\log_2 L-1)}, \dots, l_0\} \quad (13)$$

$$(l)_2 = \{l_{(\log_2 L-1)}, \dots, l_0\} \quad (14)$$

Since  $k$  is a power of 2, when  $\log_2 k < \log_2 L$ ,  $\lfloor \frac{l}{k} \rfloor$  is equivalent to the right shifting of  $l$  by  $\log_2 k$  bits. Similarly, the derivations for  $\lfloor \frac{jL}{k} \rfloor$  and  $\lfloor \frac{jL+l}{k} \rfloor$  follow a similar pattern as follows:

$$\lfloor \frac{l}{k} \rfloor_2 = \{l_{(\log_2 L-1)}, \dots, l_{\log_2 k}\} \quad (15)$$

$$\lfloor \frac{jL}{k} \rfloor_2 = \{j_{(\log_2 \frac{N}{2L}-1)}, \dots, j_0, \underbrace{0, \dots, 0}_{(\log_2 L - \log_2 k) \text{-bit}}\} \quad (16)$$

$$\lfloor \frac{jL+l}{k} \rfloor_2 = \{j_{(\log_2 \frac{N}{2L}-1)}, \dots, j_0, l_{(\log_2 L-1)}, \dots, l_{\log_2 k}\} \quad (17)$$

Hence, the following formula is valid.

$$\lfloor \frac{jL+l}{k} \rfloor = \lfloor \frac{jL}{k} \rfloor + \lfloor \frac{l}{k} \rfloor \quad (18)$$

When  $\log_2 k \geq \log_2 L$ , the proof of Equation (18) follows a similar trend. At cycle# $j$ , the required step sequence is

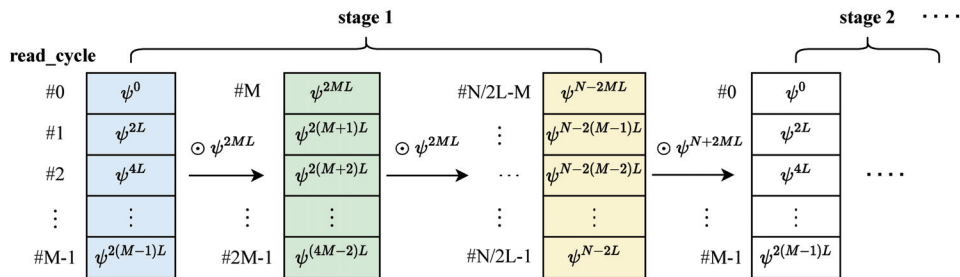
$$\psi^{2k(\lfloor \frac{jL+l}{k} \rfloor - \lfloor \frac{jL}{k} \rfloor)} = \psi^{2k\lfloor \frac{l}{k} \rfloor} \quad (19)$$

It can be observed that  $\psi^{2k\lfloor \frac{l}{k} \rfloor}$  is independent of  $l$ , indicating that the  $L$  TFGs use the same step to generate TFs for cycle# $j$ . When  $k = 1$ , i.e., in the final stage, the required step sequence follows the pattern of  $\{\psi^{2jL} | j \in [0, N/2L]\}$ . For other stages, the steps only

update to  $\psi^{2jL}$  at cycle#( $tk/L$ ) (where  $t$  is an integer) while maintaining consistency with the previous cycle. In general, based on the derivation presented above, we can draw the following three conclusions:

- The shared step across  $L$  TFGs for generating the TFs of the next cycle suggests that only one step generator needs to be designed to provide steps for  $L$  TFGs.
- The total number of unique steps is  $N/2L$  because the steps required for the final stage can be reused by the remaining stages.
- For the specified stage ( $s$ ), the required unique steps ( $\{\psi^{2jL} | j = tk/L, t \in Z\}$ ) are solely determined by  $j$ , implying that the step generation logic can be uniform across all stages.

The proposed step generator is an MM with two inputs, aiming to provide steps for  $L$  TFGs. To generate the unique step sequence  $\{\psi^{2jL} | j \in [0, N/2L)\}$  in a pipelined manner,  $M$  steps must be pre-stored in the step memory, where  $M$  is the total number of cycles involving the delay of the MM and an extra cycle. The steps in the step memory are continuously refreshed as the clock cycle increments, as shown in Figure 7. At cycle#0 of stage 1, the SG fetches the first step from address 0 and the constant  $\psi^{2ML}$  to generate the result ( $\psi^{2ML}$ ) at cycle#( $M - 1$ ). The result is written to the same address at the same time. Similarly, the second result of the SG is written to the second address of the step memory at cycle# $M$ . Once the SG handles  $M$  pre-stored steps sequentially, it circles back to fetch the step from address 0 as the input at cycle# $M$ . Continuing along the same lines, in the last cycle, we obtain the ultimate unique step ( $\psi^{N-2L}$ ), which also means that the complete set of steps ( $\{\psi^0, \psi^{2L}, \dots, \psi^{N-2L}\}$ ) is generated within the previous  $N/2L$  cycles. It is worth noting that the constant should be adjusted to  $\psi^{N+2ML}$  at cycle#( $N/2L - M$ ), aiming to replicate the generation of the same step sequence in the next stage. For  $L$  TFGs, at the  $(tk/L)$ -th cycle of each stage, the required step is retrieved from the step memory to contribute to the generation of new TFs.



**Figure 7.** The dynamic evolution of the step memory in an  $N$ -point NTT with  $L$  PEs. The steps labeled in blue represent  $M$  pre-stored steps. The  $\#j$  parameter denotes the  $j$ -th read cycle of each stage.

For better comprehension, we provide the timeline of the SG when  $N = 16$ ,  $L = 2$ , and  $M = 2$ , as shown in Figure 8. The data with blue background represent the steps pre-stored in the step memory. At cycle#0 of stage 1, the SG reads the step ( $\psi^0$ ) located at address 0 and the constant ( $\psi^8$ ) for modular multiplication. The result ( $\psi^8$ ) is written to the same address at cycle#1. In the same cycle, the SG reads the step ( $\psi^4$ ) located at address 1 for a new calculation. The result ( $\psi^{12}$ ) is stored at the same address at cycle#2. In the same cycle, the SG fetches the updated step ( $\psi^8$ ) from address 0 and the second constant ( $\psi^{24}$ ) to generate  $\psi^0$  for the next stage. Similarly, at cycle#3, the SG fetches the updated step ( $\psi^{12}$ ) from address 1 to produce the next step. The steps masked in red represent steps that need to be transmitted to TFGs, occurring at cycle#( $tk/L$ ), where  $t$  is an integer.

In conclusion, the proposed step generator reduces the total step storage from  $N/2L$  down to  $M$  and supports INTT mode through the dynamic modification of two constants and the step memory.

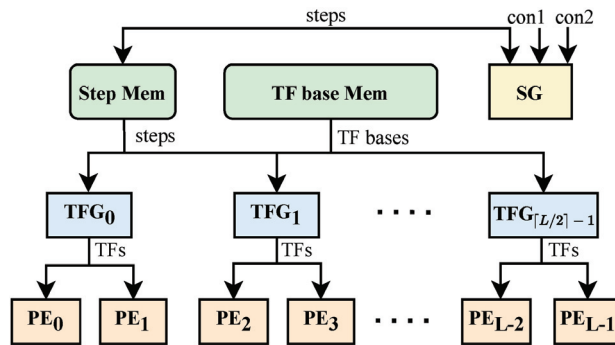
	stage 1				stage 2				stage 3				stage 4			
cycles	#0	#1	#2	#3	#0	#1	#2	#3	#0	#1	#2	#3	#0	#1	#2	#3
read_addr	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
in1	$\psi^0$	$\psi^4$	$\psi^8$	$\psi^{12}$	$\psi^0$	$\psi^4$	$\psi^8$	$\psi^{12}$	$\psi^0$	$\psi^4$	$\psi^8$	$\psi^{12}$	$\psi^0$	$\psi^4$	$\psi^8$	$\psi^{12}$
in2	$\psi^8$	$\psi^8$	$\psi^{24}$	$\psi^{24}$	$\psi^8$	$\psi^8$	$\psi^{24}$	$\psi^{24}$	$\psi^8$	$\psi^8$	$\psi^{24}$	$\psi^{24}$	$\psi^8$	$\psi^8$		
result	$\psi^8$	$\psi^{12}$	$\psi^0$		$\psi^4$	$\psi^8$	$\psi^{12}$	$\psi^0$	$\psi^4$	$\psi^8$	$\psi^{12}$	$\psi^0$	$\psi^4$	$\psi^8$		

**Figure 8.** The timeline of the SG when executing 16-point NTT with two PEs. The *in1* and *in2* signals refer to the inputs of the SG. The *result* signal denotes the output of the SG, and *read\_addr* represents the read address of the step memory.

### 3.5.2. TF Generation

The TFGs take the step from the step memory and the TF bases from the TF base memory as inputs to generate the remaining TFs for PEs. This means that the number of TFGs and PEs should be equal to maximize the computational capacity, leading to a doubling of computational resource overhead. Therefore, we conduct optimization to reduce the number of TFGs to  $\lceil L/2 \rceil$  at the cost of several additional clock cycles. The optimization strategy is not applied when one PE and one TFG exist. When  $L \geq 2$ , we can observe that, except for the final stage, at least two TFGs generate the same TF in the same cycle. Consequently, halving the total number of TFGs only slows down the operational efficiency of the last stage, requiring an extra  $N/2L$  cycles to generate TFs for the final stage. But it proves to be cost-effective, reducing the total computational resources by approximately 25%.

The proposed architecture for online TF generation, as illustrated in Figure 9, comprises one SG,  $\lceil L/2 \rceil$  TFGs, the TF base memory, and step memory. The total capacity of the TF base memory is, at most,  $(\log_2 N + 1) \lceil L/2 \rceil \times 2$  for one modulus, where the multiplication by 2 is due to the difference between TF bases in NTT and INTT. The total capacity of the step memory is  $M$ . Despite the expense of including additional  $N/2L$  clock cycles, it significantly diminishes the storage demands for TFs and steps, achieving enhanced area efficiency.



**Figure 9.** Architecture for online generation of TFs.

## 4. Evaluation

### 4.1. Experimental Environment and Methodology

We implement the proposed NTT architecture with the 2021.1 Xilinx Vivado tools. To ensure a fair comparison, the design is placed and routed on different FPGA platforms, in accordance with previous works. We conduct a thorough presentation and comparative analysis of the performance of the proposed accelerator in the following sections.

In Section 4.2, we present some details about our architecture, including resource, latency, and evaluation metrics. Subsequently, our comparison and analysis are divided into three subsections. In Section 4.3, we conduct a detailed comparison with existing works designed for HE parameters. In Section 4.4, utilizing the coefficient memory access

pattern proposed in this paper, we develop a memory-based NTT/INTT architecture that stores all TFs on chip, enabling a fair comparison with works optimized for small parameter sets. Finally, in Section 4.5, we explore the area efficiency of memory-based and online generation-based NTT architectures under RNS-based HE parameter sets.

#### 4.2. Experimental Results

First, Table 1 shows the hardware resource breakdown of our NTT architecture under the compilation parameter set ( $N = 2^{16}$ ,  $L = 32$ , and  $\lceil \log_2 q_{max} \rceil = 60$ ). We primarily focus on the BRAM and DSP overhead. In our design, each MM employs 18 DSP slices, and the PE array consumes 576 ( $= 18 \times 32$ ) DSP slices for 32 MMs. Moreover, the TFG array consumes 288 ( $= 18 \times 16$ ) DSP slices for 16 MMs. We including an additional MM to construct the SG, resulting in a total of 882 ( $= 576 + 288 + 18$ ) DSP slices. Coefficient storage consumes  $2L$  coefficient memory blocks with a depth of  $5N/8L$ , while TF base storage consumes  $\lceil L/2 \rceil$  memory blocks with a depth of, at most,  $(\log_2 N + 1) \times 2$  for one modulus. The step memory is implemented using Look-Up Table (LUT) resources, thereby avoiding the consumption of any BRAM. All BRAMs are configured in simple dual-port mode, with a total of 208 BRAMs used (i.e., 192 BRAMs for coefficients and 16 BRAMs for TF bases). Furthermore, to illustrate the reduction in storage capacity, Table 2 compares the coefficient storage overhead with the state-of-the-art CG NTT design [26] for  $N = 2^{16}$ . It also compares the total number of pre-stored TF bases for 32 RNS moduli with the design proposed in [23], which also optimizes the TF generator. The results indicate that our architecture reduces memory overhead for both coefficients and TFs in CG-based NTT designs.

**Table 1.** Resource breakdown of architecture with the compilation parameter set ( $N = 2^{16}$ ,  $L = 32$ , and  $\lceil \log_2 q_{max} \rceil = 60$ ) on a Zynq Ultrascale+ ZCU102 board.

Module	LUT (%)	FF (%)	DSP (%)	BRAM (%)
Controller	6210	113	0	0
Coef Mem	4032	128	0	192
TF base Mem	0	0	0	16
Step Mem	166	0	0	0
PE Array	51,168	75,136	576	0
└ PE	1599	2348	18	0
└ MM	690	1388	18	0
TFG Array	9008	22,784	288	0
└ TFG	563	1424	18	0
SG	626	1346	18	0
Total	71,210 (26.0)	99,507 (18.15)	882 (35)	208 (22.8)

**Table 2.** The total storage capacity required for the NTT/INTT design.

Item	Coefficients			TF Bases		
	[26]	Ours	Reduction	[23]	Ours	Reduction
Capacity	98,304	81,920	16.7%	24,000	17,408	27.5%

Secondly, for the timing information, the coefficient conversion across  $\log_2 N$  stages requires  $\log_2 N \times N/2L$  Clock Cycles (CCs). Moreover, when  $L \geq 2$ , optimization of the TF generation strategy introduces an extra  $N/2L$  cycle. To avoid access conflict, a minimum



of  $g_{min}$  idle cycles are inserted between stages, allowing adequate time for the previous stage's results to be written back. Finally, considering the delay of the SG denoted by  $M$  and that of the PE denoted by  $d$ , the total CCs and latency of NTT/INTT operations are as depicted in Equations (20) and (21) [23], where  $f_{max}$  represents the maximal clock frequency achievable on the target FPGA platform.

$$CCs = \begin{cases} \text{if } L > 1, \\ (\log_2 N + 1) \times \frac{N}{2L} + M + d + g_{min} \times (\log_2 N - 1) \\ \text{otherwise,} \\ \log_2 N \times \frac{N}{2} + M + d + g_{min} \times (\log_2 N - 1) \end{cases} \quad (20)$$

$$\text{Latency(us)} = \frac{CCs}{f_{max}(\text{MHz})} \quad (21)$$

The data throughput can be measured as the maximum number of data bits processed by the NTT/INTT module per second, as depicted in Equation (22) [23].

$$\text{Throughput(Mbps)} = \frac{\text{Number of bits}}{\text{Latency(us)}} \quad (22)$$

Moreover, when the proposed architecture is extended to support  $p$  moduli, the data movement overhead of multiple polynomials with diverse moduli is not considered, similar to prior works [20,23]. Therefore, the total number of CCs and the maximal number of data bits supported by the architecture increase by a factor of  $p$ .

Finally, a larger number of PEs results in shorter latency and higher throughput but requires more resources. Therefore, comparing only the resource usage or throughput is one-sided. For a configurable architecture, area efficiency is a more comprehensive and fair performance evaluation standard. Commonly used metrics for quantifying area efficiency include “Area  $\times$  Time” Products (ATP) [35] and “Throughput Per Slice” (TPS) [36], which differ in how they equate area. Lower ATP and higher TPS indicate better area efficiency. In our comparison, we calculate both metrics, considering the diversity of platforms and resources.

#### 4.3. Comparison to Works Considering HE Parameters

In this part, our NTT performance results are compared with those reported in related works [19–23], as shown in Table 3. Our design offers the highest level of configurability while achieving optimal area efficiency, with lower ATP and higher TPS compared to other works, except for [21].

Öztürk et al. [19] reported a block-level NTT architecture for partial HE schemes. Large amounts of on-chip memory and multipliers are utilized for TF storage and computation, respectively, leading to inefficient hardware utilization. Compared with the design proposed in [19], our design achieves higher throughput, with  $5.17\times$  higher TPS values, and the ATP decreases by 85.46%. The architecture proposed in [22] enables a full pipeline by deploying many PEs in series. However, this comes at the cost of high memory capacity and bandwidth expenses due to the presence of five intermediate buffers for coefficient reordering, which is impractical for memory-bound homomorphic evaluations. In contrast, our proposed coefficient access strategy significantly reduces memory overhead. Specifically, when performing a  $2^{17}$ -point NTT operation with a modulus bit width of 62, their architecture requires at least 10080KB of memory resources for coefficients, while ours only requires 1440KB. In all, our design considers optimal hardware efficiency through configurable parallelism, and Table 3 shows that the proposed NTT architecture outperforms that proposed in [22] in terms of both performance metrics. Su et al. [20] proposed a multi-channel and multi-PE architecture based on the CG algorithm. Their design requires  $1.49\times$  more LUTs than ours, achieving  $2\times$  higher throughput. However, due to our optimized

memory access pattern and TF generation strategy, our design decreases BRAM resources by  $7.85\times$ . The results in terms of ATP and TPS indicate that our design has a higher area efficiency. In comparison with the design proposed in [23], our architecture decreases ATP by 31.38% and obtains  $1.25\times$  higher TPS. Due to the advantage of the carefully designed pipelined architecture of PE, our NTT can run at a higher clock frequency with the same parallelism and, thus, obtain  $1.6\times$  higher throughput. Although the compared design uses less BRAM thanks to the optimized coefficient memory bandwidth based on the mixed-radix algorithm, it is worth noting that our design requires less memory for TF bases, including capacity and bandwidth. Furthermore, when supporting 32 RNS moduli, the compared design requires 24,000 pre-stored TF bases, whereas 17,408 are required in ours. In addition, due to its inconsistent design, TF bases for NTT and INTT are stored in different memory blocks, while our unified architecture does not require extra TF base memory blocks for INTT. Kurniawan et al. [21] proposed a memory-based NTT architecture that supports RTC, achieving superior area efficiency. This improvement in area efficiency is primarily attributed to the low DSP consumption resulting from the use of a specific modulus form, which allows for minimal equivalent area. However, in terms of throughput, our design achieves a  $1.12\times$  improvement with the same level of computational parallelism while also offering significantly higher configurability and allowing for more RNS moduli.

**Table 3.** Our implementation results and comparison with prior works considering HE parameter sets. The **bold** text indicates better results.

Work	Platform <sup>d</sup>	N <sup>b</sup>	q <sup>b</sup>	P	Timing		Resource				Efficiency Indicators		Configurability <sup>c</sup>	
					Freq (MHz)	Latency (us)	LUT	FF	DSP	BRAM	ATP <sup>a,r</sup> /1000	TPS <sup>t,r</sup>	CTC	RTC
[19] <sup>g,t</sup>	V690	15	32	41	250	2086.9	219.2k	90.8k	768	193	978.5k	0.08	✗	✗
[22] <sup>s</sup>	VU190	17	62	42	200	3760	365k	335k	1.3k	2.3k	4420.3k	0.19	✗	✗
[20] <sup>g,t</sup>	V1140	15	32	41	250	245.8	194.1k	153.1k	1.7k	1.8k	218.7k	0.26	✓	✗
[23] <sup>s</sup>	ZU102	16	60	32	200	2684.2	148.5k	90.9k	564	137	660.4k	0.59	✗	✗
[21] <sup>s</sup>	VU37	16	60	1	250	66.5	74.5k	61.4k	288	315	13.2k	0.92	✗	✓
Our <sup>g,t</sup>	V690 <sup>1</sup>	15	32	41	240	1405.3	35.6k	51.2k	392	88	<b>142.3k</b> (14.54%)	<b>0.41</b> (5.17×)		
	VU190 <sup>1</sup>	17	62	42	145	10,687.7	75.4k	105.8k	1.4k	368	<b>3452.4k</b> (78.1%)	<b>0.25</b> (1.28×)		
	V1140 <sup>2</sup>	15	32	41	164	520.5	130.1k	201.6k	1.5k	224	<b>183.3k</b> (83.83%)	<b>0.31</b> (1.20×)	✓	✓
	ZU102 <sup>1</sup>	16	60	32	285	1958.4	71.2k	99.5k	882	240	<b>453.2k</b> (68.62%)	<b>0.73</b> (1.25×)		
	VU37 <sup>1</sup>	16	60	1	295	59.1	72.7k	99.6k	882	208	13.2k (100%)	0.79 (0.86×)		

<sup>d</sup>: V690: Virtex-7 XC7VX690T; VU190: Virtex Ultrascale+ XCVU190; V1140: Virtex-7 XC7VX1140T; ZU102: Zynq Ultrascale+ ZCU102; VU37: Virtex Ultrascale+ XCVU37P; <sup>b</sup>: the bit width of the N or q parameter; <sup>a</sup>: ATP = Equ. LUT × latency, in which, Equ. LUT = (LUT + 100 × DSP + 300 × BRAM) [35]; <sup>t</sup>: TPS = Throughput (Mbps) / Equ. slice, in which, for 7 series, Equ. slice ≈ (LUT/4 + FF/8) × 0.97 + 102.4 × DSP + 232.4 × BRAM; for ultrascale series, Equ. slice ≈ (LUT/8 + LUT/16) × 0.97 + 51.2 × DSP + 116.2 × BRAM [36,37]; <sup>r</sup>: the ratio calculated by dividing our ATP or TPS by the ATP or TPS of other works; <sup>c</sup>: the symbol ✗ indicates no support for the corresponding configurability, while ✓ indicates support; <sup>g</sup>: general modulus; <sup>s</sup>: specific form of modulus; <sup>1</sup>: 32 PEs; <sup>2</sup>: 128 PEs; <sup>†</sup>: the unified architecture supporting NTT and INTT.

In addition, among the aforementioned works, only [22,23] implemented an online generation strategy for TFs, which is closer to impractical HE applications. In comparison with the compared designs, when supporting multiple moduli, our architecture incurs only a slight increase in BRAM usage for storage of the TF bases of different moduli. However, the compared designs not only require additional BRAM but also necessitate an increase in LUT resources. This is because the compared MM designs are optimized for specific sets with low Hamming weights, utilizing shifters for lightweight integer multiplication. Consequently, each additional supported modulus leads to increases in shifters and multiplexers. Instead, the bit-width range of the moduli supported by our MM at runtime is  $(W, \lceil \log_2 q_{max} \rceil]$ . When  $\lceil \log_2 q_{max} \rceil = 62$  and  $W = 17$ , our architecture

accommodates moduli ranging from 18 to 62 bits, which covers all RNS moduli proposed in prior works, without incurring any additional computation overhead. Furthermore, the designs proposed in [22,23] only support a fixed number of PEs, lacking configurability for architecture parallelism. In contrast, our design supports both CTC and RTC. We provide a unified architecture for NTT and INTT, avoiding redundant design efforts.

In all, our NTT/INTT module achieves improvements in terms of area efficiency and configurability. These advancements establish it as a practical NTT accelerator for RNS-based HE applications.

#### 4.4. Comparison with Works Considering Small Parameters

For small parameter sets, NTT designs typically use on-chip storage for TFs due to their less stringent storage constraints. To ensure a fair comparison with literature works, we use the TF storage strategy proposed in [26], combined with our optimized coefficient memory access method, to create a memory-based NTT design. The comparison results between our memory-based NTT and prior works considering small parameters are shown in Table 4.

**Table 4.** Our implementation results and comparison with prior works considering small parameter sets. The **bold** text indicates better results.

Work <sup>d</sup>	N <sup>b</sup>	q <sup>b</sup>	PE	Timing		Resource				ATP				ATP <sup>a,k</sup>	Configurability <sup>c</sup>	
				Freq (MHz)	Latency (us)	LUT	FF	DSP	BRAM	LUT <sup>k</sup>	FF <sup>k</sup>	DSP	BRAM		CTC	RTC
[38] <sup>†,i</sup>	12	24	1	185	132.88	802	525	4	7	106.6	69.8	532	930	438.8	✓	✗
			4 × 2	157	19.61	5.7k	3.2k	33	8.5	111.1	62.5	647	167	225.8		
			8 × 2	121	12.75	14.6k	6.5k	80	12	186.1	82.5	1020	153	334		
our <sup>†,o</sup>	12	24	1	243	101	1.1k	1.1k	6	7	107.2	113.7	607	<b>708</b>	<b>380.4</b>	✓	✓
			8	219	14.11	5.8k	5.3k	48	16	<b>82.4</b>	75.1	677	226	<b>217.9</b>		
			16	216	7.19	11.3k	10.2k	96	24	<b>81.1</b>	<b>73.3</b>	<b>691</b>	173	<b>202</b>		
[27] <sup>†,o</sup>	10	32	8	244	2.67	9.5k	4.7k	64	12	25.3	12.6	171	32	52.1	✓	✗
	12	32	4	244	25.2	5k	2.8k	32	14	126	70.6	806	353	312.5		
[39] <sup>†,i</sup>	12	32	-	200	2.3	70k	-	599	129	161	-	296.7	1377.7	387.8	✓	✗
our <sup>†,o</sup>	10	32	8	233	2.82	7.4k	7.1k	64	12	<b>21</b>	20	181	34	<b>49.3</b>	✓	✓
	12	32	4	245	25.15	4.2k	3.9k	32	14	<b>105.8</b>	97.5	<b>805</b>	<b>352</b>	<b>291.9</b>		
[26] <sup>†,o</sup>	12	60	1	154	159.6	1.9k	1.8k	42	17	303.2	287.3	6703	2713	1787.5	✓	✓
			8	150	20.54	14.1k	12.5k	336	41	289.7	256.8	6901	842	1232.4		
			32	133	5.8	52k	47k	1300	160	301.6	272.6	7540	928	1334		
[24] <sup>†,i</sup>	12	60	1	144	170.8	2.6k	2.5k	26	14	444	426.9	4440	2390.7	1605.2	✓	✓
			8	141	21.89	22.1k	19.5k	208	32	483.7	426.8	4552.4	700.4	1149		
			32	130	6.02	89.9k	77.2k	832	96	540.8	464.4	5004.8	577.5	1214.5		
[17] <sup>i</sup>	12	60	64	125	7.8	338k	138k	1984	768	2636.4	1076.4	15475	5990	5981	✓	✗
our <sup>†,o</sup>	12	60	1	198	124.2	2.2k	2.7k	28	17	<b>269.4</b>	323.5	<b>3478</b>	<b>2111</b>	<b>1250.6</b>	✓	✓
			8	180	17.16	15k	13.4k	224	32	<b>258.1</b>	<b>230.2</b>	<b>3843</b>	<b>549</b>	<b>807.1</b>		
			32	137	5.74	60k	54.4k	896	96	344	312.4	<b>5141</b>	<b>551</b>	<b>1023.2</b>		
			64	120	3.35	118.4k	108.1k	1792	192	<b>396.8</b>	<b>362.3</b>	<b>6003</b>	<b>643</b>	<b>1190.1</b>		

<sup>d</sup>: Based on Xilinx's Virtex-7 series of FPGAs; <sup>b</sup>: the bit-width of the N or q parameter; <sup>a</sup>: ATP = Equ. LUT × latency, in which, Equ. LUT = (LUT + 100 × DSP + 300 × BRAM) [35]; <sup>k</sup>: the ATP value divided by 1000; <sup>c</sup>: the symbol ✗ indicates no support for the corresponding configurability, while ✓ indicates support; <sup>†</sup>: the unified architecture supporting NTT and INTT; <sup>i</sup>: in place; <sup>o</sup>: out of place.

The designs proposed in [17,24,26,27,38] support the configuration of parameters N,  $\lceil \log_2 q_{max} \rceil$ , and L at compile time, and that proposed in [24,26] also supports diverse polynomial degrees at runtime. In comparison, our design offers greater scalability, with three

compilable parameters, namely  $N$ ,  $\lceil \log_2 q_{max} \rceil$ , and  $L$ , further enabling runtime adjustments of  $N$  and different sizes of  $q$ . Furthermore, compared to the design proposed in [38], our architecture demonstrates poorer ATP in LUT when  $L = 1$  but higher LUT efficiency for larger values of  $L$ . Moreover, our overall ATP exhibits superior performance due to its lower latency. In comparison with the design proposed in [27], BRAM consumption is the same because each PE requires two memory blocks for coefficient storage and one block for twiddle factor storage in both designs. But it is worth noting that the actual storage consumption in our design only slightly exceeds  $2.25N$ , whereas that of the compared design is approximately  $3N$ . Mert et al. [39] employed a fully parallelized architecture based on the four-step algorithm to speed up NTT but at the cost of significant resources. In comparison, our design is more area-efficient. Liu et al. [26] proposed a configurable NTT/INTT accelerator that supports both CTC and RTC while decreasing the actual memory overhead to  $2.5N$ . Their design consumes considerably fewer LUT resources because of its simplified memory access pattern. However, our design is more area-efficient thanks to its lower BRAM usage. Our architecture also outperforms the designs proposed in [24] and [17] in terms of ATP.

In general, benefiting from the proposed coefficient memory pattern, our design can fully exploit storage resources and obtain higher LUT and BRAM efficiencies than many previously proposed architectures.

#### 4.5. Comparison between Memory-Based NTT and Online Generation-Based NTT

In the preceding discussion, we learned that there are two methods to provide TFs for PE arrays, namely (i) memory-based, and (ii) online generation-based methods. This part presents a theoretical analysis of how these methods impact area efficiency.

To simplify the analysis, let us assume that all operations occur on the chip to remove the impact of data movement. This implies that storing the TFs for (i) and the TF bases for (ii) across multiple moduli in the internal memory is necessary. Moreover, we assume that only one RNS channel with one NTT core is deployed, consisting of  $L$  PE units. Consequently, the total constant storage capacity increases linearly with multiple moduli. The NTT/INTT domain transformations of polynomials across  $p$  moduli must be executed serially.

For method (i), according to [26],  $L$  memory blocks with a depth of  $(N/L + \log_2 L - 1) \times p$  are required for TFs. Method (ii) requires  $\lceil L/2 \rceil$  memory blocks for TF bases, each with a maximum depth of  $(\log_2 N + 1) \times 2 \times p$ . Table 5 presents the BRAM consumption of the proposed memory-based and online generation-based NTT architectures on a Xilinx ZCU102 FPGA when the parameter set of  $(N, \lceil \log_2 q_{max} \rceil, p)$  is set as  $(2^{16}, 60, 32)$ . Notably, the BRAM overhead for method (i) is approximately  $10.91 \times \sim 25.84 \times$  more than that required for method (ii).

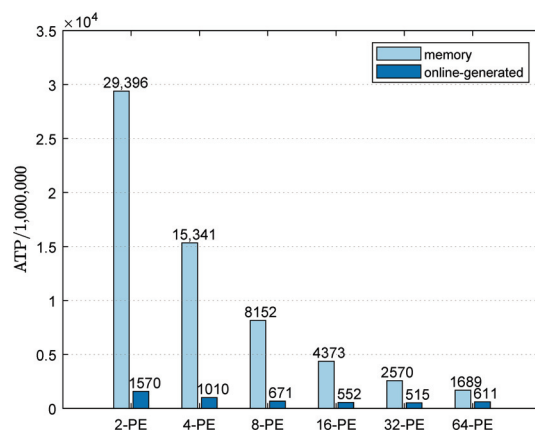
**Table 5.** BRAM consumption of the proposed memory-based NTT and online generation-based NTT with the parameter set of  $(N = 2^{16}, \lceil \log_2 q_{max} \rceil = 60, p = 32)$  on a Xilinx ZCU102.

$L$	2	4	8	16	32	64
Method (i)	3592	3604	3616	3648	3712	3840
Method (ii)	139	146	156	184	240	352
Ratio	25.84×	24.68×	23.18×	19.83×	15.47×	10.91×

In addition to the variances in BRAM occupancy, distinctions exist in latency and the overhead of LUT and DSP resources in both approaches. Therefore, we provide the ATP values of our proposed NTT architectures under the parameter set of  $(N = 2^{16}, \lceil \log_2 q_{max} \rceil = 60, p = 32)$ , as shown in Figure 10. The required operating frequency of the architecture for latency computation is considered to be consistent to the maximum operating frequency under a single modulus. It is observed that under the same computational parallelism, the online generation-based NTT architecture significantly out-

performs the memory-based architecture in terms of area efficiency. Furthermore, the ATP of the memory-based architecture gradually decreases as  $L$  increases, indicating that expanding computation units can mitigate performance limitations resulting from storage requirements. Meanwhile, the ATP of the online generation-based architecture reaches its minimum when  $L = 32$ . This suggests that the primary factor influencing performance enhancement is the availability of computational resources when  $L < 32$ . However, when  $L \geq 32$ , performance improvement becomes constrained due to the increasing storage bandwidth.

In general, our comparison results highlight the necessity of generating TFs on the fly in RNS-based HE applications. Moreover, in high-dimensional polynomials, NTT designs with high parallelism are advantageous for enhancing area efficiency.



**Figure 10.** Comparisons of ATP for different values of  $L$  between memory-based NTT and online generation-based NTT.

## 5. Conclusions and Future Works

This paper proposes an area-efficient and flexible NTT architecture suitable for RNS-based HE evaluations. The proposed conflict-free and low-complexity memory access pattern reduces the total coefficient storage requirements in CG-based NTT design. And the carefully designed twiddle factor generator saves a significant amount of memory for large sets of prime moduli. Furthermore, the proposed unified architecture offers CTC, which can be configured with various numbers of computational units and support multiple polynomial degrees and various sizes of moduli after compilation. We evaluated the proposed design under a wide range of parameters to demonstrate its advantages in terms of performance and configurability.

Future work will utilize the proposed NTT architecture to develop more complex HE operations, such as bootstrapping and key switching. Additionally, when the proposed accelerator is integrated into a system, the data transfer overhead for various moduli should also be considered.

**Author Contributions:** Conceptualization, J.H. and C.K.; methodology, J.H. and S.L.; software, J.H.; validation, J.H.; investigation, J.H. and C.K.; resources, T.S.; data curation, J.H. and C.K.; writing—original draft preparation, J.H.; writing—review and editing, S.L. and T.S.; project administration, J.H. and T.S.; funding acquisition, T.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Science and Technology Program of Guangdong Province under Grant 2022B0701180001.

**Data Availability Statement:** The data presented in this study are available upon request from the corresponding author. The data are not publicly available due to privacy concerns.

**Conflicts of Interest:** The authors declare no conflicts of interest.



## References

1. Lin, J.; Qian, J. A Multi-Party Secure SaaS Cloud Accounting Platform Based on Lattice-Based Homomorphic Encryption System. In Proceedings of the 2021 International Conference on Public Management and Intelligent Society (PMIS), Shanghai, China, 26–28 February 2021; pp. 1–4. [CrossRef]
2. Brutzkus, A.; Elisha, O.; Gilad-Bachrach, R. Low Latency Privacy Preserving Inference. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.
3. Lin, H.; Deng, X.; Yu, F.; Sun, Y. Grid Multi-Butterfly Memristive Neural Network with Three Memristive Systems: Modeling, Dynamic Analysis, and Application in Police IoT. *IEEE Internet Things J.* **2024**, 1–11. [CrossRef]
4. Lin, H.; Deng, X.; Yu, F.; Sun, Y. Diversified Butterfly Attractors of Memristive HNN with Two Memristive Systems and Application in IoMT for Privacy Protection. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, 1–12. [CrossRef]
5. Balbás, D. The Hardness of LWE and Ring-LWE: A Survey. *IACR Cryptol. ePrint Arch.* **2021**, 2021, 1358.
6. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* **2014**, 6, 1–36. [CrossRef]
7. Brakerski, Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology—CRYPTO 2012, Proceedings of the 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012*; Safavi-Naini, R., Canetti, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886.
8. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology—ASIACRYPT 2017, Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017*; Takagi, T., Peyrin, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 409–437.
9. Feldmann, A.; Samardzic, N.; Krastev, A.; Devadas, S.; Dreslinski, R.; Eldefrawy, K.; Genise, N.; Peikert, C.; Sanchez, D. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption (Extended Version). *arXiv* **2021**, arXiv:2109.05371.
10. Kim, J.; Lee, G.; Kim, S.; Sohn, G.; Kim, J.; Rhu, M.; Ahn, J.H. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, 1–5 October 2022; pp. 1237–1254. [CrossRef]
11. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, 2018, 238–268. [CrossRef]
12. Alkim, E.; Barreto, P.S.L.M.; Bindel, N.; Krämer, J.; Longa, P.; Ricardini, J.E. The Lattice-Based Digital Signature Scheme qTESLA. In Proceedings of the Applied Cryptography and Network Security, Rome, Italy, 19–22 October 2020; Conti, M., Zhou, J., Casalicchio, E., Spognardi, A., Eds.; Springer: Cham, Switzerland, 2020; pp. 441–460.
13. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography—SAC 2018, Proceedings of the 25th International Conference, Calgary, AB, Canada, 15–17 August 2018*; Springer: Cham, Switzerland, 2019; pp. 347–368.
14. Bajard, J.C.; Eynard, J.; Hasan, M.A.; Zucca, V. A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes. In *Selected Areas in Cryptography—SAC 2016, Proceedings of the 23rd International Conference, St. John's, NL, Canada, 10–12 August 2016*; Avanzi, R., Heys, H., Eds.; Springer: Cham, Switzerland, 2017; pp. 423–442.
15. Chen, H.; Chillotti, I.; Song, Y. Improved Bootstrapping for Approximate Homomorphic Encryption. In Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, 19–23 May 2019; Springer: Cham, Switzerland, 2019; pp. 34–54.
16. Han, K.; Ki, D. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology—CT-RSA 2020, Proceedings of the Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, 24–28 February 2020*; Jarecki, S., Ed.; Springer: Cham, Switzerland, 2020; pp. 364–390.
17. Mert, A.C.; Karabulut, E.; Ozturk, E.; Savas, E.; Becchi, M.; Aysu, A. A Flexible and Scalable NTT Hardware: Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 346–351. [CrossRef]
18. Mert, A.C.; Öztürk, E.; Savaş, E. FPGA Implementation of a Run-Time Configurable NTT-based Polynomial Multiplication Hardware. *Microprocess. Microsyst.* **2020**, 78, 103219. [CrossRef]
19. Ozturk, E.; Doroz, Y.; Savas, E.; Sunar, B. A Custom Accelerator for Homomorphic Encryption Applications. *IEEE Trans. Comput.* **2017**, 66, 3–16. [CrossRef]
20. Su, Y.; Yang, B.L.; Yang, C.; Zhao, S.Y. ReMCA: A Reconfigurable Multi-Core Architecture for Full RNS Variant of BFV Homomorphic Evaluation. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, 69, 2857–2870. [CrossRef]
21. Kurniawan, S.; Duong-Ngoc, P.; Lee, H. Configurable Memory-Based NTT Architecture for Homomorphic Encryption. *IEEE Trans. Circuits Syst. II Express Briefs* **2023**, 70, 3942–3946. [CrossRef]
22. Kim, S.; Lee, K.; Cho, W.; Nam, Y.; Cheon, J.H.; Rutenbar, R.A. Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme. In Proceedings of the 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Fayetteville, AR, USA, 3–6 May 2020; pp. 56–64. [CrossRef]
23. Duong-Ngoc, P.; Kwon, S.; Yoo, D.; Lee, H. Area-Efficient Number Theoretic Transform Architecture for Homomorphic Encryption. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, 70, 1270–1283. [CrossRef]



24. Hu, X.; Tian, J.; Li, M.; Wang, Z. AC-PM: An Area-Efficient and Configurable Polynomial Multiplier for Lattice Based Cryptography. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 719–732. [CrossRef]
25. Su, Y.; Yang, B.L.; Yang, C.; Yang, Z.P.; Liu, Y.W. A Highly Unified Reconfigurable Multicore Architecture to Speed Up NTT/INTT for Homomorphic Polynomial Multiplication. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2022**, *30*, 993–1006. [CrossRef]
26. Liu, S.H.; Kuo, C.Y.; Mo, Y.N.; Su, T. An Area-Efficient, Conflict-Free, and Configurable Architecture for Accelerating NTT/INTT. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2024**, *32*, 519–529. [CrossRef]
27. Geng, Y.; Hu, X.; Li, M.; Wang, Z. Rethinking Parallel Memory Access Pattern in Number Theoretic Transform Design. *IEEE Trans. Circuits Syst. II Express Briefs* **2023**, *70*, 1689–1693. [CrossRef]
28. Banerjee, U.; Ukyab, T.S.; Chandrakasan, A.P. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-Based Protocols (Extended Version). *TCHES* **2019**, *2019*, 17–61. [CrossRef]
29. Pöppelmann, T.; Güneysu, T. Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware. In *Progress in Cryptology—LATINCRYPT 2012*; Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., et al., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7533; pp. 139–158. [CrossRef]
30. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
31. Gentleman, W.M.; van der Sande, G. Fast Fourier Transforms: For fun and profit. In Proceedings of the AFIPS '66 (Fall), San Francisco, CA, USA, 7–10 November 1966.
32. Pöppelmann, T.; Oder, T.; Güneysu, T. High-Performance Ideal Lattice-Based Cryptography on 8-Bit ATxmega Microcontrollers. In *Progress in Cryptology—LATINCRYPT 2015, Proceedings of the 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, 23–26 August 2015*; Lauter, K., Rodríguez-Henríquez, F., Eds.; Springer: Cham, Switzerland, 2015; pp. 346–365.
33. Mert, A.C.; Öztürk, E.; Savaş, E. Design and Implementation of a Fast and Scalable NTT-Based Polynomial Multiplier Architecture. In Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 28–30 August 2019; pp. 253–260. [CrossRef]
34. Barrett, P. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology—CRYPTO' 86*; Odlyzko, A.M., Ed.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 263, pp. 311–323. [CrossRef]
35. Ye, Z.; Cheung, R.C.C.; Huang, K. PipeNTT: A Pipelined Number Theoretic Transform Architecture. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 4068–4072. [CrossRef]
36. Kundi, D.e.S.; Zhang, Y.; Wang, C.; Khalid, A.; O'Neill, M.; Liu, W. Ultra High-Speed Polynomial Multiplications for Lattice-Based Cryptography on FPGAs. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1993–2005. [CrossRef]
37. Liu, W.; Fan, S.; Khalid, A.; Rafferty, C.; O'Neill, M. Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 2459–2463. [CrossRef]
38. Mu, J.; Ren, Y.; Wang, W.; Hu, Y.; Chen, S.; Chang, C.H.; Fan, J.; Ye, J.; Cao, Y.; Li, H.; et al. Scalable and Conflict-Free NTT Hardware Accelerator Design: Methodology, Proof, and Implementation. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2023**, *42*, 1504–1517. [CrossRef]
39. Mert, A.C.; Öztürk, E.; Savaş, E. Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 353–362. [CrossRef]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# FPGA Implementation of Pillar-Based Object Classification for Autonomous Mobile Robot

Chaewoon Park<sup>1</sup>, Seongjoo Lee<sup>2,3</sup> and Yunho Jung<sup>1,4,\*</sup>

<sup>1</sup> School of Electronics and Information Engineering, Korea Aerospace University, Goyang-si 10540, Republic of Korea; pcw0201@kau.kr

<sup>2</sup> Department of Electrical Engineering, Sejong University, Seoul 05006, Republic of Korea; seongjoo@sejong.ac.kr

<sup>3</sup> Department of Convergence Engineering of Intelligent Drone, Sejong University, Seoul 05006, Republic of Korea

<sup>4</sup> Department of Smart Air Mobility, Korea Aerospace University, Goyang-si 10540, Republic of Korea

\* Correspondence: yjung@kau.ac.kr; Tel.: +82-2-300-0133

**Abstract:** With the advancement in artificial intelligence technology, autonomous mobile robots have been utilized in various applications. In autonomous driving scenarios, object classification is essential for robot navigation. To perform this task, light detection and ranging (LiDAR) sensors, which can obtain depth and height information and have higher resolution than radio detection and ranging (radar) sensors, are preferred over camera sensors. The pillar-based method employs a pillar feature encoder (PFE) to encode 3D LiDAR point clouds into 2D images, enabling high-speed inference using 2D convolutional neural networks. Although the pillar-based method is employed to ensure real-time responsiveness of autonomous driving systems, research on accelerating the PFE is not actively being conducted, although the PFE consumes a significant amount of computation time within the system. Therefore, this paper proposes a PFE hardware accelerator and pillar-based object classification model for autonomous mobile robots. The proposed object classification model was trained and tested using 2971 datasets comprising eight classes, achieving a classification accuracy of 94.3%. The PFE hardware accelerator was implemented in a field-programmable gate array (FPGA) through a register-transfer level design, which achieved a 40 times speedup compared with the firmware for the ARM Cortex-A53 microprocessor unit; the object classification network was implemented in the FPGA using the FINN framework. By integrating the PFE and object classification network, we implemented a real-time pillar-based object classification acceleration system on an FPGA with a latency of 6.41 ms.

**Keywords:** field-programmable gate array; autonomous mobile robot; object classification; pillar feature encoder; FINN

## 1. Introduction

Autonomous mobile robots move by recognizing their surroundings and performing physical tasks performed by people in the past. Advances in computer vision, robot control, and artificial intelligence (AI) have enhanced the ability of autonomous mobile robots to perceive and respond to their environment. Consequently, these robots are utilized in several areas of human life, including manufacturing, transportation, healthcare, and daily living [1,2]. Object classification is essential for autonomous mobile robots for identifying objects and planning travel paths [3].

Various types of sensors such as cameras and radio detection and ranging (radar) sensors have been used to perceive the surrounding environment in autonomous driving situations. Camera sensors cannot obtain depth information, making it impossible to accurately determine the distance to an object, which can lead to serious issues when the two objects overlap [4]. In addition, camera sensors are sensitive to brightness and

have a limited field of view, which results in operational constraints [3]. Radar sensors cannot determine the height information of objects and have very low resolution for angles, preventing the precise observation of objects [5]. In contrast, point clouds acquired by light detection and ranging (LiDAR) sensors contain 3D physical coordinate information that enables the acquisition of depth and height information regarding objects. Furthermore, there are fewer operational constraints, and the high resolution allows for precise observation of objects [3]. Therefore, the use of point clouds acquired by LiDAR sensors is preferred for autonomous driving.

Various deep learning algorithms are available for processing point clouds. MVCNN [6] projects a point cloud in multiple directions and classifies objects using a convolutional neural network (CNN) based on the projected images. Although it uses a simple method to convert 3D data into 2D images and utilize a CNN, information loss occurs during the projection process [7]. PointNet [8] is a permutation-invariant network designed using a symmetry function to handle unordered point sets. VoxelNet [9] encodes features on a voxel-by-voxel basis using PointNet and conducts inferences using a region proposal network. Because the inference is based on a 3D CNN, a long inference time of 225 ms is required [10]. SECOND [11] achieved an improved inference speed of 50 ms by applying a sparse CNN to VoxelNet, but the complexity of the 3D CNN computation remained high. PointPillars [12] proposed a pillar-based method for encoding 3D point clouds into 2D images. Unlike MVCNN, which only uses information from the highest element, the pillar-based method uses information from all points within the pillar. Conducting 2D CNN-based inference on an encoded image enables pillar-based methods to achieve high-speed inference and excellent performance [13–16].

Autonomous mobile robots must have real-time response time to prevent physical accidents. Under these constraints, running microprocessor unit (MPU)-based pillar-based methods are extremely time-consuming [17]. Graphic processing units (GPUs) can accelerate inference based on 2D CNNs; however, using GPUs on edge devices is inefficient in terms of power consumption [18]. Therefore, refs. [19–21] implemented a hardware accelerator for pillar-based methods using a field-programmable gate array (FPGA). However, a pillar feature encoder (PFE) was computed on an MPU in [19], requiring 71.2 ms and accounting for 19% of the total response time. Similarly, a PFE was computed on an MPU in 9.51 ms in [20], which was 22% of the total response time. Additionally, a portion of the PFE was accelerated in [21], requiring 37.7 ms for computation, which accounted for 59% of the total response time. Because the PFE occupies a significant portion of the overall response time of the pillar-based method, a hardware accelerator for PFE is useful.

The pillar-based method can perform inference on an image encoded by the PFE with a network based on a depth-wise separable CNN (DSCNN). The DSCNN used in [22–24] has been widely adopted in recent CNN models owing to its low computation cost and number of parameters. Because of these advantages, DSCNNs are often used in resource-constrained or latency-sensitive applications [25]. Additionally, despite their low complexity, they have shown successful performance in image classification models by enhancing the representational efficiency [26].

In this study, we propose an efficient pillar-based object classification model for autonomous mobile robots using a PFE and DSCNN, achieving a classification accuracy of 94.3% through training and validation with datasets acquired using Ouster OS1-32 [27]. In addition, we propose an FPGA-based acceleration system for the real-time operation of the proposed object classification model. The proposed acceleration system comprises a PFE accelerator and classification network accelerator. We designed the PFE accelerator to achieve a computation time of 0.23 ms and confirmed a 40 times acceleration performance compared with the MPU. A classification network accelerator was implemented on an FPGA using FINN. For operation in edge environments, we implemented the proposed acceleration system on an FPGA and confirmed a real-time response time of 6.41 ms.

The remainder of this paper is structured as follows. Section 2 provides an overview of the proposed acceleration system, including the PFE and DSCNN-based classification

networks, dataset used, model configuration, and evaluation results. The design of the proposed acceleration system is described in Section 3. Section 4 presents the implementation results and acceleration effects. Finally, Section 5 concludes the study.

## 2. Proposed System

Figure 1 shows an overview of the proposed acceleration system. The point cloud input to the system is encoded into an image by the PFE through three steps: (a) converting the point cloud into a set of pillar units, (b) adding hand-crafted features (HCFs), and (c) expanding the features through point-wise (PW) convolution, compressing them through a max-pool operation, and scattering them back to their original position on the  $x$ - $y$  plane to generate an image. The generated image is then fed into a classification network to classify the objects. The classification network comprises a DSCNN-based network.

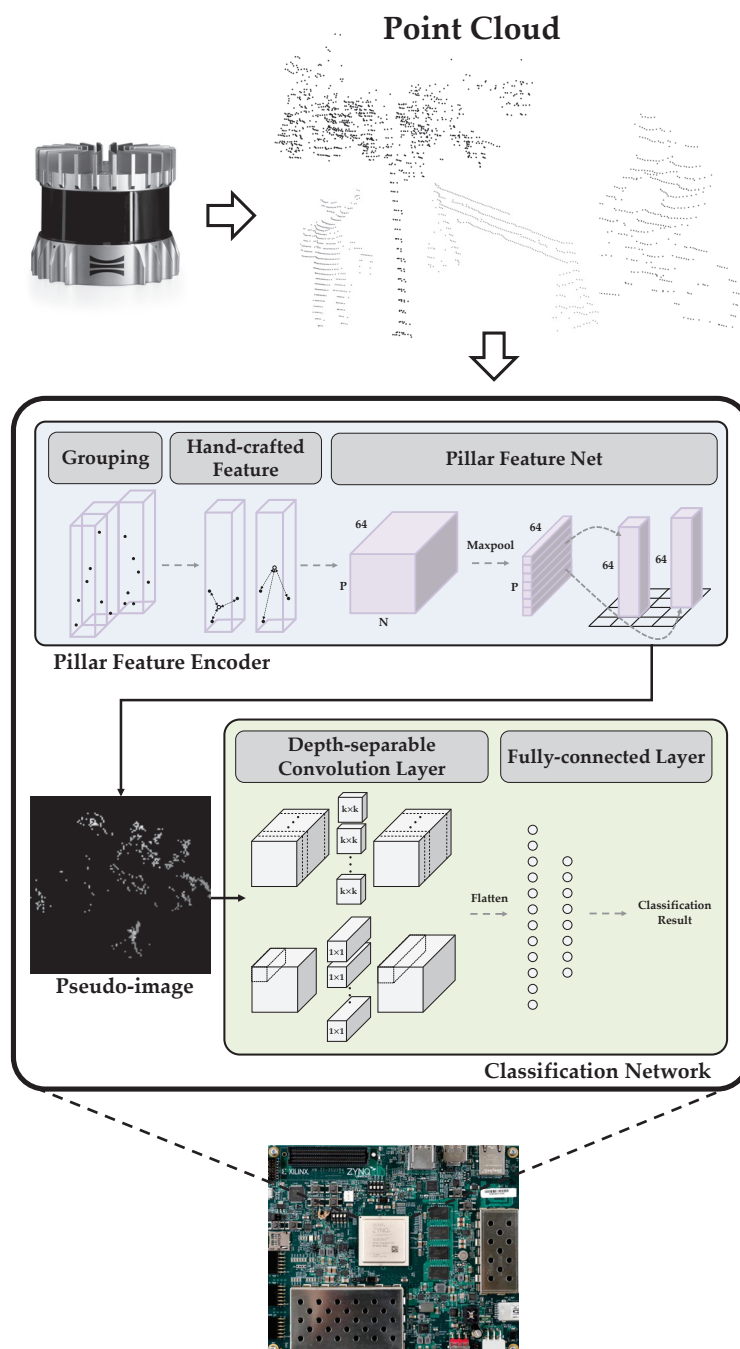


Figure 1. Overview of the proposed acceleration system.

### 2.1. PFE

The PFE, which encodes 3D point clouds into 2D images to enable 2D CNN-based inference, was proposed in [12]. First, a grid is generated by discretizing the  $x$ - $y$  plane, and then the point cloud is converted into a set of pillar units. The point cloud set  $\mathbf{g}$  is defined by Equation (1).

$$\mathbf{g} = [\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_i, \dots, \mathbf{p}_P] \quad (1)$$

We set  $\mathbf{g}$  as a set of  $\mathbf{p}_i$ , where  $P$  is the maximum number of pillars that contains points. We set  $\mathbf{p}_i$  as defined by Equation (2).

$$\mathbf{p}_i = [\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_i, \dots, \mathbf{n}_N] \quad (2)$$

We set  $\mathbf{p}_i$  as a set of  $\mathbf{n}_j$ , where  $N$  is the maximum number of points that a pillar can contain. The set  $\mathbf{n}_j$  is defined as in Equation (3).

$$\mathbf{n}_i = [x, y, z, r] \quad (3)$$

The set  $\mathbf{n}_j$  comprises the  $x$ ,  $y$ ,  $z$  coordinates and reflectivity values of the points. Finally, the point cloud is represented by a tensor of size  $(P, N, 4)$ , where  $P$  and  $N$  are hyperparameters. If the number of points in a pillar is greater than or equal to  $N$ , random sampling is used; if it is less than  $N$ , zero padding is used.

After the point cloud is converted into a set of pillar units, an HCF is added to set  $\mathbf{n}_j$ . The HCF comprises the differences  $x_m, y_m, z_m$  between the arithmetic mean coordinates of the points inside the pillar and  $x, y, z$  coordinates of each point, and the differences  $x_c, y_c$  between the center coordinates of the pillar and the  $x, y$  coordinates of each point. Setting  $\mathbf{n}_j$  with the appended HCF is given by Equation (4).

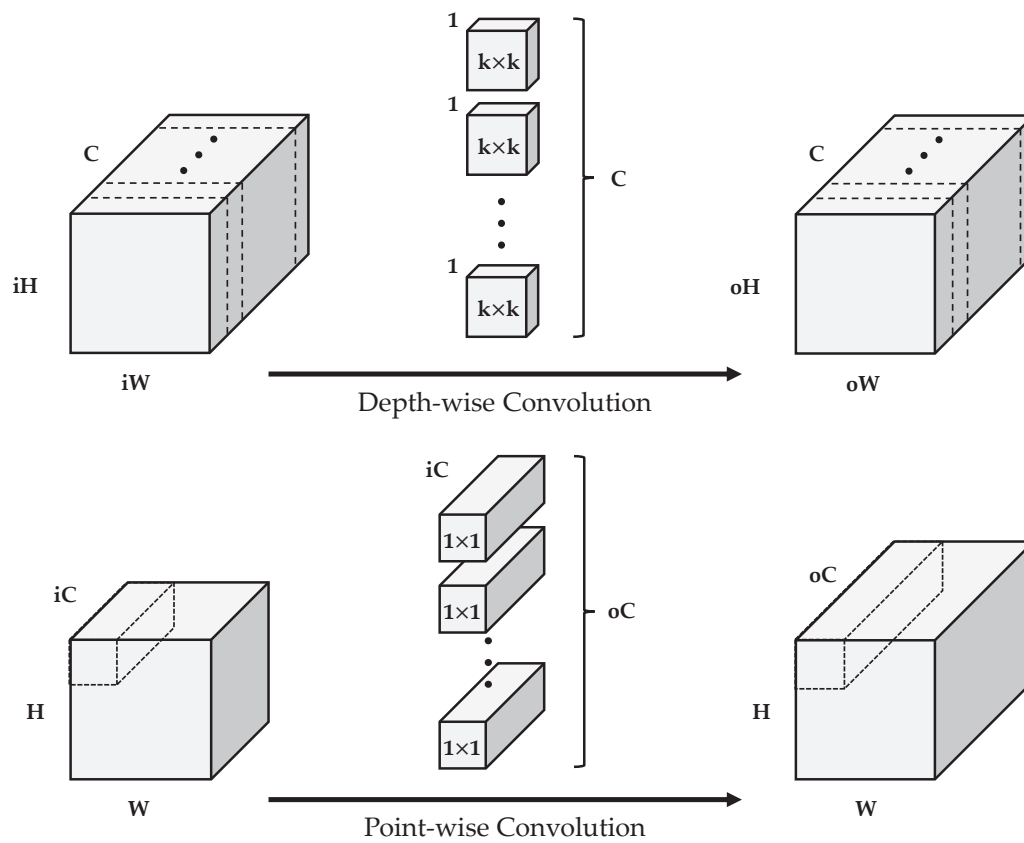
$$\mathbf{n}_i = [x, y, z, r, x_m, y_m, z_m, x_c, y_c] \quad (4)$$

A tensor of size  $(P, N, 9)$  with an added HCF undergoes PW convolution, batch normalization, and a ReLU layer to transform it into a tensor of size  $(P, N, 64)$ . Subsequently, maximum pooling is applied along the second dimension to yield a tensor of size  $(P, 64)$ . Finally, each pillar is scattered back to the original  $x$ - $y$  plane, completing the 2D image data with 64 channels.

### 2.2. Classification Network

The proposed classification network is based on the DSCNN. Figure 2 shows how the DSCNN works. The DSCNN comprises the depth-wise (DW) and PW methods. In DW, the number of channels in the input image, filters, and channels in the output image are the same. Each filter has only one channel, and performs a convolution operation on only one channel in the input image to create a one channel output image. For example, in the DW convolution shown in Figure 2, one channel of the input image, separated by a dashed line, is performed a convolution operation with one filter to produce one channel of the output image, also separated by a dashed line. Let  $iC$  be the number of channels in the input image and  $oC$  be the number of channels in the output image; PW uses  $oC$  filters of size  $1 \times 1$  with  $iC$  channels. Each filter performs a convolution operation on all the channels in the input image to create one channel of the output image. For example, in the PW convolution shown in Figure 2, one pixel of the input image, separated by a dashed line, completes one pixel of the output image, separated by a dashed line, by applying all filters in the convolution operation.





**Figure 2.** Operation mechanism of DSCNN.

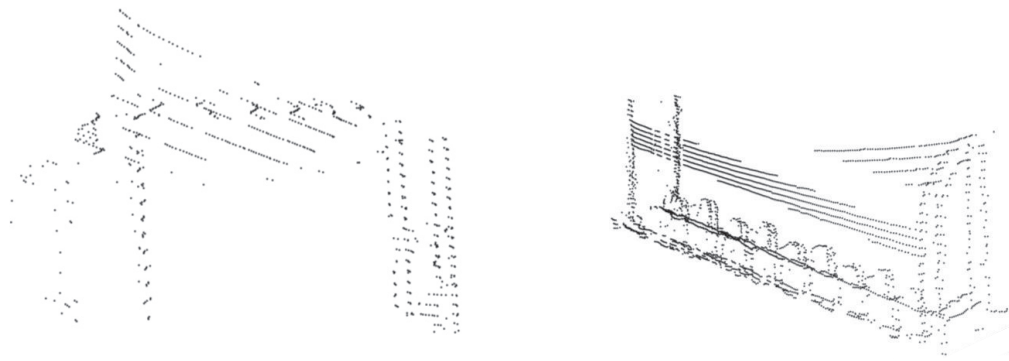
### 2.3. Dataset

To train the proposed object classification model, we obtained point clouds using an Ouster OS1-32 sensor. Figure 3 illustrates examples from a dataset comprising objects typically encountered by autonomous mobile robots in their driving environments. To enhance the training quality, we augmented the dataset using samples from the nuScenes [28] dataset. The dataset includes classes such as building, tree, vehicle, bicycle, obstacle, greenery, person, and urban fixture (street light, bicycle rack, traffic light, etc.), as depicted in Figure 4, which shows the configuration of the dataset. Out of a total of 2971 data points, 2617 were used for training and 354 were used for validation.

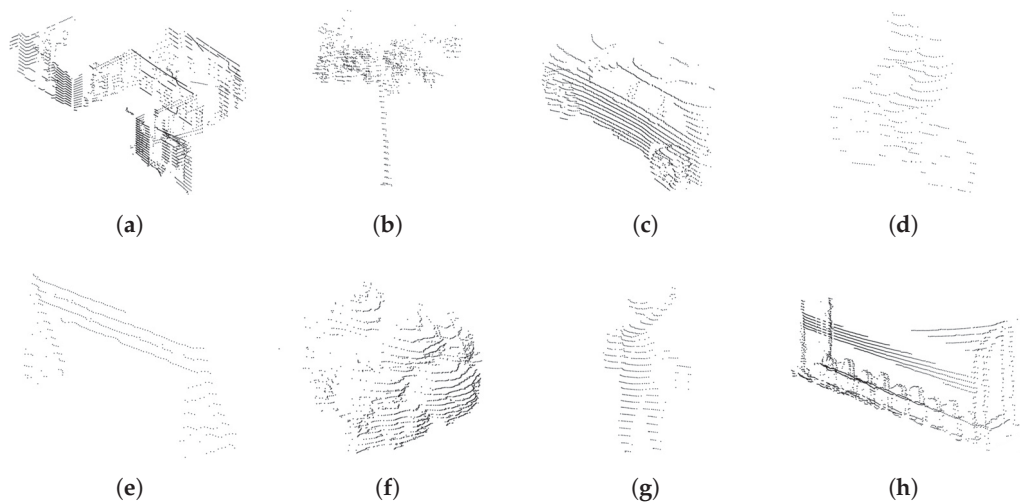


**Figure 3.** Cont.





**Figure 3.** Examples of dataset.



**Figure 4.** Configuration of dataset classes: (a) building; (b) tree; (c) vehicle; (d) bicycle; (e) obstacle; (f) greenery; (g) person; (h) urban fixture.

#### 2.4. Performance Evaluation

In the PFE, the image size, number of pillars, and number of points per pillar are hyperparameters that significantly influence both the performance and complexity of the model. The image size determines the resolution and dimensions of the encoded image. A larger image size results in higher-resolution images; however, it also increases the computational demand of the CNN. Similarly, increasing the number of pillars and points per pillar allows more detailed information to be captured from the point cloud. However, this also increases the computational load of the PFE owing to the increased number of computations required to process a larger number of data. Thus, finding a balanced combination of these hyperparameters is crucial for achieving both high performance and efficient computation in pillar-based object classification models.

Figure 5 illustrates the architectures of the networks utilized for performance evaluation, all of which were based on the DSCNN. Table 1 presents the accuracies corresponding to various hyperparameters and network structures, with the overall accuracy serving as an evaluation metric for object classification. We trained 300 epochs with a batch size of 16, and the results of the best performing epochs are presented in Table 1. The training was performed with a learning rate of 0.001 and a weight decay rate of 0.0001. First, it was observed that Networks 1 and 5 exhibited lower performance than Networks 2, 3, and 4. Among Networks 2, 3, and 4, Network 4 stands out for its balanced performance, demonstrating comparable accuracy to Networks 2 and 3, while boasting lower computational complexity and fewer parameters. Additionally, the image size of  $128 \times 128$  exhibited outstanding performance, whereas the performance was comparable when the number of pillars was 1024 and 512. The best performance was achieved when 16 points were inside

each pillar. Consequently, the proposed object classification model adopted an image size of  $128 \times 128$ , 512 pillars, and 16 points within each pillar, with the classification network structured according to Network 4.

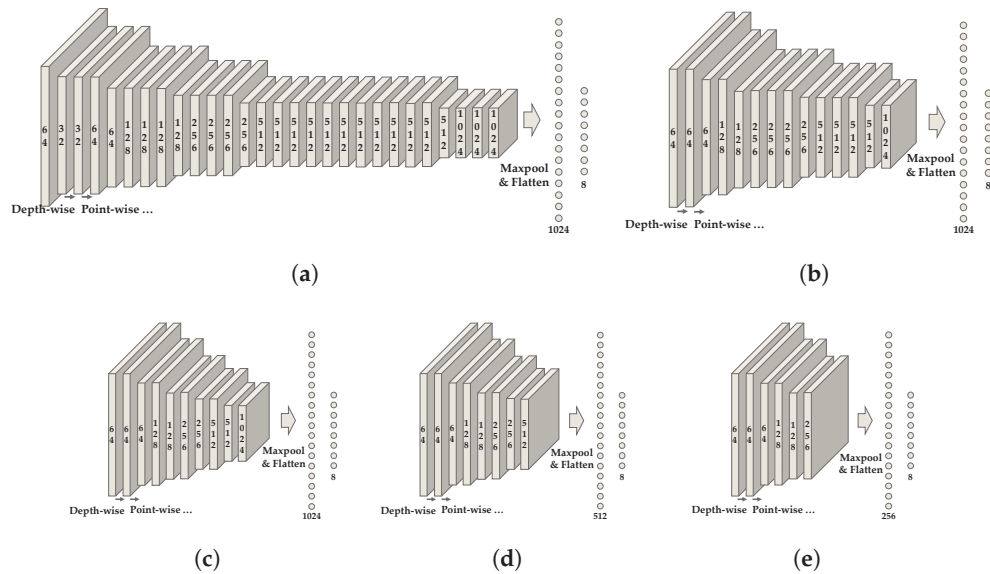
**Table 1.** Accuracy of model by various hyperparameters and networks.

Image Size	Hyperparameter		Network				
	Number of Pillars	Number of Points per Pillar	1	2	3	4	5
$256 \times 256$	2048	32	91.8%	93.5%	93.5%	92.9%	89.7%
		16	92.9%	92.9%	92.4%	92.4%	88.0%
		8	92.9%	92.9%	92.9%	92.4%	89.7%
	1024	32	92.4%	92.4%	91.8%	92.9%	89.1%
		16	92.4%	92.9%	92.4%	92.9%	90.2%
		8	93.5%	93.5%	92.4%	92.4%	89.7%
	512	32	92.4%	92.9%	93.5%	93.5%	89.7%
		16	92.9%	94.0%	92.4%	93.5%	89.7%
		8	94.6%	92.9%	92.4%	91.8%	90.8%
$128 \times 128$	1024	32	90.8%	91.8%	92.4%	94.6%	90.8%
		16	90.8%	94.6%	92.9%	92.4%	91.3%
		8	90.8%	91.8%	91.8%	94.0%	91.8%
	512	32	90.8%	94.0%	91.8%	93.5%	92.4%
		16	91.3%	92.9%	93.5%	94.6%	92.9%
		8	90.8%	93.5%	91.8%	92.9%	92.4%
	256	32	91.8%	93.5%	92.9%	91.8%	90.8%
		16	90.8%	92.9%	92.4%	91.3%	90.8%
		8	90.8%	94.0%	94.0%	92.4%	91.3%
$64 \times 64$	512	32	89.7%	89.7%	92.9%	91.8%	91.3%
		16	88.6%	90.2%	93.5%	91.8%	91.8%
		8	88.6%	90.8%	92.9%	92.4%	92.4%
	256	32	88.6%	90.2%	93.5%	90.8%	90.8%
		16	87.5%	89.7%	93.5%	91.8%	91.8%
		8	87.5%	90.8%	93.5%	91.8%	90.8%
	128	32	88.6%	89.1%	89.7%	90.8%	89.7%
		16	89.1%	90.2%	91.3%	90.2%	90.8%
		8	86.4%	92.4%	91.3%	89.1%	91.8%

Table 2 presents a performance comparison between the proposed classification network and the other classification networks in this study. The proposed classification network achieved the highest classification accuracy of 94.6% based on the DSCNN, with a small number of multiply-accumulate (MAC) operations of 37 M and a small number of parameters of 50.8 K.

**Table 2.** Performance comparison with other networks.

Network	Accuracy	Number of MACs	Number of Parameters
LeNet	92.2%	787.6 M	2.0 M
ResNet18	83.8%	1.4 G	11.4 M
ResNet34	82.3%	2.0 G	21.5 M
MobileNet	85.7%	128.3 M	3.2 M
Ours	94.6%	37.0 M	50.8 K

**Figure 5.** Architecture of DSCNN-based object classification network: (a) Network 1; (b) Network 2; (c) Network 3; (d) Network 4; (e) Network 5.

### 3. Implementation of Acceleration System

The proposed acceleration system was implemented on an FPGA by integrating the PFE hardware accelerator intellectual property (IP) and classification network hardware accelerator IP with an MPU. Section 3.1 presents the bit width of each IP obtained through quantization experiments. Section 3.2 describes the structure and operations of the proposed acceleration system. Section 3.3 describes the structure and behavior of the PFE hardware accelerator, and Section 3.4 describes the FINN used to implement the object classification network hardware accelerator.

#### 3.1. Quantization

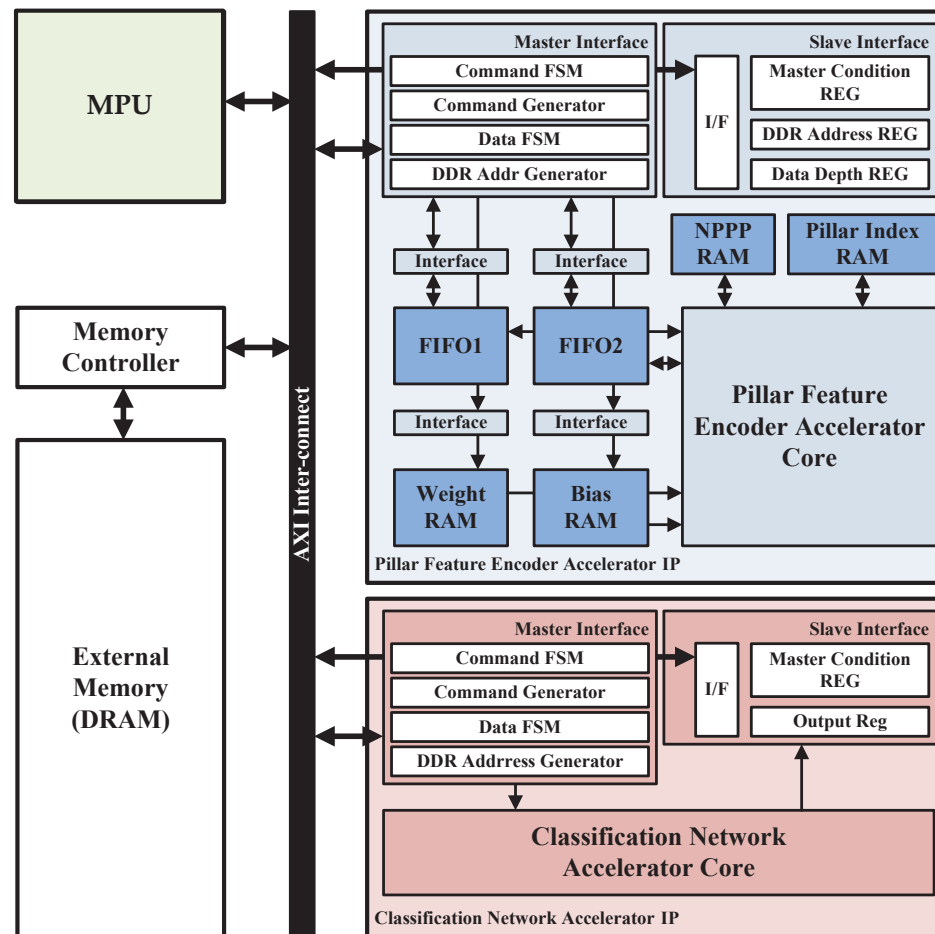
In this study, we quantized the proposed object classification model to implement the proposed acceleration system in the hardware. To minimize the performance reduction owing to errors caused by quantization, we performed quantization-aware training (QAT). The results of the experiments using different bit formats for the PFE and classification networks are listed in Table 3. The PFE is more accurate with 16 and 32 bits than with 8 bits, but there is no difference in accuracy between 16 and 32 bits. Therefore, we decided to use a 16 bit format for the PFE. The object classification network showed higher accuracy with 4 and 8 bits, but there was no difference in performance; therefore, we used 4 bits.

