

Article

Hardware-Aware Co-Design of a Lightweight FPGA Accelerator for Palm-Vein Recognition

Xunqi Fan , Yiqun Ma, Bingqing Ma  and Hao Liu * 

School of Integrated Circuits, Southeast University, Nanjing 210096, China; 220246677@seu.edu.cn (X.F.); 220246711@seu.edu.cn (Y.M.); bingqing_ma@seu.edu.cn (B.M.)

* Correspondence: nicky_lh@seu.edu.cn

Abstract

Palm-vein recognition is an attractive biometric modality for secure access control because its subcutaneous vascular patterns are difficult to observe and reproduce externally. However, existing studies optimize the recognition algorithm and the hardware accelerator in isolation, and rarely satisfy the on-chip memory and energy constraints of edge devices. This paper presents a hardware-aware co-design of a lightweight FPGA accelerator for palm-vein recognition, in which the network is shaped by the cost structure of the target fabric and the inference engine is organized around the resulting layer shapes. On the algorithm side, a hardware-aware neural architecture search with deployment cost terms is combined with divisor-aligned structured pruning and INT8 quantization-aware training. Structured pruning reduces the model parameters to 0.32 M and the MACs to 87.2 M while preserving recognition accuracy. On the hardware side, a task-specific design space exploration selects a 14×12 systolic array and an output-stationary dataflow that keeps all feature maps and weights on chip and reduces the modeled buffer-access count by 34.6% relative to the best alternative stationary dataflow. Implemented on a Xilinx Zynq-7100 at 100 MHz, the deployed INT8 checkpoint attains an accuracy of 99.50%, with a PL inference latency of 33.03 ms and an energy efficiency of 33.27 FPS/W.

Keywords: palm-vein recognition; neural architecture search; design space exploration; FPGA accelerator; systolic array; energy-efficient inference

1. Introduction

Reliable personal identity authentication underpins a growing range of security-critical applications, from access control and time attendance to financial transactions and border inspection. As digital services expand, the risks associated with identity fraud and credential theft become increasingly consequential [1]. Conventional authentication mechanisms suffer from well-known limitations. Credentials based on knowledge or physical tokens can be lost, shared, or stolen, whereas fingerprint and facial recognition systems remain vulnerable to surface contamination, presentation attacks, and spoofing. In addition, biometric modalities that require physical contact raise hygiene concerns in shared use environments [2]. These limitations motivate a new generation of identification technologies that are contactless, robust against spoofing, and deployable at the point of use across diverse user populations.

Palm-vein recognition is particularly well aligned with these requirements. Compared with finger vein recognition, the palm offers a larger imaging area and a richer vascular network while tolerating loose hand placement, enabling a more user-friendly contactless



Academic Editor: John A. Kalomiros

Received: 3 July 2026

Revised: 31 July 2026

Accepted: 2 August 2026

Published: 4 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

acquisition process [3]. Residing in the subcutaneous vascular layer, palm-vein patterns are largely immune to surface skin conditions, such as contamination, abrasion, and age-related epidermal changes, that frequently degrade fingerprint and palmprint systems [4]. More importantly, because the vascular pattern is internal and difficult to observe or replicate externally, palm-vein recognition offers potential resistance to presentation attacks and is compatible with liveness-aware acquisition [5]. Acquisition is inherently contactless, avoiding the hygiene concerns of touch-based modalities, and is typically performed under near-infrared illumination around 850 nm in enclosed optical structures that suppress ambient interference and yield high-contrast patterns under variable lighting [2,4]. These properties establish palm-vein recognition as a reliable modality for high-security authentication, restricted-area access control, and privacy-sensitive applications.

However, developing an efficient edge accelerator for palm-vein recognition remains an open challenge, and existing studies reveal a clear three-fold gap. Algorithmic studies on palm-vein recognition [6–8] have produced increasingly accurate and compact networks, though they rarely consider the constraints of the target fabric, such as on-chip memory and inference latency relative to acquisition cadence. FPGA implementations of other biometric modalities, most notably finger vein systems [3,9], demonstrate that on-device CNN inference is both feasible and practically valuable. However, their biometric modality, network structure, accelerator organization, and deployment boundary differ from the present palm-vein workload and cannot be directly inherited by the proposed design. FPGA-based CNN accelerators [10,11] offer mature techniques for dataflow design, tiling, and systolic array mapping, but they are predominantly tuned for general-purpose networks rather than the fixed input regime of a palm-vein task. To our knowledge, no dedicated FPGA CNN inference accelerator that jointly addresses palm-vein model design, task-specific array and dataflow optimization, and measured RTL deployment has been reported. Moreover, because biometric templates constitute sensitive personal data under prevailing privacy regulations such as the GDPR, an on-chip design that retains all data on-device is preferable to cloud-based inference, further reinforcing the need to jointly optimize the recognition algorithm and its hardware.

Motivated by these observations, this paper presents a lightweight FPGA accelerator for palm-vein recognition in which the algorithm and hardware are optimized under a common deployment constraint set. As shown in Figure 1, the target FPGA constraints are propagated across model search, structured compression, array sizing, dataflow selection, and on-chip memory organization, and the resulting design is closed through an RTL implementation. The contribution of this work lies in coupling their constraints for the layer shapes and deployment requirements of palm-vein recognition and, to the best of our knowledge, in providing the first reported dedicated FPGA CNN inference accelerator for palm-vein recognition with a complete hardware-aware algorithm-to-implementation flow.

The main contributions of this work are summarized as follows:

- (1) We develop a compression flow subject to explicit hardware constraints, in which computation and parameter storage budgets are imposed as explicit NAS constraints. The subsequent structured pruning stage further constrains each output channel width to match the 12-column execution granularity of the PE array. This coupling produces an INT8 network with zero channel-lane padding, reducing the model to 0.32 M parameters and 87.2 M MACs while the deployed checkpoint retains 99.50% accuracy.
- (2) We couple the resulting network structure to the accelerator configuration through a constrained PE array sweep and a finite dataflow and tiling exploration. Evaluating the complete layer-shape distribution selects a 14×12 array and a fully on-chip output-stationary mapping that retains the weights and intermediate-feature maps on chip, eliminates inter-layer off-chip traffic, and reduces the modeled buffer-access count by

34.6% relative to the best alternative stationary dataflow under the same analytical accounting boundary.

- (3) We close the co-design loop through a dedicated RTL implementation on a Xilinx Zynq-7100 platform (AMD, Sunnyvale, CA, USA). The inference engine integrates ping-pong on-chip buffers, row reordering, a systolic convolution core, fused INT8 requantization, streaming pooling, and the fully connected tail. The implemented accelerator achieves a measured PL inference latency of 33.03 ms excluding acquisition, preprocessing, and PS-side decision, a power consumption of 0.91 W, and an energy efficiency of 33.27 FPS/W.

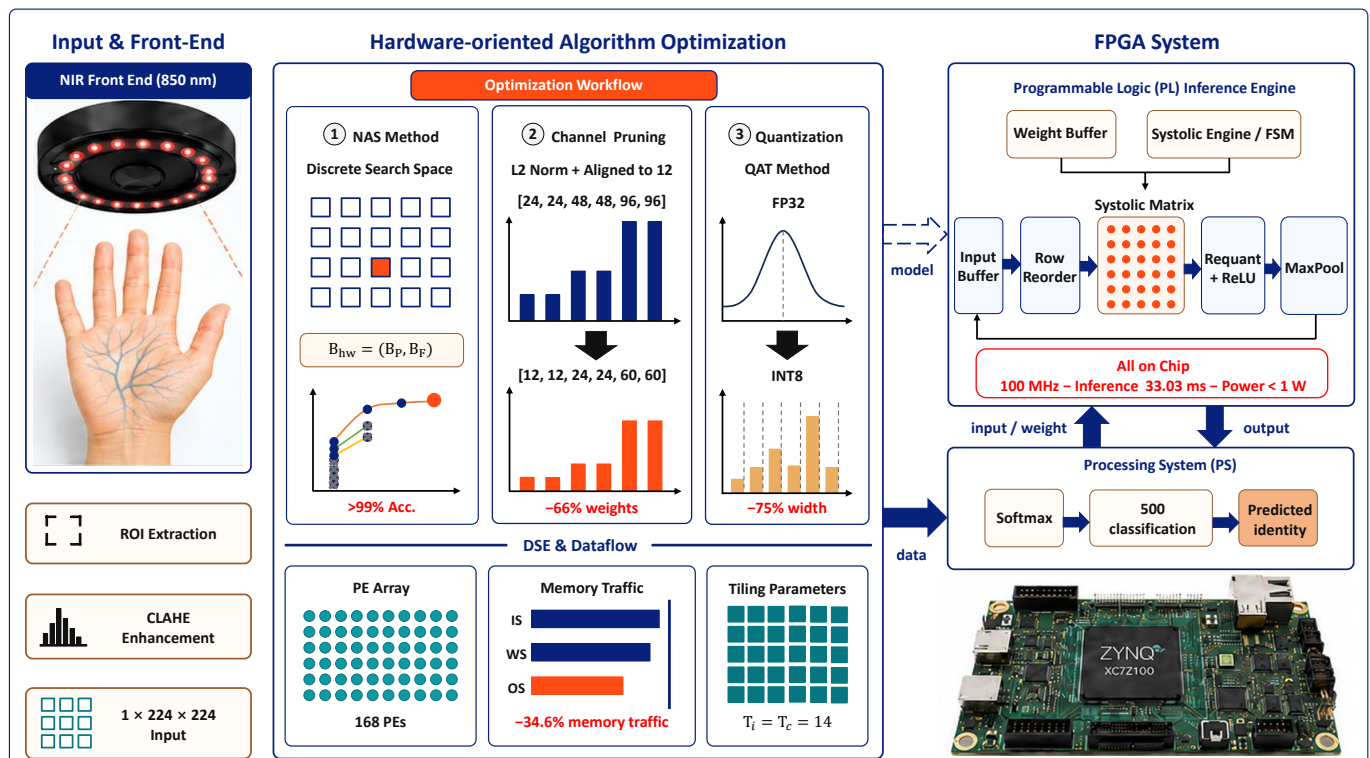


Figure 1. Overview of the proposed hardware-aware co-design framework and FPGA accelerator.

2. Related Works

2.1. Palm-Vein Recognition

Palm-vein recognition has evolved from handcrafted texture descriptors to deep learning-based representations over the past two decades. Early approaches relied on local features such as LBP, SIFT, 2D-Gabor filtering, and local tetra patterns [12–17] to extract discriminative information from near-infrared vascular images. While effective under controlled conditions, these methods are limited in robustness to rotation, translation, and scale variation, and as palm-vein datasets grow in resolution and dimensionality, they increasingly struggle to exploit the richer visual information available [18–20].

Convolutional neural networks (CNNs) have since become the dominant paradigm. Contactless palm-vein authentication has been formulated as a joint search of network architectures and training strategies through Bayesian optimization [21], and the fusion of multiple CNN branches has been shown to improve recognition stability [22]. Transfer learning from large pretrained backbones such as VGG-16 has been applied to vein feature extraction with reported recognition rates above 99% [23], whereas multi-branch architectures further exploit parameter sharing and complementary feature pathways [24]. More recent efforts explicitly focus on lightweight design. A hybrid backbone composed of Shuf-

flNetV2, MobileNetV3, and EfficientNet MBConv blocks reports a 16.78% error reduction and 10.91% fewer parameters relative to a reference baseline [6]. Neural architecture search has also been brought into this domain, with a systematic evaluation of 20 NAS methods across multiple palmprint and palm-vein datasets [7]. The Global-local Vision Mamba framework integrates CNN and Mamba branches through a dedicated GLANAS search strategy [8]. Ma et al. [25] further developed a hardware-friendly palm-vein model through hybrid compression and INT8 quantization for resource-constrained deployment. Collectively, these studies demonstrate that compact and high-accuracy palm-vein networks can be achieved through architecture design, search, and model compression. However, prior evaluations generally emphasize software recognition performance or algorithm model compactness without systematically considering hardware factors such as PE array compatibility, on-chip memory footprint, and dataflow-induced off-chip traffic. The present study extends this line of work by coupling the compressed network structure to the FPGA execution geometry and validating the array configuration, dataflow strategy, and complete inference path through RTL implementation.

