

UC Davis

UC Davis Electronic Theses and Dissertations

Title

HALO-8: Hardware Acceleration for Lightweight Object Detection on DE10-Lite

Permalink

<https://escholarship.org/uc/item/9qt3g183>

ISBN

9798189233135

Author

Saxena, Arunima

Publication Date

2026-06-25

Peer reviewed|Thesis/dissertation

HALO-8: Hardware Acceleration for Lightweight Object Detection on DE10-Lite

By

ARUNIMA SAXENA

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Hussain Al-Asaad, *Chair*

Venkatesh Akella

A. N. Gündeş

Committee in Charge

2026

Abstract

This thesis examines whether a sub-\$100 bare-fabric FPGA can run a modern one-stage object detector for PCB defect inspection, and what the silicon-grounded cost story looks like once you measure it. The target platform is the Terasic DE10-Lite, carrying an Intel MAX 10 part with 49,760 logic elements, 288 nine-bit DSP slices, 205 KB of on-chip block RAM, and no hard processor. The workload is YOLOv8n trained on DeepPCB and quantized to INT8 with ONNX Runtime’s static post-training pipeline.

The accelerator is an 8x8 Output-Stationary systolic array of INT8 multiply-and-add units written in SystemVerilog, with per-channel re-quantization and tile-based on-chip memory. All six representative convolution layers picked from the network (input channels from 3 to 384, kernel sizes 1 and 3, strides 1 and 2, output spatial dimensions 20 to 320) verify silicon bit-exact against the ONNX Runtime reference within a two-byte tolerance across 2,611,200 output bytes. A host-orchestrated coprocessor harness feeds the FPGA outputs back into a surgically modified ONNX graph; on a DeepPCB validation image the post-NMS detection list is identical to pure software (5 of 5 boxes, no deltas).

Across the six silicon data points, cycles-per-MAC settles at 0.158 in the deep regime (input channels at or above 32) and 0.195 in the shallow stem. Projecting these two regimes across the full 195-layer graph gives 6.73 seconds per inference at 50 MHz on a single board; a speculative 100 MHz design point with a PLL and BRAM banking would reach about 600 milliseconds. The measured data does not support the original per-throughput cost framing: a single \$249 Jetson Orin Nano Super outperforms any feasible HALO-8 fleet, and a saturated \$200 Zynq-7020 implementation is roughly two orders of magnitude cheaper per FPS. The contribution is therefore

reframed along three axes the data does support: the lowest single-station capital cost for end-to-end CNN-based PCB defect inspection, below both the \$249 Orin Nano and the \$30K–\$200K range typical of inline AOI; a deterministic bare-fabric datapath with no operating system, kernel, or driver in the inference loop; and the lowest steady-state per-station idle power. Among surveyed YOLO-on-FPGA implementations, none target the bare-fabric sub-\$200 tier addressed here.

Contents

Contents

Abstract.....	2
Chapter 1	1
Introduction.....	1
1.1 Defects on a PCB	1
1.2 PCB Inspection methods	3
1.3 DE-10 Lite Specifications.....	4
1.4 FPGA vs GPU	6
1.5 Research questions	7
1.6 My Contributions.....	7
1.7 Thesis Outline.....	8
Chapter 2.....	9
Literature Survey	9
2.1 Overview	9
2.2 YOLO on edge GPUs	9
2.3 YOLO on FPGAs	10
2.4 PCB Defect Detection.....	11
2.5 Commercial AOI.....	12
2.6 The empty cell HALO-8 occupies	13
Chapter 3.....	15

Software Design	15
3.1 Introduction to YOLOv8n.....	15
3.1.1 The six demonstrator layers.....	18
3.2 Need for INT8 quantization.....	18
3.2.1 Hardware Cost of Floating-Point vs Integer Math	18
3.2.2 Symmetric vs Asymmetric INT8.....	18
3.2.3 Post Layer Re-quantization.....	19
3.2.4 The Quantization DeQuantization ONNX format.....	20
3.3 CNN Dataflow Patterns.....	21
3.4 The Avalon-MemoryMapped bus.....	23
 Chapter 4	 24
 System Design and Implementation.....	 24
4.1 Pipeline overview	24
4.2 INT8 quantization pipeline	25
4.2.1 Bit-Width Design Space Exploration	25
4.2.2 Calibration and Training Challenges	26
4.2.3 The Deployed Quantization Pipeline.....	27
4.3 Host-to-board communication.....	28
4.4 On-chip memory layout.....	30
4.5 The 8x8 Processing Element array.....	31
4.5.1 PE Components	31
4.5.2 The Chosen Output Stationary Dataflow.....	31
4.5.3 The Re-Quantization block	32
4.5.4 Dataflow Model Comparison	32
4.6 Spatial tiling for the bigger layers of YOLOv8n.....	33
4.7 End-to-End Co-Processor Harness	35

4.8	Design and Verification Infrastructure	36
4.8.1	RTL Design.....	36
4.8.2	The Quartus Design Flow.....	37
4.8.3	The TCI Test-Vector Pipeline	37
4.8.4	Implementation Challenges and Workarounds	37
Chapter 5.....		39
Results and Evaluation		39
5.1	Quantization Accuracy	39
5.1.1	Configurations Analyzed	39
5.1.2	The INT8 Accuracy.....	40
5.1.3	Before-and-After Analysis Per Image.....	41
5.1.4	Experiments	41
5.2	Dataflow Comparison.....	42
5.2.1	Rationale for Output-Stationary Dataflow.....	43
5.3	Silicon Bit-Equality Verification	44
5.4	The Cycles-per-MAC Cost Model.....	45
5.4.1	The Accuracy Gap	45
5.4.2	Full-Network Projection at 50 MHz.....	46
5.4.3	The 100 MHz speculation.....	46
5.5	End-to-End NMS-Identical Result	46
5.6	Benchmarks and Results	48
Chapter 6		51
Discussion and Limitations		51
6.1	Scope and Boundaries of Data	51
6.2	Verification Gaps.....	52

6.3	Validity Threats	53
6.4	Relationship to Existing Research	55
Chapter 7		57
Conclusion and Future Work		57
7.1	Summary of Contributions.....	57
7.2	Key Findings and Insights.....	58
7.3	Future Work.....	58
7.4	Conclusion.....	59
References		61

List of Figures

1.1	The six PCB defect classes used in the DeepPCB benchmark [1]: (1) open, (2) copper residue, (3) spur, (4) short, (5) pin-hole, (6) mouse-bite.....	2
1.2	HALO-8 subsystem block diagram on the Terasic DE10-Lite. The accelerator consists of a JTAG-Avalon master [2,15], a region-decoded engine bridge with three single-buffered BRAMs (input, weight, output), a control-register block, and an 8x8 PE output-Stationary array.....	6
3.1	YOLOv8n architecture. Backbone (top) extracts multi-scale features through five stride-2 downsamples and four C2f blocks. Neck (middle) is a PAN-FPN fusing P3/P4/P5 scales. Head (bottom) emits $4 + n_c$ outputs per anchor on each scale.....	16
3.2	The integer requantization pipeline that ends every quantized Conv layer. The INT32 accumulator (acc) carries the sum over the inner reduction loop. A per-channel INT32 bias is added, then a per-channel UInt32 fixed-point multiplier rescales the result, a 31-bit right shift implements the divide, the enforces the.....	19
3.3	The four canonical CNN dataflow patterns from the Eyeriss taxonomy [3]. Each cell represents a PE in an 8 by 8 array; cell color shows what is held Stationary during the inner reduction loop.....	21
4.1	The four phases of HALO-8. Phase 1: model training on DeepPCB. Phase 2:INT8 quantization. Phase 3: hardware design. Phase 4: silicon bring-up on the DE10-Lite.....	25
4.2	The three-stage HALO-8 quantization pipeline. FP32 baseline, fake-quantization sweep across W4A4 to W16A16, then ONNX Runtime static INT8 PTQ at the deployed bit-widths.....	28
4.3	The host-to-FPGA communication stack. TCl on the host calls System Console, which	

	talks to the USB-Blaster, which drives JTAG-Avalon into the fabric. The eight 256 KB regions are decoded inside the engine bridge	29
4.4	On-chip BRAM layout for one trigger. Input tile, eight-channel weight slab, eight-channel output slab. Region addresses decode from the top three bits of the JTAG-Avalon address	30
4.5	One PE in the Output Stationary array. INT8 multiplier, INT32 partial-sum accumulator, drain path through the per-channel re-quantization block	33
4.6	Spatial tiling patterns across the six demonstrator layers. Stem uses row strips, mid-network layers use quartering, the smallest layer uses a single quadrant	34
4.7	The end-to-end coprocessor harness. Host runs ONNX Runtime on the non-demonstrator layers. FPGA runs the six demonstrator Convs, 1 bitstream at a time. Host orchestrates the handoff and stitches outputs back into the graph	36
5.1	Same DeepPCB validation image, FP32 and INT8 PTQ. Five detections in both cases at the same bounding-box positions; confidence values shift by a few percent. The deployed INT8 model produces the same defect calls as the FP32 baseline on this image.	40
5.2	mAP@0.5 across the four quantization configurations. The deployed INT8 PTQ retains roughly 93% of FP32	41
5.3	Precision-recall curves for FP32 and the deployed INT8. The curves overlap closely above 0.5 confidence, where the deployed inference operates	42
5.4	SignalTap II capture from silicon, Layer 3 Output Stationary run. Top: heartbeat and busy/done. Middle: the eight per-PE INT32 accumulators rising over the	

inner reduction. Bottom: re-quantization drain pulse and BRAM write.....	45
5.5 Measured cycles per MAC across the six demonstrator layers. Five points cluster at 0.158; the stem sits at 0.195. The horizontal lines mark the deep- regime constant and the stem outlier.....	47
5.6 USD-per-FPS across HALO-8 (50 MHz silicon, 100 MHz projection) and surveyed baselines.....	49
6.1 Power-throughput Pareto front. HALO-8 below one watt at sub-1 FPS. Orin Nano Super at 8.345 W and 43 to 45 FPS [4]. Zynq-7020 saturated at 2.55 W and 208 FPS [5].....	55

List of Tables

1.1	Structural limitations of commercial AOI.....	3
2.1	Surveyed YOLO-on-FPGA implementations, sorted by ascending board cost. Image sizes vary across rows; comparisons match model variant and precision row-by-row.....	12
3.1	The six demonstrator layers covered by HALO-8 silicon verification.....	17
4.1	Bit-width design space exploration. mAP@0.5 retention measured by fake-quantization in PyTorch on DeepPCB.....	26
5.1	Quantization accuracy on DeepPCB validation. FP32 baseline, vanilla INT8 PTQ, mixed-sensitivity INT8 (abandoned), and QAT (fell below PTQ on the small training set).....	39
5.2	All four canonical dataflows benchmarked on Layer 3 ($64 \times 64 \times 40 \times 40$, $K=3$). Cycle counts, logic-element usage, and per-MAC efficiency.....	43
5.3	Silicon verification of the six demonstrator layers. All layers verified bit-equal within the FP64-versus-fixed-point tie budget. Cumulative byte difference across the six layers is 2 out of 2,611,200.....	44

Chapter 1

Introduction

1.1 Defects on a PCB

Printed circuit boards sit inside almost every electronic product that matters. Smartphones, cars, medical devices like pacemakers and infusion pumps, and avionics flight controllers all rest on the same green substrate. The board carries every signal between every chip in the system. If the board is wrong, the system is wrong. In a safety-critical domain, a faulty board can be the failure that ends a flight or a heartbeat.