**Table 3.** Accuracy according to bit format of PFE and classification networks.

PFE	Classification Network		
	2	4	8
8	90.3%	91.7%	92.3%
16	89.7%	94.3%	94.3%
32	88.7%	92.7%	94.3%

### 3.2. System Architecture

Figure 6 shows the structure of the proposed acceleration system. To validate the system in an FPGA environment, the accelerators were integrated with an MPU via an advanced extensible interface (AXI) interconnect and implemented in an FPGA. Each accelerator IP is a slave to the MPU, and the MPU sends the start and end DRAM addresses that the IP will access, the number of words to be read from DRAM, AXI protocol parameters, and start and stop signals to the slave registers of the IP. The IP acts as a master to DRAM, generating read and write addresses through its master interface to fetch the required data from DRAM during computation and write back the results to DRAM.

**Figure 6.** Architecture of proposed acceleration system on FPGA.

The MPU first sets the slave register of the PFE accelerator IP, which starts by reading the point cloud from DRAM through the master interface to organize the point cloud into pillars. The PFE accelerator checks the coordinates of the point cloud and stores the point cloud in FIFO1 one after the other. To store the point cloud in DRAM pillar-by-pillar, it generates the DRAM addresses and stores them in FIFO2. When all the DRAM addresses

for the point cloud are generated and stored in FIFO2, the master interface references FIFO1 and FIFO2 to store the point cloud at an appropriate location in DRAM.

The PFE accelerator organizes the entire point cloud into pillars by appropriately storing it in DRAM, and then reads the point cloud back and stores it in FIFO1 and FIFO2. It also reads the weights and biases for performing PW convolution from DRAM and stores them in the weight memory (WM) and bias memory (BM). After adding the HCF to the point cloud, PW convolution is performed to expand the features, and a max-pool operation is performed. Thus, one pillar is encoded with 64 features, which is one pixel with 64 channels in a 2D image. The encoded pixel is stored in DRAM by considering its position.

After the PFE accelerator encodes all point clouds into 2D images, the MPU sets the slave register of the classification network accelerator IP and starts the accelerator to read images from DRAM through the master interface. After the classification network accelerator has read all images and has performed all computations, it writes the classification results to the slave register. The MPU completes the system operation by accessing the slave register of the classification network accelerator IP and reading the classification results.

### 3.3. PFE Hardware Architecture

Figure 7 shows a block diagram of the PFE accelerator. The proposed PFE accelerator comprises a grouping unit (GU), an HCF unit (HU), a PFN unit (PU), and eight memories. The memories are FIFO1, FIFO2, number of points per pillar (NPPP) buffer (NB), pillar index buffer (PB), HCF buffer1 (HB1), HB2, WM, and BM. HB1 and HB2 are utilized in a ping-pong scheme by the HU and PU within a pipelined structure. As the HU operates, it writes results to HB1, while the PU concurrently accesses data from HB2. Subsequently, the HU switches to writing computation results to HB2, while the PU utilizes the data stored in HB1 during this phase.

The point clouds are stored in FIFO1 in order, and the grid locator in the GU checks the coordinates of the points to obtain the coordinates of the pillars containing the points. For example, if the width of a grid cell is an integer value of  $2^5$ , the coordinates of the pillar containing a point can be determined by dividing the point's coordinates by  $2^5$ . This division operation is replaced by a 5 bit right shift operation. The coordinates of the pillars are used as the read address of the NB to determine the NPPP. If the NPPP is 0, a new pillar is created; thus, the pillar counter is incremented by one, and the value is written to the PB to be used as the index of the pillar. If the NPPP is less than 16, it is incremented by 1 and written back to the NB; otherwise, it remains unchanged. The address is generated by using the pillar index multiplied by 16 (the maximum NPPP) as an offset, and then adding the NPPP to this offset. The generated address is stored in FIFO2. With the point cloud in FIFO1 and the DRAM address in FIFO2, the point cloud is stored in the correct location in DRAM.

After all the point clouds are stored in DRAM in the pillars, they are read back into FIFO1 and FIFO2. In FIFO2, the DRAM write address is stored during the grouping operation, while the DRAM read data are stored at other times. The PFE accelerator then performs computation on all point clouds in a pillar-wise manner. The averaging unit (AU) of the HU adds the  $x$ ,  $y$ , and  $z$  coordinates of all points in the pillar and is divided by the NPPP to obtain the arithmetic mean coordinates of the points. The offset unit in the HU computes  $x_m$ ,  $y_m$ ,  $z_m$  by subtracting the center coordinates of the points obtained from the AU from the  $x$ ,  $y$ ,  $z$  coordinates of each point, and  $x_c$ ,  $y_c$  by subtracting the center coordinates of the pillar from the  $x$ ,  $y$  coordinates of each point. The  $x$ ,  $y$ ,  $z$ ,  $r$ ,  $x_m$ ,  $y_m$ ,  $z_m$ ,  $x_c$ ,  $y_c$  of the points with the HCF added are then stored in the HB.

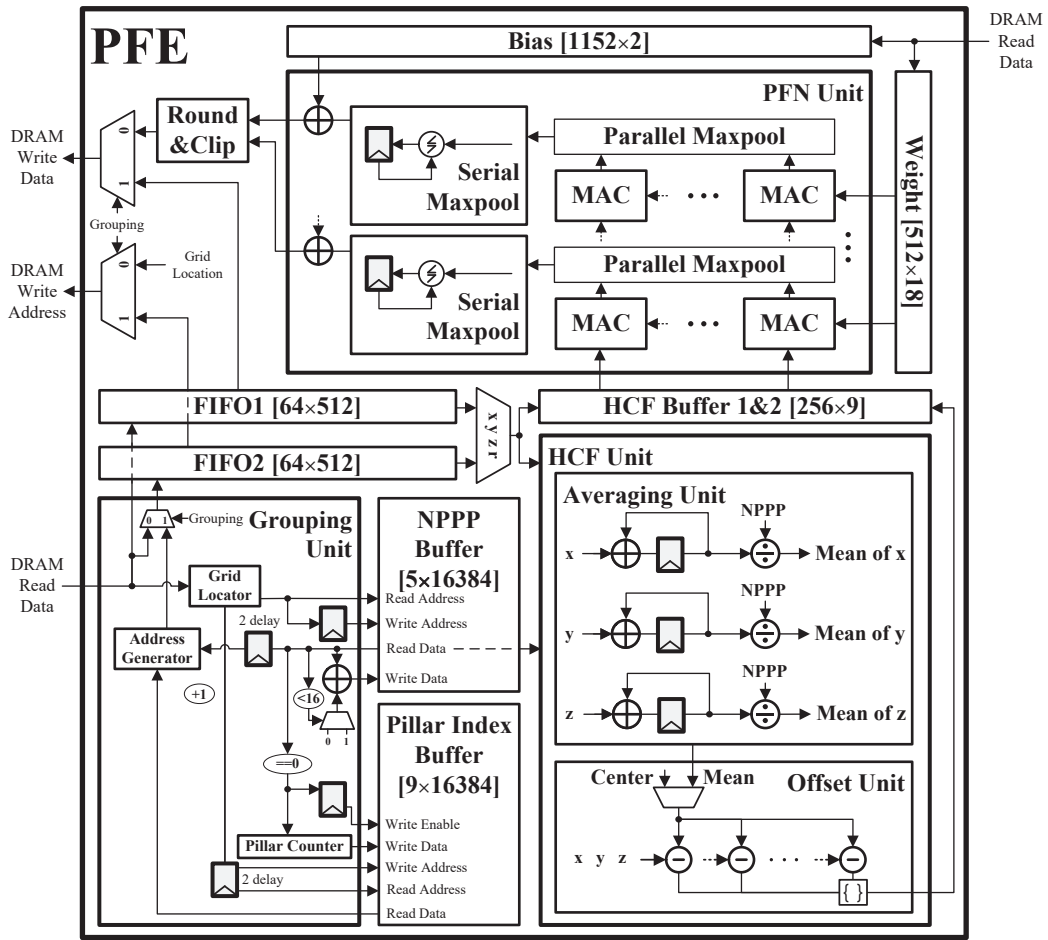


Figure 7. Block diagram of PFE accelerator.

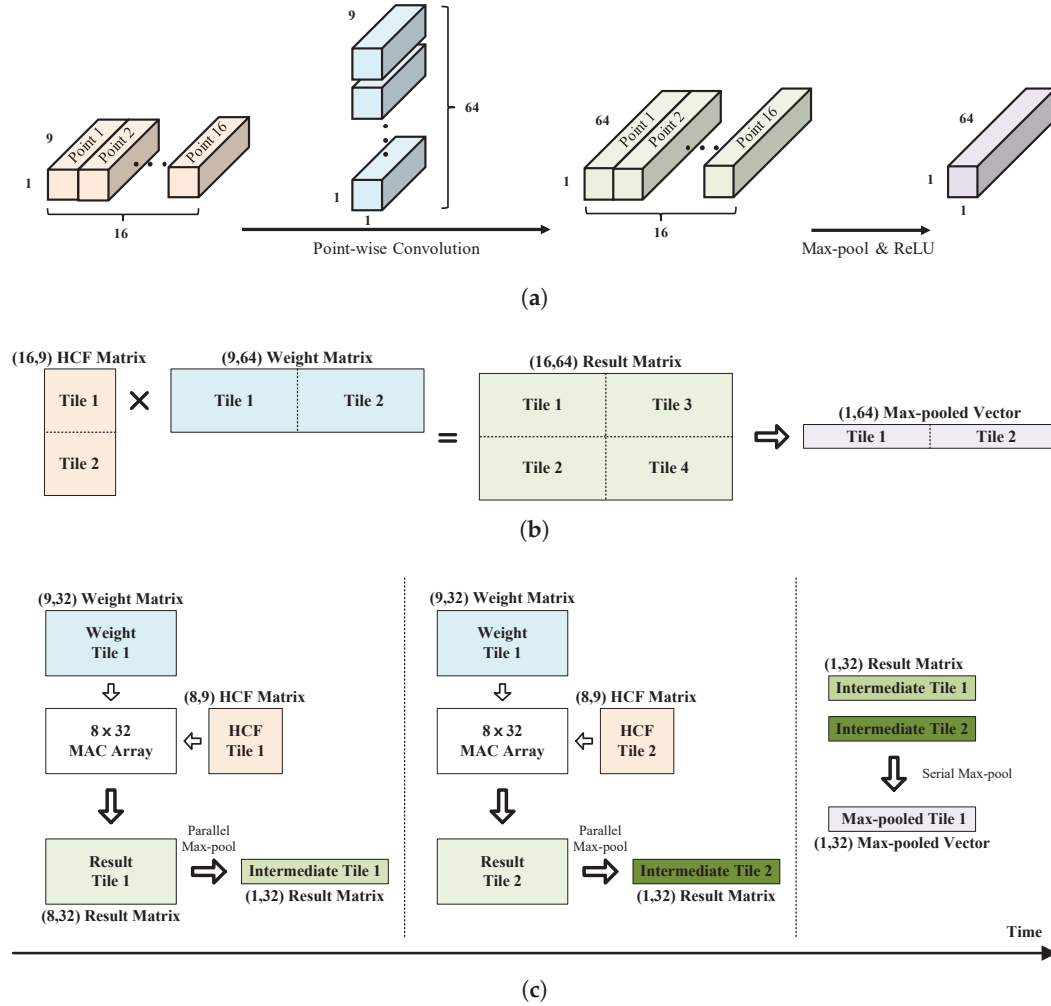
Figure 8a shows the tensor-level operation performed by the PU. The PU executes PW convolution, batch normalization, max-pool, and ReLU operations on the HCF. The weights and biases of the batch normalization can be combined with those of the PW convolution layer. In the PW convolution, only the weights are used, and the biases are added after a biased ReLU operation, which thresholds values at  $-bias$  instead of 0.

Figure 8b represents the operation in Figure 8a at the matrix level. The PU performs the multiplication of the HCF matrix of size  $(16, 9)$  and the weight matrix of size  $(9, 64)$  using an  $8 \times 32$  MAC array. To do this, the HCF matrix is divided into two matrices of size  $(8, 9)$ , and the weight matrix is divided into two matrices of size  $(9, 64)$ , performing the entire matrix multiplication in four parts. For instance, the matrix multiplication of HCF tile 2 and weight tile 1 produces result tile 2, and these tiles are max-pooled to create max-pooled tile 1.

Figure 8c illustrates the hardware-level operation of the PU in chronological order. The PU reads HCF tile 1 and weight tile 1 from the HB and WM, respectively. The PU reads HCF tile 1 column by column and weight tile 1 row by row over 9 cycles, feeding them into the MAC array. The MAC array processes the HCF and weight inputs over 9 cycles and outputs result tile 1 of size  $(8, 32)$ . Result tile 1 is then processed by the parallel max-pool unit to perform max-pool operations, producing intermediate tile 1 of size  $(1, 32)$ . Because the parallel max-pool unit immediately performs max-pool operations on the MAC array output, it saves cycles and buffer memory space by not storing and reloading result tile 1. Intermediate tile 1 is compared to the  $-bias$  value in the serial max-pool unit, and the larger value is stored in the register. By performing biased ReLU operations in the serial max-pool unit, the ReLU operation could be integrated into the max-pool operation. After creating intermediate tile 1, the PU reads HCF tile 2 and weight tile 1 to produce intermediate tile 1.



2. Intermediate tile 2 undergoes max-pool operations with intermediate tile 1 stored in the serial max-pool unit's register, resulting in max-pooled tile 1. Max-pooled tile 1 is half a pixel when the bias is added and rounded and clipped. This half pixel is stored in the correct location in DRAM based on its coordinates in the image. To create the DRAM write address, the  $x$  coordinate of the pixel multiplied by the image's width is used as an offset, and the  $y$  coordinate is added to this offset. When the PU has created both max-pooled tiles 1 and 2, there is one complete pixel in DRAM.



**Figure 8.** Operation of the PU: (a) tensor-level operation; (b) matrix-level operation; (c) hardware-level operation.

### 3.4. Classification Network Accelerator Using FINN

Xilinx offers various tools for implementing deep learning networks. Vitis AI is a platform used to implement and optimize artificial intelligence and machine learning applications on FPGAs and adaptive system-on-chip platforms. Vitis AI supports various artificial intelligence and machine learning frameworks and libraries, and generates a deep learning processing unit (DPU) IP. FINN is an open-source framework for FPGA-based quantized deep learning network inference. FINN uses an open neural network exchange model as the input, optimizes the network, and generates a hardware accelerator.

In this study, we used FINN to implement a hardware version of an object classification network. Whereas the DPU serves as a general-purpose accelerator for deep learning operations, FINN optimizes the hardware architecture and data flow specifically for the deep learning network. Consequently, FINN offers lower latency, enables real-time response speeds, and can achieve high throughput by optimizing the pipeline [29]. Moreover, net-

works with low complexity, such as the proposed classification network, can significantly reduce hardware resource usage. In addition, FINN supports 4 bit integers, providing an advantage over DPUs. Leveraging the benefits of FINN, ref. [19] implemented a backbone layer of PointPillars using FINN.

#### 4. Implementation Results

The proposed acceleration system was implemented on a Xilinx Zynq Ultrascale+ ZCU104 [30]. Table 4 summarizes the hardware resource usage and operating frequency of the proposed acceleration system. The proposed acceleration system utilized 75,548 configurable logic block (CLB) look-up tables (LUTs), 56,931 CLB registers, 374 digital signal processors (DSPs), and 65 block RAMs. The operating frequency was 187.5 MHz, which enables high-speed operation, and the response time of the proposed object classification system was 6.41 ms.

**Table 4.** Hardware resources of proposed object classification system.

Unit	CLB LUTs	CLB Registers	DSPs	Block RAM	Frequency (MHz)
PFE	35,567	28,513	302	10.5	
Classification Network	34,788	21,625	72	54.5	
AXI Interconnect	5193	6793	0	0	
Total	75,548	56,931	374	65	187.5

Table 5 compares the response times of the firmware-based PFE implementation on an ARM Cortex A53 (baseline) operating at 1.2 GHz and the proposed PFE hardware accelerator (proposed) in terms of the execution time for each operation. The grouping operation, responsible for converting a point cloud into a set of pillar units, experienced a remarkable acceleration of 23.67 times, reducing from 0.71 ms to 0.03 ms. Notably, the HCF and PFN operations achieved a significant acceleration of approximately 42.6 times, decreasing from 8.52 ms to 0.20 ms. This notable acceleration was enabled by leveraging the parallel computation capability of the MAC array in the PFE accelerator and the pipelined structure of the HCF and PFN operations. The overall execution time for PFE operations was reduced from 9.23 ms to 0.23 ms, demonstrating a 40 times acceleration effect compared with the firmware-based implementation.

**Table 5.** PFE response time comparison with baseline.

Operation	Baseline (Firmware)	Proposed (Hardware)
Grouping	0.71 ms	0.03 ms
HCF	0.20 ms	
PFN	8.32 ms	0.20 ms
Total	9.23 ms	0.23 ms

Table 6 presents a comparison between the proposed object classification system and pillar-based methods implemented in previous studies. In [19–21], a heterogeneous system between the MPU and FPGA was constructed, whereas our system represents an end-to-end hardware implementation of all operations on an FPGA. In [19], PFE computation was performed on an MPU, and deep learning network computation was executed using an accelerator implemented through FINN. Ref. [20] computed the PFE on an MPU and executed a deep learning network on an accelerator implemented using a register-transfer level (RTL) design. Ref. [21] utilized both an MPU and DPU for PFE computation, and a DPU for deep learning network computation. We computed the PFE with an accelerator

implemented through an RTL design and deep learning network with an accelerator implemented through FINN.

Our method achieved the fastest response time of 6.41 ms, because it was the only one that accelerated the PFE, and the proposed object classification model was designed to have low computational complexity through various experiments. Moreover, these experiments resulted in an object classification model with fewer parameters. Our method used more DSPs than that in [19], but significantly fewer LUTs, registers, and memory. Ref. [19] used the same FPGA platform targeting edge environments as ours, the ZCU104, achieving low power consumption. However, it still consumed more power than ours, which implemented an efficient deep learning network. Compared with the method in [20], our method used fewer resources across the board, with particularly notable differences in DSPs and memory usage. Ref. [20] also consumed low power by implementing an efficient deep learning network. Additionally, since ref. [20] designed the deep learning network accelerator at the RTL level, it consumed even less power than ours, which was designed based on HLS. In addition, our method used more LUTs but fewer registers and DSPs, and substantially fewer BRAM and URAM than that in [21]. Furthermore, our method consumed significantly less power than that in [21]. Ref. [21] used the U280 FPGA platform targeting server environments, which had much higher power consumption and is not suitable for edge environments.

**Table 6.** Comparison with other pillar-based implementations.

	[19]	[20]	[21]	Ours
Platform	ZCU104	ZC706	U280	ZCU104
Computation Device	MPU & FPGA (FINN)	MPU & FPGA (RTL)	MPU & FPGA (DPU)	FPGA (RTL & FINN)
Execution Time (ms)	377.1	43.26	64.1	6.41
CLB LUTs	189,074	128,721	48,006	75,548
CLB Registers	150,187	111,118	85,801	56,931
DSPs	88	883	525	374
Block RAM	159	382.5	131.5	65
Ultra RAM	-	-	64	1
Frequency (MHz)	150	150	300	187.5
Power (W)	6.5	3.6	73.8	4.4

## 5. Conclusions

In this study, we propose a deep learning object classification model based on a pillar-based method for real-time responses of autonomous mobile robots. The object classification network of the deep learning model is based on the DSCNN, which has a low complexity and few parameters, making it suitable for edge device environments. The object classification model was trained using a dataset that considered the operating environment of autonomous mobile robots. To minimize the performance reduction due to quantization, we quantized and tested the model using QAT, achieving a classification accuracy of 94.3%. Because the computation of the PFE in pillar-based object classification models was time-consuming, we proposed a hardware accelerator for the PFE. The PFE accelerator was implemented through RTL design and achieved a response time of 0.23 ms, which was 40 times faster than the firmware program. The accelerator for the object classification network of the proposed deep learning model was implemented through the FINN framework and achieved an execution time of 6.18 ms. We also proposed an object classification system that integrates the proposed PFE accelerator and object classification network accelerator through an AXI interconnect. The proposed object classification system

was implemented and verified on an FPGA and achieved a real-time response time of 6.41 ms.

The proposed PFE accelerator can be combined with other deep learning network accelerators to accelerate more diverse deep learning models. However, in this study, we only performed acceleration on the object classification models. In future work, we plan to implement a system that supports acceleration for more diverse and challenging deep learning models.

**Author Contributions:** C.P. designed and implemented the proposed acceleration system, performed the experiment and evaluation, and wrote the paper. S.L. evaluated the proposed acceleration system and revised this manuscript. Y.J. conceived of and led the research, analyzed the experimental results, and wrote the paper. All authors read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by Korean government (MSIT) (No. 2022-0-00960), and the CAD tools were supported by IDEC.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Dataset available on request from the authors.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Varlamov, O. “Brains” for Robots: Application of the Mivar Expert Systems for Implementation of Autonomous Intelligent Robots. *Big Data Res.* **2021**, *25*, 100241. [CrossRef]
2. Liu, Y.; Li, Z.; Liu, H.; Kan, Z. Skill Transfer Learning for Autonomous Robots and Human–robot Cooperation: A Survey. *Robot. Auton. Syst.* **2020**, *128*, 103515. [CrossRef]
3. Yoshioka, M.; Suganuma, N.; Yoneda, K.; Aldibaja, M. Real-time Object Classification for Autonomous Vehicle using LIDAR. In Proceedings of the 2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, Japan, 24–26 November 2017; pp. 210–211.
4. Gao, H.; Cheng, B.; Wang, J.; Li, K.; Zhao, J.; Li, D. Object Classification using CNN-based Fusion of Vision and LIDAR in Autonomous Vehicle Environment. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4224–4231. [CrossRef]
5. Zhou, Y.; Liu, L.; Zhao, H.; López-Benítez, M.; Yu, L.; Yue, Y. Towards Deep Radar Perception for Autonomous Driving: Datasets, Methods, and Challenges. *Sensors* **2022**, *22*, 4208. [CrossRef] [PubMed]
6. Su, H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 945–953.
7. Hoang, L.; Lee, S.H.; Lee, E.J.; Kwon, K.R. GSV-NET: A Multi-modal Deep Learning Network for 3D Point Cloud Classification. *Appl. Sci.* **2022**, *12*, 483. [CrossRef]
8. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 652–660.
9. Zhou, Y.; Tuzel, O. Voxnet: End-to-end Learning for Point Cloud based 3D Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4490–4499.
10. Bhanushali, D.; Relyea, R.; Manghi, K.; Vashist, A.; Hochgraf, C.; Ganguly, A.; Kwasinski, A.; Kuhl, M.E.; Ptucha, R. LiDAR-camera Fusion for 3D Object Detection. *Electron. Imaging* **2020**, *32*, 1–9. [CrossRef]
11. Yan, Y.; Mao, Y.; Li, B. Second: Sparsely Embedded Convolutional Detection. *Sensors* **2018**, *18*, 3337. [CrossRef] [PubMed]
12. Lang, A.H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; Beijbom, O. Pointpillars: Fast encoders for Object Detection from Point Clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 12697–12705.
13. Lis, K.; Kryjak, T. PointPillars Backbone Type Selection for Fast and Accurate LiDAR Object Detection. In Proceedings of the International Conference on Computer Vision and Graphics, Warsaw, Poland, 19–21 September 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 99–119.
14. Shu, X.; Zhang, L. Research on PointPillars Algorithm based on Feature-Enhanced Backbone Network. *Electronics* **2024**, *13*, 1233. [CrossRef]
15. Wang, Y.; Han, X.; Wei, X.; Luo, J. Instance Segmentation Frustum–PointPillars: A Lightweight Fusion Algorithm for Camera–LiDAR Perception in Autonomous Driving. *Mathematics* **2024**, *12*, 153. [CrossRef]
16. Agashe, P.; Lavanya, R. Object Detection using PointPillars with Modified DarkNet53 as Backbone. In Proceedings of the 2023 IEEE 20th India Council International Conference (INDICON), Hyderabad, India, 14–17 December 2023; pp. 114–119.

17. Choi, Y.; Kim, B.; Kim, S.W. Performance Analysis of PointPillars on CPU and GPU Platforms. In Proceedings of the 2021 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Jeju, Republic of Korea, 27–30 June 2021; pp. 1–4.
18. Silva, A.; Fernandes, D.; Névoa, R.; Monteiro, J.; Novais, P.; Girão, P.; Afonso, T.; Melo-Pinto, P. Resource-constrained onboard Inference of 3D Object Detection and Localisation in Point Clouds Targeting Self-driving Applications. *Sensors* **2021**, *21*, 7933. [CrossRef] [PubMed]
19. Stanisiz, J.; Lis, K.; Gorgon, M. Implementation of the Pointpillars Network for 3D Object Detection in Reprogrammable Heterogeneous Devices using FINN. *J. Signal Process. Syst.* **2022**, *94*, 659–674. [CrossRef]
20. Li, Y.; Zhang, Y.; Lai, R. TinyPillarNet: Tiny Pillar-based Network for 3D Point Cloud Object Detection at Edge. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *34*, 1772–1785. [CrossRef]
21. Latotzke, C.; Kloecker, A.; Schoening, S.; Kemper, F.; Slimi, M.; Eckstein, L.; Gemmeke, T. FPGA-based Acceleration of Lidar Point Cloud Processing and Detection on the Edge. In Proceedings of the 2023 IEEE Intelligent Vehicles Symposium (IV), Anchorage, AK, USA, 4–7 June 2023; pp. 1–8.
22. Tan, M.; Le, Q. Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
23. Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
24. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
25. Lu, G.; Zhang, W.; Wang, Z. Optimizing Depthwise Separable Convolution Operations on GPUs. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 70–87. [CrossRef]
26. Kaiser, L.; Gomez, A.N.; Chollet, F. Depthwise Separable Convolutions for Neural Machine Translation. *arXiv* **2017**, arXiv:1706.03059.
27. Ouster. Ouster OS1 Lidar Sensor. Available online: <https://ouster.com/products/hardware/os1-lidar-sensor> (accessed on 22 May 2024).
28. Caesar, H.; Bankiti, V.; Lang, A.H.; Vora, S.; Liong, V.E.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; Beijbom, O. nuscenes: A multimodal Dataset for Autonomous Driving. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 11621–11631.
29. Xilinx Inc. Vitis™ AI Documentation Frequently Asked Questions. Available online: <https://xilinx.github.io/Vitis-AI/3.0/html/docs/reference/faq.html#what-is-the-difference-between-the-vitis-ai-integrated-development-environment-and-the-finn-workflow> (accessed on 22 May 2024).
30. AMD. UltraSclae+ ZCU104. Available online: <https://www.xilinx.com/products/boards-and-kits/zcu104.html> (accessed on 22 May 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# A Trusted Execution Environment RISC-V System-on-Chip Compatible with Transport Layer Security 1.3

Binh Kieu-Do-Nguyen <sup>1,2</sup>, Khai-Duy Nguyen <sup>1</sup>, Tuan-Kiet Dang <sup>1</sup>, Nguyen The Binh <sup>1,2</sup>,  
Cuong Pham-Quoc <sup>2,\*</sup>, Ngoc-Thinh Tran <sup>2</sup>, Cong-Kha Pham <sup>1</sup> and Trong-Thuc Hoang <sup>1</sup>

<sup>1</sup> Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Tokyo 182-8585, Japan; binh@vlsilab.ee.uec.ac.jp (B.K.-D.-N.); duy@vlsilab.ee.uec.ac.jp (K.-D.N.); tuankiet@vlsilab.ee.uec.ac.jp (T.-K.D.); binh.nguyen288@hcmut.edu.vn (N.T.B.); phamck@uec.ac.jp (C.-K.P.); hoangtt@uec.ac.jp (T.-T.H.)

<sup>2</sup> Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet St., Dist. 10, Ho Chi Minh City 740050, Vietnam; tnthinh@hcmut.edu.vn

\* Correspondence: cuongpham@hcmut.edu.vn

**Abstract:** The Trusted Execution Environment (TEE) is designed to establish a safe environment that prevents the execution of unauthenticated programs. The nature of TEE is a continuous verification process with hashing, signing, and verifying. Such a process is called the Chain-of-Trust, derived from the Root-of-Trust (RoT). Typically, the RoT is pre-programmed, hard-coded, or embedded in hardware, which is locally produced and checked before booting. The TEE employs various cryptographic processes throughout the boot process to verify the authenticity of the bootloader. It also validates other sensitive data and applications, such as software connected to the operating system. TEE is a self-contained environment and should not serve as the RoT or handle secure boot operations. Therefore, the issue of implementing hardware for RoT has become a challenge that requires further investigation and advancement. The main objective of this proposal is to introduce a secured RISC-V-based System-on-Chip (SoC) architecture capable of securely booting a TEE using a versatile boot program while maintaining complete isolation from the TEE processors. The suggested design has many cryptographic accelerators essential for the secure boot procedure. Furthermore, a separate 32-bit MicroController Unit (MCU) is concealed from the TEE side. This MCU manages sensitive information, such as the root key, and critical operations like the Zero Stage BootLoader (ZSBL) and key generation program. Once the RoT is integrated into the isolated sub-system, it becomes completely unavailable from the TEE side, even after booting, using any method. Besides providing a secured boot flow, the system is integrated with essential crypto-cores supporting Transport Layer Security (TLS) 1.3. The chip is finally fabricated using the Complementary Metal–Oxide–Semiconductor (CMOS) 180 nm process.

**Keywords:** Trusted Execution Environment; RISC-V; secure-boot; FPGA; VLSI

## 1. Introduction

In cyber-security research, the isolation between the programs that run on an Operating System (OS) is called the Trusted Execution Environment (TEE) [1]. This is an advanced feature for a secured OS. During a typical boot procedure, the bootloader initializes the OS; subsequently, the actual OS image is loaded into the system memory. Ultimately, the machine successfully starts up and enters the operating system that has been loaded. The entire process depends on the assumption that all components of the boot process function correctly. However, trust in the realm of computers requires more than just that. Consequently, a trust mechanism is essential, an assurance mechanism that allows you to validate the integrity of each stage of the boot sequence [2], ensuring that no compromises have occurred. Subsequently, it is possible to construct a Trusted Execution Environment (TEE) using the trust above. The design idea of TEE is to segregate trusted and untrusted



codes through a divide-and-conquer technique [3]. Typically, isolation is achieved through privilege separation, which effectively establishes a barrier between different programs. Modern TEE models are currently equipped with software- and hardware-based barrier enforcers at various architectural levels. The ultimate objective of TEE is to exclusively permit the execution of authenticated codes while preventing unauthenticated code from running on the trusted side and acquiring any privileges.

For TEE papers, the most popular architectures are Intel Software Guard eXtensions (SGX) [4–7], ARM TrustZone [8,9], and AMD Secure Encrypted Virtualization (SEV) [10]. Over the years, many improvements have been made. For example, Intel SGX has Haven [11], Graphene [12], and Scone [13]; ARM TrustZone has Komodo [14], OP-TEE [15], and Sanctuary [16]; AMD SEV has SEV-ES [17] and SEV-SNP [18]. Those three TEEs (i.e., Intel, ARM, and AMD) are licensed, and any IP modification is strictly prohibited. Recently, with the trending of open-source hardware of RISC-V, many attempts at open-source TEE models were also proposed. Several examples can be listed, including Hex-Five Multi-Zone [19], Sanctum [20], TIMBER-V [21], CUstomizable and Resilient Enclaves (CURE) [22], and Keystone [23]. Nowadays, almost all smartphones possess a TEE-like characteristic, and numerous organizations, from software to hardware, promote their devices with pre-installed security attributes.

The key mechanism in TEE is the Chain-of-Trust (CoT). It is a link of many cryptographic operations like hash, sign, and verify. At each layer of the OS stack, the upper layer with lower privilege must verify the signature of the lower layer with higher privilege before doing anything [24]. The Root-of-Trust (RoT) is the initial authentication of the CoT system, which serves as its foundation. To ensure security, the Root-of-Trust (RoT) should not be accessible from the Rich Execution Environment (REE) or the Trusted Execution Environment (TEE) processors once the system has been booted. There are many ways to create an RoT. For example, it could be an asymmetrical key pair, a random value, or a pre-signed certificate. In most cryptosystems, the conventional way of implementing RoT is a hard-coded root key in Read-Only Memory (ROM). Creating RoT is extremely important in a cryptosystem; the integrity of the entire TEE depends on the secure boot procedure with such an RoT. The TEE is a self-contained environment that cannot function as the RoT. It is advisable to execute a secure boot process using an RoT and hardware primitives as a standard practice. In most TEE models, the trusted firmware is usually assumed to be properly loaded into the stack before boot because that is actually the job of the hardware, not the software. Therefore, to carry out this initial task, traditionally, TEE models rely on the hardware itself or other Intellectual Properties (IPs); for example, Platform Security Processor (PSP) [25] for AMD SEV, CryptoCell [26] for ARM TrustZone, and Active Management Technology (AMT) [27] for Intel SGX. Regarding RISC-V, since the RISC-V itself is an open-source hardware, many RoT modifications have been proposed directly into the hardware system. Many RISC-V RoT examples are the Rambus CryptoManager [28] and the OpenTitan [29]. The RoT is the designated location for keeping and overseeing the root key and certificates of the device. As per the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) [30], a hardware platform that can securely boot with an RoT must possess the following capabilities: the ability to generate cryptographic keys, wrap and bind keys, seal and unseal keys, incorporate a True Random Number Generator (TRNG), include an integrity measurement feature, and perform key attestation.

The secure boot procedure involving the RoT is one of the challenges faced by the TEE. The remote attestation of the TEE must be performed using the RoT, established through the secure boot procedure. In addition, the demand for Trusted Execution Environments is continuously growing in the Internet of Things (IoT) era. Several new attack vectors have recently been identified that could potentially compromise the secure boot process or expose the root key [31], rendering the entire TEE system susceptible to attacks. It requires a TEE system that can be updated even after manufacturing to deal with the new threats. In this situation, the combination of RISC-V's open-source Instruction Set Architecture

(ISA) and an open-source TEE is a perfect complement. The RISC-V architecture offers a wide range of customizations [32] to create a custom TEE, effectively addressing persistent issues. With the introduction of RISC-V, we can reexamine the hardware architecture to enhance the TEE and reduce the RoT to the silicon level. This can be achieved while still ensuring a safe boot program that is versatile and adjustable. Consequently, in principle, tampering with the silicon RoT entails disrupting the chip manufacturing procedure. Furthermore, leveraging the growing open-source mentality and incorporating cutting-edge security measures, the CoT's capabilities may be further improved. In essence, this proposal's primary contribution lies in introducing a method to isolate the RoT from the TEE processors while allowing the ability to modify the boot sequence.

In summary, the issue of secure boot with an RoT in a TEE system remains a great challenge that needs further investigation. This work is an improvement from our previous work [33] that was published in 2022. The contributions and improvement of this study can be classified into three primary categories, as outlined below:

1. The TEE-HW framework. A high-security computer system must be developed, and an open-source TEE-HW framework must be developed to interface with the Keystone open-source TEE software framework [23]. The suggested TEE-HW framework must address the following requirements. It must be safe, simple to use, adaptable for different security needs, and, most significantly, simple to upgrade with a new defense mechanism. Many architectural features are left optional and can be easily changed by modifying the Makefile system's parameters. The RISC-V open-source community is welcome to reuse the TEE-HW framework's source codes [34]. Future security developers will benefit from such open-source TEE hardware.
2. TEE-HW with cryptographic accelerators. A unique system designed specifically for TEE was created based on the suggested TEE-HW framework. True Random Number Generator (TRNG), Advanced Encryption Standard Galois/Counter Mode (AES\_GCM), and Secure Hash Algorithm 3 (SHA-3) are among the several introduced crypto-cores. Besides the required crypto-cores, we also introduced several crypto-cores for the Transport Layer Security 1.3 (TLS 1.3), such as HMAC-SHA2, Digital Signature Algorithm (Ed-DSA or EC-DSA), Rivest–Shamir–Adleman (RSA), and Authenticated Encryption with Associated Data (AEAD). Furthermore, a hidden write-only memory, which is inaccessible to TEE processors, is another feature of the Ed25519 crypto-core. The keys produced by the Ed25519 module will be kept in this write-only memory. We investigated the performance of the suggested TEE hardware with crypto-cores using FPGA and VLSI implementation. We also looked into the TEE boot performance.
3. TEE-HW with isolated RoT. A heterogeneous architecture for RoT-based secure boot flow was suggested by combining an isolated MicroController Unit (MCU) and Linux-bootable TEE processors. While the concealed MCU handles key generation, secure boot, and root key storage, the TEE side typically runs the TEE software stack. After reset, the very first authentication is performed by the hidden MCU. Then, the other crypto-keys are created and stored in memory. Finally, the boot process is transferred to the TEE processors to boot into the Linux kernel. By this setup, all resources are available for the hidden MCU to use, but after boot, all the peripherals inside the hidden MCU are inaccessible by the TEE domain. The secure boot procedure and the remote administration tool (RoT) are no longer within the TEE domain. This makes the secure boot procedure flexible and capable of withstanding potential future threats. The proposed architecture was developed and tested on both FPGA and VLSI on a 180 nm process.

The remaining parts of this paper are structured as follows. Section 2 presents background knowledge, including the Trusted Execution Environment and Keystone. Section 3 presents the crypto-accelerators used in the proposed system. Section 4 reveals the proposed TEE System-on-Chip. Section 5 presents the proposed secured boot flow. Section 6

summarizes the experimental results. The final portion, Section 7, concludes this study and discusses future work.

## 2. Background Knowledge

### 2.1. Trusted Execution Environment

Generally, remote computing systems are not capable of resolving security issues. For instance, consumers cannot manipulate the tangible elements on their computers. Information can be transferred, and harmful software can be executed remotely on your computer, either from another computer within the same system or from the internet. Hardware manufacturers are striving to provide a reliable mechanism to address these concerns. Therefore, a TEE is introduced. Historically, TEEs have offered three assurances: (1) integrity: ensuring that the code and data remain unaltered and cannot be manipulated, such as by executing unauthorized code within a partition; (2) confidentiality: preventing attackers from gaining knowledge about the runtime content of the application, including secret keys and code control flow; (3) attestation: providing evidence to a remote party that the environment is secure and has not been tampered with.

Trusted Execution Environment aims to provide a state of separation between applications, hence establishing a boundary between different programs. The barrier is commonly implemented using a privilege separation method and enforced by hardware primitives like memory isolation. To separate low-privilege codes (user's apps) from high-privilege codes (OS's services) or vice versa, the previous iteration of TEEs used an essential method of encrypting the codes that need protection and implementing some authentication between the parties involved. Contemporary TEEs are significantly more intricate than those designed for the trusting mechanism. Nevertheless, the central concept remains unchanged. An enclave, which is a standard configuration for a program operating in a TEE, requires a True Random Number Generator (TRNG) to generate keys and various cryptographic functions for tasks such as key creation, hashing, signature, verification, and cipher encryption/decryption. To provide the entire protection of an enclave, current TEEs commonly incorporate a Trusted Firmware (TF) at M-mode. This firmware offers exclusive services that do not depend on the operating system's services. The primary services provided by TF include dedicated memory allocation, cache flushing during enclave context switches, and encryption of messages entering and exiting an enclave. In addition, TF serves as the Trusted Computing Base (TCB) for establishing the trusted domain and preserving the integrity of enclaves' boundaries. Due to the crucial significance of TF, it is imperative to verify the integrity of TF through a secure boot procedure. The authentication of TF is commonly referred to as the RoT in a TEE system.

Every Trusted Execution Environment implementation requires an underlying hardware mechanism as a barrier enforcer. The Physical Memory Protection (PMP) function is the barrier enforcer for the RISC-V architecture. The RISC-V architecture introduces a range of privileged levels, from Machine-mode (M) and extending to User-mode (U). Every authentication is signed by a lower-privileged level and then validated by a higher level. Therefore, the CoT is formed. The initiation of CoT involves the initial verification process during a reset, referred to as the RoT.

RoT can encompass many possibilities, from a random value to a signature signed using an asymmetric key. The RoT must remain unavailable from the TEE processors to provide security once the system has been booted. The keys the RoT created are used to sign, verify, and broadcast to various components of the TEE security architecture. This process ensures that the TEE environment has a Root-of-Trust. The primary objective of TEE is to thwart the execution of unauthorized code on the trustworthy side and prevent it from acquiring any privileges. The TEE employs a combination of cryptographic processes during the boot process to verify the authenticity of the bootloaders. It then validates critical data and applications, such as OS-related programs. Consequently, when booting, we can ensure that only apps that are considered trustworthy will possess valid authentications. Both untrusted codes and infected trusted codes will no longer have valid signatures.

## 2.2. Keystone

Keystone is a promising open-source project designed explicitly for RISC-V systems. A Keystone enclave can verify its identity, verify the authenticity of software, and ensure the security and confidentiality of remote execution. D. Lee et al. state that it can provide the CoT with secure boot [23], remote attestation, and secure key provisioning. In Keystone, memory isolation at M-mode is achieved through Physical Memory Protection (PMP) and page table isolation. Keystone utilizes RISC-V's PMP feature to implement isolation and prohibit other programs from accessing the enclave memory. Keystone has minimal hardware requirements, including a common RISC-V core, a means to store device keys, and a secure bootloader. Thanks to RISC-V's privilege model and physical memory protection standard, the software can easily manage the remaining tasks. Keystone comprises a collection of software components, rules, and tools that enable the development of TEEs for standard platforms based on the RISC-V architecture. Like SGX-style enclaves, Keystone separates each application into a separate partition during execution. While SGX requires the host to handle all resource management tasks, Keystone enables each enclave to execute user- and supervisor-level code. The system employs a straightforward and adaptable reference monitor, the Security Monitor, which operates below the host operating system to impose security assurances for TEEs. This reference monitor is designed based on the principles of Komodo and Sanctum.

Keystone has multiple memory protection techniques based on specific requirements. As an illustration, the basic setup protects at the software level, cache partitioning can guard against attacks linked to the cache, and on-chip enclave and bus encryption can defend at the hardware level [23]. The Keystone SDK offers essential capabilities necessary for constructing enclave apps. The SDK comprises four components. (1) Host libraries offer an interface for managing enclave applications. (2) Enclave Application libraries provide both essential enclave tools (such as EXIT) and some fundamental libc-style functions (such as malloc and string headers). (3) Edge libraries provide features for managing edge calls to enclave applications and hosts. Edge calls refer to function calls that traverse the boundary between an enclave and its host. (4) Runtime refers to the code at the system level that executes within the enclave. The userland enclave manages the enclave entry point, basic system calls, and all call-related data transfers.

## 3. The Crypto-Accelerators

### 3.1. True Random Number Generator (TRNG)

Figure 1 shows the block diagram of the proposed True Random Number Generator (TRNG), a part of our proposed system [35]. It is responsible for generating truly random numbers. The generated bits from the TRNG core are stored in the accumulator, which accumulates up to 192 bits. Once the accumulator is full, the bits are shifted into the shift register. When the shift register is filled, the Ready signals are active, and the data are sent out. The final output is stored in a 192-bit register and can be read through Tilelink Peripheral Bus (PBus). The Arbitrator controls the operation of the TRNG core. It determines when and how the generated random numbers are transferred out. When receiving the request from the PBus, the Arbitrator initiates the sampling process. The number of samples is set through 32-bit registers. Per each cycle, the Arbitrator activates the TRNG core to generate random bits and increases the number of samples until it exceeds the set-up samples. The TRNG core passes the non-IID standard test from the NIST. The TRNG will generate the seed for the key generation step of our proposed booting flow in Section 5. In addition, the generated seeds are also necessary for the cryptographic core, as we propose in the following sub-section.

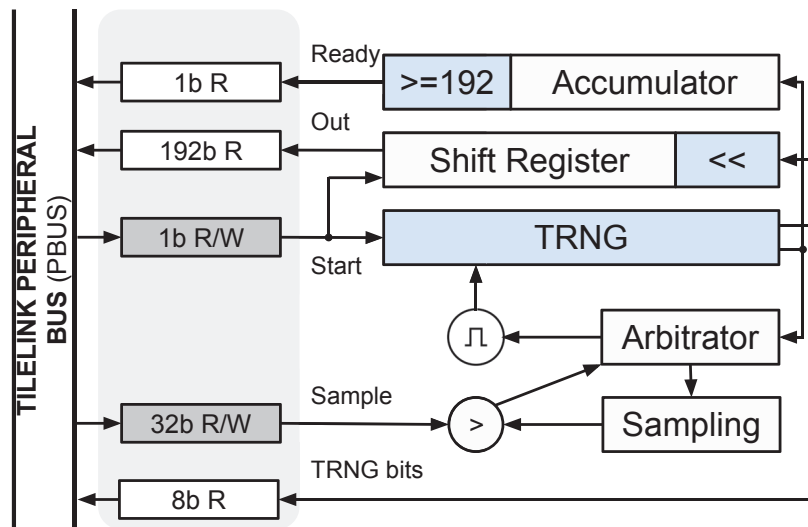


Figure 1. The proposed TRNG architecture.

### 3.2. SHA3-512

The first security accelerator utilized in the TEE boot phase is the SHA3 unit. This accelerator comprises a padding module and a Keccak-1600 calculator [36]. The padding module extracts 64-bit data from the register router and passes it via a 576-bit buffer using a shifter. Once the buffer is filled, the accelerator executes a circular computation. The Constant Counter (see Figure 2) monitors the number of rounds and the consistent non-linearity of the iota phase of the Keccak algorithm. The initial round is computed using the first 64-bit data processed by the padding module. A 1600-bit status register stores the state of each round. After the Padding Module (see Figure 2) processes the final data, the Round Calculation (see Figure 2) executes the last rounds in the status registers. Subsequently, the first 512-bit word can be utilized as the hash result. The results are then read by the processor through PBus. Figure 2 illustrates the proposed SHA3 architecture. The SHA3 unit hashes the private keys used by the Elliptic Curve/Edward Curve Cryptography module, which is used in our proposed boot process. In addition, SHA3 is the selected hash function for TLS 1.3.

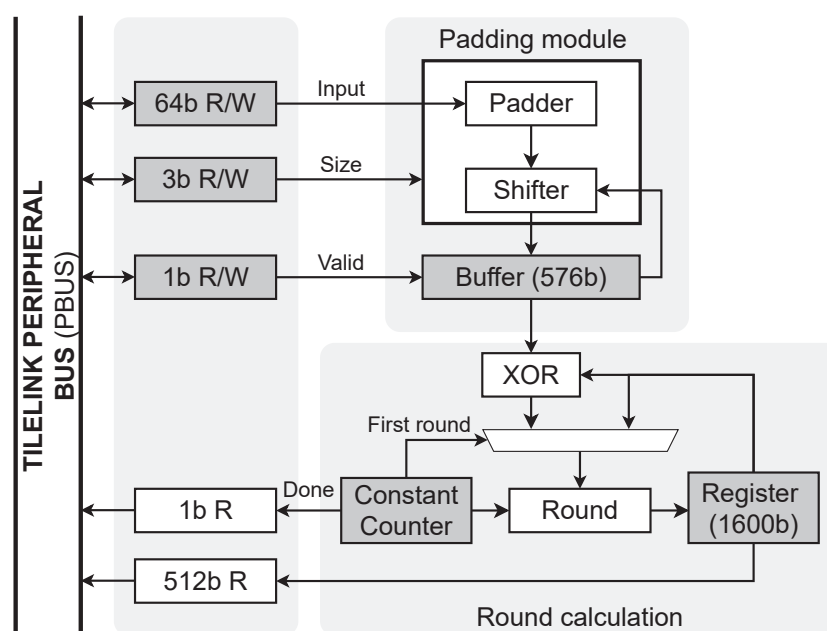


Figure 2. The proposed SHA3 architecture.



### 3.3. Advanced Encryption Standard (AES)—Galois/Counter Mode (GCM)

The second security accelerator implemented is associated with the AES cipher [37]. The AES-GCM accelerator performs encryption or decryption on blocks of either 128 bits or 256 bits, depending on the chosen configuration. It is possible to modify the bit length dynamically. Every datapath executes the Substitution Box (SBox) or Inverted S-Box (InvSBox), shifts rows, mixes columns, and an extra round key. The round key is computed externally for the 128-bit and 256-bit key variants. The AES calculation is executed by a state machine that activates the datapath and indicates data transfer to the output register. Figure 3 illustrates the proposed AES-GCM architecture. AES-GCM is a mandatory encryption of TLS 1.3.

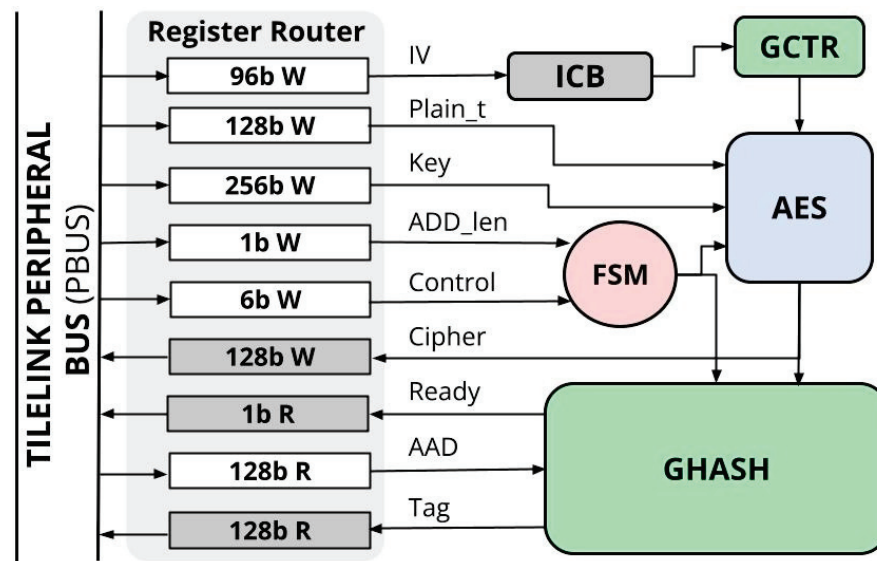


Figure 3. The proposed AES-GCM architecture.

### 3.4. Hash-Based Message Authentication Code (HMAC) with Secure Hash Algorithm 2 (SHA2)

The third security accelerator is HMAC-SHA2 [38]. The integrated HMAC-SHA2 accelerator performs a two-round process to calculate the Hash. In the first round, the inner key is derived from combining the inner pad, a constant string, and the input secret key during the initial step to produce the inner key. Subsequently, the inner key is associated with the input message. This combination is then hashed to generate the digest. In the second round, the digest is combined with the outer key before being hashed (see Figure 4). HMAC-SHA2 is an acronym for Hash-based Message Authentication Code and uses SHA2 as its underlying cryptographic hash function. The implemented SHA2 core can perform four standards, which are SHA2-224, SHA2-256, SHA2-384, and SHA2-512. The input data are expanded during the hashing process, and the input message is divided into chunks. The chunk size for SHA2-224/256 is 512 bits, while for SHA2-384/512 it is 1024 bits. The received data are compressed in the next stage. Lastly, the compressed data are used to compute the new hash value. HMAC-SHA2 is the compulsory authentication scheme of TLS 1.3.

### 3.5. Elliptic Curve (EC) and Edward Curve (Ed) Digital Signature Algorithm (DSA)

The four security accelerators are the Elliptic Curve and Edward Curve Digital Signature Algorithm (ECDSA/EdDSA). ECDSA and EdDSA generate public and private keys, which will subsequently be employed in signing and verification procedures. ECDSA and EdDSA play pivotal roles in our proposed secured boot scheme. The data are inputted into the memory-mapped Random Access Memory (RAM). The SHA3 hashed private key is read by the processor from the SHA-3 module and written into the ECDSA/EdDSA's RAM. The Processing Elements (PEs) fetch the private key from memory and then multiply it with



the base point of the selected curve. The Finite State Machine (FSM) is pre-programmed in Read-Only Memory (ROM) as the microcode that controls the operations of the PEs. The PEs execute the decoded instructions from ROM using its built-in decoder. The execution units in each Processing Element include adders, subtractors, and multipliers driven by the decoded instructions. Each calculation module has a basic calculator to round the value to the prime number depending on the selected mode, which is essential for the algorithm to execute the operations based on the Edward Curve of the Elliptic Curve. The outcomes of every operation are temporarily stored in RAM, which acts as a register file in this case. The final results are finally written back to RAM through a local bus. The RAM is also used to store constants defined by the selected curve. To enhance the parallelism, the embedded microcode in ROM includes vector-based instructions that effectively control multiple execution units to perform parallel tasks defined in ECDSA and EdDSA specifications. Finally, The Keystone system uses the produced signature to sign the bootloader program, discussed in Section 5. Figure 5 illustrates the proposed ECDSA/EdDSA combinational architecture. ECDSA and EdDSA are the compulsory key exchange schemes required by TLS 1.3.

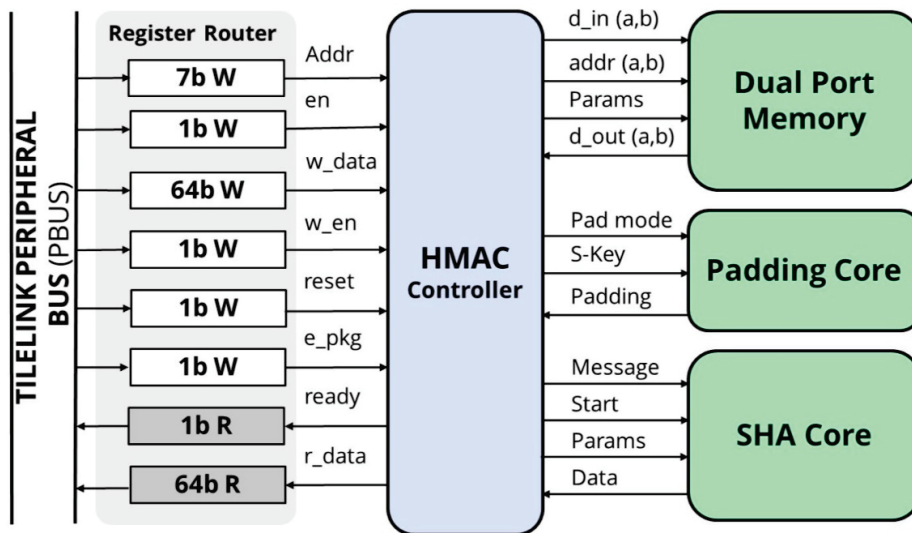


Figure 4. The proposed multi-functional HMAC-SHA2 architecture.

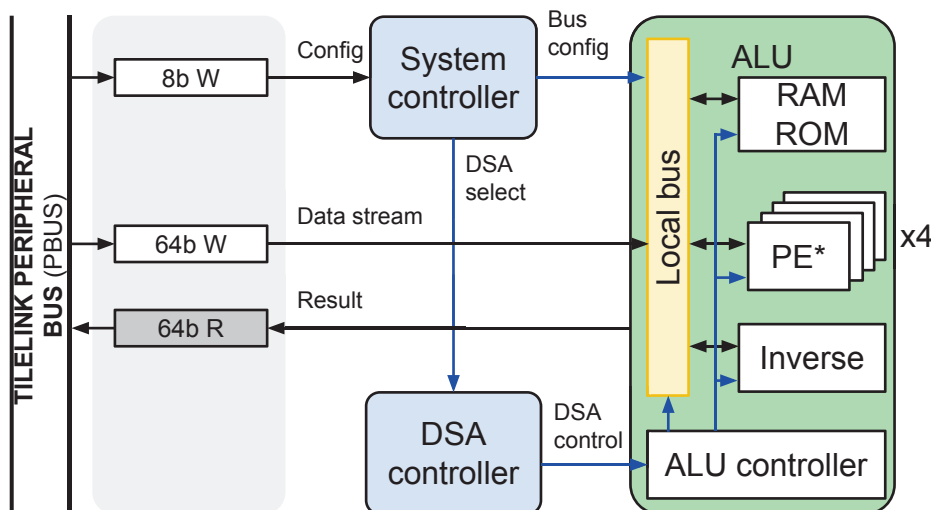


Figure 5. The proposed ECDSA/EdDSA combinational architecture. PE\*: Processing Element.

### 3.6. Rivest–Shamir–Adleman (RSA)

The fifth security accelerator is the Rivest–Shamir–Adleman (RSA) module. The proposed architecture is revealed in Figure 6. It uses 1024-bit values to calculate the power model function. To reduce the complexity of the calculation circuit, the 1024-bit register is split into multiple registers with smaller sizes. At the initial stage, the Pre-computation module (getNumBits) (see Figure 6) generates the necessary initial values. The output of these modules is stored in registers. Then, the FSM controls the accumulator ( $\pm$ ) and comparator ( $<$ ) to perform the encrypt/decrypt operations that follow the RSA's specification. The data in the output are 64-bit. Figure 6 illustrates the RSA block diagram. RSA is the compulsory encrypt/decrypt function required by TLS 1.3.

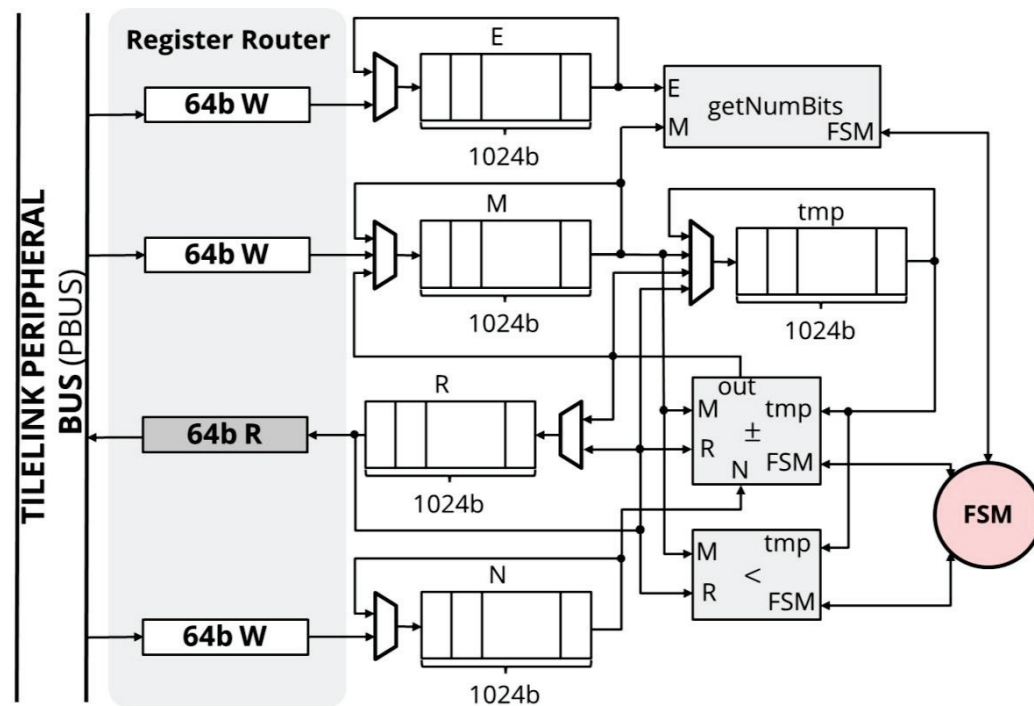


Figure 6. The proposed RSA-1024 architecture.

### 3.7. Authenticated Encryption with Associated Data (AEAD)

The sixth security accelerator is the Authenticated Encryption with Associated Data (AEAD) module. As per the specification, AEAD utilizes a 256-bit key, 96-bit nonce, plaintext of any length, and Additional Authenticated Data (AAD) of any length [39]. Figure 7 depicts the block diagram of the AEAD module, which is connected to the system through the Tilelink Peripheral bus. In the proposed architecture, the key utilized in Poly1305 is derived from ChaCha20. ChaCha20 and Poly1305 algorithms are integrated into a single peripheral to reduce the overhead from data exchange through the shared system bus. Two FSMs are designed to manipulate the operations of ChaCha20 and Poly1305 cores as well as data exchange between these two cores. The ChaCha20 core produces the cipher text based on the input plaintext, key, and nonce (see Figure 7). Meanwhile, the Poly1305 generates the authentication tag (MAC) based on half of the 512-bit generated cipher text of the Chacha20 core and the input Associate Data (AD). The final results, including 512-bit cipher text generated from Chacha20 and 128-bit MAC generated from Poly1305, are then read by the processor through PBus (see Figure 7). AEAD is a recommended encryption scheme by TLS 1.3. It provides better performance than AES-GCM.

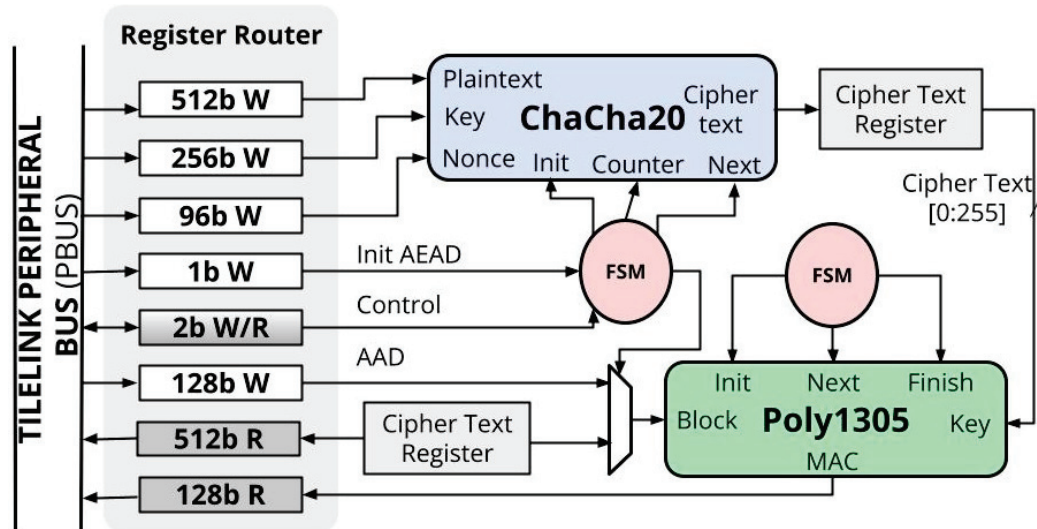


Figure 7. The proposed AEAD architecture.

#### 4. TEE System-on-Chip

##### 4.1. The Isolated Sub-System

The isolated 32-bit architecture and the standard 64-bit TEE processors are seen side by side in Figure 8. A RISC-V-based RV32IMC IBex [40] core is present in the isolated sub-system. The IBex was selected due to its compact 32-bit core with tamper awareness. The isolated design uses a TileLink bus called IBus as its main bus and a boot ROM. This sub-system also has a separate Core Local Interrupt (CLINT) and Platform-Level Interrupt Controller (PLIC). For scheduling purposes, internal core-level interrupts are handled by the isolated CLINT. The isolated core can receive commands from the external TEE processors via the PLIC. The PLIC's interrupts are then handled by the IBex core using programs that are kept in its boot ROM.

The TRNG employed in this section is derived from earlier research [35]. As Figure 8 illustrates, PBus connects the TRNG core with the system. The NIST standard demands that the TRNG be in the same environment with the derived keys [41]. Therefore, the TRNG module has two separate PBus connections, one for the data and one for the commands. As a result, the IBex core and the TRNG module have a direct connection, minimizing side-channel attack risk. The TRNG will self-reset after completing its transaction. Thus, TRNG values can be seen as non-independence, non-IID data since the commands originating from the two channels are not regarded equally. The TRNG is proven to pass the non-IID NIST test [35]. Therefore, the two-channel strategy of implementing TRNG did not affect the random quality.

##### 4.2. The Isolated TEE System

Figure 8 displays the suggested design. The architecture shown in Figure 8 also includes a variety of properties, like the number and type of cores, the ISA configuration, and the sizes of the L1 and L2 caches, which are easily reconfigurable based on specific requirements. In addition, each crypto-core, the PCIe connection, and the entire isolated sub-system can be added or removed based on requirements. By default, each core in the dual-core system contains a 16 KB instruction cache and a 16 KB data cache. The Rocket core is ranked first, followed by the BOOM core. The default configuration is RV64GC ISA, 512 KB L2 cache, including the isolated domain and all the peripherals in Figure 8, and excluding the PCIe controller.

The 32-bit isolated MCU is the special feature of this heterogeneous architecture. Upon reset, the isolated MCU boots first; it performs initial authentication and then uses root keys with random integers from TRNG to produce keys. Subsequently, the TEE processors will be activated by the standard TEE boot sequence [23]. The Isolated Bus (IBus) is the

primary bus of the isolated sub-system. It is a master-only TileLink [42] connection with the System Bus (SBus). As a result, all peripherals under the IBus are obscured from the TEE processors. In contrast, the hidden MCU can access every submodule in the SoC. Therefore, the isolated domain is the ideal location for root keys.

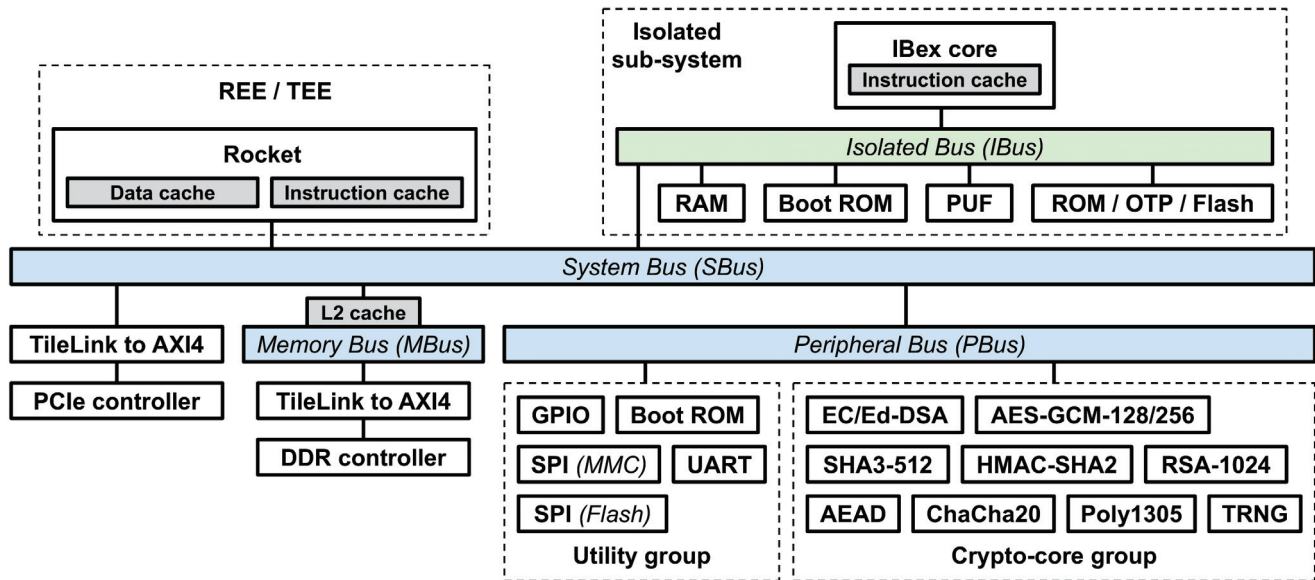


Figure 8. The proposed TEE-HW architecture with isolated sub-system.

The L2 cache is integrated with a coherence cache manager. The Peripheral Bus (PBus), as seen in Figure 8, contains a Universal Asynchronous Receiver/Transmitter (UART), several GPIOs, a boot ROM, an SPI for SD card, and an SPI for flashes. For the crypto-core group, several popular cryptographic accelerators are added, including SHA3, DSA, and TRNG. The TEE hardware is also integrated with a DDR controller for booting and running the Operating System (OS) and the software. Finally, to control the external DDR memory, a TileLink-to-AXI4 bridge is used to connect the inside Memory Bus (MBus) with the AXI4 protocol [43] to the outside DDR IP controller. The integrated devices on PBus, like GPIO, can be exported to the outside for the VLSI implementation. Consequently, the manufactured chip can connect to an FPGA platform and leverage its DDR IP.

## 5. Secured Boot Flow

The Keystone framework [23] is the base for the suggested boot process and key generation. Depending on the Keystone's definitions, two things must be trusted to create a secure boot process. (1) Manufacturer of hardware: Chip makers need to be responsible for their products. As a result, we may rely on the silicon manufacturing process to produce trusted hardware, like RoT. (2) Software providers must also adhere to security requirements to safeguard their products. However, there are two reasons why the infrastructure and data transmission environment cannot be used in these two cases. (1) Infrastructure: Many elements could reduce infrastructure security. For instance, security flaws in virtualization software enable hackers to launch direct attacks on other virtual machines from the compromised virtual machine. (2) Data transmission environments: Hackers can intercept data being transmitted via transmission lines, such as the Internet. From these perspectives, we offered a safe boot flow based on the isolated TEE system.

Initially, the RoT utilizes a secure chip with a trusted boot ROM to generate a hash of the software binary, creating  $H_S$ . The programs that require hashing are the sensitive programs, such as OS-related applications and those that require a specific privilege after booting. Every software possesses its uniquely produced  $H_S$ . Once the  $H_S$  is made, the  $SK_D$  and  $H_S$  are utilized to build the software pair keys, which consist of the secret key  $SK_S$  and the public key  $PK_S$ , using a Key Derivation Function (KDF), as depicted in the figure.

Once the  $SK_S$  and  $PK_S$  have been generated, the  $SK_D$  is employed to sign and validate the  $PK_S$  together with its  $H_S$ , creating a software certificate. The  $Cert_S$  can now be utilized to authenticate the software's integrity, as it is securely linked to  $H_S$  and endorsed by the device. We can generate an attestation report that traces back to the original manufacturer by utilizing a series of certificates. Once all the required certificates have been generated, the machine can boot into the operating system space. Due to the lack of trustworthiness of the boot image  $S$ , it is necessary to remove all sensitive data beforehand, such as the stack and  $SK_D$ .

Figure 9a shows the secure boot procedure and the Keystone boot flow performed by the heterogeneous architecture of TEE and hidden processors. The key idea is that the chip manufacturer will act as the root Certificate Authority (CA). Therefore, the root CA's public key  $P_M$  is widely recognized, and the root CA's certificate  $M_{Cert.}$  is a self-signed certificate. Each manufacturer can have multiple key pairs, but each key pair is unique for its manufacturer. Since the key pairs of  $S_M$  and  $P_M$  are generated offline, it is advisable to utilize high-bit RSA keys with an extended validity period of several years.