2.2. FPGA Implementations of Biometric Recognition Systems

FPGAs have long served as a platform for biometric recognition, valued for their deterministic latency, energy efficiency, and suitability for embedded authentication devices [26]. Prior FPGA biometric implementations range from handcrafted vein-processing pipelines to lightweight CNN-based finger vein inference on Zynq platforms [3,9]. These studies establish the feasibility of on-device biometric inference, but their modality, network structure, and deployment boundary differ from the present palm-vein workload. FPGA-based palm-vein studies have mainly focused on image processing and enhancement rather than a complete CNN recognition path [27]. Beyond biometrics, FPGA has also been widely adopted in other privacy-sensitive sensing pipelines, including medical imaging, providing a cross-domain example of partitioning image-processing workloads between programmable logic and software control under embedded deployment constraints [28]. At the same time, hardware architecture choices are tightly coupled with the features targeted for acceleration. Systolic convolution engines, weight-buffer hierarchies, and channel-wise tiling strategies characteristic of CNN inference are largely absent from these designs. As a result, the literature on biometric FPGA implementations validates the direction, whereas the specific architectural substrate required for a CNN-based palm-vein accelerator cannot be inherited directly from this body of work.

2.3. CNN Accelerators on FPGA

A parallel and increasingly mature line of research has addressed the FPGA acceleration of general-purpose convolutional neural networks. Dataflow strategies have emerged as a central design axis, with input-stationary (IS), weight-stationary (WS), and output-stationary (OS) schemes offering different reuse patterns for activations, weights, and partial sums, and thus different tradeoffs in memory-access cost and processing-element utilization [29,30]. Tiling-based mapping techniques decompose each convolution layer into resident tiles whose sizes are jointly determined by buffer capacity, array dimensions, and pipeline depth [10,11]. Systolic arrays are widely used as computing substrates for FPGA-based convolution accelerators because their regular and deeply pipelined dataflow supports high operating frequencies and efficient mapping of convolution operations. Model-side techniques including structured pruning [31,32], low bit quantization [33], and hybrid compression pipelines that combine multiple strategies [34,35] have been widely explored to reduce the hardware footprint of CNN deployment. However, most of these accelerators target large-scale networks such as VGG, ResNet, or YOLO, whose design

spaces assume large inputs, deep stacks, and wide channels, unlike the single-channel, fixed-resolution, batch-one, shallow palm-vein network. Task-specific DSE is therefore necessary, as quantified by the array-sizing sweep of Section 6.3, where ImageNet-scale array geometries lower raw latency but inflate the PE count and leave structural padding on the palm-vein output-row sizes, whereas the task-specific 14×12 point tiles every layer without waste.

2.4. Summary and Research Gap

These three lines of work define the research context of this study. Palm-vein recognition algorithms achieve high recognition accuracy but usually remain disconnected from physical accelerator constraints. Existing FPGA studies have demonstrated the feasibility of palm-vein processing on reconfigurable hardware, but they do not provide a dedicated CNN inference accelerator jointly optimized with the recognition model. FPGA implementations of other biometric modalities further demonstrate on-device feasibility, although they are commonly organized around handcrafted feature extraction or network structures different from the present workload. General FPGA CNN accelerators provide established systolic, tiling, quantization, and stationary-dataflow techniques, but their configurations are typically optimized for substantially deeper and wider networks.

To the best of our knowledge, no prior work has reported a dedicated FPGA CNN inference accelerator for palm-vein recognition that jointly integrates hardware-constrained model compression, task-specific array and dataflow exploration, on-chip intermediate-feature management, and an RTL-based prototype with measured hardware performance. This study addresses this gap by closing the design path from palm-vein recognition model construction to a physically implemented inference engine with reported resource utilization, latency, and power.

3. Palm-Vein Recognition Algorithm

3.1. Hardware-Aware NAS Objective

A palm-vein accelerator imposes constraints that are different from those of general-purpose CNN deployment. The model must fit within the on-chip memory hierarchy, complete inference within the acquisition cadence of the NIR front end, and expose regular layer shapes that can be efficiently mapped onto a systolic PE array. We therefore formulate NAS as a budget-constrained discrete selection problem.

Let $\alpha \in \Omega$ denote a candidate architecture drawn from the discrete search space Ω , and let $\mathcal{N}(\alpha)$ be the corresponding network. The search problem is defined as

$$\alpha^* = \arg \max_{\alpha \in \Omega} S(\alpha), \quad \text{s.t.} \quad \mathcal{C}_{\text{hw}}(\alpha) \leq \mathcal{B}_{\text{hw}} \quad (1)$$

where $\mathcal{B}_{\text{hw}} = (B_F, B_P)$ denotes the hardware budget vector. Here, B_F constrains the per-inference computation, B_P constrains the parameter count that must be stored on chip. Peak activation is controlled separately through the early downsampling constraint in the search space and is subsequently verified against the input/output-buffer capacity.

Among feasible candidates, architectures are ranked by an accuracy-dominant score with mild complexity penalties:

$$S(\alpha) = \text{Acc}_{\text{val}}(\mathcal{N}(\alpha)) - \lambda_F \Delta_F(\alpha) - \lambda_P \Delta_P(\alpha) \quad (2)$$

where

$$\Delta_F(\alpha) = \frac{\text{FLOPs}(\mathcal{N}(\alpha))}{\text{FLOPs}(\mathcal{N}_0)} - 1, \quad \Delta_P(\alpha) = \frac{\text{Params}(\mathcal{N}(\alpha))}{\text{Params}(\mathcal{N}_0)} - 1 \quad (3)$$

The coefficients λ_F and λ_P are set small so that recognition accuracy remains the primary selection criterion. The penalty terms mainly break ties between candidates of comparable accuracy and prevent unnecessarily large models from being selected.

The hardware-aware objective is optimized through the multi-fidelity successive-halving search summarized in Table 1. Surviving candidates are retrained under progressively larger epoch budgets, concentrating compute on the most promising architectures as the pool narrows from seven feasible candidates to one. The selected architecture reaches a validation accuracy of 99.92%, and its channel configuration [24, 24, 48, 48, 96, 96] becomes the NAS baseline carried into the compression pipeline.

Table 1. Multi-fidelity successive-halving NAS search. Candidates are filtered by a static FLOPs and parameter budget, then progressively retrained under larger epoch budgets and ranked by the score of Equation (2). The epoch column reports cumulative training budgets.

Stage	Epochs	Candidates (in → out)	Selection Rule
Budget filter	0	30 → 7	FLOPs $\leq$ 346.9 M, Params $\leq$ 0.673 M
Stage 1	5	7 → 3	top 40% by score
Stage 2	15	3 → 1	top 30% by score
Stage 3	45	1 → 1	longer retrain
ReRank	75	1 → 1	final retrain

3.2. Search Space and Candidate Selection

The search space Ω is constructed as a lightweight CNN family for single-channel 224×224 NIR palm-vein images. The stage-wise channel vector $\mathbf{c} = [c_1, \dots, c_6]$ is generated by a discrete width multiplier, divisor-aligned rounding, and a minimum-channel constraint. This channel alignment is introduced not only for algorithmic compactness but also to reduce fractional channel tiles in the systolic mapping of Section 4.

The downsampling policy is represented by a binary vector $\mathbf{p} = [p_1, \dots, p_6]$, where $p_i \in \{0, 1\}$ indicates whether a 2×2 pooling layer is inserted after stage i . At least one pooling operation is enforced within the first two stages to avoid an excessive early activation footprint. An optional extra pooling stage is also considered before the fully connected head; in the selected model, this is implemented as a 3×3 stride-2 pooling operation that maps a 7×7 feature map to 3×3 . The search also covers conservative bottleneck usage in later stages, the first fully connected dimension $\text{fc1_dim} \in \{128, 256\}$, dropout $\in \{0.3, 0.5\}$, and a small set of learning rates $\{10^{-3}, 5 \times 10^{-4}, 3 \times 10^{-4}\}$ to reduce proxy-training bias.

For each sampled candidate, FLOPs and parameter count are first estimated using a static profiler. Architectures violating the computational or parameter budgets are rejected before training:

$$\text{FLOPs}(\mathcal{N}(\alpha)) > B_F \quad \text{or} \quad \text{Params}(\mathcal{N}(\alpha)) > B_P \quad (4)$$

The remaining candidates are evaluated using a multi-fidelity successive-halving procedure. Candidates are first trained for a small number of epochs; a subset is retained according to validation accuracy; and the final survivors within that subset are selected using Equation (2). This process is repeated with progressively larger training budgets and fewer candidates. The best architectures are then retrained for a longer schedule to obtain the final ranking.

The selected NAS baseline is a six-stage CNN with channel widths [24, 24, 48, 48, 96, 96], downsampling policy $\mathbf{p} = [1, 0, 1, 1, 1, 1]$, and a two-layer fully connected head with dimensions 256 and 500. This model is used as the starting point for the compression pipeline described below.

3.3. Structured Pruning and INT8 Quantization

The network is further compressed using structured channel pruning followed by INT8 quantization-aware training (QAT). Both steps are chosen for hardware compatibility. Structured pruning preserves dense convolution kernels, while INT8 quantization matches the arithmetic mode of the FPGA convolution engine. For pruning, a final target channel sparsity of 0.3 is progressively approached over five pruning steps. At each step, convolutional filters are ranked locally according to their group-reduced L2 magnitude. Global pruning is disabled, FC1 and FC2 are excluded, and the retained convolutional widths are constrained to executable multiples of the 12-column PE array (Algorithm 1). The round-to-12 constraint folds hardware friendliness directly into the compression stage. By restricting every per-stage channel width to a multiple of the array column dimension, it prevents non-aligned widths that would otherwise leave idle channel lanes on the systolic array. Because the final widths are all exact multiples of 12, the lane-waste expression in Section 4 evaluates to zero rather than to a residual padding fraction.

Algorithm 1 Divisor-aligned iterative structured pruning

- 1: Load the converged NAS model.
 - 2: Compute layer-local filter importance using the L2 norm.
 - 3: Configure final channel sparsity $s = 0.3$, iterative steps $K = 5$, and hardware divisor $d = 12$.
 - 4: Exclude FC1 and FC2 from pruning.
 - 5: **for** $k = 1, \dots, K$ **do**
 - 6: Increase the scheduled pruning ratio toward s .
 - 7: Remove the lowest-importance convolutional dependency groups.
 - 8: Constrain each surviving output width to an executable multiple of d .
 - 9: **end for**
 - 10: Finetune the pruned model for 50 epochs.
-

After pruning, the model is fine-tuned to recover recognition accuracy. The resulting channel widths are [12, 12, 24, 24, 60, 60], which define the final convolutional workload mapped to hardware. Fake-quantization modules and calibration observers are then inserted into the pruned FP32 model to emulate the 8-bit deployment path during training and to determine the required integer quantization parameters. After convergence, the model is converted into an INT8 deployment representation. The quantized weights, bias terms, and requantization records required to map the accumulated convolution results to 8-bit activations are exported in the memory layouts expected by the accelerator. The exported records provide the layer-specific bias and requantization parameters consumed by the accelerator.