A bare board leaves a fabrication line carrying microscopic flaws from any of the steps that built it. Copper deposition, photoresist patterning, etching, drilling, plating, and solder-mask application each have their own failure modes. Six of these flaws account for the majority of bare-board defects in the public DeepPCB benchmark [1, 6], and Figure 1.1 shows one example of each. The list below names the physical cause and the downstream cost.

1. **Open:** A missing section of copper that leaves a trace disconnected. The cause is usually over-etching or a scratch that severs the conductor before plating. The downstream cost is the highest of the six: any open in a signal trace makes the device fail to work at all.
2. **Copper residue:** Leftover copper in regions that should be bare substrate. The cause is an exhausted etch bath or an incomplete etch. Residue between fine-pitch pads is a frequent source of intermittent shorts that pass initial test but fail after a few thermal cycles.
3. **Spur:** A thin copper protrusion sticking out from an otherwise correct trace. The cause is under-etching at a sharp corner or a stray particle that masked the etchant. Spurs

reduce clearance to neighboring conductors and can trigger arcing or vibration shorts in service.

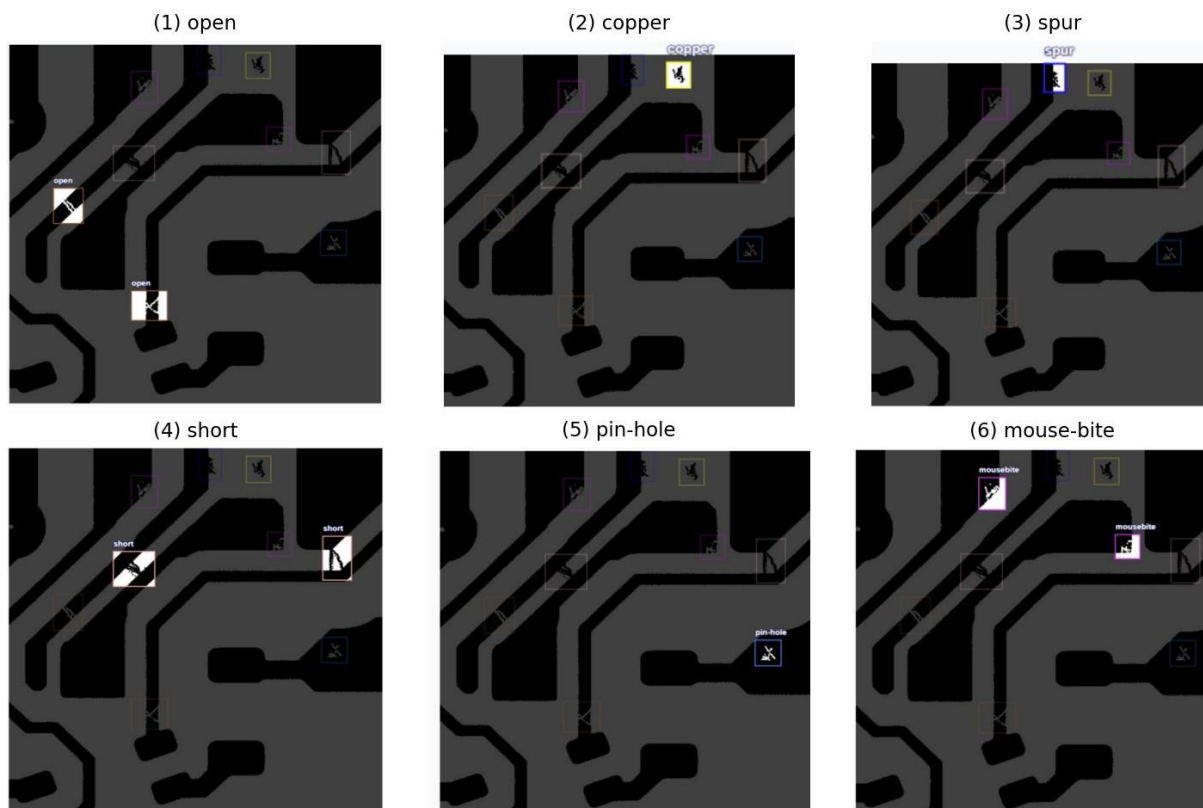


Figure 1.1: The six PCB defect classes used in the DeepPCB benchmark [1]: (1) open, (2) copper residue, (3) spur, (4) short, (5) pin-hole, (6) mouse-bite

4. **Short:** An unintended copper bridge between two conductors that should stay isolated. The cause is usually under-etching, a pinhole in the photoresist or stray solder during reflow. A short can damage adjacent circuits the moment power is applied.
5. **Pin-hole:** A small void in an otherwise solid copper region. The cause is a bubble or particle in the resist or plating step. Pin-holes reduce the cross-section available for current and create local hot spots that age the dielectric beneath them.
6. **Mouse-bite:** A small semicircular notch eaten into the edge of a trace. The cause is localized over-etching or a particle of contamination during the etch step. Mouse-bites raise local resistance and current density. They often pass initial test but shorten the useful life of the board through electromigration.

A defect that escapes inspection at the bare-board stage propagates into assembly, test, and field. Each subsequent step is more expensive to rework than the one before it. The earliest inspection point is therefore the cheapest place to catch the problem.

1.2 PCB Inspection methods

PCB inspection is not one technique. It is a sequence of checks, each one good at finding a different kind of problem at a different point in production. A standard manufacturing line uses several of these in combination.

The first check is **bare-board Automated Optical Inspection (AOI)**. A high-precision camera photographs the freshly-etched board and a rule-based pipeline compares each image against a golden reference [7]. This is the camera step that catches the six defect classes in Section 1.1. The next check, **X-ray inspection**, looks through chip packages to find solder

Table 1.1: Structural limitations of commercial AOI.

Limitation	Details
Cost	Commercial AOI systems cost USD \$30K to \$200K per unit [9], making them inaccessible to SMEs, R&D labs, and developing-nation manufacturers.
Inflexibility	AOI is reference-based; any change in board revision, component placement, or lighting requires manual recalibration [11].
No generalization	Rule-based logic cannot generalize to novel defect morphologies or previously unseen board revisions without explicit reprogramming.
False-alarm rate	Pfab et al. recorded 97.2 to 99.7 percent false-call rate across 440,274 inspections on a Siemens SMT line [10]. The dominant operational cost in classical AOI is operator review of false positives, not detection.
Infrastructure dependency	AOI requires a controlled factory environment [12]. It is not suitable for field audits or flexible production cells.

joints hidden underneath ball-grid arrays after assembly. **In-Circuit Test (ICT)**, using either a bed-of-nails fixture or a flying-probe tester, applies electrical signals to confirm that the right nets are connected. **Functional test** powers the assembled board and

checks that it behaves the way the designer intended. **Thermal imaging** on an energized board can flag hot spots that signal a power problem or a partial short [8].

This thesis compares against AOI because AOI is the camera step where the six defect classes of Section 1.1 are detected. The other methods are complementary, not competing. Catching a defect earlier with a cheaper box makes every later step cheaper too.

Commercial AOI is mature, but it has well-known structural limits. A tier-one inline AOI unit costs between 30,000 and 200,000 USD [9], which puts it out of reach for small and medium manufacturers, university labs, and shops in developing regions. The pipeline is also rule-based, so any change in board revision or lighting needs manual re-tuning. The most operationally expensive failure mode is the false call. Pfab et al. recorded a false-call rate of

97.2 to 99.7 percent across 440,274 inspections on a Siemens surface-mount line [10]. The dominant cost of running classical AOI is not catching defects; it is paying an operator to review false positives. Table 1.1 summarizes these limits.

A third path has emerged with deep learning. Convolutional neural networks, in particular the YOLO family [13], now match or beat AOI image processing on the six defect classes above. The usual hardware substrate is a small edge GPU (Graphic Processing Unit) such as the NVIDIA Jetson Orin Nano, which can run YOLOv8n on PCB workloads at 43 to 45 frames per second [4]. The Orin Nano Super carries a 249 USD list price [14]. This path solves AOI’s accuracy and flexibility problem, but it adds three new ones: non-deterministic latency from operating-system jitter, power in the single-digit to tens-of-watts range, and a closed software stack on commodity silicon.

1.3 DE-10 Lite Specifications

HALO-8 runs on a Terasic DE10-Lite development board [15]. The board carries an Intel

MAX 10 FPGA, part number 10M50DAF484C7G. The fabric holds 49,760 logic elements, 288 nine-bit DSP blocks, and roughly 205 kilobytes of on-chip block RAM [16]. A 64-megabyte SDRAM chip sits beside the FPGA. The system clock is a 50 MHz oscillator on the board, and the host link is a USB-Blaster JTAG cable. The board retails below 100 USD.

What matters as much as what the board has is what it does not have. The DE10-Lite has no hard processor. There is no ARM Cortex like the one on a Zynq board, no embedded operating system, and no Linux kernel to fall back on. There is no Ethernet PHY, no SD card slot, and no native USB device port. The chip is pure programmable logic. Everything that runs on the board has to be built in hardware from first principles. That constraint is the whole point of this thesis. The cost claim only works if the fabric stays this small.

The physical layout of the board places the MAX 10 chip at the center, with a row of ten red status LEDs (LEDR0 through LEDR9) along one edge, two push-buttons (KEY0 and KEY1) for user input, ten slide switches for binary inputs, six seven-segment displays for numeric readout, the USB-Blaster connector for host communication and programming, a 40-pin GPIO header and an Arduino-compatible header for external boards, a VGA connector, and an accelerometer chip.

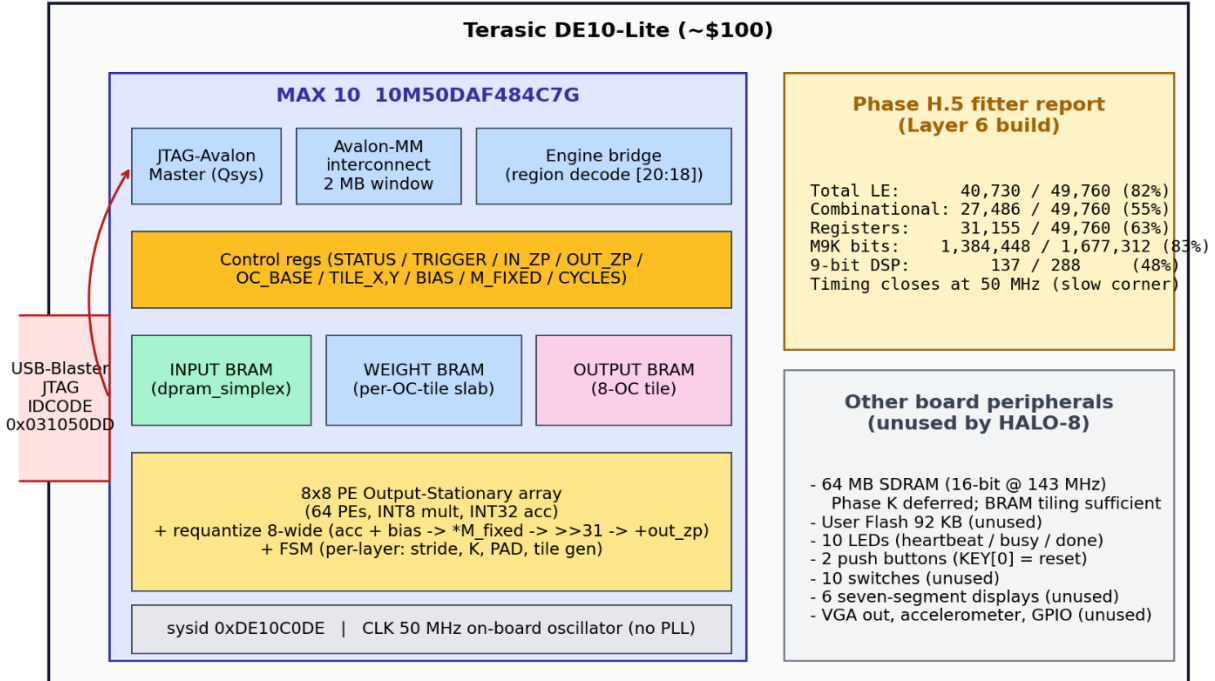


Figure 1.2: HALO-8 subsystem block diagram on the Terasic DE10-Lite. The accelerator consists of a JTAG-Avalon master [2, 15], a region-decoded engine bridge with three single-buffered BRAMs (input, weight, output), a control-register block, and an 8x8 PE output-Stationary array.

