energies

MDPI

Article

Embedding an Electrical System Real-Time Simulator with Floating-Point Arithmetic in a Field Programmable Gate Array

Janailson Queiroz [1,*], Sarah Carvalho [2], Camila Barros [2], Luciano Barros [2] and Daniel Barbosa [3]

[1] Electrical and Computer Engineering Graduate Program, Federal University of Rio Grande do Norte, Natal 59078-970, Brazil
[2] Computer Systems Department, Federal University of Paraíba, João Pessoa 58055-000, Brazil; sarahtoscano@eng.ci.ufpb.br (S.C.); camila.barros@ci.ufpb.br (C.B.); lsalesbarros@ci.ufpb.br (L.B.)
[3] Electrical and Computer Engineering Department, Federal University of Bahia, Salvador 40170-115, Brazil; dbarbosa@ufba.br
* Correspondence: janailson.maciel.111@ufrn.edu.br; Tel.: +55-84994220377

Abstract: Real-Time Digital Simulation (RTDS) is a powerful tool in modeling and analyzing electrical and drive systems because it provides an efficient and accurate process. There are several hardware devices for this type of simulation; however, their high costs have led to the increasing use of more affordable and reconfigurable technologies. In this context, many logic blocks and storage elements make the Field Programmable Gate Array (FPGA) an ideal device to perform RTDS. This work proposes a technique to embed a real-time digital simulator in an FPGA through Hardware Description Language (HDL) since it provides liberty in the architecture choice and no dependency on commercial ready-made hardware–software packages. The approach proposed focuses on system design developing with expression tree graph, synthesizing and verifying, prioritizing the performance and design accuracy concerning area and power consumption. Thus, the result acquisition occurs at a time step considered in real-time. A simulation of a direct current (DC) motor speed control has been incorporated into this work as an example of application, which includes the embedding and simulation of the electric machine and its drive system. Performance tests have shown that the developed simulator is real-time and makes possible realistic analysis of the interaction between the plant and its control. In addition, an idea of the hardware requirement for real-time simulation is proposed based on the number of mathematical operations.

Keywords: real-time simulation; field programmable gate array; reconfigurable computing; dynamic systems; hardware description language

Citation: 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. https://doi.org/10.3390/en14248404

Academic Editor: Ahmed Abu-Siada

Received: 23 October 2021
Accepted: 15 November 2021
Published: 13 December 2021

1. Introduction

Scientific computing is characterized mainly by heavy computation tasks requiring efficient number-crunching systems and rapid changes in the input data to be processed [1]. The RTDS is a powerful tool in scientific computing because it provides an efficient system design process, higher accuracy when compared with an offline simulation tool, parallel processing, and it is also being used due to its rapid prototyping [2]. To reproduce a phenomenon faithfully, the simulator needs to solve grid-scale model equations for one time-step within the same time as a real-world clock. Thus, the simulation execution time must be shorter than the selected time-step [2].

The RTDS applied to the domain of electrical and drive systems can be classified as (1) fully digital real-time (RT) simulation and (2) hardware-in-the-loop (HIL) RT simulation [3]. The first one consists of supercomputer technology, which makes it able to simulate transients in large networks. During the 90s, the production of fully digital RT simulators started and some of these tools continue to be featured in simulations and research today. The following can be cited: the RealTime Digital Simulator (RTDs) developed by the Center of Research in the HVDC of Manitoba, the HYPERSIM simulator of the

Electrical Research Institute of the HydroQuebec and eMEGAsim simulator developed by the company OPAL-RT Technologies. Some of these simulators have standard features, such as RTDs and HYPERSIM, which use the nodal electromagnetic transient program (EMTP) solution, while the eMEGAsim uses the state-space representation to build the power network and controls in Matlab/Simulink [4]. However, there are differences in its hardware components.

The RTDs are composed of dedicated hardware platform NovaCor, custom-developed and based on a powerful multicore processor. Inside each NovaCor unit is a custom-built board featuring IBM®'s POWER8™ processor [5]. The HYPERSIM was explicitly developed for HIL testing and used a combination of Intel processors to enable RT computing of extensive models and an FPGA to include ultra-fast loop time for fast-switching frequency real-time simulation. Depending on its model, the simulator can have an FPGA of Kintex7, Virtex7, Spartan3/Virtex6, or up to four units of Virtex6/Kintex7's in the same plataform [6]. The eMEGASIM uses the unique ARTEMiS and SSN solvers developed by OPAL-RT to increase speed, accuracy, and numerical stability for the simulation of critical systems. Processors and a powerful FPGA [7,8] equipped this platform. Regarding the capacity of these simulators, the HYPERSIM and eMEGAsim have network simulation capabilities with a typical time step of 5–100 us and 10–100 us, respectively, whose maximum size of a tested three-phase network is 9000 and 1500 nodes [7–9], respectively. Such processing capacity gives these simulators a high acquisition price since they have competitive technologies in the niche of RT simulators.

The use of FPGA in RTDS is noticeable in many engineering domains, such as electrical power [2,3], drive systems [10,11], and artificial intelligence [12]. FPGAs have seen a massive evolution since their inception almost three decades ago [13]. This device allows the development of dedicated hardware, whose custom logic brings massively parallel computation and faster processing. FPGA technology can embed parallel hardware components or several IPs (Intellectual Property) due to many programmable logic fabrics available on the chip [14]. In the face of power challenges, performance requirements and demands for higher flexibility, hardware designers directed towards reconfigurable computing using FPGAs that offer high computation rates per watt and adaptability to the application constraints [14]. These features make it a device used to overcome the RT processing constraints, mainly in complex modeling designs.

Considering that interactive simulations based on RT are becoming valuable test tools, the development of an affordable RT simulation platform that the user themself can design is essential.

In this context, this paper aims to describe the procedure used to develop a fully digital RT low-cost simulator in an FPGA with HDL. The performance of the proposed simulator is comparable to the renowned high-cost commercial simulators mentioned above, thus making it a more affordable technology. Moreover, HDL provides liberty in the FPGA architecture choice and no dependency on commercial ready-made hardware-software packages. Thus, user's autonomy for making their simulator with this flexibility enables the testing and development of methods and products in an abrangent range of scopes. The proposed method bases itself on the nodal EMTP solution and the methodology used to develop it consists of five steps: 1. design specification; 2. design development; 3. functional simulation; 4. synthesis; 5. floor, planning and place and route, and execution. The DC motor speed control is used as a test case to illustrate the embed and simulation process and evaluate the proposed RT performance. The DC motor drive system modeling considers the drive system consisting of the electric machine, power converter, low-pass filter and Proportional–Integrative–Derivative (PID) controller.