3.4. Final Layer Specification for Hardware Mapping

Table 2 summarizes the final pruned and quantized network used by the dataflow design. This table is retained in the algorithm section because it is the direct interface between the recognition model and the hardware mapping: (N, M, H, L, R, C, K, S) define the tiling constraints, while T_I, T_O, T_W, T_M , and T_{PO} are the inputs to the memory-access model in Section 4. Two-layer-shape characteristics guide the subsequent hardware design. The dominant computation, weight storage, and peak activation footprint occur in different layers, with CONV3 dominating MACs, CONV6 dominating convolutional weight storage, and CONV1 producing the largest output activation. In addition, all output-row sizes belong to {14, 28, 56, 112, 224}, which motivates the row parallel PE array sizing discussed in Section 4.2.

Table 2. Per-layer dimensions and data volumes of the pruned, quantized palm-vein network. Output size denotes the convolutional output before any optional pooling. The pooled feature map appears as the input size of the next layer. FC1 produces an internal 256-dimensional hidden representation, while FC2 outputs 500 class logits for closed-set identification.

Layer	CONV1	CONV2	CONV3	CONV4	CONV5	CONV6	ExtraPool	FC
Input size (H, L)	224	112	112	56	28	14	7	$3 \times 3 \rightarrow 1$
Input channels (N)	1	12	12	24	24	60	60	FC1: 540 FC2: 256
Output size (R, C)	224	112	112	56	28	14	3	1
Output channels (M)	12	12	24	24	60	60	60	FC1: 256 FC2: 500
Kernel (K)	3	3	3	3	3	3	3	—
Stride (S)	1	1	1	1	1	1	2	—
T_I (input vol.)	50.2 K	150.5 K	150.5 K	75.3 K	18.8 K	11.8 K	2.9 K	540
T_O (output vol.)	602.1 K	150.5 K	301.1 K	75.3 K	47.0 K	11.8 K	0.540 K	FC1: 256 FC2: 500
T_W (weights)	0.108 K	1.296 K	2.592 K	5.184 K	12.960 K	32.400 K	—	FC1: 138.240 K FC2: 128.000 K
T_M (MACs)	5.419 M	16.257 M	32.514 M	16.257 M	10.161 M	6.350 M	—	FC1: 138.240 K FC2: 128.000 K
T_{PO} (post-pool)	150.5 K	—	75.3 K	18.8 K	11.8 K	2.9 K	0.540 K	—

4. DSE and Dataflow Design

The design variables are the PE array size (x, y), the dataflow strategy $D \in \{IS, WS, OS\}$, and the per-layer tiling parameters. For IS, the spatial input tile is denoted by (T_h, T_l) . For WS and OS, the spatial output tile is denoted by (T_r, T_c) . In all cases, (T_n) and (T_m) denote the input- and output-channel tile sizes, respectively. In the adopted mapping, y unrolls the output-channel dimension, x unrolls the output-row dimension, and the input-channel dimension is accumulated sequentially inside each PE column.

4.1. Cost Model

For the i th convolutional layer, the input-activation volume, convolution-output volume, weight volume, and post-pooling transfer volume are denoted by $T_{I,i}$, $T_{O,i}$, $T_{W,i}$, and $T_{PO,i}$, respectively. These quantities are obtained directly from Table 2. To compare the mapping-dependent data movement of the candidate stationary strategies, we use the following analytical buffer-access metric:

$$MA_i = T_{I,i}\alpha_i^{\text{in}} + T_{O,i}\alpha_i^o + T_{W,i}\alpha_i^w + T_{PO,i}, \quad (5)$$

where α_i^{in} , α_i^o , and α_i^w denote the dataflow-dependent access factors for input activations, output activations, or intermediate partial sums, and weights, respectively. In particular, α_i^o represents output-buffer and partial-sum traffic introduced by the adopted mapping rather than every physical write performed by the complete RTL datapath. Accordingly, $\alpha_i^o = 0$ for OS means that intermediate partial sums remain resident in the PEs and are forwarded to the streaming postprocessing path after completion, without spill and reload through the modeled output/partial-sum storage level. The $T_{PO,i}$ term represents the additional post-pooling transfer included in the analytical model.

The network modeled access count is

$$MA^D = \sum_{i=1}^6 MA_i^D, \quad D \in \{IS, WS, OS\}. \quad (6)$$

This metric is applied consistently to all candidates in the declared finite search space and is used for relative dataflow selection. It should therefore be interpreted as a mapping analytical access count rather than a cycle-accurate enumeration of every physical SRAM access in the implemented accelerator.

For an $x \times y$ systolic array driven at f_{clk} , the fill-and-drain latency of a tile is

$$T_{\text{tile}} = Q + (x - 1) + (y - 1) \quad (7)$$

where Q is the number of inner-product steps required to compute one output element. The corresponding utilization is approximated by

$$U_{PE} = \frac{Q}{Q + x + y - 2} \quad (8)$$

Since Q is much larger than the fill-and-drain term for the convolutional layers considered here, the practical utilization is dominated by how well the output rows and output channels are tiled by (x, y) .

4.2. PE Array Sizing

The first DSE stage selects the shared array size (x, y) , where x is mapped to the output spatial rows, and y is mapped to the output channels. A custom analytical program exhaustively evaluates the finite grid $x \in \{7, 14, 28\}$ and $y \in \{4, 6, 8, 12, 16, 24\}$. The row candidates are derived from the regular convolutional output dimensions, while the column candidates include divisors of the pruned channel widths and nearby alternatives used to expose channel padding and lane utilization. Since the array serves all six convolutional layers, every configuration is evaluated using the complete layer-shape distribution rather than a single dominant layer.

The output-row dimension is highly regular. All $R_i \in \{14, 28, 56, 112, 224\}$ are integer multiples of 14, so $x = 14$ tiles every convolutional layer without row padding. The pruned channel widths M_i are all integer multiples of 12, so $y = 12$ tiles every output-channel dimension exactly

$$1 - \frac{M_i}{\lceil M_i/12 \rceil \cdot 12} = 0, \quad M_i \in \{12, 24, 60\} \quad (9)$$

For each candidate, the analytical program estimates the network-level latency and MAC-weighted PE utilization. The primary selection jointly considers these estimates, the target budget of 168 PEs, and the divisibility of the output-row and output-channel dimensions. Configurations beyond the target PE count remain in the finite sweep only to expose the latency and resource trend. No heuristic optimizer or Timeloop mapper is used in this primary array selection.

To independently validate the array selected by the analytical DSE, we perform a Timeloop and Accelergy ablation study using a 28 nm technology model. The ablation set contains the selected point, one alternative geometry with the same 168 PE count, two configurations with 224 PEs, and two larger arrays that expose the latency and resource trend. All six configurations are evaluated over the six convolutional layers under the same INT8 network shapes and memory hierarchy. For each fixed array, the Timeloop mapper searches the legal schedule and mapping space, while Accelergy evaluates the corresponding arithmetic and memory actions. Timeloop is used to evaluate fixed array configurations. The first metric is the total cycle count

$$T(x, y) = \sum_{\ell=1}^L T_{\ell}(x, y), \quad T_{\ell}(x, y) \approx \frac{\text{MAC}_{\ell}}{x y \eta_{\ell}(x, y)} \quad (10)$$

where $L = 6$, MAC_{ℓ} is the multiply accumulate workload of convolutional layer ℓ , and $\eta_{\ell} \in (0, 1]$ is the layer level array utilization. The factor η_{ℓ} captures both temporal stalls and the spatial idling that arises when R_{ℓ} is not divisible by x or when M_{ℓ} is not divisible

by y , so any geometric mismatch is faithfully reflected in $T(x, y)$. The second metric is the total energy

$$E(x, y) = \sum_{\ell=1}^L \sum_{m \in \mathcal{M}} A_{\ell,m}(x, y) e_m \quad (11)$$

where $\mathcal{M}$ enumerates the storage levels and the arithmetic units, $A_{\ell,m}$ is the access count obtained from the chosen schedule, and e_m is the per-access energy returned by the technology library. Because global-buffer accesses consume about an order of magnitude more energy per word than PE register-file accesses, $E(x, y)$ favors mappings with high local reuse and penalizes global-level spills.

For the ablation comparison, we report the energy delay product normalized against the selected DSE point,

$$\widetilde{\text{EDP}}(x, y) = \frac{E(x, y) T(x, y)}{E(x^*, y^*) T(x^*, y^*)} \quad (12)$$

together with the processing-element count $N_{\text{PE}} = xy$ as a separate area proxy. These metrics are used to examine whether any representative alternative improves the normalized EDP without a disproportionate increase in N_{PE} . Geometric divisibility is considered separately because structural padding affects lane utilization, control logic, and memory bank organization in ways that are only partially represented by Equations (10) and (11).

Figure 2 summarizes the complete finite sweep and identifies $(x^*, y^*) = (14, 12)$ as the selected configuration under the 168 PE target. Timeloop and Accelergy subsequently evaluate this point and five representative alternatives, with the ablation results reported in Section 6.3.

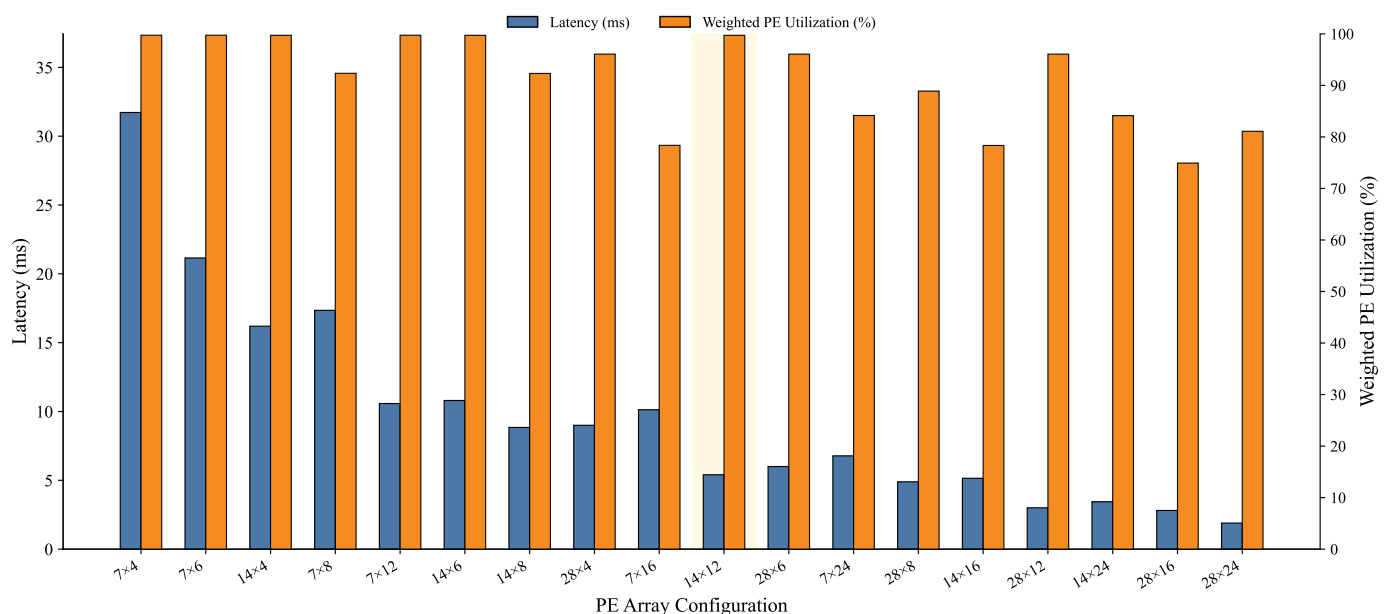


Figure 2. DSE of PE array configurations. The finite sweep compares network-level latency and MAC-weighted PE utilization under different array dimensions. The selected 14×12 point satisfies the 168 PE target and exactly matches the output row and channel dimensions of the pruned network.