1.4 FPGA vs GPU

An FPGA at this price point is a deliberate choice. The work needed to design it is high. In exchange the design gets five things that the GPU and AOI paths give up. Latency is deterministic at the cycle, with no operating system in the loop to add jitter. Static power is low, which suits an always-on inspection station. Parallelism is shaped to the workload, so a multiply-accumulate array can be laid out the way a convolution wants it. The unit cost is lower than any alternative on the market. The system is standalone at the edge with no cloud, no host x86, and no GPU in the inference path.

Earlier FPGA work on PCB defect detection has run on Zynq-class SoC boards. Pan et al. accelerate YOLOx-Plus(You Only Look Once) on a PYNQ-Z2 [17], and Yan et al. run YOLOv5s on a ZCU102 [18]. Both of those boards carry a hard ARM processor on

the same die as the programmable logic. The ARM handles the parts of the network the fabric cannot easily run and provides a Linux environment for control. The DE10-Lite has no such crutch. Removing the ARM is what makes this contribution novel and what makes the cost target reachable.

1.5 Research questions

This thesis asks three research questions:

RQ1. Can INT8 post-training quantization of YOLOv8n hold acceptable accuracy on a six-class industrial PCB defect benchmark, and what fails first when calibration data is scarce?

RQ2. Can six representative YOLOv8n convolution layers be mapped onto an Intel MAX 10 FPGA using a custom 8x8 systolic PE array with on-chip-only memory tiling, and what cycles-per-MAC cost does the silicon actually deliver?

RQ3. Can a host-orchestrated coprocessor harness over JTAG-Avalon eliminate the need for a soft-core CPU in the fabric, and what is the resulting end-to-end inference flow?

1.6 My Contributions

This thesis makes four contributions.

C1. The first publicly documented silicon-verified YOLOv8n accelerator at the bare-fabric MAX 10 sub-200 USD cost tier. None of the surveyed YOLO-on-FPGA papers target this tier; the closest peers are PYNQ-class Zynq-7000 systems at roughly 200 USD [5, 19], each of which carries a hard ARM Cortex-A9 that HALO-8 does not.

C2. A two-regime cycles-per-MAC cost model grounded on six silicon-verified data points. The deep regime, with input channel count of 32 or more, sits at 0.158 cycles per MAC. The shallow stem regime, with three input channels, sits at 0.195 cycles per MAC. Both numbers come from on-board cycle counters and are load-bearing for the full-

network projection in Chapter 5.

C3. An end-to-end coprocessor demonstration that produces NMS-identical detections on a representative DeepPCB image. Five of five detections match the pure-ORT reference exactly in class, bounding box, and confidence. The FPGA runs the six demonstrator convolution layers and the host runs the rest of the network in ONNX Runtime.

C4. An honest, literature-grounded reframing of the cost claim. Aggregate-throughput parity at 0.149 frames per second and 671 USD per FPS does not survive against a 5.79 USD-per-FPS Orin Nano [4, 20] or a 0.96 USD-per-FPS saturated Zynq-7020 peer [5]. Three narrower axes do survive: single-station capital cost, deterministic datapath fixed at synthesis time, and idle power per station.

1.7 Thesis Outline

The rest of this manuscript is organized as follows. Chapter 2 surveys related work on YOLO on FPGA, edge-GPU baselines, PCB defect benchmarks, and commercial AOI. Chapter 3 covers the background a reader needs: YOLOv8n architecture, INT8 quantization with the QDQ format, the dataflow taxonomy for multiply-accumulate arrays, and the Avalon-MM bus [2]. Chapter 4 presents the system design and implementation: the six-layer demonstrator set, the quantization pipeline that produces a deployed INT8 model at mAP@0.5 of 0.901 against an FP32 ceiling of 0.968, and the host-orchestrated coprocessor harness. Chapter 5 reports results and evaluation across all six demonstrator layers, the cycles-per-MAC regime characterization, the full-network projection, and the NMS-identical end-to-end demonstration. Chapter 6 discusses the cost-claim reframing, threats to validity, and limitations of the demonstrator. Chapter 7 concludes and points to avenues for follow-on work.

Chapter 2

Literature Survey

2.1 Overview

This survey collects 41 sources across four buckets. The buckets are PCB defect detection methods, YOLO on FPGAs, YOLO on edge GPUs, and commercial Automated Optical Inspection (AOI) references. Each cited result is recorded with platform, framework, model variant, dataset, precision, throughput, latency, accuracy, power, and board cost in USD when available. Comparisons in this chapter match on those axes and flag deviations explicitly.

Three findings shape the rest of the thesis. First, the legacy 99-USD Jetson Nano baseline is technically defective. Its Maxwell GPU (compute capability 5.3) lacks the DP4A INT8 instruction, so every public "Nano INT8 YOLO" number conflates the legacy device with the 249-USD Orin Nano (compute capability 8.7, DP4A-capable). The legacy device was discontinued in 2023. Second, no surveyed YOLO-on-FPGA paper targets the bare-fabric, sub-200-USD tier; every 200-USD peer is a PYNQ-class Zynq-7000 with a hard ARM. Third, tier-1 AOI machines pair 3D moire profilometry with AI-augmented threshold tuning. The dominant operational cost is operator review of false calls, not detection itself [10].

2.2 YOLO on edge GPUs

The legacy 99-USD Jetson Nano baseline is broken. The launch device [29] has a Maxwell GPU at compute capability 5.3, which predates the Pascal-introduced DP4A INT8 instruction (compute capability 6.1). On this device INT8 falls back to FP16 or FP32

emulation. Every public "Nano INT8 YOLO" benchmark therefore conflates the legacy device with the Orin Nano (compute capability 8.7). The legacy device was discontinued in 2023.

The best credible YOLO numbers on the legacy Nano are three. Anand [30] reports 6.1 FPS for YOLOv8n in PyTorch FP32. YOLOv5s at TensorRT FP16 reaches 27 FPS [31]. YOLOv7-tiny at FP16 reaches 15 FPS [32]. None of these clears 30 FPS on a single 640 by 640 streams.

The Orin Nano Super at 249 USD MSRP [14] is the currently-purchasable edge-GPU peer at HALO-8's price point. It carries an Ampere GPU at compute capability 8.7, 67 TOPS, and 7 to 25 W TDP. The cleanest peer-reviewed YOLOv8n INT8 benchmark on this device is Rey et al. [4] at 43 to 45 FPS and 8.345 W under TensorRT INT8 at 640 by 640 with batch size 1. They also report a 14 percentage-point drop from FP16 (0.9619) to INT8 (0.8272), the most relevant PTQ-accuracy precedent in this survey. Alqahtani et al. [20] contribute the survey's only wall-meter power measurement, about 5 W above the SoC INA-chip reading on Jetson dev kits. A community trt exec benchmark [33] reports 157 QPS for YOLOv8s INT8 at 6.4 ms; real video pipelines drop about 50 percent [34]. The AGX class [35, 36] reaches 313 to 719 FPS at 2K to 3.5K USD, outside HALO-8's direct comparison band.

2.3 YOLO on FPGAs

Three observations follow from Table 2.1 which summarizes the surveyed YOLO-on-FPGA implementation-

- First, no surveyed paper targets the 100-USD tier. The cheapest peers (PYNQ-Z2, Zybo-Z7-20, Zynq-7020) sit at 200 USD, all with hard ARMs.

- Second, every Xilinx peer is a Zynq SoC, which enables Vitis AI, FINN+Brevitas, and PetaLinux runtimes. The MAX 10 has no such vendor stack. HALO-8’s RTL is hand-written SystemVerilog and the host drives the board through System Console TCl over JTAG-Avalon. This is a real cost-axis difference that trades developer effort for hardware cost.
- Third, the precision distribution skews sub-INT8. Three papers use W4A4 [5, 19, 27]; one uses INT4 [18]; one uses INT16 [25]. HALO-8 sticks with INT8 because it is the most widely-supported production quantization across ORT, TensorRT, and FPGA toolchains.

2.4 PCB Defect Detection

DeepPCB [1] is the de facto public PCB benchmark. It contains 1,500 paired 640 by 640 crops across six defect classes: open, short, mouse-bite, spur, copper, and pin-hole. HALO-8 trains, calibrates, and evaluates on this dataset.

The original Tang et al. number of 98.6 percent is reported at $\text{IoU} = 0.33$. That threshold is not directly comparable to the COCO-standard $\text{IoU} = 0.5$. This is a recurring source of inflated PCB-defect numbers in the literature, so the threshold is cited every time a number appears.

Recent DeepPCB-targeted work at the standard IoU drives $\text{mAP}@0.5$ into the high nineties on workstation GPUs. YOLO-MBBi [21] reaches 99.0 percent at 69.5 FPS. YOLO-BFRV [22] reaches 98.6 percent at 142 FPS. CDI-YOLO [23] reports $\text{mAP}@0.5:0.95 = 0.511$, which is one of the few papers to give the stricter metric. YOLO-PCB [24] is another DeepPCB-targeted entry in this band.

HALO-8’s deployed INT8 PTQ achieves 0.901 $\text{mAP}@0.5$ and 0.517 $\text{mAP}@0.5:0.95$ on DeepPCB. The gap on $\text{mAP}@0.5$ is about 9 percentage points below the FP32 state of the art, but the stricter metric sits within 0.6 points of CDI-YOLO. The gap on the looser metric is consistent

with INT8 PTQ losses on small datasets. Rey et al. [4] report a 14 percentage-point drop (0.9619 FP16 to 0.8272 INT8) under TensorRT INT8 PTQ on a similarly small drone set, which is the closest published precedent.

Two hardware-deployment PCB papers anchor the positioning. Pan et al. [17] run YOLOX-Plus on a PYNQ-class Zynq at 72.6 FPS and 93.2 percent mAP on a custom 4,000-image PCB set. The PYNQ-Z2 carries a hard ARM Cortex-A9 that the DE10-Lite does not have, so the comparison is not platform-equivalent. Yan et al. [18] run W4 YOLOv5s on a 2,500-USD ZCU102 at 79.1 FPS and 12.3 W, which is one cost tier above HALO-8.

Table 2.1: Surveyed YOLO-on-FPGA implementations, sorted by ascending board cost. Image sizes vary across rows; comparisons match model variant and precision row-by-row.