2. DC Motor Drive System Modelling

The test system adopted in this work is the DC motor speed control. Hence, this section gives a brief description of the whole system operation and modeling stages. The DC motor is composed of a field winding in the stator and an armature winding in the rotor [14].

Figure 1 shows the equivalent circuit for the independent excitation DC motor type and Table 1 presents all parameters and variables used in this modeling.

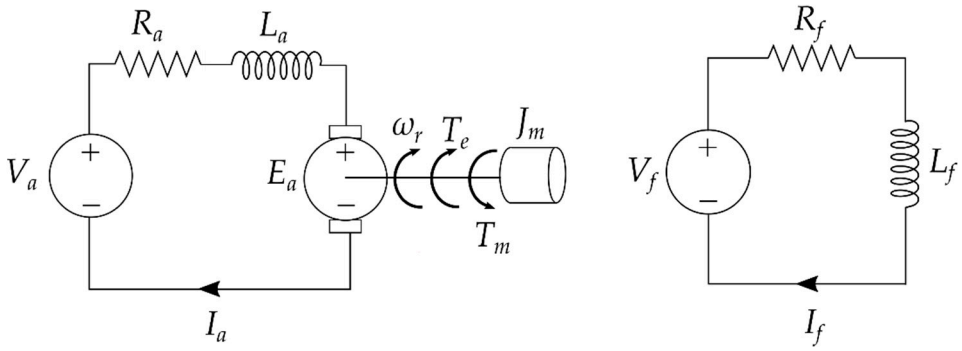

Figure 1. Equivalent circuit of a separately excited DC motor.

Table 1. Units for motor modelling.

Symbol	Parameter
Jm	Inertia moment (Kg·m^2)
ω_r	Angular speed (rad/s)
L_a	Armature inductance (H)
R_a	Armature resistance (Ω)
V_a	Armature voltage (V)
I_a	Armature current (A)
L_f	Field inductance (H)
R_f	Field resistance (Ω)
V_f	Field voltage (V)
If	Field current (A)
T_e	Electrical torque (N·m)
T_m	Mechanical torque (N·m)
E_a	Back EMF (V)
F_m	Friction coefficient (N)
K_m	Torque constant (Nm/A)

According to the Kirchhoff's Voltage Law, the differential equations obtained from Figure 1 are (1)–(2), where E_a is given by (3).

$$V_f(t) = R_f I_f(t) + L_f \frac{dI_f(t)}{dt}. \tag{1}$$

$$V_a(t) = R_a I_a(t) + L_a \frac{dI_a(t)}{dt} + E_a(t). \tag{2}$$

$$E_a(t) = K_m I_f(t) \omega_r(t). \tag{3}$$

This back-emf is induced on the armature circuit due to its kinetic movement in relation to the field winding. The motor movement equation is represented as follows:

$$T_e(t) - T_m(t) = J_m \frac{d\omega_r(t)}{dt} + F_m \omega_r(t). \tag{4}$$

where,

$$T_e(t) = K_m I_f(t) I_a(t). \tag{5}$$

The DC motor speed control requires a transfer function relating the state variable ω_r to the control variable V_a to be imposed on the motor to establish the angular speed

tracking setpoint. Applying Laplace's Transform in Equations (1)–(5) and rewriting them, it is possible to reach the transfer function:

$$G(s) = \frac{\omega_r(s)}{V_a(s)} = \frac{\frac{K_m I_f}{L_a J_m}}{\left(s + \frac{R_a}{L_a}\right)\left(s + \frac{F_m}{J_m}\right) + \frac{(K_m I_f)^2}{L_a J_m}}. \quad (6)$$

2.1. Model Description and Discretization

The DC motor drive system is composed of an electric machine, power converter, low-pass filter and PID controller, whose structure is illustrated in Figure 2. To simulate it in a programmable logic platform is necessary to apply a discretization method in the whole system equations. For this purpose, the most straightforward and classic method is used to solve numerically differential equations, Euler's method.

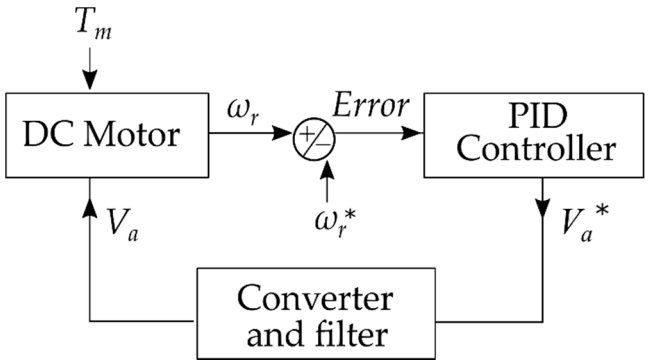

Figure 2. Block diagram of the DC motor drive system simulated.

2.1.1. DC Motor

Applying Euler's method to the DC motor equations, (2)–(4), (7)–(9) are obtained, where h represents the sampling time.

$$I_f(t) = I_{f(t-1)} + \left[-\frac{R_f}{L_f} I_{f(t-1)} + \frac{1}{L_f} V_{f(\mathrm{t}-1)}\right] h. \quad (7)$$

$$I_a(t) = I_{a(t-1)} + \left[-\frac{R_a}{L_a} I_{a(t-1)} - \frac{K_m}{L_a} I_{f(t-1)} \omega_{r(t-1)} + \frac{1}{L_f} V_{a(t-1)}\right] h. \quad (8)$$

$$\omega_r(t) = \omega_{r(\mathrm{t}-1)} + \left[-\frac{F_m}{J_m} \omega_{r(t-1)} + \frac{K_m}{J_m} I_{a(t-1)} I_{f(t-1)} - \frac{1}{J_m} T_m\right] h. \quad (9)$$

2.1.2. PID Controller

The PID controller calculates the reference armature voltage, V_a*, to be sent to the converter. The controller plant is described in (10).

$$c(s) = K_p + \frac{K_i}{s} + K_d s. \quad (10)$$

The PID discrete implementation is obtained through backward finite differences and algebraic manipulations. Thus, (10) is transformed in (11)–(13). These three equations act to regulate the error of different instants of time, been (11) responsible for the current instant, (12) previous instant and (13) for the instant before the PD.

$$PID = K_p + \frac{K_d}{h} + K_i h. \quad (11)$$

$$PD = -K_p - 2\frac{K_d}{h}. \tag{12}$$

$$D = \frac{K_d}{h}. \tag{13}$$

Applying (11)–(13) in (2), V_a is now represented by (14).

$$V_a(t) \ = Error_{\ (t)} PID + Error_{\ (t-1)} PD + Error_{\ (t-2)} D + V_{a(t-1)}. \tag{14}$$

2.1.3. Power Converter

The converter used is of the H-bridge type, which has the schematic diagram shown in Figure 3. The pair of switches S1 and S4 work as the asynchronous buck mode while the pair S2 and S3 work as the boost mode. Thus, this circuit can impose a two-level output voltage. Pulse-width modulation (PWM) signals are used to control the switching to obtain the average value of the output equal to V_a*.