4.3. Strategies of Dataflow

After the array size is fixed, the dataflow and tiling parameters determine the modeled buffer-access count. We consider three stationary strategies. For IS mapping, the input tile is kept resident while outputs are repeatedly read, accumulated, and written back across input-channel tiles:

$$\alpha_i^{\text{in}} = 1, \quad \alpha_i^o = 2 \left\lceil \frac{N_i}{T_n} \right\rceil - 1, \quad \alpha_i^{wv} = \left\lceil \frac{H_i}{T_h} \right\rceil \left\lceil \frac{L_i}{T_l} \right\rceil \quad (13)$$

For WS mapping, the weight tile is kept resident. Inputs are reloaded for different output-channel tiles, while output partial sums still require read–modify–write across input-channel tiles:

$$\alpha_i^{\text{in}} = \left\lceil \frac{M_i}{T_m} \right\rceil, \quad \alpha_i^o = 2 \left\lceil \frac{N_i}{T_n} \right\rceil - 1, \quad \alpha_i^w = 1 \quad (14)$$

For OS mapping, the output tile and its partial sums stay resident until the full channel accumulation is completed:

$$\alpha_i^{\text{in}} = \left\lceil \frac{M_i}{T_m} \right\rceil, \quad \alpha_i^o = 0, \quad \alpha_i^w = \left\lceil \frac{R_i}{T_r} \right\rceil \left\lceil \frac{C_i}{T_c} \right\rceil \quad (15)$$

After the array is fixed at 14×12 , a separate custom program exhaustively evaluates $T_h, T_l \in \{7, 14\}$ for IS, $T_r, T_c \in \{7, 14\}$ for WS and OS, $T_n \in \{4, 12, 24, 60\}$, and $T_m = 12$ for IS, WS, and OS. For every tuple, the modeled buffer-access count is accumulated over the six convolutional layers using Equation (6). The tuple with the minimum modeled buffer-access count is retained for each dataflow. This stage does not use a heuristic optimizer or the Timeloop mapper. The resulting comparison is reported in Section 6.3.

5. Hardware Optimization and Implementation

5.1. System Integration and Engine Overview

The prototype evaluated in this work focuses on the FPGA inference subsystem of a palm-vein recognition pipeline. The NIR acquisition and preprocessing blocks are shown only to define the accelerator input, as shown in Figure 3. An 850 nm NIR illumination ring and a monochrome imager in a light-tight housing capture single-channel vein frames, the wavelength chosen because subcutaneous hemoglobin absorbs strongly there and renders the vascular network as dark texture under suppressed ambient light. Each frame is reduced to the network input by a fixed off-engine preprocessing path: an inscribed-circle method crops the palm region of interest, and contrast-limited adaptive histogram equalization enhances the local vein texture, yielding a single-channel 224×224 image at batch size one that matches the input regime assumed by the on-chip memory budget and the systolic tiling factors of Section 3.4. As standard acquisition-side functions rather than a contribution of this work, these steps are described only to define the pipeline in which the inference engine operates.

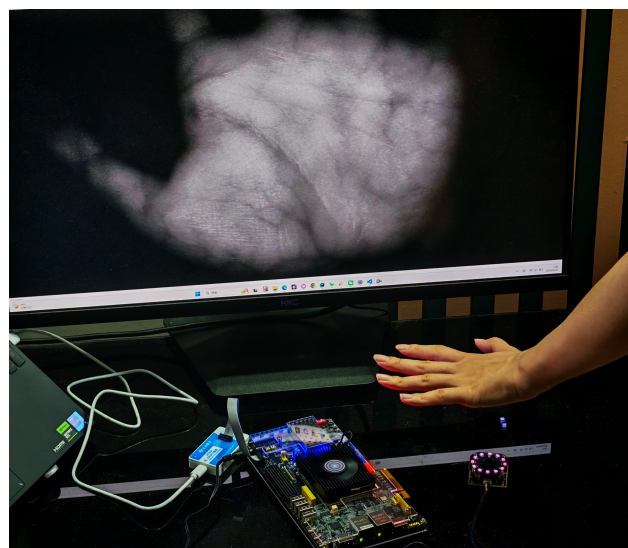


Figure 3. Hardware connection of the palm-vein recognition prototype.

The remainder of this section focuses on the PL inference engine, whose top architecture is shown in Figure 4. The engine integrates a configurator, a banked on-chip memory subsystem, a row reordering window register file, 14 row shift-register chains, a 14×12 systolic array, 12 column-parallel requantization units, a streaming max-pooling unit, and a memory controller FSM, all operating off a single 100 MHz inference clock. Following the data path in Figure 4, a convolutional layer flows through four stages. The memory subsystem supplies the input feature map and, in parallel, the INT8 weights and biases. The row reorder pipeline forms 14 row-aligned 3×3 windows and aligns the kernel taps in time. The 14×12 array then accumulates each output pixel locally under the OS dataflow. Finally, the column-parallel requantization and streaming pooling path writes the result into the inactive ping-pong page as the next layer input, with the memory controller FSM sequencing the phases. The following subsections describe these modules in dataflow order.

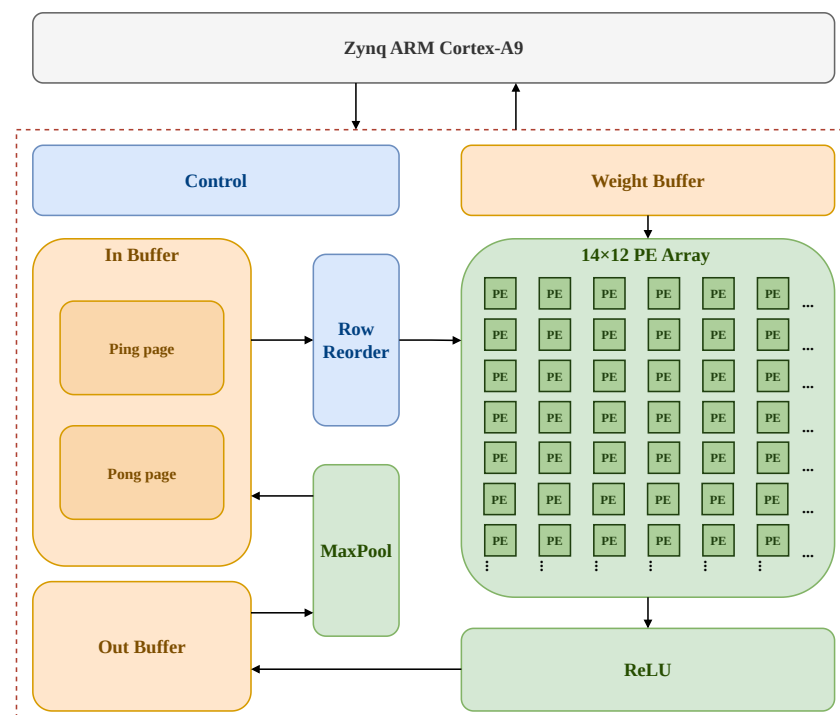


Figure 4. Top architecture of the PL inference engine.

5.2. Configurator and Network Control Plane

Before inference, the PS preloads the per-layer configuration records listed in Table 3, including the corresponding shape, address, flow-control, and requantization fields, and then initiates execution through a single network-start handshake. The PL controller subsequently sequences the convolutional layers, ExtraPool, and the fully connected tail without further PS intervention and selects the appropriate record for each layer. The RTL supports retrained or pruned instances of the documented network topology without resynthesis, provided that the resulting layer dimensions and parameter footprint remain within the declared hardware limits of $H \leq 224$, $N \leq 60$, $M \leq 60$, $K = 3$, and $S = 1$.

A central memory controller sequences the four execution phases summarized in Table 4. The controller alone issues memory enables, addresses, and inter-block strobes, while the buffer modules are kept purely passive. This control-centric partitioning keeps the buffers reusable and lets a layer be re-tiled by changing only the FSM bounds. These four phases execute per layer under the memory controller FSM. Because the design keeps the network parameters and intermediate-feature maps on chip, the PS performs three one-time preload groups before inference begins: the complete per-layer configuration table, including the requantization fields, the preprocessed input image written to the first ping page, and the full weight and bias payload

written to their respective on-chip memories. Once these preload groups have completed and the network-start handshake is asserted, the PL executes the complete convolutional and fully connected inference path. Within each layer, phase (ii) selects the preloaded configuration record and updates the datapath control and parameter-base fields before the layer run. The weights, biases, and requantization information already reside in their on-chip storage, so this phase does not initiate a new PS-to-PL parameter transfer. Across layers, execution is sequential rather than pipelined: the input of layer $L + 1$ is the on-chip output that layer L wrote to the alternate ping-pong page during phase (iv), so the run of $L + 1$ cannot begin until the streaming postprocessing of L has finished writing that page.

Table 3. Per-layer configuration fields preloaded before inference and selected autonomously by the PL network controller. Each field belongs to Shape (S), Flow (F), or Quantization (Q).

Register	Type	Meaning
cur_layer_id	S	Layer index, also selects ping-pong page
cur_ifmap_dim	S	Input feature-map size H_i
cur_cin/cur_cout	S	N_i, M_i (after pruning)
cur_do_pool	F	Insert $2 \times 2/2$ MaxPool after layer
cur_do_bias	F	Apply per-channel bias before requantization
cur_do_relu	F	Apply ReLU before INT8 saturation
cur_weight_base	F	Base offset of this layer's weight segment in the unified weight buffer
cur_bias_base	F	Base offset of the current layer's bias segment in the on-chip bias register file
multi_count_o_in	F	Input-channel accumulation depth for the OS PE array
cur_out_shift	Q	Layer-specific requantization right-shift selected from the preloaded configuration record

Table 4. Per-layer execution phases of the PL engine after the one-time PS preload stage. The four phases execute sequentially through the shared ping-pong feature pages.

#	Phase	Activity
i	Input-page setup	Select the active ping-pong page and initialize the current-layer input traversal
ii	Layer parameter setup	Update the weight/bias base addresses and requantization controls
iii	Layer run	Perform window generation, PE array MACs, requantization, and activation
iv	Streaming postproc.	Pool or bypass outputs, write the pong page, and signal completion

5.3. On-Chip Memory Subsystem

The on-chip memory subsystem is tailored to the OS strategy and comprises a ping-pong input buffer, a banked output buffer, a banked weight buffer, and a small bias register file. The bank multiplicity of each major buffer matches the corresponding PE array dimension. The three BRAM-backed activation and weight buffers provide $2 \times 14 \times 12 \text{ KiB} + 12 \times 8 \text{ KiB} + 12 \times 28 \text{ KiB} = 768 \text{ KiB}$ of logical capacity. The small bias register file is implemented separately in distributed LUT RAM and is not included in this subtotal. Together, these memories keep the weights, biases, and intermediate-feature maps on chip during inference.

Ping-pong input buffer. The 14-bank organization matches the 14-row input dimension of the PE array for parallel feature-map delivery. Two pages alternate: one is read as the current-layer input while the other receives that layer's post-processed output and then serves as the next-layer input, keeping intermediate-feature maps on chip and avoiding off-chip write-back between convolutional layers. Each bank holds the largest per-layer input slice, which occurs at CONV2 and CONV3 with $112 \times 112 \times 12/14 \approx 10.5 \text{ KiB}$. Each bank is provisioned to a depth of 12 KiB, giving about 12% margin over this worst case and fitting the full input of every layer without intra-layer tiling.

Output buffer. The 12 banks correspond to the 12 output-channel lanes of the PE array, letting the requantization units write INT8 activations in parallel without bank conflicts. Under the OS dataflow tiling with $T_r = T_c = 14$, one output tile per bank is $14 \times 14 \times (60/12) \approx 1$ KiB, so the 8 KiB bank depth accommodates the largest per-layer output tile with wide margin.