Citation	Board	Model	Precision	FPS	Lat. (ms)	Power (W)	Cost (USD)
Gunay 2022 [19]	PYNQ-Z2	TinyYOLOv3	W4A4	18	55	2.4	200
Cali 2025 [5]	Zynq-7020	YOLOv3-Tiny	W4A4/W8A8	208	4.8	2.55	200
Kim 2025 [25]	Zybo-Z7-20	YOLOv2	INT16	0.08	12200	2.04	200
Pan 2024 [17]	PYNQ-class	YOLOX-Plus	INT8	72.6	13.8	n/r	~200
Chen 2023 [26]	AXU2CGB	YOLOv4-Tiny	fixed-pt	2.14	468	3.52	300
Yan 2024 [18]	ZCU102	YOLOv5s	INT4	79.1	12.6	12.3	2500
Danilowicz 2025 [27]	ZCU102	YOLOv8n	W4A4	195.3	n/r	n/r	2500
Wang 2019 [28]	Arria-10 GX1150	YOLOv2	INT8	72.5	13.8	n/r	5000

2.5 Commercial AOI

Tier-1 AOI vendors (Koh Young, Mirtec, Omron, Saki) do not publish unit pricing. Triangulated cost ranges from public sources [9, 11, 12] are as follows. Bench-top 2D AOI runs 3K to 20K USD. Mid-range inline 2D runs 30K to 60K USD. Tier-1 inline 3D runs 120K to 200K-plus USD new and 60K to 90K USD used. A five-year total-cost-of-ownership

multiplier of about 2.3 times purchase price is reported on a worked EMS-tier example [12].

The only public power datapoint is 0.6 kW from one inline 2D system [37]. Tier-1 3D systems are reasonably estimated at 1 to 3 kW, but that is not citable from datasheets and is flagged as an estimate in the evaluation chapter.

Pfab et al. [10] release a 132-day Siemens SMT-line dataset of 440,274 AOI inspections. The per-inspection-type true-defect-to-false-call ratio is 0.003 to 0.028, so 97.2 to 99.7 percent of flags are false calls. The dominant operational cost is therefore operator review of false positives, not detection itself. Tier-1 AOI machines are not end-to-end CNN systems; they pair 3D moire profilometry with AI-augmented threshold tuning on top [7–9, 11]. The natural integration for a low-cost CNN accelerator like HALO-8 is therefore as a false-call-reduction post-processor downstream of classical AOI, not as a replacement for the optical front end.

2.6 The empty cell HALO-8 occupies

Table 2.2 places the surveyed papers on a two-axis taxonomy. The rows separate hard-processor SoC parts from bare-fabric FPGAs. The columns separate sub-200-USD, 200-USD, and 2,500-USD-plus boards.

HALO-8 is the first silicon-grounded data point in the bare-fabric sub-200-USD cell. The (bare-fabric, 200-USD) cell is also empty, and the (hard-CPU SoC, sub-200-USD) cell is empty because the smallest Zynq dev boards retail at 200 USD. This positioning has a corollary that the results chapter discusses in detail: 671 USD per FPS at 50 MHz does not beat the Zynq peer on a single-board basis, so the cost claim is reframed along three narrower axes.

3 FPGA implementations sit closest to HALO-8 in cost-class positioning:

- **Cali et al. [5]** report a saturated 200-USD Zynq-7020 design that reaches 208 FPS at 2.55 W on YOLOv3-Tiny, occupying 98.6 percent of the available BRAM and 92.7 percent of the

DSP blocks. The corresponding 0.96 USD per FPS is approximately a factor of 700 below HALO-8's projected 671 USD per FPS at 50 MHz.

- **Günay et al. [19]** map a redesigned TinyYOLOv3 in W4A4 to the 200-USD PYNQ-Z2 and report 18 FPS at 2.4 W.

- **Danilowicz and Kryjak [27]** deploy W4A4 YOLOv8n on a 2,500-USD ZCU102 through a modified FINN streaming compiler, reaching 195.3 FPS. This result demonstrates YOLOv8n at W4A4 on UltraScale+ devices rather than on a sub-200-USD bare-fabric FPGA.

The first axis is single-station capital cost. The second is the deterministic datapath, which is useful for hard-real-time-deadline inspection cells. The third is idle power per station, which compounds across an SME fleet where most stations are between boards. Each of these is defended on its own terms in the evaluation chapter.

Chapter 3 that follows build the technical foundation needed to evaluate those three claims. It begins with the network and hardware layers HALO-8 targets, the quantization scheme that makes integer inference possible, the dataflow pattern that governs the processing-element array, and the bus interface that connects the FPGA to the host.

Chapter 3

Software Design

This chapter describes the technical details a reader needs to follow the rest of the thesis. Section 3.1 demonstrates the YOLOv8n network and the six representative layers HALO-8 builds in hardware. Section 3.2 explains INT8 quantization, the integer requantization step every layer end with, and the QDQ ONNX representation. Section 3.3 covers the four canonical dataflow patterns for a systolic processing-element array. Section 3.4 describes the Avalon-MM bus, the memory-mapped interface the host uses to read and write FPGA memory over the USB-Blaster JTAG cable.

3.1 Introduction to YOLOv8n

YOLOv8 [36, 38] is a one-stage object detector descended from the original YOLO [13]. A one-stage detector predicts bounding boxes and class labels for every object in an image with a single forward pass over the network. There is no separate region-proposal stage, which keeps inference fast and the implementation compact. The "n" (nano) suffix marks the smallest deployment configuration of the family. It is the variant HALO-8 targets because its parameter counts fits inside the BRAM budget of a 100-dollar FPGA. The network has three sections as shown in Figure 3.1. The backbone runs the input image through a sequence of stridden convolutions and bottleneck blocks to extract features at progressively smaller spatial resolutions. The neck fuses the features the backbone produced at three different scales. The head emits the final bounding-box coordinates and class scores for each scale.

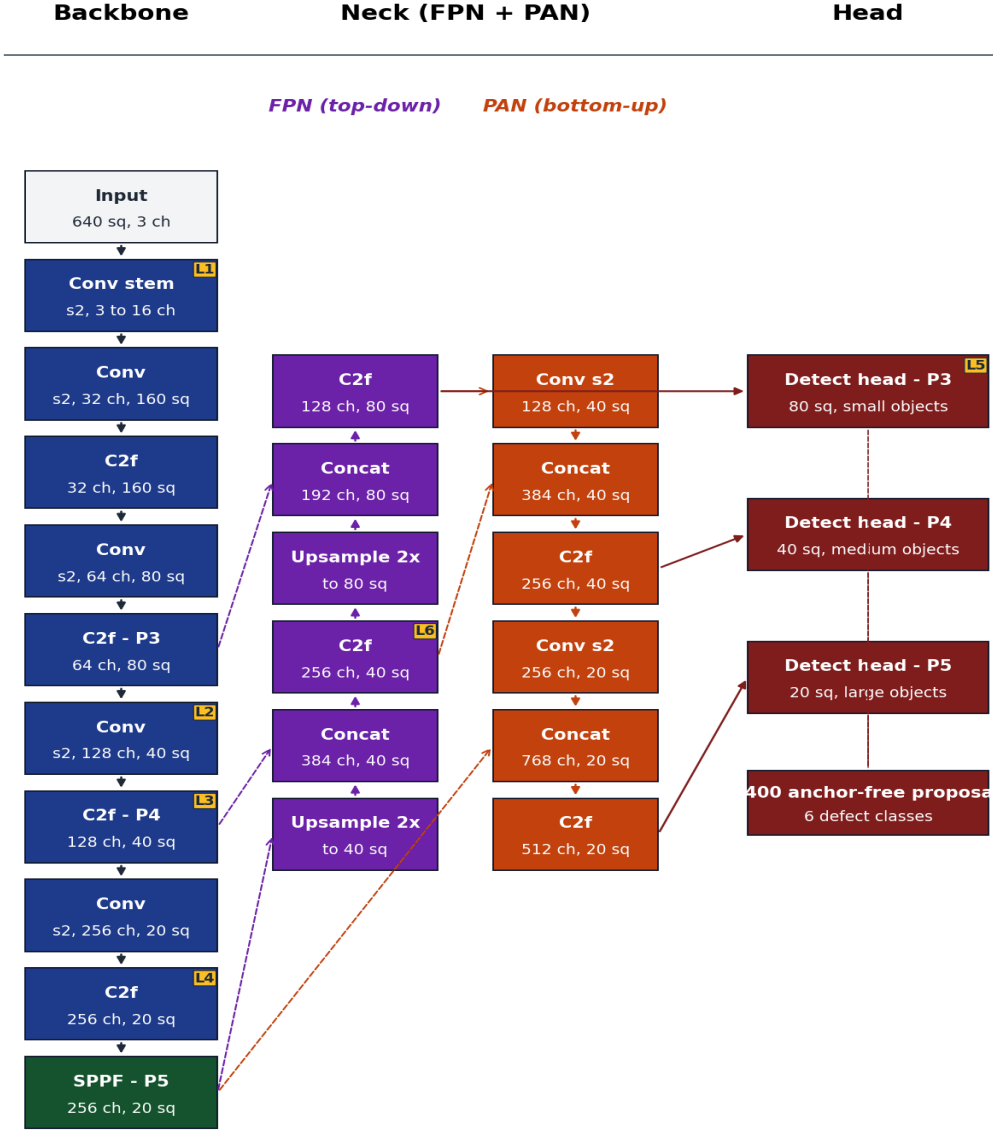


Figure. 3.1: YOLOv8n architecture. Backbone (top) extracts multi-scale features through five stride-2 downsamples and four C2f blocks. Neck (middle) is a PAN-FPN fusing P3/P4/P5 scales. Head (bottom) emits $4 + n_c$ outputs per anchor on each scale.

smaller spatial resolutions. The neck fuses the features the backbone produced at three different scales. The head emits the final bounding-box coordinates and class scores for each scale.

The backbone starts with a stride-2 stem that takes the 640 by 640 RGB image and produces a 320 by 320 feature map with 16 channels. Four C2f (Cross-Stage Partial, fast) stages follow. Each C2f stage doubles the channel count and halves the spatial dimensions,

so the channel progression is 16, 32, 64, 128, 256. A C2f block contains a 1 by 1 projection, several residual bottleneck blocks of two 3 by 3 convolutions, and a final 1 by 1 projection. The backbone ends with an SPPF (Spatial Pyramid Pooling, Fast) layer that combines multiple pooling window sizes into one tensor.

The neck is a PAN-FPN, a top-down and bottom-up feature pyramid [22]. It fuses three feature scales: P3 at 80 by 80, P4 at 40 by 40, and P5 at 20 by 20. Upsample-concatenate operations carry high-level features down to finer scales; downsample-concatenate operations carry fine-scale features back up. C2f blocks sit between the fusion points.

The cv3 branch predicts n_c class logits. For DeepPCB [1] the dataset has six $4+n_c$, 8400), where the head runs two decoupled branches per scale, called cv2 and cv3. The cv2 branch predicts $4 \times \text{reg_max}$ distribution focal loss (DFL) logits [39] for the four box coordinates. defect classes, so $n_c = 6$. The branches are concatenated into a pre-NMS tensor of shape $(N, 8400 = 80^2 + 40^2 + 20^2)$ is the sum of anchor positions across the three scales. Non-maximum suppression then collapses overlapping predictions into the final detection list. The fully unrolled inference graph contains 195 convolutional layers and roughly 2.13 billion multiply-accumulate operations per image.

Table 3.1: The six demonstrator layers covered by HALO-8 silicon verification.

#Layer	K	S	Channels	Out H×W	MACs	Why This Layer
L1	3	2	3→16	320×320	44.2M	Stem; bandwidth-bound, low input channels
L2	3	2	64→128	40×40	118M	Mid-backbone downsample
L3	3	1	64→64	40×40	59M	Core dataflow benchmark
L4	3	1	128→128	20×20	59M	Late backbone, compute-dense; largest 3×3, head branch
L5	3	1	64→64	80×80	236M	Mid-magnitude 1×1, no spatial-reuse regime
L6	1	1	384→128	40×40	78.6M	Mid-magnitude 1×1, no spatial-reuse regime

3.1.1 The six demonstrator layers

HALO-8 picks six representative Conv layers for silicon verification. The six together span the workload’s diversity along the dimensions that drive cost: number of input channels, kernel size, stride, and output spatial dimensions. If the hardware computes these six correctly, the remaining 189 follow by interpolation. Table 3.1 lists them.