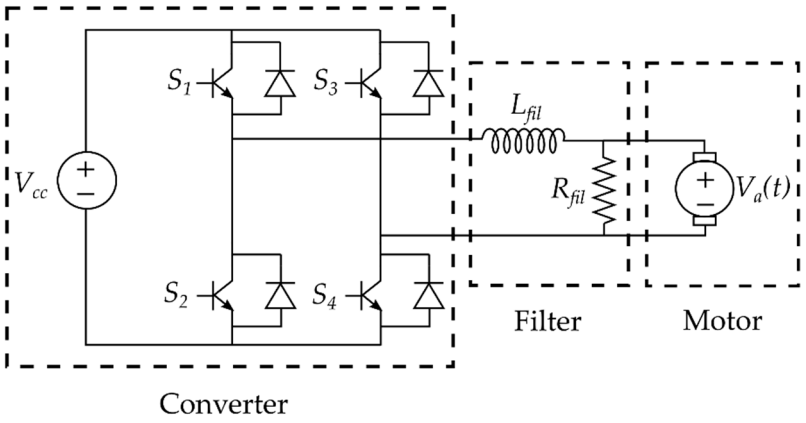

Figure 3. Schematic of the modeled dynamic system.

Considering the duty-cycle ratio T_1/T [10], T_1 being the time interval for pair S1 and S4 turned on and T the PWM period, the average value of armature voltage can be considered as in (15).

$$\overline{V_a} = \frac{V_{cc}T_1 \ - V_{cc}(T - T_1 \ )}{T}. \tag{15}$$

From (15), it is possible to obtain (16), where the instant T_1 calculation can be seen based on V_a*. In this way, it is possible to determine the exact time of switching for each PWM period.

$$T_1(t) \ = \left[\frac{Va^*_{(t)}}{2\ V_{cc}} + \frac{1}{2}\right]. \tag{16}$$

2.1.4. Filter

As shown in Figure 3, after V_a* is converted into a physical value, it is also addressed by a low-pass filter. This filter attenuates the ripple produced by the converter switching, allowing the passage only of low frequencies. Using converter output voltage, V_c, and the filter current, I_c, and applying the Kirchhoff's Voltage Law on the filter circuit, it is possible to find the converter voltage, described in (17):

$$V_c = L_f \frac{dI_c(t)}{dt} + R_f I_c(t). \tag{17}$$

Utilizing Euler's Method in (17) and performing algebraic manipulations with I_c, we can obtain (18):

$$I_{c(t)} = I_{c(t-1)} + \left[-\frac{R_f \, Ic_{(t-1)} + \overline{Va_{(t-1)}}}{L_f} \right] h. \quad (18)$$

where V_c is considered to meet the same value as the average output control signal, V_a. Thus, the necessary electric current that will scroll thought the converter is determined.

3. FPGA Features

In this section all the relevant points in FPGA architecture and numerical patterns are discussed, considering their applications to RT simulator development.

The hardware device employed to embed this digital system design was chosen based on the desired performance, amount of logic cells, hardware efficiency, intellectual property (IP) and memory blocks. The FPGA is composed of 220 K logic elements, 80,330 adaptive logics, 162 variable-precision DSP blocks, 38,418 × 19 multipliers, 312,320 registers, 11,740 M20K and 284 GPIO. This device is optimized for high-bandwidth performance applications and provides 12 units of 12.5 G transceiver-based functions, 1.4 Gbps Low Voltage Differential Signaling (LVDS), and up to a 72 bits wide DDR3 SDRAM interface at up to 1866 Mbps [15].

3.1. FPGA Architecture

The most basic configuration of an FPGA is based around a two-dimensional array of configurable logic blocks (CLBs) and I/O blocks interconnected via a switching matrix of wires. The modern FPGAs comprise many memory blocks and specialized circuits that enhance the efficiency of digital signal processing (e.g., transceivers channels, DSP blocks, etc).

Figure 4 [15] depicts the Intel Cyclone 10 architecture; it has adopted a column I/O structure with 12.5 Gbps transceivers on the left-hand side of the die. The GPIO in vertical columns is in banks of 48 I/Os, each with a high-efficiency memory controller and an I/O phase-locked loop (PLL) [16].

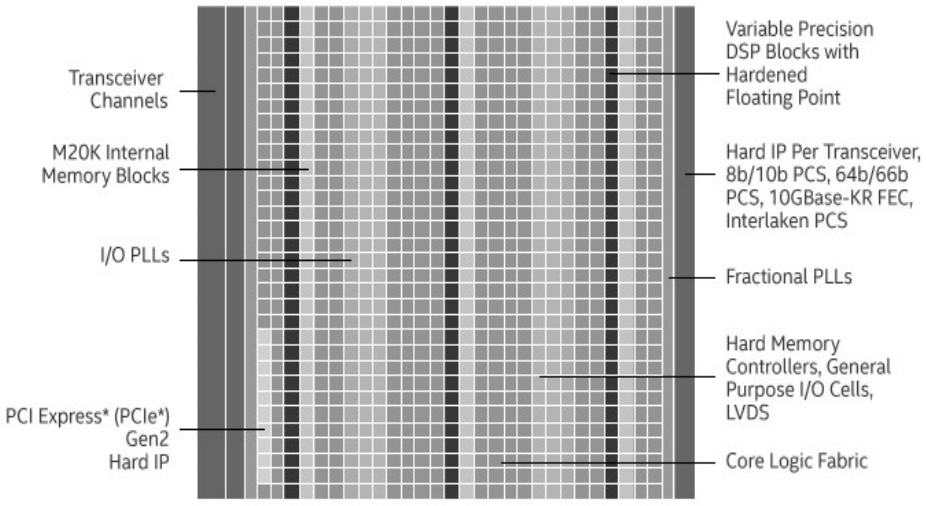

Figure 4. Intel Cyclone 10 GX FPGA architecture © [15].