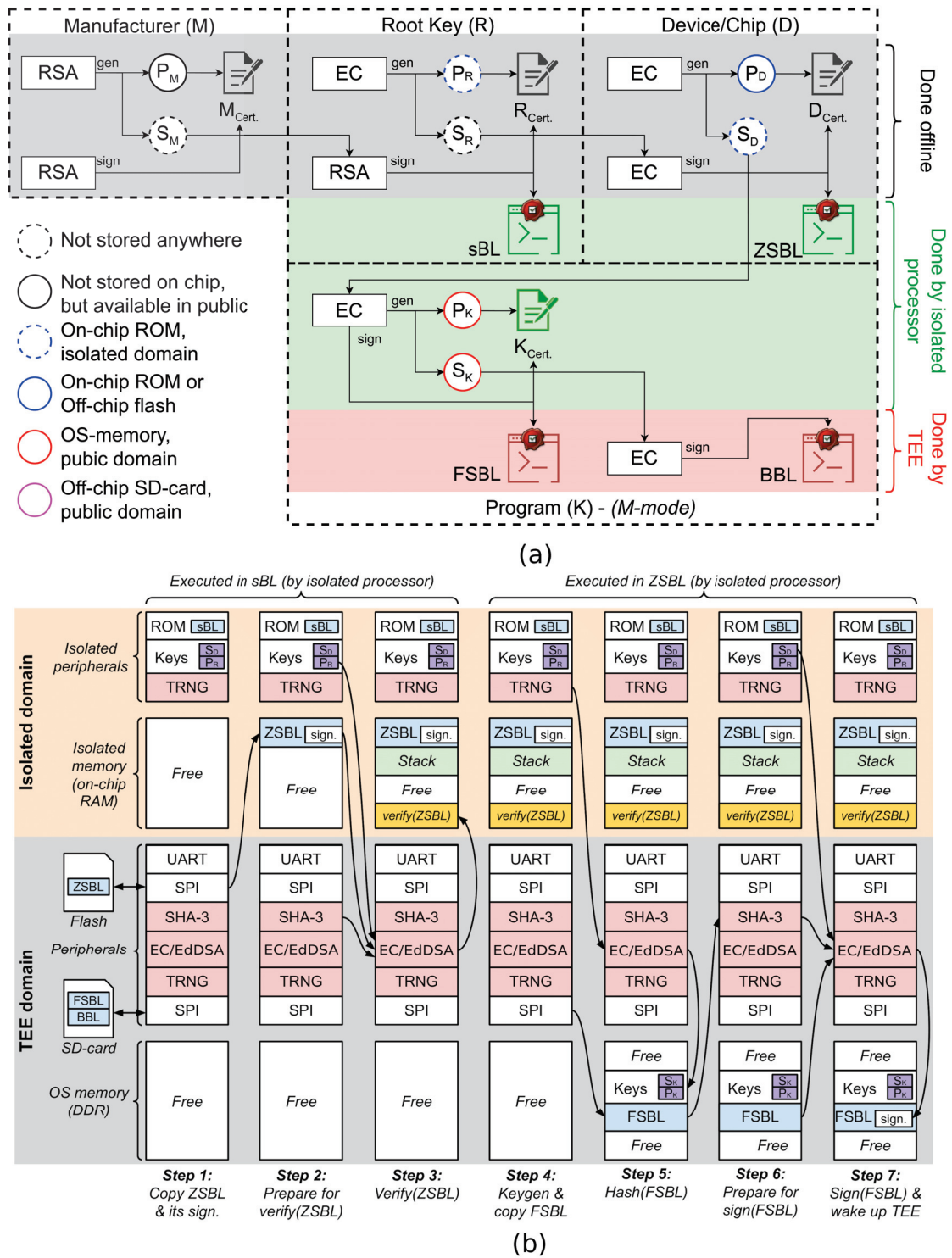
To improve the security level, the  $S_R$  and  $P_R$  root keys should be elliptic curve keys. These keys are generated by the manufacturer during the offline design phase. As mentioned earlier, the  $R_{Cert.}$  root certificate is a self-signed certificate using the secret key  $S_M$  of the manufacturer. Furthermore, the root secret key  $S_R$  is not saved anywhere, but the root public key  $P_R$  is stored in the boot ROM of the isolated domain. The purpose  $P_R$  is for the first authentication in the ZSBL. The isolated boot ROM also holds the very first boot loader called the secure BootLoader (sBL). As shown in Figure 9a, the very first task of the hidden processor is to verify the sBL content using the  $S_M$ . The hidden processor of IBex is also the core that runs the sBL content, which involves verifying and loading the ZSBL using the provided  $P_R$ .

The subsequent stage generates the EC key pair,  $S_D$  and  $P_D$ , for the device/chip. As depicted in Figure 9a, the manufacturer also produces them offline. The confidential key  $S_D$  of the device is securely stored in a separate Read-Only Memory (ROM). In contrast,  $P_D$ , the device public key, is stored in a publicly accessible location. The ZSBL is located in the same place as the  $P_D$ . Its job is to verify the signature signed by  $S_R$ , the root secret key. Since the isolated processor's initial action involves verifying and loading the ZSBL, this approach enables the manufacturer to securely update it, even if it is kept publicly, such as in an off-chip nonvolatile memory.

Once confirmed and loaded, ZSBL utilizes the True Random Number Generator as a seed for the EC-genkey algorithm to generate a pair of subsequent keys, namely  $S_K$  and  $P_K$ , referred to as Keystone keys. The keys are stored in a publicly accessible Random Access Memory (RAM) on the Trusted Execution Environment's side. Next, the secret key  $S_D$  of the device is utilized to sign the public key  $P_K$  of the Keystone, resulting in the creation of the Keystone certificate  $K_{Cert.}$ , as depicted in Figure 9a. Subsequently, the FSBL's content is transferred from the SD card to the main memory of the TEE domain, where the  $S_D$  is used to hash and sign its content. In the next step, both the FSBL and Keystone key pair are stored in the main memory and prepared for execution by the TEE processors. Ultimately, the isolated core activates the TEE processors to continue the conventional TEE boot flow.

Figure 8 depicts the connection between the IBus and the SBus, which uses the master-only TileLink protocol. As a result, all the resources below the SBus can be reached through the IBus, but not vice versa. Therefore, the TEE processors cannot access information within the hidden MCU due to this master-only bus. The Read-Only Memories (ROMs) and Random Access Memories (RAMs) located within the hidden MCU are well-suited for storing keys and carrying out the secured Bootloader (sBL) and Zero Stage Bootloader (ZSBL) operations. During the boot process, the isolated sub-system will be the first to boot to establish the RoT. Figure 9b illustrates the program execution process within a controlled environment. This application will execute once the system has been restored to the reset state.





**Figure 9.** (a) Key management in the secure boot procedure. The recommended storages for sBL, ZSBL, FSBL, and BBL are the isolated ROM, off-chip flash, SD card, and SD card/hard drive, respectively [33]. (b) Boot flow in the isolated environment (green part of a).

Upon reset, the TEE processors, a key component of our system, will enter a state of waiting for interrupt. With the support of the crypto-accelerators, the isolated sub-system retrieves the root/device keys from the ROM and combines them with TRNG to form a seed for Ed25519, resulting in the generation of the Keystone key pair. Once the  $S_K$  and  $P_K$  are created, the  $S_K$  will be stored in a write-only memory. This memory, designed explicitly



for the Ed25519 crypto-core, can only be accessed by crypto-cores for their operations. TEE processors and even the IBex core cannot read from this memory. Cryptographic cores can utilize this non-writable memory to compute the signature of a sequence of bits. In this scenario, the OS Bootloader (S) undergoes a process of hashing and is then internally signed using the previously saved private key of the Curve25519 function.

Because of the absolute separation between the two domains, it is impossible for any external program on the TEE processor to manipulate the operations of the isolated domain. This architectural feature is a strong defense against unauthorized access. The main possible threat from the TEE side is the interrupt exploitation requesting the authentication from the hidden sub-system. However, since the IBex's behavior solely depends on the program in its isolated boot ROM, we can reprogram the IBex core to cope with new threats. This flexibility in our system's design is another layer of defense against potential threats.

## 6. Experimental Results

### 6.1. Experimental Setups

The proposed Trusted Execution Environment hardware system supports Rocket and Ibex with Instruction Set Architecture (ISA) settings of RV32GC and RV32IMAC. The proposed TEE-HW SoC in Figure 8 was implemented in both FPGA (Xilinx Virtex-7 XC7VX485T) and VLSI (CMOS 180 nm technology). A single-core RV32GC Rocketchip was used as the TEE processor. It has 16 KB of instruction cache and 16 KB of data cache. For the hidden sub-system side, the IBex core with 4 KB of instruction cache was used. Compared to Figure 8, the PICe module is excluded, and the utility and crypto-core groups are included.

### 6.2. Resource and Power Consumption

The proposed system was implemented and tested on the Virtex-7 FPGA with the chip series of XC7VX485T; the results are given in Table 1. As seen from the table, the EdDSA/ECSA module occupied almost half of the design with 42.61% LUTs and 12.84% registers. For the whole design, 31.9% of the FPGA resources were spent. The isolated sub-system costs only 5.08% of LUTs, nearly half compared to the Rocket core. Table 1 provides the resource consumption for variable crypto-cores. The relation between the used Lookup Tables (LUTs) and Registers in Table 1 is illustrated in Figure 10.

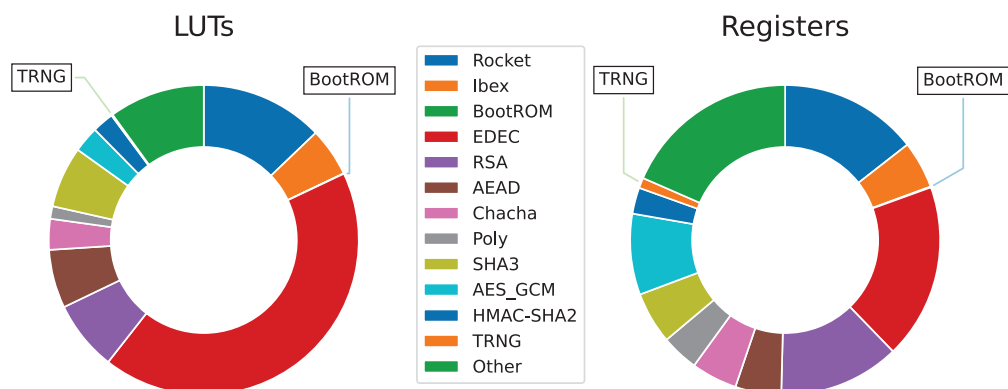


Figure 10. FPGA resource consumption.

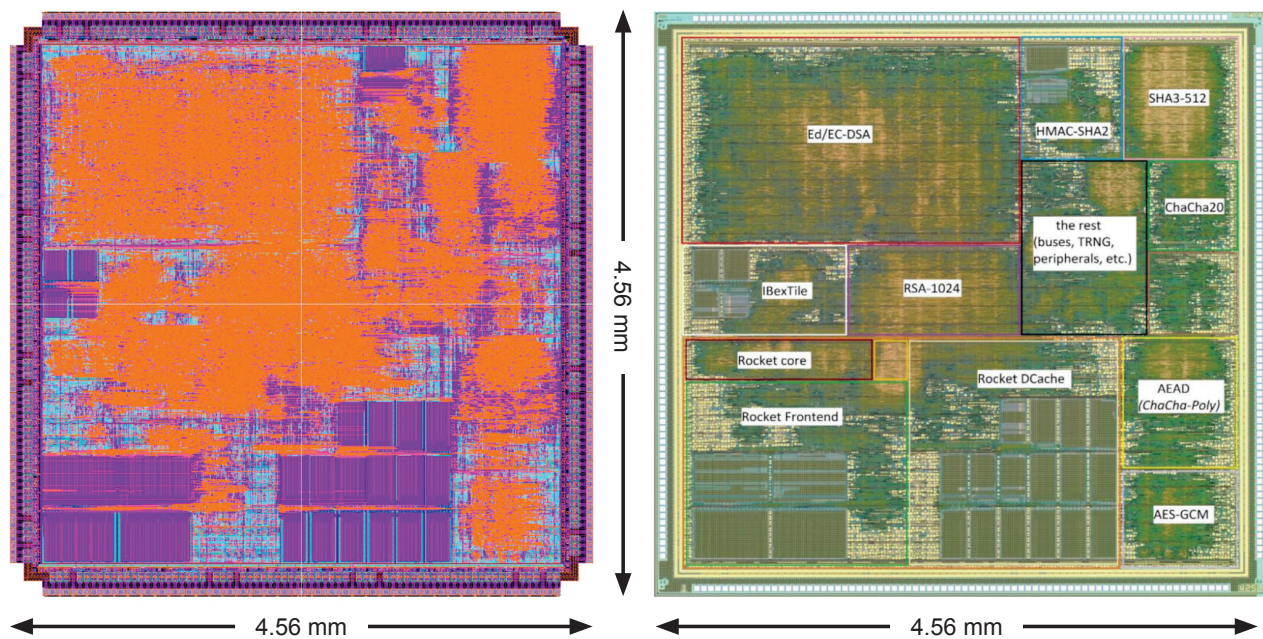
For ASIC implementation, the proposed SoC was synthesized in the conventional bulk CMOS 180 nm process. Figure 11 illustrates the layout and micro-graph of the fabricated chip. The results of the system with 100-MHz constraints are given in Table 2. The sizes of the chip and the submodules are revealed in Tables 2 and 3. According to the comparison table, nearly a third of the area was dedicated to the Rocket-tile at 34.59%, while the power consumption is just 13.82%. The Ed-DSA combined with the EC-DSA, the EDEC module, consumes the most power at 42.63% while costing 24.68% of the area. The whole hidden

sub-system, the IBex-tile, is quite small, with only 5.00% area and 2.24% power. The relation of sizes versus powers of different components in the TEE SoC is revealed in Figure 12 following the statistics in Table 2.

**Table 1.** Proposed 32-bit TEE SoC performances on Virtex-7 FPGA.

Instance	LUTs		Registers		BRAM	Size (KB)	DSP Blocks
Total system	97,040	100.00%	52,099	100.00%	28	619	16
Rocket	12,465	12.84%	7530	14.45%	12	544	2
core	3478	3.58%	1521	2.92%	0	0	2
dcache	2107	2.17%	3716	3.77%	2	16	0
icache	5982	6.16%	3716	7.13%	2	16	0
Ibex <sup>1</sup>	4929	5.08%	2575	4.94%	2	8	0
BootROM	38	0.04%	43	0.08%	4	32	0
EDEC	41,353	42.61%	9524	18.28%	8	2	0
RSA	7087	7.30%	6589	12.65%	0	0	0
AEAD	5925	6.12%	2497	4.79%	0	0	7
Chacha	3118	3.21%	2497	4.79%	0	0	0
Poly	1268	1.31%	2023	3.88%	0	0	7
SHA3	6200	6.39%	2820	5.41%	0	0	0
AES_GCM	2635	2.71%	4400	8.45%	0	0	0
HMAC-SHA2	2178	2.24%	1425	2.74%	2	1	0
TRNG	136	0.14%	563	1.08%	0	0	0
Other *	9708	10.00%	9613	18.45%	0	0	0

<sup>1</sup> Including the isolated sub-system. \* Bus system, debug module, peripherals, interrupt.



**Figure 11.** ASIC layout and micro-graph.

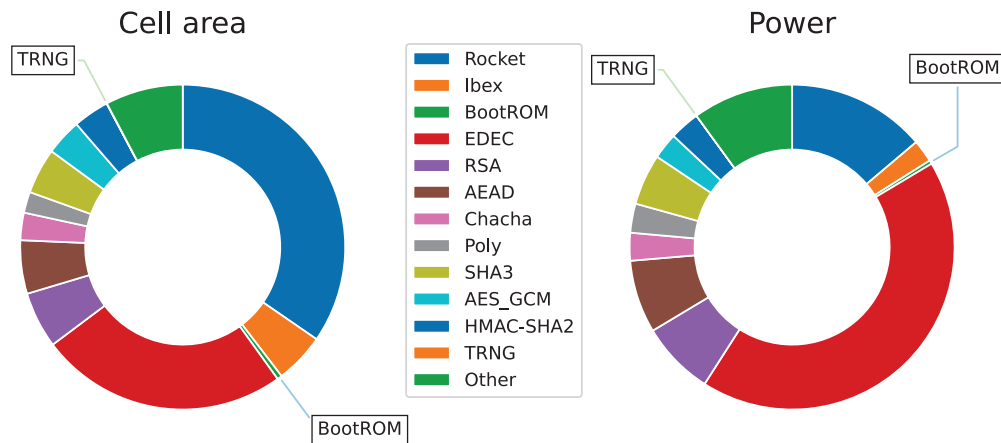
**Table 2.** Proposed TEE SoC in CMOS 180 nm synthesis result.

	Gate Equiv. (NAND2)	Area		Power			
		$\mu\text{m}^2$	%	Leakage (nW)	Dynamic (mW)	Total (mW)	%
Total	460,195	14,744,115	100.00	5487	3075	3075	100
Rocket	75,030	5,100,826	34.59	1213	425	425	13.82
core	15,337	372,392	2.53	177	182	182	5.92
dcache	25,398	2,509,375	17.02	456	154	154	5.01
icache	32,127	2,169,710	14.72	555	77	77	2.50
IBex <sup>1</sup>	17,681	737,478	5.00	201	69	69	2.24
BootROM	4272	70,672	0.48	21	11	11	0.36
ECED	166,720	3,638,115	24.68	1664	1311	1311	42.63
RSA	35,754	827,563	5.61	385	226	226	7.35
AEAD	30,345	783,675	5.32	349	223	223	7.25
Chacha	16,723	402,309	2.73	178	85	85	2.76
Poly	10,966	308,602	2.09	136	89	89	2.89
SHA3	26,873	669,773	4.54	292	156	156	5.07
AES_GCM	20,753	532,594	3.61	266	80	80	2.60
HMAC-SHA2	13,155	529,278	3.58	176	92	92	2.99
TRNG	268	3983	0.03	1	0.15	0.15	0.01
Other *	41,655	1,139,247	7.74	605	308	308	10.03

<sup>1</sup> Including the isolated sub-system. \* Bus system, debug module, peripherals, interrupt.

**Table 3.** Summary of TEE-HW with isolated architecture chip features.

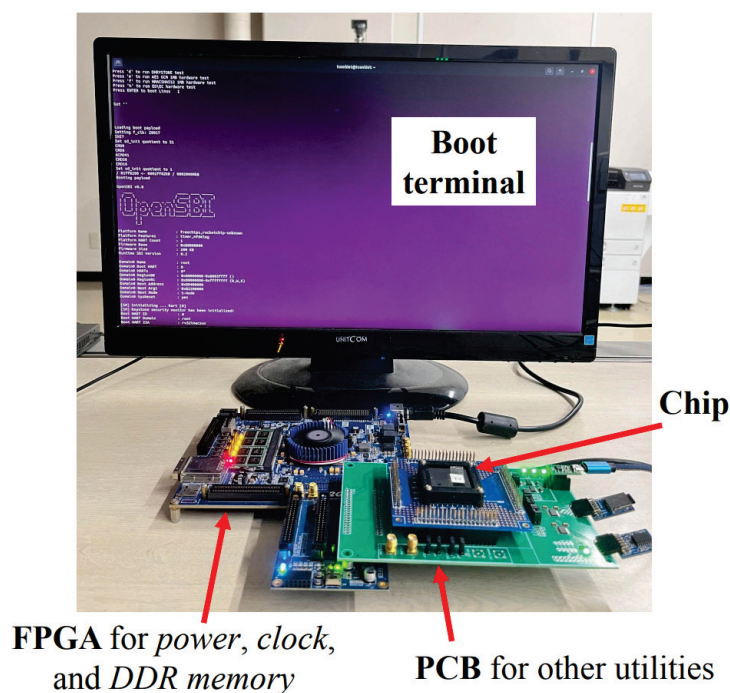
Process		CMOS 180 nm
Cores		Rocket
ISA		RV32GC
Caches	Instruction	16-KB (Rocket) + 4-KB (Ibex)
	Data L2	16-KB (Rocket) + 4-KB (Ibex) 512-KB
Area	Die	$5.0 \times 5.0\text{-mm}^2$
	Core	$4560.52 \times 4561.16\text{-}\mu\text{m}^2$ $=20.79\text{-mm}^2$ $\approx 1,535,406\text{-NAND2}$
	Cell	466,882
	MOSFET	7,982,582
$V_{DD}$	I/O	1.8-V
	Core	1.0-V to 2.0-V
Peak performance		at 2.0-V $V_{DD}$ $F_{Max} = 30\text{-MHz}$ $P_{Active} = 7.6\text{-mW/MHz}$



**Figure 12.** Cell area and power.

### 6.3. Performance Analysis

Besides an FPGA-based implementation, CMOS 180 nm chips were made for the demonstration. For better stability, the critical peripherals, such as Secure Digital card (SD card), Universal Asynchronous Receiver/Transmitter (UART), and Flash rely on Peripheral MODdule (PMOD) headers, were acquired from Digilent and attached with small circuits outside. Similar to the previous PCBs, this PCB also can choose a power supply and clock source. The power and clock can be provided by the FPGA or external sources via jumpers. Figure 13 shows the working PCB mounting on the TR5 FPGA board to use the FPGA's Dual In-line Memory Module (DIMM) Random Access Memory (RAM). In this way, we ensure that the peripherals, especially DIMM-RAM, work properly.

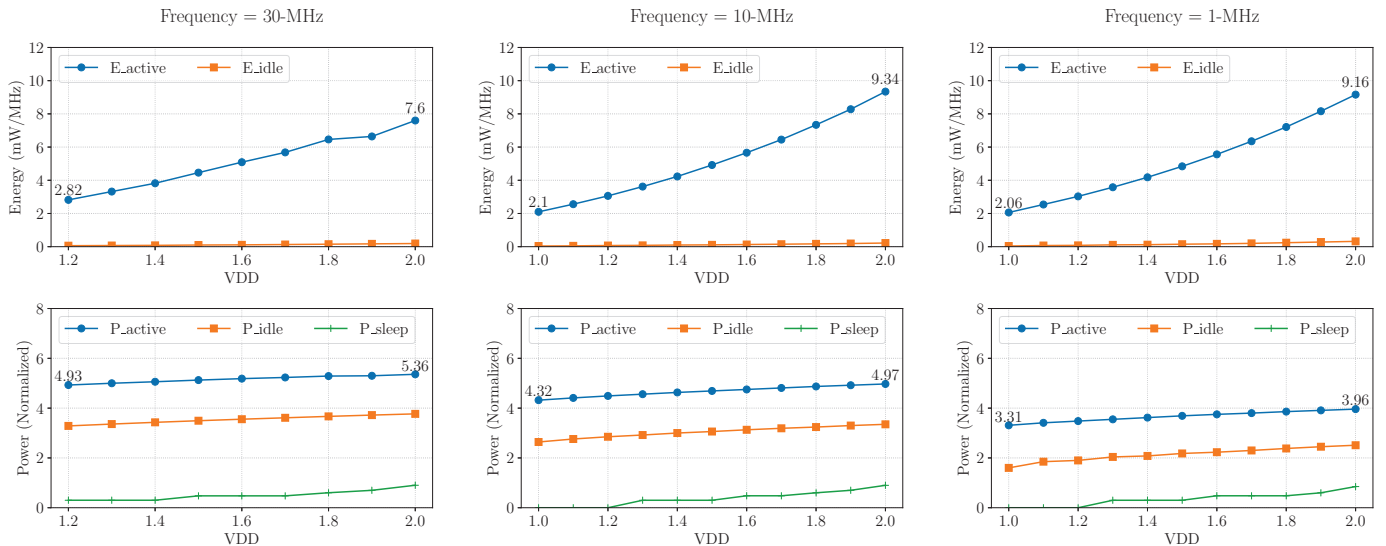


**Figure 13.** The TEE-HW with isolated architecture PCB mounts on the TR5 FPGA board.

In CMOS 180 nm technology, the default  $V_{TH}$  is about 1.0 V and the recommended operating  $V_{DD}$  is 1.8 V. Therefore, the CMOS 180 nm chip measurement was carried out with the  $V_{DD}$  range of 1.0 V to 2.0 V. The system is measured and works at a voltage of 30 MHz from a voltage higher than 1.2 V. However, it can work at the voltage from 1.0 V to 1.2 V for frequencies lower than 10 MHz. Figure 14 shows the changes in power and energy with different  $V_{DD}$  for the 32-bit  $5.0 \times 5.0\text{-mm}^2$  version. The statistic is collected for



three cases, including 30 MHz (which is the maximum frequency overall), 10 MHz (which is the maximum frequency at which the system can work in all ranges of voltage), and 1 MHz (which is the minimum voltage at which the system could work). Because there is a huge gap among active power  $P_{\text{active}}$ , which is the power when the system works, idle power  $P_{\text{idle}}$ , which is the power when the system does not work, and sleep power  $P_{\text{sleep}}$ , which is the power when the input clock is cut off, we normalize the power by the function  $\text{power}_{\text{normalized}} = 3 * \log_{10}(\text{power})$ . While the sleep power  $P_{\text{sleep}}$  is almost identical for different scenarios, the active power  $P_{\text{active}}$  increases with the  $V_{\text{DD}}$  and the frequency. Despite having the highest frequency after place and route, at 71 MHz, the fabricated chip can only work stably at 30 MHz due to the limitations of bonding and packaging techniques. Figure 14 also shows the active energy  $E_{\text{active}}$  and the idle energy  $E_{\text{idle}}$ . Despite the power being small and having a low frequency, the increase in execution time causes a reduction in power efficiency. The system achieves the best power efficiency, which is 7.6 W/MHz, when working at 30 MHz. Table 3 summarizes the features of the TEE SoC on CMOS 180 nm. Although the system's memory is identical between FPGA and ASIC deployment, the ROMs and small-sized BRAMs are converted to registers instead of SRAMs to reduce the delay. Therefore, the total size of SRAMs on ASIC is smaller than that of BRAMs on FPGA.



**Figure 14.** ASIC power and energy consumption.

#### 6.4. Security Analysis

The goals of the proposed TEE-HW are (i) accelerating the boot process by using crypto-cores and (ii) isolating the boot program with RoT from the TEE processors. The used TEE model in the implementation is Keystone [23]; thus, the proposed design inherits all of Keystone's advantages and disadvantages.

To summarize Keystone, the threat model of Keystone considers a strong software adversary that can compromise all of the software stack and a strong physical adversary that can corrupt peripherals and memory communications. A malicious enclave is also considered in the Keystone model. In Supervisor-mode (S-mode), the Eyrie runtime provides the Operating System (OS) equivalent services and ensures the validity of address mappings, thus preventing mapping attacks [44]. Furthermore, thanks to the runtime, the enclaves do not have to rely on the OS for critical functions; hence, they can defend against controlled SCAs that exploit the sharing states across domains, like interrupt handlers and table paging. In Machine-mode (M-mode), the Physical Memory Protection feature defines the memory access rights. Therefore, the PMP will deny any direct attempt to read the enclave's data, protecting against the software adversary [23]. In Keystone, the Secure Monitor (SM) performs a clean context switch by flushing enclave states. Together

with cache partitioning, the cache-based SCAs are prevented. Moreover, an enclave can be encrypted, and its page table can be self-managed; thus, any subtle attacks like controlled SCAs are impossible. Finally, Keystone also offers plugins to strengthen the TEE, such as memory encryption, enclave dynamic resizing, edge call service, and syscall services [23]. In Keystone and our implementation, the speculative attacks, timing SCAs, and SCAs that exploit hardware flaws in the off-chip components are considered out of scope. Finally, although possible, enclave-to-peripheral binding is not recommended in the current implementation. Introducing a peripheral driver into the runtime is not a two-way binding process, thus allowing Direct Memory Access (DMA)-capable peripheral attacks.

Regarding the boot flow, most TEE models, Keystone included, generally consider the RoT-based secure boot flow to be out of scope. That makes sense because, by definition, TEE is an isolated environment, not the RoT, and it should not include RoT. It is recommended that the RoT-based secure boot process be run by hardware primitives rather than the TEE processors themselves. Typically, propriety TEE models use third-party IPs or some extra hardware mechanism to deliver the secure boot. In Keystone, the trusted domain operates based on the assumption that the hardware signed the SM during boot. Therefore, RoT hardware is needed to deliver that secure boot process. By introducing a secure boot mechanism with silicon-level RoT to complete the CoT, the device's integrity is guaranteed.

With all of the cryptographic keys interlocked with each other, a direct attack on the key chain is impossible. For example, the public root key  $P_R$  and the secret device key  $S_D$  are stored in the hidden ROM inside the isolated domain.  $S_R$ , the secret root key, is not stored in the SoC or anywhere. In the public domain, only the public device key  $P_D$  is available after boot and for verification. Additionally, due to the isolation dictated by the bus architecture, even if a malicious enclave could hack the TEE side, it cannot retrieve any data in the hidden ROMs by any means. From the software perspective, exploiting the interrupt channel for attestation is the only attack surface left. But, as mentioned earlier, the IBex core's behavior is solely dictated by its program inside the isolated domain. Thus, if such an attack threat exists, the IBex program can be updated anytime to adapt to the new attack vector. To conclude, the proposed secure boot scheme can still safeguard its secrets even if the TEE processors were compromised. Our fabricated VLSI chip successfully boots with the proposed boot flow (see Figure 15).

### 6.5. Comparison and Discussion

Table 4 shows the Dhrystone test results between cores. Due to the difference in the ISAs, for a fair comparison, the Dhrystone test was repeated by the same IBex ISA of RV32IMC. For each test, the Dhrystone program was run 500 times, and the average value was recorded. As seen from the table, the Rocket achieved a good result of 1.573~1.713-DMIPS/MHz, while the 0.434-DMIPS/MHz result of IBex is a mid-range processor [45]. Specifically, the Rocket's DMIPS/MHz results were about  $3.62\times$  to  $3.95\times$  compared to IBex's. Although the IBex core is much slower than Rocket's, considering it still can use crypto-cores to accelerate its boot program, the boot speed when swapping a Rocket for an IBex is not much of a change.

**Table 4.** Dhrystone test comparison between IBex and Rocket cores.

Core	ISA	Dhrystone/s	DMIPS/MHz	Changes
Rocket	RV64GC	150,511	1.713	$3.95\times$
	RV32IMC	138,197	1.573	$3.62\times$
IBex	RV32IMC	38,165	0.434	$1.00\times$



```

/ 81ff8200 <- 0001ff82kB / 00020000kB
Booting payload

OpenSBI v0.8

      _ _ _ _ _
     | | | | |
    _||_|_|_|_
   | | | | |
   ||_|_|_|_|
  _||_|_|_|_
 | | | | |
|_|_|_|_|_|

Platform Name       : freechips,rocketchip-unknown
Platform Features   : timer,mfdeleg
Platform HART Count : 1
Firmware Base       : 0x80000000
Firmware Size       : 200 KB
Runtime SBI Version : 0.2

Domain0 Name        : root
Domain0 Boot HART    : 0
Domain0 HARTs        : 0*
Domain0 Region00     : 0x80000000-0x8003ffff ()
Domain0 Region01     : 0x00000000-0xffffffff (R,W,X)
Domain0 Next Address : 0x80400000
Domain0 Next Arg1     : 0x82200000
Domain0 Next Mode     : S-mode
Domain0 SysReset      : yes

[SM] Initializing ... hart [0]
[SM] Keystone security monitor has been initialized!
Boot HART ID         : 0
Boot HART Domain      : root
Boot HART ISA         : rv32imafdcxux
Boot HART Features     : scounteren,mcounteren
Boot HART PMP Count    : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 30
Boot HART MHPM Count   : 0
Boot HART MHPM Count   : 0
Boot HART MIDELEG      : 0x00000222
Boot HART MEDELEG      : 0x00000b109
[ 0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80400000
[ 0.000000] Linux version 5.7.0-dirty (tuankiet@ubuntu) (gcc version 11.1.0 (GCC), GNU ld (GNU Binutils)
[ 0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[ 0.000000] printk: bootconsole [sbi0] enabled
[ 0.000000] initrd not found or empty - disabling initrd
[ 0.000000] Zone ranges:
[ 0.000000]   Normal [mem 0x0000000080400000-0x00000000bfffffff]
[ 0.000000]   Movable zone start for each node
[ 0.000000]   Early memory node ranges
[ 0.000000]     node 0: [mem 0x0000000080400000-0x00000000bfffffff]
[ 0.000000]   Initmem setup node 0 [mem 0x0000000080400000-0x00000000bfffffff]
[ 0.000000] cma: Reserved 512 MiB at 0x9f800000
[ 0.000000] SBI specification v0.2 detected
[ 0.000000] SBI implementation ID=0x1 Version=0x8
[ 0.000000] SBI v0.2 TIME extension detected
[ 0.000000] SBI v0.2 IPI extension detected
[ 0.000000] SBI v0.2 RFENCE extension detected
[ 0.000000] SBI v0.2 HSM extension detected
[ 0.000000] riscv: ISA extensions acdfim
[ 0.000000] riscv: ELF capabilities acdfim
[ 0.000000] percpu: Embedded 12 pages/cpu s18956 r8192 d22004 u49152
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 259080
[ 0.000000] Kernel command line: console=hvc0 earlycon=sbi
[ 0.000000] Dentry cache hash table entries: 131072 (order: 7, 524288 bytes, linear)
[ 0.000000] Trade cache hash table entries: 65536 (order: 6, 262144 bytes, linear)

```

Figure 15. Boot on chip.

Table 5 compares the security and flexibility features. In ITUS [46,47], they try to solve the secure boot in TEE by a pure hardware approach. The new hardware modules, the Code Authentication Unit (CAU) and Key Management Unit (KMU), have been introduced. The KMU handles the key generation and key distribution by utilizing a PUF and a TRNG. For the CAU, an EC-DSA and an SHA-3 were used for authentication. Because its approach is solely hardware, it lacks the flexibility to adapt to new threats. In contrast, our suggested isolated sub-system is flexible and can be programmed for any cryptographic function. Compared to the works in [46,47], our proposed system has enough crypto-cores to provide a secure boot required by the TLS-1.3 standard.

**Table 5.** Comparison in terms of security and flexibility with recent security-driven implementations; ●, ◐, and ○, respectively, rank the performance from best to worst.

	CURE [22]	HECTOR-V [48]	WorldGuard [49]	ITUS [46,47]	This work
Open-source	○	○	◐	○	◐
Secure boot	◐	●	◐	●	●
Flexible boot	●	●	●	○	●
TEE isolation	○	○	○	●	●
Exclusive TEE processor	◐	●	◐	○	○
Exclusive secure storage	○	●	○	●	●
Secure I/O paths	●	●	◐	○	○
Crypto. accel.	○	○	◐	●	●
SCA resilience	●	●	◐	○	○
Hardware cost	●	◐	●	○	◐
High expressiveness	◐	●	◐	○	◐
Low porting efforts	○	○	◐	●	●

WorldGuard [49] enhances the security level of TEEs by implementing various IDs across the entire system; this improves the isolation between various OS stacks. However, because their goal is not the secure boot flow but to strengthen the existing TEE models, they use the conventional boot flow for the secure boot process. Specifically, they use the bootloader with hard-coded root keys pre-stored in the ROM. This bootloader is executed first to verify and load the secured channel into the main memory; that means both the boot program and the RoT are still in the TEE domain. Therefore, the WorldGuard approach is still vulnerable to conventional software-based side-channel attacks.

In HECTOR-V [48], the design comes from two novel ideas: the heterogeneous architecture for separating REE and TEE domains and the security-hardened TEE processor for SCA resilience. In HECTOR-V, the TEE processor is the one to execute the secure boot process. That means its secure boot program can be updated in the same way as our approach. However, because the secure boot program is still accessible from the TEE processor, and the REE and TEE share the same processor, there is still a risk of exposing the RoT to the public side, even though the secure storage element was introduced. In contrast, our method completely moves the RoT and its secure boot program from the TEE's eyes, thus eliminating the potential threats from the malicious TEE's enclaves.

CURE [22] is a new model that uses new hardware primitives to raise the security strength and fine-tune various TEE applications. Their implementation can support many types of enclaves simultaneously without affecting the isolation between them. In order to do that, many hardware modifications are introduced, from registers in the core and shared caches partitioning to the bus controller. Although the CURE implementation has achieved solid work for TEEs, it still assumes that the RoT was carried out during the reset. Therefore, regarding the RoT-based secure boot flow, CURE did not provide a solution other than the conventional method of hard-coded keys in ROM.

## 7. Conclusions

This work proposes a Trusted Execution Environment HardWare (TEE-HW) framework that is easy to use, flexible for various needs, and easy to update in the future. The framework offers not only a secured boot process but also sufficient crypto-accelerators, which are required by TLS 1.3. Based on the framework, a completed TEE-HW computer system was developed and tested. The proposed TEE-HW architecture contains several cryptographic accelerators to enhance boot performances and increase security. Finally, a heterogeneous architecture with an isolated sub-system was developed. The hidden Micro-Controller Unit (MCU) in the isolated architecture provides not only the secure RoT implementation but also the ability to adapt to the future changes of the boot sequence. The

architecture contains several crypto-cores, such as AES\_GCM, SHA3, and TRNG. Besides the essential cores for the boot process, the crypto-accelerators, such as HMAC-SHA2, Ed/EDDSA, RSA, and AEAD, allow the system to perform different secured protocols. The crypto-cores have been proven to be efficient not only for performance but also for security strength. The proposed TEE-HW SoC was tested on Field-Programmable Gate Array (FPGA) and then realized in a Very Large-Scale Integrated circuit (VLSI). Fabrication is performed with the CMOS 180 nm process, and the measurements are delivered.

There are some limitations that we are working to improve. Firstly, despite the provided framework supporting a secured boot process, the protection against side-channel attacks for the crypto-cores is not carefully considered. We are evaluating the side-channel attack scenarios on the proposed crypto-cores and will provide a better design for future work. Secondly, if the protection on I/O paths is not implemented, it could become a weak point in the design against potential attacks. We are fixing it in the next version of our proposed framework. Furthermore, we are considering expanding this work to Network-on-Chip (NoC) systems. Generally, an NoC system boots up a single main CPU first, establishing the secured functions for the network and its other cores. In this model, we must first ensure the security of the main core's boot process. This is precisely what we have achieved through our work. Next, the main core can use the supported crypto-accelerators in this framework to establish Network-on-Chip security.

**Author Contributions:** Supervision, C.-K.P. and T.-T.H.; methodology, T.-T.H., B.K.-D.-N., T.-K.D. and K.-D.N.; investigation, T.-T.H., B.K.-D.-N., T.-K.D., N.T.B. and K.-D.N.; writing—original draft preparation, B.K.-D.-N. and C.P.-Q.; writing—review and editing, N.-T.T., C.-K.P. and T.-T.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding

**Data Availability Statement:** Data are contained within the article.

**Acknowledgments:** The VLSI chip in this study has been fabricated through the activities of VLSI Design and Education Center (VDEC), the University of Tokyo, in collaboration with Synopsys, Inc., Cadence Design Systems Inc., Mentor Inc., Rohm Semiconductor (ROHM), and Nippon Systemware Co., Ltd. We also acknowledge the collaboration with Ho Chi Minh City University of Technology (HCMUT), VNU-HCM in facilitating this research.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Quarkslab. Introduction to Trusted Execution Environment: ARM's TrustZone. Retrieved Oct. 2018, 8, 2019.
2. Oracle Corporation. Working with UEFI Secure Boot. Available online: <https://docs.oracle.com/en/operating-systems/oracle-linux/secure-boot/sboot-OverviewofSecureBoot.html#sb-overview> (accessed on 21 June 2024).
3. Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted Execution Environment: What It is, and What It is Not. In Proceedings of the IEEE Trustcom/BigDataSE/ISPA (TrustCom), Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 57–64.
4. Intel Corp. *Intel Software Guard Extensions (Intel SGX) Developer Guide*; Intel Corp.: Santa Clara, CA, USA, 2018.
5. Costan, V.; Devadas, S. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086, January 2016. Available online: <https://eprint.iacr.org/2016/086> (accessed on 21 June 2024).
6. Costan, V.; Lebedev, I.; Devadas, S. Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture. *Found. Trends<sup>®</sup> Electron. Des. Autom.* **2017**, *11*, 1–248. [CrossRef]
7. Costan, V.; Lebedev, I.; Devadas, S. Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture. *Found. Trends<sup>®</sup> Electron. Des. Autom.* **2018**, *11*, 249–361. [CrossRef]
8. ARM Ltd. *ARM Security Technology: Building a Secure System Using TrustZone Technology*; Technical Report PRD29-GENC-009492C; ARM Ltd.: Cambridge, UK, 2009.
9. Pinto, S.; Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* **2019**, *51*, 1–36. [CrossRef]
10. Buhren, R.; Werling, C.; Seifert, J.-P. Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), London, UK, 11–15 November 2019; pp. 1087–1099.
11. Baumann, A.; Peinado, M.; Hunt, G. Shielding Applications from an Untrusted Cloud with Haven. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Broomfield, CO, USA, 6–8 October 2014; pp. 267–283.

12. Tsai, C.-C.; Porter, D.E.; Vij, M. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In Proceedings of the USENIX Annual Technical Conference (ATC), Santa Clara, CA, USA, 12–14 July 2017; pp. 645–658.
13. Arnautov, S.; Trach, B.; Gregor, F.; Knauth, T.; Martin, A.; Priebe, C.; Lind, J.; Muthukumaran, D.; O’Keeffe, D.; Stillwell, M.L.; et al. SCONE: Secure Linux Containers with Intel SGX. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA, USA, 2–4 November 2016; pp. 689–703.
14. Ferraiuolo, A.; Baumann, A.; Hawblitzel, C.; Parno, B. Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Shanghai, China, 28–31 October 2017; pp. 287–305.
15. Linaro Ltd. *Open Portable Trusted Execution Environment*; Linaro Ltd.: Cambridgeshire, UK, 2021.
16. Brasser, F.; Gens, D.; Jauernig, P.; Sadeghi, A.-R.; Stapf, E. SANCTUARY: ARMing TrustZone with User-space Enclaves. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 24–27 February 2019; pp. 1–15.
17. Kaplan, D. Protecting VM Register State with SEV-ES. *White Paper* 17 February 2017. Available online: <https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Protecting-VM-Register-State-with-SEV-ES.pdf> (accessed on 21 June 2024).
18. Sev-Snp, A.M.D. Strengthening VM Isolation with Integrity Protection and More. *White Paper* January 2020. Available online: <https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf> (accessed on 21 June 2024).
19. Hex Five Security, Inc. *MultiZone Hex-Five Security*; Hex Five Security, Inc.: Redwood Shores, CA, USA, 2024.
20. Costan, V.; Lebedev, I.; Devadas, S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 857–874.
21. Weiser, S.; Werner, M.; Brasser, F.; Malenko, M.; Mangard, S.; Sadeghi, A.-R. TIMBER-V: Tag-Isolated Memory Bringing Fine-Grained Enclaves to RISC-V. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–27 February 2019; pp. 1–15.
22. Bahmani, R.; Brasser, F.; Dessouky, G.; Jauernig, P.; Klimmek, M.; Sadeghi, A.-R.; Stapf, E. CURE: A Security Architecture with CUSTOMizable and Resilient Enclaves. In Proceedings of the USENIX Security Symposium (USENIX Security), Virtual Event, 11–13 August 2021; pp. 1073–1090.
23. Lee, D.; Kohlbrenner, D.; Shinde, S.; Asanovic, K.; Song, D. Keystone: An Open Framework for Architecting Trusted Execution Environments. In Proceedings of the European Conference on Computer Systems (EUROSYS), Heraklion, Greece, 27–30 April 2020; pp. 1–16.
24. He, F.; Zhang, H.; Wang, H.; Xu, M.; Yan, F. Chain of Trust Testing Based on Model Checking. In Proceedings of the International Conference on Networks Security, Wireless Communications and Trusted Computing (NSWCTC), Wuhan, China, 24–25 April 2010; Volume 1, pp. 273–276.
25. AMD Inc. Inside a Deeply Embedded Security Processor. In Proceedings of the Black Hat USA, Virtual Event, 1–6 August 2020; AMD Inc.: Santa Clara, CA, USA, 2020.
26. ARM Ltd. *ARM Security IP: CryptoCell-700 Family*; ARM Ltd.: Cambridge, UK, 2017.
27. Intel Corp. *Intel Active Management Technology (AMT) Developers Guide*; Intel Corp.: Santa Clara, CA, USA, 2024.
28. Rambus, Inc. *Security CryptoManager Provisioning*; Rambus, Inc.: Sunnyvale, CA, USA, 2022.
29. lowRISC CIC. OpenTitan. Available online: <https://github.com/lowRISC/opentitan> (accessed on 21 June 2024).
30. ISO/IEC 11889-1:2015; Information Technology—Trusted Platform Module Library—Part 1: Architecture. ISO/IEC: Geneva, Switzerland, 2015.
31. Furtak, A.; Bulygin, Y.; Bazhaniuk, O.; Loucaides, J.; Matrosov, A.; Gorobets, M. BIOS and Secure Boot Attacks Uncovered. In Proceedings of the Ekoparty Security Conference, Buenos Aires, Argentina, 29–31 October 2014; pp. 1–79.
32. Cui, E.; Li, T.; Wei, Q. RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access* **2023**, *11*, 24696–24711. [CrossRef]
33. Hoang, T.T.; Duran, C.; Serrano, R.; Sarmiento, M.; Nguyen, K.D.; Tsukamoto, A.; Suzuki, K.; Pham, C.K. Trusted Execution Environment Hardware by Isolated Heterogeneous Architecture for Key Scheduling. *IEEE Access* **2022**, *10*, 46014–46027. [CrossRef]
34. PHAM Laboratory. TEE Hardware Platform. Available online: <https://github.com/uec-hanken/tee-hardware> (accessed on 21 June 2024).
35. Serrano, R.; Duran, C.; Hoang, T.-T.; Sarmiento, M.; Nguyen, K.-D.; Tsukamoto, A.; Suzuki, K.; Pham, C.-K. A Fully Digital True Random Number Generator with Entropy Source Based in Frequency Collapse. *IEEE Access* **2021**, *9*, 105748–105755. [CrossRef]
36. Dworkin, M.J. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. August 2015. Available online: [https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions?pub\\_id=919061](https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions?pub_id=919061) (accessed on 21 June 2024).
37. FIPS-197; Advanced Encryption Standard (AES). NIST Standard: Gaithersburg, MD, USA, November 2001.
38. Krawczyk, H.; Bellare, M.; Canetti, R. RFC2104: HMAC: Keyed-Hashing for Message Authentication. February 1997. Available online: <https://dl.acm.org/doi/abs/10.17487/RFC2104> (accessed on 21 June 2024).
39. Nir, Y.; Langley, A. RFC8439: ChaCha20 and Poly1305 for IETF Protocols. June 2018. Available online: <https://datatracker.ietf.org/doc/rfc8439/> (accessed on 21 June 2024).
40. lowRISC CIC. IBEX RISC-V Core. Available online: <https://github.com/lowRISC/ibex> (accessed on 21 June 2024).

41. Barker, E.; Roginsky, A.; Davis, R. *Recommendation for Cryptographic Key Generation*; Technical Report; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2020.
42. SiFive, Inc. *SiFive TileLink Specication*; SiFive, Inc.: Santa Clara, CA, USA, 2019.
43. ARM. *AMBA AXI and ACE Protocol Specification*; Technical Report ARM IHI 0022H.c; ARM: Cambridge, UK, 2021.
44. Hofmann, O.S.; Kim, S.; Dunn, A.M.; Lee, M.Z.; Witchel, E. InkTag: Secure Applications on an Untrusted Operating System. *ACM SIGPLAN Not.* **2013**, *48*, 265–278. [CrossRef]
45. Stratify Labs. *Dhrystone Benchmarking on MCUs*; Stratify Labs: Highland, UT, USA, 2019.
46. Kumar, V.B.Y.; Chattopadhyay, A.; Yahya, J.H.; Mendelson, A. ITUS: A Secure RISC-V System-on-Chip. In Proceedings of the IEEE International System-on-Chip Conference (SOCC), Singapore, 3–6 September 2019; pp. 418–423.
47. Yahya, J.H.; Wong, M.M.; Pudi, V.; Bhasin, S.; Chattopadhyay, A. Lightweight Secure-Boot Architecture for RISC-V System-on-Chip. In Proceedings of the International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 11–13 March 2019; pp. 216–223.
48. Nasahl, P.; Schilling, R.; Werner, M.; Mangard, S. HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment. In Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS), Hong Kong, China, 7–11 June 2021; pp. 187–199.
49. SiFive, Inc. *Securing the RISC-V Revolution*; SiFive, Inc.: Santa Clara, CA, USA, 2019.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.



## Article

# FPGA-Based Acceleration of Polar-Format Algorithm for Video Synthetic-Aperture Radar Imaging

Dongmin Jeong <sup>1</sup>, Myeongjin Lee <sup>1,2</sup>, Wookyung Lee <sup>2</sup> and Yunho Jung <sup>1,2,\*</sup>

<sup>1</sup> Department of Smart Air Mobility, Korea Aerospace University, Goyang-si 10540, Republic of Korea; ehdals5307@kau.kr (D.J.); artistic@kau.ac.kr (M.L.)

<sup>2</sup> School of Electronics and Information Engineering, Korea Aerospace University, Goyang-si 10540, Republic of Korea; wklee@kau.ac.kr

\* Correspondence: yjung@kau.ac.kr; Tel.: +82-2-300-0133

**Abstract:** This paper presents a polar-format algorithm (PFA)-based synthetic-aperture radar (SAR) processor that can be mounted on a small drone to support video SAR (ViSAR) imaging. For drone mounting, it requires miniaturization, low power consumption, and high-speed performance. Therefore, to meet these requirements, the processor design was based on a field-programmable gate array (FPGA), and the implementation results are presented. The proposed PFA-based SAR processor consists of both an interpolation unit and a fast Fourier transform (FFT) unit. The interpolation unit uses linear interpolation for high speed while occupying a small space. In addition, the memory transfer is minimized through optimized operations using SAR system parameters. The FFT unit uses a base-4 systolic array architecture, chosen from among various fast parallel structures, to maximize the processing speed. Each unit is designed as a reusable block (IP core) to support reconfigurability and is interconnected using the advanced extensible interface (AXI) bus. The proposed PFA-based SAR processor was designed using Verilog-HDL and implemented on a Xilinx UltraScale+ MPSoC FPGA platform. It generates an image 2048 × 2048 pixels in size within 0.766 s, which is 44.862 times faster than that achieved by the ARM Cortex-A53 microprocessor. The speed-to-area ratio normalized by the number of resources shows that it achieves a higher speed at lower power consumption than previous studies.

**Keywords:** synthetic-aperture radar (SAR); video synthetic-aperture radar (ViSAR); polar-format algorithm (PFA); fast Fourier transform (FFT); systolic array (SA); interpolation; field-programmable gate array (FPGA)

## 1. Introduction

Synthetic-aperture radar (SAR) is used in military and civilian applications to observe land and ocean surfaces by transmitting and receiving radio waves in the air. The radar is aircraft- or satellite-mounted and uses the changes in antenna position as it moves around to mathematically synthesize antennas with small apertures, so as to create antennas with large apertures, allowing for a high azimuth resolution. This allows high-resolution images to be generated and, since it uses radio waves, it has the advantage of being able to acquire images day or night and in foggy, cloudy, or misty conditions. Thus, SAR's role has been increasing in recent years and it has become an important and active field in radar research [1–3].

Used for the important task of Earth observation, SAR is unaffected by environmental conditions and can be for emergency response to disasters including earthquakes, volcanoes, floods, landslides, and coastal inundation, as well as for military surveillance and reconnaissance [4–7]. Crucially, to analyze and identify targets in these images for rapid response, SAR images must be acquired in real time. More recently, researchers have been working on creating videos from multiple images generated in real time using SAR. Video



SAR (ViSAR) observes dynamic targets by surveilling the target area and generating images at a high frame rate [8–12].

In practice, however, a challenge in ViSAR imaging is the enormous amount of signal processing required to produce the final images from large amounts of raw echo data. SAR images can be generated using multiple pulses in the azimuth direction, and the more pulses used, the better the resolution of the image. This limits the ability to generate video because the aircraft needs to collect raw data for a long time while the frame rate is fixed by the PRF and the number of pulses. To solve this problem, pulses are overlapped to generate an image, and the frame rate can be increased by increasing the number of overlapping sections. However, to achieve a high overlap rate, the image generation must be sufficiently fast.

Due to these issues, ViSAR systems have typically been deployed on satellites or large aircraft because they require high power and large processing units. However, recent improvements in hardware accelerator performance, coupled with the rapid development of the small drone industry, have encouraged research into SAR-mounted small drones [13]. SAR systems mounted on drones have the advantage of being able to navigate terrain that is inaccessible to people, making them very useful for surveillance and reconnaissance applications. However, drone-mounted ViSAR systems are highly constrained by weight, space, and power, making it critical to choose the right imaging algorithm and acceleration platform.

Since improving efficiency from a signal processing perspective is limited, ideas for accelerating SAR imaging include using a graphic processing unit (GPU) or accelerating using a field-programmable gate array (FPGA) [14–22]. GPUs are developed for high-performance computations by utilizing characteristics such as high parallelism, multi-threading, and large bandwidth utilizing many cores. Existing research has shown that GPU-based methods can process large amounts of raw data and increase computational efficiency, which can improve the efficiency of SAR imaging by an order of magnitudes [14,18,19]. However, for use on drones, they are disadvantaged by extremely high power consumption, size, and weight. Therefore, embedded GPUs must be used, which still suffer from high power consumption.

Compared to general-purpose CPUs and GPUs, FPGAs offer significant advantages regarding high throughput, low latency, and energy efficiency for compute-intensive applications. Over the years, the rapid evolution of FPGAs with high processing power has led to their widespread use and adoption for SoCs with on-chip CPUs, domain-specific programmable accelerators, and multiple connectivity options [23,24]. Compared to GPUs, FPGAs consume much less power, have high throughput with abundant on-chip memory and computational resources, and in a multichannel configuration enable powerful parallelism like GPUs, allowing SAR signal-processing algorithms to run at very high speeds [25,26]. Unlike GPUs, which are used for general purposes, FPGAs can be easily optimized for specific applications, and they can thus operate faster than GPUs in situations requiring specific performance characteristics [16]. Their low power consumption, small size, and high computing power make FPGAs well-suited to accelerating ViSAR systems on drones.

Algorithm selection is also an important decision for ViSAR systems, and the choice depends on the operating mode of the SAR system. SAR systems operate in different modes, including spotlight mode, strip-map mode, scan mode, and inverse SAR (ISAR). In spotlight mode, the antenna is pointed at the target area to collect data, allowing the detection of individual objects or the collection of data on a specific area. This mode is ideal for applications that require high-resolution imaging generated over a long period of data collection or small chunks of data over multiple scenes. Therefore, for applications involving drone-based surveillance of a specific area, it is often preferable to operate in spotlight mode.

There are several representative algorithms for high-resolution image generation using SAR, such as the polar-format algorithm (PFA), the back-projection algorithm (BPA), and

the range-Doppler algorithm (RDA) [27–31]. Imaging algorithms for spotlight mode mainly use the BPA and the PFA [32–36]. The BPA is an algorithm that, ideally, produces high-resolution images in the time domain with no degradation due to motion. It has the best image quality but it requires  $O(N^3)$  computation to form an  $N \times N$ -sized image, which extends the image generation time. Conversely, the PFA is computationally efficient, and the number of operations it requires to process an  $N \times N$  pixel SAR image is  $O(N^2 \log_2 N)$ , which is fast [37]. In addition, the PFA also has the advantage of achieving fine resolution when operated in spotlight mode because it properly compensates for the constantly changing reflectivity of the target at much larger scene sizes.

The PFA converts the phase data collected in the polar coordinate system from the center of the observed location to the Cartesian coordinate system by performing range and azimuth interpolation, and it then employs fast Fourier transform (FFT) to generate an image. Therefore, to accelerate the PFA, the interpolation operation and the FFT operation must be accelerated. The FFT unit's hardware architecture is categorized as either a single butterfly structure [38], a pipeline structure [39], or a systolic array structure [40]. Although the single butterfly and pipeline structures can be implemented in a small area, ViSAR systems require high computational speed and the fastest and most suitable is the parallel structure based on systolic arrays [41,42]. Among various systolic array structures, the base-4-based systolic array structure is easy to implement, scalable, and best satisfies the trade-off between area and execution time [43–48]. Therefore, the base-4 systolic array structure was adopted. In addition, there are various interpolation algorithms, such as linear interpolation, sinc interpolation, and spline interpolation. Among these, the linear interpolation algorithm requires the smallest hardware area and the fastest speed when implemented, so we adopted the linear interpolation method because it is advantageous for drone ViSAR applications. Each accelerator integrated with an FPGA can be fine-tuned for PFA optimization and can yield a significant acceleration benefit.

This study proposes a PFA-based SAR processor for small drone ViSAR imaging. To accelerate the PFA-based ViSAR imaging, an FFT unit based on the systolic array structure was implemented, and an interpolation unit optimized for the PFA was designed. By designing each unit as a reusable block (IP core), reconfigurability is supported, and the units are interconnected using the advanced extensible interface (AXI) bus. In VLSI implementation, SRAM uses multiple transistors and is costlier, less dense, and more power-consuming than capacitor-based DRAM, which occupies less space. To compensate for these drawbacks, the proposed design minimizes the use of SRAM by using an AXI4 bus-based design to transfer data to and from DRAM. This achieves fast acceleration while reducing cost, footprint, and power usage, which benefits ViSAR applications in drones.

The remainder of this paper is organized as follows. Section 2 describes the ViSAR image frame analysis and the PFA image-generation algorithm; Section 3 describes the hardware architecture of the proposed PFA-based SAR processor; Section 4 presents the implementation results of the proposed design and the acceleration experimental results for different sizes of data and compares them with previous works. Section 5 concludes this paper.

## 2. Background

### 2.1. Geometry and Frame Rate of Video SAR

Figure 1 illustrates the geometry used for image generation in a SAR system in Spotlight mode:  $\theta$  represents the azimuth angle between pulses,  $\phi$  denotes the elevation angle, and  $R_a$  is the slant range from the center  $O$  to the antenna phase center (APC). In a SAR system using spotlight mode, the aircraft flies in a circle around the center  $O$  of the ground to be observed. By attaching radar sensors to the sides of the aircraft, the data are obtained by continuously looking at the center of the ground to be observed, so that data are obtained for a polar coordinate grid. In this case,  $\theta$  can be expressed as

$$\theta = \frac{V}{R_a \cos \phi \cdot f_{\text{PRF}}} \quad (1)$$

where  $f_{PRF}$  denotes the pulse repetition frequency and  $V$  represents the velocity of the platform. To create a SAR image, the radar sensor moves in the azimuth direction, receiving multiple azimuth pulses, which are used to generate the image. The synthetic aperture angle can be expressed as

$$\theta_{az} = N_p \cdot \theta \quad (2)$$

where  $N_p$  represents the number of pulses obtained as the aircraft moves.

Resolution is a critical parameter in SAR imaging. The range resolution  $\rho_r$  and the azimuth resolution  $\rho_a$  are defined as shown in Equation (3):

$$\rho_r = \frac{c}{2B \cos \phi}, \quad \rho_a = \frac{\lambda}{2\theta_{az} \cos \phi} \quad (3)$$

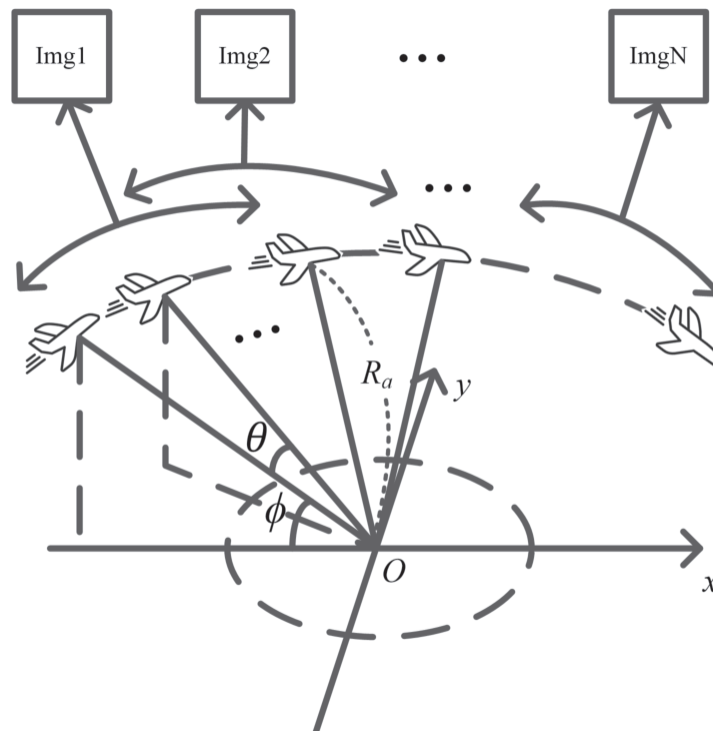
where  $c$  represents the speed of light,  $\lambda$  denotes the wavelength, and  $B$  signifies the bandwidth. Therefore, increasing the bandwidth can improve the range resolution, and to enhance the azimuth resolution many pulses in the azimuth direction must be used to increase  $\theta_{az}$ . Consequently, the aircraft must collect myriad pulse data, and the frame rate  $r$  is defined as [49–51]

$$r = \frac{f_{PRF}}{N_p} \quad (4)$$

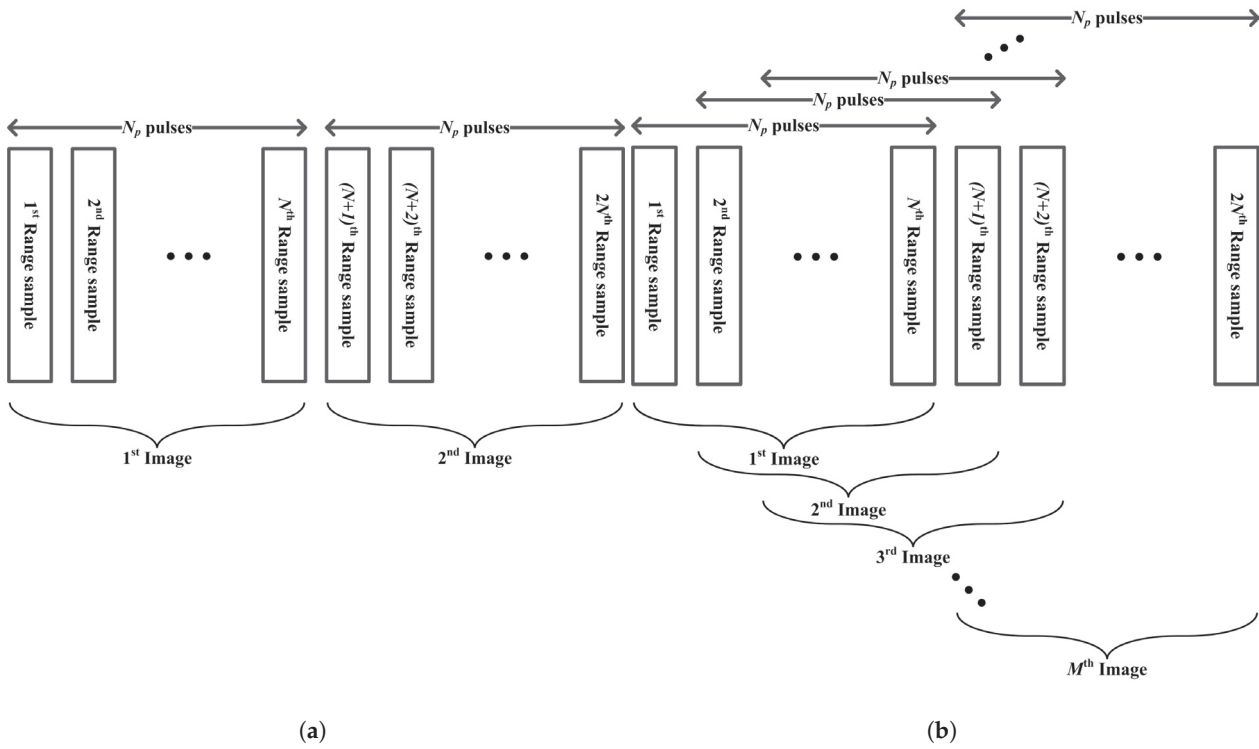
High frame rates must be achieved to generate video SAR images, and, according to Equation (4), they have a limit. An overlapping method is utilized to solve this issue. This approach is illustrated in Figure 2, where the equation is expressed as

$$r = \frac{f_{PRF}}{N_p - N_a} \quad (5)$$

where  $N_a$  is the number of overlapping pulses. A higher  $N_a$  enables higher frame rates, but to increase the number of overlapping pulses, the data obtained after the overlap interval must be imaged immediately, which requires sufficiently fast image generation.



**Figure 1.** Geometry for SAR image generation.



**Figure 2.** Aperture mode for video SAR: (a) non-overlapped mode; (b) overlapped mode.

## 2.2. Polar-Format Algorithm

Utilized in spotlight mode, the PFA is valued for its computational efficiency, characterized by fast  $O(N^2 \log_2 N)$  computation. This contrasts with the BPA's  $O(N^3)$  and matched filtering algorithms'  $O(N^4)$  spotlight generation complexities. The PFA includes a two-dimensional scattering model that assumes a flat scene, even in three-dimensional SAR systems, and image generation involves an IFFT of the received phase history. However, since the acquired data are structured on a polar grid it needs two-dimensional interpolation to convert it into a rectangular grid. Ideally, two-dimensional interpolation would suffice. However, in practice, this is achieved by first performing one-dimensional interpolation in the range direction, followed by another in the azimuth direction, thus avoiding the computational burden of two-dimensional interpolation. Afterward, the data interpolated onto the rectangular grid can be used to generate an image by performing FFT in the range and azimuth directions.

The transmitting linear frequency modulation (LFM) echo signal in the SAR system shown in Figure 3 is defined as

$$s_t(n, \tau) = e^{j2\pi f_c \tau + j\pi \gamma \tau^2} \quad (6)$$

where  $n$  is the pulse number,  $\tau$  represents the fast time,  $f_c$  denotes the center frequency, and  $\gamma$  signifies the chirp rate. When this signal is reflected by the target, the received signal is defined as

$$s_r(n, \tau) = \sigma(x_t, y_t, z_t) \cdot e^{j2\pi f_c \left(\tau - \frac{2R_t}{c}\right)} \cdot e^{j\pi \gamma \left(\tau - \frac{2R_t}{c}\right)^2} \quad (7)$$

where  $\sigma(x_t, y_t, z_t)$  represents the target reflectivity at position  $(x, y, z)$  and  $R_t$  is the distance between the SAR sensor and the target. The objective of SAR image generation is to obtain the reflectivity corresponding to  $\sigma$  through mathematically converting or approximating

the received signal. The received signal must first be motion-compensated for the center point, and the required reference signal is expressed as

$$s_{\text{ref}}(n, \tau) = e^{j2\pi f_c(\tau - \frac{2R_a}{c})} \cdot e^{j\pi\gamma(\tau - \frac{2R_a}{c})^2} \quad (8)$$

where  $R_a$  represents the distance between the SAR sensor and the scene center, which becomes the reference signal needed for compensation. Then, the complex conjugate of Equation (8) is multiplied by Equation (7) to perform motion compensation:

$$s(n, \tau) = s_r \cdot s_{\text{ref}}^* \quad (9)$$

Equation (9) can be expressed as

$$s(n, \tau) = \sigma(x_t, y_t, z_t) e^{-j\frac{4\pi}{c}(f_c + \gamma(\tau - \frac{2R_a}{c}))(R_t - R_a) + \frac{4\pi\gamma}{c^2}(R_t - R_a)^2} \quad (10)$$

where the last term in Equation (10) represents the residual video phase error, which should generally be removed. This is the range deskew process, and to compensate for it we take the derivative of the instantaneous frequency regarding fast time and multiply it by the following formula:

$$S_{\text{com}}(f_\tau) = e^{-j\left(\frac{\pi f_\tau^2}{\gamma}\right)} \quad (11)$$

We can multiply Equation (11) by the Fourier transform of Equation (10) and then re-perform the inverse Fourier transform to obtain the following equation with the residual video phase (RVP) removed:

$$s(n, \tau) = \sigma(x_t, y_t, z_t) \cdot e^{-j\frac{4\pi}{c}(f_c + \gamma(\tau - \frac{2R_a}{c}))(R_t - R_a)} \quad (12)$$

The Taylor approximation is used for Equation (12) to approximate the neighborhood of  $(x, y, z) = (0, 0, 0)$  for a function called  $R_t$ . In this case,  $R_t - R_a$  is defined as

$$R_t - R_a = -\sin \phi_a (x_t \cos \theta_a + y_t \sin \theta_a + z_t \cot \phi_a) \quad (13)$$

where  $\phi_a$  is the incidence angle and  $\theta_a$  is the squint angle. Since we assume a flat two-dimensional scene, the value along the  $z$ -axis can be ignored. If we define  $K_R$ ,  $K_u$ , and  $K_v$  as the spatial wavenumber, range wavenumber, and azimuth wavenumber, respectively, it can be expressed as follows:

$$K_R = \frac{4\pi}{c} \left( f_c + \gamma \left( \tau - \frac{2R_a}{c} \right) \right) \quad (14)$$

$$K_u = K_R \sin \phi_a \cos \theta_a \quad (15)$$

$$K_v = K_R \sin \phi_a \sin \theta_a \quad (16)$$

Substituting the above equation into Equations (12) and (13), it can be rewritten as

$$s(n, K_R) = \sigma(x_t, y_t) e^{j(x_t K_u + y_t K_v)} \quad (17)$$

where  $s(n, K_R)$  is uniformly distributed in the  $(n, K_R)$  domain but not uniformly in  $(K_v, K_u)$ . To utilize an efficient 2D fast Fourier transform (2D-FFT), 2D resampling must be performed on the wavenumber domain signal to distribute it uniformly in a rectangular format. Ideally, this is achieved through two-dimensional wavenumber domain interpolation, but it is much more computationally efficient to perform a two-dimensional interpolation through one-dimensional interpolation in the range direction followed by another in the azimuth direction. After the complete 2D resampling of the signal, the PFA image can be obtained

by using the 2D-FFT once for a wavenumber domain signal distributed in a rectangular format, expressed by the following equation:

$$I_p(x, y) = \int_{K_{ui}} \int_{K_{vi}} S(K_{ui}, K_{vi}) \cdot \exp\{-j(xK_{ui} + yK_{vi})\} dK_{ui} dK_{vi} \quad (18)$$

where  $K_{ui}$  and  $K_{vi}$  are the values obtained by linear interpolation of the  $K_u$  and  $K_v$  values, distributed in a uniform rectangular format. Figure 4 shows a block diagram of the image generation process after motion compensation and de-chirping:

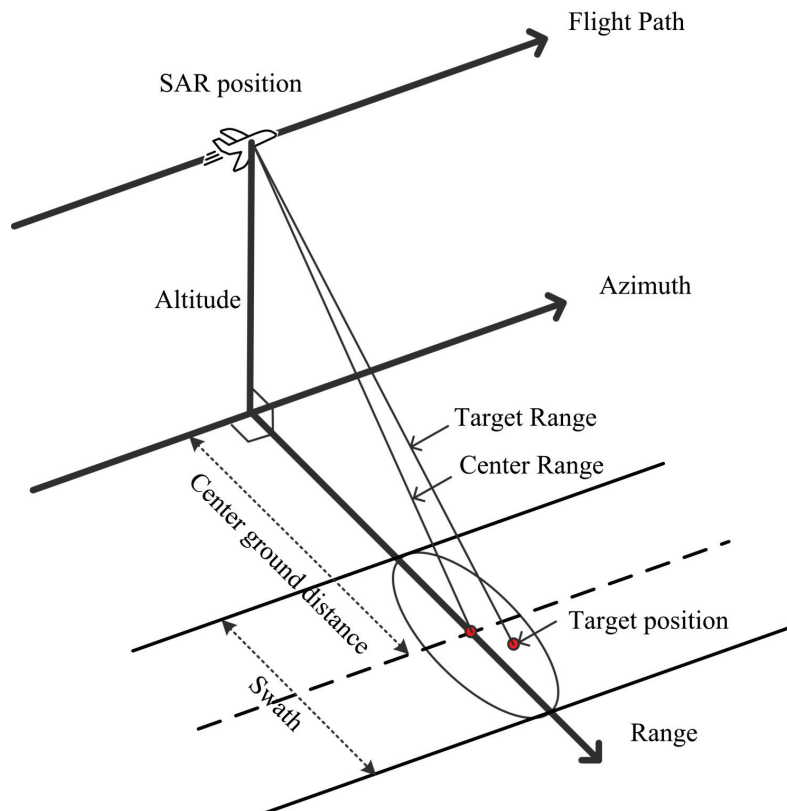


Figure 3. Geometry for SAR operation.

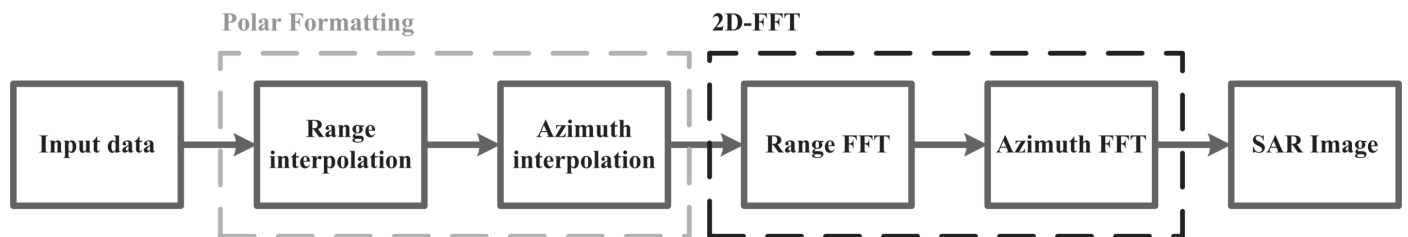


Figure 4. Procedure of the PFA.