Weight buffer. The buffer delivers one INT8 weight per column lane per cycle, with all weights preloaded once at startup so no weight traffic reaches external memory during inference. The six convolutional layers contain 54,540 INT8 weights, corresponding to 54,540 bytes or 53.26 KiB. FC1 and FC2 contain 138,240 and 128,000 bytes, corresponding to 135 KiB and 125 KiB, respectively. The unified weight footprint is therefore $53.26 + 135 + 125 = 313.26$ KiB. This maps to a 336 KiB buffer organized as 12 banks $\times$ 28 KiB, leaving 22.74 KiB, or approximately 6.8% of the provisioned capacity, unused. A six-entry lookup table drives `cur_weight_base` for the convolutional layers, and the fully connected tail indexes the same banks through its own base offsets, so layer switching reduces to a single-cycle base-register update.

Bias register file. Holding biases of all six layers, it is realized as distributed LUT RAM rather than BRAM, so it consumes no BRAM budget. The `cur_bias_base` register selects the active segment, mirroring the weight-buffer addressing, and forwards the biases to the requantization stage.

5.4. Row Reorder Pipeline

A row reorder pipeline is inserted between the input buffer and PE array to adapt the buffer output layout to the spatial access pattern of the 3×3 convolution. Since each output pixel requires a local 3×3 neighborhood, the column-wise data supplied by the 14-bank input buffer must be reorganized into row-wise convolution windows. A 16×3 window register file is used, where the additional two rows accommodate boundary padding for the 14 active PE rows. A lightweight scan controller first fills the register file with three consecutive column slices and then enables row scanning. During the scan phase, 14 local convolution windows are generated in parallel and forwarded to the PE input path.

The reordered windows are further passed through 14 depth-9 shift-register chains, which temporally align the nine kernel taps before they enter the systolic array. This staggered feeding scheme converts the spatial window data into a continuous input stream compatible with the array schedule. Consequently, the fill-and-drain overhead in Equation (7) is amortized over each column scan rather than being repeated for every output pixel. The resulting row reorder pipeline is shown in Figure 5. This row reorder pipeline is the hardware embodiment of $T_r = 14$ chosen in Section 4 and is what allows the array to remain close to 100% utilization for the spatially large early layers, where row reuse is the dominant optimization opportunity.

5.5. PE Array

The convolution core is a homogeneous 14×12 systolic array, whose datapath is depicted in Figure 6. Each PE maintains a 32-bit accumulator and consumes one INT8 feature sample and one INT8 weight per cycle. The array exhibits three orthogonal dataflows: feature samples propagate horizontally rightward along each row in a register-pipelined fashion, weights propagate vertically downward along each column with identical per-stage latency, and each PE retains its own output pixel locally until the full kernel-window and input-channel accumulation depth has been exhausted. Once a PE completes its output, the value is injected onto a column-wise upward result relay bus that is time multiplexed among the 14 PEs in the same column, with each PE granting priority to its own completed result and otherwise forwarding the value arriving from below. With 168 DSP48E1 slices

mapped one-per-PE, confirmed by post-synthesis utilization, the array sustains 168 INT8 MACs per cycle at 100 MHz, yielding a throughput of 16.8 GMACS.

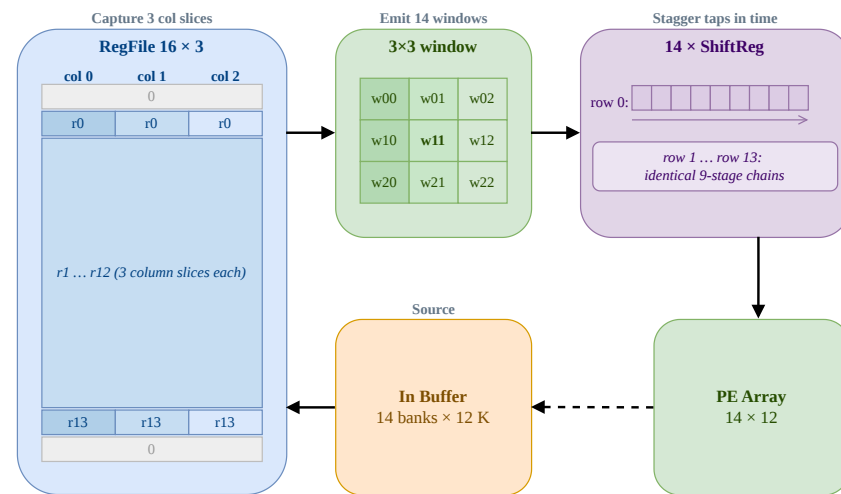


Figure 5. Row reorder pipeline between the input buffer and the PE array. The pipeline consists of column slice buffering, window register update, row-wise window emission, and shift-register-based tap alignment, producing the synchronized row-aligned input feature stream for the systolic array.

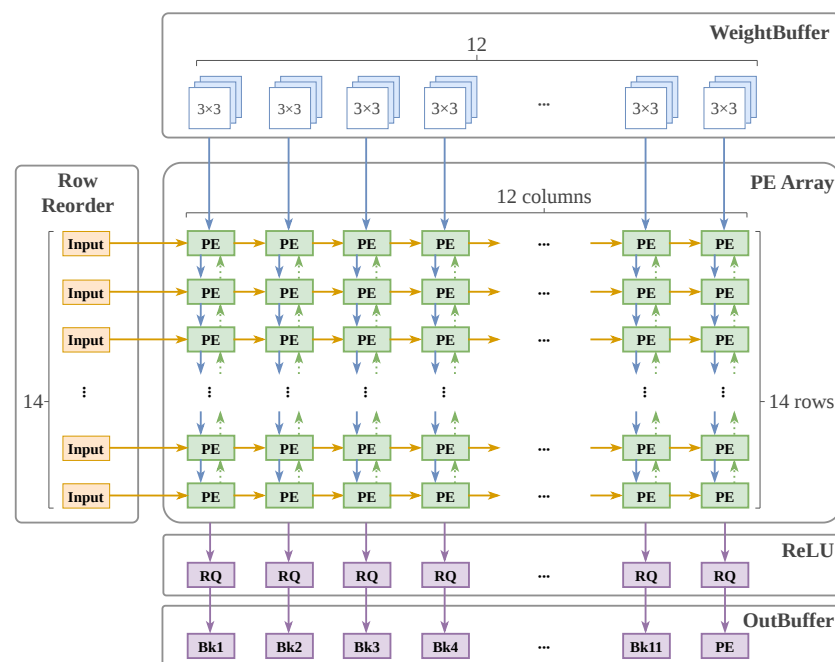


Figure 6. Hardware datapath of the 14×12 PE array.

The 12-column relay buses are exposed at the top of the PE array as 12 parallel 32-bit output lanes, each connected to a dedicated RequantReLU unit. Because all six layers' weights and biases reside in the unified weight buffer and the bias register file, layer switching does not reload parameters. At the start of each layer, the FSM latches *cur_weight_base* and *cur_bias_base* from two six-entry constant lookup tables, and the lane counters then index into the buffers through these base offsets, adding only a small lookup table and a single base addition to the address path without affecting the per-cycle critical path. For the *m*th output channel, the fused postprocessing is expressed as

$$y_m = \text{clip}_{\text{INT8}} \left[\text{ReLU} \left(\text{round} \left(\frac{\text{psum}_m + b_m}{2^{s_m}} \right) \right) \right] \quad (16)$$

where psum_m is the accumulated partial sum, b_m is the channel-wise bias, and s_m denotes the layer-specific requantization shift derived from quantization-aware training. In hardware, the division by 2^{s_m} is implemented as an arithmetic right shift with half-up rounding. The bias addition and ReLU operation can be enabled or bypassed by layer configuration signals.

After requantization, the 12 INT8 output lanes are written to the twelve-bank output buffer through a column-aligned parallel interface. Each lane has an independent address counter and maps directly to one output-buffer bank, matching the column dimension of the PE array. This one-to-one mapping supports conflict-free output writes and decouples the postprocessing stage from the staggered completion of pixels across the relay chains.

5.6. Streaming Pooling and Layer Stitching

After each convolution stage, the memory controller forwards the output stream either to a pooling path or to a direct-copy path according to the layer configuration. In both cases, the output buffer is drained in column major order, and the processed stream is written to the inactive page of the ping-pong input buffer, where it becomes the input feature map of the next layer. For downsampling layers, a streaming 2×2 pooling unit with stride 2 reduces adjacent pixels within each column, buffers the intermediate results, and combines them with the next column to produce pooled outputs. Since only column-wise intermediates are buffered, the line-buffer depth is bounded by $\lceil H_{\max}/2 \rceil = 112$ entries, avoiding a full-frame buffer. Layers without pooling use the direct-copy path, enabling convolution and next-layer input preparation to remain on chip. The final ExtraPool operation and the two fully connected layers are executed on the PL by a compact tail engine that shares the convolutional input and weight buffers, so the tail engine executes FC1 and FC2 on the PL and produces 500 class logits. The PS applies Softmax to the 500 FC2 logits and selects the maximum-probability class to obtain the fixed closed-set identity prediction.

6. Experiment

6.1. Experimental Setup

All experiments used the publicly available PolyU dataset containing 500 identity classes, partitioned at a ratio of 8:2:2 into training, validation, and test subsets. Recognition performance was evaluated using Top-1 accuracy. The same dataset partition and preprocessing pipeline were used throughout all compression stages to ensure direct comparability. Each sample was resized to a single-channel 224×224 image, normalized to $[-1, 1]$, and processed with a batch size of 32.

Structured pruning was performed using layer-wise L2 filter importance with a target channel sparsity of 0.3 over five iterative steps. The convolutional channel widths were constrained by the round-to-12 rule, while the fully connected layers were excluded from pruning. The pruned model was fine-tuned for 50 epochs using SGD with a momentum of 0.9, a weight decay of 5×10^{-4} , and a learning rate of 1×10^{-3} . QAT was subsequently initialized from the converged pruned checkpoint and conducted for 10 epochs using the same optimizer settings with an initial learning rate of 2×10^{-2} . This higher rate was adopted for the short QAT stage to facilitate adaptation to the clipping and rounding perturbations introduced by fake quantization. The resulting model was converted into the INT8 deployment representation, and the inference engine was synthesized and implemented on a Xilinx Zynq-7100 using Vivado v2018.3.

To characterize run-to-run variability after architecture selection, the complete post-selection procedure, including model training, structured pruning, post-pruning fine-tuning, and QAT, was repeated five times using different random seeds. The dataset partition, selected NAS architecture, pruning configuration, and training hyperparameters

were held fixed across all runs. Let a_r denote the final INT8 Top-1 accuracy obtained in run r . The mean and sample standard deviation were calculated as

$$\bar{a} = \frac{1}{5} \sum_{r=1}^5 a_r, \quad s_a = \sqrt{\frac{1}{4} \sum_{r=1}^5 (a_r - \bar{a})^2} \quad (17)$$

where $\bar{a}$ and s_a denote the mean accuracy and its sample standard deviation, respectively. The five final INT8 models achieved Top-1 accuracies ranging from 98.80% to 99.50%, corresponding to $99.00 \pm 0.29\%$, where the variation is expressed in percentage points. The checkpoint originally used for FPGA deployment achieved 99.50% accuracy and was retained as the concrete parameter instance for the subsequent hardware evaluation.