3.2. DPS—Floating-Point Precision

The Institute of Electrical and Electronics Engineering (IEEE) published the 754 standards for binary floating-point arithmetic in 1985 [6,7]. This standard went through a series of updates until the name of the 32 bits binary baseversion was renamed to "32-bit single-precision" [8]. The IEEE 754 single and double precision formats are shown in

Figure 5a,b, respectively. The floating-point representation allows use of a wide dynamic range automatically. In contrast, the fixed-point representation is limited and requires the user to track the magnitude of numbers and deal with the problems that arise during operations with numbers of different magnitudes [8,16].

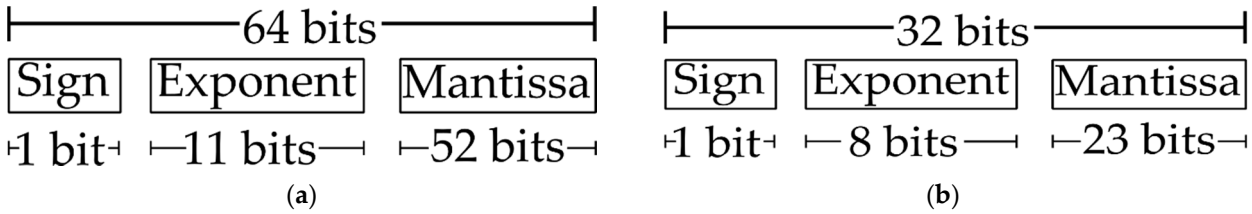

Figure 5. (**a**) IEEE 754 Single-precision 32-bit format. (**b**) IEEE 754 Double-precision 64-bit format.

The both versions of IEEE 754 single and double precision convert the value of the decimal number into one floating through its three parts: signal, exponent and mantissa.

The signal is equal to 1 if the number is negative or 0 otherwise.

The Exponent is the component of a finite floating-point representation that signifies the integer power; the radix is raised in determining the value of that floating-point representation. It is used when the significand is regarded as an integer digit and fraction field, and the exponent q is used when the significand is regarded as an integer [16].

Lastly, the mantissa contains the significant digits except for the leading digit. The board devices are enhanced with hardened floating-point operators in the digital signal processing (DSP) block. In this work, the DSPs slices were used through the Ips' instances to include the Multi-Cycle Custom Instruction for Floating-point embedded system. Figure 6 shows the circuit designed to execute a floating-point operation.

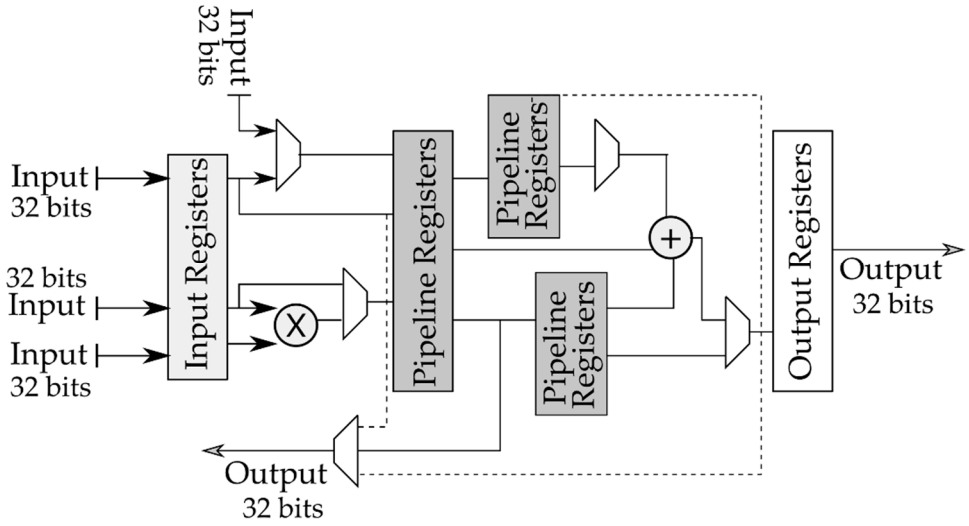

Figure 6. Single-precision floating-point mode (adapted from © [15]).

This work was adopted the IEEE 754 single-precision format. Thus, to realize a floating-point operation the IP's module was instated: Multi-Cycle Custom Instruction for Floating point. It is passed the operands and the code of arithmetic operation to identify the type of arithmetic computation, which could be a sum, multiplication, subtraction or division. Then, these values performed the process illustrated in Figure 6 and as the output module, which had obtained the result of its operation. In this case, the inputs abide by the IEEE 754 standard, and consequently, the output returns in the same standard format.

The Multi-cycle custom instruction supports operations (add, sub, multiply, divide) and adds support for square root, comparisons, negate and other functions [8,17]. Each of these operations has a different processing time, which is illustrated in Table 2. This way,

to perform more than one operation during the same clock cycle, several modules of the multi-cycle can be instated and organized to be processed in parallel. This architecture can be customized using the Dynamic Partial Reconfiguration (DPR) feature, a reconfiguration that can be performed all or for a subset of the IPs [18,19].

Table 2. Cycle numbers to execute an instruction.

Operation	Cycles	Code
Division	16	255
Subtraction	5	254
Add	5	253
Multiply	4	252
Square root	8	251
Int to float	4	250
Float to Int	2	249
Fmins	1	233
Fmaxs	1	232
Fnegs	1	225
Fabss	1	224

4. Hardware Development

The Quartus Prime Pro software 19.4 edition was the programming environment used to develop the codes regarding the real-time simulator implementation. Its compiler transforms the project algorithm into hardware commands through fast and straightforward operations. Thus, it allows reconfigurable computing in fixed hardware that adapts to the algorithm.

The methodology used for the FPGA design flow consisting of 6 stages: design specification, design development, functional simulation, synthesis and the floor, planning and place and route. This setup is shown in Figure 7.

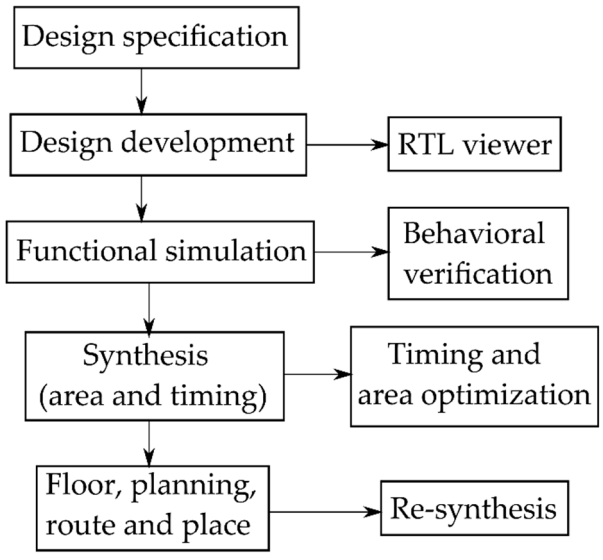

Figure 7. Design flow.