### 3. Proposed Hardware Architecture

#### 3.1. Base-4 Systolic Array FFT Unit Hardware Architecture

The  $N$ -length discrete Fourier transform (DFT) is defined as Equation (19):

$$Z(k) = \sum_{n=0}^{N-1} W_N^{nk} X(n) \quad (19)$$



where  $W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$ , also termed the twiddle factor, where  $n$  is the input value in the time domain and  $k$  is the output value in the frequency domain. If we can decompose  $N = N_1 N_2$  then we can express  $n$  and  $k$  as

$$\begin{aligned} n &= n_1 + N_1 n_2, \quad (0 \leq n_1 \leq N_1 - 1, \quad 0 \leq n_2 \leq N_2 - 1) \\ k &= k_1 + N_1 k_2, \quad (0 \leq k_1 \leq N_1 - 1, \quad 0 \leq k_2 \leq N_2 - 1) \end{aligned} \quad (20)$$

Substituting the above equation into the DFT definition, Equation (19) can be rewritten as

$$Z(k_1 + N_1 k_2) = \sum_{n_1=0}^{N_1-1} \left( W_N^{n_1 k_1} \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_1} W_{N_2}^{n_2 k_2 N_1} X(n_1 + N_1 n_2) \right) W_{N_2}^{n_1 k_2} \quad (21)$$

where, assuming that  $N_1/N_2$  are integer values, we have  $W_{N_2}^{n_2 k_2 N_1} = e^{-j\left(\frac{2\pi n_2 k_2 N_1}{N_2}\right)} = 1$ , and we can express Equation (21) as Equation (22):

$$Z(k_1 + N_1 k_2) = \sum_{n_1=0}^{N_1-1} \left( W_N^{n_1 k_1} \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_1} X(n_1 + N_1 n_2) \right) W_{N_2}^{n_1 k_2} \quad (22)$$

In Equation (22), the value in parentheses is termed  $Y$  and the sigma operation is expressed as a matrix operation, as shown in Equation (23):

$$Y(k_1, n_1) = W_N^{n_1 k_1} \begin{bmatrix} W_{N_2}^0 & W_{N_2}^{k_1} & W_{N_2}^{2k_1} & \dots & W_{N_2}^{(N_2-1)k_1} \end{bmatrix} \times \begin{bmatrix} X(n_1) \\ X(n_1 + N_1) \\ X(n_1 + 2N_1) \\ \vdots \\ X(n_1 + (N_2 - 1)N_1) \end{bmatrix} \quad (23)$$

Then, the overall sigma operation in Equation (22) can be represented as a matrix operation using  $Y(k, n)$  obtained from Equation (23), as shown in Equation (24):

$$Z(k_1 + N_1 k_2) = \begin{bmatrix} W_{N_2}^0 & W_{N_2}^{k_2} & W_{N_2}^{2k_2} & \dots & W_{N_2}^{(N_2-1)k_2} \end{bmatrix} \times \begin{bmatrix} Y(k_1, 0) \\ Y(k_1, 1) \\ Y(k_1, 2) \\ \vdots \\ Y(k_1, N_2 - 1) \end{bmatrix} \quad (24)$$

Finally, the process of Equations (23) and (24) obtained earlier can be represented by the respective matrix expressions as Equation (25):

$$\begin{aligned} Y &= W_M \circ C_{M1} X \\ Z &= C_{M2} Y^T \end{aligned} \quad (25)$$

where  $W_M = W_N^{n_1 k_1}$ , an  $N_1 \times N_1$  matrix,  $\circ$  is elementwise multiplication, and  $C_{M1} = W_{N_2}^{n_2 k_1}$ , an  $N_1 \times N_2$  matrix.  $X = X(n_1 + N_1 n_2)$  and is an  $N_2 \times N_1$  matrix; thus,  $Y$  is an  $N_1 \times N_1$  matrix.  $C_{M2} = W_{N_2}^{n_1 k_2}$ , which is the same as  $C_{M1}^T$ ; therefore, it is an  $N_2 \times N_1$  matrix and  $Z$  is an  $N_2 \times N_1$  matrix. In the base-4 FFT algorithm,  $N_2 = 4$  is fixed, which can be changed depending on the application.

The base- $b$  FFT algorithm operates on one-dimensional data of length  $N$  using two-step factorization. Firstly, the data are partially separated into rows and columns, resulting in  $N = N_r N_c$ , where  $N_r$  represents the length of the rows and  $N_c$  represents the length of the columns. Second, the partial separation is further split into  $N_r = N_{1r} N_2$  and  $N_c = N_{1c} N_2$ , and the FFT is applied using Equation (25). This involves three main steps: firstly, the FFT operation defined in Equation (25) is applied  $N_r$  times in the row direction using column data of length  $N_c$ . Next, each element is multiplied by  $W_N$ . Finally, the FFT operation is

performed  $N_c$  times in the column direction using row data of length  $N_r$ . In essence, the one-dimensional data are transformed into a two-dimensional matrix of  $N_r \times N_c$  and the result is obtained in three steps: column FFT,  $W_N$  multiplication, and row FFT.

A systolic array comprises multiple process element (PE) cells that are locally connected, with each PE cell performing operations and passing data to the connected PE cells, as illustrated in Figure 5. Due to their regular, local data flow and the ability of multiple PE cells to perform operations simultaneously, systolic arrays are well-suited for algorithms requiring numerous operations, such as matrix product operations. Therefore, employing the systolic array structure for the FFT operation using Equation (25), which is organized as a product of matrices, enables high-speed processing of the FFT because the matrix product operation can be executed swiftly.

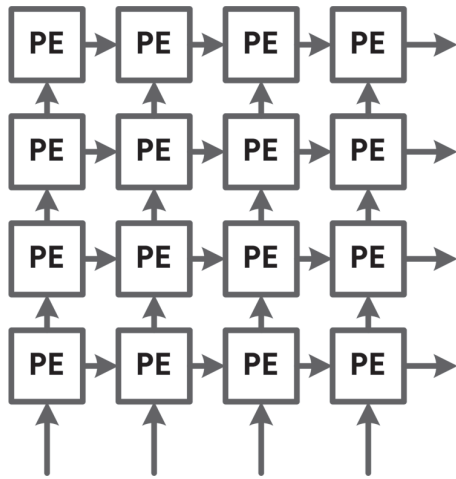


Figure 5. Architecture of the systolic array.

The systolic array FFT unit, designed to perform FFTs of length  $N = N_r N_c$ , has a four-channel structure based on the mathematically efficient  $b = 4$  in 'base-b'. It primarily consists of five components: an  $(N_r/4) \times 4$  array of PE cells on the left, referred to as the left-hand side (LHS); an  $(N_r/4) \times 1$  array of complex multipliers for multiplying  $W_M$ , the right-hand side (RHS) comprising an  $(N_r/4) \times 4$  array of PE cells on the right, four complex multipliers for multiplying  $W_N$ , and four memories, each sized  $N/4$ . Figure 6 illustrates the operations conducted within the PE cells of the LHS and RHS, respectively.

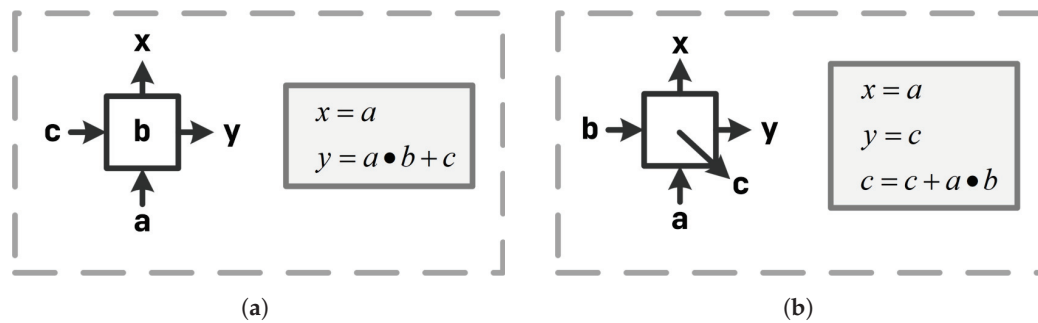


Figure 6. Operations inside PE cells: (a) LHS PE; (b) RHS PE.

The hardware structure of a base-4 systolic array FFT unit capable of performing a 4096-length FFT is depicted in Figure 7. It comprises a  $16 \times 4$  array of PE cells on the LHS and a  $16 \times 4$  array of PE cells on the RHS, with a  $16 \times 1$  array of  $W_M$  multipliers positioned between the LHS and RHS. Additionally, there are four  $W_N$  multipliers located on the right-hand side of the RHS, along with four 32-bit memories.

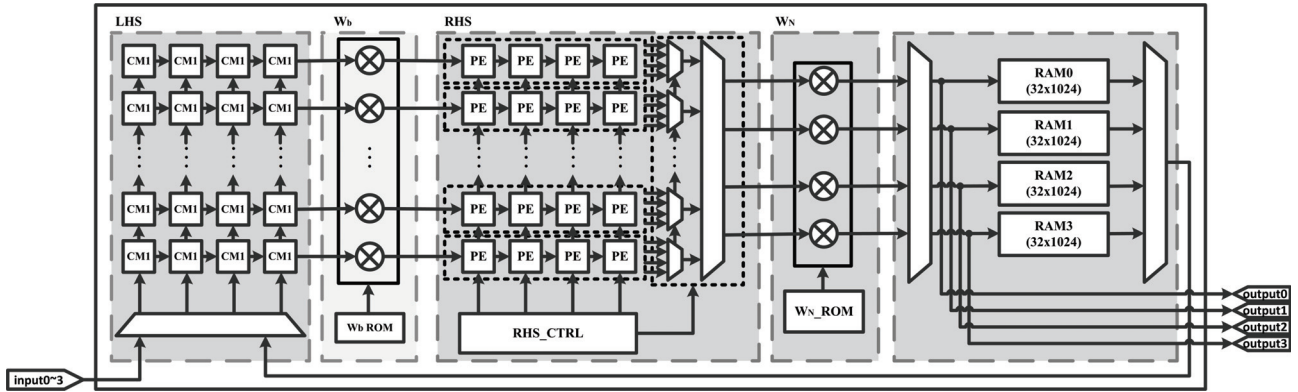


Figure 7. Hardware architecture of the proposed FP16 base-4 systolic array FFT unit.

The FFT process sequentially progresses through column DFT and row DFT. Initially, during the columnar DFT process, input  $X$  is received from the LHS. The present values of  $C_{M1}$  and  $X$  inside the PE cells undergo a matrix product operation, as depicted in Figure 8. The resulting product,  $C_{M1} \times X$ , is then passed to the  $W_M$  complex multiplier for multiplication by  $W_M$ . Subsequently, the product,  $W_M \circ C_{M1} \times X$ , is transmitted to the RHS, yielding the result  $Y$  as per Equation (23). Simultaneously, the  $C_{M2}$  value is received from the bottom, and  $C_{M2} \times Y^t$  is computed, resulting in  $Z$  as per Equation (24). This value represents the output of column DFT and is further processed by the  $W_N$  complex multiplier for multiplication by  $W_N$  before being stored in memory. Once the columnar DFT operation is completed, the results stored in memory are fed back to the LHS in the row direction. The modules  $W_M$ , RHS, and  $W_N$  (with  $W_N$  being multiplied by 1, essentially not performing the  $W_N$  operation at this stage) operate similarly to column DFT. Subsequently, the row DFT process is finalized and the result is output.

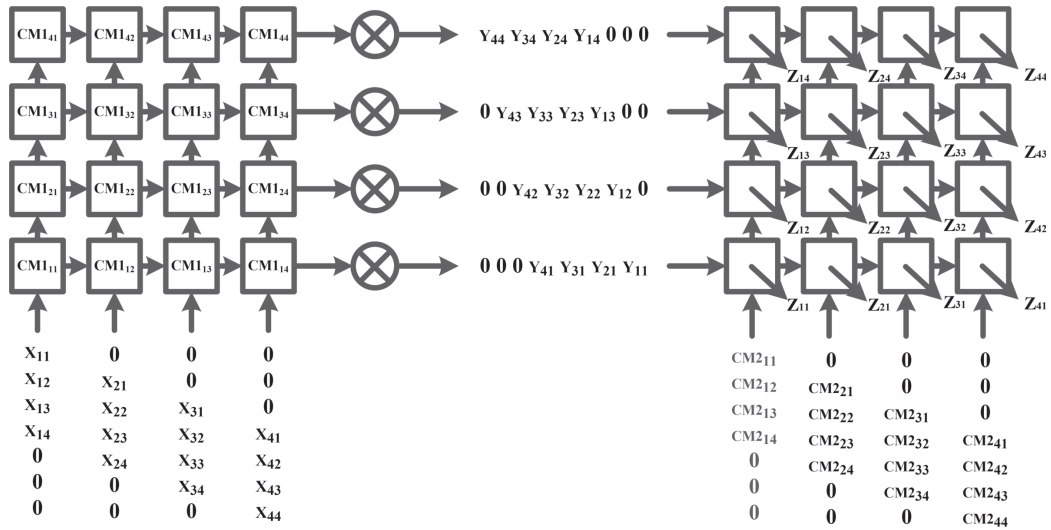


Figure 8. The base-4 systolic array FFT unit's computational procedure.

### 3.2. Interpolation Unit Hardware Architecture

The computational process of the PFA involves resampling the wavenumber domain signal into a uniform rectangular format. While two-dimensional wavenumber domain interpolation is often computationally expensive, a cost-saving approach involves performing interpolation in two separate parts: one in the range direction and another in the azimuth direction. Consequently, hardware for interpolation must be designed to expedite this computational process. Linear interpolation is typically favored for resampling tasks, due to its low hardware complexity and high speed.

The general interpolation equation for interpolating the value  $(x, y)$  between two points  $(x_0, y_0)$  and  $(x_1, y_1)$  is as follows:

$$y = y_0 + (x - x_0) \cdot \frac{y_1 - y_0}{x_1 - x_0} \quad (26)$$

where  $x_0 < x < x_1$  is satisfied. Substituting the above equation for the interpolation operation in the PFA process can be expressed as follows:

$$\hat{P}(a, r) = P(a, r) + (K_{ui} - K_u(a, r)) \cdot \frac{P(a, r+1) - P(a, r)}{K_u(a, r+1) - K_u(a, r)} \quad (27)$$

where  $a$  represents the pulse number in the azimuth direction,  $r$  signifies the sample number in the range direction,  $\hat{P}$  denotes the linearly interpolated phase history values,  $P$  stands for the uninterpolated phase history data,  $K_{ui}$  refers to the values at uniform locations in the  $K_u$  domain to be interpolated, and  $K_u$  represents the location values of each phase history uniformly present in the original  $K_R$  domain.

In the PFA, interpolation begins in the range direction. The process involves converting data uniformly distributed in the  $K_R$  domain to uniform data in the  $K_u$  domain. To achieve interpolation to uniformly distributed values in the  $K_u$  domain, we first need  $K_{ui}$ , which is the position value of the data uniformly distributed in the  $K_u$  domain. This value can be calculated and generated from the SAR system parameters: number of samples, number of pulses, platform distance, and resolution. With the obtained  $K_{ui}$  value, we can perform linear interpolation using Equation (27) to transfer the data uniformly present in the  $K_R$  domain to the uniform data in the  $K_u$  domain, as shown in Figure 9. Similarly, after the range interpolation, we perform the azimuth interpolation in the same manner, converting data uniformly present in the  $n$ -domain to uniform data in the  $K_v$ -domain. The end result is a rectangular shape. Since range interpolation and azimuth interpolation are similarly performed, a single piece of interpolation unit can perform both range and azimuth interpolation.

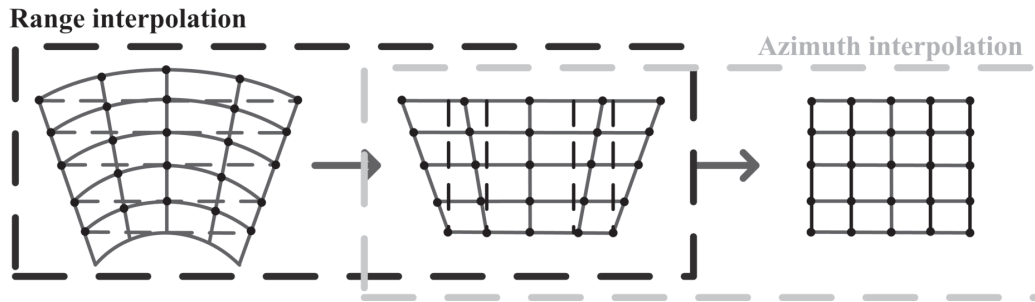
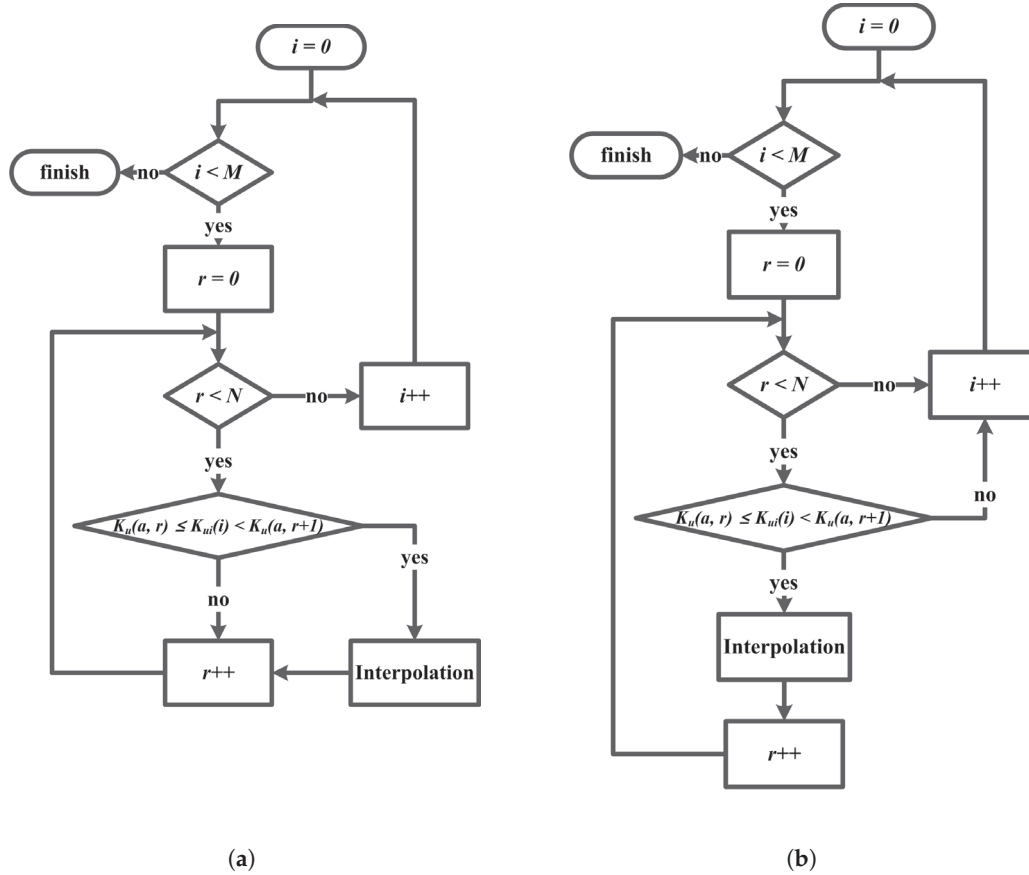


Figure 9. 2D resampling process.

To optimize the acceleration effect of a hardware accelerator, which is designed using an FPGA, it is critical to minimize the time taken for data transfer between the DRAM and the accelerator during computations. As Equation (27) shows, the process of linear interpolation to determine the value of  $\hat{P}$  requires two neighboring phase history data points, their corresponding values in the  $K_R$  domain,  $K_u$ , and the  $K_{ui}$  value at the location where interpolation is to be performed. In this case, the linear interpolation for the range direction must be repeated as many times as the number of azimuth pulses for every sample in the range direction, and the linear interpolation for the azimuth direction must be similarly performed, resulting in a large amount of data being transferred.

For example, suppose we want to perform a linear interpolation with a length of 512 in the range direction and 512 in the azimuth direction for phase history data with a pulse count of 352 and a sample count of 424. First, to perform the linear interpolation on the primary distance data, we require 424 phase history data points corresponding to  $P(0, 0) \dots P(0, 423)$  per Equation (27), and their corresponding values in the  $K_R$  domain,

$K_u(0,0) \dots K_u(0,423)$ . This process must be repeated 352 times, which is the length of the azimuth direction. This means that 424  $K_u$  data and 424 phase history data must be transferred for each iteration, which is extremely time-consuming. Furthermore, the typical procedure for executing the interpolation operation is depicted in Figure 10a. This involves executing a ‘for’ statement within a double loop. Due to the computational complexity being  $O(MN)$ , this process can be time-consuming.



**Figure 10.** Flow charts of interpolation operations: (a) typical; (b) proposed.

To address these issues, the proposed interpolation unit is organized as follows. First, to minimize data transmission, the proposed interpolation unit is designed to internally compute the  $K_u$  value inside the interpolation unit to be used in the computation, exploiting the fact that the value of  $K_u$  can be calculated from the parameter values of the SAR system. The  $K_u$  value can be calculated from the parameters of the SAR system, as shown in Equation (28):

$$K_u(a, r) = \hat{r} \cdot \hat{u} \cdot k_{\text{freq}} \quad (28)$$

where  $\hat{r}$  and  $\hat{u}$  are the unit vectors for the direction of the SAR system and  $k_{\text{freq}}$  is the frequency value of the LFM signal of the SAR system. Therefore, if  $K_u$  is calculated and used within the interpolation unit by utilizing the parameter values of the SAR system, the interpolation operation can be performed directly without data transfer. Furthermore, the number of operations can be reduced by utilizing the characteristics of the  $K_u$  value calculated from the SAR system parameter values. Since  $k_{\text{freq}}$  in Equation (28) refers to the LFM signal in the SAR system, its value is seen to change linearly. Therefore, the  $K_u$  value calculated with a linearly increasing LFM signal will progressively increase. Consequently, using this, the process of the calculation can be shown in Figure 10b. Since  $K_u(a, r)$  is consistently less than or equal to  $K_u(a, r+1)$ , it becomes a condition that cannot be satisfied even if we continue to increase the  $r$  value. Therefore, the amount of computation becomes  $O(N)$ , which is significantly faster than  $O(MN)$ .

The interpolation unit design comprises the  $K_u$  generator to calculate the  $K_u$  value, the  $K_{ui}$  generator to generate the  $K_{ui}$  value, the comparator to assess whether the conditions are met to perform interpolation, and the interpolation unit to perform the interpolation operation, as shown in Figure 11. The  $K_u$  generator and  $K_{ui}$  generator take the SAR system parameter values to generate the  $K_{ui}$  and  $K_u$  values, the comparator compares the  $K_{ui}$  and  $K_u$  values. If the appropriate conditions are satisfied, the interpolation unit performs the interpolation. The proposed interpolation unit is designed using a floating-point number system. This has the advantage of a higher signal-to-quantization noise ratio (SQNR) because it generally exhibits superior precision than using a fixed-point counting system but it introduces more complexity. To resolve this and improve precision, the FP32 number system is used for the  $K_u$  and  $K_{ui}$  values and FP16 is used for phase history data to reduce complexity. In Equation (27),  $\frac{K_{ui} - K_u(a,r)}{(K_u(a,r+1) - K_u(a,r))}$  is first calculated with FP32 and then converted to FP16 to perform the interpolation operation to reduce the complexity.

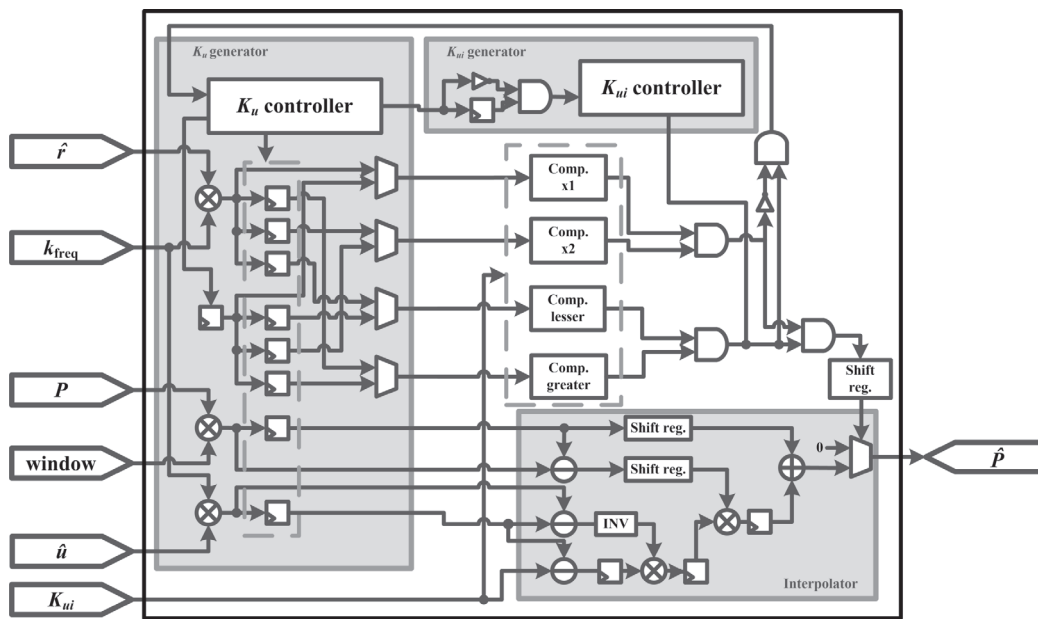


Figure 11. Hardware architecture of the proposed FP16 interpolation unit.

#### 4. Implementation and Acceleration Results

The proposed PFA-based SAR processor consists of an interpolation unit for interpolation and an SA-FFT unit for compression in the range or azimuth direction, as shown in Figure 12. Each unit is designed as a reusable block (IP core) to support the reconfigurability and is interconnected using the AXI4 bus. The IP consists of a master interface and a slave interface to communicate with the DDR memory controller and microprocessor, the registers to change the mode of each device, and RAM to temporarily store input and output data. The master interface is connected to the 128-bit AXI4 bus, which allows four 32-bit bits of data to be sent and received per cycle, enabling efficient parallel processing. In addition, the AXI4 bus-based design minimizes the use of SRAM inside the IP by exchanging data with the DDR memory, which means that the VLSI implementation can be employed with a smaller footprint and lower cost while reducing power usage, all being advantageous for drone-mounted operations.

The proposed PFA-based SAR processor was implemented using Verilog HDL on the Xilinx Zynq UltraScale+ FPGA platform. As shown in Table 1, the SA-FFT unit using the FP16 number system was implemented with 99,610 CLB LUTs, 21,921 CLB registers, 78 DSPs, and 12 Block RAMs, and the interpolation unit was implemented with 5722 CLB LUTs, 2306 CLB registers, and 17 DSPs. The design has a maximum operating frequency of 150 MHz and measured power consumption of 3.677 W. Figure 13 shows the verification environment of the FPGA platform.



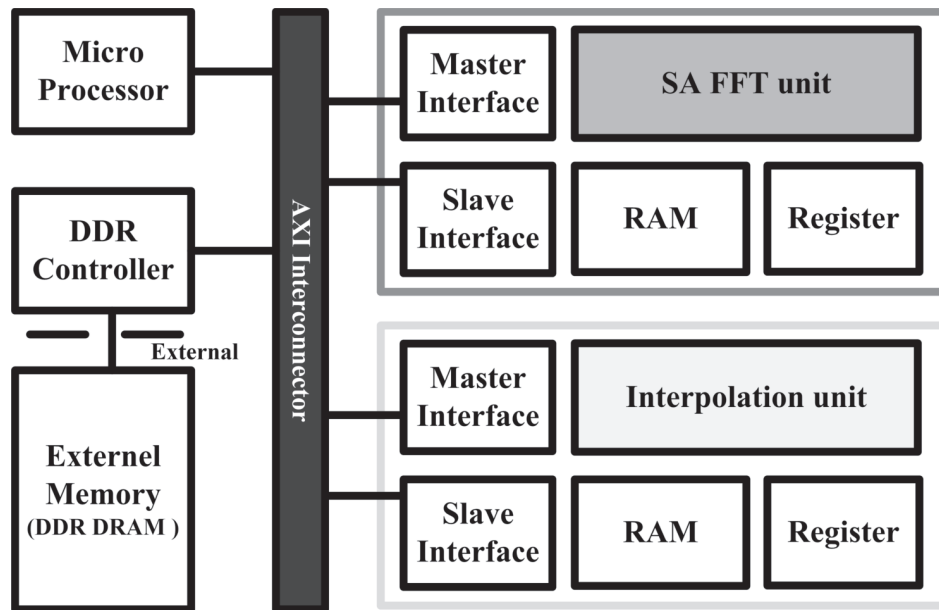


Figure 12. FPGA platform for verifying the proposed PFA-based SAR processor.

Table 1. Implementation results based on the Xilinx Zynq Ultrascale+ FPGA device.

Unit	# CLB LUTs	# CLB Registers	# DSPs	# Block RAMs	Max. Op. Freq.
SA-FFT	99,610	21,921	78	12	150 MHz
interpolation	5722	2306	17	-	150 MHz

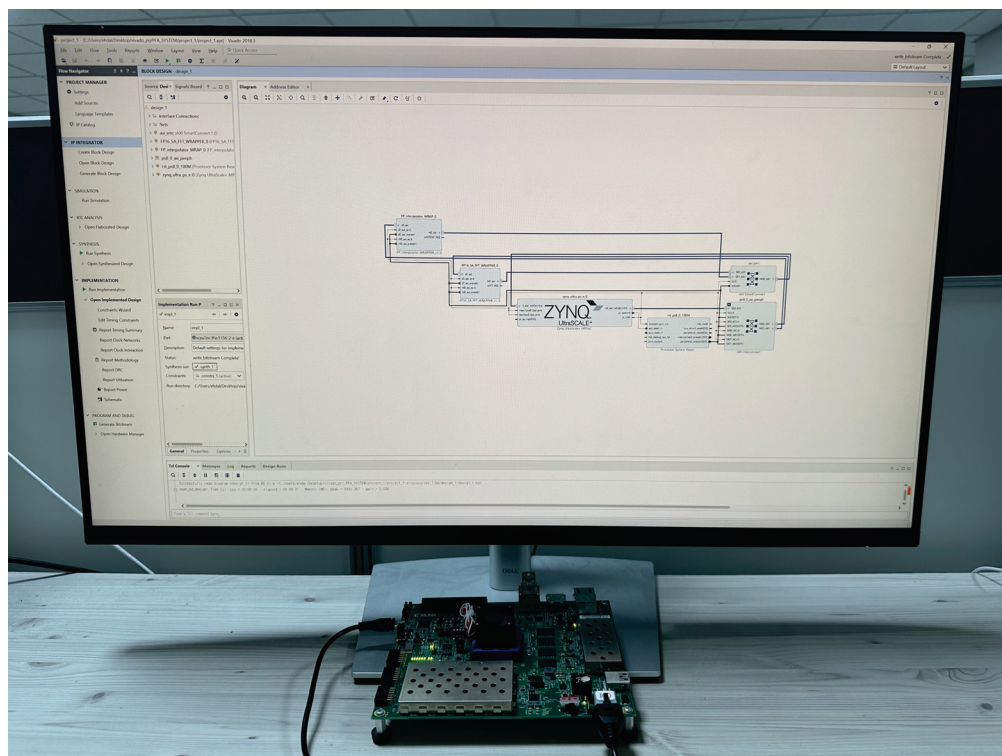


Figure 13. Verification environment for the proposed FPGA implementation.

To evaluate the proposed PFA-based SAR processor's performance, we compared the execution times of various suboperations with the software implementation based on an

ARM Cortex-A53 using the Gotcha and Sandia datasets. The FFT process was accelerated by the SA-FFT unit and the interpolation process was accelerated by the interpolation unit to measure the time. For the  $512 \times 512$  image on the Gotcha dataset, the image generation time decreased from 1.732 s to 0.046 s, achieving a 37.39 times speed increase. Similarly, for the  $2048 \times 2048$  image on the Sandia dataset, the time decreased from 34.364 s to 0.766 s, achieving a 44.862 times speed increase. The summarized results are presented in Table 2:

**Table 2.** PFA execution time.

Image Size	Execution Time (s)				Speedup Ratio (SW vs. HW)
	Full SW	Interp. Accel.	FFT Accel.	Full HW	
$512 \times 512$	1.732	1.654	0.124	0.046	37.393
$2048 \times 2048$	34.364	32.654	2.476	0.766	44.862

We compared the quality of the images generated by the ARM Cortex-A53-based software with those generated by the proposed PFA-based SAR processor. There are three metrics that measure the performance of an image: the SQNR, the peak signal-to-noise ratio (PSNR), and the structural similarity index map (SSIM). The SQNR is defined as follows:

$$\text{SQNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} \quad (29)$$

where  $P_{\text{signal}}$  represents the mean of the squares of the images before quantization and  $P_{\text{noise}}$  represents the mean of the squares of the difference between the image before quantization and the image after quantization. Then, the PSNR is defined as follows:

$$\text{PSNR} = 10 \log_{10} \left( \frac{\text{MAX}_I^2}{\text{MSE}} \right) \quad (30)$$

where  $\text{MAX}_I$  represents the maximum possible pixel value of the image and  $\text{MSE}$  represents the mean square error, which is the difference between the original image and the image generated by the SAR processor. Lastly, the SSIM is defined as follows:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (31)$$

where  $\mu_x$  and  $\mu_y$  represent the mean values of the images  $x$  and  $y$ ,  $\sigma_x^2$  and  $\sigma_y^2$  represent the variances of the images  $x$  and  $y$ ,  $\sigma_{xy}$  represents the covariance between images  $x$  and  $y$ , and  $C_1$  and  $C_2$  represent small constants for stabilization.

We measured the SQNR, PSNR, and SSIM. For the 'Gotcha image', the SQNR was 50.31 dB, the PSNR was 61.75 dB, and the SSIM was 0.9941, and for the 'Sandia image' the SQNR was 51.71 dB, the PSNR was 65.96 dB, and the SSIM was 0.9986. Since the proposed processor uses a floating-point number system to generate the images, they exhibit high similarity to the software-generated images as seen in Figure 14.

Table 3 compares the execution times of the proposed PFA-based SAR processor with the previous studies presented in references [52–55]. Due to the different image sizes presented in each study, we present the execution time normalized by the image size for comparison:

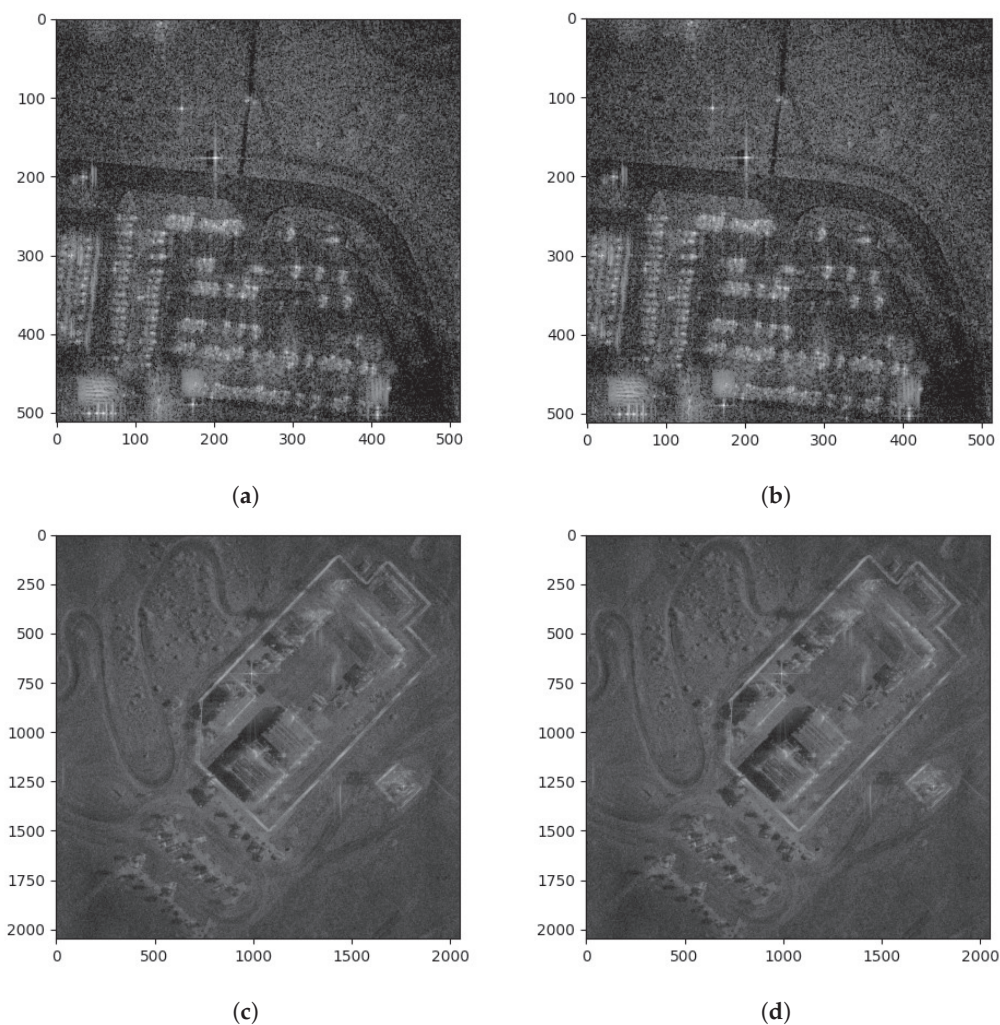
$$T_{\text{norm}} = \frac{\text{Execution Time}}{\text{Image Size}} \quad (32)$$

Although [52] achieved a higher speed than the proposed design, its power consumption of 247 W was very high and it used a large Tesla C2075 GPU, which is not suitable for drone SAR applications. Compared to [53], the proposed design occupies a smaller area and achieves a higher speed. Refs. [54,55] exhibited higher speeds than the proposed

design. However, in [54], 1.33 times the number of CLB LUTs, 8.1 times the number of CLB registers, 8.54 times the number of DSPs, and 31.25 times the block RAM were used, and in [55] about 2.74 times the number of CLB LUTs, 14.88 times the number of CLB registers, 12.58 times the number of DSPs, and 48.25 times the block RAM were used. To measure the execution time normalized for resource usage compared to the proposed design, we define the following metrics:

$$T_X = T_{\text{norm}} \cdot \frac{X}{Y} \quad (33)$$

where  $X$  represents the number of CLB LUTs, CLB Registers, DSPs, or block RAM in the design to be compared and  $Y$  represents the number of CLB LUTs, CLB Registers, DSPs, or block RAM in the proposed design. If  $X$  is greater than  $Y$ , the value of  $T_X$  will be greater because it indicates that the resources of the compared design are more heavily utilized than the resources of the proposed design. A smaller  $T_X$  value shows how fast the design is relative to its usage.



**Figure 14.** Generated SAR images: (a) Cortex-A53-based Gotcha image; (b) proposed processor-based Gotcha image; (c) Cortex-A53-based Sandia image; (d) proposed processor-based Sandia image.

This metric was used to compare the proposed design. Regarding the CLB LUTs usage, the proposed design is slightly slower, but regarding other resources and SRAM, the proposed design achieves higher speeds per unit area. As a result, compared to previous studies, the proposed PFA-based SAR processor achieves the fastest image generation time regarding power, area, and resource usage. Therefore, the proposed design is advantageous for small drone ViSAR applications that require less weight, space, and high speed. The proposed design generates one image frame in 0.046 s using  $512 \times 512$  single-precision

floating-point complex data, which means that it can generate images at up to 21.74 Hz. Assuming the PRF is 250 Hz and the number of pulses is 352, and substituting it into Equation (5), the pulses can be overlapped by 96.7%, which means that the proposed architecture can achieve real-time imaging with high frame rates.

**Table 3.** Comparison with previous implementation.

	[52]	[53]	[54]	[55]	Proposed
Platform	GPU	FPGA	FPGA	FPGA	FPGA
Operating Freq.	1.15 GHz	200 MHz	200 MHz	200 MHz	150 MHz
Power (W)	247	-	-	-	3.677
Image Size	4096 × 8192	4096 × 2048	4096 × 4096	2048 × 2048	512 × 512 2048 × 2048
Exec. Time (s)	2.16	2.1	1	0.18	0.046 0.766
$T_{\text{norm}}$ (ns)	64.373	250.339	59.604	42.915	175.476 182.629
throughput (MB/s)	62.14	15.98	67.11	93.21	22.79 21.91
# CLB LUTs	-	247,906	139,586	289,075	105,332
# CLB Registers	-	433,200	196,258	360,486	24,227
# DSPs	-	1093	811	1195	95
# Block RAMs	-	1056	375	579	12
$T_{\text{CLB LUTs}}$ (ns)	-	589.189	78.987	117.776	175.476
$T_{\text{CLB Registers}}$ (ns)	-	4476.281	482.839	638.554	
$T_{\text{DSPs}}$ (ns)	-	2880.216	508.829	539.826	
$T_{\text{Block RAMs}}$ (ns)	-	22,029.832	1862.625	2070.649	

## 5. Conclusions

In this study, we propose a design for the architecture of a PFA-based SAR processor consisting of an SA-FFT unit and an interpolation unit. The SA-FFT unit is designed as a systolic array structure to achieve high computation speed, and the interpolation unit uses a linear interpolation algorithm with a small area and high speed and is optimized for the PFA. Each unit is designed as a reusable block (IP core) to support reconfigurability and is interconnected using the AXI4 bus. In addition, since SRAM requires more power, a larger area, and is more expensive than DRAM, using less SRAM is desirable for VLSI implementations. Therefore, transferring data to and from the DDR memory via the AXI4 bus minimizes the use of SRAM within the IP. The proposed design was implemented on the Xilinx Zynq UltraScale+ FPGA platform. The SA-FFT unit was implemented with 99,610 CLB LUTs, 21,921 CLB registers, 78 DSPs, and 12 Block RAMs, and the interpolation unit was implemented with 5722 CLB LUTs, 2306 CLB registers, and 17 DSPs. For comparison, the execution time of the ARM Cortex-A53-based software was measured for an image of 2048 × 2048 pixels, and the proposed model achieved a 44.862 times acceleration. By normalizing the execution time by the number of pixels and comparing the speed-to-area ratio by normalizing the number of resources for each, we achieved higher speed with less power consumption compared to previous studies. Therefore, the proposed design is more suitable for small drone ViSAR applications.

In future research, we will implement a processor that can support various algorithms, such as the RDA, the CSA, and the BPA, simultaneously, employing the AXI4 bus-based design. This would also include the implementation of an ASIC that can be used on a small SAR platform based on the validated design with an FPGA. This model is expected to be more power-efficient and to be usable in multiple applications to run various algorithms.



**Author Contributions:** D.J. designed the PFA-based SAR processor, performed the experiment and evaluation, and wrote the paper. M.L. and W.L. implemented the processor and performed the revision of this manuscript. Y.J. conceived of and led the research, analyzed the experimental results, and wrote the paper. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors gratefully acknowledge the support of the Next-Generation SAR Research Laboratory at Korea Aerospace University, originally funded by the Defense Acquisition Program Administration (DAPA) and the Agency for Defense Development (ADD).

**Data Availability Statement:** All data underlying the results are available as part of the article and no additional source data are required.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Brown, W.M.; Porcello, L.J. An introduction to synthetic-aperture radar. *IEEE Spectr.* **1969**, *6*, 52–62. [CrossRef]
2. Munson, D.C.; Visentin, R.L. A signal processing view of strip-mapping synthetic aperture radar. *IEEE Trans. Acoust. Speech Signal Process.* **1989**, *37*, 2131–2147. [CrossRef]
3. Moreira, A.; Prats-Iraola, P.; Younis, M.; Krieger, G.; Hajnsek, I.; Papathanassiou, K.P. A tutorial on synthetic aperture radar. *IEEE Geosci. Remote Sens. Mag.* **2013**, *1*, 6–43. [CrossRef]
4. Lou, Y.; Clark, D.; Marks, P.; Muellerschoen, R.J.; Wang, C.C. Onboard radar processor development for rapid response to natural hazards. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 2770–2776. [CrossRef]
5. Percivall, G.S.; Alameh, N.S.; Caumont, H.; Moe, K.L.; Evans, J.D. Improving disaster management using earth observations—GEOSS and CEOS activities. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 1368–1375. [CrossRef]
6. Tralli, D.M.; Blom, R.G.; Zlotnicki, V.; Donnellan, A.; Evans, D.L. Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards. *ISPRS J. Photogramm. Remote Sens.* **2005**, *59*, 185–198. [CrossRef]
7. Sharma, R.K.; Kumar, B.S.; Desai, N.M.; Gujrati, V. SAR for disaster management. *IEEE Aerosp. Electron. Syst. Mag.* **2008**, *23*, 4–9. [CrossRef]
8. Yang, X.; Shi, J.; Zhou, Y.; Wang, C.; Hu, Y.; Zhang, X.; Wei, S. Ground moving target tracking and refocusing using shadow in video-SAR. *Remote Sens.* **2020**, *12*, 3083. [CrossRef]
9. Guo, P.; Wu, F.; Tang, S.; Jiang, C.; Liu, C. Implementation Method of Automotive Video SAR (ViSAR) Based on Sub-Aperture Spectrum Fusion. *Remote Sens.* **2023**, *15*, 476. [CrossRef]
10. Kim, C.K.; Azim, M.T.; Singh, A.K.; Park, S.O. Doppler shifting technique for generating multi-frames of video SAR via sub-aperture signal processing. *IEEE Trans. Signal Process.* **2020**, *68*, 3990–4001. [CrossRef]
11. Yang, C.; Chen, Z.; Deng, Y.; Wang, W.; Wang, P.; Zhao, F. Generation of Multiple Frames for High Resolution Video SAR Based on Time Frequency Sub-Aperture Technique. *Remote Sens.* **2023**, *15*, 264. [CrossRef]
12. Cheng, Y.; Ding, J.; Sun, Z.; Zhong, C. Processing of airborne video SAR data using the modified back projection algorithm. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 1–13. [CrossRef]
13. Bimber, O.; Kurmi, I.; Schedl, D.C. Synthetic aperture imaging with drones. *IEEE Comput. Graph. Appl.* **2019**, *39*, 8–15. [CrossRef]
14. Liu, B.; Wang, K.; Liu, X.; Yu, W. An efficient SAR processor based on GPU via CUDA. In Proceedings of the 2009 2nd International Congress on Image and Signal Processing, Tianjin, China, 17–19 October 2009; pp. 1–5.
15. Tang, H.; Li, G.; Zhang, F.; Hu, W.; Li, W. A spaceborne SAR on-board processing simulator using mobile GPU. In Proceedings of the 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Beijing, China, 10–15 July 2016; pp. 1198–1201.
16. Wielage, M.; Cholewa, F.; Fahnemann, C.; Pirsch, P.; Blume, H. High performance and low power architectures: GPU vs. FPGA for fast factorized backprojection. In Proceedings of the 2017 Fifth International Symposium on Computing and Networking (CANDAR), Aomori, Japan, 19–22 November 2017; pp. 351–357.
17. Wang, S.; Zhang, S.; Huang, X.; An, J.; Chang, L. A highly efficient heterogeneous processor for SAR imaging. *Sensors* **2019**, *19*, 3409. [CrossRef]
18. Hartley, T.D.; Fasih, A.R.; Berdanier, C.A.; Ozguner, F.; Catalyurek, U.V. Investigating the use of GPU-accelerated nodes for SAR image formation. In Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–8.
19. Ning, X.; Yeh, C.; Zhou, B.; Gao, W.; Yang, J. Multiple-GPU accelerated range-Doppler algorithm for synthetic aperture radar imaging. In Proceedings of the 2011 IEEE RadarCon (RADAR), Kansas City, MO, USA, 23–27 May 2011; pp. 698–701.
20. Le, C.; Chan, S.; Cheng, F.; Fang, W.; Fischman, M.; Hensley, S.; Johnson, R.; Jourdan, M.; Marina, M.; Parham, B.; et al. Onboard FPGA-based SAR processing for future spaceborne systems. In Proceedings of the 2004 IEEE Radar Conference (IEEE Cat. No. 04CH37509), Philadelphia, PA, USA, 29 April 2004; pp. 15–20.
21. Wiehle, S.; Mandapati, S.; Günzel, D.; Breit, H.; Balss, U. Synthetic aperture radar image formation and processing on an MPSoC. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 1–14. [CrossRef]

22. Xie, Y.; Zhong, Z.; Li, B.; Xie, Y.; Chen, L.; Chen, H. An ARM-FPGA Hybrid Acceleration and Fault Tolerant Technique for Phase Factor Calculation in Spaceborne Synthetic Aperture Radar Imaging. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2024**, *17*, 5059–5072. [CrossRef]
23. Wang, J.; Feng, D.; Xu, Z.; Wu, Q.; Hu, W. Time-domain digital-coding active frequency selective surface absorber/reflector and its imaging characteristics. *IEEE Trans. Antennas Propag.* **2020**, *69*, 3322–3331. [CrossRef]
24. Zhou, X.; Yu, Z.J.; Cao, Y.; Jiang, S. SAR imaging realization with FPGA based on VIVADO HLS. In Proceedings of the 2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP), Chongqing, China, 11–13 December 2019; pp. 1–4.
25. Milton, M.; Benigni, A.; Monti, A. Real-time multi-FPGA simulation of energy conversion systems. *IEEE Trans. Energy Convers.* **2019**, *34*, 2198–2208. [CrossRef]
26. Waidyasooriya, H.M.; Hariyama, M. Multi-FPGA accelerator architecture for stencil computation exploiting spacial and temporal scalability. *IEEE Access* **2019**, *7*, 53188–53201. [CrossRef]
27. Brown, W.M.; Fredricks, R.J. Range-Doppler imaging with motion through resolution cells. *IEEE Trans. Aerosp. Electron. Syst.* **1969**, *AES-5*, 98–102. [CrossRef]
28. Sun, J.; Mao, S.; Wang, G.; Hong, W. Polar format algorithm for spotlight bistatic SAR with arbitrary geometry configuration. *Prog. Electromagn. Res.* **2010**, *103*, 323–338. [CrossRef]
29. Yegulalp, A.F. Fast backprojection algorithm for synthetic aperture radar. In Proceedings of the 1999 IEEE Radar Conference. Radar into the Next Millennium (Cat. No. 99CH36249), Waltham, MA, USA, 22 April 1999; pp. 60–65.
30. Raney, R.K.; Runge, H.; Bamler, R.; Cumming, I.G.; Wong, F.H. Precision SAR processing using chirp scaling. *IEEE Trans. Geosci. Remote Sens.* **1994**, *32*, 786–799. [CrossRef]
31. Shin, H.S.; Lim, J.T. Omega-K algorithm for spaceborne spotlight SAR imaging. *IEEE Geosci. Remote Sens. Lett.* **2011**, *9*, 343–347. [CrossRef]
32. Desai, M.D.; Jenkins, W.K. Convolution backprojection image reconstruction for spotlight mode synthetic aperture radar. *IEEE Trans. Image Process.* **1992**, *1*, 505–517. [CrossRef]
33. Zhang, B.; Xu, G.; Zhou, R.; Zhang, H.; Hong, W. Multi-channel back-projection algorithm for mmwave automotive MIMO SAR imaging with Doppler-division multiplexing. *IEEE J. Sel. Top. Signal Process.* **2022**, *17*, 445–457. [CrossRef]
34. Walker, J.L. Range-Doppler imaging of rotating objects. *IEEE Trans. Aerosp. Electron. Syst.* **1980**, *AES-16*, 23–52. [CrossRef]
35. Yuan, Y.; Sun, J.; Mao, S. PFA algorithm for airborne spotlight SAR imaging with nonideal motions. *IEE Proc.-Radar Sonar Navig.* **2002**, *149*, 174–182. [CrossRef]
36. Jiang, J.; Li, Y.; Zheng, Q. A THz Video SAR Imaging Algorithm Based on Chirp Scaling. In Proceedings of the 2021 CIE International Conference on Radar (Radar), Haikou, China, 15–19 December 2021; pp. 656–660.
37. Rigling, B.D.; Moses, R.L. Polar format algorithm for bistatic SAR. *IEEE Trans. Aerosp. Electron. Syst.* **2004**, *40*, 1147–1159. [CrossRef]
38. Baas, B.M. A 9.5 mW 330 sec 1024-point FFT Processor. In Proceedings of the Custom Integrated Circuits Conference, Santa Clara, CA, USA, 11–14 May 1998; pp. 11–14.
39. He, S.; Torkelson, M. Design and implementation of a 1024-point pipeline FFT processor. In Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No. 98CH36143), Santa Clara, CA, USA, 14 May 1998; pp. 131–134.
40. Lee, M.K.; Shin, K.W.; Lee, J.K. A VLSI array processor for 16-point FFT. *IEEE J.-Solid-State Circuits* **1991**, *26*, 1286–1292. [CrossRef]
41. Kung, H.T. Why systolic architectures? *Computer* **1982**, *15*, 37–46. [CrossRef]
42. Kung, S.Y. VLSI array processors. *IEEE ASSP Mag.* **1985**, *2*, 4–22. [CrossRef]
43. Chan, L.W.; Chen, M.Y. A new systolic array for discrete Fourier transform. *IEEE Trans. Acoust. Speech Signal Process.* **1988**, *36*, 1665–1666. [CrossRef]
44. Wang, C.L.; Chang, Y.T. Efficient 2-D systolic array implementation of a prime factor DFT algorithm. In Proceedings of the TENCON'92-Technology Enabling Tomorrow, Notre Dame, IN, USA, 4–5 March 1992; pp. 56–60.
45. Lee, M.H. High speed multidimensional systolic arrays for discrete Fourier transform. *IEEE Trans. Circuits Syst. II Analog. Digit. Signal Process.* **1992**, *39*, 876–879. [CrossRef]
46. Lim, H.; Swartzlander, E.E. Multidimensional systolic arrays for the implementation of discrete Fourier transforms. *IEEE Trans. Signal Process.* **1999**, *47*, 1359–1370.
47. Meher, P.K. Efficient systolic implementation of DFT using a low-complexity convolution-like formulation. *IEEE Trans. Circuits Syst. II Express Briefs* **2006**, *53*, 702–706. [CrossRef]
48. Nash, J.G. Computationally efficient systolic architecture for computing the discrete Fourier transform. *IEEE Trans. Signal Process.* **2005**, *53*, 4640–4651. [CrossRef]
49. Zhao, S.; Chen, J.; Yang, W.; Sun, B.; Wang, Y. Image formation method for spaceborne video SAR. In Proceedings of the 2015 IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR), Singapore, 1–4 September 2015; pp. 148–151.
50. Liu, B.; Zhang, X.; Tang, K.; Liu, M.; Liu, L. Spaceborne video-SAR moving target surveillance system. In Proceedings of the 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Beijing, China, 10–15 July 2016; pp. 2348–2351.
51. Khosravi, M.R.; Samadi, S. Frame rate computing and aggregation measurement toward QoS/QoE in Video-SAR systems for UAV-borne real-time remote sensing. *J. Supercomput.* **2021**, *77*, 14565–14582. [CrossRef]
52. Xu, Z.; Zhu, D. High-resolution miniature UAV SAR imaging based on GPU architecture. *J. Phys. Conf. Ser.* **2018**, *1074*, 012122. [CrossRef]



53. Liu, R.; Zhu, D.; Wang, D.; Du, W. FPGA implementation of SAR imaging processing system. In Proceedings of the 2019 6th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR), Xiamen, China, 26–29 November 2019; pp. 1–5.
54. Linchen, Z.; Jindong, Z.; Daiyin, Z. FPGA implementation of polar format algorithm for airborne spotlight SAR processing. In Proceedings of the 2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, Chengdu, China, 21–22 December 2013; pp. 143–147.
55. Wang, D.; Zhu, D.; Liu, R. Video SAR high-speed processing technology based on FPGA. In Proceedings of the 2019 IEEE MTT-S International Microwave Biomedical Conference (IMBioC), Nanjing, China, 6–8 May 2019; Volume 1, pp. 1–4.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# Lightweight and Error-Tolerant Stereo Matching with a Stochastic Computing Processor

Seongmo An, Jongwon Oh, Sangho Lee, Jinyeol Kim, Youngwoo Jeong, Jeongeun Kim  
and Seung Eun Lee \*

Department of Electronic Engineering, Seoul National University of Science and Technology,  
Seoul 01811, Republic of Korea; ahnseongmo@seoultech.ac.kr (S.A.); ohjongwon@seoultech.ac.kr (J.O.);  
leesangho@seoultech.ac.kr (S.L.); kimjinyeol@seoultech.ac.kr (J.K.); jeongyoungwoo@seoultech.ac.kr (Y.J.);  
kimjeongeun@seoultech.ac.kr (J.K.)

\* Correspondence: seung.lee@seoultech.ac.kr; Tel.: +82-2-970-9021

**Abstract:** Stereo matching, utilized in diverse fields, poses a challenge to systems in resource-constrained environments due to the significant growth of computational load with image resolution. The challenge is crucial for the systems because fields utilizing stereo matching require short operational time for real-time applications and low power architecture. Stochastic computing (SC) is able to be a valuable approach to address the challenge by reducing the computational load by representing binary numbers with stochastic sequences, which are encoded as a probability value, and by leveraging the concept of mathematical probability. Also, it is possible for a system to be error-tolerant by utilizing the characteristics of stochastic computing. Therefore, in this paper, we propose an approach for lightweight and error-tolerant stereo matching with a hardware-implemented stochastic computing processor. To verify the feasibility and error tolerance of the proposed system, we implemented the proposed system and conducted experiments comparing depth maps with or without stochastic computing by calculating similarities. According to the experimental results, the proposed system indicated no significant differences in output depth maps and achieved an improvement in the depth maps from error-injected input images by an average of 58.95%. Therefore, we demonstrated that stereo matching with stochastic computing is feasible and error-tolerant.

**Keywords:** stochastic computing (SC); stereo matching; error tolerance; processor

## 1. Introduction

Stereo matching, a computer vision technique that estimates 3D structures from 2D images, has been widely employed in various fields such as autonomous driving and augmented reality (AR) to generate depth maps of environments [1]. The primary goal of stereo matching is to determine disparities, which represent the displacements between corresponding features in each pair of stereo images, as illustrated in Figure 1 [2]. However, the process of determining disparities involves comparing all pixel values within a search window against those in a template block, leading to an increase in computational complexity with the rise in image resolution. This presents significant challenges for systems operating in resource-limited environments [3].

Recognizing the limitations of current stereo matching methods, this paper proposes a novel approach to address these challenges with SC. SC is a computing paradigm that employs stochastic sequences to represent and process data. A stochastic sequence is a bit stream of '0's and '1's obtained by comparing a binary number with random numbers, as shown in Figure 2. By approximating a binary number into a probability value between '0' and '1' and leveraging the concept of mathematical probability [4], SC replaces traditional arithmetic units with concise logic gates [5]. This results in outputs with relatively low accuracy but high error tolerance [6], making SC a promising solution for computationally intensive applications such as image processing and artificial intelligence [7–9].

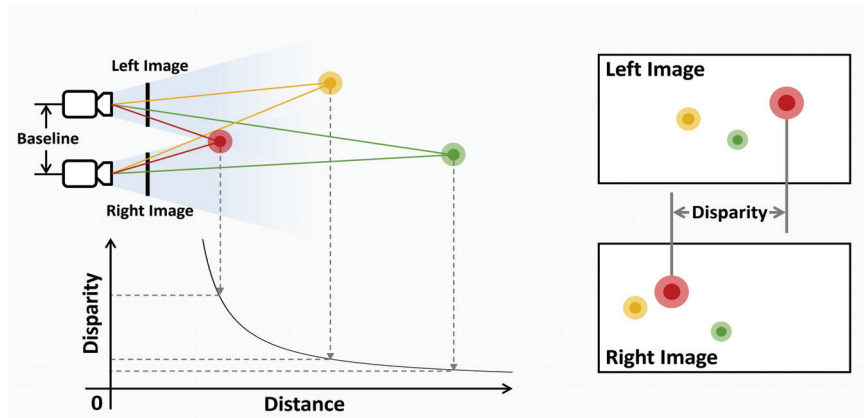


Figure 1. An overview of how to obtain the disparity.

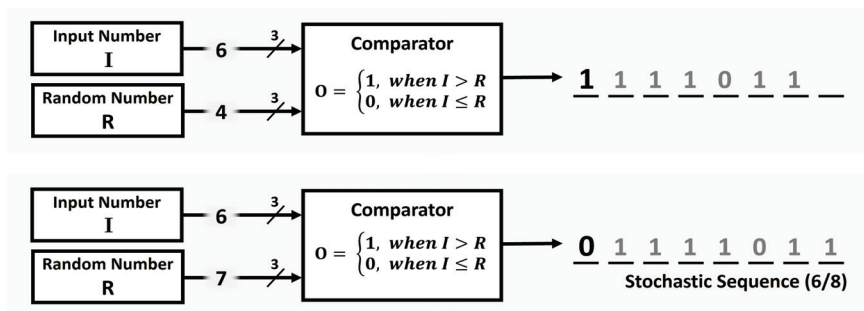


Figure 2. The extraction process of the stochastic sequence.

In this paper, we introduce a stereo matching system that includes a hardware-implemented SC unit. The SC unit, featuring a specially designed parallel linear feedback shift register (LFSR), addresses the latency issue of the random number generator and offers benefits in terms of area efficiency and computation time. We connected the SC unit and the ARM-based core via a bus, implemented the SC processor on an FPGA, and performed stereo matching algorithm operations through the SC unit to obtain a depth map.

Our proposed system not only addresses the computational challenges of stereo matching but also improves the error tolerance of the process. We validated the feasibility of our system by comparing the depth map obtained with the SC unit to that obtained without it. Furthermore, we verified the error tolerance of our system by obtaining depth maps from input images with injected errors. Remarkably, when utilizing the SC unit, the output depth map for the input images containing errors improved by an average of 58.95% compared to when not utilizing the SC unit. The contributions of this work are as follows:

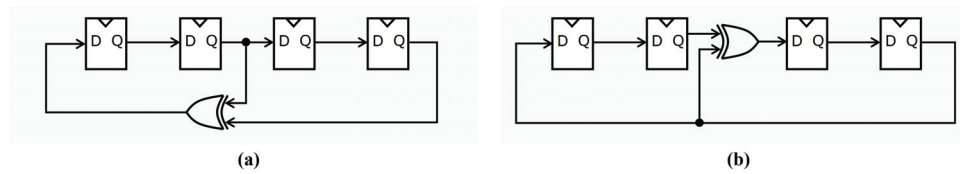
- The proposed system shows that there is no significant difference between the depth map obtained with the SC unit and the one obtained without the SC unit.
- The proposed system demonstrates better tolerance to errors when the SC unit is utilized.
- The architecture of the LFSR included in the proposed system is parallelized to achieve area efficiency.

This paper, therefore, presents a significant advancement in stereo matching techniques, offering a solution that is both computationally efficient and error-tolerant. This paper consists of the following. Section 2 outlines the background on the SC and stereo matching algorithm employed in this work. Section 3 introduces related works about the systems utilizing SC. Section 4 demonstrates the proposed overall system architecture and the detailed aspects of the SC unit. Section 5 presents the implementation of the proposed system and shows the experimental results. Lastly, Section 6 concludes this paper.

## 2. Background

### 2.1. Stochastic Computing

The implementation of SC requires three steps: stochastic number generation, stochastic computing operation, and binary conversion [6]. In stochastic number generation, binary numbers are converted into stochastic sequences with LFSRs and comparators. The LFSR generates pseudo-random numbers by shifting bits and applying a feedback function, which is a linear combination of certain bits in the register. As shown in Figure 3, an LFSR forms a feedback loop consisting of a shift register and an XOR. By utilizing the structure of the LFSR, a pseudo-random number is generated. The stochastic computing operation is used to compute operations with the stochastic sequence, and the binary conversion converts the sequence back into a binary number.

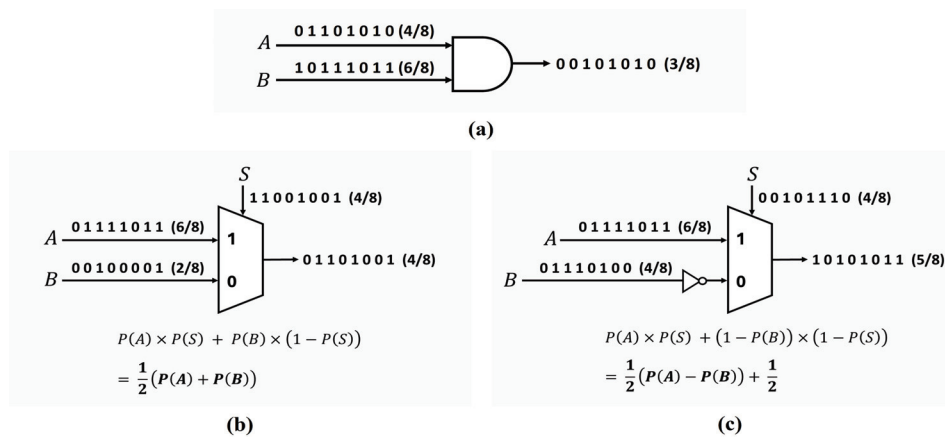


**Figure 3.** Circuits of Fibonacci LFSR and Galois LFSR: (a) 4-bit Fibonacci LFSR, (b) 4-bit Galois LFSR.

#### 2.1.1. Area Efficiency

With SC, the traditional arithmetic operations are replaced with brief logic gates. For example, a common binary multiplier is replaced by an AND gate, and a common binary adder is replaced by a multiplexer (MUX), which is possible because of the application of the concept of mathematical probability. Since the stochastic sequence refers to a probability value, independently generated stochastic sequences represent independent probability values [10].

Figure 4 demonstrates the stochastic computing operations. The probability that two independent events occur at the same time is mathematically the multiplication of the probabilities. Therefore, if two independent stochastic sequences are the inputs of an AND gate, then the output of the gate is equal to the product of the probabilities. Similarly, the probability that an event  $A$  or  $B$  occurs is the sum of their respective probability values. Therefore, if the inputs of the MUX are independent stochastic sequences, the output is related to the probability of the MUX select signal  $S$  being '1' or being '0'. When  $S$  is '1', the output of the MUX is the probability of the  $S$  and the  $A$  to occur simultaneously,  $P(A) \times P(S)$ . When  $S$  is '0', the output is  $P(B) \times (1 - P(S))$ . Consequently, the final output of the MUX is  $P(A) \times P(S) + P(B) \times (1 - P(S))$ . If the probability of  $S$  is  $4/8$ , the output of the MUX is equal to the sum of the probabilities scaled to  $1/2$ . The subtraction operation is also possible by adding an NOT gate, as shown in Figure 4c [11].



**Figure 4.** Example of stochastic computing operations: (a) stochastic multiplication, (b) stochastic addition, (c) stochastic subtraction.

### 2.1.2. Error Tolerance

Because each digit in a stochastic sequence has the same weight, errors occurring in the sequence have a relatively small influence. For example, if an error occurs in one part of the 256-bit stochastic sequence representing an 8-bit binary number, the difference in the result is only  $1/256$  [12]. Figure 5 demonstrates a case of bit flip occurring in a stochastic sequence and the comparison of the results. As shown in Figure 5, even if an error occurs in any bit of a stochastic sequence, the impact on the output is relatively small compared to that of a binary number, in which errors of higher-weighted bits have a significant impact [13].



**Figure 5.** Comparison of ideal case and error-occurred case of stochastic multiplication.

### 2.2. Stereo Matching

To extract disparities from stereo images, corresponding features between both images are to be detected. One technique for detecting corresponding features is template matching, which involves searching for the most correlated block in an image with a template block in another image [14]. The stereo images have two characteristics: the cameras are horizontally aligned, and searches for correlated blocks are conducted in a single image. As a result, the correlated block exists on the same horizontal line as the template block and has a position within a certain range of x-coordinates. In other words, a search window for the correlated block is determined based on the position of the template block. The disparity is decided by calculating the difference in the x-coordinates of the template block and the most correlated block in the search window [15].

As described in Figure 6, stereo matching consists of four steps. In the cost computation, the matching cost is decided according to the differences in values between the corresponding pixels. The cost aggregation refers to the aggregation of the matching costs to obtain the most correlated block within the search window. In the disparity computation, the disparity is obtained by calculating the distance of the coordinates between the template block and the correlated block selected through the cost aggregation. The disparity refinement aims to rectify the incorrect disparities obtained by stereo matching [16].



**Figure 6.** Composition of stereo matching.

#### 2.2.1. Cost Aggregation

After the cost computation, which calculates the matching cost (in this work, the difference in pixel values) of corresponding pixels in two blocks, cost aggregation proceeds. In the cost aggregation, block-based cost matching functions are commonly employed to aggregate costs [2]. One of the uncomplicated cost matching functions is the sum of absolute difference (SAD). When the blocks are identical, the result is '0'. Another function is the sum of squared difference (SSD), which requires multiplication operations resulting in higher computational complexity than the SAD [17]. Normalized cross correlation (NCC) is a method of normalizing the similarity between the two blocks, which is more computationally complicated than the SAD and the SSD as it involves multiplication, division, and square root operations [18]. Equations (1)–(3) describe mathematical expressions for the SAD, SSD, and NCC, respectively.

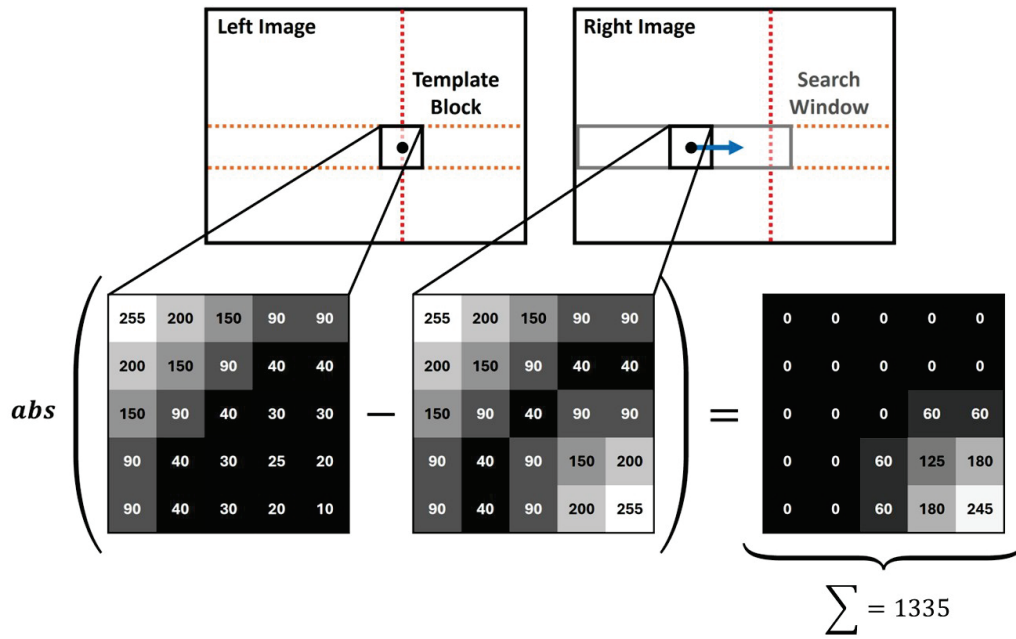
$$SAD = \sum_{j=1}^H \sum_{i=1}^W |a_{ij} - b_{ij}| \quad (1)$$

$$SSD = \sum_{j=1}^H \sum_{i=1}^W (a_{ij} - b_{ij})^2 \quad (2)$$

$$NCC = \sum_{j=1}^H \sum_{i=1}^W \frac{(a_{ij} - \bar{a})(b_{ij} - \bar{b})}{\sigma_a \sigma_b} \quad (3)$$

$$\text{(Where } \bar{a} = \frac{1}{N} \sum_{i,j} a_{ij}, \bar{b} = \frac{1}{N} \sum_{i,j} b_{ij}, \sigma_a = \sqrt{\frac{1}{N} \sum_{i,j} (a_{ij} - \bar{a})^2}, \sigma_b = \sqrt{\frac{1}{N} \sum_{i,j} (b_{ij} - \bar{b})^2} \text{)}$$

Among the three functions mentioned above, the SAD is the most suitable function for cost-effective architectures. This is because the SAD employs only the most uncomplicated operations [19]. Therefore, we utilized the SAD as the cost matching function to make it suitable for an architecture with SC. Figure 7 demonstrates an overview of performing the cost computation and aggregation employing the SAD.



**Figure 7.** Cost computation and aggregation with the SAD.

### 2.2.2. Disparity Computation

The process of disparity computation is a critical step in stereo matching. The disparity is obtained as the difference between a template block and the block having the highest correlation in the search window [20].

To begin with, we calculate and store the correlation values, i.e., SAD values, for each pair of blocks—the template block and all the blocks within the search window. Once we have the SAD values, we proceed to obtain the disparity. This is accomplished by calculating the difference between the x-coordinate of the template block and that of the block having the minimum result value from the SAD. The block with the minimum SAD value is considered to have the highest correlation with the template block, and thus, its x-coordinate is employed in the disparity calculation.