3.2 Need for INT8 quantization

3.2.1 Hardware Cost of Floating-Point vs Integer Math

Trained YOLOv8n stores weights and activations as 32-bit floating-point numbers. Floating-point hardware is expensive. An INT8 multiplier takes roughly one-sixteenth of the silicon area of an FP32 multiplier and roughly one-quarter of an FP16 multiplier at the same throughput [40]. The DE10-Lite has 288 nine-bit DSP blocks; using those at INT8 widths gives the design enough multiplier budget to run a useful slice of the network in parallel. The price is some accuracy loss, which is the trade quantization makes.

3.2.2 Symmetric vs Asymmetric INT8

Quantization maps a real-valued tensor X to an integer tensor with a scale factors and a zero-point z . The scale s is a positive floating-point number that fixes the step size between adjacent integer levels. The zero-point z is an integer offset that lets the integer range sit anywhere along the real-number line, not just centered at zero. The forward mapping is:

$$\hat{X} = \text{clip}(\text{round}(X/s) + z, a_{\min}, a_{\max})$$

For signed INT8 the range is $[a_{\min}, a_{\max}] = [-128, 127]$. For unsigned UInt8 the range is $[0, 255]$. The inverse mapping (used when an integer tensor needs to be read

as a real number) is:

$$X = s \times (\hat{X} - z)$$

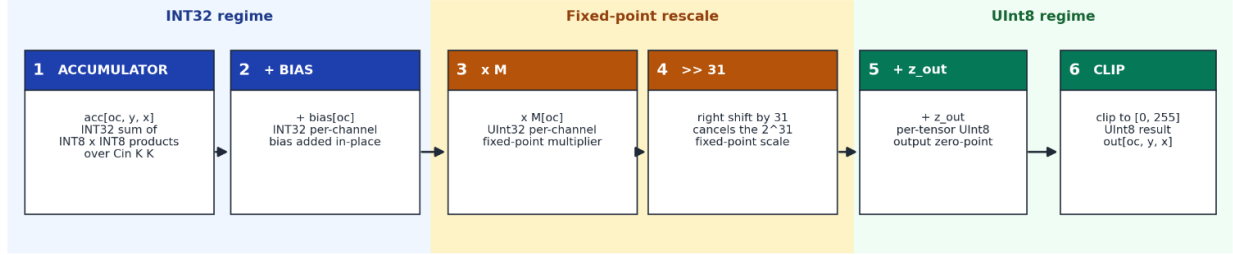


Figure 3.2: The integer requantization pipeline that ends every quantized Conv layer. The INT32 accumulator (acc) carries the sum over the inner reduction loop. A per-channel INT32 bias is added, then a per-channel UInt32 fixed-point multiplier rescales the result, a 31-bit right shift implements the divide, the output zero-point shifts the answer into the UInt8 range, and a clip enforces the [0,255] range.

HALO-8 follows the ONNX Runtime static PTQ recipe. Weights are per-channel symmetric INT8. Each output channel gets its own scale $s_w[oc]$; the zero-point is fixed at zero. Symmetric weights are a good fit for trained networks because the weight distribution is roughly centered at zero, so the integer range matches the value range without wasting half the levels on values that never occur. Zero zero-point also simplifies the multiplier datapath; there is no zero-point correction term to add.

Activations are per-tensor asymmetric UInt8. SiLU outputs are mostly positive with a long-left tail; they are not centered at zero. A symmetric quantization would waste the negative half of the range. Per-tensor (rather than per-channel) is enough for activations because the dynamic range varies less across activation channels than across weight channels.

3.2.3 Post Layer Re-quantization

The multiplications inside a Conv layer happen in INT8. Each product is INT16 or INT17; the running sum across the inner reduction loop fits in an INT32 accumulator. At the end of the layer, the INT32 accumulator must be compressed back to UInt8 so the next layer can consume it. This step is called requantization. Figure 3.2 traces the pipeline. The

hardware performs five operations per output value, in order:

First, add the per-channel INT32 bias to the accumulator. Second, multiply by the per-channel UInt32 fixed-point scaling factor $M[oc]$. Third, right-shift the 64-bit product by 31, which implements the division by 2^{31} . Fourth, add the per-tensor output zero-point z_{out} .

Fifth, clip the result to the UInt8 range $[0, 255]$. In one equation:

$$out[oc, y, x] = \text{clip} \left(\text{round} \left((acc[oc, y, x] + bias[oc]) \times M[oc] / 2^{31} \right) + z_{out}, 0, 255 \right)$$

The fixed-point multiplier $M[oc]$ is computed once at model-conversion time by the host.

Its definition is:

$$M[oc] = \text{round} \left(s_{in} \times s_w[oc] / s_{out} \times 2^{31} \right)$$

The 2^{31} factor turns the floating-point ratio $s_{in} \times s_w[oc] / s_{out}$ into a fixed-point integer the multiplier can consume directly. The right shift by 31 cancels the factor back out. The whole pipeline matches what ONNX Runtime’s QLinearConv kernel does on the CPU side, which is why FPGA outputs match ORT outputs bit-for-bit.

3.2.4 The Quantization DeQuantization ONNX format

ONNX Runtime’s static quantization produces a Quantize-Dequantize graph. The deployed model is still nominally an FP32 graph; the Conv operators are sandwiched between two helper nodes. A QuantizeLinear node sits in front of each Conv and turns FP32 activations into UInt8. A DequantizeLinear node sits behind each Conv and turns the UInt8 output back to FP32 for the next operator to consume. The ORT CPU runtime recognizes the Quant-Conv-Dequant pattern and fuses it into a single QLinearConv kernel at execution time. Tools that do not understand the INT8 path can still read the file as a regular ONNX graph. For HALO-8 the QDQ format matters in two

places. First, the per-layer extractor walks each demonstrator’s QDQ subgraph to recover everything the hardware needs: INT8 weights, INT32 bias, FP32 input and weight scales, and zero-points. These are saved as eight NumPy arrays per layer and become the golden test vectors for silicon verification. Second, the end-to-end demonstrator surgically replaces six Conv blocks in the DQ graph with new

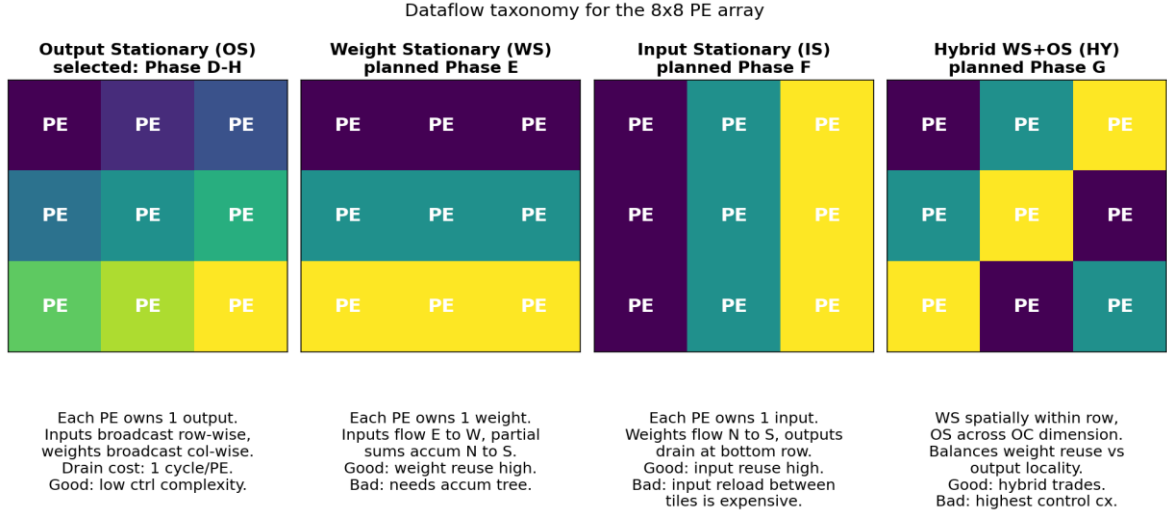


Figure 3.3: The four canonical CNN dataflow patterns from the Eyeriss taxonomy [3]. Each cell represents a PE in an 8 by 8 array; cell colour shows what is held Stationary during the inner reduction loop.

UInt8 input ports, so that the host can feed FPGA-computed outputs back into the ORT graph as if they had come from the original Conv. The QDQ wrapping makes the substitution clean; the FPGA-computed integer drops into the exact tensor position the rest of the graph expects.

3.3 CNN Dataflow Patterns

When the design instantiates a systolic PE array, every PE has three operands per multiply: an input activation, a weight, and a partial sum. Only two of the three can be moving between PEs on any given cycle; the third has to sit still in a PE register. Which operand stays still is the dataflow choice. The Eyeriss taxonomy [3] catalogues four

common answers. Figure 3.3 illustrates each one on the 8 by 8 grid HALO-8 uses.

Output Stationary (OS). Each PE owns one output anchor (one specific output channel and one specific output pixel). Inputs broadcast row-wise, weights broadcast column-wise. Each PE accumulates into a local INT32 register; at the end of the inner loop the accumulator drains in place into the requantization block and the result is written out. OS has the simplest control logic and the lowest per-PE register pressure of the four.

Weight Stationary (WS). Each PE holds one weight value for the layer. Inputs flow east-to-west through the grid; partial sums accumulate north-to-south down the columns. This is the dataflow used by the original Google TPU [41]. WS maximises weight reuse, so each weight loaded into a PE is multiplied against many input values before being replaced. **Input Stationary (IS).** Each PE preloads one input pixel. Weights stream past column-wise, partial sums drain at the bottom row. IS maximises input reuse, but the Stationary property breaks if the input volume is larger than the grid can hold at once. For YOLOv8n’s typical feature-map sizes the input is much larger than 64 pixels, so the Stationary register has to be reloaded between tiles and the savings shrink.

Hybrid (WS + OS). Weights Stationary within rows, output channels across columns. This balances weight reuse against output locality, at the cost of higher control logic. HALO-8’s hybrid variant puts one weight per PE in a row and one output channel per column, so each PE has both a weight register and a partial-sum register.

HALO-8 implements all four variants (Phases D through G of the build log) on the same Layer 3 workload and on the same Avalon-MM shell, then compares them on cycle count and logic-element usage. The Output Stationary variant wins on both axes and is the dataflow ported to the other five layers.

3.4 The Avalon-MemoryMapped bus

The Avalon Memory-Mapped (Avalon-MM) interface is the synchronous, byte-addressable bus used pervasively in Intel FPGA SoC designs [2]. From the host PC's perspective, the FPGA's internal memory blocks appear as a flat range of byte addresses. The host writes or reads 32-bit values to specific addresses, and the FPGA fabric routes those transactions to the right BRAM block or control register.

HALO-8 exposes one Avalon-MM master, the JTAG-Avalon Master, driven from host System Console TCI across the USB-Blaster cable. The address bus is 21 bits wide and

the data bus is 32 bits, so the host sees a 2 MB window. The top three address bits (`s_address[20:18]`) select one of eight 256 KB regions. Region 0 holds the control registers (layer shape, trigger, status, cycle counter). Region 1 holds the input BRAM. Region 2 holds the weight BRAM. Region 3 holds the output BRAM. Regions 4 through 7 are reserved for the variant dataflows' scratchpad memories.