The specification stage is the reference used for development design. It defines functional characteristics, defining its functionalities (development of a DC motor), performance required (real-time execution) and the device used.

During the development, the circuit must have its definition and description conducted by an HDL. In this work, the DC motor's real-time simulator was divided into three modules: the DC motor, controller and the converter with filter. Figure 8 illustrates this. These blocks are connected by a Datapath and instantiated in the top-level design.

The internal structure of each module is composed of its respective operating equations that were previously described in Section 2.

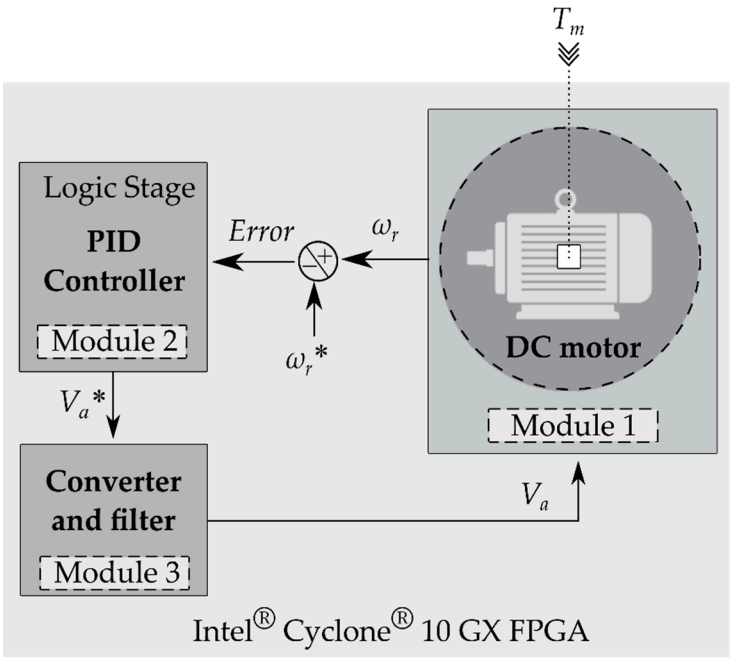

Figure 8. Project Structure in modules.

The module architecture must be planned in order to consider the trade-off between time, power and area. Given the physical area constraint of a chip, architecture designers need to determine parameters, such as the size and the amount of the logic and routing resources, so that the designed architectures support high system integration, high performance, and high resource utilization [6,20]. In this work, the system described is a real-time simulator, so it is fundamental to have several operations in parallel. Consequently, there will be higher power consumption and a larger area occupied by the projected circuit.

This work applied a technique similar to the expression of three solutions to optimize the computation process of Figure 9. As it is a hardware project and the logic used to represent the model describes its physical circuit, it is necessary to use an instance of the Multi-Cycle Custom Instruction DSP block for each arithmetic floating operation. Once this information is processed, the same optimization can be performed with level 1, in which the division and multiplication operations occur in parallel. Thus, all independent operations can be processed simultaneously, as illustrated by levels 2–3. The three modules highlighted in Figure 9 contain this methodology.

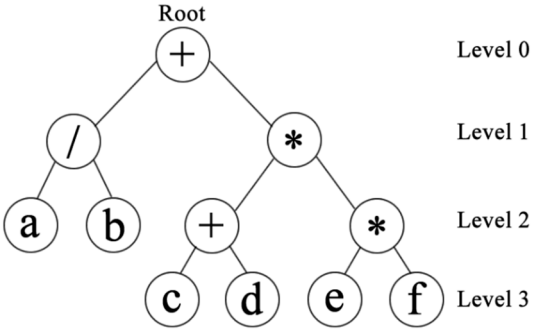

Figure 9. Expression tree schematic.

Eventually, the latency time for processing different arithmetic operations will be distinctive, so it is necessary to delay on the system to ensure that all the results are already ready.

In the development stage, a relevant subtask consists of an analysis of the RTL viewer to ensure that the interconnection between the gates and project modules is correct. Thus, with the right setup, the hardware description must be performed according to the preferences and needs of the application.

In the subsequent stage, the functional simulation should be performed by testbench (TB) verification, which demonstrates that a model has met the requirements of the specification. In this work, two types of testbench were developed, one for the blocks of the microarchitecture stage and the other for the complete system.

The TB provides incentives in the model (blocks or top, whole system) and analysis of its response compared to the ideal results. If there is a large discrepancy between them, the modeling needs to be reviewed (debugging process). These tests are also described in HDL and guarantee the functional correctness of the model.

Then, in stage 4, the timing analysis is performed with the TimeQuest Timing Analyzer to verify circuit performance and detect possible timing violations. The TimeQuest analyzer determines the timing relationships and checks arrival times against the times required to verify timing [21–23]. If the report describes that there is no violation of the requirements imposed by the designer, the project is almost finished.

On the last stage of the design flow, the place and route, floor and planning are performed. Then, the project is embedded on the FPGA device.

Thus, it was initialized, after the design flow conclusion, to obtain the simulation data via Signal Tap. Implementing the simulator in real-time is extensive and its proposed algorithm is described in Figure 10.

Data: case test parameters and variables
Result: output variables graph
1- Create the main project;
2- Instance the Floating-point from IP;
3- Describe the modules in verilog;
4- **While** *is not right the verification* **do**:
    5- Optimize the paralell operations;
    6- Review the modules;
    7- Conncet the modules;
    8- Design under verification;
9- Compile it;
10 - Analyze TimeQuest and check max. freq.;
11- **While** *maximum frequency is not adequated* **do**:
    12- Review the project logic;
    13- Compile it;
14- Programme the FPGA;
15- Get the results in the IEEE 754 standard;
16- Convert it to decimal base;
17- Plot the data and analyze.

Figure 10. Pseudo-algorithm used.

Verifying a block-based design requires planning to ensure visibility of logic inside partitions and communication with the Signal Tap logic analyzer [24,25]. The Signal Tap logic analyzer captures and displays the real-time signal behavior in an FPGA design [26]. This platform connects with the FPGA, which already has the program running, and can access the physical addresses of the entity that have the required information [27].

In this application, the number of samples is required according to the clock as defined in the GUI. Thus, this information traveled through the cable connection from the FPGA device to the computer and was interpreted by Signal Tap. This information was standardized according to the floating-point representation format.