6.2. Algorithm Compression

Table 5 summarizes the compression trajectory of the model instance deployed on the FPGA. For this checkpoint, the Top-1 accuracy changes from 99.92% for the NAS baseline to 99.79% after structured pruning and to 99.50% after QAT. The QAT stage therefore introduces a loss of 0.29 percentage points relative to the pruned FP32 model, while the total loss relative to the NAS baseline is 0.42 percentage points. QAT is initialized from the converged pruned FP32 checkpoint, and fake quantization simulates the clipping and rounding behavior of INT8 weights and activations during training, allowing the model parameters to adapt to these perturbations before final conversion. In the implemented inference path, the PL produces 500 FC2 logits and the PS performs Softmax-based fixed 500-class closed-set identification; therefore, the quantization effect is evaluated using the resulting Top-1 accuracy. The repeated-run analysis in Section 6.1 characterizes the stochastic variability of the post-selection compression pipeline, whereas Table 5 reports the specific checkpoint used for physical implementation. All repeated runs share the same network topology, INT8 operator graph, and layer dimensions. Therefore, the checkpoint parameters affect recognition accuracy but do not alter the accelerator architecture or the reported resource utilization, latency, and power consumption.

Table 5. Effect of each compression stage for the model instance selected for FPGA deployment. Params denotes the total parameter count of the whole model, including the convolutional weights and the fully connected head.

Model	Bit-Prec.	Params (M)	MACs (M)	Top-1 Acc. (%)
NAS baseline	FP32	0.51	320.1	99.92
After structured pruning	FP32	0.32	87.2	99.79
After INT8 quantization	INT8	0.32	87.2	99.50
Drop (vs. baseline)	-	0.19	232.8	0.42

Figure 7 traces the accuracy of the surviving candidates across the successive-halving stages. The stages correspond to cumulative training budgets of 5, 15, 45, and 75 epochs. The selected architecture separates from the field at Stage 2 and converges to 99.92% after full retraining, whereas the two runners-up reach 82.4% and 69.8%, respectively. The gap between the selected model and its no-extra-pool counterpart with identical channel widths isolates the contribution of the extra pooling stage, which compresses the final feature map and reduces the input dimensionality of the fully connected head. The four candidates eliminated after Stage 1 achieve validation accuracies of 9.6%, 8.4%, 8.3%, and 2.1%, respectively.

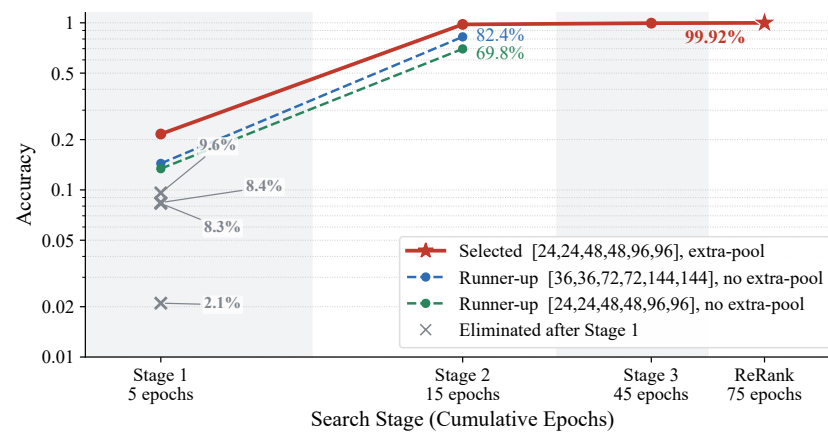


Figure 7. Accuracy of the NAS candidates across the multi-fidelity successive-halving stages.

Ablation study. As shown in Table 6, the four configurations differ by no more than 0.13 percentage points in the reported pruning run, whereas their channel regularity and array utilization differ substantially. The selected configuration is therefore guided primarily by hardware mapping efficiency rather than by the small observed accuracy differences. The unaligned and power-of-two settings retain more weights yet map their 16-channel stages to two thirds of the lanes, spending those extra parameters on slots the fixed array can never fill, whereas the divisor-aligned model carries the fewest weights yet is the only one whose every stage is an exact multiple of the column dimension, driving the lane-waste of Equation (9) to zero. This result indicates that the dominant factor is likely to be divisibility by the array width rather than power-of-two alignment, since the latter retains the same 66.7% utilization as the unaligned configuration. Enforcing divisibility by the array width eliminates structural padding, limits the accuracy change to 0.06 percentage points, and enables full output-channel lane occupancy in the implemented array.

Table 6. Unconstrained pruning versus divisor-aligned pruning under the same target layer-wise channel-sparsity setting.

Pruning Setting	Channel Widths	Convolutional Weights (K)	Top-1 Acc. (%)
NAS baseline	[24, 24, 48, 48, 96, 96]	160.9	99.92
Unaligned ratio	[16, 16, 33, 33, 67, 67]	77.3	99.85
Aligned to 8	[16, 16, 32, 32, 64, 64]	71.6	99.85
Aligned to 12	[12, 12, 24, 24, 60, 60]	54.5	99.79

6.3. DSE and Dataflow Results

The design space exploration of Section 4 fixes two decisions in sequence: the systolic array size (x, y) and the dataflow strategy. We report them here as the two experimental outcomes that justify the adopted 14×12 output-stationary configuration. Both sweeps use the same pruned and quantized network and the same layer shapes, so the differences reflect the mapping choice rather than the workload.

PE array sizing. Figure 2 reports the exhaustive analytical sweep over the declared PE array grid. The 14×12 point is selected under the 168 PE target by jointly considering network latency, MAC-weighted utilization, and exact compatibility with the output row and channel dimensions.

Ablation study. Table 7 groups the candidates by N_{PE} and evaluates the sensitivity of the selected design point to array geometry. Within the 168 PE budget, 14×12 and 28×6 exhibit nearly identical energy, while 14×12 reduces the cycle count by 3.1% and preserves regular spatial tiling across all output-row dimensions. In contrast, 28×6 uses the same number of PEs but leaves half of the rows idle on the 14×14 feature maps. Increasing the

array to 224 PEs gives mixed benefits: 28×8 reduces latency by 16.8% with a 9.5% energy increase, whereas 14×16 provides only a small latency reduction but increases energy by 28.2%, resulting in a higher normalized EDP than the selected baseline. The larger 28×12 and 28×16 arrays further reduce latency, but they require $2.0\times$ and $2.67\times$ more processing elements, respectively. Therefore, although the larger arrays are attractive when latency is the only objective, $(x^*, y^*) = (14, 12)$ remains the most balanced resource-efficient point under the 168 PE budget. Combined with the zero channel-lane waste established in Equation (9), these results independently support the 14×12 tradeoff selected by the analytical DSE between latency, energy, processing-element count, and geometric regularity. The decisive factor is geometric rather than nominal, as the configurations that fail to tile every R_i pay a structural padding cost that the raw PE count alone does not reveal.

Table 7. Timeloop and Accelergy ablation of the selected 14×12 PE array against five representative alternatives. Cycle count, energy, and EDP are normalized to the selected point. The last column indicates exact tiling of all output-row dimensions.

Array	N_{PE}	T (Cycles)	E (μJ)	$\tilde{T}$	$\tilde{E}$	$\widetilde{EDP}$	Tiles All R_i
28×6	168	536,276	36.3668	1.032	1.000	1.032	✗
14×12	168	519,564	36.3682	1.000	1.000	1.000	✓
28×8	224	432,264	39.8377	0.832	1.095	0.911	✗
14×16	224	500,056	46.6165	0.962	1.282	1.234	✓
28×12	336	260,760	36.3682	0.502	1.000	0.502	✗
28×16	448	252,548	46.6165	0.486	1.282	0.623	✗

The near-equality between 28×6 and 14×12 is genuine, with only a $0.00135 \mu J$ difference in unrounded energy that was hidden by the previous two-decimal reporting. Within the action-count model, the exact equality between 14×12 and 28×12 , and between 14×16 and 28×16 , arises because configurations sharing the same column dimension and channel-padding rule execute the same total number of modeled arithmetic and memory actions. Increasing the row dimension changes cycle parallelism but not the total modeled action count.

Dataflow strategy. With the array fixed at 14×12 , Table 8 reports the layer-wise analytical buffer-access count of the three stationary strategies under the minimum-access tiling parameters found within the declared finite search set. Under the common accounting boundary of Section 4.1, OS achieves the lowest modeled access count of 1508.3 K, corresponding to reductions of 42.6% relative to IS and 34.6% relative to WS. The difference is most visible in the high-resolution early stages and in CONV3, where spatial tiling and repeated output or partial-sum movement make a large contribution to the analytical cost. The distribution also shows that the mapping cost is not determined by weight access alone, since activation and output-related terms dominate the spatially large layers. This result supports the implemented OS schedule, in which intermediate partial sums remain resident in the PEs until the input-channel accumulation is completed, thereby avoiding the spill and reload actions represented by the modeled output/partial-sum term. The values in Table 8 are used for relative mapping selection and do not constitute a cycle-accurate count of every physical on-chip memory transaction.

Table 8. Layer-wise analytical buffer-access count under the three dataflow strategies at the minimum-access tiling parameters found within the declared finite search set. The percentage columns report the contribution of each layer to the corresponding modeled network total.

Selected Tiling	IS		WS		OS	
	$T_h = 14, T_l = 14, T_n = 60$		$T_r = 7, T_c = 7, T_n = 60, T_m = 12$		$T_r = 14, T_c = 14, T_n = 4, T_m = 12$	
Layer	MA (K)	(%)	MA (K)	(%)	MA (K)	(%)
CONV1	834.0	31.76	802.9	34.80	228.3	15.14
CONV2	406.0	15.46	302.3	13.10	233.4	15.48
CONV3	736.9	28.06	680.0	29.48	542.2	35.95
CONV4	299.0	11.39	249.9	10.83	252.3	16.73
CONV5	194.2	7.40	165.8	7.19	157.6	10.45
CONV6	156.1	5.95	106.1	4.60	94.3	6.25
Total	2626.2		2307.0		1508.3	

6.4. FPGA Implementation

Table 9 first reports the realized on-chip storage of the proposed accelerator after pruning, INT8 quantization, and the OS dataflow tiling. Although the naive sum of the largest per-layer input and output activations reaches about 1 MiB, OS dataflow tiling and streaming pooling ensure that no full layer output ever resides on chip simultaneously. The realized major BRAM-backed buffer capacity is 768 KiB, well within the BRAM budget of the target Zynq-7100. The unified weight buffer holds all convolutional and fully connected weights at once, while the separate bias storage and intermediate-feature buffers remain on chip during inference.

Table 9. Logical capacities of the major BRAM-backed buffers and the separate distributed bias storage at INT8 precision. The unified weight buffer holds all convolutional and fully connected weights simultaneously, and the input ping-pong buffer is sized to fit the largest per-layer input without intra-layer tiling.

Buffer	Configuration	Storage
Input ping-pong buffer	2 pages $\times$ 14 banks $\times$ 12 KiB	336 KiB
Output buffer	12 banks $\times$ 8 KiB	96 KiB
Unified weight buffer	12 banks $\times$ 28 KiB	336 KiB
Bias register file	LUT RAM, 264 B	—
Major BRAM-backed buffer capacity	—	768 KiB

Table 10 reports the post-implementation resource utilization, operating frequency, and on-chip power. The PE array fixes 168 DSP48E1 slices mapped one per PE, which sustain the 16.8 GMACS peak throughput at 100 MHz. The remaining DSP slices are inferred for the on-chip fully connected tail and the requantization lanes outside the systolic core, so the convolutional array arithmetic itself is bounded to 168 multipliers. The control and datapath logic occupies 101,553 LUTs and 165,049 FFs, reflecting a streaming, pipeline-dominated implementation. The on-chip buffers map to 228 BRAMs (30.20%). The reported BRAM utilization is taken from the post-implementation report and includes the major activation and weight buffers together with any parameter, line, and FIFO memories inferred as block RAM. Since the mapping from logical memories to RAMB36 primitives depends on the finalized port widths, depths, banking, and synthesis packing, the physical BRAM count is reported from implementation rather than inferred solely from the logical capacities. The remaining resource headroom on the Zynq-7100 motivates evaluation on

lower-cost Zynq devices, although device-specific rebanking or resource tradeoffs may still be required.