System Console's `master_write_32` and `master_read_32` commands each return in roughly half a millisecond over USB-Blaster, which gives a sustained throughput of about 1.8 KB per second. That figure is the bottleneck on the end-to-end demonstrator's wall-clock time. The accelerator itself runs at 50 MHz internally and could keep up with a link several thousand times faster; the slow link is a property of the JTAG demonstrator build, not of the accelerator design.

Each region's address window is 256 KB, but the physical BRAM behind the window is only 16 to 64 KB depending on the layer. Out-of-range reads alias back to the start of the region. This is acceptable because the host driver knows each layer's dimensions and never issues an out-of-range read; if it did, the alias would silently return the wrong data, but in practice the driver bounds-checks every transaction at the Python layer before issuing it.

Chapter 4

System Design and Implementation

4.1 Pipeline overview

The HALO-8 system splits into four pieces of work as shown in Figure. 4.1. The first piece is model preparation: take the FP32 YOLOv8n weights trained on DeepPCB and quantize them to INT8 so the math fits inside fixed-point hardware. The second piece is hardware design: lay out the on-chip storage, build a systolic array of multiply-and-add units, and write the address generators. The third piece is the link between the host PC and the FPGA. The DE10-Lite has no operating system and no fast network port, so all traffic moves through a JTAG cable. The fourth piece is the coprocessor harness. The FPGA runs six of the network's layers, the ones we built in silicon. The remaining layers stay on the host CPU, and a Python driver stitches the FPGA outputs back into the rest of the YOLOv8n graph so a real image runs end-to-end. Each piece is testable in isolation, and that is how the project was built. The quantization pipeline runs on the host and produces an ONNX file plus per-layer golden tensors. The hardware design runs in Quartus and ModelSim and produces six bitstreams. The communication link uses System Console and Tcl. The harness wires the three together. The rest of this chapter takes each piece in turn.

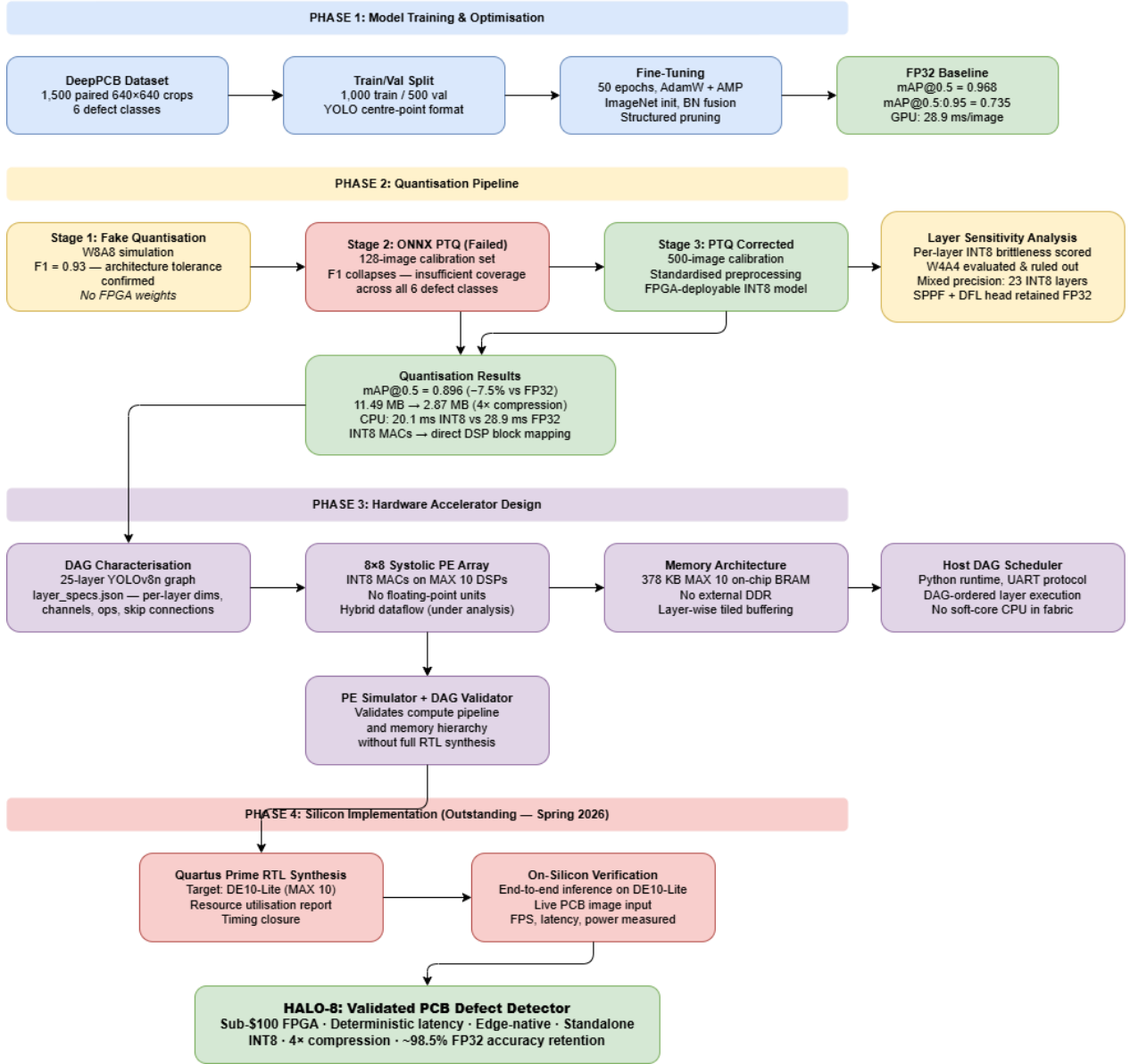


Figure 4.1: The four phases of HALO-8. Phase 1: model training on DeepPCB. Phase 2: INT8 quantization. Phase 3: hardware design. Phase 4: silicon bring-up on the DE10-Lite.

4.2 INT8 quantization pipeline

4.2.1 Bit-Width Design Space Exploration

Before committing to INT8, we walked the weight-bit by activation-bit design space using fake-quantization in PyTorch. The candidates ran from W16A16 (a 16-bit weight and 16-bit

activation hybrid that mostly matches FP32 accuracy) down to W4A4 (a four-bit-everywhere setting that pushes the model into integer-MAC territory).

Each configuration was evaluated against the DeepPCB validation set as shown in Table 4.1 using simulated quantization, with the original FP32 baseline kept as the reference ceiling.

Table 4.1: Bit-width design space exploration. mAP@0.5 retention measured by fake-quantization in PyTorch on DeepPCB.

Configuration	W bits	A bits	Size (MB)	Speedup	Accuracy	Deploy verdict
FP32 (baseline)	32	32	11.49	1.0x	100.00%	reference
BF16 / FP16	16	16	5.74	4.0x	99.90%	rejected (>8b: needs DSP cascade)
W16A8	16	8	5.74	8.0x	99.50%	rejected (>8b weight: DSP cascade)
W8A16	8	16	2.87	8.0x	99.60%	rejected (>8b act: DSP cascade)
INT8 (W8A8)	8	8	2.87	16x	98.50%	deployed (native 9x9 MAX 10 DSP)
W8A4	8	4	2.87	32x	96.50%	rejected (non-uniform datapath)
W4A8	4	8	1.44	32x	95.00%	rejected (non-uniform datapath)
INT4 (W4A4)	4	4	1.44	64x	88.00%	rejected (12 pp acc drop; no MAX 10 LUT-pack)

Three observations settled the choice. First, W4A4 dropped detection accuracy below the level we would call useful for inspection work, and the smaller multiplier did not buy us a corresponding throughput win on a MAX 10 because the LUT-packed four-bit multiplier flow does not exist for this device family. Second, INT8 retained roughly 93% of FP32 mAP@0.5 once measured on real silicon with realistic calibration, well within the band a quality team would tolerate for a screening pass. Third, INT8 is the most widely supported precision in production toolchains (ONNX Runtime, TensorRT, OpenVINO), which matters for a thesis that wants to be reproducible without lock-in to one vendor. INT8 won by default.

4.2.2 Calibration and Training Challenges

Our first attempt used ONNX Runtime’s static quantization with the default 128-image calibration set. The pipeline ran cleanly and produced a valid INT8 graph. The mAP@0.5 then collapsed to near zero. The first convolution and a handful of head layers had

activation ranges broader than anything the 128 images had shown the calibrator. The calibrator picked scale factors that fit its sample, but during validation those scales saturated the activation histograms and threw away most of the signal. This is a known PTQ failure mode when the calibration set is too small to capture the tails of the distribution, and the symptom (a model that loads fine but predicts nothing) is easy to misread as a quantization bug.

The fix was to grow the calibration set to 500 images drawn from the DeepPCB validation split. The histograms then covered the operating distribution properly, the calibrator picked sensible scales, and the deployed model settled at $\text{mAP}@0.5 = 0.901$ against the FP32 baseline of 0.968. The story is worth telling because the failure mode looks like the model is broken when only the calibration set was undersized.

4.2.3 The Deployed Quantization Pipeline

The deployed pipeline as shown in Figure. 4.2, uses per-channel symmetric INT8 for weights (zero-point fixed at zero, one scale per output channel), per-tensor asymmetric UInt8 for activations (because SiLU outputs are not zero-centered and forcing them to be symmetric wastes one bit of dynamic range), the QuantizeLinear-DequantizeLinear ("QDQ") storage format that ONNX Runtime expects, and INT32 biases stored alongside their parent convolutions. One node, the final cross-scale concatenation in the detection head, is excluded from quantization. Without that exclusion every output confidence dequantizes to zero, a quirk well known in the YOLO post-training quantization community. We also tried full quantization-aware training (QAT) on the same DeepPCB training set. The eager-mode PyTorch QAT pipeline ran cleanly after eight bug fixes and ten epochs of fine-tuning, but the final $\text{mAP}@0.5$ settled at 0.860, below the PTQ baseline of 0.901. The

cause is the small training set: 1,000 images is not enough for the FakeQuantize observers to find a quantization-friendly weight distribution within a reasonable budget. Rey et al.

[4] report a similar 14 percentage point drop in a small-dataset drone QAT setting. PTQ won by default. The QAT artefact is archived for a future iteration on an augmented DeepPCB.

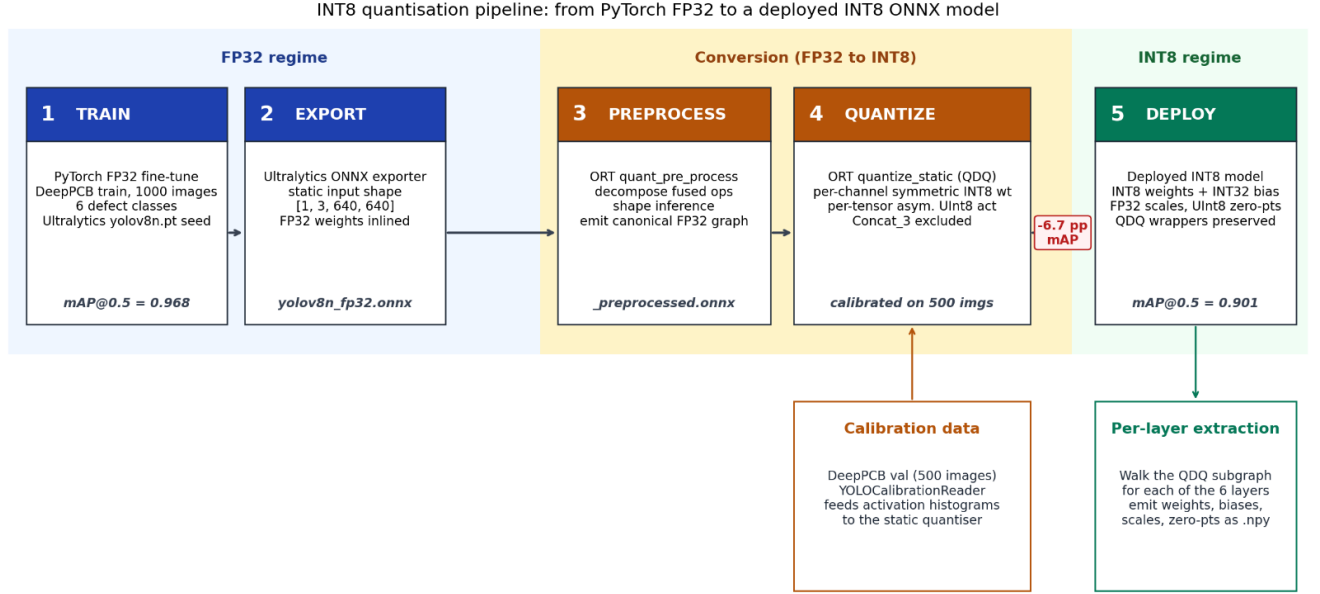


Figure 4.2: The three-stage HALO-8 quantization pipeline. FP32 baseline, fake-quantization sweep across W4A4 to W16A16, then ONNX Runtime static INT8 PTQ at the deployed bit-widths.