Finally, as the last step in developing of the simulator, a conversion program from the IEEE 754 standard to real decimal numbers was performed using Python as a programming

language. The verification of the results of this converter was carried out according to [28]. Thus, with all the information standardized to real numbers, it was possible to plot the data for results analysis.

Such a procedure for the development of a real-time simulator can be replicated for tests and studies proposed in other works of power system area and mathematical modeling techniques. For example: in [4], an RT simulation of an asymmetrical phase domain synchronous machine on FPGA was proposed; the development of automated HIL tests with RTDS for verifying the protective relay performance is described in [4]; techniques for implementing elliptic curve point multiplication on hardware are presented in [29] and an electric field evaluation using a finite element method and proxy models to design stator slots in a PMSG is reported in [30].

5. Results and Discussion

The algorithm was configured to optimize arithmetic operations in parallel using the same clock cycle. Thus, in (12), for example, the first operation of multiplication was performed in parallel with the subtraction, that is, both operations were processed during the same five clock cycles. Then, three other operations were performed sequentially, multiplication, subtraction and division, so the result was computed with 30 clock cycles.

Similarly, this parallelization was applied to all modules. Thus, it is possible to synchronize the response time of the arithmetic operations of each computation through a delay, which is defined to ensure that all values were finalized at the moment the longest operation is processed. Therefore, when using the 125 MHz clock with the dynamic system with about 45 operations per cycle for this model, the time wasted to process one time-step calculation was approximately 360 ns.

Two test cases were performed to analyze the simulator's performance: 1. a DC motor startup dragging a mechanical load followed by the load withdrawal; 2. Failure in the converter switching during the DC motor speed control operation. The DC motor nominal parameters are: w_r = 125 rad/s, V_a = 240 V, I_a = 10 A, T_m = 19 Nm. The controller parameters are presented in Table 3.

Table 3. PID Controller Parameters.

Symbol	Parameter	Value
K_p	Proportional Gain	−0.2339
K_i	Integral Gain	47.6164
K_d	Derivative Gain	0.0012

5.1. DC Motor Startup with Mechanical Load Followed by Load Withdrawal

The vertical graphics of the left column of Figure 11 describe, from the top to the bottom, the PID control signal, converter and filter output voltages, armature current and angular speed. It is noticeable that the reference voltage imposed by the controller is modified from a logic signal through converter switching into a physical signal that is filtered by the low-pass filter. In this case, the motor starts up moving the nominal mechanical load connected to its shaft and, at 2.3 s, the load withdrawal occurs. It notes angular speed, armature voltage and current starting up at their nominal conditions since the motor is dragging its nominal load. After a transient time interval due to the load withdrawal, the angular speed goes to 126 rad/s and the reference voltage imposed by PID slightly decreases, since the motor changes to the empty operation.

During the transient a PID output voltage peak occurs, raising it to approximately 260 V. This variation, in turn, directly impacts the switching dynamics of the converter, thus a slight increase occurs in the output voltage of the low-pass filter. The armature current decreases to almost 0 A, since the motor torque must be enough to overcome the friction force between the rotor and the airgap. Finally, the speed graphic pictures out the speed boost during the instance of removed load, but the control system stabilized it,

imposing a voltage to establish w_r to track w_r*. The following aspects corresponding to the angular speed transitory behavior were obtained for this scenario:

- Settling time: 0.15 s;
- Overshoot: 3.1%.

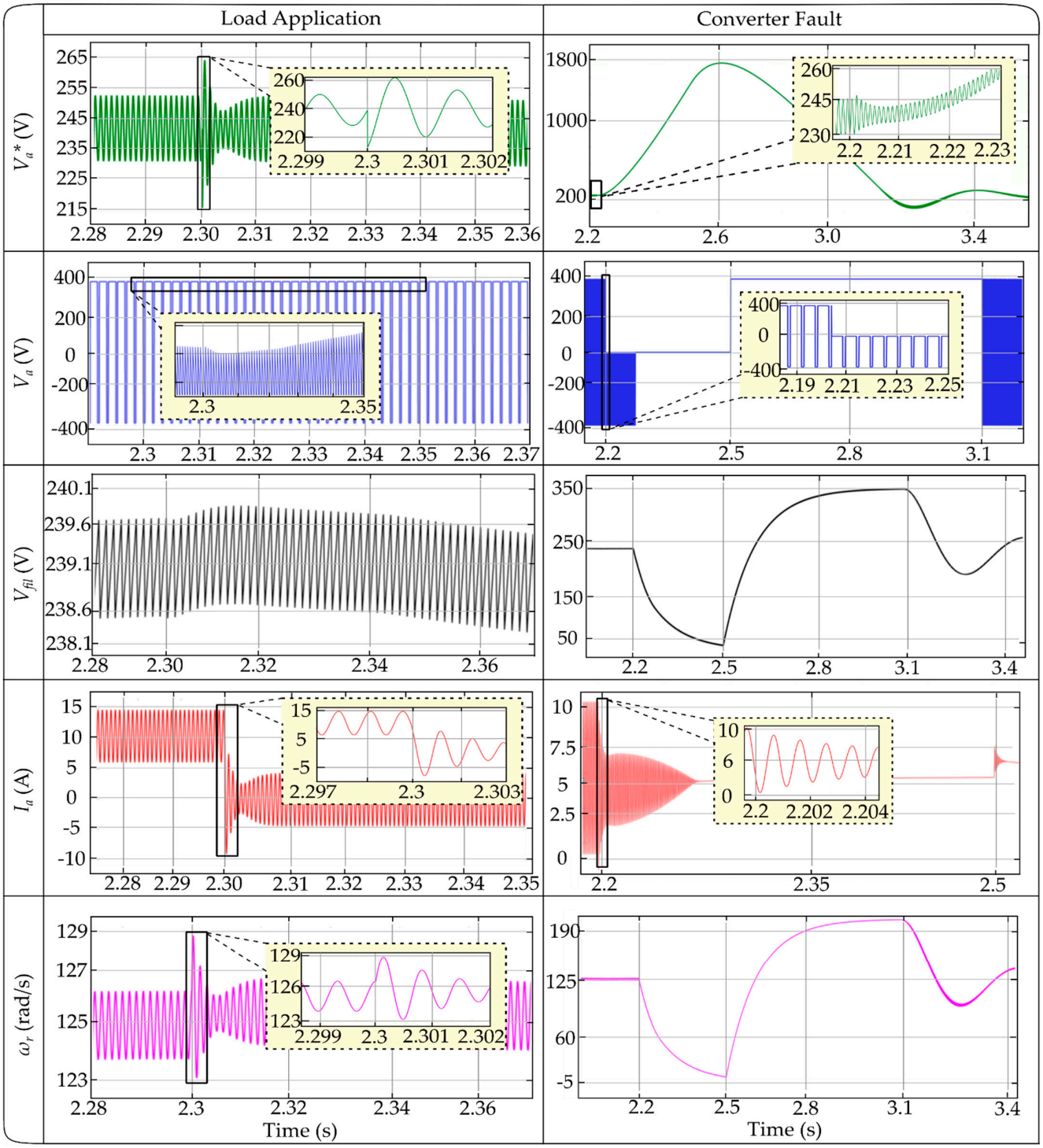

Figure 11. Test cases performed in an RT simulator.