### 2.2.3. Disparity Refinement

After the disparity computation, the generated disparity map often contains noises such as invalid matches. These are disparities that do not accurately represent the depth information of the scene and are caused by various factors such as occlusions, featureless regions, or repetitive patterns [21].

To address this issue, we implemented methods such as filtering techniques. One common approach is to apply a median filter to the initial disparity maps. The median filter works by replacing each disparity value with the median of the disparities in its neighborhood. This has the effect of preserving the edges while removing isolated noises.

Applying a median filter not only helps in reducing the noise in the disparity map but it also minimizes the computational complexity. This is because the median filter operates in a local neighborhood and does not require knowledge of the entire image. Therefore, the median filter is able to be efficiently implemented even on large images [22].

## 3. Related Work

SC is a non-traditional computing technology that encodes information with finite-length stochastic sequences of '0's and '1's. The probabilistic elements enable the simplification of complex operations and resilience to errors, and various research projects utilizing these properties have been carried out [7,9,23–28]. Particularly, studies have been conducted to enhance the performance of SC and apply it to applications requiring robustness against noise or demanding low power consumption, such as artificial intelligence filtering operations [7,23,24] and image processing [9,25–28].

In [7], Extended Stochastic Logic (ESL) was applied as a method to solve the problem of low accuracy in utilizing stochastic computing for artificial neural networks (ANNs). The study replaced the accumulation process in ANN computing with an ESL-based adder and substituted the conventional activation function with an ESL-based ReLU. This approach resulted in a 48% improvement in accuracy compared to conventional SC-based methods, an 84% reduction in area compared to non-SC-based methods, and a 60% decrease in power consumption. In [23], a parallel SC-based neural network (NN) accelerator was proposed to enhance the fault tolerance. This yielded a  $2.8\times$  improvement in energy efficiency compared to traditional binary computing methods. The authors of [24] conducted research to reduce overhead convolutional neural networks (CNNs) utilizing SC by addressing high parallelism. Pseudo-Sobol sequences were proposed for SC-CNN, and an efficient parallel computation-conversion hybrid convolution architecture was developed, leading to improvements of 41% in energy efficiency and 36% in area.

In [9], non-scaling adders and subtractors were introduced which efficiently performed cascade computations compared to scaling adders. These were verified by applying them to an image sharpening filter. The accuracy of computations and the quality of the sharpened images were measured by peak signal-to-noise ratio and the structural similarity index measure. Ref. [25] proposed a simple method to improve the accuracy of SC by exchanging the wires used in operations and suggests adders and multipliers for SC. These were validated by applying adders and multipliers to the edge detection algorithm, resulting in a reduction in area utilization (64%) and power consumption (96%) compared to the accurate edge detection. In [26], a new technique called approximate stochastic computing (ASC) for improving computation time in image processing was proposed. The research verified its effect on computation time by an edge detection algorithm. Ref. [27] designed a fuzzy noise reduction filter based on SC. The experimental result showed a reduction in the hardware area and power consumption compared to conventional binary implementations. And the proposed design preserved the quality of the results. In [28], the research analyzed the stochastic absolute value function, stochastic tanh function, and one-parameter linear gain functions in SC. The study verified the validity of SC through four basic image processing algorithms: edge detection, frame difference-based image segmentation, median filter-based image enhancement, and image contrast stretching. The result showed SC has noise tolerance and consumes less hardware than the conventional methods.

Table 1 displays the related works regarding stochastic computing utilized for image processing.

**Table 1.** Analysis of related works.

Source	Proposed Approach	Application	Pros	Cons
Temenos, N. et al., 2022 [9]	Non-scaling stochastic computing adder and subtracter.	Sharpening filter.	More accuracy in cascaded computations. Fewer resources are required than in stochastic computing with MUX.	It is impossible to utilize the efficiency that is obtained by utilizing a scaled stochastic computing circuit.
Joe, H et al., 2019 [25]	Stochastic computing by the wire exchanging method.	Edge detection algorithm.	It improved the accuracy of stochastic computing.	The hardware circuit design may become complicated, which may increase the power consumption and area of the hardware.
R. Seva et al., 2016 [26]	Approximate stochastic computing focusing on image processing.	Edge detection algorithm.	It reduced the long run-time of stochastic computing.	Stochastic computing may not be suitable for applications requiring high accuracy, such as edge detection.
S. N. Estiri et al., 2022 [27]	Stochastic computing is applied to a fuzzy noise reduction filter.	Fuzzy noise reduction filter.	Saving in the hardware area and power costs compared to the conventional binary implementation while preserving the quality of the results.	Since the accuracy of the results is not always guaranteed, it can be an important problem in noise filters.
P. Li et al., 2011 [28]	They analyzed the stochastic absolute value function, stochastic tanh function, and one-parameter linear gain function in stochastic computing.	Edge detection, frame difference-based image segmentation, median filter-based image enhancement, and image contrast stretching.	It proved stochastic computing is tolerant of soft errors and consumes fewer hardware resources than conventional computing.	As with other applications utilizing SC, it can be a problem if high accuracy is required.

The following is a summary of how the works introduced in Table 1 address the potential critical drawback of low accuracy in stochastic computing.

In [9], the authors propose an efficient yet simple stochastic computation technique for multipliers and adders by exchanging the wires used for their operation. This design reduces the relative error in computation compared to conventional designs.

In [26], the authors propose a new technique called the approximate stochastic computing (ASC) approach focusing on image processing applications. This approach reduces the computation time of an SC by a factor of 16 at a trade-off of an error percentage of 3.13% in the absolute stochastic value.

And in [27], the authors implement an efficient hardware design for a well-known fuzzy noise reduction filter based on stochastic computing. The filter consists of two main stages: edge detection and fuzzy smoothing. The results demonstrate that the proposed design reduces the relative error in computation compared to the conventional designs and has a smaller area.

#### 4. System Architecture

Figure 8 provides a comprehensive illustration of the overall architecture of our proposed system. This system is designed for lightweight and error-tolerant stereo matching,

and it incorporates a stochastic computing processor as a key component. The SC processor is composed of two main parts: an SC module and an ARM Cortex-M0 core [11].

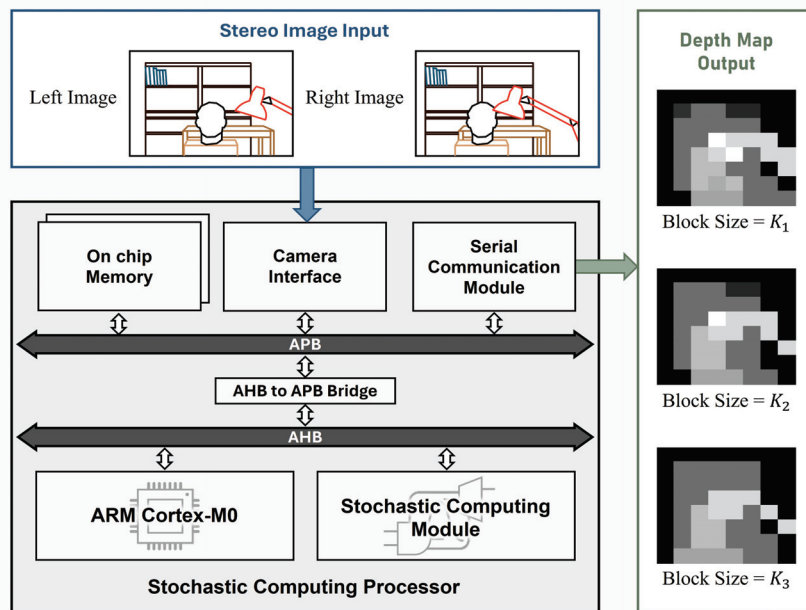


Figure 8. The overall architecture of the proposed system.

The SC module is responsible for the stochastic computations required for the stereo matching process. It is designed to efficiently calculate the SAD values, which are crucial for disparity computation in stereo matching. The ARM Cortex-M0 core, on the other hand, serves as the control unit of the system [29,30]. Once the stereo images are received from a camera module via the camera interface, the Cortex-M0 controls the stereo matching process by commanding the SC module to calculate the SAD values.

Once the SAD values are computed, the Cortex-M0 core proceeds to the next step of the process: disparity computation, which computes the disparities based on the SAD values and generates a depth map. The stereo matching operations are performed with a preset block size. Finally, the resulting depth map is transmitted to the outside through the serial communication module.

#### 4.1. Stochastic Computing Module

As shown in Figure 9, the stochastic computing module includes a stochastic configuration register and stochastic computing core, which consists of stochastic number generators (SNGs), a stochastic computing unit, and a probability estimator (PE).

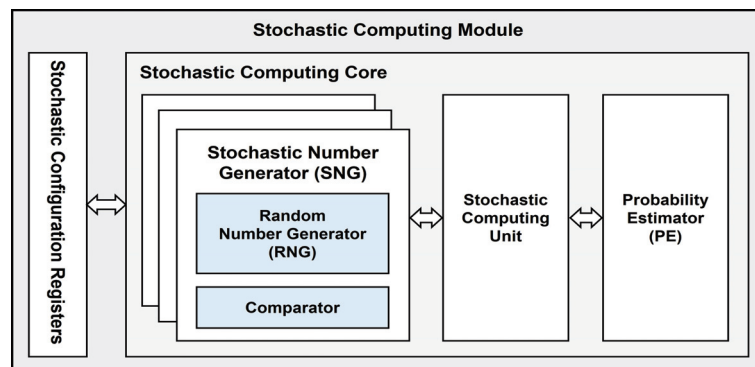


Figure 9. A block diagram of the stochastic computing module.

#### 4.1.1. Stochastic Number Generator

In this work, we have employed a parallel architecture of an LFSR in the random number generator (RNG) to effectively address the issue of latency [31]. The latency here is reduced by generating multiple random numbers simultaneously during the stochastic sequence generation process.

Figure 10 provides an illustration of this concept. It shows the architecture of the basic SNG and that of the SNG equipped with a parallel LFSR [32]. The parallel LFSR, as shown in Figure 10b, is capable of generating multiple random numbers at the same time. This is achieved with a relatively low circuit area, especially when compared to an architecture that simply utilizes multiple LFSRs.

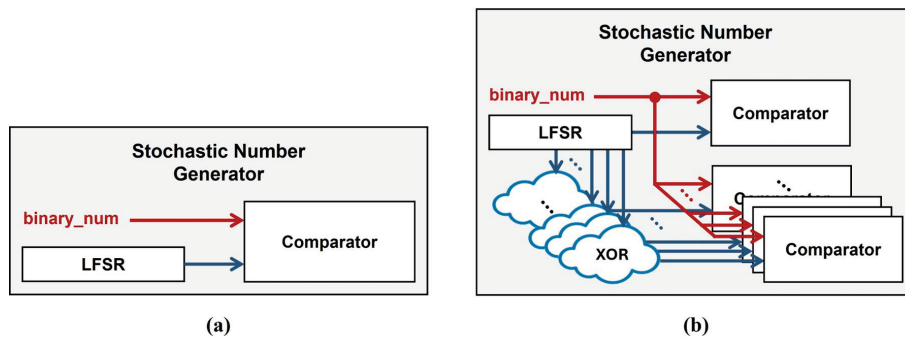


Figure 10. (a) The basic SNG architecture. (b) The SNG with a parallel LFSR.

The basic LFSR outputs a random number in a clock cycle. Therefore, if 256 random numbers are to be obtained through an 8-bit LFSR, 256 clocks are required [33]. The circuit employed in this work, however, requires only one clock cycle by designing the parallel LFSR circuit with 256 stages, as shown in Figure 11. In the circuit, the first random number output of the LFSR becomes the input of the 256th random number output, and the 256th random number output becomes the input of the 255th random number output, and so forth. The parallel LFSR circuit employs only one LFSR, eliminating the risk of loss in error rate due to correlations. Also, the circuit utilizes only XOR gates so that area efficiency is achieved without additional registers.

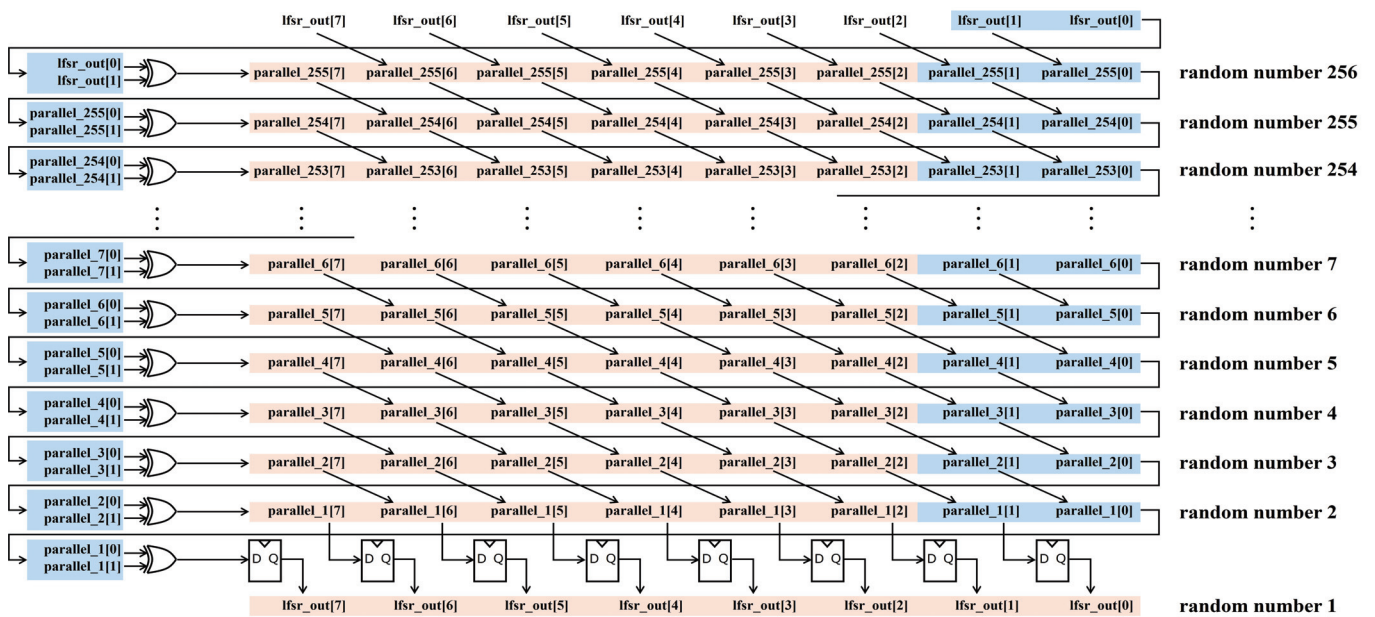


Figure 11. Circuit of 256-stage parallel LFSR.



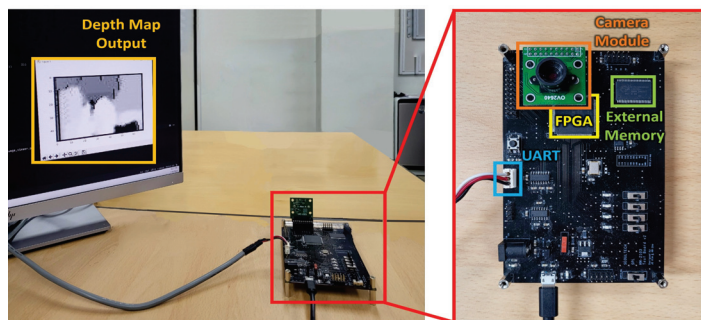
#### 4.1.2. Probability Estimator

Since the data output through the SC unit is still a stochastic sequence, it is necessary to convert the sequence back to the binary number, and the PE performs this conversion. In the probability value of  $P = N/L$  encoded from the stochastic sequence, the number of '1's ( $N$ ) is approximated to a binary number. Therefore, the number of '1's on a stochastic sequence output as a result of the operation is counted, and the conversion to the binary number is performed in consideration of the case of being scaled [34].

### 5. Implementation and Experiments

#### 5.1. Hardware Implementation

Figure 12 demonstrates the hardware-implemented system. We designed the proposed SC processor, which includes the 8-bit parallel LFSRs, with Verilog HDL and downloaded it to an Altera MAX10, 10M50SCE144C8G FPGA. The SC processor interfaces with the external system through a universal asynchronous receiver/transmitter (UART), a serial communication module that enables serial communications. Furthermore, commands and input data for performing the stereo matching operation are stored through an external memory (4Mbit SRAM, CY7C1049GN30). The computation result, the depth map, is transmitted to the external system via UART.

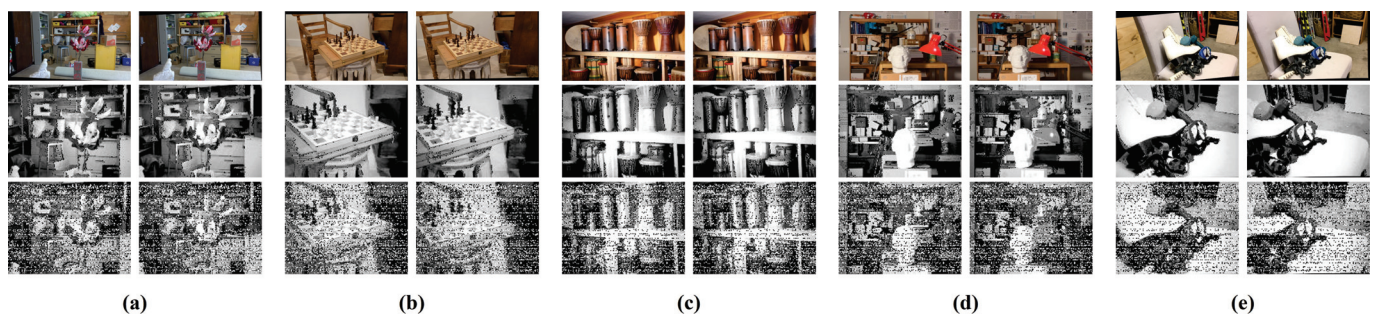


**Figure 12.** The hardware-implemented SC processor.

#### 5.2. Experiments

##### 5.2.1. Experimental Procedures

For the experiments, we employed a stereo image dataset [35]. To make the dataset images suitable for our proposed system, we converted them to grayscale and resized and cropped them [36], setting the resolution of the input images to  $160 \times 120$ . Figure 13 shows the employed dataset and the input images of the experiments.



**Figure 13.** The stereo image dataset, noise-free input images, and noise-injected input images: (a) artroom, (b) chess, (c) djembe, (d) newkuba, (e) skates.

Before conducting the experiments, we performed a software simulation. This allowed us to determine the block sizes that would yield suitable results from the employed stereo matching algorithm. We settled on block sizes of 15, 19, and 23. In the resulting output depth maps, the disparities of each pixel were mapped to values between 0 and 255.

To compare the depth maps obtained through this process, we converted the depth maps to histograms. We then computed the similarity between the histograms with several functions: correlation, chi-square, intersection, bhattacharyya distance, peak signal-to-noise ratio (PSNR), and NCC.

The first experiment out of two experiments compared the depth map obtained utilizing SC with the depth map obtained without utilizing SC on error-free input images to verify the feasibility of the proposed system. The second experiment repeated the first experiment but with 30% salt-and-pepper noise injected into the input images. The depth maps obtained utilizing SC and without utilizing SC on the noisy images were each compared with the depth map obtained without utilizing SC on noise-free images. This was performed to verify the error tolerance of the proposed system.

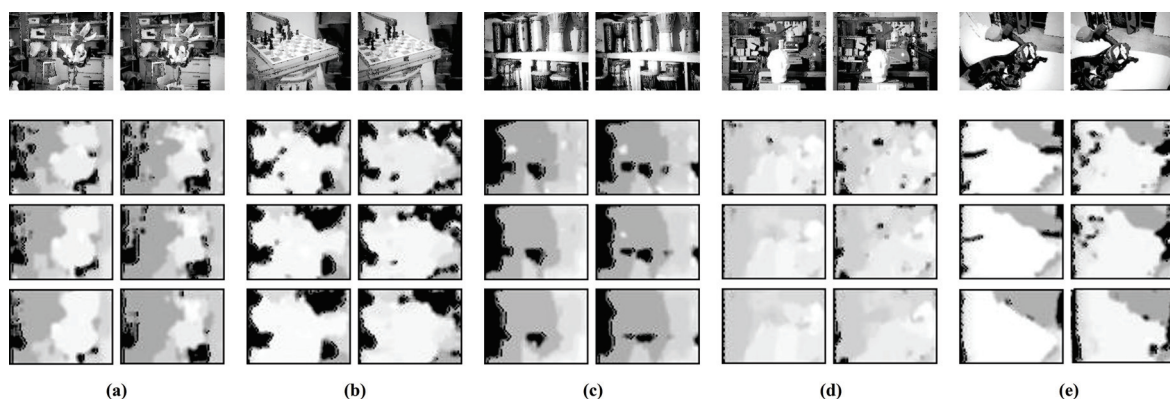
### 5.2.2. Feasibility Verification

Figure 14 and Table 2 display the results with noise-free input images. In Figure 14, the top figures show the input images, followed by the depth maps obtained when the block size is 15, 19, and 23, respectively, in order. The left depth maps were obtained without SC, and the right ones with SC. When comparing the two depth maps, they have no significant differences visually.

**Table 2.** The comparison results of depth maps from noise-free input images.

Input Dataset	Block Size	Comparison Method					
		A *	B *	C *	D *	PSNR (db)	NCC
artroom	15	0.925	1.759	0.810	0.216	16.395	0.866
	19	0.891	0.956	0.726	0.235	16.337	0.868
	23	0.868	0.997	0.693	0.261	16.212	0.855
chess	15	0.986	2.127	0.910	0.207	12.265	0.749
	19	0.985	2.167	0.908	0.236	12.759	0.792
	23	0.988	1.454	0.915	0.232	12.864	0.803
djembe	15	0.995	0.627	0.912	0.151	17.109	0.896
	19	0.997	0.260	0.943	0.137	17.904	0.910
	23	0.997	0.210	0.916	0.145	18.922	0.926
newkuba	15	0.953	2.876	0.896	0.228	16.846	0.853
	19	0.954	1.286	0.857	0.234	19.357	0.910
	23	0.948	0.931	0.792	0.213	21.588	0.944
skates	15	0.994	1.377	0.979	0.231	13.923	0.784
	19	0.997	0.992	0.957	0.213	14.413	0.808
	23	0.997	0.690	0.962	0.206	13.707	0.759

\* A: correlation, B: chi-square, C: intersection, D: bhattacharyya distance.



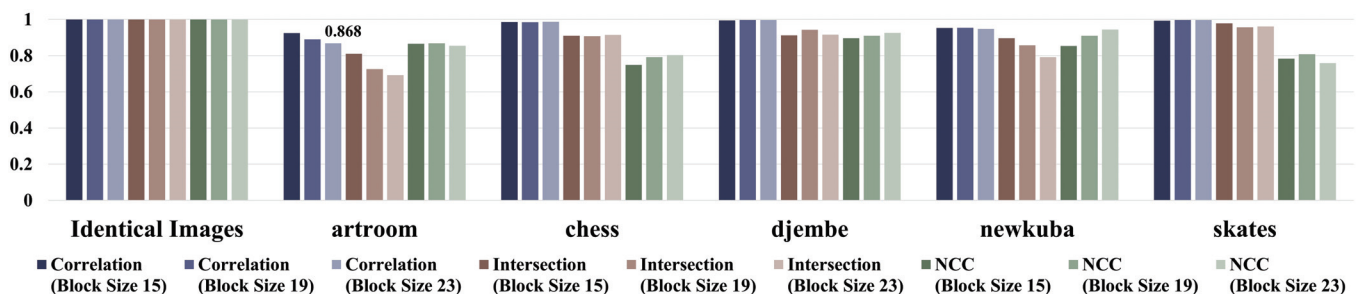
**Figure 14.** The stereo matching results from noise-free input images: (a) artroom, (b) chess, (c) djembe, (d) newkuba, (e) skates.



Upon a more detailed comparison through the similarity values calculated and presented in Table 2, we notably observed the high similarity. This is particularly true for functions such as correlation, intersection, and NCC, which yield a result of ‘1.000’ when computed between identical images, as shown in Table 3. These high values confirmed a significant degree of similarity between the two depth maps. Notably, in the case of correlation, all similarity values were found to be ‘0.868’ or higher, further substantiating this observation. In Figure 15, the similarity values through the above-mentioned functions are expressed as graphs.

**Table 3.** The similarity values when both images are identical.

Comparison Method					
Correlation	Chi-Square	Intersection	Bhattacharyya Distance	PSNR (db)	NCC
1.000	0.000	1.000	0.000	$\infty$	1.000



**Figure 15.** Graphs of similarity values between two depth maps obtained from noise-free input images.

Upon analyzing the experimental results in Figures 14 and 15 and Table 2, it is visually observed from Figure 14 that for relatively simple input images such as ‘artroom’, ‘chess’, and ‘skates’, the difference between the depth maps obtained with and without SC is somewhat larger compared to other datasets. This observation is also confirmed through the experimental results displayed in Table 2 and Figure 15. It is inferred that this is due to the fact that the more complex the image being compared in the template matching process (i.e., the more features present), the more advantageous it is to find similar images. Furthermore, through Figure 15, it is confirmed that as the block size increases, the similarity between the two depth maps gradually decreases.

### 5.2.3. Error Tolerance Verification

Figure 16 and Tables 4 and 5 display the results with noise-injected input images. In Figure 16, the top figures illustrate the noise-injected input images, followed by the depth maps obtained with the block sizes. As shown in Figure 14, the left depth maps were obtained without SC, and the right ones with SC. Comparing both depth maps, it is confirmed that more parts remain when the depth map is obtained with SC than when it is without SC, that is, it is more tolerant to the injected errors.

Table 4 shows the similarity values calculated by comparing depth maps obtained without SC from the noise-injected input images and depth maps obtained without SC from the noise-free input images. Conversely, Table 5 shows the similarity values calculated by comparing depth maps obtained with SC from the noise-injected input images and depth maps obtained without SC from the noise-free input images. That is, both depth maps obtained from the noise-injected input images were compared with the same depth map to find out how much improvement is achieved when utilizing SC.

**Table 4.** The comparison results of depth maps without SC from noise-injected input images.

Input Dataset	Block Size	Comparison Method					
		A *	B *	C *	D *	PSNR (db)	NCC
artroom	15	0.125	9.448	0.065	0.800	3.890	0.285
	19	0.090	11.741	0.059	0.827	3.520	0.244
	23	0.050	22.174	0.047	0.857	3.282	0.226
chess	15	0.253	4.993	0.117	0.756	3.296	0.188
	19	0.289	3.950	0.144	0.743	3.114	0.152
	23	0.292	3.635	0.156	0.757	3.030	0.158
djembe	15	0.342	3.535	0.152	0.764	4.176	0.228
	19	0.293	3.693	0.145	0.775	4.197	0.347
	23	0.261	3.997	0.137	0.783	4.204	0.526
newkuba	15	0.034	6.100	0.044	0.812	2.920	0.246
	19	0.037	5.386	0.042	0.822	2.743	0.229
	23	0.039	5.091	0.042	0.824	2.658	0.222
skates	15	0.367	26.082	0.278	0.658	4.087	0.234
	19	0.379	56.550	0.298	0.676	4.417	0.439
	23	0.201	46.279	0.177	0.734	3.558	0.351

\* A: correlation, B: chi-square, C: intersection, D: bhattacharyya distance.

**Table 5.** The comparison results of depth maps with SC from noise-injected input images.

Input Dataset	Block Size	Comparison Method					
		A *	B *	C *	D *	PSNR (db)	NCC
artroom	15	0.171	8.833	0.141	0.605	5.244	0.417
	19	0.144	11.250	0.130	0.645	4.586	0.342
	23	0.098	21.865	0.106	0.707	4.068	0.285
chess	15	0.342	4.695	0.255	0.542	5.162	0.445
	19	0.373	3.598	0.258	0.567	4.558	0.390
	23	0.373	3.413	0.246	0.595	4.103	0.332
djembe	15	0.153	8.474	0.068	0.885	5.307	0.240
	19	0.604	3.634	0.444	0.440	7.227	0.449
	23	0.593	3.228	0.436	0.455	7.044	0.419
newkuba	15	0.134	5.661	0.145	0.664	4.759	0.395
	19	0.125	4.689	0.111	0.717	4.143	0.345
	23	0.137	4.526	0.102	0.745	3.877	0.326
skates	15	0.595	30.309	0.691	0.475	7.383	0.568
	19	0.582	59.356	0.661	0.511	6.574	0.537
	23	0.425	46.695	0.450	0.587	5.593	0.518

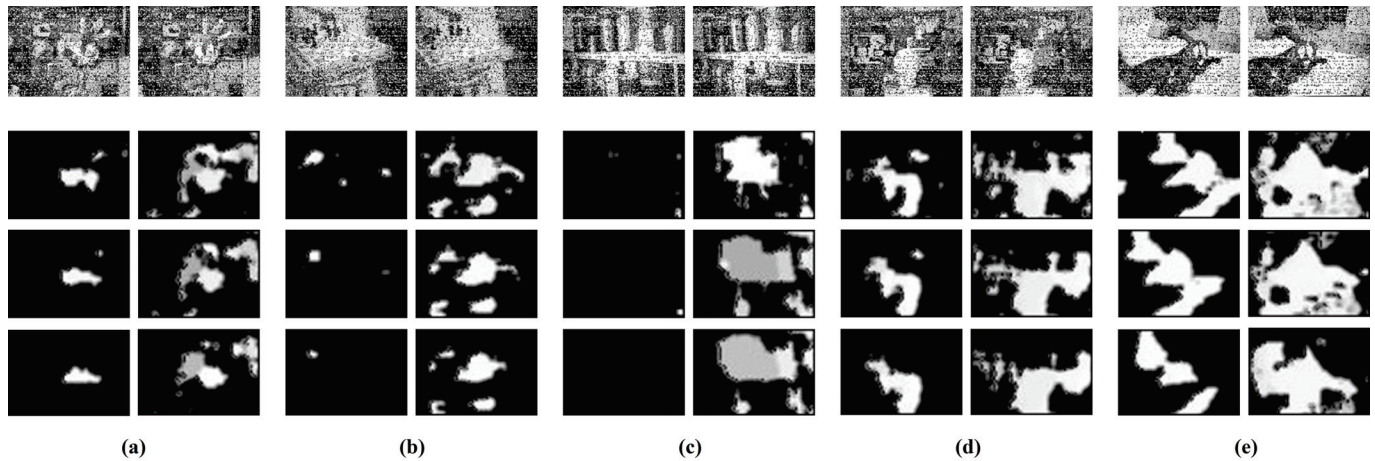
\* A: correlation, B: chi-square, C: intersection, D: bhattacharyya distance.

Upon comparing Tables 4 and 5, it is evident that the depth maps obtained with SC generally approached the similarity values calculated between identical images. To quantitatively assess the improvement, we calculated improvement rates with a formula:  $(Final\ Value - Initial\ Value) / Initial\ Value \times 100\ [\%]$ .

The improvement rates of the depth maps for each function, input dataset, and block size were calculated and are presented in Table 6.

In Table 6, the overall average of the improvement rate was approximately 58.95%. The average improvement rates for the block sizes starting from 15 were approximately 51.03%, 62.50%, and 63.33%, respectively. The results indicate a different trend from the previous experiments, where the similarity decreased as the block size increased. This is able to be explained from the fact that the cost matching function, SAD, employed in our work does not consider the position of pixels during calculation. Therefore, while larger

block sizes are disadvantageous for performing template matching, the cumulative error in calculation when not using SC for noisy input images increases with block size. This means that it is possible for the degree of improvement when using SC to be made larger. These experimental results allow us to confirm the error tolerance of the proposed system.



**Figure 16.** The stereo matching results from noise-injected input images: (a) artroom, (b) chess, (c) djembe, (d) newkuba, (e) skates.

**Table 6.** The improvement rates of depth maps.

Input Dataset	Block Size	Comparison Method					
		A *	B *	C *	D *	PSNR (db)	NCC
artroom	15	36.80%	6.51%	116.92%	24.38%	34.81%	46.32%
	19	60.00%	4.18%	120.34%	22.01%	30.28%	40.16%
	23	96.00%	1.39%	125.53%	17.50%	23.95%	26.11%
chess	15	35.18%	5.97%	117.95%	28.31%	56.61%	136.70%
	19	29.07%	8.91%	79.17%	23.69%	46.37%	156.58%
	23	27.74%	6.11%	57.69%	21.40%	35.41%	110.13%
djembe	15	−55.26%	−139.75%	−55.26%	−15.84%	27.08%	5.26%
	19	106.14%	1.60%	206.21%	43.23%	72.19%	29.39%
	23	127.20%	19.24%	218.25%	41.89%	67.55%	−20.34%
newkuba	15	294.12%	7.20%	229.55%	18.23%	62.98%	60.57%
	19	237.84%	12.94%	164.29%	12.77%	51.04%	50.66%
	23	251.28%	11.10%	142.86%	9.59%	45.86%	46.85%
skates	15	62.13%	−16.21%	148.56%	27.81%	80.65%	142.74%
	19	53.56%	−4.96%	121.81%	24.41%	48.83%	22.32%
	23	111.44%	−0.90%	154.24%	20.03%	57.20%	47.58%

\* A: correlation, B: chi-square, C: intersection, D: bhattacharyya distance.

## 6. Conclusions

This work presents a novel approach for lightweight and error-tolerant stereo matching with a hardware-implemented SC processor. SC, which represents and processes data utilizing stochastic sequences and leverages the concept of mathematical probability, replaces traditional arithmetic units with concise logic gates such as AND or MUX. This makes the SC processor lightweight compared to methods that rely on complex 3D CNNs or weighted loss functions used in other works. Furthermore, each digit in a stochastic sequence has the same weight in SC, so even if an error occurs in any bit of the stochastic sequence, the impact on the output is relatively small. This makes SC more error-tolerant compared to methods that may struggle with disparity estimation in occluded regions or unsupervised training.

In this work, we have successfully implemented a system that leverages the characteristics of SC to obtain depth maps through stereo matching operations. For this purpose, we designed an SC processor with Verilog HDL and implemented it on an FPGA. The implemented SC processor includes an SC unit for SC operations, which contains a stochastic number generator composed of a random number generator and a comparator to generate stochastic sequences. The random number generator is designed as a 256-stage parallel LFSR capable of generating 256 random numbers within one clock cycle. This makes it more efficient compared to methods that require a balance between disparity estimation accuracy and efficiency [37–39].

We conducted experiments to verify the feasibility of the proposed system by obtaining depth maps from noise-free input images and to validate its error tolerance by obtaining depth maps from noise-injected images. The experimental results demonstrated that obtaining depth maps from noise-injected input images with SC improved by an average of 58.95% compared to without SC. This indicates that the proposed system exhibits error tolerance without significantly degrading the quality of the output depth map.

The system is expected to be applicable in environments where errors are likely to occur due to instability, such as inside vehicles, and where available resources are limited [40]. However, the system proposed in this paper did not demonstrate an operation time fast enough to be suitable for real-time applications during the experimental process. This is able to be attributed to various delays occurring within the system, such as the delay in data movement through the bus and the delay in re-converting the stochastic sequence into a binary number. Therefore, in future work, it will be necessary to conduct research to implement a system in which it is possible to be applied not only to images but also to video applications. This could be achieved by directly mounting the SC module inside the processor core to reduce the delay from data movement or by further optimizing the architecture of the SC module to reduce delay. This will enhance the versatility and applicability of the system in various real-world scenarios.

**Author Contributions:** Conceptualization, S.A.; methodology, S.L., Y.J. and J.K. (Jeongeun Kim); hardware, J.K. (Jeongeun Kim); software, S.A. and J.K. (Jeongeun Kim); validation, J.O.; formal analysis, S.L.; investigation, S.A., J.O., S.L. and J.K. (Jinyeol Kim); data curation, S.A. and J.O.; writing—original draft preparation, S.A.; writing—review and editing, S.A., J.O., S.L. and Y.J.; visualization, S.L. and J.K. (Jinyeol Kim); supervision, S.E.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This study was supported by SeoulTech (Seoul National University of Science and Technology).

**Data Availability Statement:** The data presented in this study are available in this article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

SC	stochastic computing
AR	augmented reality
LFSR	linear feedback shift register
FPGA	Field-Programmable Gate Array
MUX	multiplexer
SAD	sum of absolute difference
SSD	sum of squared difference
NCC	normalized cross correlation
ESL	Extended Stochastic Logic
ANN	artificial neural network
NN	neural network
CNN	convolutional neural network
ASC	approximate stochastic computing

SNG	stochastic number generator
RNG	random number generator
PE	probability estimator
UART	universal asynchronous receiver/transmitter
PSNR	peak signal-to-noise ratio

## References

- Domínguez-Morales, M.J.; Jiménez-Fernández, Á.; Jiménez-Moreno, G.; Conde, C.; Cabello, E.; Linares-Barranco, A. Bio-Inspired Stereo Vision Calibration for Dynamic Vision Sensors. *IEEE Access* **2019**, *7*, 138415–138425. [CrossRef]
- Hallek, M.; Boukamcha, H.; Smach, F.; Atri, M. Real Time Stereo Matching Using Two Step Zero-Mean SAD and Dynamic Programing. In Proceedings of the 2018 15th International Multi-Conference on Systems, Signals & Devices (SSD), Yasmine Hammamet, Tunisia, 19–22 March 2018; pp. 1234–1240. [CrossRef]
- Lin, X.; Wang, J.; Lin, C. Research on 3D Reconstruction in Binocular Stereo Vision Based on Feature Point Matching Method. In Proceedings of the 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 27–29 September 2020; pp. 551–556. [CrossRef]
- Liu, S.; Gross, W.J.; Han, J. Introduction to Dynamic Stochastic Computing. *IEEE Circuits Syst. Mag.* **2020**, *20*, 19–33. [CrossRef]
- Shivanandamurthy, S.M.; Thakkar, I.G.; Salehi, S.A. A scalable stochastic number generator for phase change memory based in-memory stochastic processing: Work-in-progress. In Proceedings of the Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion, New York, NY, USA, 13–18 October 2019. [CrossRef]
- Kim, J.; Jeong, W.S.; Jeong, Y.; Lee, S.E. Parallel stochastic computing architecture for computationally intensive applications. *Electronics* **2023**, *12*, 1749. [CrossRef]
- Chen, K.C.; Wu, C.H. High-Accurate Stochastic Computing for Artificial Neural Network by Using Extended Stochastic Logic. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–4. [CrossRef]
- Jang, S.Y.; Yoon, Y.H.; Lee, S.E. Stochastic Computing based AI System for Mobile Devices. In Proceedings of the 2020 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 4–6 January 2020; pp. 1–2. [CrossRef]
- Temenos, N.; Sotiriadis, P.P. Modeling a Stochastic Computing Nonscaling Adder and its Application in Image Sharpening. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 2543–2547. [CrossRef]
- Kang, S.; Pan, R.; Zhao, K.; Dong, J.; Zhao, Y. Implementation of Stochastic Computing-based Image Compression System Using Probabilistic Switching Behavior of RRAM. In Proceedings of the 2022 15th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 5–7 November 2022; pp. 1–6. [CrossRef]
- Kim, J.; Jeong, Y.R.; Cho, K.; Jeong, W.S.; Lee, S.E. Reconfigurable Stochastic Computing Architecture for Computationally Intensive Applications. In Proceedings of the 2022 19th International SoC Design Conference (ISOCC), Gangneung-si, Republic of Korea, 19–22 October 2022; pp. 61–62. [CrossRef]
- Ma, C.; Zhong, S.; Dang, H. High Fault Tolerant Image Processing System Based on Stochastic Computing. In Proceedings of the 2012 International Conference on Computer Science and Service System, Nanjing, China, 11–13 August 2012; pp. 1587–1590. [CrossRef]
- Zhang, Z.; Wang, R.; Zhang, Z.; Zhang, Y.; Guo, S.; Huang, R. Circuit Reliability Comparison Between Stochastic Computing and Binary Computing. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 3342–3346. [CrossRef]
- Liu, Z.; Song, B.; Guo, Y.; Xu, H. Improved Template Matching Based Stereo Vision Sparse 3D Reconstruction Algorithm. In Proceedings of the 2020 Chinese Control And Decision Conference (CCDC), Hefei, China, 22–24 August 2020; pp. 4305–4310. [CrossRef]
- Perri, S.; Frustaci, F.; Spagnolo, F.; Corsonello, P. Design of Real-Time FPGA-based Embedded System for Stereo Vision. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5. [CrossRef]
- Zhao, J.; Liang, T.; Feng, L.; Ding, W.; Sinha, S.; Zhang, W.; Shen, S. FP-Stereo: Hardware-efficient stereo vision for embedded applications. In Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 31 August–4 September 2020; IEEE: New York, NY, USA, 2020; pp. 269–276.
- Choi, C.H.; Kim, Y.; Ha, J.; Moon, B. Haar Filter Hardware Architecture for the Accuracy Improvement of Stereo Vision Systems. In Proceedings of the 2021 18th International SoC Design Conference (ISOCC), Jeju Island, Republic of Korea, 6–9 October 2021; pp. 401–402. [CrossRef]
- Gani, S.F.A.; Miskon, M.F.; Hamzah, R.A. Depth Map Information from Stereo Image Pairs using Deep Learning and Bilateral Filter for Machine Vision Application. In Proceedings of the 2022 IEEE 5th International Symposium in Robotics and Manufacturing Automation (ROMA), Malacca, Malaysia, 6–8 August 2022; pp. 1–6. [CrossRef]
- Chen, F.; Liu, X.; Yu, H.; Ha, Y. CLIF: Cross-Layer Information Fusion for Stereo Matching and its Hardware Implementation. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–5. [CrossRef]
- Yang, L.; Wang, B.; Zhang, R.; Zhou, H.; Wang, R. Analysis on Location Accuracy for the Binocular Stereo Vision System. *IEEE Photonics J.* **2018**, *10*, 1–16. [CrossRef]



21. Jia, T.; Ma, J.; Li, W.; Zhang, Y.; Zeng, Z.; Huang, J. Implementation of Real-Time Stereo Matching System Based on Speckle Structured Light. In Proceedings of the 2021 33rd Chinese Control and Decision Conference (CCDC), Kunming, China, 22–24 May 2021; pp. 1537–1542. [CrossRef]
22. George, G.; Oommen, R.M.; Shelly, S.; Philipose, S.S.; Varghese, A.M. A Survey on Various Median Filtering Techniques For Removal of Impulse Noise From Digital Image. In Proceedings of the 2018 Conference on Emerging Devices and Smart Systems (ICEDSS), Tiruchengode, India, 2–3 March 2018; pp. 235–238. [CrossRef]
23. Zhang, Y.; Lin, S.; Wang, R.; Wang, Y.; Wang, Y.; Qian, W.; Huang, R. When Sorting Network Meets Parallel Bitstreams: A Fault-Tolerant Parallel Ternary Neural Network Accelerator based on Stochastic Computing. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 1287–1290. [CrossRef]
24. Hu, A.; Li, W.; Lv, D.; He, G. An Efficient Stochastic Convolution Accelerator Based on Pseudo-Sobol Sequences. In Proceedings of the 17th ACM International Symposium on Nanoscale Architectures, Virtual, OR, USA, 7–9 December 2022; pp. 1–6.
25. Joe, H.; Kim, Y. Novel stochastic computing for energy-efficient image processors. *Electronics* **2019**, *8*, 720. [CrossRef]
26. Seva, R.; Metku, P.; Kim, K.K.; Kim, Y.B.; Choi, M. Approximate stochastic computing (ASC) for image processing applications. In Proceedings of the 2016 International SoC Design Conference (ISOCC), Jeju, Republic of Korea, 23–26 October 2016; pp. 31–32. [CrossRef]
27. Estiri, S.N.; Jalilvand, A.H.; Naderi, S.; Najafi, M.H.; Fazeli, M. A Low-Cost Stochastic Computing-based Fuzzy Filtering for Image Noise Reduction. In Proceedings of the 2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC), Pittsburgh, PA, USA, 24–25 October 2022; pp. 1–6. [CrossRef]
28. Li, P.; Lilja, D.J. Using stochastic computing to implement digital image processing algorithms. In Proceedings of the 2011 IEEE 29th International Conference on Computer Design (ICCD), Amherst, MA, USA, 9–12 October 2011; pp. 154–161. [CrossRef]
29. Jeong, Y.; Oh, H.W.; Kim, S.; Lee, S.E. An Edge AI Device based Intelligent Transportation System. *J. Inf. Commun. Converg. Eng.* **2022**, *20*, 166–173. [CrossRef]
30. Cho, K.N.; Oh, H.W.; Lee, S.E. Vision-based Parking Occupation Detecting with Embedded AI Processor. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 10–12 January 2021; pp. 1–2. [CrossRef]
31. Laskin, E. *On-Chip Self-Test Circuit Blocks for High-Speed Applications*; University of Toronto: Toronto, ON, Canada, 2006.
32. Salehi, S.A. Low-Cost Stochastic Number Generators for Stochastic Computing. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 992–1001. [CrossRef]
33. Frasser, C.F.; Linares-Serrano, P.; Ríos, I.D.d.I.; Morán, A.; Skibinsky-Gitlin, E.S.; Font-Rosselló, J.; Canals, V.; Roca, M.; Serrano-Gotarredona, T.; Rosselló, J.L. Fully Parallel Stochastic Computing Hardware Implementation of Convolutional Neural Networks for Edge Computing Applications. *IEEE Trans. Neural Networks Learn. Syst.* **2023**, *34*, 10408–10418. [CrossRef] [PubMed]
34. Lin, Z.; Xie, G.; Xu, W.; Han, J.; Zhang, Y. Accelerating Stochastic Computing Using Deterministic Halton Sequences. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 3351–3355. [CrossRef]
35. Scharstein, D.; Hirschmüller, H.; Kitajima, Y.; Krathwohl, G.; Nešić, N.; Wang, X.; Westling, P. High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth. In *Pattern Recognition*; Jiang, X., Hornegger, J., Koch, R., Eds.; Springer: Cham, Switzerland, 2014; pp. 31–42.
36. Park, J.; Shin, J.; Kim, R.; An, S.; Lee, S.; Kim, J.; Oh, J.; Jeong, Y.; Kim, S.; Jeong, Y.R.; et al. Accelerating Strawberry Ripeness Classification Using a Convolution-Based Feature Extractor along with an Edge AI Processor. *Electronics* **2024**, *13*, 344. [CrossRef]
37. Poggi, M.; Pallotti, D.; Tosi, F.; Mattoccia, S. Guided stereo matching. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 979–988.
38. Bi, W.; Chen, M.; Wu, D.; Lu, S. EBStereo: Edge-based loss function for real-time stereo matching. *Vis. Comput.* **2023**, *40*, 2975–2986. [CrossRef]
39. Lee, S.H.; Kanatsugu, Y.; Park, J.I. MAP-based stochastic diffusion for stereo matching and line fields estimation. *Int. J. Comput. Vis.* **2002**, *47*, 195–218. [CrossRef]
40. Han, C.Y.; Jeong, Y.S.; Lee, S.E. Simulation-Based Fault Analysis for Resilient System-On-Chip Design. *J. Inf. Commun. Converg. Eng.* **2021**, *19*, 175–179. [CrossRef]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.



## Article

# Highly Fault-Tolerant Systolic-Array-Based Matrix Multiplication

Hsin-Chen Lu, Liang-Ying Su and Shih-Hsu Huang \*

Department of Electronic Engineering, Chung Yuan Christian University, Taoyuan 320314, Taiwan;  
hsinchen@cycu.org.tw (H.-C.L.); g11202601@cycu.edu.tw (L.-Y.S.)

\* Correspondence: shhuang@cycu.edu.tw; Tel.: +886-3-2654611

**Abstract:** Matrix multiplication plays a crucial role in various engineering and scientific applications. Cannon's algorithm, executed within two-dimensional systolic arrays, significantly enhances computational efficiency through parallel processing. However, as the matrix size increases, reliability issues become more prominent. Although the previous work has proposed a fault-tolerant mechanism, it is only suitable for scenarios with a limited number of faulty processing elements (PEs). This paper introduces a pair-matching mechanism, assigning a fault-free PE as a proxy for each faulty PE to execute its tasks. Our fault-tolerant mechanism comprises two stages: in the first stage, each fault-free PE completes its designated computations; in the second stage, computations intended for each faulty PE are executed by its assigned fault-free PE proxy. The experimental results demonstrate that compared to the previous work, our approach not only significantly improves the fault tolerance of systolic arrays (applicable to scenarios with a higher number of faulty PEs) but also reduces circuit areas. Therefore, the proposed approach proves effective in practical applications.

**Keywords:** fault tolerance; integrated circuits; parallel processing; processing elements; reliability

## 1. Introduction

Matrix multiplication plays a vital role in diverse engineering and scientific applications. For example, in convolutional neural networks, convolutions can also be depicted as matrix multiplications. To fulfill the needs of high-speed processing, minimizing the computation time for matrix multiplication is crucial. Therefore, several algorithms have been introduced in the past to accelerate matrix multiplications, such as Cannon's algorithm [1] and Winograd's algorithm [2].

A systolic array [3–7] serves as a general matrix multiplication accelerator, achieving matrix multiplication in a regular and parallel manner via data transmission between processing elements (PEs). Each PE is primarily responsible for executing multiply–accumulate computations [8,9]. Cannon's algorithm can also be implemented through a systolic array. Taking a  $4 \times 4$  systolic array as an example, if matrix operations are performed using the traditional algorithm, it would require 10 clock cycles, whereas utilizing Cannon's algorithm for matrix operations would only require four clock cycles.

Although a systolic array reduces computation time through parallel processing, it also increases the likelihood of hardware failures. Note that these hardware failures are caused by latent manufacturing defects [10,11] or circuit aging effects [12,13], potentially resulting in functional errors. We can employ post-manufacturing testing mechanisms [14,15] to identify latent manufacturing defects. Additionally, run-time detection mechanisms [16,17] can be used to identify the effects of circuit aging.

As the size of the matrix increases, the probability of hardware failures also increases. Therefore, besides identifying hardware failures, it is imperative to incorporate fault-tolerance mechanisms to enhance the reliability of the systolic array for matrix multiplication. Especially in applications such as healthcare, aviation, autonomous driving,

underwater communication, etc., where system reliability is crucial, fault-tolerance mechanisms are indispensable [18–20].

The research area of fault tolerance has a rich history, starting with early work on addressing permanent faults in memory systems using error-correcting codes [21,22] or redundant memories (such as spare rows and columns) [23,24], as well as triple modular redundancy strategies [25,26], to mitigate faults in computational logic. Nevertheless, these fault-tolerant design methodologies do not directly extend to systolic arrays.

Kim and Reddy [27] delved into fault-tolerant systolic array design, focusing on the notion of treating a faulty systolic array as a simplified version with fewer rows and columns. Stojanovic et al. [28] achieved fault tolerance through triplicated computations and majority voting, offering two hardware solutions for the voting process. Milovanovic et al. [29] introduced a systematic approach to designing fault-tolerant systolic arrays with orthogonal interconnects and unidirectional data flow for matrix multiplication. It is important to note that all of these methods [27–29] require the inclusion of redundant PEs.

Jan and Huang [30] discuss the fault-tolerant design of a systolic array executing Cannon's algorithm [1] for matrix multiplications. Their basic idea involves assigning the task of a faulty PE to a fault-free PE for execution. Therefore, their approach does not require redundant PEs. In [30], they propose a mapping algorithm to implement both a twisted column scheme and a column replacement scheme. If the number of faulty PEs is small, their method can achieve very high fault tolerance.

However, the approach proposed by Jan and Huang [30] cannot be applied when there is a large number of faulty PEs. The main reason is that their method requires the existence of at least one fault-free column (even in the form of a twisted column). However, if there are many faulty PEs, it may not be possible to find a fault-free column (even in the form of a twisted column). For certain application domains, such as space or deep sea exploration, it is difficult to repair even if many PEs fail. Therefore, we need to develop design methods with higher fault tolerance to enhance the lifespan of systolic arrays.

In this paper, we propose a highly fault-tolerant approach for a systolic array executing Cannon's algorithm for matrix multiplication. We observe that the main limitation of the previous work [30] lies in the necessity to find a fault-free column (even in the form of a twisted column). Consequently, the previous work is only suitable for situations where the number of faulty PEs is relatively low. In contrast to the previous approach [30], our method does not require the existence of a fault-free column, thus enabling its application in scenarios with a higher number of faulty PEs.

Our fundamental idea is to pair each faulty PE with a corresponding fault-free PE to act as its proxy. Therefore, our matrix multiplication computation is divided into two primary stages: in the first stage, each fault-free PE completes the computations that it should perform. In the second stage, the computations that each faulty PE should carry out are completed by the fault-free PE for which it is acting as a proxy. Since the data required by each fault-free PE are directly transmitted by the controller, the proposed approach does not require a fault-free column.

This paper introduces two pair-matching schemes: row-based pair-matching and column-based pair-matching. Based on these schemes, we propose two pair-matching algorithms: one-dimensional pair-matching (i.e., utilizing only row-based pair-matching) and two-dimensional pair-matching (i.e., utilizing both row-based pair-matching and column-based pair-matching). Employing two-dimensional pair-matching offers higher fault tolerance compared to using one-dimensional pair-matching, but it also increases the controller area.

We used the TSMC 40nm cell library to implement the proposed approach. The experiment results demonstrate that compared to the previous work [30], the proposed approach (whether utilizing one-dimensional pair-matching or two-dimensional pair-matching) not only enhances the fault tolerance but also reduces the circuit area. This is attributed to the fact that the previous work [30] requires consideration of the twisted column scheme, which not only complicates the data path design for each PE but also complicates the

controller design for the systolic array. Therefore, the proposed pair-matching method can simplify the design of both the controller and PEs.

The primary contributions of our work are elaborated below.

1. We introduce a fault-tolerant approach using pair-matching, with the analysis results showcasing a notable enhancement in the fault-tolerance capabilities compared to those in the previous work [30].
2. We implemented the pair-matching approach in circuitry, with the experimental results indicating reduced areas for both the controller and PEs in comparison to those in the previous work [30].

Note that, in recent years, there have also been some studies [31–34] exploring fault-tolerant designs for systolic-array-based deep neural network (DNN) accelerators. Due to the inherent fault-tolerant nature of artificial intelligence applications, these studies' fault-tolerant designs may allow for some errors. Unlike these studies [31–34], we must ensure the correctness of the matrix multiplication results. Therefore, the fault-tolerance issues that we investigate in matrix multiplication differ fundamentally from those addressed in the literature on the fault tolerance of DNN accelerators.

The remainder of this paper is structured as follows. Section 2 offers background materials. Section 3 introduces the proposed fault-tolerance approach, encompassing one-dimensional pair-matching and two-dimensional pair-matching. Section 4 showcases our experimental results. Lastly, concluding remarks are provided in Section 5.

## 2. Preliminaries

In this section, we present background materials. Section 2.1 introduces the systolic array architecture. Section 2.2 presents Cannon's algorithm. Section 2.3 discusses the fault-tolerant mechanism proposed by Jan and Huang [30].

### 2.1. The Systolic Array Architecture

Suppose that  $n \times n$  matrices  $A$  and  $B$  are represented as Equations (1) and (2), respectively:

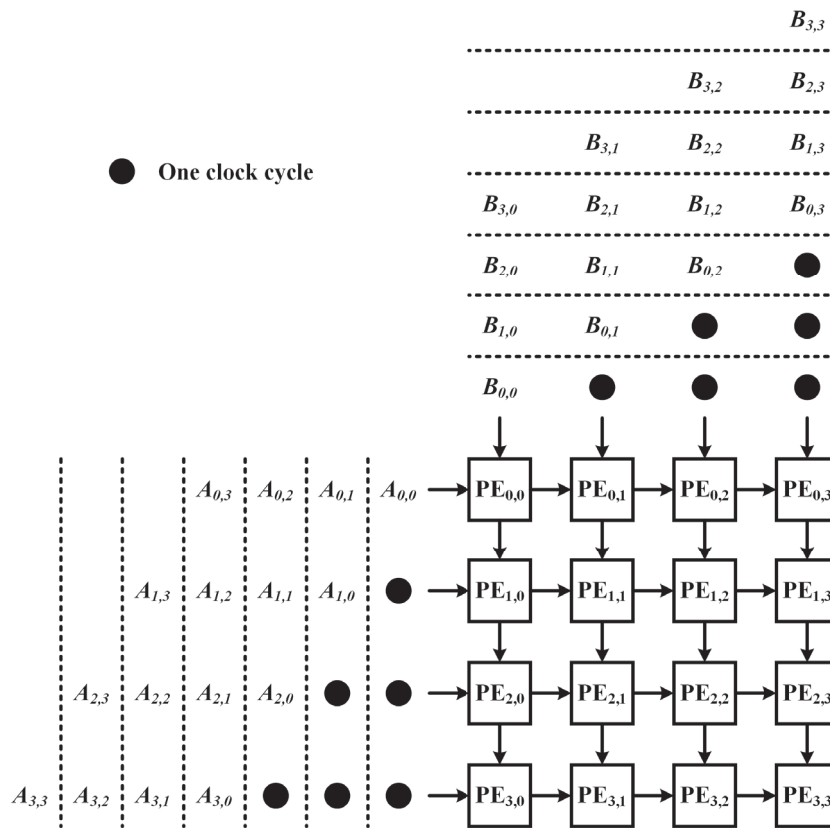
$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,n-1} \\ \vdots & \ddots & \vdots \\ A_{n-1,0} & \cdots & A_{n-1,n-1} \end{bmatrix} \quad (1)$$

$$B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,n-1} \\ \vdots & \ddots & \vdots \\ B_{n-1,0} & \cdots & B_{n-1,n-1} \end{bmatrix} \quad (2)$$

Then, the  $n \times n$  matrix multiplication  $A \times B$  can be expressed as Equation (3):

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} \times B_{k,j} \quad (3)$$

A systolic array functions as a versatile accelerator for matrix multiplication, accomplishing the task in a structured and parallel fashion through data exchange among PEs. Each PE is primarily tasked with performing multiply-accumulate computations. Additionally, each PE also requires interconnection circuits for data transmission. Taking a  $4 \times 4$  systolic array as an example, with each clock cycle, matrix  $A$  moves one step to the right, and matrix  $B$  moves one step downward, as shown in Figure 1. After 10 clock cycles, matrix multiplication can be completed.



**Figure 1.** A  $4 \times 4$  systolic array for matrix multiplications.

## 2.2. Cannon's Algorithm

Cannon's algorithm can be used to accelerate matrix multiplications. It primarily involves three steps:

1. Initial alignment: The rows of matrix  $A$  are shifted to the left by  $i$  steps ( $0 \leq i \leq n$ ), with the  $0$ -th row shifted 0 steps, the  $1$ -st row shifted 1 step, and so forth; the columns of matrix  $B$  are shifted upwards by  $j$  steps ( $0 \leq j \leq n - 1$ ), with the  $0$ -th column shifted 0 step, the  $1$ -st column shifted 1 step, and so forth.
2. Multiplication and accumulation: The two aligned matrices are overlapped, and multiply-accumulate operations are conducted on the PEs.
3. Global matrix shifting: All elements in matrix  $A$  are shifted one step to the left, and all elements in matrix  $B$  are shifted one step upward. Then, one proceeds to step 2.

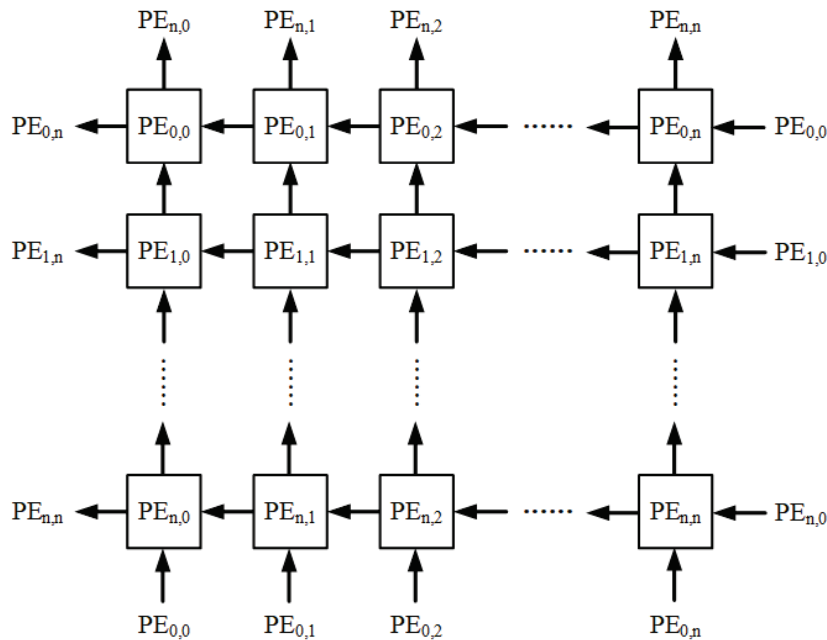
In Cannon's algorithm, step 2 must be executed a total of  $n$  times, and step 3 must be executed  $n - 1$  times in total. Then, the resulting matrix  $C$  is obtained.

In conjunction with Cannon's algorithm, each PE is connected to two registers, which store a value from the aligned matrix  $A$  and a value from the aligned matrix  $B$ , respectively. The PEs in the  $0$ -th row are connected to those in the last row, and similarly, those in the  $0$ -th column are connected to those in the last column, as illustrated in Figure 2.

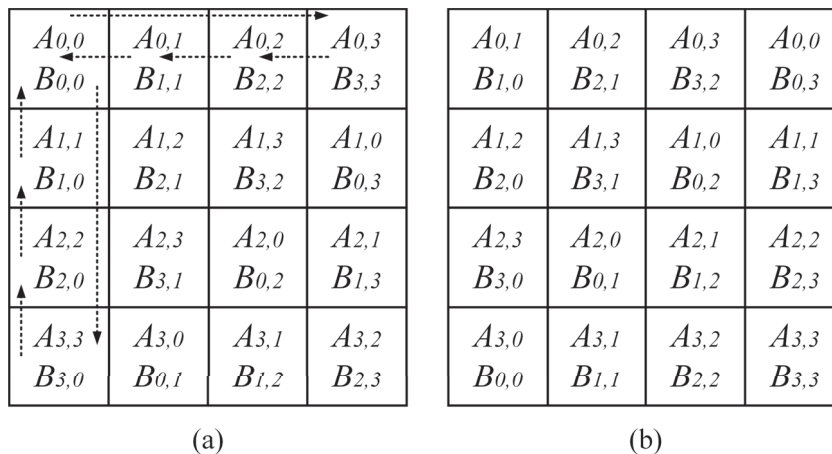
Based on the initial alignment of matrix  $A$  and matrix  $B$ , corresponding values are simultaneously transmitted to each PE. Subsequently, multiply-accumulate operations are executed in all PEs. During each global matrix shifting, the values of matrix  $A$  are sent to the left neighboring PEs, while the values of matrix  $B$  are sent upward to the neighboring PEs. As a result, only  $n$  clock cycles are needed to complete the matrix multiplication.

We illustrate an example using  $4 \times 4$  matrix multiplication. Figure 3a overlaps the aligned matrix  $A$  and the aligned matrix  $B$ . For example, at this point, the elements overlapping at  $PE_{1,2}$  are  $A_{1,3}$  and  $B_{3,2}$ . Next, we shift all elements of matrix  $A$  one step to the left and all elements of matrix  $B$  one step up. For clarity, in Figure 3a, we use the  $0$ -th row as an example to indicate the shift direction of the elements in matrix  $A$  and the

0-th column as an example to indicate the shift direction of the elements in matrix  $B$ . Figure 3b gives the results after the first global matrix shifting is performed. As shown in Figure 3b, the elements overlapping at  $PE_{1,2}$  become  $A_{1,0}$  and  $B_{0,2}$ . This process is repeated 4 times for multiply-accumulate and 3 times for global matrix shifting to obtain the final result. For instance, at the position  $PE_{1,2}$ , we obtain the final result of  $C_{1,2}$  as  $A_{1,3} \times B_{3,2} + A_{1,0} \times B_{0,2} + A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2}$ .



**Figure 2.** The systolic array architecture applied to Cannon's algorithm.



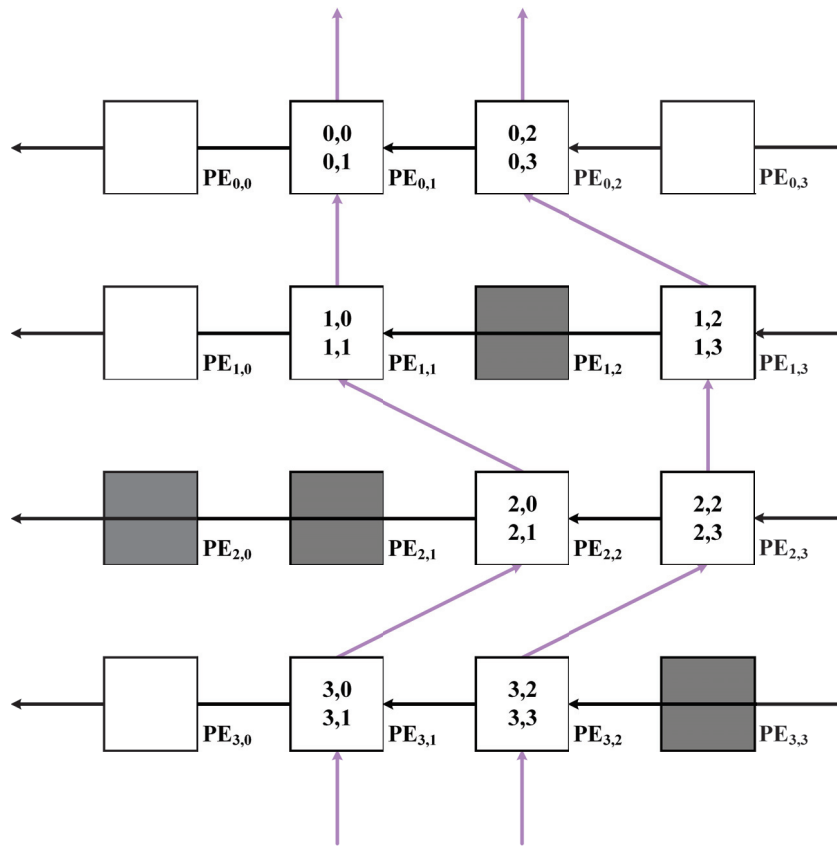
**Figure 3.** (a) The overlap of aligned matrices A and B. (b) The results after the first global matrix shifting.

### 2.3. Fault-Tolerant Matrix Multiplier Design

Jan and Huang [30] studied the fault-tolerant design of a systolic array executing Cannon's algorithm for matrix multiplications. Their approach requires identifying fault-free columns, including those in the form of twisted columns.

Figure 4 is taken as an example. The PEs shaded in gray indicate faulty PEs. There are two twisted columns. One twisted column comprises  $PE_{0,1}$ ,  $PE_{1,1}$ ,  $PE_{2,2}$ , and  $PE_{3,1}$ , while the other twisted column comprises  $PE_{0,2}$ ,  $PE_{1,3}$ ,  $PE_{2,3}$ , and  $PE_{3,2}$ . From Figure 4, it can be observed that  $PE_{0,1}$  performs the tasks that originally belonged to  $PE_{0,0}$  and  $PE_{0,1}$ ,  $PE_{1,1}$

performs the tasks that originally belonged to  $PE_{1,0}$  and  $PE_{1,1}$ ,  $PE_{2,2}$  performs the tasks that originally belonged to  $PE_{2,0}$  and  $PE_{2,1}$ , and so forth.



**Figure 4.** An example of twisted columns.

In [30], it is assumed that there exists a controller, referred to as a host control circuit, capable of directly transmitting data to each PE. This assumption is necessary because the systolic array must have data that correspond to aligned matrices to execute Cannon's algorithm. Moreover, for each PE, the host control circuit needs to manage two multiplexers (in this PE) for data selection.

1. To circumvent faulty PEs when data flows from bottom to top, each PE can transmit data to the upper-left PE, the upper PE, and the upper-right PE, respectively. This means that in each PE, a multiplexer is required to select the data coming from the bottom-left PE, the bottom PE, the bottom-right PE, and the host control circuit.
2. To bypass a faulty PE when data flows from right to left, the data will pass through the faulty PE (without executing any multiply-accumulate operations) and enter the next PE in the subsequent cycle. This implies that in each PE, another multiplexer is necessary to choose the data arriving from the multiply-accumulate unit (MAC unit), the right PE, and the control circuit.

From the above discussion, we find that in the previous work, to support twisted columns, data may come from different directions, making the design of the host control circuit complex and adding extra circuits to each PE. Moreover, the previous work must find a fault-free column, which also limits the capability for fault tolerance.

### 3. The Proposed Approach

With analysis, we observe that in [30], there already exists a data transmission path between the host control circuit and each PE. Therefore, we propose a new fault-tolerant approach that utilizes the existing data transmission path between the host control circuit and each PE for data transfer, eliminating the need to search for a fault-free column. This



approach not only simplifies the host control circuit but also streamlines the data path design of PEs.

It should be noted that our objective is to ensure the accuracy of matrix multiplication results. Therefore, the fault-tolerance issues that we address in matrix multiplication are different from those in the literature on DNN accelerators [31–34], which may tolerate some errors.

In this section, we introduce the proposed approach. Section 3.1 covers our fault-tolerance mechanism while also discussing the corresponding PE architecture. Section 3.2 presents two pair-matching algorithms: one-dimensional pair-matching and two-dimensional pair-matching.

### 3.1. Fault-Tolerance Mechanism and Corresponding PE Architecture

Note that, after post-manufacturing testing [14,15], the host control circuit can establish information about faulty PEs. Then, throughout the circuit's lifespan, the number of faulty PEs may increase as the circuit ages. Hence, the host control circuit can incorporate run-time detection mechanisms [16,17] to dynamically update the information on faulty PEs.

The main drawback of the previous work [30] is the necessity to find at least one fault-free column (even in the form of a twisted column), which limits the potential for fault tolerance. Additionally, to support twisted columns, both the host control circuit and the data path of each PE incur significant additional circuitry, resulting in area overhead.

In contrast to the previous work, we adopt a pair-matching mechanism. For each faulty PE, we assign a fault-free PE to act as its proxy. In other words, this fault-free PE not only needs to complete its own task but also the task of its corresponding faulty PE. The proposed pair-matching mechanism does not require the identification of a fault-free column. Therefore, the proposed approach exhibits higher fault tolerance.

The proposed fault-tolerance mechanism follows the assumptions outlined in [30]. Firstly, we assume that faults in each PE can only occur in the MAC unit and not in the data transmission paths or storage components. Secondly, we assume the existence of data transmission paths between the host control circuit and each PE.

The proposed fault-tolerance mechanism consists of two stages:

1. In the first stage, Cannon's algorithm is executed in the systolic array. Each fault-free PE completes the computations that it should perform. However, each faulty PE does not engage in computations. Nevertheless, during the process of global shifting, both faulty PEs and fault-free PEs engage in data transmissions.
2. In the second stage, the computations that each faulty PE should carry out are completed by the fault-free PE that it is acting as a proxy for. Since there are data transmission paths between the host control circuit and each PE, the data required by each fault-free PE can be directly transmitted by the host control circuit.

To implement the proposed fault-tolerance mechanism, we need to identify a fault-free PE to replace each faulty PE. The pairing of faulty PEs with fault-free PEs is managed by the host control circuit, which takes the matrix size and the list of faulty PEs as input. It then executes the pair-matching algorithm to establish the corresponding pairs. In Section 3.2, we will present two pair-matching algorithms. These two algorithms offer a trade-off between fault tolerance and controller area overhead.

In [30], the authors do not discuss the bandwidth of data transfer between the host control circuit and PEs. Therefore, here, we also assume no limitation on the data transfer bandwidth. It is worth noting that even if the data transfer bandwidth is limited, it does not affect the fault tolerance of our approach. However, if the data transfer bandwidth is limited, the time that it takes for the host control circuit to transmit data to PEs will be longer. For example, if the host control circuit can only transmit data to a maximum of 4 PEs at a time, it will need to transmit data to 8 PEs in two separate transmissions.

We illustrate our fault-tolerance mechanism with an example. In the first stage, the systolic array executes Cannon's algorithm. Initially, based on the results of the initial alignment, the host control circuit sends data to each PE. Figure 5a depicts the overlap between

the aligned matrix  $A$  and the aligned matrix  $B$ . For example,  $PE_{0,0}$  receives data  $A_{0,0}$  and  $B_{0,0}$ ,  $PE_{0,1}$  receives data  $A_{0,1}$  and  $B_{1,0}$ ,  $PE_{0,2}$  receives data  $A_{0,2}$  and  $B_{2,0}$ , and so on. It is worth noting that PEs shaded in gray are faulty PEs. In other words,  $PE_{0,1}$ ,  $PE_{0,3}$ , and  $PE_{3,3}$  are faulty PEs. Faulty PEs are unable to perform MAC operations but can still transmit data.