4.3 Host-to-board communication

The PC and the FPGA talk over a USB-Blaster cable plugged into the board's JTAG header. On the PC side, Intel's System Console runs a TCL script that opens a JTAG-Avalon Master and reads or writes 32-bit words at 21-bit byte addresses [2] as shown in Figure.4.3. On the FPGA side, that master is the only "outside world" port. Everything inside the design (control registers, input memory, weight memory, output memory, the engine itself) connects to the master directly or through small Avalon bridges.

We carved the 2 MB address window into eight 256 KB regions, decoded from the top three bits of the byte address. Region 0 holds about 26 control registers, one for each piece of per-trigger configuration (layer dimensions, the input and output zero-points, per-channel biases and rescaling constants, and the trigger bit itself). Region 1 holds the input

feature-map BRAM. Region 2 holds the weight BRAM. Region 3 holds the output BRAM. Regions 4 through 7 are reserved for dataflow-variant scratchpads used by the weight-Stationary and input-Stationary alternatives we built for comparison. Each `master_write_32` or `master_read_32` returns in about half a millisecond, so sustained throughput is about 1.8 KB per second. For a single-layer trigger this is fast enough. Writing a 64-by-40-by-40 input tile is roughly 100 KB and takes about a minute, fine for a one-image bring-up run. For the end-to-end six-layer demo the JTAG link is the wall-clock bottleneck. The silicon-side compute number (the cycles-per-MAC that the cost projection rests on) is unaffected, because the FPGA's compute timer is independent of the host.

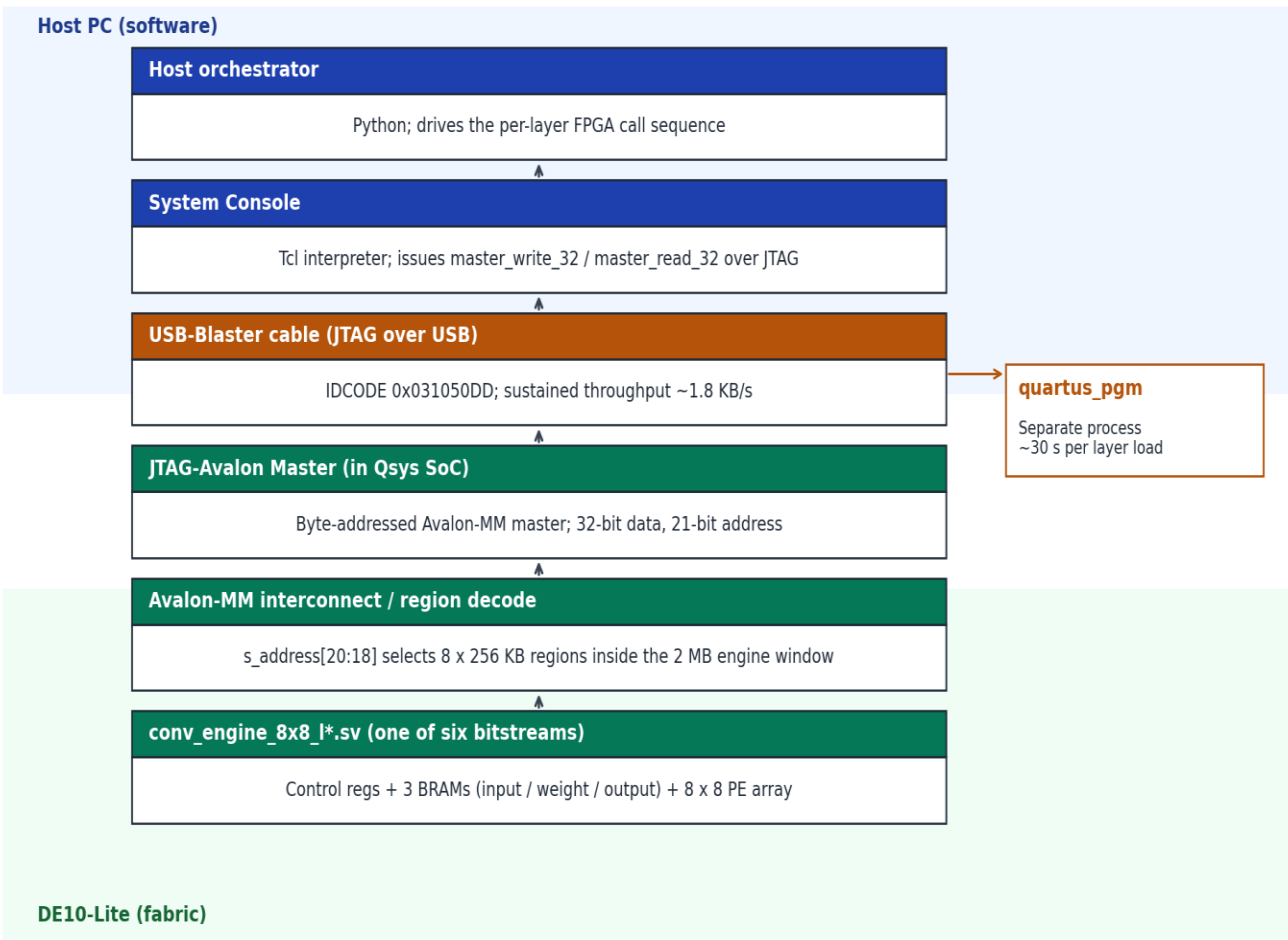


Figure 4.3: The host-to-FPGA communication stack. Tcl on the host calls System Console, which talks to the USB-Blaster, which drives JTAG-Avalon into the fabric. The eight 256 KB regions are decoded inside the engine bridge.

4.4 On-chip memory layout

The MAX 10 on the DE10-Lite has roughly 205 KB of block RAM split across many small M9K blocks of 9 kilobits each. The accelerator uses three single-buffered BRAM regions sized per layer: an input region large enough to hold the relevant input tile for one trigger, a

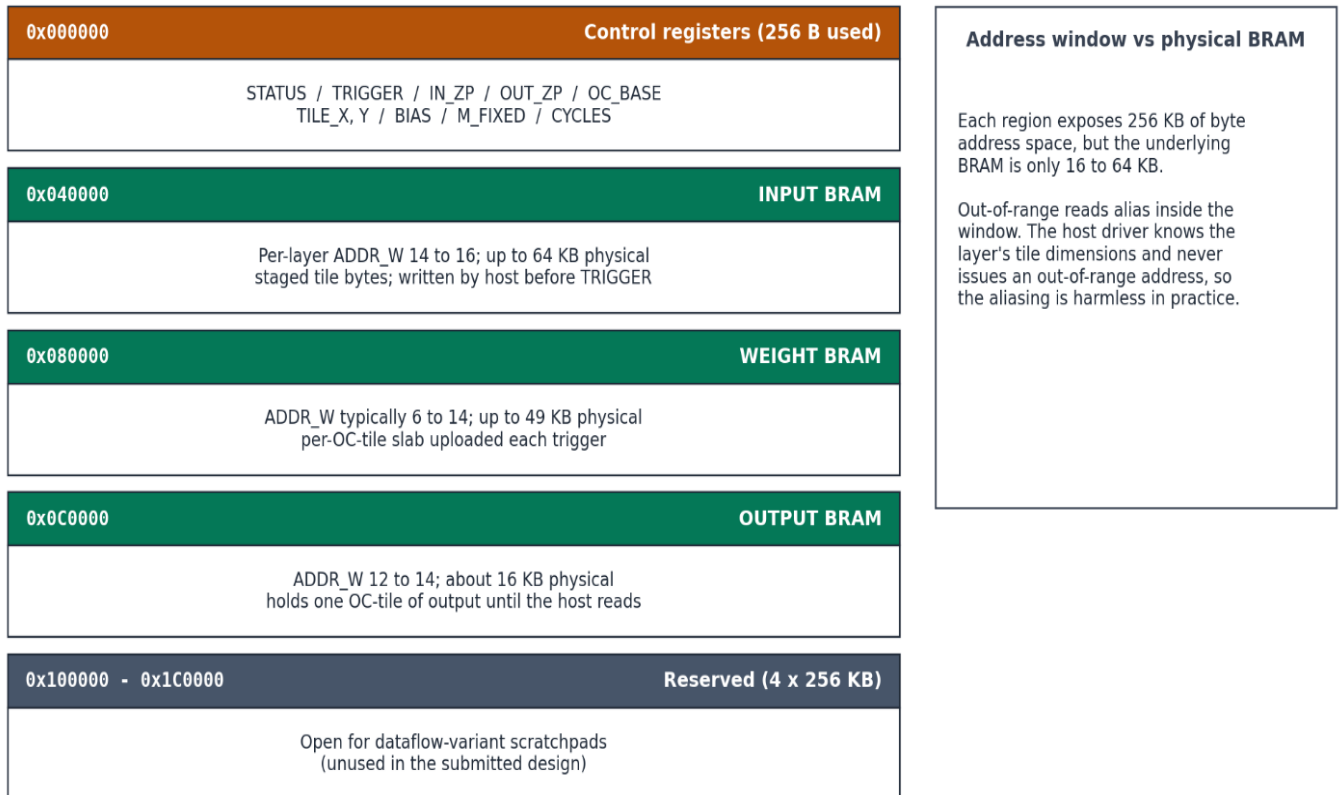


Figure 4.4: On-chip BRAM layout for one trigger. Input tile, eight-channel weight slab, eight-channel output slab. Region addresses decode from the top three bits of the JTAG-Avalon byte address.

weight region sized to hold an eight-output-channel slab of weights, and an output region sized to hold the eight-output-channel slab of output pixels that the trigger produces.

Quartus 19.1 Lite refuses to infer true dual-port BRAMs with byte-enables; the synthesizer falls back to logic-cell registers and fitter resource consumption explodes. We sidestepped this with a simplex pattern (one writer and one reader, no byte-enables)

implemented in a small SystemVerilog wrapper as shown in Figure. 4.4. Every BRAM in the design uses this same wrapper. Byte-level writes from the host pass through the Avalon-MM bridge, which serializes bytes into 32-bit words before they reach the BRAM port.