5.2. Failure in the Converter Switching

This scenario consists of a switching failure on the switch pair S2 and S3 of the converter, Figure 3. During the interval from 2.2 to 2.5 s, the switches show a failure, remaining at a constant high logic level. Due to this failure, only switches S1 and S4 are switched during this time interval, while S2 and S3 are kept closed. For this case, the right column of Figure 11 describes PID control signal, converter and filter output voltages, armature current and angular speed, from the top to the bottom. During the failure occurrence, there is a high increase in the reference voltage from the PID since the control system does not obtain the angular speed error decreasing and, hence, increases the control signal to force this error to decrease. This voltage increase leads to the overmodulation of the converter, decreasing the filter output voltage and the armature current amplitude. Thus, the angular speed follows the behavior of the armature current, going through a drop during the fault, and a soft peak when the fault is removed and, finally, the operation is normalized.

5.3. Real-Time Simulation Performance

The algorithms embedded in the FPGA to simulate the DC motor model required less than 1% of the logic cells, 2.8% of the slice registers, and 4.3% of the DSPs. These low-level cells are due to the use of IPs instances for floating-point operations, which had been implemented with pipeline synthesis in specific blocks.

Considering the development methodology and techniques used to model this simulator, the time-step for simulation of other systems can be estimated, such as with that described in [31]. This system is composed of a wind turbine based on a Permanent Magnet Synchronous Generator (PMSG), whose control system imposes a power income maximization algorithm to a wind turbine. The results obtained would be approximately 81 operations in parallel, which would result in a 648 ns time-step for a physical clock of 125 MHz. It is important to emphasize that this is a time step shorter than that commonly propitiated by commercial RT simulators.

Lastly, Table 4 presents a comparison of the performance of FPGAs. Four FPGAs of different brands and prices were selected; their identifiers consist of the FPGA ID, in which ID 1 refers to [32], ID 2 [33], ID3 [34] and ID 4 [35], in which each device's clock and the number of operations required for the DC motor simulation were considered. Thus, an approximation of the time-step was performed. Among the models analyzed, some FPGAs used in HYPERSIM were selected. It is noteworthy that, among the models presented, the one with the best cost–benefit and processing capacity for larger systems is the FPGA that has ID 1, that is the board used in this work.

Table 4. FPGAs comparative performance.

FPGA ID	Logic Cells	DSP Slices	I/O	Memory	Clock	Price	Time-Step
1	220 K	192	284	11 Kb	125 MHz	US$ 1200	360 ns
2	330 K	384 [1]	960	16 Kb	500 MHz	US$ 9761	90 ps
3	326 K	84	500	16 Kb	200 MHz	US$ 1917	225 ns
4	33 K	90	250	4 Kb	450 MHz	US$ 189	1 ns

[1] Each DSP48E slice contains a 25 × 18 multiplier, an adder, and an accumulator.

6. Conclusions

This work proposes a methodology to develop of an electrical system real-time simulator using hardware description language and embedding in a field-programmable gate array. Step by step instructions to embed and simulate a system model in real-time are provided. It introduces the IEEE 754 floating-point standard to realize arithmetic operations and it also addresses the internal structure of field-programmable gate array architecture and its fundamental design flow, in detail. An application example is tested through DC motor speed control simulation and analysis. For the simulated cases, results are com-

patible with the expected results, according to the theory of electric machines and power electronics. Furthermore, the results have shown the performance of the proposed methodology, compatible with the real-time simulation requirements. Estimating the proposed methodology's performance for simulation of a more complex system, a permanent magnet synchronous generator-based wind turbine, corroborates this conclusion. Finally, this work concludes that the proposed methodology propitiates low-cost due to the liberty in the architecture choice and no dependency on commercial ready-made hardware–software packages.

Author Contributions: The authors have made equivalent contributions. All authors have read and agreed to the published version of the manuscript.

Funding: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001, and by the National Council for Scientific and Technological (CNPq).

Acknowledgments: For the opportunity to carry out this work, the authors are grateful to the Electrical and Computer Engineering Graduate Program of the Federal University of Rio Grande do Norte (PPgEEC/UFRN) and to the Digital Systems Laboratory of Informatics Center of the Federal University of Paraíba (LASID/CI/UFPB). This work was supported by the National Council for Scientific and Techonological (CNPq) and by the Coordenação de Aperfeiçoa-mento de Pessoal de Nível Superior—Brasil (CAPES).

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

Abbreviations

CLB	Configurable logic block
DC	Direct Current
DSP	Digital Signal Processing
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HIL	Hardware-in-the-loop
IP	Intellectual Property
PMSG	Permanent Magnetic Synchronous Generator
PID	Proportional–Integrative–Derivative
PWM	Pulse-width Modulation
RT	Real-time
RTDS	Real-Time Digital Simulation
TB	Testbench

References

1. Hsiung, P.-A.; Santambrogio, M.D.; Huang, C.-H. Introduction to Reconfigurable Computing. In *Reconfigurable System Design and Verification*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2009; Chapter 1, Section 1.7.4; p. 26.
2. Faruke, M.O.; Strasser, T.; Lauss, G.; Jalili-Marandi, V.; Forsyth, P.; Dufour, C.; Paolone, M. Real-Time Simulation Technologies for Power Systems Design, Testing, and Analysis. *IEEE Power Energy Technol. Syst. J.* **2015**, *2*, 63–73.
3. Guillaud, X.; Faruque, M.O.; Teninge, A.; Hariri, A.H.; Vanfretti, L.; Paolone, M.; Davoudi, A. Applications of Real-Time Simulation Technologies in Power and Energy Systems. *IEEE Power Energy Technol. Syst. J.* **2015**, *2*, 103–115. [CrossRef]
4. Iracheta-Cortez, R.; Flores-Guzman, N. Developing automated Hardware-In-the-Loop tests with RTDS for verifying the protective relay performance. In Proceedings of the 2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI), San José, Costa Rica, 9–11 November 2016.
5. RTDS Technologies—Simulation Hardware: Scalability and Flexibility without Performance Loss. Available online: https://www.rtds.com/technology/simulation-hardware/ (accessed on 8 October 2021).
6. OPAL-RT Technologies: Simulator Op5707XG. Available online: https://www.rtds.com/technology/simulation-hardware/ (accessed on 8 October 2021).
7. OPAL-RT Technologies: System eMEGASIM. Available online: https://www.opal-rt.com/system-emegasim/ (accessed on 8 October 2021).
8. OPAL-RT Technologies: System Hypersim. Available online: https://www.opal-rt.com/system-hypersim/ (accessed on 8 October 2021).