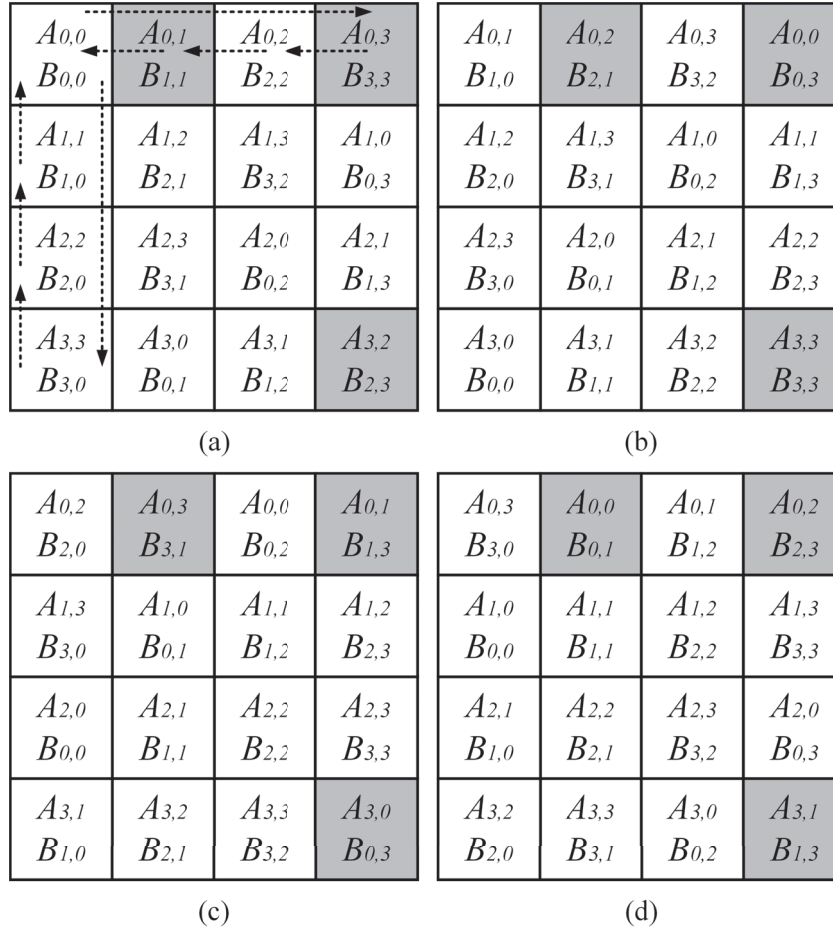
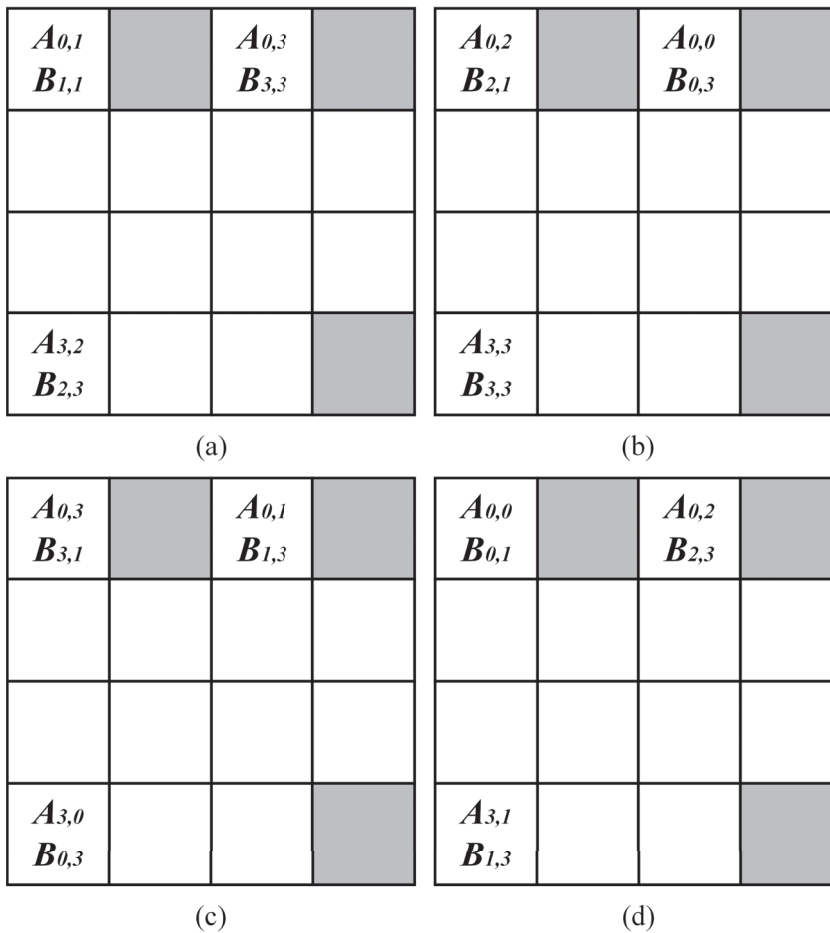


Figure 5. The first stage of our fault-tolerance mechanism.

All fault-free PEs simultaneously perform MAC operations. Then, we conduct the first global matrix shifting. To clarify, in Figure 5a, we use the 0-th row as an example to demonstrate the shift direction of the elements in matrix  $A$  and the 0-th column as an example to demonstrate the shift direction of the elements in matrix  $B$ . The results of data transmission to each PE are shown in Figure 5b. Subsequently, all fault-free PEs simultaneously perform MAC operations again. Then, we conduct the second global matrix shifting, and the results of data transmission to each PE are shown in Figure 5c. Then, all fault-free PEs simultaneously perform MAC operations again. Following this, we proceed with the third global matrix shifting, and the results of data transmission to each PE are shown in Figure 5d. Finally, all fault-free PEs simultaneously perform MAC operations. At this point, each fault-free PE has completed its originally assigned task. For example, the accumulation result of  $PE_{0,0}$  corresponds to the value of  $C_{0,0}$ , where  $C_{0,0} = A_{0,0} \times B_{0,0} + A_{0,1} \times B_{1,0} + A_{0,2} \times B_{2,0} + A_{0,3} \times B_{3,0}$ .

Next, we proceed to the second stage. Based on the outcomes of the pair-matching algorithm, the host control circuit assigns a fault-free PE to manage the tasks of each faulty PE. In Section 3.2, we will introduce two pair-matching algorithms. Here, we assume that the matching pairs are as follows:  $\langle PE_{0,1}, PE_{0,0} \rangle$ ,  $\langle PE_{0,3}, PE_{0,2} \rangle$ , and  $\langle PE_{3,3}, PE_{3,0} \rangle$ , where  $PE_{0,0}$  takes over the task of  $PE_{0,1}$ ,  $PE_{0,2}$  takes over the task of  $PE_{0,3}$ , and  $PE_{3,0}$  takes over the task of  $PE_{3,3}$ .

In the second stage, only fault-free PEs serving as proxies perform MAC operations. Initially, the host control circuit sends data  $A_{0,1}$  and  $B_{1,1}$  to  $PE_{0,0}$ , data  $A_{0,3}$  and  $B_{3,3}$  to  $PE_{0,2}$ , and data  $A_{3,2}$  and  $B_{2,3}$  to  $PE_{3,0}$ . The data transmission results are shown in Figure 6a. Subsequently,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{3,0}$  simultaneously perform MAC operations. Then, the host control circuit sends data  $A_{0,2}$  and  $B_{2,1}$  to  $PE_{0,0}$ , data  $A_{0,0}$  and  $B_{0,3}$  to  $PE_{0,2}$ , and data  $A_{3,3}$  and  $B_{3,3}$  to  $PE_{3,0}$ . The data transmission results are shown in Figure 6b. Afterwards,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{3,0}$  simultaneously perform MAC operations. Next, the host control circuit sends data  $A_{0,3}$  and  $B_{3,1}$  to  $PE_{0,0}$ , data  $A_{0,1}$  and  $B_{1,3}$  to  $PE_{0,2}$ , and data  $A_{3,0}$  and  $B_{0,3}$  to  $PE_{3,0}$ . The data transmission results are shown in Figure 6c. Then,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{3,0}$  simultaneously perform MAC operations. Following that, the host control circuit sends data  $A_{0,0}$  and  $B_{0,1}$  to  $PE_{0,0}$ , data  $A_{0,2}$  and  $B_{2,3}$  to  $PE_{0,2}$ , and data  $A_{3,1}$  and  $B_{1,3}$  to  $PE_{3,0}$ . The data transmission results are shown in Figure 6d. Subsequently,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{3,0}$  simultaneously perform MAC operations. As a result,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{3,0}$  complete their proxy tasks.

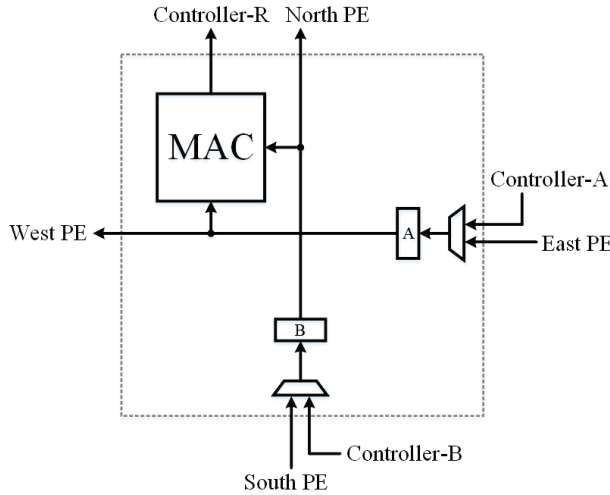


**Figure 6.** The second stage of our fault-tolerance mechanism.

We illustrate with the pair  $\langle PE_{0,1}, PE_{0,0} \rangle$ .  $PE_{0,0}$  acts as the proxy for the tasks of  $PE_{0,1}$ . Therefore, in this stage,  $PE_{0,0}$  computes the value of  $C_{0,1}$ , where  $C_{0,1} = A_{0,1} \times B_{1,1} + A_{0,2} \times B_{2,1} + A_{0,3} \times B_{3,1} + A_{0,0} \times B_{0,1}$ .

Finally, let us discuss the architecture design of each PE. Our PE design is depicted in Figure 7. Essentially, each PE primarily executes MAC operations. For each PE, data for matrix  $A$  can be sourced either from the host control circuit (referred to as *Controller-A* in Figure 7) or from the PE located to the right (referred to as the east PE in Figure 7) because global matrix shifting necessitates shifting each element of matrix  $A$  to the left. Therefore, a two-to-one multiplexer is required to select the data source for matrix  $A$ . Similarly,

for each PE, data for matrix  $B$  can be sourced either from the host control circuit (denoted as *Controller-B* in Figure 7) or from the PE located below (referred to as the south PE in Figure 7) because global matrix shifting requires shifting each element of matrix  $B$  upward. Thus, a two-to-one multiplexer is also necessary to select the data source for matrix  $B$ .



**Figure 7.** The architecture of our PE.

For each PE, due to global matrix shifting requiring each element of matrix  $A$  to shift left by one position, the data of matrix  $A$  need to be transmitted to the left-side PE (referred to as the west PE in Figure 7). Similarly, due to global matrix shifting necessitating each element of matrix  $B$  to shift upward by one position, the data of matrix  $B$  need to be transmitted to the PE above (referred to as the north PE in Figure 7). Upon completing all multiplications and accumulations, the final result is sent back to the host control circuit (referred to as *Controller-R* in Figure 7).

In comparison to the previous work [30], the data path of our PE is relatively simple, resulting in circuit area savings for the PE. Additionally, since our control logic is also relatively simple, the area of the controller (i.e., the host control circuit) can also be saved.

### 3.2. Proposed Pair-Matching Algorithms

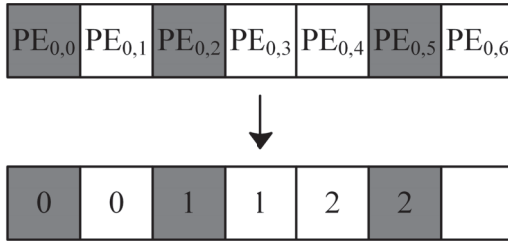
In the proposed fault-tolerance mechanism, for each faulty PE, we must find a fault-free PE to act as its proxy in execution. Therefore, the host control circuit must employ a pair-matching algorithm. Considering hardware efficiency, we use the following two principles to develop the pair-matching algorithms:

1. Each fault-free PE can only proxy for at most one faulty PE. Thus, all tasks of faulty PEs can be executed simultaneously by fault-free PEs acting as their proxies.
2. We perform pair-matching independently for each row (or each column). Hence, pair-matching can be conducted in parallel for all rows (or all columns).

We refer to the scheme of performing pair-matching independently for each row as row-based pair-matching. Similarly, the scheme of performing pair-matching independently for each column is referred to as column-based pair-matching. Essentially, the concepts of row-based pair-matching and column-based pair-matching are identical. Without loss of generality, we illustrate the row-based pair-matching scheme using Figure 8.

For each row, the number of matching pairs is the minimum of the number of faulty PEs and the number of fault-free PEs in that row. In Figure 8, we illustrate this using row 0. Since there are 3 faulty PEs (i.e.,  $PE_{0,0}$ ,  $PE_{0,2}$ , and  $PE_{0,5}$ ) and 4 fault-free PEs (i.e.,  $PE_{0,1}$ ,  $PE_{0,3}$ ,  $PE_{0,4}$ , and  $PE_{0,5}$ ) in this row, the number of matching pairs is 3 (i.e.,  $\min(3,4) = 3$ ). We scan from left to right in our row-based matching approach. The first faulty PE is  $PE_{0,0}$ , and the first fault-free PE is  $PE_{0,1}$ , so the first matching pair is  $\langle PE_{0,0}, PE_{0,1} \rangle$ . The second

faulty PE is  $PE_{0,2}$ , and the second fault-free PE is  $PE_{0,3}$ , so the second matching pair is  $\langle PE_{0,2}, PE_{0,3} \rangle$ . The third faulty PE is  $PE_{0,5}$ , and the third fault-free PE is  $PE_{0,4}$ , so the third matching pair is  $\langle PE_{0,5}, PE_{0,4} \rangle$ .



**Figure 8.** Illustration of the row-based pair-matching scheme.

Based on the row-based pair-matching scheme and the column-based pair-matching scheme, we present two pair-matching algorithms.

1. One-dimensional pair-matching algorithm: This algorithm only executes the row-based pair-matching scheme.
2. Two-dimensional pair-matching algorithm: This algorithm first executes the row-based pair-matching scheme. If there are faulty PEs that fail to complete pairing, this algorithm then attempts pairing using the column-based pair-matching scheme.

Clearly, the control logic of the one-dimensional pair-matching algorithm is simpler, but its fault tolerance is lower. Conversely, the control logic of the two-dimensional pair-matching algorithm is more complex, but its fault tolerance is higher.

Algorithm 1 displays the proposed one-dimensional pair-matching algorithm. For each row, the number of matching pairs corresponds to the minimum value between the number of faulty PEs and the number of fault-free PEs in that row. We define  $rp[i]$  as the number of matching pairs in row  $i$ . Each matching pair is assigned a unique index (starting from 0). We define  $index[i]$  as the index of the first matching pair in row  $i$ . In other words, we have  $index[0] = 0$ . For index  $i$ , where  $i > 0$ , we have Equation (4) as follows:

$$index[i] = \sum_{s=0}^{i-1} rp[s] \quad (4)$$

---

**Algorithm 1** One-Dimensional Pair-Matching

---

```

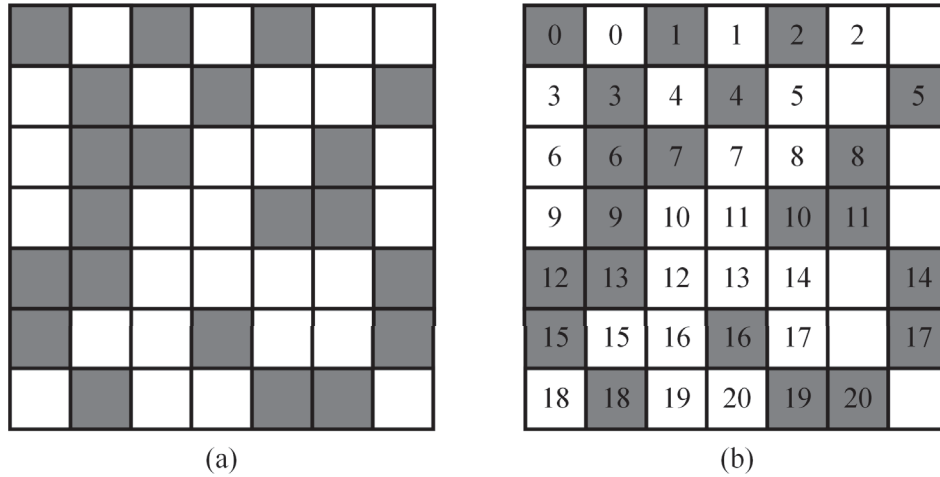
1: for  $i$  from 0 to  $n - 1$  do
2:   for  $j$  from 0 to  $n - 1$  do
3:      $k = index[i]$ ;
4:      $p = 0$ ;  $q = 0$ ;
5:     if  $PE_{i,j}$  is faulty then
6:       if  $p < rp[i]$  then
7:         assign  $PE_{i,j}$  as the faulty PE of  $pair[k + p]$ ;
8:          $p = p + 1$ ;
9:       end if
10:    else
11:      if  $q < rp[i]$  then
12:        assign  $PE_{i,j}$  as the fault-free PE of  $pair[k + q]$ ;
13:         $q = q + 1$ ;
14:      end if
15:    end if
16:  end for
17: end for

```

---

We use the PE array shown in Figure 9a for illustration. In Figure 9a, row 0 has 3 faulty PEs and 4 fault-free PEs. Thus, we have  $rp[0] = 3$ . Since we start indexing from 0,

we have  $index[0] = 0$ . Row 1 has 3 faulty PEs and 4 fault-free PEs. Thus, we have  $rp[1] = 3$ . The index of the first matching pair in row 1 is 3, so  $index[1] = 3$ . Row 2 also has 3 faulty PEs and 4 fault-free PEs. Thus,  $rp[2] = 3$ . The index of the first matching pair in row 2 is 6, so  $index[2] = 6$ .



**Figure 9.** An example of our one-dimensional pair-matching algorithm.

The controller (i.e., the host control circuit) can easily determine the value of  $index[i]$  for each row  $i$ . Since each row knows the index of its first matching pair, in hardware implementation, each row can proceed in parallel using the row-based pair-matching scheme. In Algorithm 1, we employ an array named “pair” to store matching pairs. Each matching pair within this array comprises a faulty PE and a fault-free PE, organized based on their respective indices.

We illustrate the proposed one-dimensional pair-matching algorithm (i.e., Algorithm 1) using the PE array in Figure 9a. From Figure 9a, we have  $index[0] = 0$ ,  $index[1] = 3$ ,  $index[2] = 6$ ,  $index[3] = 9$ ,  $index[4] = 12$ ,  $index[5] = 15$ , and  $index[6] = 18$ . Then, we can perform the row-based pair-matching scheme for each row. As a result, we can obtain the results, as displayed in Figure 9b.

Next, we extend the proposed one-dimensional pair-matching algorithm to a two-dimensional context. Our two-dimensional pair-matching algorithm includes two phases: in the first phase, we apply the row-based pair-matching scheme for each row. The task of this phase is the same as that of our one-dimensional pair-matching algorithm. However, we must mark the PEs that have been paired. In the second phase, we apply the column-based pair-matching scheme for each column. In this phase, we only need to perform pair-matching for unmarked PEs (i.e., PEs that have not yet been paired).

In the second phase, since we only consider unmarked PEs, for each column, the number of matching pairs is the minimum value of the number of unmarked faulty PEs and the number of unmarked fault-free PEs in this column. We define  $cp[j]$  as the number of matching pairs in column  $j$ . We also continue to assign a unique index to each matching pair (continuing from the numbering in the first phase). We define  $col\_index[j]$  as the index of the first matching pair in column  $j$ . In other words, we have Equation (5) as follows:

$$col\_index[0] = \sum_{s=0}^n rp[s] \quad (5)$$

For  $j > 0$ , we have Equation (6) as follows:

$$col\_index[j] = \sum_{s=0}^n rp[s] + \sum_{s=0}^{j-1} cp[s] \quad (6)$$



We illustrate the proposed two-dimensional pair-matching algorithm using Figure 10 as an example. In the first phase, we apply the row-based pair-matching scheme. The results of the first phase are displayed in Figure 10a. Upon completion of the first phase, we have 21 matching pairs (numbered from 0 to 20). However, we find that in column 5, there are still 3 faulty PEs that have not been paired.

0	0	1	1	2		2
3	3	4	4	5		5
6	6	7	7	8	8	
9	9	10	10	11		11
12	13	12	13	14		14
15	15	16	16	17		17
18	18	19	19	20		20

(a)

0	0	1	1	2	21	2
3	3	4	4	5	21	5
6	6	7	7	8	8	
9	9	10	10	11	22	11
12	13	12	13	14	22	14
15	15	16	16	17	23	17
18	18	19	19	20	23	20

(b)

**Figure 10.** An example of our two-dimensional pair-matching algorithm.

In the second phase of the proposed two-dimensional pair-matching algorithm, we only consider unmarked PEs. Thus, we have  $cp[0] = 0$ ,  $cp[1] = 0$ ,  $cp[2] = 0$ ,  $cp[3] = 0$ ,  $cp[4] = 0$ ,  $cp[5] = 3$ , and  $cp[6] = 0$ . Therefore, since we continue numbering from the first phase, we have  $col\_index[0] = 21$ ,  $col\_index[1] = 21$ ,  $col\_index[2] = 21$ ,  $col\_index[3] = 21$ ,  $col\_index[4] = 21$ ,  $col\_index[5] = 21$ , and  $col\_index[6] = 24$ .

Then, we apply the column-based pair-matching scheme for each column. After applying the column-based pair-matching scheme, the three faulty PEs in column 5 can be paired (the indices of these matching pairs are 21, 22, and 23). Upon completion of the second phase, we obtain the results, as displayed in Figure 10b. We find that all of the faulty PEs have been paired.

It is noteworthy that the proposed pair-matching algorithms, including both our one-dimensional and two-dimensional approaches, can be implemented in either software or hardware. However, for real-time applications, it is essential to implement these algorithms in hardware circuits to achieve high speed.

#### 4. Experimental Results

In the experiments, we address the fault-tolerance capability and circuit area overhead of the proposed approach. For comparison, we also implemented the approach presented by Jan and Huang [30].

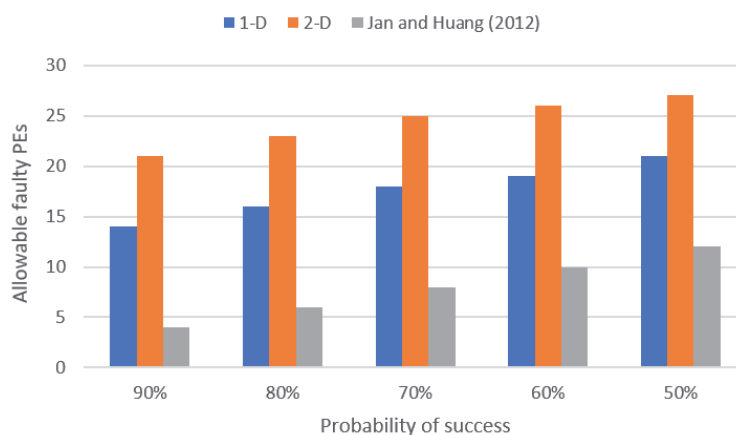
Regarding the fault-tolerance capability, we conducted separate analyses on an  $8 \times 8$  systolic array,  $16 \times 16$  systolic array, and  $32 \times 32$  systolic array. We randomly assume the positions of faulty PEs within the systolic arrays. Then, we can evaluate the fault-tolerance capabilities of different methods (including the proposed one-dimensional pair-matching algorithm, the proposed two-dimensional pair-matching algorithm, and the approach proposed by Jan and Huang [30]).

The detailed analysis methodology is as follows. Given a specific number of faulty PEs, we randomly generate the locations of these faulty PEs. Additionally, for each specific number of faulty PEs, we generate 10,000 cases randomly. For each case, we analyze whether various methods (including the proposed one-dimensional pair-matching algorithm, the proposed two-dimensional pair-matching algorithm, and the approach proposed by Jan and Huang [30]) can successfully tolerate faults. Subsequently, we can calculate the

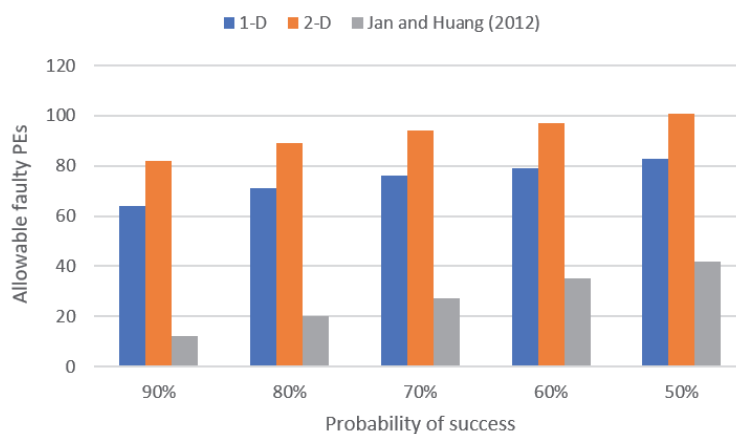
probability of success (i.e., the success rate) of each method concerning a specific number of faulty PEs.

Figure 11–13 depict our analysis results for the  $8 \times 8$  systolic array,  $16 \times 16$  systolic array, and  $32 \times 32$  systolic array, respectively. In these figures, 1-D represents the proposed one-dimensional pair-matching algorithm, 2-D represents the proposed two-dimensional pair-matching algorithm, and *Jan an Huang (2012)* represents the method presented in [30]. Moreover, in these figures, the x-axis represents the probability of success (i.e., the success rate), and the y-axis represents the allowable quantity of faulty PEs. For instance, as shown in Figure 11, to attain a success rate of 90% in an  $8 \times 8$  systolic array, the permissible numbers of faulty PEs for our one-dimensional pair-matching algorithm, our two-dimensional pair-matching algorithm, and the method proposed in [30] are 14, 21, and 4, respectively; as shown in Figure 11, to attain a success rate of 80% in an  $8 \times 8$  systolic array, the permissible numbers of faulty PEs for our one-dimensional pair-matching algorithm, our two-dimensional pair-matching algorithm, and the method proposed in [30] are 16, 23, and 6, respectively.

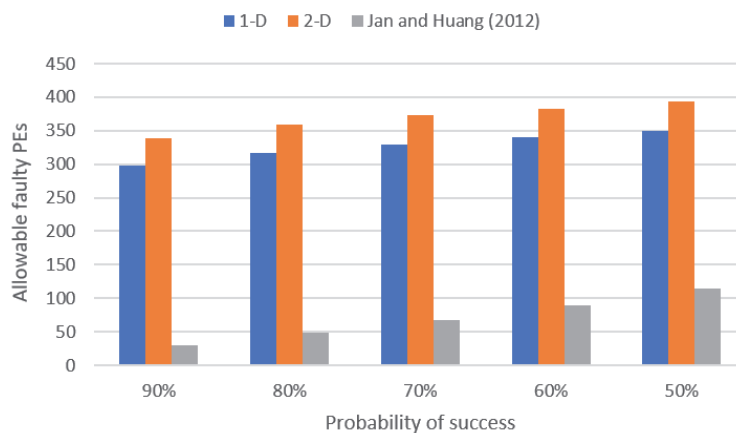
From Figures 11–13, we observe that our two-dimensional pair-matching algorithm exhibits the highest fault-tolerance capability. Furthermore, we also note that both the proposed one-dimensional pair-matching algorithm and the proposed two-dimensional pair-matching algorithm outperform the approach presented in [30] in terms of fault tolerance. Essentially, the approach presented in [30] is only suitable for scenarios with a smaller number of faulty PEs. In contrast, both our one-dimensional pair-matching algorithm and our two-dimensional pair-matching algorithm can be applied in situations with a higher number of faulty PEs.



**Figure 11.** The fault-tolerance capabilities of different methods in an  $8 \times 8$  systolic array. The bar *Jan and Huang (2012)* denotes the method proposed in [30].



**Figure 12.** The fault-tolerance capabilities of different methods in a  $16 \times 16$  systolic array. The bar *Jan and Huang (2012)* denotes the method proposed in [30].



**Figure 13.** The fault-tolerance capabilities of different methods in a  $32 \times 32$  systolic array. The bar *Jan and Huang (2012)* denotes the method proposed in [30].

Next, we explore the circuit area overhead. For our implementations, we assume that the size of the systolic array is  $8 \times 8$ . The circuits are implemented using the TSMC 40 nm cell library. We begin by analyzing the area of the controller (i.e., the host control circuit). Table 1 presents the controller area for executing the proposed one-dimensional pair-matching algorithm (referred to as *1-D*), the proposed two-dimensional pair-matching algorithm (referred to as *2-D*), and the method proposed in [30]. As depicted in Table 1, the controller area utilizing the proposed one-dimensional pair-matching algorithm is the smallest. Conversely, the controller area of the approach proposed in [30] is the largest. This is because their twisted column scheme is more complex than the proposed pair-matching scheme.

**Table 1.** Comparison of the areas of controllers.

Algorithm	Area
1-D	9553 $\mu\text{m}^2$
2-D	11,675 $\mu\text{m}^2$
[30]	14,589 $\mu\text{m}^2$

We also examine the areas occupied by PEs. Table 2 presents the areas of various PE designs, including the conventional PE design (i.e., without any fault-tolerance mechanism), our PE design, and the approach proposed in [30]. In comparison to the conventional PE design, our area overhead is only 6%. It is noteworthy to mention that the conventional PE design lacks a fault-tolerance mechanism. On the other hand, when compared with the conventional PE design, the area overhead of the approach proposed in [30] reaches 70% (owing to the twisted column scheme). Therefore, the area overhead of our PE design is small.

**Table 2.** Comparison of the areas of different PE designs.

PE Design	Area	Normalization
Conventional	434 $\mu\text{m}^2$	100%
Ours	462 $\mu\text{m}^2$	106%
[30]	736 $\mu\text{m}^2$	170%

Note that Table 2 refers to the area of a single PE. When considering a PE array, its total area can be determined by multiplying the area of a single PE by the array size. Therefore, for PE arrays of the same size, compared to utilizing the conventional PE design, the area

overhead of employing our PE design remains at 6%, while the area overhead of utilizing the approach proposed in [30] also stands at 70%.

## 5. Conclusions

This paper introduces a highly fault-tolerant approach designed for a systolic array executing Cannon's algorithm for matrix multiplication. Our core concept involves pairing each faulty PE with a corresponding fault-free PE to serve as its proxy. We propose two pair-matching algorithms: one-dimensional pair-matching and two-dimensional pair-matching. These two pair-matching algorithms offer a trade-off between fault tolerance capability and circuit area overhead. We employed the TSMC 40 nm process technology to implement the proposed approach. The experimental results demonstrate that, compared to the previous work, our approach (whether employing our one-dimensional pair-matching algorithm or our two-dimensional pair-matching algorithm) not only improves fault tolerance but also reduces circuit area overhead. In certain application domains, such as space or deep-sea exploration, repairs are challenging even if many PEs fail. Therefore, the proposed approach is particularly well-suited for these applications.

Since our current pair-matching algorithms necessitate finding a dedicated fault-free PE to substitute for each faulty PE, the number of faulty PEs cannot exceed half of the total PE count. Our future work will concentrate on developing methods to overcome this limitation.

**Author Contributions:** Conceptualization, investigation, and methodology, H.-C.L., L.-Y.S. and S.-H.H.; validation and formal analysis, H.-C.L. and L.-Y.S.; writing—original draft preparation, H.-C.L.; writing—review and editing, L.-Y.S. and S.-H.H.; supervision, S.-H.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported in part by the National Science and Technology Council, Taiwan, under grant number 112-2221-E-033-050-MY3.

**Data Availability Statement:** The data used to support the findings of this study are included in this paper.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Cannon, L. A Cellular Computer to Implement The Kalman Filter Algorithm. Ph.D. Dissertation, Montana State University, Bozeman, MT, USA, 1969.
2. Coppersmith, D.; Winograd, S. Matrix Multiplication via Arithmetic Progressions. In Proceedings of the Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987; pp. 1–6.
3. Kung, H. Why Systolic Architectures? *Computer* **1982**, *15*, 37–46. [CrossRef]
4. Wei, X.; Yu, C.; Zhang, P.; Chen, Y.; Wang, Y.; Hu, H.; Liang, Y.; Cong, J. Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. In Proceedings of the ACM/IEEE Design Automation Conference, Austin, TX, USA, 18–22 June 2017; pp. 1–6.
5. Lahari, P.; Yellampalli, S.; Vaddi, R. Systolic Array Based Multiply Accumulation Unit for IoT Edge Accelerators. In Proceedings of the IEEE International Symposium on Smart Electronic Systems, Jaipur, India, 18–22 December 2021; pp. 220–223.
6. Chipier, D.; Cracan, A.; Andries, V.D. An Overview of Systolic Arrays for Forward and Inverse Discrete Sine Transforms and Their Exploitation in View of an Improved Approach. *Electronics* **2022**, *11*, 2416. [CrossRef]
7. Inayat, K.; Muslim, F.B.; Iqbal, J.; Hassnain Mohsan, S.; Alkahtani, H.; Mostafa, S. Power-Intent Systolic Array Using Modified Parallel Multiplier for Machine Learning Acceleration. *Sensors* **2023**, *23*, 4297. [CrossRef] [PubMed]
8. Aviles, P.; Schäfer, L.; Lindoso, A.; Belloch, J.; Entrena, L. High Complexity Reliable Space Applications in Commercial Microprocessors. *Microelectron. Reliab.* **2022**, *138*, 114679. [CrossRef]
9. Ra, H.; Youn, C.; Kim, K. High-Reliability Underwater Acoustic Communication Using an M-ary Cyclic Spread Spectrum. *Electronics* **2022**, *11*, 1698. [CrossRef]
10. O'Dougherty, P.; Ferrel, K.; Varol, S. A Study of Semiconductor Defects within Automotive Manufacturing using Predictive Analytics. In Proceedings of the IEEE International Symposium on Digital Forensics and Security, Elazig, Turkey, 28–29 June 2021; pp. 1–6.
11. Kundu, S.; Banerjee, S.; Raha, A.; Natarajan, S.; Basu, K. Toward Functional Safety of Systolic Array-Based Deep Learning Hardware Accelerators. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 485–498. [CrossRef]

12. Huang, S.H.; Tu, W.P.; Chang, C.M.; Pan, S.B. Low-power Anti-aging Zero Skew Clock Gating. *ACM Trans. Des. Autom. Electron. Syst.* **2013**, *18*, 27. [CrossRef]
13. Meric, I.; Ramey, S.; Novak, S.; Gupta, S.; Mudanai, S.P.; Hicks, J. Modeling Framework for Transistor Aging Playback in Advanced Technology Nodes. In Proceedings of the IEEE International Reliability Physics Symposium, Dallas, TX, USA, 28 April–30 May 2020; pp. 1–6.
14. Cheong, M.; Lee, I.; Kang, S. A Test Methodology for Neural Computing Unit. In Proceedings of the IEEE International SoC Design Conference, Daegu, Republic of Korea, 12–15 November 2018; pp. 11–12.
15. Solangi, U.; Ibtesam, M.; Ansari, M.; Kim, J.; Park, S. Test Architecture for Systolic Array of Edge-Based AI Accelerator. *IEEE Access* **2021**, *9*, 96700–96710. [CrossRef]
16. Vijayan, A.; Koneru, A.; Kiammehr, S.; Chakrabarty, K.; Tahoori, M. Fine-Grained Aging-Induced Delay Prediction Based on the Monitoring of Run-Time Stress. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 1064–1075. [CrossRef]
17. Huang, K.; Hasan A.M.T.; Zhang, X.; Karimi, N. Real-Time IC Aging Prediction via On-Chip Sensors. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, 7–9 July 2021; pp. 13–18.
18. Bjelica, M.; Mrazovac, B. Reliability of Self-Driving Cars: When Can We Remove the Safety Driver? *IEEE Intell. Transp. Syst. Mag.* **2023**, *15*, 46–54. [CrossRef]
19. Wahba, A.; Fahmy, H. Area Efficient and Fast Combined Binary/Decimal Floating Point Fused Multiply Add Unit. *IEEE Trans. Comput.* **2017**, *66*, 226–239. [CrossRef]
20. Tung, C.W.; Huang, S.H. A High-Performance Multiply-Accumulate Unit by Integrating Additions and Accumulations into Partial Product Reduction Process. *IEEE Access* **2020**, *8*, 87367–87377. [CrossRef]
21. She, X.; Li, N.; Jensen, D. SEU Tolerant Memory Using Error Correction Code. *IEEE Trans. Nucl. Sci.* **2012**, *59*, 205–210. [CrossRef]
22. Wu, M.S.; Chua, Y.L.; Li, J.F.; Chuan, Y.T.; Huang, S.H. Fault-Aware ECC Scheme for Enhancing the Read Reliability of STT-MRAMs. In Proceedings of the IEEE International Test Conference in Asia, Matsue, Japan, 13–15 September 2023; pp. 1–6.
23. Chen, T.J.; Li, J.F.; Tseng, T.W. Cost-efficient Built-in Redundancy Analysis with Optimal Repair Rate for RAMs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2012**, *31*, 930–940. [CrossRef]
24. Lee, H.; Kim, J.; Cho, K.; Kang, S. Fast Built-in Redundancy Analysis Based on Sequential Spare Line Allocation. *IEEE Trans. Reliab.* **2018**, *67*, 264–273. [CrossRef]
25. Johnson, J.; Wirthlin, M. Voter Insertion Algorithms for FPGA Designs Using Triple Modular Redundancy. In Proceedings of the ACM International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2010; pp. 249–258.
26. Santhiya, M.; Saranya, S.; Vijayachitra, S.; Lavanya, C.; Rajarajeswari, M. Application of Voter Insertion Algorithm for Fault Management Using Triple Modular Redundancy (TMR) Technique. In Proceedings of the IEEE International Conference on Intelligent Communication Technologies and Virtual Mobile Networks, Tirunelveli, India, 4–6 February 2021; pp. 578–583.
27. Kim, J.; Reddy, S. On The Design of Fault-Tolerant Two-Dimensional Systolic Arrays for Yield Enhancement. *IEEE Trans. Comput.* **1989**, *38*, 515–525. [CrossRef]
28. Stojanovic, N.; Milovanovic, E.; Stojmenovic, I.; Milovanovic, T.; Tokic, T. Mapping Matrix Multiplication Algorithm onto Fault-Tolerant Systolic Array. *Comput. Math. Appl.* **2004**, *48*, 275–289. [CrossRef]
29. Milovanovic, I.; Milovanovic, E.; Stojcev, M. A Class of fault-Tolerant Systolic Arrays for Matrix Multiplication. *Math. Comput. Model.* **2011**, *54*, 140–151. [CrossRef]
30. Jan, B.Y.; Huang, J.L. A Fault-Tolerant PE Array Based Matrix Multiplier Design. In Proceedings of the IEEE VLSI Design, Automation and Test, Hsinchu, Taiwan, 23–25 April 2012; pp. 1–4.
31. Zhang, J.; Gu, T.; Basu, K.; Garg, S. Analyzing and Mitigating The Impact of Permanent Faults on a Systolic Array Based Neural Network Accelerator. In Proceedings of the IEEE VLSI Test Symposium, San Francisco, CA, USA, 22–25 April 2018; pp. 1–6.
32. Zhang, J.; Basu, K.; Garg, S. Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution. *IEEE Des. Test* **2019**, *36*, 44–53. [CrossRef]
33. Zhao, Y.; Wang, K.; Louri, A. FSA: An Efficient Fault-tolerant Systolic Array-based DNN Accelerator Architecture. In Proceedings of the IEEE International Conference on Computer Design, Olympic Valley, CA, USA, 23–26 October 2022; pp. 545–552.
34. Moghaddasi, I.; Gorgin, S.; Lee, J.A. Dependable DNN Accelerator for Safety-Critical Systems: A Review on the Aging Perspective. *IEEE Access* **2023**, *11*, 89803–89834. [CrossRef]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# New High-Rate Timestamp Management with Real-Time Configurable Virtual Delay and Dead Time for FPGA-Based Time-to-Digital Converters

Fabio Garzetti <sup>†</sup>, Gabriele Bonanno <sup>†</sup>, Nicola Lusardi <sup>\*,†</sup>, Enrico Ronconi, Andrea Costa and Angelo Geraci

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano,  
Via Golgi 40, 20133 Milano, Italy; fabio.garzetti@polimi.it (F.G.); gabriele.bonanno@polimi.it (G.B.);  
enrico.ronconi@polimi.it (E.R.); angelo.geraci@polimi.it (A.G.)

\* Correspondence: nicola.lusardi@polimi.it

<sup>†</sup> These authors contributed equally to this work.

**Abstract:** Modern applications require the ability to measure time events with high resolution, a full-scale range, and multiple input channels. Time-to-Digital Converters (TDCs) are a popular option to convert time intervals into timestamps. To reduce the time-to-market and Non-Recurring Engineering (NRE) costs, a Field-Programmable Gate Array (FPGA) implementation has been chosen. The high number of requested bits and channels, however, gives rise to routing congestion issues when routed in a parallel manner. In this paper, we will propose and analyze a novel solution, the Belt-Bus (BB), which involves a parallel-to-serial conversion of the timestamp stream coming from the TDC while maintaining chronological order and a sufficient high rate, and flagging the presence of timestamp overflow. Moreover, two new useful features are added. The first is a “Virtual Delay” to compensate for offsets due to cable length and FPGA routing path mismatch. The second is a “Virtual Dead-Time” to filter out unforeseen events. Finally, the BB was tested on a Xilinx 28 nm 7-Series Kintex-7 325T FPGA, achieving an overall data rate of 199.9 Msps with very limited resource usage (i.e., lower than a total of 4.5%), consuming only 480 mW in a 16-channel implementation.

**Keywords:** Time-to-Digital Converter (TDC); data serialization; timestamp; Belt-Bus (BS); Field-Programmable Gate Array (FPGA)

## 1. Introduction

The Time-to-Digital Converter (TDC), which assigns a timestamp to an event, is a device used in various commercial and industrial settings, ranging from basic experimental setups to complex research and development projects [1–4]. Prominent examples of its major uses include Time-of-Flight Positron Emission Tomography (TOF-PET) [5] in the biomedical field and Laser Rangefinder [6] techniques for 3D imaging in industry and the automotive field. In the context of time-resolved spectroscopic experiments, TDCs are extensively utilized in academic settings, particularly in techniques like Time-Correlated Single Photon Counting (TCSPC) and the pump-and-probe experiment performed using a Free Electron Laser (FEL) or a synchrotron light [7].

The majority of modern 3D industrial image sensors employ a TDC system to measure the time it takes for a laser pulse to be detected after emission. These sensors, known as Light Detection and Ranging (LIDAR) or Time-of-Flight (TOF) sensors [8,9], find wide applications in areas such as aerial inspection [10] and autonomous driving [6]. Specifically, LIDARs [11] require a TDC with a large number of channels to benefit from a high frame rate, wide field of view, and excellent reliability [12].

Thanks to their cost-effective Non-Recurring Engineering (NRE) expenses, their excellent performance achieved, and their reprogrammable nature, FPGA-based TDC systems



stand out as an optimal solution for fast prototyping, both in the realm of research and in industrial Research and Development (R&D) [13].

All of the modern applications mentioned above, both academic and industrial, require TDCs with a high Full-Scale Range (FSR) and resolution (LSB); thus, with a high number of bits on a significant number of parallel channels and the ability to operate at high rates (e.g., tens of megahertz per channel) [14]. In addition, these timestamps must be processed immediately in real time by various modules working in parallel typically hosted in programmable logic devices like Systems-on-Chip (SoCs) and Field-Programmable Gate Arrays (FPGAs) posing a routing challenge between the TDC and processing modules, both in terms of the congestion of the routing itself (i.e., a high number of required wires) and the potential generation of Cross-Talk (XT) events.

Hence, there is a drive to conceive and develop a parallel-to-serial timestamp data transmission architecture to facilitate the routing of tens of bits (i.e., high FSR and resolution) in multi-channel systems. This is complicated by the need for the serialization process to maintain the chronological order of the timestamps generated by the TDCs while simultaneously managing overflow phenomena, which can significantly impact processing efficiency. As a solution, a novel high-efficiency parallel-to-serial timestamp data transmission architecture protocol based on the AXI4-Stream protocol [15] (also known as AXIS), named Belt-Bus (BB), has been fully developed and validated as an IP-Core in TDC architectures implemented in FPGAs. It is worth noting that this architecture is equally suitable for implementation in ASICs.

This paper is structured as follows: After a description of TDCs and an overview of multi-channel system interconnection, Section 3 describes the proposed protocol and structure, while Section 4 addresses the main issues and their respective resolutions. The final structure is outlined in Section 5. Characterization in terms of area occupancy, power dissipation, and performance along with measurements conducted on a 16-channel TDC implemented in a Xilinx 28 nm 7-Series Kintex 325-T FPGA, is presented in Section 6.

## 2. Time-to-Digital Converter

In Section 2.1, the main Figures-of-Merit (FoMs) of the TDC will be summarized; additionally, in Section 2.2, the issue of connections for multi-channel TDC systems will be illustrated, with a related overview of the state of the art.

### 2.1. Backgrounds

In the scientific literature and in the industrial field, there are various architectures of TDCs implementable in both ASICs [16] and FPGAs [13]. Regardless of the type of structure, a TDC assigns a timestamp, referring to the clock with which the TDC is powered, to the occurrence of a low-high and/or high-low transition on the inputs. Being a digital device, in addition to the temporal reference for the timestamp, the clock of the TDC serves to manage the internal logic. Regardless of the architecture and their implementation in programmable logic (i.e., FPGA, SoC) or ASIC, TDCs are characterized by the following FoMs:

- Resolution or LSB: the smallest time interval that can be accurately measured.
- Precision or Jitter: variation in the output timing accuracy of the TDC.
- Linearity: the degree to which the digital output is proportional to the input time interval, expressed as Differential and Integral Non-Linearity (DNL and INL).
- Full-Scale Range (FSR): the maximum time interval measurable without encountering overflow issues.
- Frequency of Overflow ( $f_{ovfl}$ ).
- Number of bits ( $N_{bit}$ ).
- Number of channels operating in parallel ( $N_{CH}$ ).
- Dead Time (DT): the time that elapsed between two successive measurements on the same channel.

- Maximum Channel Rate (R): the maximum rate of measurements that a single channel can perform.
- Maximum Output Data Rate (ODR): the maximum rate of output processed timestamps.
- Area Occupation: physical size (for ASIC) or number of resources (for ASIC and FPGA) occupied.
- Power Consumption: the amount of power consumed.

From these FoMs, we can easily calculate some relationships that exist among them, such as the connection between FSR, LSB, and  $N_{bit}$  (1), between FSR and  $f_{ovfl}$  (2), between FSR,  $N_{bit}$ , and LSB (3), and the obvious inequalities that link DT with R (4), and ODR with R and  $N_{ch}$  (5).

$$N_{bit} = \log_2(FSR/LSB) \quad (1)$$

$$f_{ovfl} = 1/FSR \quad (2)$$

$$FSR = 2^{N_{bit}} \cdot LSB \quad (3)$$

$$R \leq 1/DT \quad (4)$$

$$ODR \leq N_{CH} \cdot R \quad (5)$$

## 2.2. Multi-Channel Connection Issues and State of the Art

Considering a multi-channel TDC, regardless of its architecture and implementation (e.g., FPGA/SoC vs ASIC), different solutions for timestamp read-out can be employed: serial or parallel. The difference between the two approaches lies in the fact that in the parallel read-out, each channel has a dedicated output line for the timestamp, whereas in a serial solution, there exists an arbitration mechanism for serialization. The adopted approach is relatively insignificant if the number of channels is low but becomes crucial for systems with eight or more channels.

If we consider a parallel read-out approach with a high number of implemented channels, we will have a total of  $N_{CH} \times N_{bit}$  lines to route within our device. This creates substantial internal congestion that severely limits place and route operations. Furthermore, this solution is inconvenient if the information needs to be transferred externally, as it would require a package with a high number of pins. Moreover, managing a large number of lines further increases the likelihood of generating XT events that interfere with sensitive parts of the circuitry. For this reason, an output serialization mechanism is incorporated into TDCs with a high number of channels. The effectiveness of such a circuit will significantly impact various FoMs of the TDC, such as the ODR, DT, R, area occupancy, and power consumption.

Indeed, a non-optimized and simpler serialization and sorting mechanism such as a round-robin (e.g., Timepix3 ASIC-TDC) algorithm performs well in terms of the ODR but not in terms of area and power consumption [17,18]. In this context, a system with  $N_{CH}$  channels requires, approximately, a multiplexer with  $N_{CH}$  inputs (i.e.,  $N_{CH}$ -to-1 MUX), whose area occupation and power dissipation exponentially increase with  $N_{CH}$  [3]. On the other hand, there are serialization systems that, to keep power consumption and area low, rely on memories (e.g., PicoTDC) that record all timestamps for a certain acquisition time and then serially output them. Some of them, however, have high DTs and low rates (e.g., PETsys) [19,20], while others output timestamps without any order and sorting [21,22], requiring an additional processing stage downstream of the TDC if real-time processing is required by various modules working in parallel, such as histograms, counters, and coincidence detectors [4].

The proposed BB solution consists of an innovative serialization structure based on timestamp sorting through comparison, similar to what happens in round-robin. However, the distinctive feature is the distribution of the comparison process on 2-to-1 MUX distributed within the  $N_{CH}$  nodes. This allows for high efficiency in terms of area occupancy and power consumption that scale linearly with  $N_{CH}$ . Furthermore, the presence of memories and pipeline structures enables high data acquisition rates (ODR) to be achieved without compromising DT and R.

### 3. The Belt-Bus

In Section 3.1, the BB protocol is explained; the operating principles are described in Section 3.2, while a detailed logical description of the functioning is presented in Section 3.3, analyzing the submodules. The area occupancy and power dissipation of each submodule are presented in Section 3.4, with the Xilinx 28 nm 7-Series Kintex 325-T FPGA used as a case study.

#### 3.1. Protocol

The BB is a synchronous bus based on AXI4-Stream and utilizes only the TVALID, TREADY, and TDATA signals. As a convention, a logical one for both TVALID and TREADY signifies a valid TDATA. The TDATA signal, as illustrated in Figure 1, comprises three portions: the Timestamp (TS) field with an obvious dimension of  $N_{bit}$ , a 2-bit wide Function Identifier (FID) field, and the Number of Channel (NUM\_CH) field. The latter field has a non-defined a priori dimension to appropriately accommodate the number of channels involved in the measurement (i.e.,  $\log_2(N_{CH})$ ), representing the channels' numerical value.

The architecture of the BB was also designed to address the main issues related to the operation of the TDC without modifying the number of bits of the timestamps, specifically addressing overflow concerns. This was achieved through the utilization of the FID and TS, providing information to downstream modules about particular characteristics deemed useful in subsequent processing.

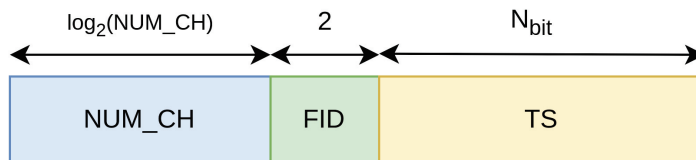
In the implementation presented here, the FIDs are coded as follows:

- FID = 00: overflow event (in TS the overflow value is sent);
- FID = 01: timestamp coming from a rising edge event;
- FID = 10: unused;
- FID = 11: timestamp coming from a falling edge event.

Each time an overflow occurs, a new frame with FID = 00 is injected into the BB. Now the FSR can be increased by a factor  $2^{N_{bit}}$  from  $2^{N_{bit}} \cdot LSB$  up to  $2^{N_{bit}} \times (2^{N_{bit}} \cdot LSB)$ . However, this improvement comes at a cost to the Output Data Rate (ODR), as an overflow event must be sent once every  $2^{N_{bit}} \cdot LSB$  instead of the current timestamp. With the overflow frequency denoted as  $f_{ovfl} = 1/(2^{N_{bit}} \cdot LSB)$  and  $f_{CLK,BB}$  as the clock frequency of the BB, the rate is given by:

$$ODR = f_{CLK,BB} - N_{CH} \cdot f_{ovfl} = f_{CLK,BB} - N_{ch} \frac{1}{2^{N_{bit}} \cdot LSB} \quad (6)$$

Equation (6) shows that there is a trade-off regarding this aspect. By decreasing  $N_{bit}$ , the ODR also decreases. It is important to notice that this trade-off is heavier as the number of channels increases. However, this depends on the used FPGA's size. With larger and more complex FPGAs, routing issues can be minimized, enabling a slight increase in  $N_{bit}$  and, in most cases,  $f_{CLK,BB}$ , thus enhancing the available data rate.

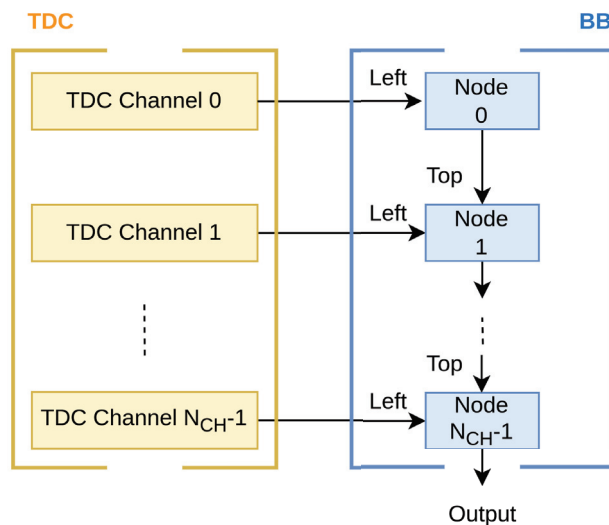


**Figure 1.** Fields of BB in TDATA signal.

#### 3.2. Principle of Operation

With each node representing a single channel, the BB is composed of a cascade of nodes (light blue in Figure 2) that serialize, in a pipeline way, the timestamp coming from TDC channels (yellow in Figure 2) in a chronological sequence. Every node has two inputs in BB protocol: the output of the preceding node, the “Top” port in Figure 2, and the current channel, the “Left” port in Figure 2. The only restriction on the number of channels that

may be added with this chain arrangement is the amount of hardware that can be used by implementing the nodes and the constraint on the average channel rate (i.e., the ratio between the ODR and the total number of channels).



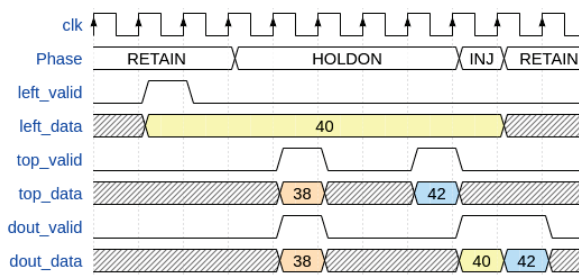
**Figure 2.** Structure of the Belt-Bus.

When a timestamp arrives at a node from the channel entrance (also known as “Left” port), it propagates through subsequent nodes to the terminal one. While timestamps can be arranged chronologically within a single channel, as they move through the chain of nodes, that arrangement may be lost. To prevent this, based on which of the two timestamps is temporally earlier, each node decides whether to prioritize the input from the channel (also known as “Left”) or that from the previous node (also known as “Top”). It might not be feasible to compare the input timestamp in the present node, though, because of potential delays in measurements on channels connected to earlier nodes and, consequently, the absence of the comparative timestamp. In this instance, the current timestamp could be propagated without adhering to the chronological order. Of course, the first node allows only the injection of the “Left” signal, so it possesses the “Top” signal with TVALID hardcoded to “0”.

In order to address this problem, there are four phases involved in obtaining the timestamp from the input channel through the node. Considering the contextualization of their dynamics in the architecture described in the next paragraph, from the perspective of their respective functions, these phases are, as follows, the:

1. **Retain Phase:** the timestamp from the TDC channel (also known as “Left” port) is blocked for a proper time at the input of the node in order to compensate for the pipeline introduced by the registers and FIFO present on the previous nodes.
2. **Hold-on Phase:** the timestamp from the TDC channel (also known as “Left” port) waits for a timestamp from the previous node for comparison for a finite time. If this occurs, the older timestamp is propagated at the node output.
3. **Inject Phase:** if the timestamp from the previous node (also known as “Top” port) is not valid or older, the timestamp from the channel (also known as “Left” port) is propagated at the node output.
4. **Discard Phase:** the timestamp from the TDC channel (also known as “Left” port) is simply discarded (not propagated in BB) because there is no propagation permission in the node chain within a finite time (for instance, if the chain were full).

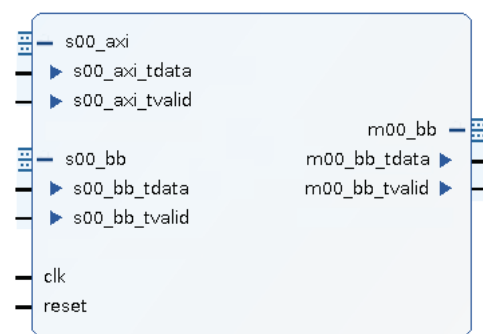
For example, a graphical view of the operation of these phases on three timestamps is shown in Figure 3.



**Figure 3.** Timing diagram showing the phases through which the timestamps 38, 40, and 42 enter the node chain.

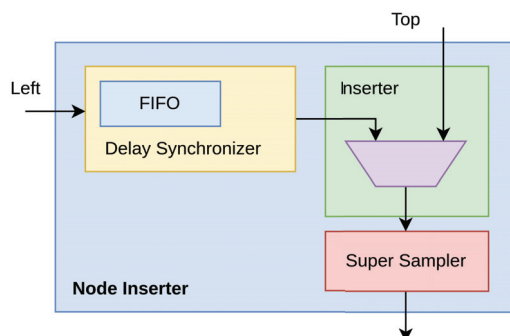
### 3.3. Architecture

The designed architecture of the BB was implemented on a Xilinx 28 nm 7-Series FPGA as an IP-Core and constituted by a cascade of stages called Node Inserters (Figure 4).



**Figure 4.** Node Inserter IP-Core; m00\_bb is the output port and s00\_axi and s00\_bb are the “Left” and “Top” input ports, respectively.

- With reference to Figure 5, three components go into making up each Node Inserter; i.e.,
- The Delay Synchronizer implements the Retain Phase shown in Section 3.2.
  - The Inserter is driven by logic that, using the information from the Delay Synchronizer, generates the selection signal for a multiplexer between the timestamp from the current channel (also known as “Left” port) and the one coming from the previous node (also known as “Top” port). Thus, it implements the Hold-on, Inject, and Discard Phases shown in Section 3.2.
  - The Super Sampler, which is a register that propagates the selected input to the output, ensuring the ready–valid handshake proper to the AXI4-Stream protocol without losing a clock cycle.



**Figure 5.** Top level block schematic of Node Inserter structure with submodules.

The Delay Synchronizer makes the current timestamp from the TDC channel (also known as “Left” port) comparable to the one coming from the preceding node (also known as “Top” port). First of all, the incoming timestamp from the TDC channel (also known

as “Left”) enters into a synchronous First-In First-Out (FIFO) clocked at  $f_{CLK,BB}$  hosted in the Synchronizer. The validity of the output data is deasserted, preventing its propagation, until the propagation time through the FIFO has elapsed (i.e., Retain Phase). The TREADY signal at the “Top” interface allows the information to be stored in the FIFOs and registers of the previous nodes, thus avoiding the presence of an additional FIFO.

Due to the potential for skew and jitter phenomena when signals spread over a wide region, from a timing perspective, a highly intricate and sophisticated data management system is required inside the Delay Synchronizer to ensure that timestamp values can be compared. Thus, the Inserter module behaves as a multiplexer driven by logic that compares the current timestamp (also known as “Left” port) with the one coming from the preceding node (i.e. “Top” port) based on information returned by the Delay Synchronizer and moves ahead with overflows with the highest possible priority, followed by timestamps from the oldest to the newest. If the bandwidth saturates, newer timestamps are discarded. In detail, in the Inserter, if an overflow condition is communicated from the timestamp at the output of the Delay Synchronizer (also known as “Left” port) or from an older timestamp present from the previous node (also known as “Top” port), the incoming timestamp (i.e., the output of the Delay Synchronizer) is propagated forward. The assessment of this condition continues until the timeout, equal to an interval comprising the clock jitter, skews, and the delays of the pipeline stages that constitute the implementation (i.e., Hold-on Phase). At the timeout of the Hold-on Phase, if the node’s bus is ready to receive, the timestamp is propagated (Inject Phase); otherwise, it is discarded, allowing for a more recent timestamp to be placed at the FIFO output (Discard Phase).

### 3.4. Area Occupancy and Power Dissipation

The area occupation of the Node Inserter and its related submodules is a function of the number of bits in the TS fields (i.e.,  $N_{bit}$ ). Table 1 presents the area occupancy in terms of Carry Logic (CARRY), Look-Up Tables (LUT), Flip-Flops (FF), and Look-Up Table RAM (LUTRAM) occupied. No resources in terms of Digital Signal Processor (DSP) modules and Block RAM (BRAM) are utilized. Additionally, the same table provides information on power dissipation (only dynamic considering that the power dissipated by the module is primarily of a dynamic nature), considering a maximum clock frequency of 130 MHz.

**Table 1.** Area occupancy and power dissipation of Node Inserter and its related submodules clocked at 130 MHz, as presented in Section 3.

$N_{bit}$	Module/Submodules	Power [mW]	CARRY	LUT	FF	LUTRAM
16	Node Inserter	7	8	137	148	20
	Delay Synchronizator	5		126	64	20
	Inserter	1	8	2	2	
	Super Sampler	1		9	82	
24	Node Inserter	7	12	149	180	24
	Delay Synchronizator	5		139	114	24
	Inserter	1	12	2	2	
	Super Sampler	1		8	64	
32	Node Inserter	9	16	169	212	28
	Delay Synchronizator	7		158	128	28
	Inserter	<1%	20	2	2	
	Super Sampler	2		9	82	



Table 1. Cont.

$N_{bit}$	Module/Submodules	Power [mW]	CARRY	LUT	FF	LUTRAM
40	Node Inserter	10	20	234	244	36
	Delay Synchronizator	8		223	144	36
	Inserter	<1%	20	2	2	
	Super Sampler	2		9	98	
48	Node Inserter	11	24	264	276	40
	Delay Synchronizator	9		253	150	40
	Inserter	<1%	24	2	2	
	Super Sampler	2		9	124	
56	Node Inserter	13	28	292	308	44
	Delay Synchronizator	9		281	176	44
	Inserter	1	28	2	2	
	Super Sampler	2		9	130	
64	Node Inserter	14	32	324	340	48
	Delay Synchronizator	10		313	189	48
	Inserter	1	32	2	2	
	Super Sampler	3		9	149	

#### 4. Main Issues and Solutions

As presented in Section 3, the BB also has two limitations.

The principal issue is that the present structure does not fully account for the uncertainty of the timestamp arrival time. In fact, two timestamps produced from distinct channels during the same TDC clock cycle can arrive at the Node Inserter at different times. This is particularly true at high channel rates. The primary cause of this is the needs of asynchronous FIFOs to accommodate the Clock Domain Crossing (CDC) between the clock of the TDC ( $f_{CLK,TDC}$ ) and the clock of the BB ( $f_{CLK,BB}$ ). This establishes the likelihood of unordered timestamps in specific scenarios. The solution to these issues is addressed in Section 4.1.

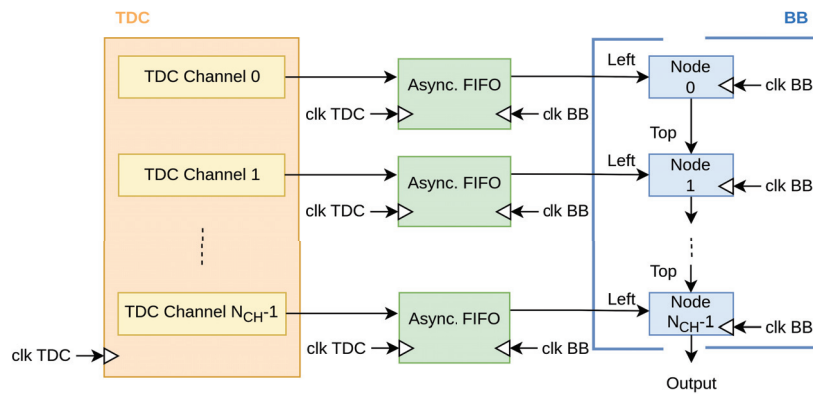
Another issue is that, considering the Xilinx 28 nm 7-Series FPGAs as a technological node, due to the architecture presented in Section 3, the maximum available BB clock frequency is not very high, about 130 MHz (i.e.,  $f_{CLK,BB} < 130$  MHz), which corresponds to only 16.25% of the maximum clock frequency that these technological nodes support (i.e., 800 MHz). The cause of these issues is analyzed and discussed in Section 4.2. Thanks to these two modifications, a frequency of 200 MHz (25% of the maximum available) can be achieved. The area occupancy and power dissipation of each submodule are presented in Section 4.3.

##### 4.1. Unsorted Timestamps Issue

The first issue that has been addressed is related to the presence of different clock domains, which require asynchronous FIFOs between the TDC and BB as a CDC (Figure 6).

Under this condition, there are two further causes of timestamp unsorting. The first one, deterministic, is due to the different clock frequencies in case one channel has a high timestamp rate and another has a lower one; a timestamp entering in the first asynchronous FIFO can exit from it in a different time instant and so is injected late in the BB with respect to the other one in a less crowded channel since the data already stored in the asynchronous FIFO must exit first.

The second one is non-deterministic and is due to unpredictable CDC propagation delay. To better understand this, let us focus briefly on how a CDC works in the following subparagraphs.

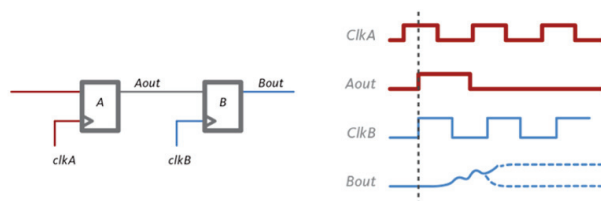


**Figure 6.** Connection between the TDC (orange), clocked at clk TDC, and the BB (blue), clocked at clk BB, is established using an asynchronous FIFO (Async. FIFO, green) employed as CDC.

#### 4.1.1.1. CDC Uncertainty

In a basic CDC circuit, the simplest 1-bit two-stage architecture can be considered (Figure 7). In the first stage (flip-flop A), data are captured by a register in the source clock domain on the rising edge of the source clock (i.e., clk A). In the second stage (flip-flop B), the captured data are then transferred to a register in the destination clock domain on the rising edge of the destination clock (i.e., clk B). The clock uncertainty in this circuit arises because the rising edges of the source and destination clocks may not be perfectly aligned in time due to factors such as clock skew, jitter, or delay. As a result, data may be captured by the source register at a slightly different time than they are transferred to the destination register.

As consequence of that, the sampling register could enter into metastability [23]; on average, each Mean Time Between Failures (MTBF) given by the relation  $MTBF = \frac{f_r}{t_0 \cdot f_{CLK,A} \cdot f_{CLK,B}}$ , where  $f_r$  is a parameter that depends on the flip-flop used,  $t_0$  is a constant related to the width of the time window or aperture wherein a data edge triggers a metastable event,  $f_{CLK,A}$  is the source clock domain frequency, and  $f_{CLK,B}$  is the destination clock domain frequency. To quickly exit a possible metastability transient, the well-established cascade of registers must be added. Now, if, for example, two registers are put in cascade, there is not only one clock uncertainty due to sampling but another one, with lower probability, needed from the first register to recover from metastability. This uncertainty increases as the number of cascaded registers grows. Even in an asynchronous FIFO, if the two clocks are not derived from the same source (e.g., a divided clock), similar mechanisms are used internally, giving rise to a temporal uncertainty at the FIFO output.

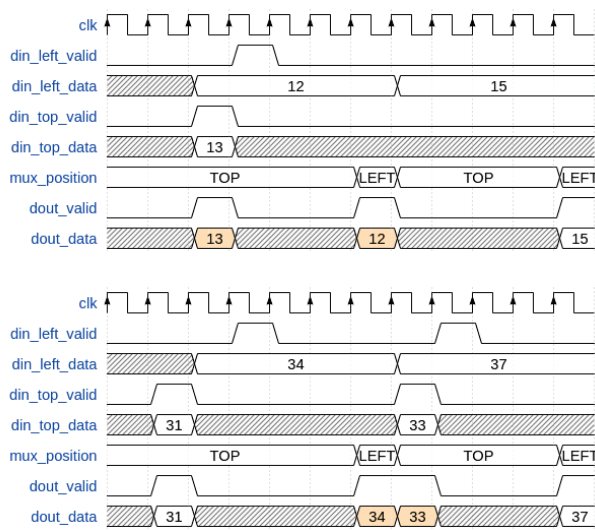


**Figure 7.** Basic Clock Domain Crossing structure and waveforms.

#### 4.1.1.2. Issue Evidence

By examining simulated waveforms focusing on two channels for simplicity, two situations emerge, resulting in unsorted timestamps. Figure 8 shows these two situations. In the first case (on the top side of Figure 8), timestamps “12” and “13” are sent to the “Left” port of the relative Node Inserter at the same instant, but due to the CDC issue, timestamp “13” is read before timestamp “12”, resulting in an unordered timestamp error on BB. In the second case (on the bottom side of Figure 8), timestamp “33” stays, due to CDC uncertainty,

in the asynchronous FIFO for more time compared to timestamp “34”, causing unordered issues at the output.



**Figure 8.** Waveforms with CDCs modeled.

#### 4.1.3. Issue Solving

In order to mitigate this failure, two modifications have been introduced in the Node Inserter.

Since the timestamp entering the Node Inserter cannot be injected until the Hold-on Phase is active unless a newer timestamp reaches the “Top” port, the Retain Phase on the Delay Synchronizer is increased by a value larger than the time uncertainty introduced by the asynchronous FIFO used as a CDC, allowing the data to be properly compared by the Inserter. This way, the issue arrived on the top in Figure 8 is solved.

The second modification to the Inserter is mandatory to solve the issues present on the bottom of Figure 8. Instead of sending data to the Super Sampler when the Inject phase begins, another check is performed by simply waiting some clock cycles after the timestamp coming from the previous node is propagated. In this way, when the bus is full, the data comparison is always performed, avoiding unsorted timestamps. The number of cycles to wait is proportional to the node number to compensate for pipelines and the ratio between  $f_{CLK,TDC}$  and  $f_{CLK,BB}$  (where  $f_{CLK,TDC} > f_{CLK,BB}$ ). No wait cycles are requested if  $f_{CLK,TDC} < f_{CLK,BB}$ .

#### 4.1.4. Order Checker

After the modifications introduced in Section 4.1.3, the chronological order issue becomes very rare and thus almost negligible: fewer than one in a billion samples (i.e.,  $1 \times 10^{-9}$ ). This residual error is due to the stochastic nature of the CDC (i.e., MTBF), especially when the number of channels is high, and events occur randomly and at a high rate; occasionally, unsorted timestamps may be present. A possible way to solve the problem could be to increase the asynchronous FIFO depth inside the Delay Synchronizer excessively, leading to area occupancy problems in the FPGA. To avoid this issue, given the very low probability of encountering unsorted data, these instances are simply discarded, resulting in a negligible loss.

To perform the chronological order check, another IP-Core has been developed, the Order Checker, which takes as input the data from the last Node Inserter and checks timestamp sorting, deasserting the validation if incoming data do not respect this condition.

As can be seen from Figure 9, the order checker has AXI4-Stream input (i.e., s00\_bb in Figure 9) and AXI4-Stream output (i.e., M00\_bb in Figure 9) for BB data, along with an AXI4 Memory-Mapped port to read out the number of unsorted timestamps, solely for debugging purposes (i.e., S00\_axi in Figure 9). This module checks the incoming timestamp

and compares it with the already stored one: if the new one is more recent, the data are propagated and replace the already stored one; if it is older, the valid signal is deasserted, and the relative counter is incremented by one. If no data are present, i.e., the module has been initialized, the first timestamp is stored and propagated.