Table 10. Post-implementation FPGA resource utilization, timing, and power of the proposed palm-vein inference engine on the Xilinx Zynq-7100.

Resource	Utilization	Available	Utilization (%)
LUT	101,553	277,400	36.61
FF	165,049	554,800	29.75
BRAM	228	755	30.20
DSP	227	2020	11.24
IO	108	362	29.83
BUFG	1	32	3.13
Clock frequency (MHz)	100		

Latency. The reported 33.03 ms latency is measured from assertion of the PL inference-start signal, after the preprocessed input tensor is resident in the accelerator-visible input buffer, to completion of FC2 and assertion of the inference-done signal. It includes the six convolutional stages, ExtraPool, FC1, and FC2, while excluding camera acquisition, ROI extraction, CLAHE, input transfer into the PL buffer, and the PS-side Softmax and class-selection operations. Of this total, the six convolutional stages complete in 22.47 ms at 100 MHz. Because the pruned channel widths and output-row sizes are exact multiples of the array dimensions, the mapping incurs no structural row or output-channel padding. Nevertheless, the measured per-layer profile in Figure 8 still departs from the MAC distribution in Table 2, since the non-compute overhead of each layer state machine contributes a layer-dependent fraction of the total latency. CONV1 dominates at 6.06 ms despite its relatively small MAC workload because, at $N = 1$, the 14-row skew feeder pays its fixed feeding and draining cost against a single input channel, and the array operates at only 5.5% effective utilization for this layer. In contrast, CONV3 carries the peak-MAC workload but completes in only 3.41 ms by amortizing the fill-and-drain term in Equation (7) over long channel accumulation and column scans. The convolutional bottleneck therefore arises from low input-channel utilization and fixed pipeline overhead rather than from the peak-MAC layer. This on-device observation supports the row-aligned $T_r = 14$ tiling selected in Section 4 and motivates a lightweight first-layer specialization that processes multiple spatial windows concurrently without altering the shared datapath.

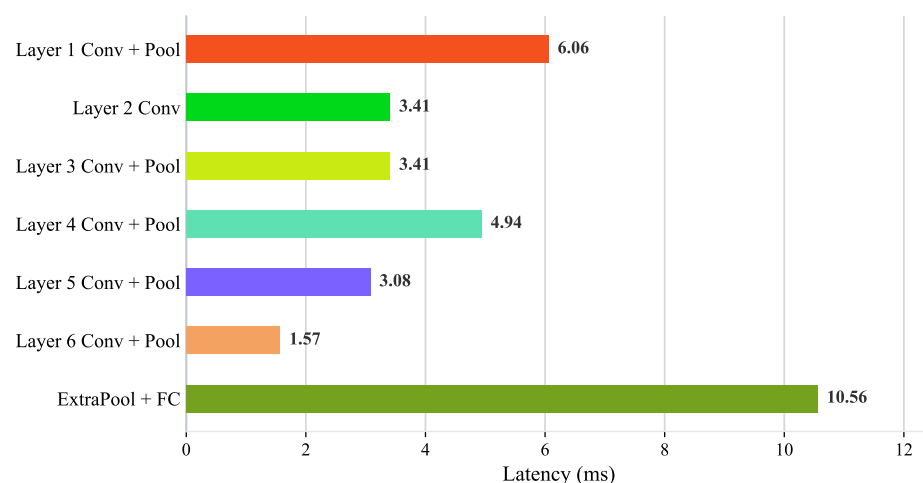


Figure 8. Per-layer inference latency of the six convolutional stages at 100 MHz.

The combined ExtraPool and FC tail is the largest single contributor at 10.56 ms. The projections 540×256 and 256×500 contain 266.24 K INT8 weight coefficients and perform 266.24 K MAC operations per inference. Despite this small arithmetic workload, the FC weights have limited reuse, with each coefficient fetched and used only once, making the tail primarily memory-bound after the OS dataflow has minimized convolutional traffic. Structurally reducing FC1 from 256 to 128 would halve the FC weight and MAC counts to 133.12 K. However, because the reported 10.56 ms combines ExtraPool, FC1, FC2, and their control overhead, the actual latency benefit cannot be inferred directly and requires network retraining and hardware measurement. Structured FC-width reduction and overlap between CONV6 draining and tail processing are promising directions for further optimization.

Power. Figure 9 shows the on-chip power breakdown. The accelerator draws 0.91 W at 100 MHz, of which 0.236 W is static power, and the remainder is dynamic. Within the dynamic budget, clocks and logic contribute 0.323 W and 0.165 W, respectively, signals contribute 0.159 W, and BRAM and DSP together account for less than 0.03 W. The dominant clock, logic, and signal components are consistent with the parallel banked datapath. The fully on-chip memory organization removes inter-layer off-chip transfers, while the OS mapping reduces the modeled output/partial-sum movement and the divisor-aligned array eliminates output-channel padding.

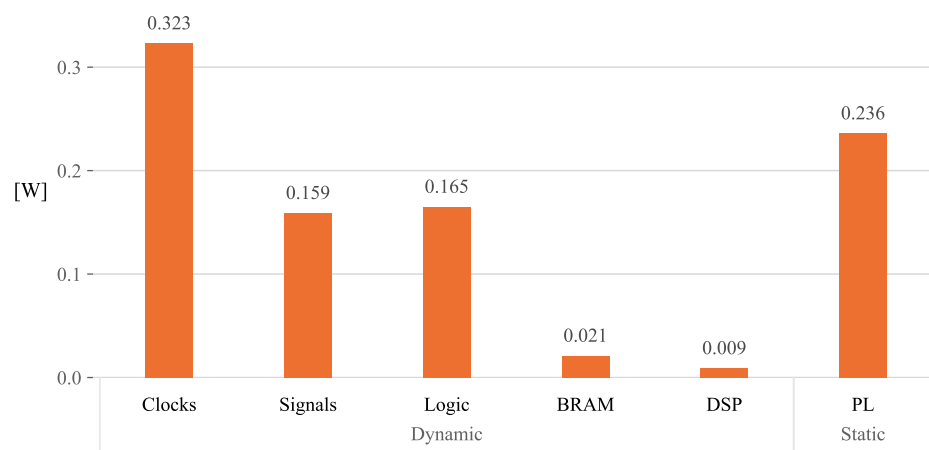


Figure 9. On-chip power breakdown of the accelerator at 100 MHz.

6.5. System Performance

The design is assessed from two complementary angles: recognition quality and model size against representative palm-vein algorithms and on-device latency, power, and energy efficiency against representative biometric and edge-CNN implementations.

Recognition accuracy and model size. The INT8 accelerator produces 500 class logits, and the predicted identity is obtained under the fixed closed-set protocol. As Table 11 shows, the deployed checkpoint achieves 99.50% accuracy with only 0.32 M parameters, the smallest model in the comparison. The performance balance among the other methods is also informative. NAS designs achieve high accuracy but remain computationally heavy and rely on floating-point precision, as demonstrated by Qin [8], which attains 99.40% accuracy with an FP32 model with 9.50 M parameters. In contrast, aggressive compression often leads to accuracy degradation, with Wulandari [36] and Ma [25] reporting accuracies of 85.97% and 98.69%, respectively. By coupling architecture search with hardware-aligned compression in a single pipeline, our model is searched, pruned, and quantized under one deployment budget rather than compressed after the fact, yielding a $29.7\times$ smaller model than Qin at comparable accuracy and a 0.81 percentage point accuracy gain over Ma with a further 15.8% parameter reduction.

Table 11. Recognition accuracy and model size comparison on PolyU palm-vein datasets.

Method	Bit-Prec.	Acc. (%)	Parameters (M)
Si Kaddour [37]	FP32	92.23	62.30
Jia [7]	FP32	97.39	5.30
Qin [8]	FP32	99.40	9.50
Wulandari [36]	FP32	85.97	0.39
Ma [25]	INT8	98.69	0.38
Ours	INT8	99.50	0.32

On-device inference latency and hardware comparison. Table 12 provides external deployment context. Wang [38] is the closest same-device reference because both implementations use the Zynq-7100 FPGA. Chang [3] provides a same-Zynq-7000-family and vein-biometric reference, whereas Janaki [9] provides a CNN-based vein-biometric deployment reference on an FPGA SoC. Nevertheless, these implementations differ in application task, network architecture, arithmetic precision, clock definition, computational workload, and latency and power measurement boundaries. The low engine power and resulting FPS/W of the proposed accelerator are consistent with its task-specific OS mapping, fully on-chip parameter and intermediate-feature storage, INT8 arithmetic, and array geometry matched to the pruned channel widths. However, the heterogeneous comparison cannot isolate the contribution of any individual factor. The approximately $10.4\times$ FPS/W ratio relative to Wang is therefore presented as a same-device deployment comparison rather than a workload-normalized efficiency gain.

Table 12. System comparison with representative biometric and edge-CNN implementations. Latency, power, and energy efficiency are reproduced as reported in the original works. Since the compared designs differ in application task, network architecture, arithmetic precision, platform, and latency-measurement boundary, the results are provided for deployment context rather than workload-normalized ranking.

Method	Platform	Freq. (MHz)	Latency (ms)	Power (W)	Energy Efficiency (FPS/W)
Chang [3]	XC7Z020	650	356.00	N/A	N/A
Janaki [9]	XCZU4EV	N/A	387.20	3.67	0.70
Wang [38]	XC7Z100	200	36.69	8.54	3.19
Ours	XC7Z100	100	33.03	0.91	33.27

7. Conclusions

This paper presented a lightweight FPGA accelerator for palm-vein recognition based on a deployment-driven design methodology that carries target device constraints from model construction through hardware implementation. The proposed framework jointly optimizes recognition accuracy, model size, on-chip storage, data movement, and mapping efficiency. At the algorithm level, a constrained NAS procedure generated a compact 6-stage CNN for 224×224 NIR palm-vein images, followed by array-aligned structured pruning and INT8 quantization-aware training. The pruning process reduced the MAC count from 320.1 M to 87.2 M, while constraining each channel width to a multiple of the systolic array column count eliminated lane padding by design. The resulting INT8 model achieved a recognition accuracy of 99.50%.

On the hardware side, the layer-shape characteristics of the pruned network were explicitly exploited to guide dataflow selection and PE array sizing. An exhaustive analytical DSE over the declared finite grid selected a 14×12 PE array by jointly considering estimated latency, MAC-weighted utilization, the 168 PE target, and geometric compatibility with the network dimensions. Timeloop and Accelergy then provided an independent

cycle and energy ablation of the selected point and five representative alternatives. Under the common analytical accounting boundary, OS reduced the modeled buffer-access count by 42.6% relative to IS and by 34.6% relative to WS. The resulting FPGA inference engine integrates ping-pong on-chip buffers, row reordering, a systolic convolution core, and streaming pooling. Implemented on the Xilinx Zynq-7100 at 100 MHz, the accelerator achieves an on-chip power of only 0.91 W and an energy efficiency of 33.27 FPS/W, reporting approximately $10.4\times$ higher FPS/W than a same-device reference under a different application workload and evaluation protocol. These results demonstrate that palm-vein recognition can be mapped to a compact, low-power FPGA inference engine with fully on-chip model storage and intermediate-feature processing without sacrificing high closed-set identification accuracy.

Nevertheless, several limitations remain.

Algorithmic and evaluation limitations. The current accuracy results were obtained under a controlled single-dataset protocol on PolyU. Because the deployed model is a fixed 500-class classifier, evaluation on datasets with non-overlapping identities requires retraining and therefore assesses architecture portability rather than direct generalization of the same trained model. The repeated-run analysis fixes the architecture selected by a single NAS execution and characterizes the variability of post-selection training and compression rather than that of the complete architecture search; repeating the full NAS procedure across independent seeds remains future work. Variations in participant demographics and acquisition conditions, including age, skin condition, NIR wavelength, camera hardware and spectral response, exposure, ambient-light leakage, acquisition distance, hand pose, ROI alignment, image noise, and long-term cross-session changes, have not yet been comprehensively evaluated. The current closed-set protocol also does not report threshold-based FAR, FRR, EER, or ROC curves.