9. Amin, M.; Rehmani, M.H. *Operation, Construction, and Functionality of Direct Current Machines*; IGI Global: Hershey, PI, USA, 2015; pp. 1–56.
10. Grégoire, L.-A.; Vian, J.; Cense, S.; Belanger, J. Real-time simulation of an asymmetrical phase domain synchronous machine on FPGA. In Proceedings of the IEEE 12th International Conference on Power Electronics and Drive Systems, Honolulu, HI, USA, 12–15 December 2017; pp. 659–664.
11. Rama, D.; Ananthan, T. FPGA Based Emulator Design of a DC Motor. In Proceedings of the International Conference on Communication and Electronics Systems, Coimbatore, India, 17–19 July 2019.
12. Liu, Q.; Gao, M.; Zhang, Q. Knowledge-Based Neural Network Model for FPGA Logical Architecture Development. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *24*, 1. [CrossRef]
13. Farooq, U.; Baig, I.; Alzahrani, B.A. An Efficient Inter-FPGA Routing Exploration Environment for Multi-FPGA Systems. *IEEE Access* **2018**, *6*, 56301–56310. [CrossRef]
14. Inacio, C.; Ombres, D. The DSP decision: Fixed point or floating? *IEEE Spectr.* **1996**, *33*, 72–74. [CrossRef]
15. Intel® Cyclone® 10 GX FPGA Features. Intel® Cyclone®. Available online: https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/10/gx/article.html (accessed on 10 October 2021).
16. Intel®. (2018, August). Intel® Cyclone® 10 GX FPGA Development Kit User Guide. Intel® Cyclone®, User Guide. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-c10-gx-fpga-devl-kit.pdf (accessed on 8 October 2021).
17. IEEE Computer Society. *IEEE Standard for Binary Floating Point Arithmetic*; IEEE Computer Society: Washington, DC, USA, 1985; IEEE Std 754-1985.
18. Kahan, W. IEEE Standard 754 for Binary Floating-Point Arithmetic. *Lect. Notes Status IEEE* **1996**, *754*, 11.
19. Cherian, R.; Thomas, N.; Shyju, Y. Implementation of binary to floating point converter using HDL. In Proceedings of the 2013 International Mut-liConference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), Kerala, India, 22–23 March 2013.
20. Cohen, N.; Weiss, S. Complex Floating Point—A Novel Data Word Representation for DSP Processors. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **2012**, *59*, 2252–2262. [CrossRef]
21. IEEE Standard for Floating-Point Arithmetic. In IEEE Std 754-2019 (Revision of IEEE 754-2008). pp. 1–84, 22 July 2019. Intel ®. (2020, April). Nios Custom Instruction User Guide. Nios II Floating Point Hardware 2 Component. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ugnios2custominstruction.pdf (accessed on 8 October 2021).
22. Ben Atitallah, R.; Viswanathan, V.; Belanger, N.; DeKeyser, J.-L. FPGA-Centric Design Process for Avionic Simulation and Test. *IEEE Trans. Aerosp. Electron. Syst.* **2017**, *54*, 1047–1065. [CrossRef]
23. Leal, M.S.R. Master's Thesis, Federal University of Rio Grande do Norte, Natal, Brazil. 2019. Available online: https://repositorio.ufrn.br/jspui/bitstream/123456789/27982/1/DevelopmentofanFPGA-basedLeal2019.pdf (accessed on 8 October 2020).
24. Betz, V.; Rose, J.; Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAS*; Kluwer: Norwell, MA, USA, 1999. [CrossRef]
25. Intel®. (2012, June). Timing Analysis Overview, Quartus II Handbook. Volume 3: Verification. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qtsqii53030.pdf (accessed on 25 October 2021).
26. Intel®. (2019, August). An 847: Signal Tap Tutorial with Design Block Reuse, for Intel® Arria® 10 FPGA Development Board. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/archives/an847-19-1.pdf (accessed on 25 October 2021).
27. Intel®. (2020, March). Intel® Quartus® Prime Pro Edition User Guide, Debug Tools. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qpp-debug.pdf (accessed on 25 October 2021).
28. IEEE-754 Floating Point Converter. Available online: https://www.hschmidt.net/FloatConverter/IEEE754.html (accessed on 25 October 2021).
29. Lucca, A.V.; Sborz, G.M.; Leithardt, V.; Beko, M.; Zeferino, C.A.; Parreira, W. A Review of Techniques for Implementing Elliptic Curve Point Multiplication on Hardware. *J. Sens. Actuator Networks* **2020**, *10*, 3. [CrossRef]
30. Stefenon, S.F.; Seman, L.O.; Neto, C.S.F.; Nied, A.; Seganfredo, D.M.; Da Luz, F.G.; Sabino, P.H.; González, J.T.; Leithardt, V.R.Q. Electric Field Evaluation Using the Finite Element Method and Proxy Models for the Design of Stator Slots in a Permanent Magnet Synchronous Motor. *Electronics* **2020**, *9*, 1975. [CrossRef]
31. Barros, C.M.V. Control Strategy to Improve Dynamic Behavior of PMSG-based Wind Turbines connected to the Electric Grid. Ph.D. Thesis, EED, Campina Grande, Brazil, 2015. Available online: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/9508 (accessed on 25 October 2021).
32. Cyclone 10 GX Device Datasheet. Available online: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10gx-51002.pdf (accessed on 12 September 2021).
33. Virtex 5 XC5VFX200T. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds100.pdf (accessed on 12 September 2021).
34. Kintex-7 KC705. Available online: https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html (accessed on 12 September 2021).

35. Arty A7 35T: Artix-7 FPGA. Available online: https://digilent.com/shop/arty-a7-artix-7-fpga-development-board/ (accessed on 12 September 2021).