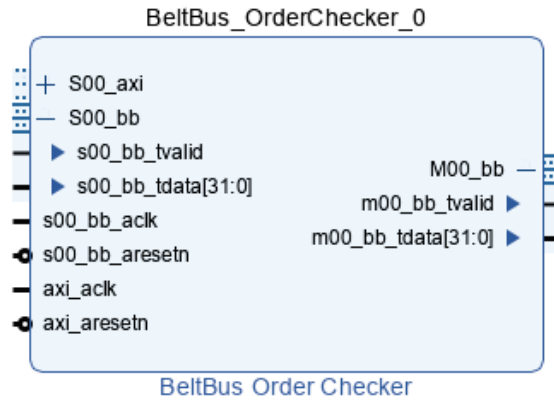


Figure 9. Order Checker IP-Core.

#### 4.2. Limited Output Data Rate Issue

By implementing the Node Inserters in Vivado, considering the Xilinx 28 nm 7-Series FPGAs as a technological node, it can be clearly seen that the maximum clock frequency is limited by two different sources.

The main one comes from the way of performing the comparison (i.e., the symbols “>” in VHDL code) between the two timestamps entering the node (i.e., those coming to the “Left” and those coming from the “Top” ports), which requires, by the default encoding performed by Vivado, the computation of two subtractions and an unsigned comparison. This results in a very high requirement in terms of logic resources, mainly LUTs and CARRY, because the number of bits of the timestamps is high (e.g., 32 to 64). The intervention carried out to increase the maximum clock frequency was replacing the comparison operation (i.e., the symbols “>” in VHDL code) with a simpler one. Only a signed subtraction between the timestamps (i.e., “Left” minus “Top” in VHDL code) is performed, and then, a check is performed on the sign of the result. If the result is positive, the “Top” timestamp is older (i.e., “Left” is bigger than “Top” so more recent in time) and has the priority; otherwise, the “Top” data are propagated. Since performing the sign check is enough to observe the MSB of the result, the number of Carry Logic decreases by a factor of two, as they are only needed to perform one operation instead of three.

The second improvement can be introduced by replacing the Super Sampler with a more efficient pipelined structure, called AXIS Register Slice, that occupies the same hardware resources. The working principle is similar to having two-slot FIFOs. The data entering the module are stored in the output register if nothing is already stored in it. Thanks to these two modifications, a frequency of 200 MHz (20% of the maximum available) can be achieved.

#### 4.3. Area Occupancy and Power Dissipation

The area occupation of the Node Inserter and its related submodules (with the modification proposed in this Section) is a function of the number of bits in the TS fields (i.e.,  $N_{bit}$ ). Table 2 presents the area occupancy in terms of CARRY, LUT, FF, and LUTRAM occupied. No resources in terms of DSP and BRAM are utilized. Additionally, the same table provides information on power dissipation, considering a maximum clock frequency of 200 MHz. Comparing Table 2 to Table 1, it is possible to observe a similar occupation and an increase by a factor of two in the CARRY occupied by the Inserter, along with the replacement of the Super Sampler with the AXIS Register Slice. Moreover, a higher usage of LUTs and FFs is

observed in the Delay Synchronizer to address the issues outlined in Section 4.1. The higher power dissipation is attributed to a higher clock frequency (200 MHz instead of 130 MHz).

**Table 2.** Area occupancy and power dissipation of Node Inserter and its related submodules presented in Section 4.

$N_{bit}$	Module/Submodules	Power [mW]	CARRY	LUT	FF	LUTRAM
16	Node Inserter	10	4	128	192	20
	Delay Synchronizator	8		127	108	20
	Inserter	<1%	4	2	2	
	AXIS Register Slice	2		9	82	
24	Node Inserter	10	6	150	240	24
	Delay Synchronizator	8		140	174	24
	Inserter	1	6	2	2	
	AXIS Register Slice	1		8	64	
32	Node Inserter	13	8	170	288	28
	Delay Synchronizator	10		159	204	28
	Inserter	1	8	2	2	
	AXIS Register Slice	2		9	82	
40	Node Inserter	15	10	236	336	36
	Delay Synchronizator	12		224	236	36
	Inserter	<1%	10	2	2	
	AXIS Register Slice	3		9	98	
48	Node Inserter	17	12	265	384	40
	Delay Synchronizator	13		254	258	40
	Inserter	<1%	12	2	2	
	AXIS Register Slice	4		9	124	
56	Node Inserter	19	14	293	432	44
	Delay Synchronizator	14		282	300	44
	Inserter	1	14	2	2	
	AXIS Register Slice	4		9	130	
64	Node Inserter	21	16	325	480	48
	Delay Synchronizator	16		314	329	48
	Inserter	<1%	16	2	2	
	AXIS Register Slice	5		9	149	

## 5. Main New Features

The description of two new features, the Virtual Delay in Section 5.1, and Virtual Dead Time in Section 5.2, is the purpose of this section. These two improvements are performed with very careful attention to the timing analysis to ensure a maximum clock frequency of 200 MHz, as described in Section 4.2. Lastly, an overview of the complete structure of the BB is shown in Section 5.3. The area occupancy and power dissipation of each submodule are presented in Section 5.4

### 5.1. Virtual Delay

In many applications, only relative times must be measured by computing differences between different channels. For this reason, static offset compensation can be very useful, for example, to have the resulting histogram centered at zero. In order to perform this task, a Virtual Delay feature has been developed in the BB. In this way, the timestamps coming out from the Node Inserters are not only chronologically ordered but also translated in time. This allows compensating offsets due to both different cable lengths and FPGA routing path mismatches between TDC channels.

### 5.1.1. Architecture

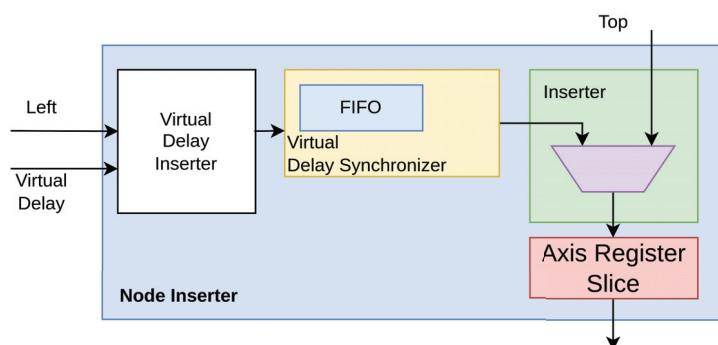
A simple summing of the incoming timestamp delay is not sufficient to accomplish this feature. Indeed, if some synchronization mechanism is not present, the BB would “brake”, leading to unsorted timestamps. As will be explained later, since synchronization requires memory, the Virtual Delay cannot reach very high values (i.e., up to  $2^{20} \times \text{LSB}$ ), otherwise, the resource usage in the FPGA would be enormous. On the other hand, since the static offset due to FPGA routing is a few tens of picoseconds, considering an LSB of tens of femtoseconds, implementing a maximum delay in the order of hundreds of nanoseconds, which is quite feasible, would be enough. For example, by approximating the speed of signals at 30 cm/ns, a 1  $\mu\text{s}$  delay would be sufficient to compensate for a 300 m cable length offset, which is a very high value. This is the main reason because the maximum delay value is less than or equal to the maximum timestamp value.

However, this is not the only thing that must be managed very carefully. When summing a value to a timestamp, the result can be larger than the maximum value of  $2^{N_{bit}} - 1$ . In this case, an overflow has to be generated, and careful attention must be paid to discard the next incoming one. With that said, another reason for choosing the maximum timestamp value is that the maximum overflow difference between the original and the delayed sample is one, which makes the process much simpler to implement.

Moreover, for the implementation of the Virtual Delay feature, a modular architecture has been used. A new module, the Virtual Delay Inserter, has been developed and instantiated in series before a modified version of the Delay Synchronizer called Virtual Delay Synchronizer. In detail:

- The Virtual Delay Inserter handles the summation between the delay and the timestamp. It is also responsible for overflow handling when overflows must be generated or discarded.
- The Virtual Delay Synchronizer handles the synchronization of the delayed timestamps.

Figure 10 shows the modular structure of the Node Inserter with the Virtual Delay functionality.



**Figure 10.** Modular structure of the Node Inserter with the Virtual Delay functionality.

### 5.1.2. Virtual Delay Inserter

This module consists of two pipeline stages.

The first stage is responsible for the timestamp computation. Since the summing of the delay can introduce an overflow, the second stage is needed to handle the overflow generation and the correct sampling and propagation of the timestamp, in order not to lose data. Since when an overflow is generated the next received one must be discarded, the data that are overwritten by the generated one are stored in a register. After this event, the timestamps are propagated through this register until there are no valid data to be sent. In cases where the rate is at maximum, this happens when an overflow is received from the TDC. Since the Virtual Delay can vary over time, another possible issue arises: if an overflow has been generated and the delay value decreases, the new timestamp can have a value that refers to the previous one. In order to solve this issue when an overflow is generated, the virtual delay, if it is lower than the previous one, is updated only after an



overflow from the TDC has been received. Finally, the generated overflow flag is needed by the Virtual Delay Synchronizer.

### 5.1.3. Virtual Delay Synchronizer

Compared to the previously introduced Delay Synchronizer, the Virtual Delay Synchronizer converts the Virtual Delay into a pulse of clocks at the BB clock ( $\Delta VD$ ) to wait before starting the Retain Phase in order to synchronize the delayed timestamp injection into the BB.

### 5.2. Virtual Dead Time

The ability to insert a programmable dead time between measurements on the same channel is a really helpful feature. When a signal from a detector has a rising and/or a falling edge, a TDC timestamp is produced. Although filtering is typically requested, in fact, some input noise can still exist and cause unforeseen timestamps (red in Figure 11). These spurious timestamps could be discarded in post-processing by the elaboration modules; however, when the BB rate is high, such discarding might lead to saturations and result in the loss of samples. The Virtual Dead-Time functionality, which stops the incoming events for a programmable period of time (i.e., Virtual Dead Time, represented as keyword KILL in Figure 11) after one has been received, has been added to prevent this.

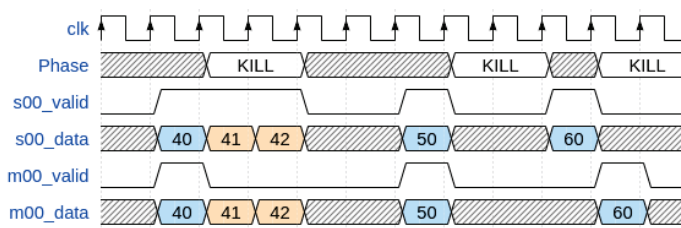


Figure 11. Waveform of the Virtual Dead-Time functionality.

A new IP-Core, named Time Killer (Figure 12), has been developed to enable this feature. The output valid is deasserted if the difference between the receiving timestamp and the input is smaller than the Virtual Dead-Time value. The IP-Core accepts timestamp from TDC (S00\_AXIS input port Figure 12), a Virtual Dead-Time value, and provides a timestamp to the left port of the Node Inserter (M00\_AXIS port in Figure 12).

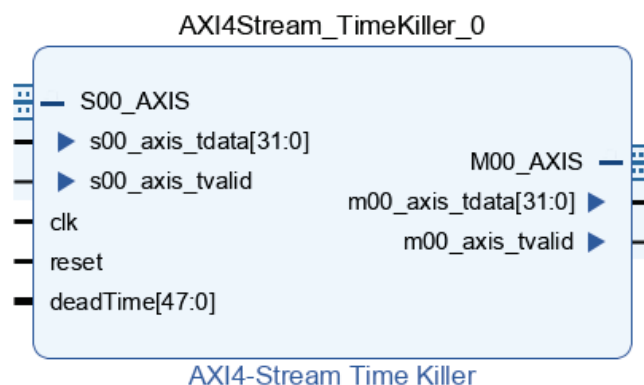
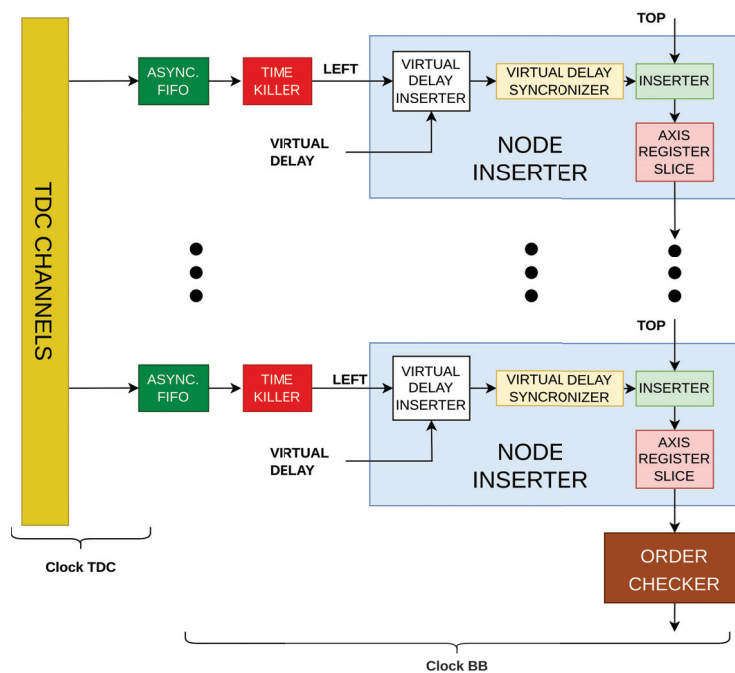


Figure 12. Time Killer IP-Core.

### 5.3. Final Belt-Bus Structure

In conclusion, the new BB structure's design, including the Virtual Delay and Virtual Dead Time functions, is shown in Figure 13.



**Figure 13.** Structure of the improved BB.

#### 5.4. Area Occupancy and Power Dissipation

The area occupation of the Node Inserter and its related submodules (with the modification proposed in this Section) is a function of the number of bits in the TS fields (i.e.,  $N_{bit}$ ). Table 3 presents the area occupancy in terms of CARRY, LUT, FF, and LUTRAM occupied. No resources in terms of DSP and BRAM are utilized. Additionally, the same table provides information on power dissipation, considering a maximum clock frequency of 200 MHz. Comparing Table 3 to Table 1, it is possible to observe a similar occupation and an increase by a factor of two in the CARRY occupied by the Inserter, along with the replacement of the Super Sampler with the AXIS Register Slice. Moreover, a higher usage of LUTs and FFs is observed in the Delay Synchronizer to address the issues outlined in Section 4.1. The higher power dissipation is attributed to a higher clock frequency (200 MHz instead of 130 MHz).

**Table 3.** Area occupancy and power dissipation of Node Inserter and its related submodules presented in Section 5.

$N_{bit}$	Module/Submodules	Power [mW]	CARRY	LUT	FF	LUTRAM
16	Node Inserter	18	4	328	453	58
	Virtual Delay inserter	8		190	261	38
	Delay Synchronizator	8		127	108	20
	Inserter	<1%	4	2	2	
	AXIS Register Slice	2		9	82	
	Time Killer	7		35	159	
24	Node Inserter	18	6	409	632	72
	Virtual Delay inserter	8		259	392	48
	Delay Synchronizator	8		140	174	24
	Inserter	1	6	2	2	
	AXIS Register Slice	1		8	64	
	Time Killer	7		72	223	

Table 3. Cont.

$N_{bit}$	Module/Submodules	Power [mW]	CARRY	LUT	FF	LUTRAM
32	Node Inserter	23	8	429	785	84
	Virtual Delay inserter	10		259	497	56
	Delay Synchronizator	10		159	204	28
	Inserter	1	8	2	2	
	AXIS Register Slice	2		9	82	
	Time Killer	7		94	415	
40	Node Inserter	27	10	549	937	108
	Virtual Delay inserter	12		314	601	72
	Delay Synchronizator	12		224	236	36
	Inserter	<1%	10	2	2	
	AXIS Register Slice	3		9	98	
	Time Killer	10		101	407	
48	Node Inserter	30	12	633	1090	120
	Virtual Delay inserter	13		368	706	80
	Delay Synchronizator	13		254	258	40
	Inserter	<1%	12	2	2	
	AXIS Register Slice	4		9	124	
	Time Killer	16		126	479	
56	Node Inserter	33	14	716	1242	132
	Virtual Delay inserter	14		423	810	88
	Delay Synchronizator	14		282	300	44
	Inserter	1	14	2	2	
	AXIS Register Slice	4		9	130	
	Time Killer	18		140	415	
64	Node Inserter	37	16	802	1395	144
	Virtual Delay inserter	16		477	915	96
	Delay Synchronizator	16		314	329	48
	Inserter	<1%	16	2	2	
	AXIS Register Slice	5		9	149	
	Time Killer	20		161	535	

## 6. Measures and Characterizations

A 3- and 16-channel TDC IP-Cores (with 3 and 16 parallel outputs each), provided by TEDIEL S.r.l. [24], was utilized to test the entire system and undertake the validation of what has been proposed. In Table 4, all the performance metrics of the two TDCs are reported, identical in all respects except for the number of channels. Obviously, the ODR is expressed as the output rate of each individual channel, which will be modified by subsequently inserting the Node Inserter and the structure of the BB.

Tests on the reference TDC architectures are performed on different FPGAs. The host FPGAs are both Xilinx 28 nm 7-Series: an Artix-7 100T for the 3-channel TDC (Figure 14) and a Kintex-7 325T for the 16-channel solution (Figure 15).

Table 4. TDC performance.

Feature	Value
Number of Channels	3 and 16
$N_{bit}$	32
LSB	36.6 fs
FSR	157.3 $\mu$ s

Table 4. Cont.

Feature	Value
$f_{ovfl}$	6.36 kHz
Dead Time	5 ns
Maximum Channel Rate	120 MHz
ODR/Ch	120 Msps
Precision	<12 pr r.m.s.
DNL	<800 fs
INL	<16 ps
LUT/Channel	3869
FF/Channel	5255
LUTRAM/Channel	75
CARRY/Channel	390
BRAM/Channel	2
Power/Channel	284 mW



Figure 14. Picture of the FELIX board (left) hosting the Artix-7 100T for the 3-channel TDC IP-Core and the setup (right).

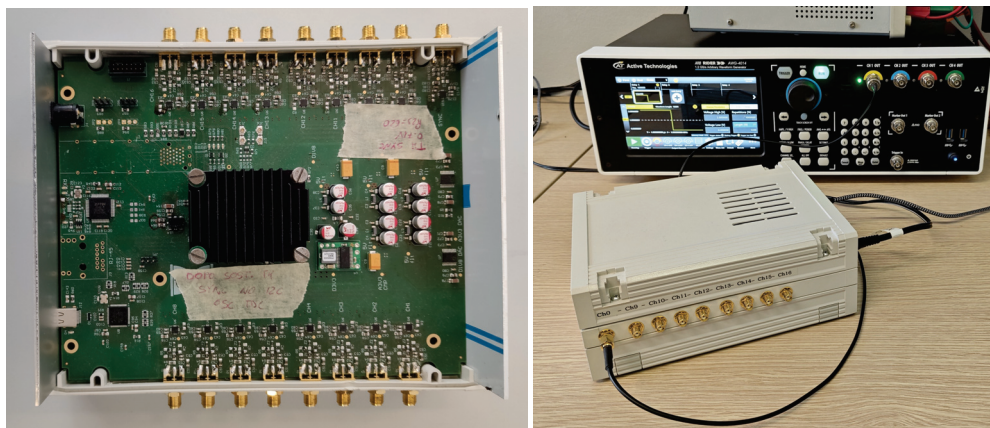


Figure 15. Picture of the Panther (left) board hosting the Kintex-7 325T for the 16-channel TDC IP-Core and the setup (right).

The performed tests involve injecting a pseudo-random signal into each channel of the TDC using the ACTRIVE Arbitrary Function Generator (AWG). Since the AWG has 4 channels in the 16-channel solution, it was decided to divide each channel of the AWG so that it controls 4 channels of the TDC. Pseudo-random signals were generated to ensure a distance between successive events greater than the set dead time. The experiment was conducted by uniformly increasing the rates of all TDC channels, monitoring and subsequently analyzing the output of the BB to verify its correct operation. The status of the various nodes and the Order Checker was also monitored to verify the occurrence of discard phases and the absence of unordered timestamps. To validate the correct operation

of the Time Killer and Virtual Delay modules, the experiment was automated with a script and repeated for numerous values of dead time (1024) and virtual delay (512).

Another test performed, once again with the help of the AWG and a script for its automation, both on the 3-channel and 16-channel versions, involved keeping the rate of all channels except one at zero and then increasing it uniformly, while monitoring and subsequently analyzing the output of the BB and the Order Checker. The experiment was repeated for each channel, always with 1024 values of dead time and 512 of virtual delay.

Outcomes of the experiments are reported in Table 5 for the 3-channel TDC and in Table 6 for the 16-channel solution.

For each of the tests described before, the absence of unsorted timestamps at the BB output was firstly checked. Even though the value of the Order Checker counter is always 0, this module has been kept in for safety reasons; mainly because, although it is simple to model during simulation, the clock uncertainty caused by the CDC is high in rare cases in real situations. Another milestone concerns the ODR, which can achieve up to 149.9 Msps and 199.9 Msps for Artix-7 and Kintex-7, respectively. Additionally, the presence or absence of at list one Discard Phase were monitored; it was observed when the sum of the total rates across the 16 channels reached 97% of the ODR in both solutions.

**Table 5.** BB performance in presence of different versions of BB proposed in this work for the 3-channel TDC.

Feature	Section 3	Section 4	Section 5
Number of Channels	3	3	3
$N_{bit}$	32	32	32
LSB	36.6 fs	36.6 fs	36.6 fs
FSR	157.3 $\mu$ s	157.3 $\mu$ s	157.3 $\mu$ s
$f_{CLK,BB}$	130 MHz	150 MHz	150 MHz
$f_{ovfl}$	6.36 kHz	6.36 kHz	6.36 kHz
ODR	129.9 Msps	149.9 Msps	149.9 Msps
Rate w/o Discard	126.0 Msps	146.0 Msps	145.0 Msps
Dead Time	5 ns	5 ns	5 ns $\div$ 1 ms
Virtual Delay	N.A.	N.A.	0 $\div$ 78.6 $\mu$ s
Unsorted Timestamp	1%	N.A.	N.A.
BB Total Occupancy LUT	0.96%	0.96%	2.43%
BB Total Occupancy FF	0.49%	0.69%	1.79%
BB Total Occupancy LUTRAM	0.91%	0.91%	2.74%
BB Total CARRY	0.30%	0.15%	0.15%
TDC Total Occupancy LUT	18.5%	18.5%	18.5%
TDC Total Occupancy FF	12.6%	12.6%	12.6%
TDC Total Occupancy LUTRAM	0.55%	0.55%	0.55%
TDC Total CARRY	7.43%	7.43%	7.43%
TDC Total BRAM	4.39%	4.39%	4.39%
BB Total Power	144 mW	208 mW	480 mW
TDC Total Power	4544 mW	4544 mW	4544 mW

In both solutions, the ODR, being  $f_{ovfl}$  negligible as per (6), is very close to  $f_{CLK,BB}$ , which is, respectively, 150 MHz for the Artix-7 solution and 200 MHz for the Kintex-7, representing 24% and 25% of the maximum clock frequency that the two devices can handle (625 MHz for the Artix-7 and 800 MHz for the Kintex-7). This is an excellent result considering that, typically, the maximum clock frequency of a system in an FPGA is between 10% and 15% of the maximum frequency. Moreover, from the perspective of area utilization and power consumption, we observe that the presence of the BB is negligible (at least a factor 10) compared to that of the TDC.

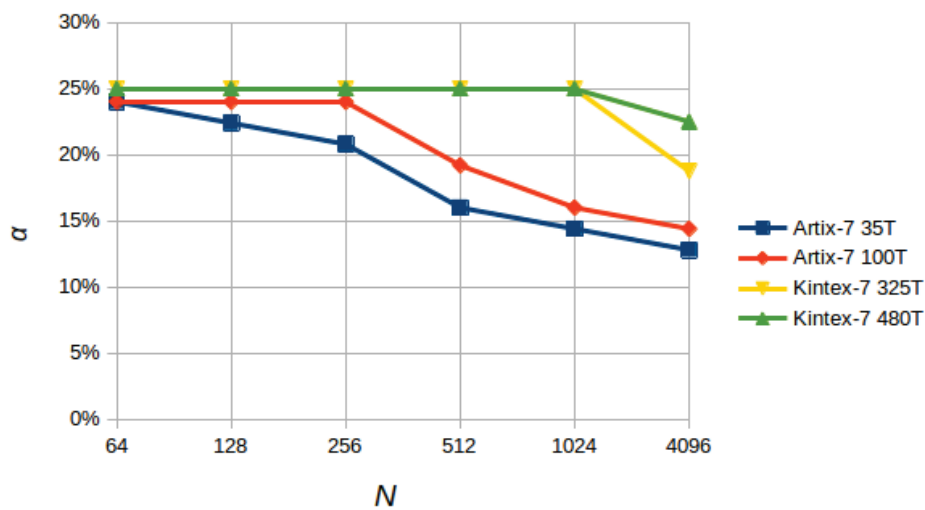
Furthermore, the dependence of the maximum clock frequency of the BB (expressed as a ratio to the maximum frequency allowed by the FPGA,  $f_{FPGA}^{MAX}$ ) on the number of nodes/channels ( $N_{CH}$ ) and the number of bits ( $N_{bit}$ ) on different devices of the Artix-7 family (i.e., 35T and 100T with 32,280 and 101,440 logic cells, respectively, and a maxi-

imum clock frequency of 625 MHz) and Kintex-7 (i.e., 325T and 480T with 326,080 and 477,760 logic cells, respectively, and a maximum clock frequency of 625 MHz) was analyzed by compiling different versions of the BB based on  $N_{bit}$  and  $N_{CH}$ .

**Table 6.** BB performance in presence of different versions of BB proposed in this work for the 16-channel TDC.

Feature	Section 3	Section 4	Section 5
Number of Channels	16	16	16
$N_{bit}$	32	32	32
LSB	36.6 fs	36.6 fs	36.6 fs
FSR	157.3 $\mu$ s	157.3 $\mu$ s	157.3 $\mu$ s
$f_{CLK,BB}$	130 MHz	200 MHz	200 MHz
$f_{ovfl}$	6.36 kHz	6.36 kHz	6.36 kHz
ODR	129.9 Msps	199.9 Msps	199.9 Msps
Rate w/o Discard	126.0 Msps	193.0 Msps	192.0 Msps
Dead Time	5 ns	5 ns	5 ns $\div$ 1 ms
Virtual Delay	N.A.	N.A.	0 $\div$ 78.6 $\mu$ s
Unsorted Timestamp	1%	N.A.	N.A.
BB Total Occupancy LUT	1.58%	1.58%	3.98%
BB Total Occupancy FF	0.80%	1.08%	2.94%
BB Total Occupancy LUTRAM	1.50%	1.50%	4.50%
BB Total CARRY	0.50%	0.25%	0.25%
TDC Total Occupancy LUT	30.4%	30.4%	30.4%
TDC Total Occupancy FF	20.6%	20.6%	20.6%
TDC Total Occupancy LUTRAM	0.9%	0.9%	0.9%
TDC Total CARRY	12.2%	12.2%	12.2%
TDC Total BRAM	7.2%	7.2%	7.2%
BB Total Power	144 mW	208 mW	480 mW
TDC Total Power	4544 mW	4544 mW	4544 mW

The results, shown in Figure 16, highlight a dependence of the ratio  $\alpha$  defined as  $f_{CLK,BB}/f_{FPGA}^{MAX}$  (where  $f_{FPGA}^{MAX}$  is 625 MHz for Artix-7 and 800 MHz for Kintex-7) on the product  $N$  defined as  $N_{CH} \times N_{bit}$ ; we can observe a drop in  $\alpha$  when the ratio between Logic Cells (LCs) and  $N$  is below a value roughly between 300 and 500. This trend is due to routing difficulties caused by the reduction in available resources, indicated by the number of Logic Cells (LC) provided by the device, and the linearity with which the internal modules of the BB scale in terms of area occupation.



**Figure 16.** Picture of the Panther board hosting the Kintex-7 325T for the 16-channel TDC IP-Core.



## 7. Conclusions

This work focuses on a new timestamp management system called BB. The objective is to implement a parallel-to-stream conversion to alleviate the routing of timestamps from high-performance TDCs (i.e., high resolution and FSR, resulting in a high number of bits at a high rate) to the processing module in multichannel applications. The key characteristic of this method is the serialization of several TDC channels in a modular approach, producing timestamps in chronological order while flagging overflow.

In this paper, two issues have been addressed, and two new functionalities have been introduced. The first issue pertains to the occurrence of unsorted timestamps due to the CDC between the clock of the TDC and the BB. Subsequently, by enhancing the FPGA's critical path operations, a second issue related to the Belt-Bus's restricted output rate was resolved.

Additionally, two new features have been incorporated. The first is a Virtual Delay, utilized to compensate for offsets resulting from varying wire lengths between TDC channels and mismatched FPGA routing circuits. The second is Virtual Dead Time, employed to eliminate unforeseen events caused by residual noise at the TDC input.

The BB has been tested on a Xilinx 28 nm 7-Series Kintex-7 325T FPGA, yielding an overall data rate of 199.9 Msps with very limited resource usage (i.e., less than a total of 4.5%) and a power consumption of only 480 mW, considering a 16-channel implementation.

**Author Contributions:** Methodology, F.G.; Software, G.B.; Validation, E.R. and A.C.; Writing—original draft, N.L.; Writing—review & editing, A.G. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The data presented in this study are available in this article.

**Acknowledgments:** A special thanks goes to TEDIEL S.r.l., a spin-off of Politecnico di Milano, for providing the TDC IP-Core.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Lusardi, N.; Geraci, A. 8-Channels high-resolution TDC in FPGA. In Proceedings of the 2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), San Diego, CA, USA, 31 October–7 November 2015; pp. 1–2. [CrossRef]
2. Tancock, S.; Arabul, E.; Dahnoun, N. A Review of New Time-to-Digital Conversion Techniques. *IEEE Trans. Instrum. Meas.* **2019**, *68*, 3406–3417. [CrossRef]
3. Portaluppi, D.; Pasquinelli, K.; Cusini, I.; Zappa, F. Multi-Channel FPGA Time-to-Digital Converter With 10 ps Bin and 40 ps FWHM. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 2002109. [CrossRef]
4. Wang, Y.; Xie, W.; Chen, H.; Li, D.D.U. Multichannel Time-to-Digital Converters with Automatic Calibration in Xilinx Zynq-7000 FPGA Devices. *IEEE Trans. Ind. Electron.* **2022**, *69*, 9634–9643. [CrossRef]
5. Lewellen, T.K. Time-of-flight PET. *Semin. Nucl. Med.* **1998**, *28*, 268–275. [CrossRef] [PubMed]
6. Li, Y.; Ibanez-Guzman, J. Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. *IEEE Signal Process. Mag.* **2020**, *37*, 50–61. [CrossRef]
7. Costa, A.; Lusardi, N.; Garzetti, F.; Ronconi, E.; Maffessanti, S.; Danilevski, C.; Lomidze, D.; Turcato, M.; Porro, M.; Geraci, A. A Study of the Latest Updates of the DAQ Firmware for the DSSC Camera at the European XFEL. *IEEE Access* **2023**, *11*, 84323–84335. [CrossRef]
8. Garzetti, F.; Salgaro, S.; Venialgo, E.; Lusardi, N.; Corna, N.; Geraci, A.; Charbon, E. Plug-and-play TOF-PET Module Readout Based on TDC-on-FPGA and Gigabit Optical Fiber Network. In Proceedings of the 2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Manchester, UK, 26 October–2 November 2019; pp. 1–4. [CrossRef]
9. Nogrette, F.; Heurteau, D.; Chang, R.; Bouton, Q.; Westbrook, C.; Sellem, R.; Clément, D. Characterization of a detector chain using a FPGA-based Time-to-Digital Converter to reconstruct the three-dimensional coordinates of single particles at high flux. *Rev. Sci. Instrum.* **2015**, *86*, 113105. [CrossRef] [PubMed]
10. Chiu, C.L.; Fei, L.Y.; Liu, J.K.; Wu, M.C. National airborne LiDAR mapping and examples for applications in deep-seated landslides in Taiwan. In Proceedings of the 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Milan, Italy, 26–31 July 2015; pp. 4688–4691. [CrossRef]
11. Stoppa, D.; Gonzo, L.; Simoni, A. Scannerless 3D imaging sensors. In Proceedings of the IEEE International Workshop on Imaging Systems and Techniques, Niagara Falls, ON, Canada, 13 May 2005; pp. 58–61. [CrossRef]

12. Lusardi, N.; Garzetti, F.; Costa, A.; Ronconi, E.; Geraci, A. From Multiphase to Novel Single-Phase Multichannel Shift-Clock Fast Counter Time-to-Digital Converter. *IEEE Trans. Ind. Electron.* **2023**, 1–9. [CrossRef]
13. Machado, R.; Cabral, J.; Alves, F.S. Recent Developments and Challenges in FPGA-Based Time-to-Digital Converters. *IEEE Trans. Instrum. Meas.* **2019**, 68, 4205–4221. [CrossRef]
14. Wang, Y.; Xie, W.; Chen, H.; Li, D.D.U. Low-Hardware Consumption, Resolution-Configurable Gray Code Oscillator Time-to-Digital Converters Implemented in 16 nm, 20 nm, and 28 nm FPGAs. *IEEE Trans. Ind. Electron.* **2023**, 70, 4256–4266. [CrossRef]
15. AMBA 4 AXI4-Stream Protocol Specification. Available online: <https://developer.arm.com/documentation/ih0051/a/Introduction/About-the-AXI4-Stream-protocol> (accessed on 10 March 2024).
16. Sesta, V.; Incoronato, A.; Madonini, F.; Villa, F. Time-to-digital converters and histogram builders in SPAD arrays for pulsed-LiDAR. *Measurement* **2023**, 212, 112705. [CrossRef]
17. Timepix. Available online: <https://medipix.web.cern.ch/taxonomy/term/221> (accessed on 10 March 2024).
18. Timepix3. Available online: <https://medipix.web.cern.ch/taxonomy/term/236> (accessed on 10 March 2024).
19. PETsys TOF PET ASIC. Available online: <https://www.petsyseletronics.com/web/product1> (accessed on 10 March 2024).
20. PETsys TOFPET2 ASIC. Available online: <https://www.petsyseletronics.com/web/public/products/1> (accessed on 10 March 2024).
21. Zhang, M.; Wang, H.; Liu, Y. A 7.4 ps FPGA-Based TDC with a 1024-Unit Measurement Matrix. *Sensors* **2017**, 17, 865. [CrossRef] [PubMed]
22. PicoTDC. Available online: [https://kt.cern/sites/default/files/technology/picotdc/tech-brief/picotdc\\_0.pdf](https://kt.cern/sites/default/files/technology/picotdc/tech-brief/picotdc_0.pdf) (accessed on 10 March 2024).
23. Wellheuser, C. *Metastability Performance of Clocked FIFOs*; Texas Instruments Inc.: Dallas, TX, USA, 1996.
24. TEDIEL. Available online: <https://tediel.com/> (accessed on 10 March 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

## Article

# Agile FPGA Computing at the 5G Edge: Joint Management of Accelerated and Software Functions for Open Radio Access Technologies

Nikolaos Bartzoudis <sup>1,\*</sup>, José Rubio Fernández <sup>1</sup>, David López-Bueno <sup>1</sup>, Antonio Román Villarroel <sup>1</sup> and Angelos Antonopoulos <sup>2</sup>

<sup>1</sup> Centre Tecnològic de Telecomunicacions de Catalunya-CERCA), 08860 Castelldefels, Barcelona, Spain; dlopez@cttc.es (D.L.-B.)

<sup>2</sup> Nearby Computing S.L., Carrer de Tuset 32, 08006 Castelldefels, Barcelona, Spain; aantoniopoulos@nearbycomputing.com

\* Correspondence: nbartzoudis@cttc.es; Tel.: +34-936452900

**Abstract:** This paper presents ReproRun, a flexible and extensible run-time framework for the reconfiguration of functions in field programmable gate array (FPGA) devices used in popular software-defined radio (SDR) platforms. The FPGA devices embed a hardwired or soft processing system (PS) which communicates with the programmable logic (PL) using a standard embedded bus interface. In order to apply a seamless run-time partial reconfiguration, we made use of all the related building blocks, design guidelines, and tools offered by AMD-Xilinx. In ReproRun, each partial bitstream targeting a reconfigurable region (RR) of the PL area comes with its respective firmware (i.e., software functions) that runs on the PS side. Our work guarantees run-time updates of the firmware without interrupting the functionality of other software processes running in the PS or PL, by employing a specialized controller, denoted as Run-timeE firmWare reconfIGuration contRoller (REWIRE). The latter leverages the open asymmetric multiprocessing framework (OpenAMP). The partial bitstreams and respective firmware are fetched from a remote location using the trivial file transfer protocol (TFTP). ReproRun can be applied in different FPGA accelerators residing in disaggregated open radio access network (RAN) equipment, adaptive radio access technologies, and Edge servers hosting virtualized functions.

**Keywords:** FPGA; 5G; B5G; SDR; RAN; reconfigurable computing; adaptive computing; function accelerators

## 1. Introduction

The field programmable gate array (FPGA) devices are widely used as function accelerators in numerous end applications. Their inherent computing parallelism and re-programmability provide a tradeoff between the processing flexibility of general-purpose processors (GPP) and the optimized performance of application-specific integrated circuits (ASIC). On top of that, their low power consumption when compared to graphical processing units (GPU) makes them appealing for numerous scenarios where the optimal computation capacity per watt is a hard system constraint. The increasing need for flexible accelerator devices has brought a new family of system-on-chip (SoC) devices that may combine in the same silicon fabric FPGA programmable logic (PL), a processing system (PS), a real-time processing unit, a small-size GPU, ASIC cores, vector-processing engines dedicated to artificial intelligence (AI) and machine learning (ML) workloads, hardwired communication cores and different built-in high-speed input/output (I/O) interfaces [1].

The native system prototyping flow for such complex FPGA-based SoC devices requires a rigorous hardware–software co-design approach. Moreover, coherent runtime programmability is required to fully exploit the capacity of the underlying processing

elements in adaptive computing use cases with dynamic workloads. To achieve this, the set of functions for each application needs to be profiled, partitioned, and distributed across the different processing elements of the FPGA-based SoC, guaranteeing their seamless on-chip interlinking and the efficient management of shared resources (e.g., preventing performance loss and deadlocks). Moreover, in systems with field upgradability requirements, uninterrupted system operation needs to be guaranteed during and after a runtime reconfiguration of intercorrelated functions targeting the different processing elements of an FPGA-based SoC. To tackle the mentioned challenges, the research community and FPGA vendors have made considerable efforts over the past years to produce different design tools, reconfigurable computing solutions, and multi-core processor management frameworks. Towards this end, mature techniques are available to enable the dynamic reconfiguration of a function hosted in a given processing element of the FPGA-based SoCs. However, this is not the case for interlinked functions which are partitioned and placed in different types of on-chip processing elements.

This work focuses on how FPGA-based SoCs can be adequately utilized in mobile network infrastructure equipment, and in concrete in open and programmable radio access networks (RAN). Programmability at the RAN level first received attention in the fourth generation (4G) of cellular networks (e.g., in the context of self-organized networks and cognitive radio use cases [2]). Things have moved a long way since then and in the fifth generation (5G) of broadband cellular networks, agile programmable technologies have been introduced across the entire network stratum, contemplating among others software-defined networking, network slicing, network function virtualization (NFV) [3], functional splits [4], and adaptive software defined radio (SDR) [5]. Given the stringent performance requirements of 5G applications in terms of latency, computing complexity, and energy consumption, the run-time reconfigurable function accelerators like the FPGA-based SoCs, are considered key processing elements for the previously mentioned 5G technology innovations. Ultra high-performance FPGA-based SoCs hosting agile function accelerators are also critical for AI-ML operations in beyond 5G distributed Edge computing architectures [6].

In this paper, we present ReproRun framework (Reprogramming accelerated and software functions at Runtime in FPGA-based SoC devices) that was designed, implemented, and validated using popular SDR systems. ReproRun is able to apply a seamless joint reconfiguration of functions targeting both the PL and the PS parts of Xilinx FPGA-based SoCs.

The run-time reconfiguration of accelerated FPGA-based functions is a key feature that aims to truly exploit the processing flexibility of FPGA devices in the context of reconfigurable computing [7]. The partial reconfiguration features have been following the architectural evolution of FPGA devices [8]. However, despite the undeniable benefits of PR, its adoption on behalf of different technology stakeholders and vertical application sectors has been rather modest. The run-time spatial-temporal exploitation of FPGA compute resources requires certain applications to fulfill some strict latency requirements in order not to suffer service downtimes. Different works tried to improve the PR latency and reconfiguration control overheads [9–11]. Another major setback for more widespread use of the PR methods is concerns related to data privacy and security when FPGA devices are used as shared computing resources in multi-tenant edge or cloud applications. Data or application security and privacy concerns have been historically addressed either through key upgrades of the native PR frameworks offered by the main FPGA vendors by including new security features, or by contributions coming from the academic community [12–14]. However, cloud and edge computing are the domains driving the FPGA reconfigurable computing innovations, by introducing virtualized frameworks that orchestrate FPGA accelerators in 5G Cloud Environments [15]. These efforts are valuable to increase the availability of FPGAs to virtual machines or containers and to enhance the flexibility of cloud FPGA deployments [16,17]. The value of FPGA computing with PR is also manifested by its adoption in different vertical sector use cases [18–21]. One of the latest research interests is to use FPGA computing and PR in those use cases where ML models

need to be deployed and reconfigured on the fly [22,23]. Finally, run-time PR in FPGA devices is used in programmable and virtualized networking deployments [23–26] and SDR [27,28] infrastructures.

All the previously cited works focusing on FPGA-based reconfigurable computing use cases, share conceptual communalities with the framework presented in this paper. However, the key differentiating factor is that the reconfigurable framework presented in this paper jointly manages the run-time reconfiguration of functions featuring an inter-linked software and accelerated portion in a seamless way; the former is implemented in a processor embedded in the FPGA device, whereas the latter is implemented in a given partially reconfigurable PL area. Hence, the idea of reconfiguring on the fly either the software or PL-accelerated portion of the function without affecting the availability of the other part is a notable contribution compared to the state of the art. The novelty of ReproRun is important when considering that every time more processing elements are packed in the same FPGA fabric to yield complex SoC devices with heterogeneous processing resources. As a consequence, the mindset of applying a hardware-software co-design flow in an inherently monolithic or static way is challenged when the compute capacity and elasticity of this type of device are leveraged in dynamic and adaptive computing environments. ReproRun advocates that software–hardware co-design has to be rethought to contemplate run-time procedures; its contribution can be further extended and exploited in different types of use cases not necessarily limited to SDR and 5G networking.

The remainder of the paper is organized as follows: Section 2 states the motivation and contribution of the work; Section 3 provides an overview of some assumptions and also the specifications that were taken into account during the development of the system; Section 4 presents the design of ReproRun with its main components; Section 5 provides the FPGA implementation details for the two SDR platforms; Section 6 presents the results of the experimental validation that was conducted using the two frameworks supporting the two SDR platforms (in the first based on Analog Devices FPGA firmware and graphical tools and in the second based on the GNU radio companion and the RFNoC framework of Ettus Research); Section 7 provides the conclusions; and finally, the Appendix A includes detailed information of a limitation encountered in one of the two experimental platforms.

## 2. Motivation and Contributions

Parts of the PL area in FPGA devices could be defined as reconfigurable regions (RR) at design time. One RR can host suitably pre-compiled functions, which could be replaced or interchanged at run-time, without interrupting the operation of other functions running in the static part of the FPGA (i.e., the rest of the PL area that is not defined as an RR) or in other RRs. This procedure is called dynamic FPGA partial reconfiguration (PR), which since the 2019.2 version of the Vivado FPGA design toolchain has been enriched and rebranded as dynamic function exchange (DFX). The PL functions targeting the RR are denoted as partial bitstreams. However, the run-time update, upgrade, or replacement of interdependent functions that are split and placed in the PL and PS parts of the FPGA (or other on-chip processing elements) is hiding numerous challenges in terms of performance, availability, and security. This is exactly the key contribution of ReproRun that allows to upgrade or replace interdependent PL or PS functions at run-time without compromising the availability of other services and applications running on the chip. This is a novel top-up feature to the existing PR framework offered by AMD-Xilinx.

In our case, each partial bitstream targeting an RR of the PL area comes with its respective firmware (i.e., software functions) that runs on the PS side. In order to apply a run-time partial reconfiguration, we made use of all the related building blocks, design guidelines, and tools offered by Xilinx, and also extended, modified, and combined available reference designs.



On the PS side, we leveraged asymmetric multiprocessing (AMP) principles to manage the inter-processor communication, signaling, and resource sharing. Our work guarantees the seamless run-time update of applications running at one core of the PS without interrupting the functionality of other software processes running in the same core (i.e., the rest of the PS firmware) or the interdependent functions in its PL counterpart. This was made feasible by designing a novel computing kernel denoted as a *run-time firmware reconfiguration controller* (REWIRE). Hence, applications comprising a PL and a PS part can update/replace any of these components without service interruption.

Adding a run-time reconfiguration framework in FPGA-powered SDR platforms serving both the PL accelerated functions and the PS firmware, opens up many research opportunities in the field of reprogrammable 5G communication systems. For instance, ReproRun can be used as the basis of an abstraction layer that could extend NFV towards those RAN infrastructure equipment residing at the edge of 5G networks, where FPGA acceleration is paramount for serving the needs of highly demanding Edge applications. Similarly, SDR systems built to host adaptive multi-radio access technologies could use ReproRun to add, replace, or scale PS/PL functions. Adaptive RAN FS can also be supported by the ReproRun framework, adding dynamicity in the run-time placement of functions, especially when considering run-time migration scenarios from split option 7.2 to split option 7.3 [4].

ReproRun was applied in two SDR platforms featuring different 7 series AMD-Xilinx FPGA devices, and a number of different specifications in terms of on-board components and connectivity options. This alone is an important contribution of ReproRun because it implied a different design architecture for updating or replacing the PS firmware in the two SDR platforms and for applying the PR in PL-accelerated functions. Moreover, it has proved the flexibility and versatility of ReproRun to simultaneously foster accelerated computing and agile multicore reconfiguration, tailored to FPGA device specifications, SDR hardware limitations, and specific design requirements. In this respect, ReproRun could be seen as an enabler for those Open RAN and Edge computing use cases that require field adaptive function acceleration, efficient resource orchestration, and agile split computing support.

### 3. Base Assumptions and Specifications

ReproRun provides a flexible and extensible run-time framework for the partial reconfiguration of FPGA devices used in popular SDR platforms. The FPGA devices embed a hardwired or soft PS which communicates with the PL using a standard embedded bus interface. Our work is based on the PR design flow of AMD-Xilinx, which makes use of the partial reconfiguration controller (PRC) intellectual property (IP) core and the internal configuration access port (ICAP) port.

It is important to highlight that we have not used the newest AMD-Xilinx dynamic function exchange (DFX) framework, the successor of the Xilinx Partial Reconfiguration flow, in order to maintain compatibility with specific firmware versions of the SDR boards of interest. Notwithstanding, the PR flow is still valid across numerous platforms featuring different FPGA devices. AMD-Xilinx provides three core documents [29–31], on how to use the PR on its FPGA devices. On top of this, AMD-Xilinx also disposes of some key Application Notes, which help to understand the proposed reconfiguration framework. The radio frequency network on-chip (RFNoC) programming framework natively used in numerous Ettus Research SDR platforms was also used in this work; the “Getting Started with RFNoC Development” ([https://kb.ettus.com/Getting\\_Started\\_with\\_RFNoC\\_Development](https://kb.ettus.com/Getting_Started_with_RFNoC_Development), accessed on 1 February 2024) Application Note underpins the dependency with the AMD-Xilinx Vivado 2017.4 toolchain version. The concrete version of RFNoC that was used in this work is the UHD\_4.0.0.rfnoc-devel-161-. The designed run-time PR framework has been tested and validated in the following SDR platforms:

SDR1: Combines the Xilinx ZC706 development kit featuring the Xilinx Zynq XC7Z045 SoC device with the AD-FMCOMMS-2/3 evaluation board from Analog Devices. The



latter features the AD9361  $2 \times 2$  RF transceiver IC (TX band: 47 MHz to 6.0 GHz, RX band: 70 MHz to 6.0 GHz, tunable bandwidth: 200 kHz to 56 MHz).

SDR2: The Ettus Research USRP X310 SDR board, features a Xilinx Kintex 7 FPGA device. The X310 includes an agile RF transceiver module able to be programmed for a large range of RF frequencies (from 1.2 GHz to 6 GHz, channel bandwidth of 40 MHz). The fundamental difference of this device when compared to SDR1 is the absence of a hardwired PS.

The run-time reconfiguration in ReproRun comprises the following two parts:

- i. The partial bitstream that is meant to reconfigure a reconfigurable module (RM). The latter is a PL reconfigurable area defined at design time. The partial bitstreams could be digital signal processing (DSP) functions typically encountered in SDR systems (e.g., a finite impulse response (FIR) filter), or any other type of PL-accelerated functions.
- ii. Firmware functions residing at the PS (hardwired or soft-embedded microprocessor) are essentially the piece of software that communicates, controls, and extends the operation of the partial bitstream configured in a PL reconfigurable area (e.g., a software function that programs, registers, and manages different coefficient sets of the FIR filter based on performance indicators). Each firmware function is linked with an equivalent partial bitstream, forming a bonded hardware–software application.

ReproRun provides a seamless run-time reconfiguration of both the PL and PS-based functions. The starting assumption in both SDR platforms was that the FPGA device includes functions running at the static part of the design and also a corresponding firmware running at the PS, whose operation must remain uninterrupted during and after the PR process. The FPGA devices in the two SDR platforms include one RM.

#### 4. System Design

System design aspects included in two AMD-Xilinx Application Notes were reutilized in ReproRun. The first one [32] shows how to use the lightweight internet protocol (lwIP) open-source TCP/IP networking stack to add networking capability to an embedded system. The lwIP was utilized to develop an echo server, a web server, a trivial file transfer protocol (TFTP) server, and receive/transmit throughput tests. The Vivado SDK provides lwIP software customized to run on the AMD MicroBlaze processor. An AMD-Xilinx Kintex-7 FPGA KC705 Evaluation Kit was used to test the system, which had to be adapted and modified for the specific design needs of ReproRun. The second Xilinx Application Note [33] provides a software library written in C that can be used to fetch partial bitstreams over Ethernet with the help of a TFTP server. A partial bitstream discovery mechanism is provided for applications that expect their available partial bitstreams to change over time. Both applications run on a MicroBlaze Processor and help to fetch partial bitstreams that reconfigure the FPGA RM. Similarly, the Kintex-7 FPGA KC705 Evaluation Kit was used to validate the PR features. Useful features of both AMD-Xilinx Application Notes were combined, extended, and modified to serve the needs of ReproRun. For instance, the partial bitstreams and corresponding firmware objects are fetched from a remote location using the TFTP service and stored in a defined range in the external synchronous dynamic random-access memory (SDRAM) of the SDR platforms accessible by both the PL and the PS side.

ReproRun was designed to handle the seamless reconfiguration of interdependent partial bitstreams and firmware object files. This means that a partial bitstream reconfiguration in the PL does not lead to a deadlock in the firmware function associated with that partial bitstream and vice versa. In this respect, ReproRun provides consistency between RMs making sure that when a PR bitstream is swapped for another, the connections between the static design and the RM are identical, both logically and physically. This was made feasible by defining at an early design stage the interfaces between the RM and the static design. Choosing the DSP functions to be implemented inside each RM posed requirements on the type of resources an RR must comprise and, on the FPGA floor planning.

The PRC IP core guarantees a high throughput while fetching the bitstream [34] and, accordingly, a low reconfiguration time. It also provides flexibility when managing RMs without adding any overhead to the PS (i.e., it does not compromise the execution of other tasks). Finally, the PRC IP core provides the necessary functionality to avoid deadlocks during the reconfiguration process (i.e., preserve the static part of the design) [31] by employing isolation logic and error detection features.

The PRC IP fetches bitstreams from a double data rate (DDR) SDRAM memory, which is specially thought to be used by the PL, and thus we call it SDRAM-PL. One of the ideas behind REWIRE and especially its part running under Linux, is that it should be able to fetch a partial bitstream and place it in a predefined memory region of the SDRAM-PL memory. Thus, SDRAM-PL must be accessible from PS as well. Then, on receiving a software trigger command the PRC core can access that bitstream and use it for partial reconfiguration. It is important to note that memory transactions between the PRC and PL-DDR are not going through the microprocessor, a fact that guarantees minimal latency during partial reconfiguration. The AXI4 interconnect was added to the design in order to arbitrate transactions from the PRC and microprocessor to the memory interface generator (MIG) controller.

### REWIRE

An AMP approach [35] was adopted for the SDR1 exploiting the dual-core ARM A9 CPU hardwired in the Xilinx Zynq XC7Z045 device. The first core, denoted thereafter as Processor 0, needs to operate in real-time, having direct access to the hardware and deterministic processing latencies. This is mainly due to the fact that wireless transceivers typically handle high-load data processing, control several IP cores inside the PL, and need to interface with high-speed I/O protocols. The reconfigurability requirements can be considered as a control plane issue of the entire embedded design. The OpenAMP (<https://github.com/OpenAMP>, accessed on 1 February 2024) solution was applied in unsupervised mode, using a bare-metal application and other firmware functions for the real-time communication functionalities in Processor 0 (slave), along with a flexible reconfigurable Linux kernel responsible for realizing administration and control functions in the second core of the ARM A9 processor denoted as Processor 1 (master).

The communication channel was built on top of the OpenAMP framework and the *libmetal* library [36]. The latter provides user application programming interfaces (API) that allow to access devices, handle device interrupts, and request memory across different operating environments (*libmetal* is available for Linux, FreeRTOS, and bare-metal environments). On the other hand, OpenAMP provides life cycle management and inter-processor communication capabilities to control remote compute resources, a standalone library usable with bare-metal software environments and compatible interfacing with upstream Linux remoteproc, rpmsg, and VirtIO components. The detailed flow diagrams of OpenAMP are provided in figures 1-1 and C-1 in [35]. REWIRE integrates OpenAMP and disposes of a set of message queues deployed over a shared memory region. The software-generated interrupts, denoted as inter-processor interrupts (IPI), were adopted for the notifications among the two cores of the ARM A9, taking care of new pending messages. This design serves to lower the response latency when compared to polling shared memory, a fact that is particularly important for time-critical routines. Proper synchronization methods were applied to avoid contention on shared resources. Xilinx reference examples [37–39] and example applications of the OpenAMP framework were leveraged towards this end.

In the case of SDR2, the communication channel was built on top of the TCP/IP stack. As Processor 1 is represented in that case by a personal computer (PC) microprocessor, we have used a standard portable operating system interface (POSIX) compliant TCP/IP stack implementation provided by Linux. For Processor 0, we have used a Microblaze core [40] with a bare-metal environment and the Lightweight IP (lwIP) TCP/IP stack [32] for enabling basic networking operations. For the firmware, we used the executable and

linking format (ELF) for the object and executable files since Xilinx provides a distribution of the GNU compiler and linker collection.

In order to provide further insights into the REWIRE functionality, we take as a reference FPGA architecture the one included in SDR1, because the Zynq device offers a more versatile embedded environment due to its embedded dual-core CPU. In a nutshell, the REWIRE controller manages the AMP framework and the reconfiguration procedure of the firmware functions running at the PS by performing the following tasks (Figure 1):

- establishes the communication channel between Processor 1 and Processor 0;
- parses user-input commands;
- reads the specified partial bitstream and object files from the filesystem;
- relocates the latter into dedicated memory regions;
- sends control commands to Processor 0 specifying the reconfiguration flow procedure.

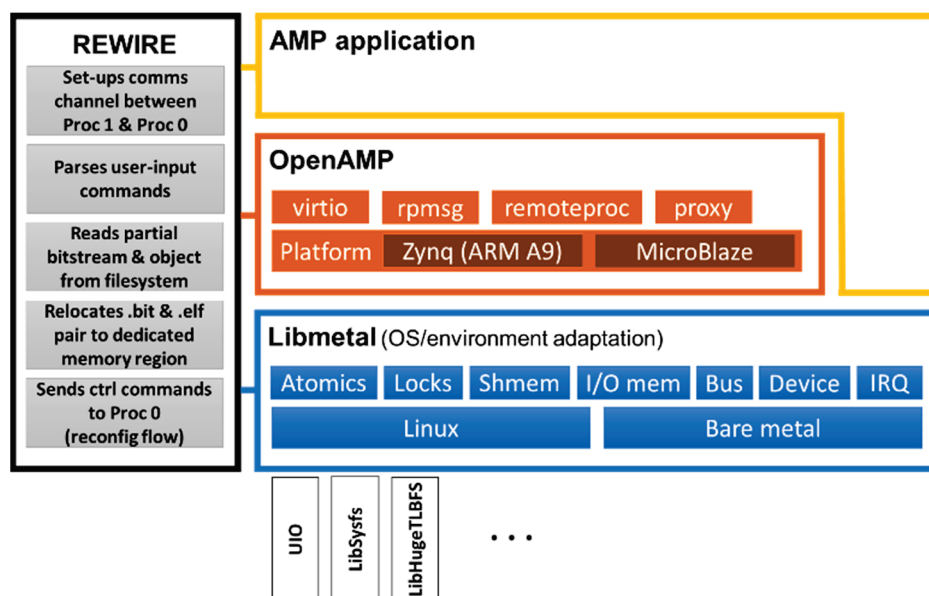


Figure 1. A high-level overview of REWIRE.

We logically separated these functionalities of REWIRE into two parts. The first one is responsible for the communication between the processors, while the second one manages the firmware objects, as detailed in the following:

Part I: The inter-processor communication stack running on Processor 1 utilizes interfaces exposed to Linux user space by the OpenAMP framework. From a user-space perspective, it looks like any Linux character device, and accordingly, it provides the standard system calls-based API to interact with a device of this type. In fact, it represents an RPMsg device, which is managed by means of a series of kernel drivers implementing APIs defined in VirtIO, Remoteproc, and RPMsg frameworks of Linux. In Processor 0, the communication must be supported by the bare-metal counterpart of the mentioned frameworks. Both parts of the REWIRE, running at Processor 0 and Processor 1, interact with each other through an RPMsg communication channel. This is made possible by utilizing an external SDRAM memory for the data exchanges and IPIs for the notifications about new data transactions.

Part II: The second main part of REWIRE processes the firmware objects. As already mentioned, the ELF format is used for the object and executable files. Another part of the firmware responsible for the RR is a function or, in terms of C language a set of functions and associated structures, with a “self-contained” state. This function is mainly responsible for the initial configuration of the IP blocks inside the PL (i.e., executed once at the time when the Linux side of REWIRE notifies the bare-metal side that a new partial firmware is available). Essentially this part of the firmware must represent a single C translation

unit, which after compilation will result in a single object file. In turn, the main static executable stores a pointer to the function managing the associated RR and indirectly calls this function when a new object file is delivered and processed by REWIRE. Accordingly, a predefined address of a memory region is assigned to this function pointer. This approach is applied for all sections of the object file (i.e., code, data, read-only data, and so on); in other words, a predefined amount of memory in the main executable is reserved in order to account for different firmware object deliverables. Laying out memory regions of firmware is achieved by the means of low-level GCC attributes and a linker script.

The static executable ELF file for Processor 0 is created among other files from a function `main.c` and the firmware dummy function `dummy_prm_func.c`, which simply displays a “Hello world” message. The linking of object files is done using a linker script file, which includes the following specifications:

- (1) The size of the sections that make up the firmware function code is defined; before proceeding with reprogramming a newly produced and fetched firmware, the size compliance of the sections must be verified, because the new code will be downloaded into the memory where the bare-metal program of Processor 0 is executed. Any potential failure could block the execution of the code on Processor\_0:

```
_PR_MODULE_FUNC_CODE_SIZE = 0x10,000;
_PR_MODULE_FUNC_DATA_SIZE = 0x1000;
_PR_MODULE_FUNC_RODATA_SIZE = 0x1000;
```

- (2) The memory direction for loading the static ELF executable file is defined as follows:

```
MEMORY
{
    ps7_dds_0_S_AXI_BASEADDR: ORIGIN = 0x3e000000, LENGTH = 0x00400000
    ps7_ram_0_S_AXI_BASEADDR: ORIGIN = 0x00000000, LENGTH = 0x00030000
    ps7_ram_1_S_AXI_BASEADDR: ORIGIN = 0xFFFF0000, LENGTH = 0x0000FE00
}
```

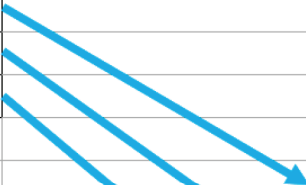
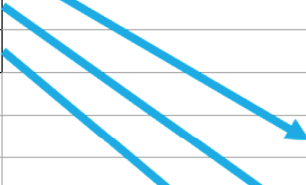
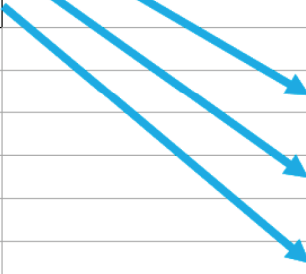
- (3) The sections of the firmware `dummy_prm_func.o` object code are defined and the `pr_func_start`, `pr_data_start`, and `pr_rodata_start` symbols are created:

```
pr_mod_func : {
    . = ALIGN(0x10,000);
    __pr_func_start = .;
    *dummy_prm_func.o(.text)
    *dummy_prm_func.o(.text.*)
    . = __pr_func_start + _PR_MODULE_FUNC_CODE_SIZE;
    __pr_func_end = .;
} > ps7_dds_0_S_AXI_BASEADDR

.pr_mod_data : ALIGN(0x1000) {
    __pr_data_start = .;
    *dummy_prm_func.o(.data)
    *dummy_prm_func.o(.data.*)
    . = __pr_data_start + _PR_MODULE_FUNC_DATA_SIZE;
    __pr_data_end = .;
} > ps7_dds_0_S_AXI_BASEADDR

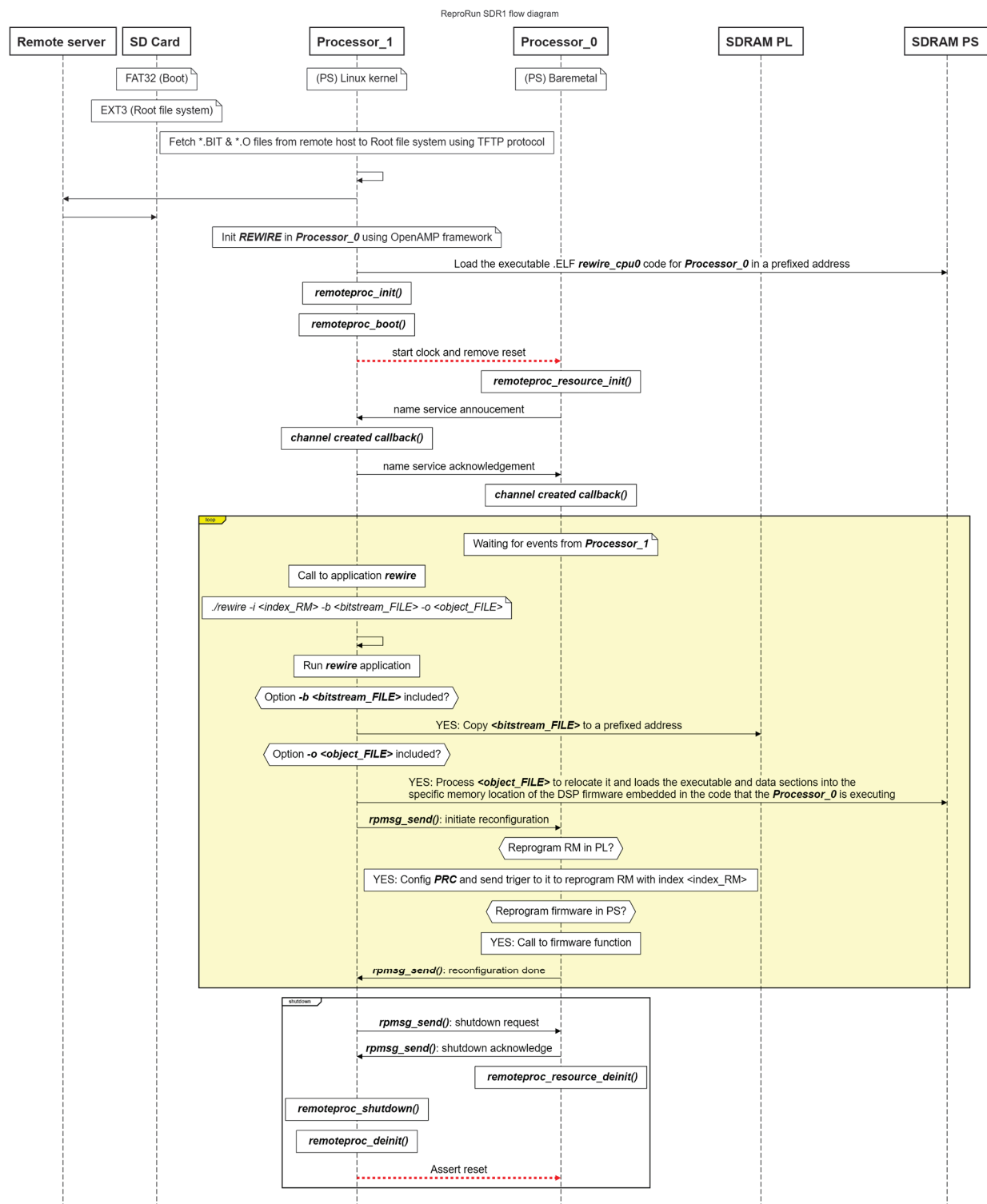
.pr_mod_rodata : ALIGN(0x1000) {
    __pr_rodata_start = .;
    *dummy_prm_func.o(.rodata)
    *dummy_prm_func.o(.rodata.*)
    . = __pr_rodata_start + _PR_MODULE_FUNC_RODATA_SIZE;
    __pr_rodata_end = .;
} > ps7_dds_0_S_AXI_BASEADDR
```

On the host side, the dummy firmware object was compiled and linked with the mentioned main executable in order to reference its defined symbols. REWIRE creates at run-time a look-up table filled with the symbol values of a static executable and uses this table to resolve global symbol references in newly delivered firmware objects. When all necessary symbol resolutions, address fixups and the relocation and copy of code and data into the dedicated memory region are done (Figure 2), REWIRE in Processor 1 (under Linux) notifies its bare-metal counterpart on Processor 0 that it can reprogram the RR with a new bitstream and execute new firmware code.

File.o		PS SDRAM memory
.text		.text (ps7_ddr_0_S_AXI_BASEADDR)
.data		...
.rodata		.data
		...
		__pr_func_start
		...
		__pr_data_start
		...
		__pr_rodata_start
		...
		...

**Figure 2.** Copying the different sections of an object file to the dedicated memory space of the PS SDRAM.

A complete flow diagram of ReproRun's reconfiguration of partial bitstream and firmware objects is shown in Figure 3.



**Figure 3.** The ReproRun flow diagram. OpenAMP's Rpmc in Proc 1 was implemented in Linux userspace.

## 5. Implementation

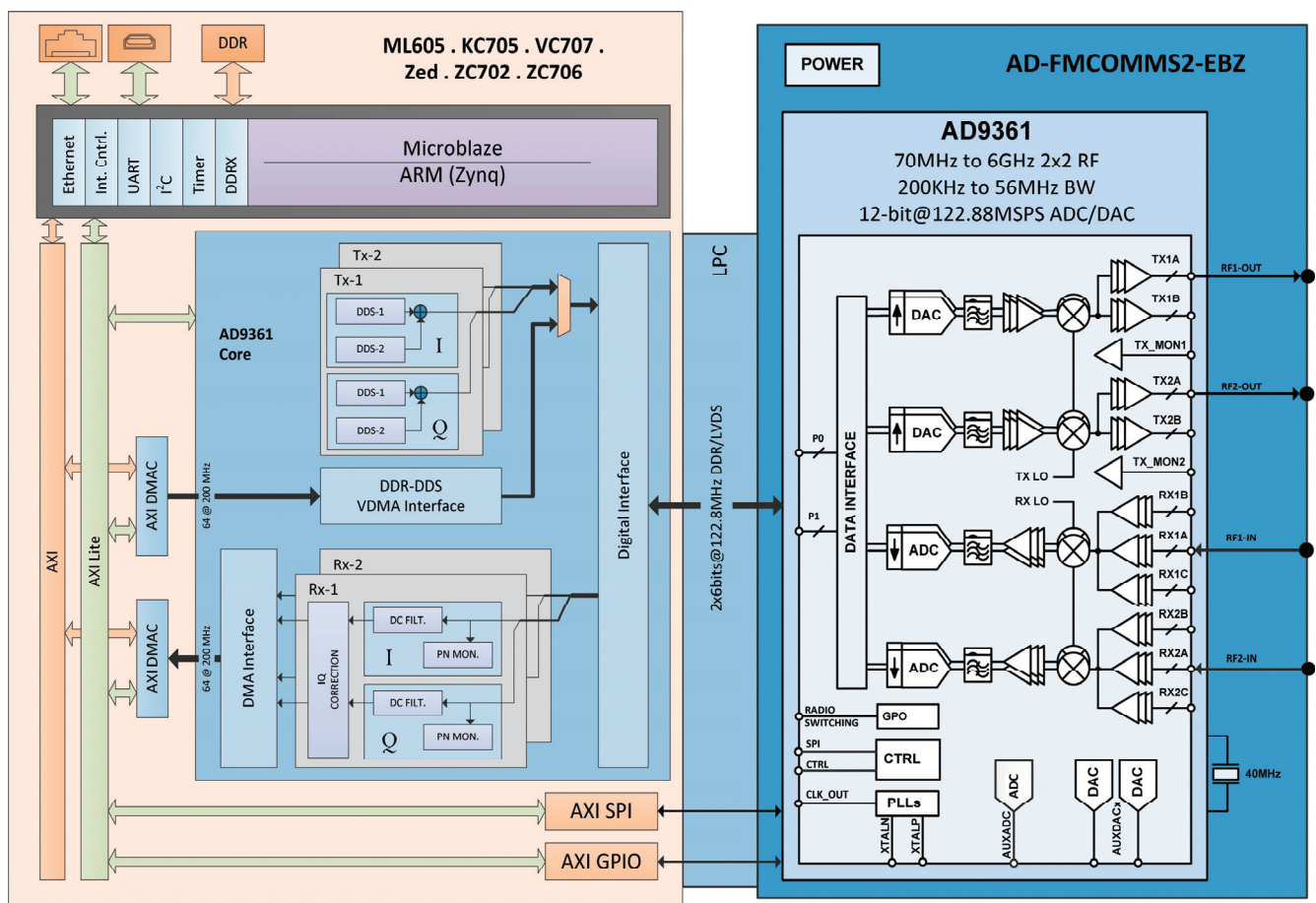
Given that the two SDR boards feature several differences apart from the different FPGA devices, they also concern other I/O and hardware specifications, a slightly different design and implementation approach was adopted for the PR solution in the two SDR boards. In fact, for the case of the SDR2 board, an important hardware limitation was



encountered after applying a multi-layer debugging of the PRC core and ICAP port, taking into account guidelines [29] such as the Pblocks recommended configuration, the global clocking rules for PR, the reset monitoring and CRC checking after reconfiguration, and other PR debugging tips. This hardware limitation deters the SDR2 board from being used in an automated run-time PR mode. For more information regarding this issue, please refer to the Appendix A.