Hardware and system limitations. The current prototype mainly validates the FPGA recognition engine rather than a fully integrated biometric system. The optical acquisition front end, user-enrollment workflow, and complete system interaction process are not integrated into the reported accelerator and latency-measurement path. Although the convolutional stages, ExtraPool, and fully connected head run on the PL, the final closed-set Softmax and class selection are handled by the PS; a fully PL-resident decision or similarity-matching stage remains future work. Finally, comparisons with prior works are limited by differences in datasets, evaluation protocols, biometric modalities, workloads, FPGA platforms, and latency boundaries, so the cross-system results provide deployment context rather than a workload-normalized ranking.

Future work will therefore focus on three directions. The first is to build a complete acquisition-to-identification prototype by tightly integrating the NIR imaging module, FPGA inference engine, identity management workflow, and user interaction interface. The second is to validate architecture portability on additional public palm-vein datasets through retraining and hardware remapping, together with cross-device and cross-session robustness tests and open-set verification using metric-learning embeddings, enrollment templates, cosine-similarity matching, and standard ROC, FAR, FRR, and EER protocols. The third is to further optimize the hardware architecture by exploring lower-cost Zynq devices, mixed-precision arithmetic, a fully PL-resident decision stage supporting either closed-set class selection or open-set similarity matching, and secure storage mechanisms for model parameters and enrollment templates. At the datapath level, a lightweight first-layer execution mode that packs or broadcasts multiple spatial windows per cycle will be explored to improve PE utilization for CONV1 when $N = 1$. These extensions will move the proposed design from a validated FPGA inference accelerator toward a deployable palm-vein recognition system for access control and other privacy-sensitive applications.

Author Contributions: Conceptualization, X.F.; methodology, X.F. and Y.M.; software, X.F. and Y.M.; validation, X.F.; formal analysis, X.F.; investigation, X.F., Y.M. and B.M.; resources, X.F.; data curation, X.F.; writing—original draft preparation, X.F.; writing—review and editing, X.F., Y.M., B.M. and H.L.; visualization, X.F.; supervision, H.L.; project administration, H.L.; funding acquisition, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Jain, A.K.; Ross, A.; Prabhakar, S. An introduction to biometric recognition. *IEEE Trans. Circuits Syst. Video Technol.* **2004**, *14*, 4–20.
2. Wu, W.; Elliott, S.J.; Lin, S.; Sun, S.; Tang, Y. Review of palm vein recognition. *IET Biom.* **2020**, *9*, 1–10.
3. Chang, R.C.-H.; Wang, C.-Y.; Li, Y.-H.; Chiu, C.-D. Design of low-complexity convolutional neural network accelerator for finger vein identification system. *Sensors* **2023**, *23*, 2184.
4. Kauba, C.; Prommegger, B.; Uhl, A. Combined fully contactless finger and hand vein capturing device with a corresponding dataset. *Sensors* **2019**, *19*, 5014.
5. Zhang, Z.; Wang, M. A simple and efficient method for finger vein recognition. *Sensors* **2022**, *22*, 2234.
6. Luo, S.; Huang, X. A Lightweight Neural Network for Palm Vein Recognition. *Front. Comput. Intell. Syst.* **2022**, *2*, 101–105.
7. Jia, W.; Xia, W.; Zhao, Y.; Min, H.; Chen, Y.X. 2D and 3D Palmprint and Palm Vein Recognition Based on Neural Architecture Search. *Int. J. Autom. Comput.* **2021**, *18*, 377–409.
8. Qin, H.; Fu, Y.; Chen, J.; El-Yacoubi, M.A.; Gao, X.; Xi, F. Neural Architecture Search-Based Global–Local Vision Mamba for Palm-Vein Recognition. *IEEE Trans. Inf. Forensics Secur.* **2026**, *21*, 3766–3780.
9. Janaki, K.; Srinivasan, C.; Hema Malini, A. FPGA-Enhanced System-on-Chip for Finger Vein-Based Biometric System Using Novel DL Model. *Integration* **2024**, *98*, 102231.
10. Reginald, P.J. Energy-Efficient FPGA-Based Accelerator Architecture for Real-Time Edge AI Applications in Embedded Systems. *J. VLSI Embed. Syst. Des.* **2026**, 1–10.
11. Wan, Y.; Xie, X.; Chen, J.; Xie, K.; Yi, D.; Lu, Y.; Gai, K. ADS-CNN: Adaptive Dataflow Scheduling for Lightweight CNN Accelerator on FPGAs. *Future Gener. Comput. Syst.* **2024**, *158*, 138–149.
12. Mu, H.; Guo, J.; Cheng, Y.; Liu, X.; Han, C.; Sun, L. Palm vein recognition based on dual-channel convolutional neural network. In *Proceedings of the 6th International Conference on Computational Intelligence and Applications (ICCIA), Xiamen, China, 11–13 June 2021*; IEEE: Piscataway, NJ, USA, 2021; pp 155–159.
13. Jia, W.; Gao, J.; Xia, W.; Zhao, Y.; Min, H.; Lu, J.T. A performance evaluation of classic convolutional neural networks for 2D and 3D palmprint and palm vein recognition. *Int. J. Autom. Comput.* **2021**, *18*, 18–44.
14. Zhou, Y.; Kumar, A. Human identification using palm-vein images. *IEEE Trans. Inf. Forensics Secur.* **2011**, *6*, 1259–1274.
15. Wu, K.-S.; Lee, J.-C.; Lo, T.-M.; Chang, K.-C.; Chang, C.-P. A secure palm vein recognition system. *J. Syst. Softw.* **2013**, *86*, 2870–2876.
16. Wirayuda, T.A.B. Palm vein recognition based-on minutiae feature and feature matching. In *Proceedings of the International Conference on Electrical Engineering and Informatics (ICEEI), Denpasar, Indonesia, 10–11 August 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 350–355.
17. Wang, C.; Peng, M.; Xu, L.; Chen, T. A single scale retinex based method for palm vein extraction. In *Proceedings of the IEEE Information Technology, Networking, Electronic and Automation Control Conference, Chongqing, China, 20–22 May 2016*; IEEE: Piscataway, NJ, USA, 2016; pp. 75–78.
18. Guo, X.; Wang, C. A palm-vein recognition algorithm based on LPP and HM-LBP. In *Proceedings of the 2018 2nd International Workshop on Renewable Energy and Development (IWRED 2018), Guilin, China, 20–22 April 2018*; IOP Conference Series: Earth and Environmental Science; IOP Publishing: Bristol, UK, 2018; Volume 153.
19. Sun, S.; Cong, X.; Zhang, P.; Sun, B.; Guo, X. Palm vein recognition based on NPE and KELM. *IEEE Access* **2021**, *9*, 71778–71783.
20. Wu, W.; Li, Y.; Zhang, Y.; Li, C. Identity recognition system based on multi-spectral palm vein image. *Electronics* **2023**, *12*, 3503.
21. Obayya, M.I.; El-Ghandour, M.; Alrowais, F. Contactless palm vein authentication using deep learning with Bayesian optimization. *IEEE Access* **2020**, *9*, 1940–1957.
22. Babalola, F.O.; Bitirim, Y.; Toygar, Ö. Palm vein recognition through fusion of texture-based and CNN-based methods. *Signal Image Video Process.* **2021**, *15*, 459–466.

23. Alrahawe, E.A.; Humbe, V.T.; Shinde, G.N. A contactless palm veins biometric system based on convolutional neural network. In *Proceedings of the 1st International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, Sana'a, Yemen, 10–12 August 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
24. Shen, W.; Liang, J. End-to-end multibranch network for palm vein recognition and liveness detection. *J. Electron. Imaging* **2024**, *33*, 013054.
25. Ma, Y.; Pu, N.; Chen, Z.; Xu, Y.; Liu, H. Hardware-Friendly Palm Vein Recognition Algorithm Using Hybrid Compressed Convolutional Neural Network. In *Proceedings of the 2025 13th International Conference on Bioinformatics and Computational Biology (ICBCB)*, Seoul, Republic of Korea, 27 February–2 March 2025; IEEE: Piscataway, NJ, USA, 2025; pp. 22–29.
26. Murillo-Escobar, M.A.; Meranza-Castillón, M.O.; López-Gutiérrez, R.M.; Cruz-Hernández, C. Suggested integral analysis for chaos-based image cryptosystems. *Entropy* **2019**, *21*, 815.
27. Vats, P.; Tadepalli, S.K. Palm vein image processing and enhancement in vein pattern recognition system on FPGA. In *Proceedings of the 2nd International Conference on Emerging Frontiers in Electrical and Electronic Technologies (ICEFEET)*, Patna, India, 24–25 June 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–5.
28. Yadav, A.; Gupta, V.K.; Kumar, B. FPGA-based medical image processing using hardware-software co-design approach. *IEEE Trans. Biomed. Circuits Syst.* **2025**, *20*, 57–68.
29. Liu, P.; Wang, Y. A low-power general matrix multiplication accelerator with sparse weight-and-output stationary dataflow. *Micromachines* **2025**, *16*, 101.
30. Mao, W.; Yang, P.; Wang, Z. FTA-GAN: A computation-efficient accelerator for GANs with fast transformation algorithm. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *34*, 2978–2992.
31. Zhou, Y.; Yen, G.G.; Yi, Z. A knee-guided evolutionary algorithm for compressing deep neural networks. *IEEE Trans. Cybern.* **2021**, *51*, 1626–1638. <https://doi.org/10.1109/TCYB.2019.2928174>.
32. Mondal, M.; Das, B.; Roy, S.D.; Singh, P.; Lall, B.; Joshi, S.D. Adaptive CNN filter pruning using global importance metric. *Comput. Vis. Image Underst.* **2022**, *222*, 103511.
33. Kim, N.; Shin, D.; Choi, W.; Kim, G.; Park, J. Exploiting retraining-based mixed-precision quantization for low-cost DNN accelerator design. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 2925–2938. <https://doi.org/10.1109/TNNLS.2020.3008996>.
34. Xu, Y.; Liu, H. Lightweight arrhythmia classifier using hybrid compressed convolutional neural network. In *Proceedings of the 11th International Conference on Bioinformatics and Computational Biology (ICBCB)*, Hangzhou, China, 21–23 April 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 64–70. <https://doi.org/10.1109/ICBCB57893.2023.10246560>.
35. Liu, D.; Zhu, Y.; Liu, Z.; Liu, Y.; Han, C.; Tian, J.; Li, R.; Yi, W. A survey of model compression techniques: Past, present, and future. *Front. Robot. AI* **2025**, *12*, 1518965.
36. Wulandari, M.; Chai, R.; Basari, B.; Gunawan, D. Hybrid Feature Extractor Using Discrete Wavelet Transform and Histogram of Oriented Gradient on Convolutional-Neural-Network-Based Palm Vein Recognition. *Sensors* **2024**, *24*, 341.
37. Si Kaddour, S.; Aberni, Y.; Boubchir, L.; Raddadi, M.; Daachi, B. Convolutional Neural Algorithm for Palm Vein Recognition using ZFNet Architecture. In *Proceedings of the 2021 4th International Conference on Bio-Engineering for Smart Technologies (BioSMART)* Paris / Créteil, France, 8–10 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–4.
38. Wang, W.; Zhou, D.; Xie, W.; Zhang, W. An Energy-Efficient FPGA-Based CNN Accelerator with Dual-Multiply Packing and Ping-Pong Buffering for Real-Time Object Detection. *Electronics* **2026**, *15*, 2442.

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.