### 5.1. SDR1

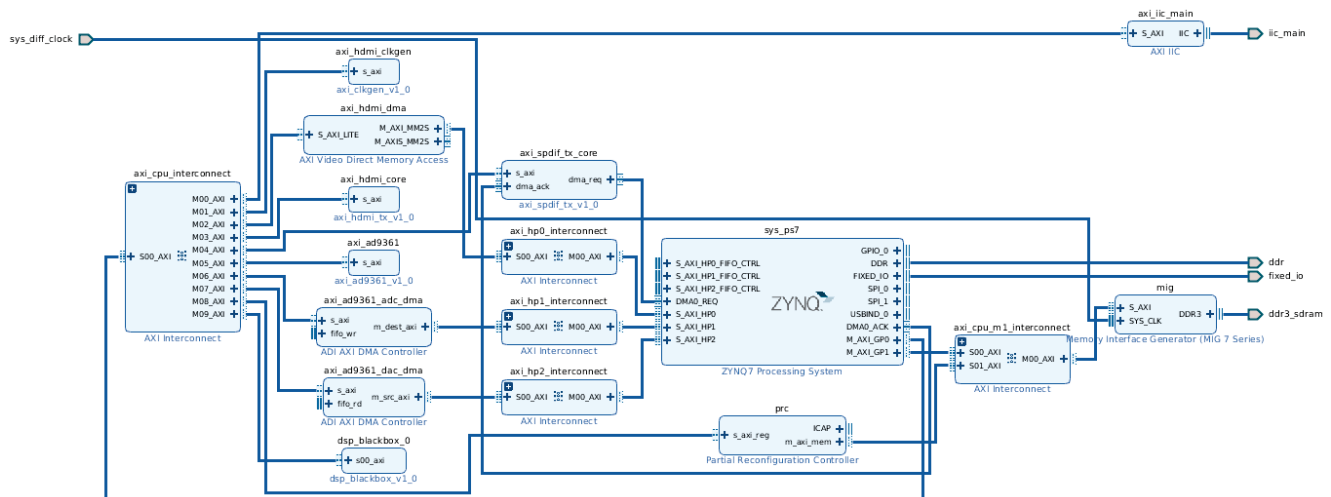
In order to enable the partial reconfiguration on SDR1 we have extended the reference design provided by Analog Devices for the Xilinx ZC706 board and the AD-FMCOMMS-2/3 RF-frontend (https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/reference\_hdl, accessed on 1 February 2024) (Figure 4). In the static part of the reference design, we added the PRC IP core, a wrapper around the ICAP port, a MIG IP core, a block controlling the onboard LEDs, and a DSP block, which represents the reconfigurable part of the design. The block controlling the LEDs aims to show an uninterrupted operation of the static part of the design. We also applied changes to the original blocks of the reference design to enable multiplexing of data on the digital-to-analog converter (DAC) path, to be able to switch between two sources of data (i.e., generated by the PS or the PL). The resumed version of the Vivado block design responsible for the partial reconfiguration of the PL is shown in Figure 5.



**Figure 4.** The AD9361 hardware description language (HDL) reference design in SDR1.

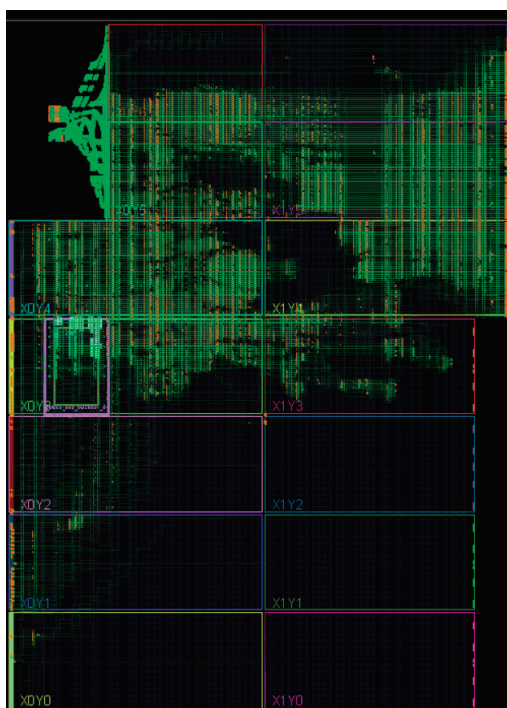
Regarding the RR, two functions have been successfully implemented that can be replaced at run-time through the PRC controller: a direct digital synthesizer (DDS) and a 5G new radio (NR)-like signal that is cyclically play-backed. The first block reuses the Xilinx DDS IP core [41] and allows for generating a tone signal. The second block comprises

a memory array built with memory elements embedded in the FPGA fabric (i.e., block-RAMs-BRAM-) that allows for storing a 5G NR-like signal, along with a state-machine controlling the playback process of the signal. Each processing block incorporates an AXI4 slave interface; this allows the PS to write into memory-mapped registers that define the behavior of each processing block. For example, in the case of the DDS, the register sets a parameter allowing to change the synthesized frequency at run-time. Changing the processing block behavior is done in a part of the firmware responsible for the RR, which could be reconfigured by REWIRE. The reconfiguration flow of the PRC core is controlled by the static part of the REWIRE firmware running at Processor 0.



**Figure 5.** The top-level Vivado Block Design of SDR1.

A view of the implementation results after placement and routing featuring a single RR are shown in Figure 6 and a breakdown of the resources' usage in Table 1. It reveals that the FPGA implementation is low-dense and occupies a relatively small amount of the total resources, leaving the possibility of having more than one RR.



**Figure 6.** The routed FPGA design in SDR1 showing the RR.

**Table 1.** Implementation results of the SDR1 FPGA design where the maximum values of the FPGA resources per category are quoted in parentheses.

Slice LUTs (218,600)	Slice Regs (437,200)	F7/F8 Muxes (109,300/54,650)	Slice (54,650)	LUT as Logic/Memory/FFs (218,600/70,400/218,600)	BRAMs (545)	DSPs (900)	Bonded IOB/IOPADs (362/130)
29,296	37,628	591/24	13,174	25,961/3335/13,390	18	71	223/130

## 5.2. SDR2

The implementation of the system in the case of SDR2 required an elevated effort due to the inherent design complexity of the RFNoC framework. RFNoC's main mission is to facilitate the use of hardware-accelerated DSP functions running in Ettus Research USRP devices, abstracting away their implementation complexity. The RFNoC offers likewise a hardware-in-the-loop system between a host PC and the SDR platform of interest, along with an automated software-hardware development framework tailored for SDR researchers and software engineers familiar with the GNU Radio design flow targeting USRP SDR devices. It is important to highlight that the RFNoC is not meant to be integrated with other third-party code, but solely used for its default operating scope. Extensions of the RFNoC Verilog code indeed take place within the RFNoC project but always have as an objective to abstract the low-level hardware description language (HDL) design details from the users. In the case of the PR framework proposed in this paper, several modifications and extensions of the RFNoC Verilog source code were required, with the goal of keeping its native operation intact.

When building an FPGA image for SDR2, the RFNoC framework allows configuring the two gigabit Ethernet (GigE) interfaces either as 1 GigE or as 10 GigE. The HDL firmware uses the 1 G/2.5 G Ethernet PCS/PMA and the 10 G Ethernet PCS/PMA Xilinx IP cores for the 1 GigE and 10 GigE, respectively. As far as the Ethernet medium access layer (MAC) layer is concerned (both for the 1 GigE and 10 GigE interfaces), RFNoC provides a custom Verilog implementation instead of using the relevant IP cores of Xilinx. In this context, a central element of RFNoC's Verilog firmware is the ZPU embedded soft microprocessor (<https://github.com/pdsmart/ZPU>, accessed on 1 February 2024), which interfaces with peripherals and custom Verilog glue logic via the Wishbone Bus [42] (i.e., an open-source embedded bus standard). The Ethernet port 0 is configured to be 1 GigE serving exclusively the needs of ReproRun, whereas the Ethernet port 1 is configured to be 10 GigE serving exclusively the needs of the RFNoC firmware. While the HDL implementation of the 10 GigE link was maintained as in the original version of RFNoC, the implementation of the 1 GigE link was implemented based on the AXI 1G/2.5G Ethernet Subsystem IP core of Xilinx, which helped us to establish a bidirectional AXI4 connection between the Ethernet MAC-layer and an instance of a MicroBlaze processor (i.e., acting as Processor 0). Some necessary modifications were applied to the RFNoC Verilog code towards this end.

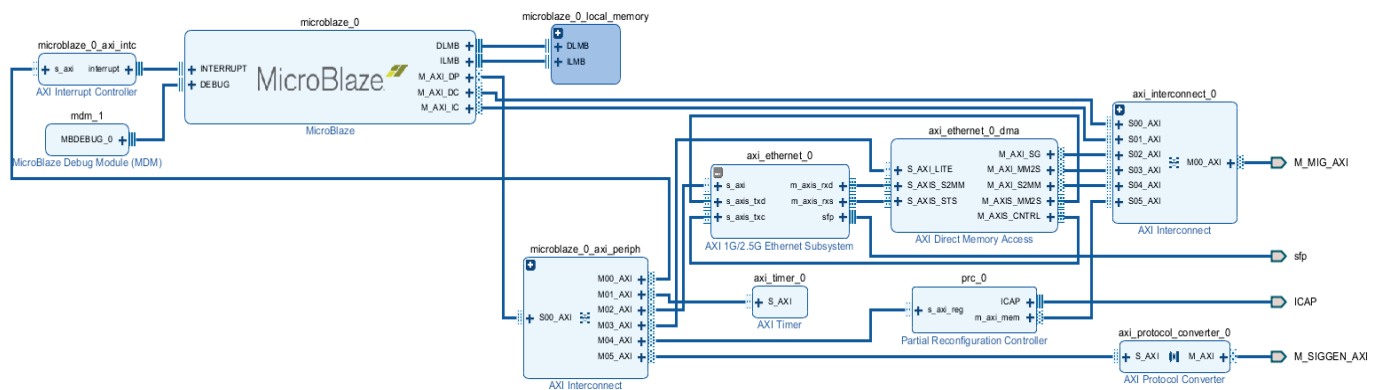
The use of a MicroBlaze soft processor core instead of the ZPU processor that already exists in the RFNoC HDL firmware was decided due to several design and implementation reasons. First, the MicroBlaze processor is a well-documented Xilinx IP core with detailed design guidelines and examples. The Xilinx software development kit (SDK) offers standard development tools, design automation options, and application examples for the MicroBlaze processor. In this respect, reusing the ZPU-embedded soft processor would have probably required additional effort (e.g., due to limited related documentation). Moreover, the coexistence of software functions in the ZPU processor serving both the RFNoC and ReproRun could imply the use of a real-time operating system (RTOS), increasing again the overall design effort. Hence, the MicroBlaze microprocessor core was added to accelerate and simplify the development and also to decouple the processing load of the software functions running for the RFNoC-ZPU- and ReproRun-MicroBlaze-. The MicroBlaze processor is hosting a TCP/IP stack solution, the bare-metal firmware, and a subset of the SDR1 REWIRE functionality. An optimal system design would have required a redesign of the RFNoC framework to use a single Ethernet interface for the RFNoC and

ReproRun and a single embedded microprocessor running RTOS. However, such redesign falls beyond the scope of this paper.

The access to the DDR3 memory of the SDR1 had to be shared among the RFNoC and ReproRun without generating any read or write conflicts. This was made feasible by using an AXI Interconnect IP core of Xilinx and adjusting the memory-mapped address ranges for each of the two SDRAM accesses. Hence, the existing MIG [43] configuration of the RFNoC Verilog firmware was reutilized and suitably modified.

As already mentioned before, the PRC IP provides management functions for the PR designs. Upon a trigger event initiated in the PL or PS, the PRC fetches partial bitstreams from an external memory and delivers them to the ICAP. The PRC also assists with logical decoupling and startup events, customizable per reconfigurable partition. By default, the PRC operates with RM known a priori to the controller. An AXI4-Lite register interface allows the core to be reconfigured at run-time, which means that the PRC can also be used in systems where the RMs could change at run-time. The core can be customized for a number of RMs per Virtual Socket and interface. The PRC in SDR2 was configured in the same way we did so in the case of SDR1.

The Vivado block design of the SDR2 with the key building blocks of ReproRun is shown in Figure 7 (e.g., MicroBlaze, PRC, and AXI 1G/2.5G Ethernet Subsystem IP cores). The shared access to the SDR2 SDRAM takes place in two nested Vivado block designs. Other parts of ReproRun include blocks developed in Verilog code and a series of modifications applied to the native RFNoC Verilog source code.



**Figure 7.** The top-level Vivado Block Design of SDR2.

The static FPGA design (static bitstream) is the combination of the modified RFNoC Verilog firmware and the HDL code of ReproRun. Two counters were developed to show the uninterrupted operation of the static bitstream (i.e., counting up and down) and their operation was mapped to a Xilinx virtual input–output (VIO) IP core to monitor them in real-time. In order to demonstrate the partial reconfiguration, we have selected the `rfnoc_siggen.grc` GNU radio companion (GRC) example for RFNoC (Figure 8). This GRC RFNoC example uses a coordinate rotation digital computer (CORDIC) function to produce a sinusoid in the FPGA device of SDR2 and then passes the I and Q samples to the host PC, where the samples are processed and visualized in the GRC project (e.g., time and frequency domain analysis); some user set parameters, such as the digital gain of the signal, can be manually modified from the host PC.

The definition of the RR for the partial reconfiguration cannot be applied at the boundaries of the RFNoC blocks. By trying to do so, we noticed that when invoking RFNoC from the host it was not responsive. Hence, the RR had to be defined in one of the DSP functions of an RFNoC computation engine (CE), denoted as User IP in Figure 8. This in turn resulted in several limitations for the PR, when for instance we want to use one RR to run different partial bitstreams (e.g., size of RR, I/O compatibility of the different partial reconfigurable functions targeting the RR). The partial bitstream was produced following the standard PR design guidelines of Xilinx and the instructions for building

a user-defined RFNoC CE [44]. For the case of the `rfnoc_siggen.grc` example, we have applied the PR in the CORDIC function residing in this CE. The static bitstream was also modified to provide programming access to the user registers (shown in Figure 9) from the MicroBlaze processor through an AXI protocol converter IP core (from AXI4 to AXI4Lite) and a series of state machines to manage the AXI4Lite transactions. This feature allows the modified version of REWIRE to change at run-time the gain of the signal in the `rfnoc_siggen.grc` example. Certain registers are monitored through VIO cores for testing and debugging purposes.

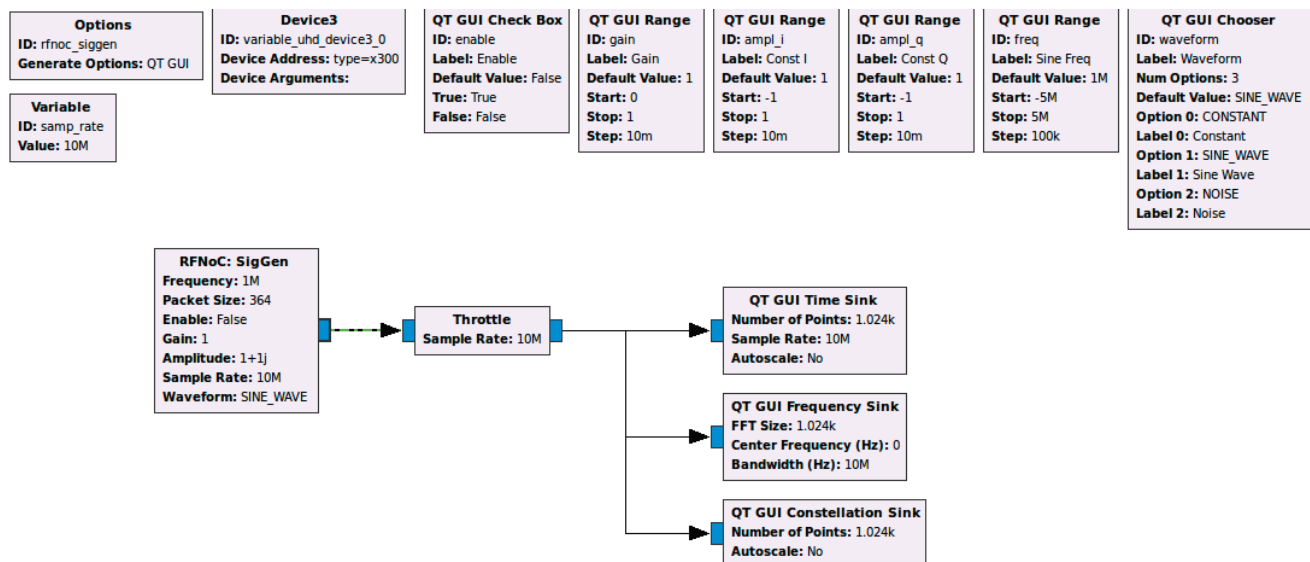


Figure 8. The `rfnoc_siggen.grc` GRC example of RFNoC.

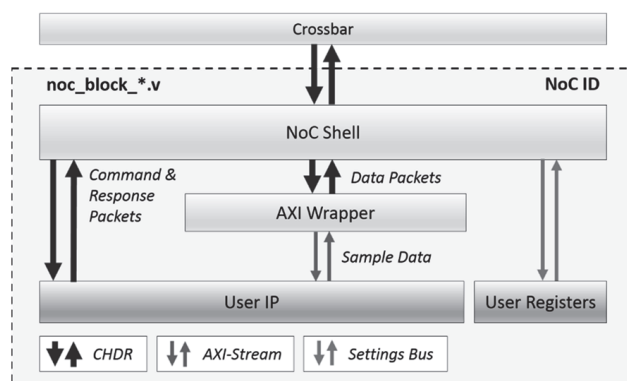
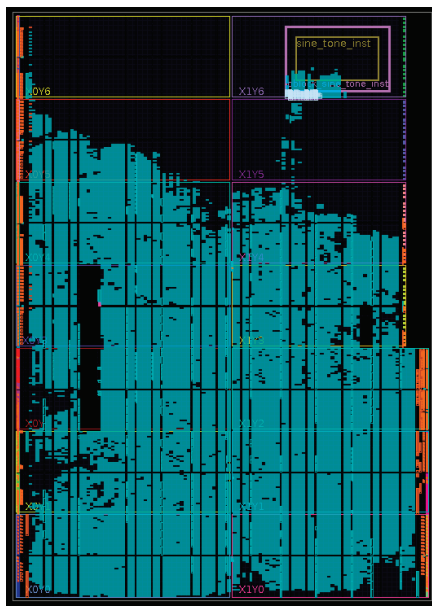


Figure 9. The RFNoC's Computation Engine.

A view of the place and routed FPGA implementation (i.e., static bitstream with a single RR) and the detailed implementation results are shown in Figure 10 and Table 2. The FPGA implementation is medium-dense, it features various clock domains with some resources being overused (e.g., all the available BUFGs are used, making it impossible to use ILA cores to debug the system).





**Figure 10.** The routed FPGA design in SDR2 showing the RR.

**Table 2.** Implementation results of the SDR2 FPGA design.

Slice LUTs (254,200)	Slice Regs (508,400)	F7/F8 Muxes (127,100/63,550)	Slice (63,550)	LUT as Logic/Memory/FFs (254,200/90,600/254,200)	BRAMs (795)	DSPs (1540)	Bonded IOB (484)
97,574	118,633	678/11	38,971	85,673/11,901/50,398	305	73	414

## 6. Experimental Validation

### 6.1. SDR1 Experimental Setup

The SDR1 setup included a mainstream laptop, the Xilinx ZC706 development board, and the AD-FMCOMMS-2/3 RF front-end evaluation board featuring the popular AD9361 RF transceiver IC (RFIC). The laptop featured an Intel Core i5-8300H central processing unit (CPU) running at 2.30 GHz, 16 GB of RAM, and a 64-bit Ubuntu 18.04.1 operating system. For the experiments with the SDR1, the following software components were installed in the laptop: Vivado System Edition 2017.4 with the Xilinx SDK, GNU compiler collection and GNU *binutils*, and ARM cross-compilation toolchain. The joint test action group (JTAG) and UART ports of the development board were connected to the laptop. Since Linux running on the SDR1 is basically an Ubuntu distro, we have also connected a keyboard and a monitor to it. The Ethernet port was connected to a switch in order to bring up a network connection on the SDR1 and to enable remote access through a secure socket shell (SSH) protocol session.

An overview of the experimental setup with all the functional blocks is shown in Figure 10. The Ubuntu OS boots from an SD card. The static bitstream is programmed by the first stage bootloader, which together with other files forms the Xilinx bootable image, *BOOT.BIN* (also stored in the SD card). At the end of the start-up of Ubuntu, two applications provided by Analog Devices are automatically launched (i.e., the Oscilloscope and a configuration graphical user interface (GUI) for controlling the AD9361 RFIC). The connections described above allow us to flexibly configure the platform using the keyboard and the monitor. We have configured the AD-FMCOMMS-2/3 RF frontend using one of the standard 5 G configurations stored in an SD card (i.e., waveform playback); in concrete, the one supporting 1.4 MHz bandwidth signal.

The 100 Mb Ethernet port allows fetching partial bitstreams and firmware objects from a remote location and placing them in the root filesystem. As Figure 11 shows, the REWIRE is comprised of two parts running in separate Processors: one in a Linux environment and the other in bare-metal environment under an AMP system configuration.



The static part of the PL includes among others the block-controlling GPIO LEDs, which are aimed at showing the uninterrupted operation of the platform before, during, and after the partial reconfiguration (i.e., LEDs are blinking all the time). The REWIRE part running on Processor 1 accepts a few command line parameters specifying the ID of the RM to be reconfigured, the path to the bitstream, and the path to the object files. When started without command line parameters (or with incorrect parameters), the REWIRE shows the following usage message:

Usage: ./rewire -i <RM\_IDX> [-b <FILE>] [-o <FILE>]

Options:

-i <RM\_IDX> -specify index of RM which will be configured with a passed bitstream (-b)

-b <FILE> -specify path to a bitstream file

-o <FILE> -specify path to an object file

Examples:

(1) ./rewire -i 0 -b pr0\_rm0\_leds\_blink.bin -o blinking\_freq.o

Description:

Load .bin into PL DDR and write its size and address to a PRC config register associated with RM0. Do the FPGA PR reconfiguration, FW reconfiguration and finally jump to a new FW code.

(2) ./rewire -i 1 -b pr0\_rm1\_leds\_shift.bin -o leds\_shift\_config.o

Description:

Do the same using different input files, but associate bitstream with RM1

(3) ./rewire -i 0 -o blinking\_freq\_slow.o

Description

Program FPGA PR area with RM0 bitstream (preloaded) and execute specified.o firmware

(4) ./rewire -i 1

Description:

Program FPGA with RM1 bitstream (do not perform FW reconfiguration)

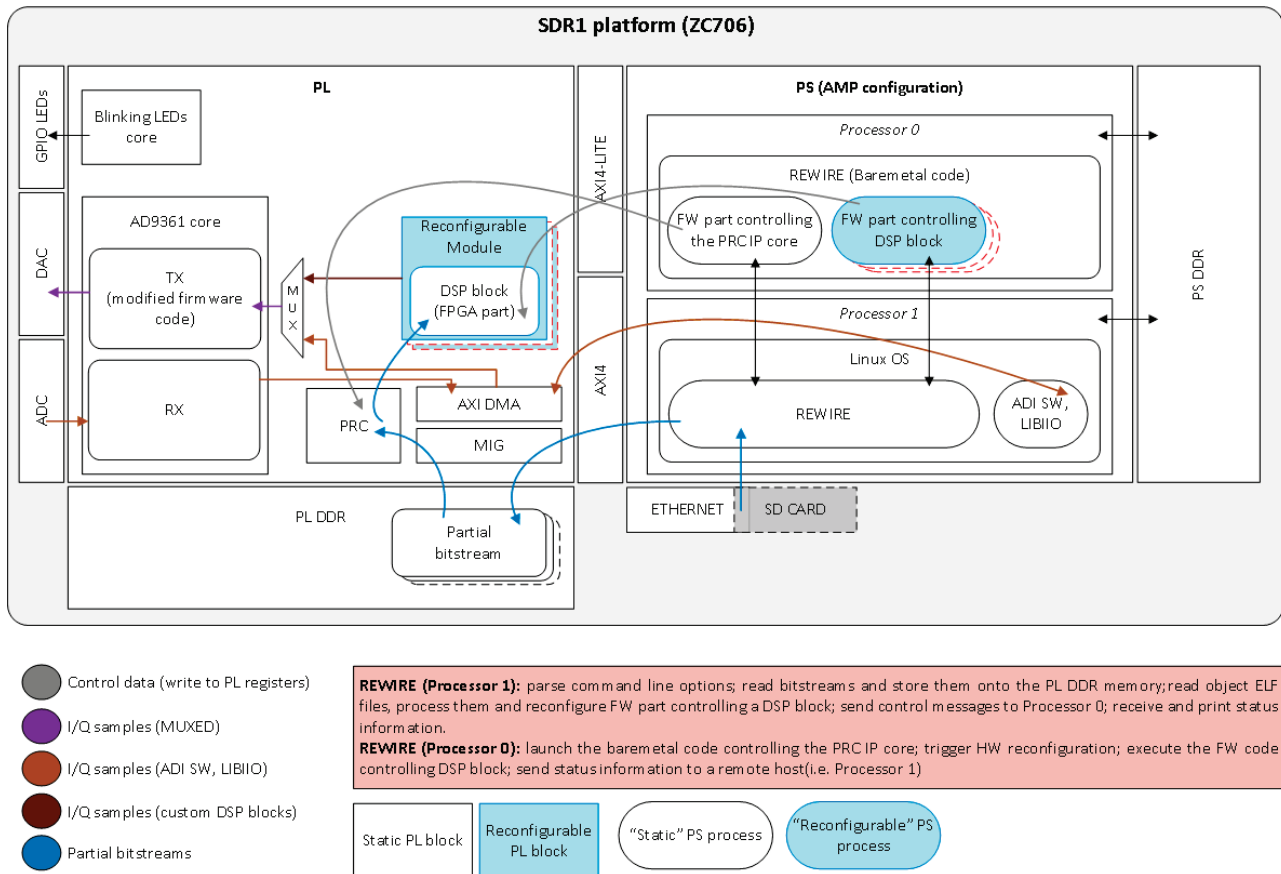


Figure 11. Overview of the experimental setup of SDR1.

As can be seen above, providing a path to the partial bitstream and object file is optional. As long as different configurations are initially loaded for different RMs, REWIRE allows to switch between them specifying the ID of the RM to be used in the reconfigurable region. Of course, the described functionality must be supported by the other part of the REWIRE running in the bare-metal environment. Processor 1 sends to Processor 0 a reconfiguration message complying with the predefined format known to both processors. Depending on the user's input, this message may contain the information about the new bitstream to be reprogrammed and in particular (i) its size and the SDRAM address where it has been loaded; (ii) what RM must be associated with this bitstream; (iii) whether the firmware has been reconfigured and thus must be executed after the PRC IP core finishes its task.

The effect of programming a new partial bitstream and/or reconfiguring the firmware can be observed in the ADI Oscilloscope or in the spectrum analyzer connected to the RF output port of the AD-FMCOMMS-2/3. The following steps produce the output of ReproRun:

- On the bootup of the SDR1, we need to program the filter coefficients with a 1.4 MHz 5 G NR-like config file using Analog Devices IIO Oscilloscope GUI (Figure 12) and disable outputs, i.e., the DAC path. In some tests, not disabling the transmit path at the start results in a DC component presented at the output.
- We load a waveform from a set of predefined signals, using the DAC buffer output option of the IIO Oscilloscope GUI (e.g., 5GNR\_5MHz signal), and we check the spectrum in IIO Oscilloscope GUI or in a spectrum analyzer connected to the TX1 port of the AD-FMCOMMD-2/3.
- We open the Vivado hardware manager, and we assign the dbg\_probes.ltx file in the probes.
- We set the value of the VIO register from logical '0' to '1'. This register is basically the control signal of a multiplexor that allows our custom PL block to provide data to the DAC.
- As the RR is empty there is basically no output signal in the IIO Oscilloscope GUI (the power of the output signal visually decreases) or the SA.
- We use the REWIRE solution to program a DDS DSP block inside RR (one of the two partial bitstreams prepared for this experimental validation):
  - `./rewire -i 0 -b ../pr_bs/DSP_DDS.bin`
- A tone signal appears in the SA, as shown in Figure 13. The frequency was set to 100 KHz by default, but it can be changed by doing a firmware reconfiguration (as explained later). You can also switch to a time-domain view in the IIO Oscilloscope GUI (i.e., by setting the resolution bandwidth parameter at 56 kHz to conveniently visualize the signal) or connect an oscilloscope equipment to the TX1 port of the AD-FMCOMMD-2/3 (Figure 14).
- At this stage the RM can be changed in real-time by running the REWIRE again, this time using the 5 G NR-like 1.4 MHz playback block:
  - `./rewire -i 1 -b ../pr_bs/DSP_5GNR.bin`
- When the reconfiguration is done, the visualized spectrum is the one shown in Figure 15.
- We can switch back to the first configuration by simply executing the following command:
  - `./rewire -i 0`
- As far as the firmware reconfiguration is concerned, we have built two partial firmware objects containing different DDS configurations, i.e., each one of them sets different values for a synthesized tone frequency.
- By executing the following commands, the tone frequency is reconfigured on the fly:
  - `./rewire -i 0 -o ../pr_fw/fw_dds_config_1.o`
  - `./rewire -i 0 -o ../pr_fw/fw_dds_config_2.o`

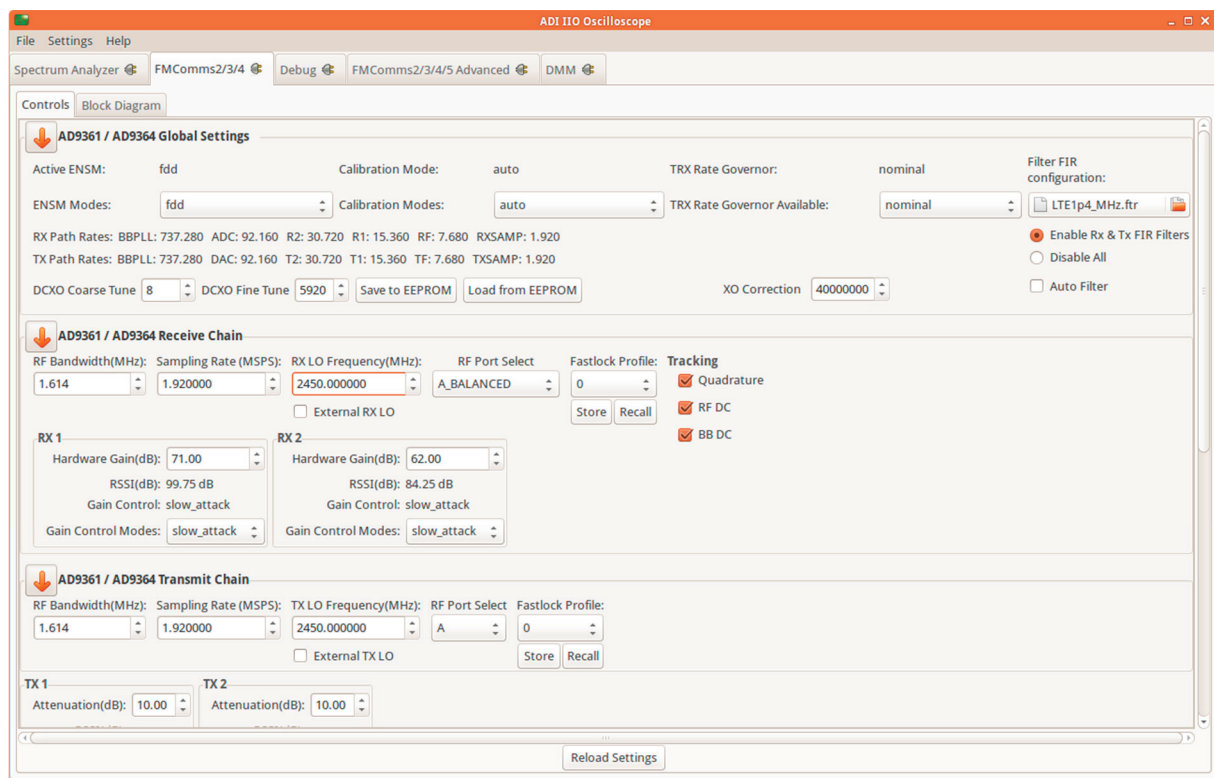


Figure 12. A screen capture of some of the settings applied in the Analog Devices IIO Oscilloscope GUI.

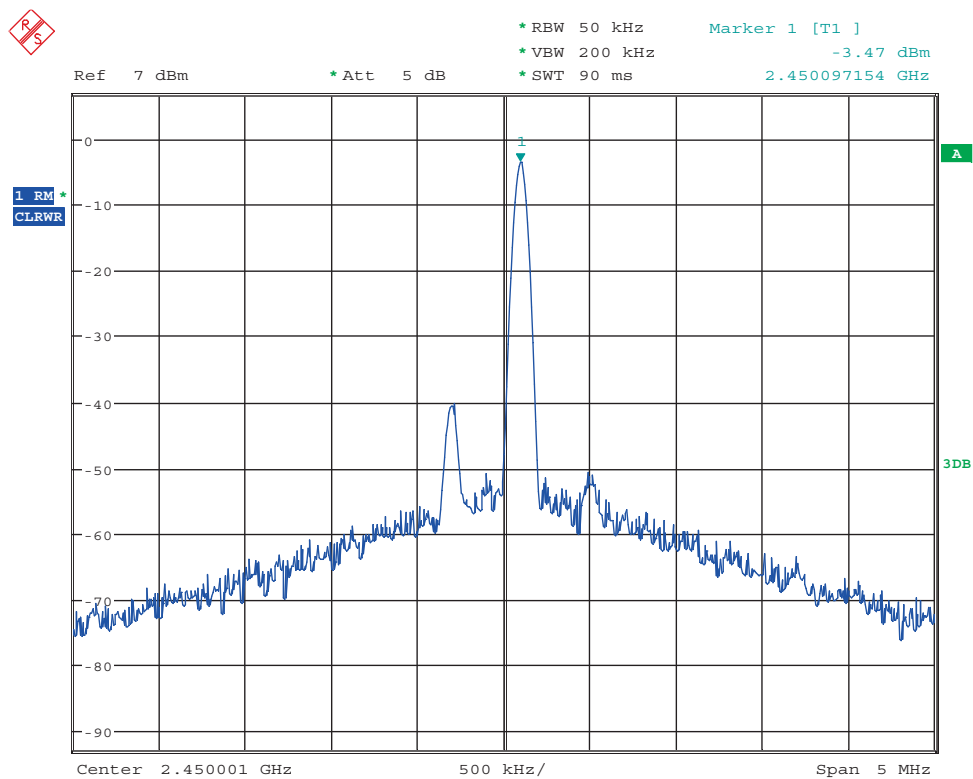
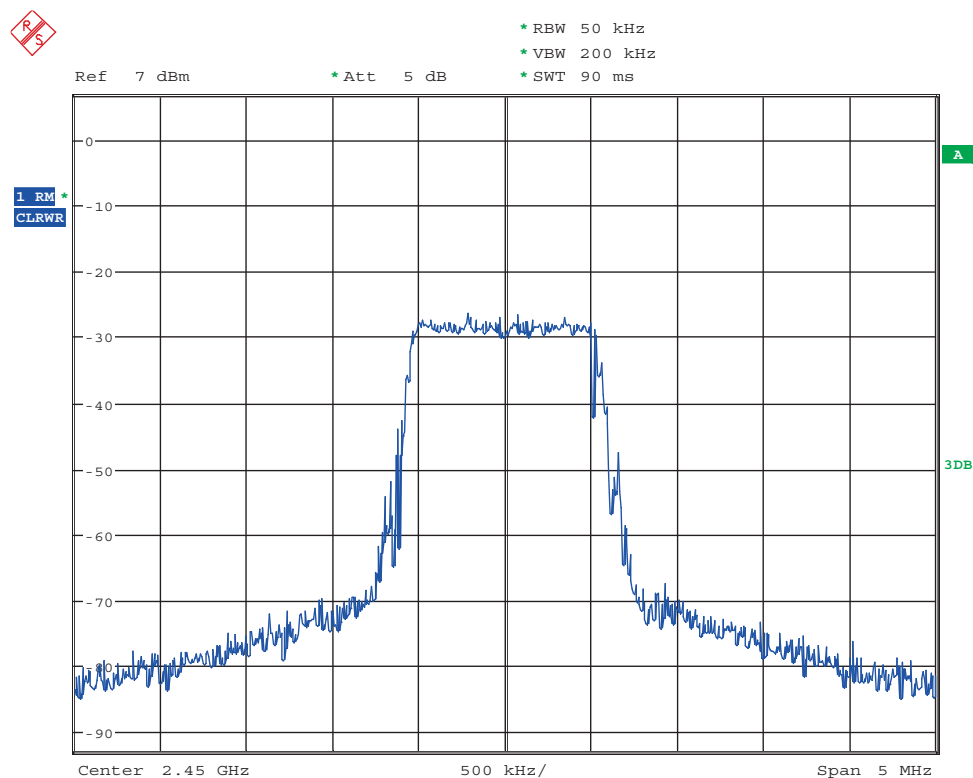


Figure 13. The output of the AD-FMCOMMS-2/3 when the REWIRE configures the DSP\_DDS.bin.



**Figure 14.** A time-domain screen-capture of an oscilloscope connected to the TX1 port of the AD-FMCOMMD-2/3, when the REWIRE configures the DSP\_DDS.bin partial bitstream.



**Figure 15.** The output of the AD-FMCOMMS-2/3 when the REWIRE configures the DSP\_5GNR.bin partial bitstream.

In the following, we also present the example output of the REWIRE program execution, including the initial AMP configuration:

```

root@analog:/home/analog/reprorun/rewire# echo rewire_cpu0 >
/sys/class/remoteproc/remoteproc0/firmware
root@analog:/home/analog/reprorun/rewire# echo start >
/sys/class/remoteproc/remoteproc0/state
CPU0: Starting application...
CPU0: VSM 0 status:
CPU0: Try to init remoteproc resource
CPU0: Init remoteproc resource succeeded
CPU0: Waiting for events...
root@analog:/home/analog/reprorun/rewire# modprobe rpmsg_user_dev_driver
root@analog:/home/analog/reprorun/rewire# ./rewire -i 0 -b ../pr_bs/DSP_DDS.bin
-- Starting REWIRE app --
[INFO] Opening rpmsg dev...success.
[INFO] Successfully opened .BIN file
[INFO] Finished transferring bitstream to PL DDR
CPU0: Reconfiguration Msg received
      [LOG] received BS_ADDR = 80A00000
      [LOG] received BS_SIZE = CB7DC
      [LOG] received RM_ID    = 0
CPU0: VSM status after SW_TRIGGER:
      Mode: ACTIVE
      STATE: FULL (7)
      RM_ID: 0
      BS_ID: 0
      ERROR: NO ERROR (0)
[INFO] CPU1: received message: RECONFIG OK!
root@analog:/home/analog/reprorun/rewire#

```

## 6.2. SDR2 Experimental Setup

The experimental setup included a high-performance computer acting as the host featuring an Intel Core i7-7700K CPU running at 4.2 GHz, with 32 GB RAM memory. This host computer also includes a: 1 GigE interface used by ReproRun (residing in the motherboard of the PC); a dual 10 Gigabit Ethernet PCIe Card used by the RFNoC framework (i.e., one of the two interfaces); another 1 GigE interface providing the Internet access.

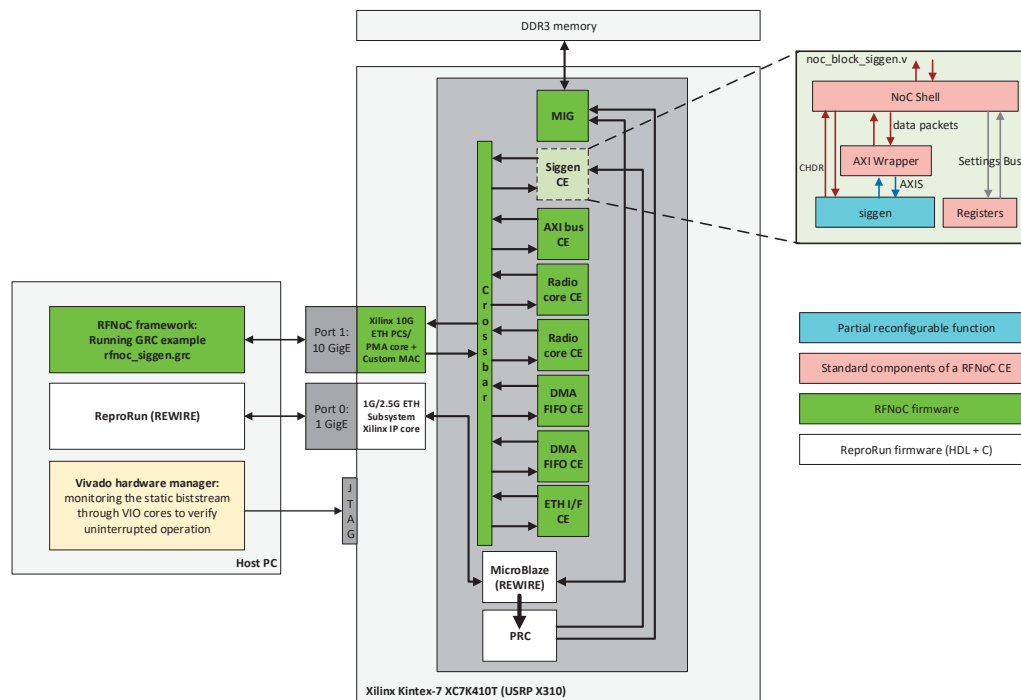
The host is a dual boot system (Fedora and Windows 10). In the Windows partition, we installed the Vivado System Edition 2017.4 and the Xilinx SDK. An Ubuntu virtual box (VB) machine was also installed in the Windows partition. The RFNoC framework with all its dependencies was installed in the VB (i.e., development branch, version 4.0.0).

The following communication interfaces of the USRP X310 device were connected to the host PC:

- The Ethernet port 0 is configured as 1 GigE.
- The Ethernet port 1 is configured as 10 GigE.
- The JTAG port is used for programming the FPGA device, for early system validation, and for in-system debugging through the Vivado Hardware manager.

The two bitstreams, the static and partial one, are stored in the host PC where Processor 1 runs specific REWIRE functions. We program first the static part of the FPGA design using the Vivado Hardware Manager. As already mentioned before, the static bitstream includes two counters mapped to a VIO core in order to monitor the uninterrupted operation of the static FPGA area during and after the configuration of the partial bitstream. The other part of REWIRE runs in the bare-metal firmware of Processor 0 (i.e., MicroBlaze processor configured in the FPGA device of the X310 USRP platform). This REWIRE part receives commands from the host, specifying the name of the partial bitstream, which can be fetched from a TFTP server running at the host. The command message may also provide control

information for the firmware reconfiguration. In this case, Processor 0 receives also a firmware object, which is serialized together with the rest of the commands and data forming the full reconfiguration message. The REWIRE stores bitstreams locally in the SDRAM memory of the SDR2, reprograms registers of the PRC core triggering it to start the PR procedure, and in case of success the REWIRE jumps to a new firmware if this was specified by the control commands; this firmware can access the control registers of a newly programmed CE in order to set/change the digital gain of the signal. At the same time, the `rfnoc_siggen.grc` RFNoC example must run at the host side, which shows the result of the reconfiguration in real time. The general overview of the experimental setup is shown in Figure 16.



**Figure 16.** Overview of the experimental setup of SDR2.

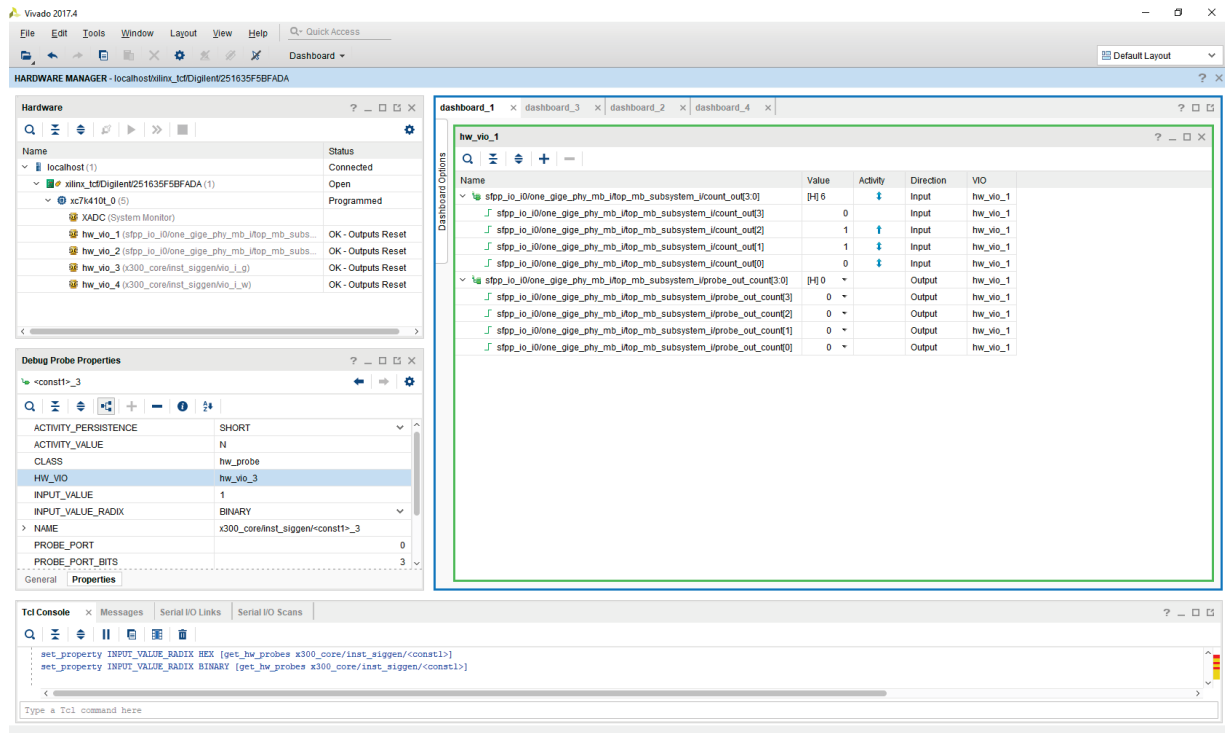
In the first stage of testing our PR solution in SDR2, none of the examples included in the RFNoC framework and running in the GRC could be used in a run-time PR context. In fact, a specific configuration-invoking procedure had to be followed to bypass the deadlocks during the PR (i.e., probably produced in the communication between the ZPU and the host PC) and bring up the correct operation of the GRC example.

The following steps produce the output of ReproRun:

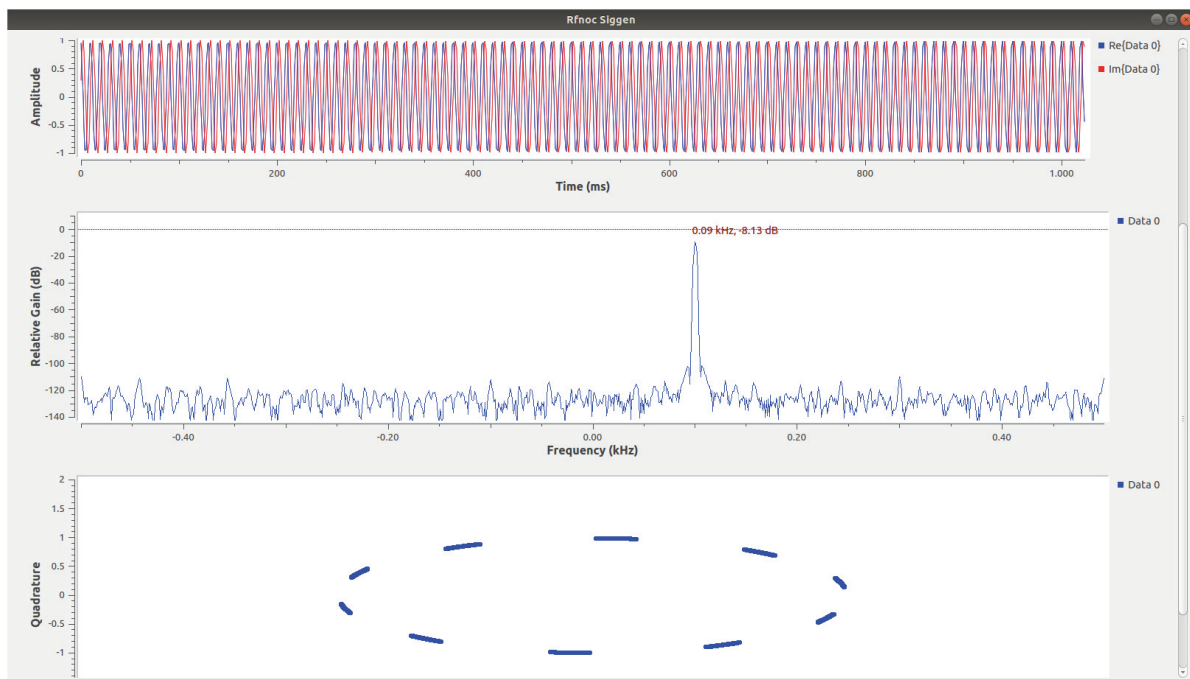
- The static bitstream is programmed using the Vivado Hardware Manager (Figure 17).
- We observe the operation of the VIO core that monitors two counters (up and down).
- The GRC RFNoC example `rfnoc_siggen.grc` is launched setting before 1K in the sample rate box and the IP address in the device arguments (in our case 192.168.40.2).
- Since the SigGen example includes an RR where the partial bitstream will be configured (the latter includes the CORDIC function of the example), the GRC project GUI initially does not produce an output.
- We run the following command to check that RFNoC is up and running: `uhd_usrp_probe --args="type=x300,addr=192.168.40.2"`.
- After receiving commands from the host, REWIRE fetches the partial bitstream making use of a TFTP server; Processor 0 also receives a firmware object.
- Bitstreams are stored in the SDRAM memory of the SDR2.
- REWIRE programs register the PRC core triggering it to start the PR procedure.



- Once the partial bitstream is configured, we can observe in real time that the GRC GUI provides the time and frequency domain signal analysis (Figure 18).
- REWIRE jumps to a new firmware, which allows to access the control registers of the SigGen RFNoC example and modifying the digital gain of the signal.
- Throughout the mentioned procedure, we can observe that the counters monitored by the VIO core operate uninterrupted.



**Figure 17.** The Vivado Hardware Manager programs the static bitstream and shows the monitored VIO probes.



**Figure 18.** After configuring the partial bitstream, the SigGen RFNoC example produces the correct/expected output.

### 6.3. Discussion

In broad terms, the design and features of ReproRun are served more efficiently by SDR1. The following variations are present in the two SDR platforms due to their inherent hardware and firmware features:

In SDR1, one of the cores of the ARM A9 (processor 0) hosts a bare-metal application linked with a partial bitstream and other necessary ReproRun functions that enable the run-time firmware reconfiguration, whereas the other core, Processor 1, hosts the Linux distribution of Analog Devices. The native version of the AMP framework was only applied in SDR1 since the architecture of the FPGA device in SDR2 does not favor its full use. For this reason, ReproRun was designed in a different way in the case of SDR2, since processor 0 was implemented using a MicroBlaze soft microprocessor IP core, while processor 1 was represented by a host processor. As a result, the REWIRE controller in SDR2 features a subset of the functionalities of its SDR1 counterpart.

In SDR2, it was required to integrate the PR glue logic with the RFNoC firmware without compromising its native operation. The RFNoC FPGA images running at the SDR2 can be built to include up to ten DSP functions, which then can be interconnected based on a GRC project. The latter provides a hardware-in-the-loop environment. In SDR1 Analog Devices provides the necessary HDL reference design, a Linux distribution running at the PS, and several other tools for a user to be able to prototype its own hardware-software (HW-SW) system. It is important to highlight that the user code integration in SDR1 is simpler and less dependable compared to SDR2 which uses the RFNoC framework. The native functionality of the RFNoC HDL firmware of SDR2 was modified and extended to contemplate ReproRun, resulting in an elevated code integration effort. As a general conclusion, it has to be noted that the conceptual idea of the RFNoC framework is to be used in its native version and not modified to add PR features. On top of that the SDR2 features an important limitation for being able to apply run-time PR, as it is further detailed in Appendix A.

## 7. Conclusions and Future Work

In this work, we propose a novel framework that leverages the partial reconfiguration solution of AMD-Xilinx along with the AMP framework in order to provide a seamless run-time reconfiguration of interlinked functions executed in the PL and PS portion of FPGA devices.

The heart of ReproRun is a purpose-built controller denoted as REWIRE that is able to micro-orchestrate the PL-based partial bitstreams along with their corresponding PS-based firmware. REWIRE acts as a sort of adaptive linker for the object files and interacts both with the native partial reconfiguration building blocks of AMD-Xilinx, as well as with the AMP framework. In a Zynq 7000 family of devices, REWIRE resides in the same processor core that hosts the operating system (e.g., PetaLinux), whereas the reconfigurable software application is executed as a bare-metal one in the other processor core of the FPGA system.

ReproRun automates the management of all the processes related to the AMP and partial reconfiguration functionality and allows for replacing at run-time the software object file or its interlinked partial bitstream counterpart, without downtimes of the combined function executed simultaneously in the PS and PL. Hence, ReproRun can be considered the run-time, dynamic re-programming extension of the hardware-software co-design flow that is typically used during the implementation of an FPGA system featuring embedded microprocessors. To showcase the versatility of the proposed solution, two different types of FPGA devices were used featuring different functional characteristics, interfaces, and programming procedures. During the testing and debugging stage of ReproRun, functional differences and limitations were revealed between the two SDR boards.

The result of this combined run-time reconfiguration of hardware-accelerated and firmware functions is a novel run-time programming framework that could serve the emerging needs of programmable 6G radio and network technologies. For instance, ReproRun can be applied to FPGA-based SoC devices encountered beyond 5G disaggregated

RAN architectures like the one specified by the O-RAN alliance to provide an agile re-configuration framework of the high and low physical layer in order to achieve service requirements related to performance, low latency, and energy efficiency.

Looking forward and in the short term, it is planned to migrate ReproRun to the DFX framework using UltraScale+ Zynq as the target device family [1]. The combination of the DFX with the isolation design flow makes this extension very promising to efficiently address the long-standing security concerns related to PR. ReproRun is also planned to be integrated into a virtualization framework and exposed to an intelligent controller at the RAN level that will cognitively take decisions for a flexible hardware–software run-time reconfiguration of network functions hosted in FPGA-based SoC devices.

Finally, considering the insecure nature of TFTP, which was adopted in this work as a simplified method to fetch partial bitstreams and object files from a remote location (i.e., accelerating the validation and testing stages of ReproRun), the goal will be to replace it with a safer networking connectivity solution. Towards this end, a secure shell (SSH) FTP solution will be leveraged to offer the full security and authentication functionality of SSH for the file transfer of the partial bitstreams and object files. On top of that and given the foreseen migration of ReproRun to UltraScale + Zynq devices, it is planned to exploit their bitstream encryption system which uses the advanced encryption standard Galois counter mode (AES-GCM) authenticated encryption algorithm. The AES-GCM encryption standard supports built-in authentication that will increase the security of the ReproRun framework, since without knowledge of the AES-GCM key, the partial bitstream cannot be modified or forged. Thus, this type of authentication will guarantee both data integrity and authenticity of the partial bitstreams. Upon authentication failure, the reconfigurable region will not start up and ReproRun will provide a fallback mechanism (i.e., a fail-safe partial bitstream will be loaded) to guarantee the availability of dependable services running at the PS side of the device.

**Author Contributions:** Conceptualization, N.B.; methodology, N.B. and J.R.F.; software, J.R.F., N.B. and A.R.V.; validation, J.R.F., N.B., D.L.-B. and A.R.V.; formal analysis, N.B. and J.R.F.; investigation, N.B.; resources, N.B., J.R.F., D.L.-B. and A.R.V.; writing—original draft preparation, N.B.; writing—review and editing, N.B., J.R.F., D.L.-B. and A.A.; supervision, N.B.; project administration, N.B.; funding acquisition, N.B.; All authors have read and agreed to the published version of the manuscript.

**Funding:** The work in this paper was supported in part by the Horizon Europe SNS JU VERGE project funded by the European Commission (Grant agreement ID 101096034), the project ORIGIN (PID2020-113832RB-C22) funded by the Agencia Estatal de Investigación MCIN/AEI/10.13039/501100011033 at the Ministerio de Ciencia, Innovación y Universidades from Gobierno de España, the project FREE6G-RegEdge (TSI-063000-2021-144) funded by the Ministerio de Asuntos Económicos y Transformación Digital from Gobierno de España, and the grant 2021 SGR 00772 funded by the Agència de Gestió d'Ajuts Universitaris i de Recerca from the Generalitat de Catalunya.

**Data Availability Statement:** Data is unavailable due to IP exploitation policies that apply.

**Acknowledgments:** The authors of the paper would like to thank Oriol Font-Bach and Pavel Harbanau for their contributions to this work. We would also like to thank Ingrid Moerman (IMEC) and her team at Ghent University (Xianjun Jiao, Wei Liu), which gave us the opportunity to develop this work in the context of the H2020 Orchestration and Reconfiguration Control Architecture (ORCA) project. CTTC was subcontracted by IMEC and entered the ORCA consortium as a third party. The experimental validation of this work was also conducted remotely in IMEC's w-iLab.t lab and was successfully showcased during an ORCA project review conducted by EC.

**Conflicts of Interest:** Angelos Antonopoulos was employed by the Nearby Computing S.L. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Appendix A

In SDR2, we have made use of the Xilinx PRC IP core and the ICAP port to apply the run-time partial reconfiguration for a design clocked at 100 MHz. Based on the test, validation, and extensive debugging of the system, we have spotted a hardware limitation in the USRP X310 board that deters it from running the PR flow using the PRC and ICAP port. In contrast, the PR works well in SDR2 when we reconfigure the same static and partial bitstreams through the JTAG port (i.e., using the Vivado Hardware Manager). The following points describe the debugging procedure and the limitation we found in the USRP X310 board:

- A process in MicroBlaze fetches the partial bitstream (.BIN file format) from the host PC via TFTP and copies the contents in the DDR3 memory.
  - Using a breakpoint in the executed code, we are able to verify in SDK that the contents in the DDR3 are the correct ones.
  - The PRC is triggered by a process in the MicroBlaze to access the .BIN stored in the SDRAM through its m\_axi\_mem port. Using an ILA core, we monitor that the transaction takes place without any problems.
- The PRC IP core pushes the .BIN partial bitstream to the ICAP port to apply partial reconfiguration. Another ILA debug core monitors the ICAP port (Figure A1). By reading the ICAP “O” Port bits, we check the status bits as follows (see also Table A1):
  - Initial status: 1001
  - The ICAP receives the sync word: 1101
  - The ICAP receives the DESYNC command: 0101
  - The ‘icap\_csib’ port produces a value of ‘1’: 0001
  - The latter means that the ICAP indicates an error directly after receiving all the data of the partial bitstream (.BIN) as it could be seen in Figure 18 (i.e., ILA screen capture).
- At that stage of debugging, we went deeper to analyze why the partial reconfiguration was not completed successfully.
  - In order to make the partial reconfiguration work through the ICAP interface, we need to activate the Slave SelectMAP configuration interface in our system [45,46]. Table 2-1 in [45] (page 17) indicates that ports M [2:0] need to have a ‘110’ value to set the Slave SelectMAP configuration.
  - However, when revising the schematic diagrams of the USRP X310 board (page 9), we realized that the M [2:0] ports take the value ‘111’, which in fact is a hardwired value (thus, not reprogrammable by software).
  - This value must change in order to be able to use the PRC and ICAP on this SDR board.
  - Looking at the schematic diagrams, the only work around is to remove the R70, in which case, M2\_0 will be connected to GND through the R71 resistance (i.e., likewise setting the M [2:0] with the required ‘110’ value).

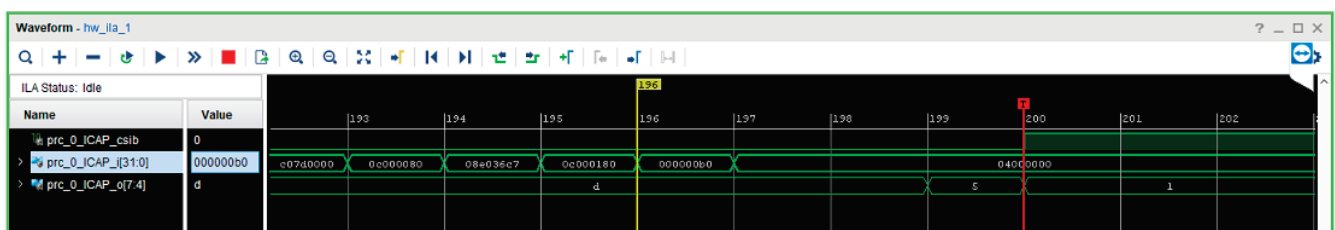


Figure A1. ILA debug core monitoring the ICAP port.

**Table A1.** the ICAP “O” Port bits [29].

ICAP “O” Port Bits	Status Bit	Meaning
O [7]	CFGERR_B	Configuration error (active-Low) 0 = A configuration error has occurred. 1 = No configuration error.
O [6]	DALIGN	Sync word received (active-High) 0 = No sync word received. 1 = Sync word received by interface logic
O [5]	RIP	Readback in progress (active-High) 0 = No readback in progress. 1 = A readback is in progress
O [4]	IN_ABORT_B	ABORT in progress (active-Low) 0 = Abort is in progress. 1 = No abort in progress.
O [3:0]	1	Reserved

## References

1. AMD. *Zynq UltraScale+ Device Technical Reference Manual*; AMD-Xilinx UG1085 (v2.3.1); AMD: Santa Clara, CA, USA, 2023.
2. Font-Bach, O.; Bartzoudis, N.; Pascual-Iserte, A.; Payaro, M.; Blanco, L.; López, D.; Molina, M. Interference Management in LTE-based HetNets: A Practical Approach. *Trans. Emerg. Telecommun. Technol.* **2015**, *26*, 195–215. [CrossRef]
3. Sabella, D.; Serrano, P.; Stea, G.; Viridis, A.; Tinnirello, I.; Giuliano, F.; Garlisi, D.; Vlacheas, P.; Demestichas, P.; Foteinos, V.; et al. Designing the 5G network infrastructure: A flexible and reconfigurable architecture based on context and content information. *EURASIP J. Wirel. Commun. Netw.* **2018**, *2018*, 199. [CrossRef]
4. Font-Bach, O.; Bartzoudis, N.; Miozzo, M.; Donato, C.; Harbanau, P.; Requena-Esteso, M.; López-Bueno, D.; Serrano, P.; Mangues-Bafalluy, J.; Payaró, M. Design, implementation and experimental validation of a 5G energy-aware reconfigurable hotspot. *Comput. Commun.* **2018**, *128*, 1–17. [CrossRef]
5. Gerzaguet, R.; Bartzoudis, N.; Baltar, L.G.; Berg, V.; Doré, J.B.; Kténas, D.; Font-Bach, O.; Mestre, X.; Payaró, M.; Färber, M.; et al. The 5G candidate waveform race: A comparison of complexity and performance. *EURASIP J. Wirel. Commun. Netw.* **2017**, *2017*, 13. [CrossRef]
6. Kartsakli, E.; Perez-Romero, J.; Sallent, O.; Bartzoudis, N.; Frascella, V.; Mohalik, S.K.; Metsch, T.; Antonopoulos, A.; Tuna, Ö.F.; Deng, Y.; et al. AI-Powered Edge Computing Evolution for Beyond 5G Communication Networks. In Proceedings of the 2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Gothenburg, Sweden, 6–9 June 2023; pp. 478–483.
7. Tessier, R.; Pocek, K.; Dehon, A. Reconfigurable Computing Architectures. *Proc. IEEE* **2015**, *103*, 332–354. [CrossRef]
8. Vipin, K.; Fahmy, S.A. FPGA Dynamic and Partial Reconfiguration. *ACM Comput. Surv.* **2018**, *51*, 1–39. [CrossRef]
9. Vipin, K.; Fahmy, S.A. ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq. *IEEE Embed. Syst. Lett.* **2014**, *6*, 41–44. [CrossRef]
10. Zamacola, R.; Otero, A.; Garcia, A.; De La Torre, E. An Integrated Approach and Tool Support for the Design of FPGA-Based Multi-Grain Reconfigurable Systems. *IEEE Access* **2020**, *8*, 202133–202152. [CrossRef]
11. Valente, G.; Di Mascio, T.; Pomante, L.; D’Andrea, G. Dynamic Partial Reconfiguration Profitability for Real-Time Systems. *IEEE Embed. Syst. Lett.* **2021**, *13*, 102–105. [CrossRef]
12. Elnaggar, R.; Karri, R.; Chakrabarty, K. Multi-Tenant FPGA-based Reconfigurable Systems: Attacks and Defenses. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 7–12. [CrossRef]
13. Dessouky, G.; Sadeghi, A.-R.; Zeitouni, S. SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities. In Proceedings of the 2021 IEEE European Symposium on Security and Privacy (EuroS&P), Vienna, Austria, 6–10 September 2021; pp. 487–506. [CrossRef]
14. Pundir, N.; Rahman, F.; Farahmandi, F.; Tehranipoor, M. What is All the FaaS About?—Remote Exploitation of FPGA-as-a-Service Platforms. *IACR Cryptol. ePrint Arch.* **2021**, *2021*, 746.
15. Pinneterre, S.; Chiotakis, S.; Paolino, M.; Raho, D. vFPGAManager: A Virtualization Framework for Orchestrated FPGA Accelerator Sharing in 5G Cloud Environments. In Proceedings of the 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Valencia, Spain, 6–8 June 2018; pp. 1–5. [CrossRef]
16. Bobda, C.; Mbongue, J.M.; Chow, P.; Ewais, M.; Tarafdar, N.; Vega, J.C.; Eguro, K.; Koch, D.; Handagala, S.; Leeser, M.; et al. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Trans. Reconfigurable Technol. Syst.* **2022**, *15*, 34. [CrossRef]



17. Ringlein, B.; Abel, F.; Diamantopoulos, D.; Weiss, B.; Hagleitner, C.; Reichenbach, M.; Fey, D. A Case for Function-as-a-Service with Disaggregated FPGAs. In Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 5–10 September 2021; pp. 333–344.
18. Cañas, J.M.; Fernández-Conde, J.; Vega, J.; Ordóñez, J. Reconfigurable Computing for Reactive Robotics Using Open-Source FPGAs. *Electronics* **2021**, *11*, 8. [CrossRef]
19. Elhosary, H.; Zakhari, M.H.; Elgammal, M.A.; Kelany, K.A.H.; El Ghany, M.A.A.; Salama, K.N.; Mostafa, H. Hardware Acceleration of High Sensitivity Power-Aware Epileptic Seizure Detection System Using Dynamic Partial Reconfiguration. *IEEE Access* **2021**, *9*, 75071–75081. [CrossRef]
20. Cervero, T.G.; Caba, J.; Lopez, S.; Dondo, J.D.; Sarmiento, R.; Rincon, F.; Lopez, J. A Scalable and Dynamically Reconfigurable FPGA-Based Embedded System for Real-Time Hyperspectral Unmixing. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *8*, 2894–2911. [CrossRef]
21. Wisniewski, R. Dynamic Partial Reconfiguration of Concurrent Control Systems Specified by Petri Nets and Implemented in Xilinx FPGA Devices. *IEEE Access* **2018**, *6*, 32376–32391. [CrossRef]
22. Youssef, E.; Elsimary, H.A.; El-Moursy, M.A.; Mostafa, H.; Khattab, A. Energy-Efficient Precision-Scaled CNN Implementation with Dynamic Partial Reconfiguration. *IEEE Access* **2022**, *10*, 95571–95584. [CrossRef]
23. Seyoum, B.; Pagani, M.; Biondi, A.; Balleri, S.; Buttazzo, G. Spatio-Temporal Optimization of Deep Neural Networks for Reconfigurable FPGA SoCs. *IEEE Trans. Comput.* **2020**, *70*, 1988–2000. [CrossRef]
24. Almeida, L.F.; Pereira, S.S.; Domingues, J.D.; Oliveira, A.S.R.; Carvalho, N.B. Moving NFV Toward the Antenna Through FPGA-Based Hardware Reconfiguration. *IEEE Commun. Lett.* **2022**, *27*, 342–346. [CrossRef]
25. Li, X.; Wang, X.; Liu, F.; Xu, H. DHL: Enabling Flexible Software Network Functions with FPGA Acceleration. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1–11. [CrossRef]
26. Sharma, G.P.; Tavernier, W.; Colle, D.; Pickavet, M. Dynamic Hardware-Acceleration of VNFs in NFV Environments. In Proceedings of the 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 10–13 June 2019; pp. 254–259. [CrossRef]
27. Pham, T.H.; Fahmy, S.A.; McLoughlin, I.V. An End-to-End Multi-Standard OFDM Transceiver Architecture Using FPGA Partial Reconfiguration. *IEEE Access* **2017**, *5*, 21002–21015. [CrossRef]
28. Hosny, S.; Elnader, E.; Gamal, M.; Hussien, A.; Mostafa, H. Multi-Partitioned Software Defined Radio Transceiver Based on Dynamic Partial Reconfiguration. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–4. [CrossRef]
29. AMD. *Vivado Design Suite User Guide, Partial Reconfiguration*; UG909 (v2017.4); AMD: Santa Clara, CA, USA, 2017.
30. AMD. *Vivado Design Suite Tutorial Partial Reconfiguration*; UG947 (v2017.4); AMD: Santa Clara, CA, USA, 2017.
31. AMD. *Partial Reconfiguration Controller v1.3 LogiCORE IP Product Guide, Vivado Design Suite*; PG193; AMD: Santa Clara, CA, USA, 2018.
32. Sarangi, A.; MacMahon, S.; Cherukupaly, U. *LightWeight IP Application Examples*; XAPP1026 (v5.1); Xilinx: San Jose, CA, USA, 2014.
33. Robinson, D. Loading Partial Bitstreams Using TFTP. In *Xilinx Application Note: Vivado Design Suite Partial Reconfiguration Flow*; XAPP1292 (v1.0); AMD: Santa Clara, CA, USA, 2016.
34. Kamaleldin, A.; Mohamed, A.; Nagy, A.; Gamal, Y.; Shalash, A.; Ismail, Y.; Mostafa, H. Design guidelines for the high-speed dynamic partial reconfiguration based software defined radio implementations on Xilinx Zynq FPGA. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.
35. AMD. *OpenAMP Framework for Zynq Devices*; Getting Started Guide UG1186 (v2017.1), Xilinx-AMD; AMD: Santa Clara, CA, USA, 2017.
36. AMD. *Libmetal and OpenAMP User Guide*; Xilinx User's Guide UG1186 (v2017.4); AMD: Santa Clara, CA, USA, 2018.
37. AMD. *Christian Kohn Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor*; XAPP1231 (v1.1); AMD: Santa Clara, CA, USA, 2015.
38. McDougall, J. *Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors*; XAPP1078 (v1.0); AMD: Santa Clara, CA, USA, 2013.
39. McDougall, J. *Simple AMP: Zynq SoC Cortex-A9 Bare-Metal System with MicroBlaze Processor*; XAPP1093 (v1.0.1); AMD: Santa Clara, CA, USA, 2014.
40. Kale, V. *Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development*; Xilinx White Paper WP469 (v1.0.1); Xilinx: San Jose, CA, USA, 2016.
41. AMD. *DDS Compiler v6.0 Xilinx LogiCORE IP Product Guide, Vivado Design Suite*; PG141; AMD: Santa Clara, CA, USA, 2021.
42. Herveille, O.R. Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores; Revision B.4, OpenCores. 2010. Available online: <http://www.opencores.org/> (accessed on 1 February 2024).
43. AMD. *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v4.2, User Guide UG586*; AMD: Santa Clara, CA, USA, 2018.
44. Getting Started with RFSoc Development, Application Note 823. Available online: [https://kb.ettus.com/Getting\\_Started\\_with\\_RFSoc\\_Development](https://kb.ettus.com/Getting_Started_with_RFSoc_Development) (accessed on 1 February 2024).



45. Nielson, M. *Using a Microprocessor to Configure 7 Series FPGAs via Slave Serial or Slave SelectMAP Mode*; Xilinx Application Note XAPP583 (v1.0); Xilinx: San Jose, CA, USA, 2012.
46. 7 Series FPGAs Configuration User Guide, UG470 (v1.16) 1 February 2023. Available online: [https://docs.xilinx.com/r/en-US/ug470\\_7Series\\_Config/7-Series-FPGAs-Configuration-User-Guide](https://docs.xilinx.com/r/en-US/ug470_7Series_Config/7-Series-FPGAs-Configuration-User-Guide) (accessed on 1 February 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.



MDPI AG  
Grosspeteranlage 5  
4052 Basel  
Switzerland  
Tel.: +41 61 683 77 34

*Electronics* Editorial Office  
E-mail: [electronics@mdpi.com](mailto:electronics@mdpi.com)  
[www.mdpi.com/journal/electronics](http://www.mdpi.com/journal/electronics)



Disclaimer/Publisher's Note: The title and front matter of this reprint are at the discretion of the Guest Editor. The publisher is not responsible for their content or any associated concerns. The statements, opinions and data contained in all individual articles are solely those of the individual Editor and contributors and not of MDPI. MDPI disclaims responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.





Academic Open  
Access Publishing

[mdpi.com](http://mdpi.com)

ISBN 978-3-7258-6102-6