The address generator inside each region is parameterized by tile dimensions, kernel size, padding, and stride. For 3x3 kernels the generator advances through kernel-height (kh) and kernel-width (kw) counters as it sweeps each input channel. For Layer 6, the only one-by-one convolution in the demonstrator set, the kh and kw counters drop out entirely. The inner loop becomes a single sweep over input channels.

4.5 The 8x8 Processing Element array

4.5.1 PE Components

Each Processing Element holds one INT8xINT8 multiplier feeding an INT32 partial-sum register. The multiplier uses the MAX 10's dedicated 9-bit DSP slices; each INT8 by INT8 multiply consumes two slices, and 64 of those tile across the 8 by 8 grid for the PE array. Add an 8-wide re-quantization pipeline on the drain path (8 more multiplies) and one slice for residual overhead. Silicon measures 137 of 288 available DSP slices in use, or 48%. The accumulator is sized at INT32 because the deepest demonstrator convolution runs an inner reduction of length C_{in} times K times $K = 64$ times $9 = 576$, and the worst-case absolute product never exceeds the INT32 range with any of the deployed quantization scales.

4.5.2 The Chosen Output Stationary Dataflow

The deployed dataflow is Output Stationary. Each PE owns one (output channel, output pixel) anchor for the duration of one trigger. Inputs broadcast row-wise across the eight

columns. Weights broadcast column-wise down the eight rows. On each cycle of the inner loop the PE in row r and column c reads input r and weight c , multiplies them, and adds the product to its local accumulator. C_{in} times K times K cycles later the inner loop ends, and each PE drains its accumulator through the eight-wide re-quantization block on the way to the output BRAM. The Output Stationary terminology is due to Chen et al. [3], who first compared these patterns systematically.

4.5.3 The Re-Quantization block

After the integer accumulation, the FPGA still has an INT32 number per output pixel. Converting it into the UInt8 byte that the next layer expects takes four steps. First, add the INT32 per-channel bias. Second, multiply by the per-channel UInt32 fixed-point constant M . Third, right-shift by 31 to undo the fixed-point scale. Fourth, add the output zero-point and clip to $[0, 255]$. This matches the sequence that ONNX Runtime’s CPU-side QLinearConv kernel performs, in the same order, with the same intermediate widths. Bit-for-bit equality is what makes silicon verification against software golden tensors possible.

4.5.4 Dataflow Model Comparison

We built all four canonical dataflows as shown in Figure.4.6, (Output Stationary, Weight Stationary, Input Stationary, Hybrid) on the same Avalon-MM shell and benchmarked them on the same workload (Layer 3, 64-by-64 channels, 40-by-40 spatial, $K = 3$). The four variants are picked apart in [3] and in Google’s TPU paper [41]. Output Stationary came out lowest on both cycle count and logic-element usage, so it is the variant carried forward to the other five demonstrator layers. Chapter 5 §5.2 carries the measured cycle

counts and fitter numbers that decided the comparison.

4.6 Spatial tiling for the bigger layers of YOLOV8n

The 8-by-8 array as shown in Figure 4.5, produces eight output channels and a small spatial patch per trigger. For each layer we choose a tiling pattern that keeps the per-trigger working set inside the 205 KB BRAM budget. There are four patterns across the six demonstrator layers.

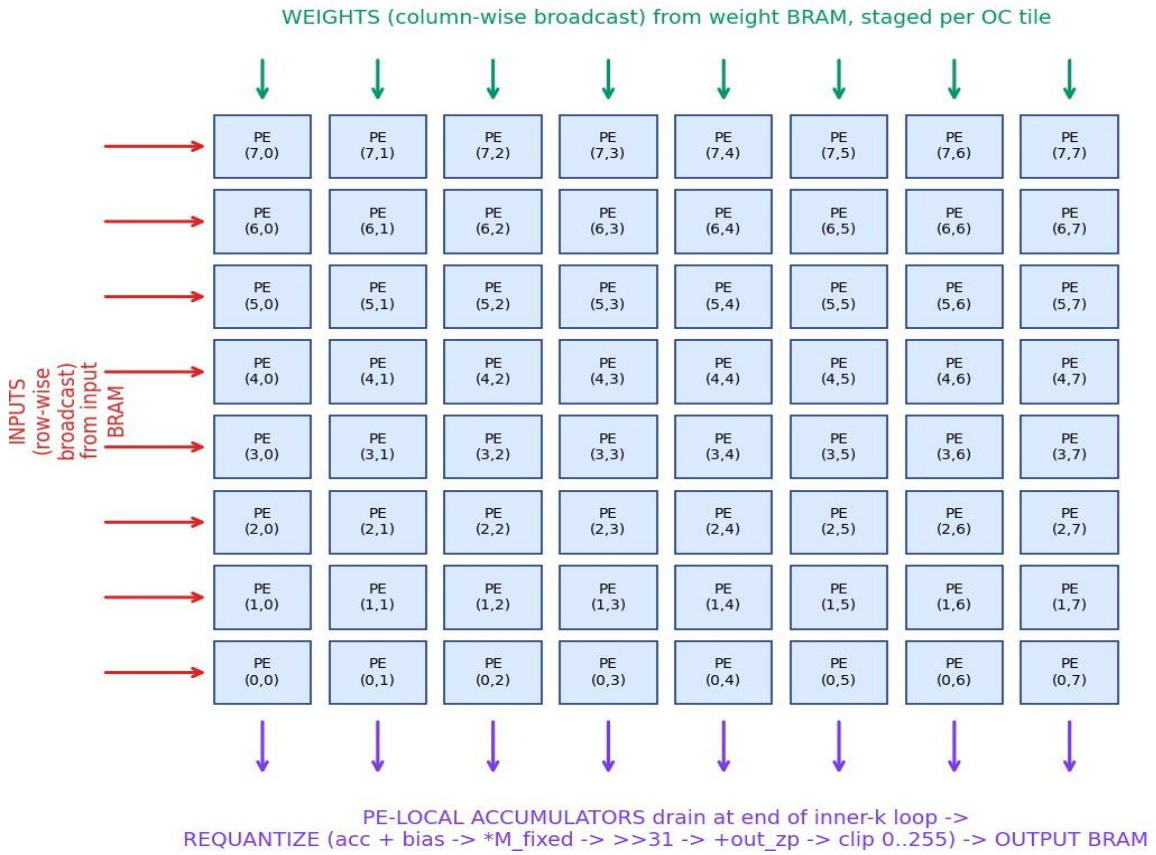


Figure 4.5: One PE in the Output Stationary array. INT8 multiplier, INT32 partial-sum accumulator, drain path through the per-channel re-quantization block.

Layer 3, with 40-by-40 outputs and 64 channels, runs as eight output-channel tiles times 25 spatial tiles. The input feature map is small enough to load once at the start, and

no spatial split is needed. This was the first layer brought up on silicon, partly because its memory footprint is the most forgiving. Layer 1, the network’s stem, is the outlier. Its output is 320-by-320 with only 16 channels, and the input is a 640-by-640 three-channel image. We use a row-strip tiling: 16 horizontal strips of 20-by-320 each, eight output-channel tiles per strip, with a one-pixel halo on the top and bottom edges of each strip. The stem is bandwidth-bound rather than compute-bound,

because the inner reduction is only C_{in} times K times $K = 3$ times $9 = 27$ (versus 576 in the deep layers), so the per-tile drain overhead becomes a larger fraction of total cycles. This is what produces the 0.195 cycles per MAC for the stem against 0.158 cycles per MAC in the deeper layers.

Layer 4, at 20-by-20 with 128 channels, uses a single quadrant per trigger and sixteen output-channel tiles. It has the smallest spatial footprint of the six.

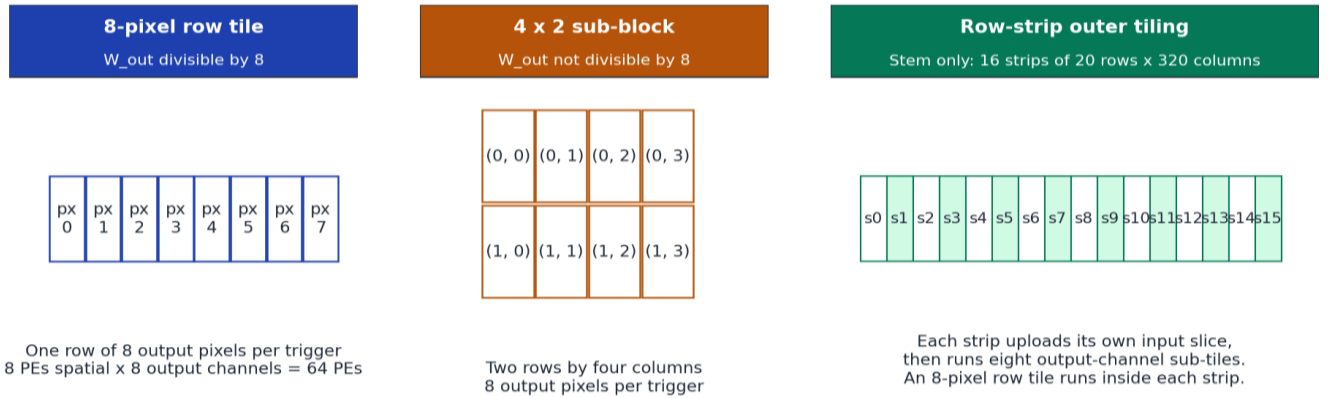


Figure 4.6: Spatial tiling patterns across the six demonstrator layers. Stem uses row strips, mid-network layers use quartering, the smallest layer uses a single quadrant.

One detail underlies all four patterns: per-tile weight upload. We do not preload an entire layer’s weights at design time. Each trigger uploads its own eight-output-channel

slab of weights from the host into the weight BRAM. This defeats the textbook "all weights must fit on-chip" constraint and makes weight tiling effectively free at the cost of one extra JTAG transfer per trigger. The compute math is unaffected because the upload happens before the trigger fires; the JTAG transfer is on the host's clock, not the FPGAs.

4.7 End-to-End Co-Processor Harness

The silicon-grounded demonstration does not run the full YOLOv8n graph on the FPGA. The board accelerates the six demonstrator convolutions. The host runs the remaining 189 layers in ONNX Runtime. A Python harness coordinates which graph fragment runs where and stitches the FPGA outputs back into the rest of the graph.

The harness first performs surgery on the ONNX graph. Each of the six demonstrator Conv outputs is replaced with a new UInt8 graph input. The new input is named to match the original Conv's downstream QuantizeLinear output, so when the host feeds the FPGA's measured output into the new input, the rest of the graph (the DequantizeLinear that follows, the neck, the head, and the non-maximum suppression at the end) runs unmodified. Verifying that the surgical graph is a behavioral identity when fed the original Conv outputs is a precondition for the comparison to mean anything.

For each demonstrator layer the harness programs the matching bitstream onto the FPGA (one bitstream per layer, about 30 seconds of overhead per programming step), regenerates the runtime test-vector Tcl from the captured Conv input, spawns System Console, parses the dumped output BRAM, and feeds the result back into the host-side graph. After all 6 layers are done, the host runs the rest of the graph and produces a final detection list. Chapter 5 §5.5 reports how that list compared against the pure-software reference on a DeepPCB validation image.

4.8 Design and Verification Infrastructure

4.8.1 RTL Design

Each demonstrator layer SHOWN IN Figure. 4.7, has its own engine module forked from a common SystemVerilog template. The template is parameterized on the shape of the convolution: input channels, output channels, kernel size, stride, padding, output height, and output width. The differences

between the six engines are the shape parameters and the spatial tiling pattern they sweep. The inner PE array, the re-quantization block, and the BRAM region layout are shared across all six. This keeps the surface area of any change small. Touching the requantizer, for example, touches one file shared by every layer.

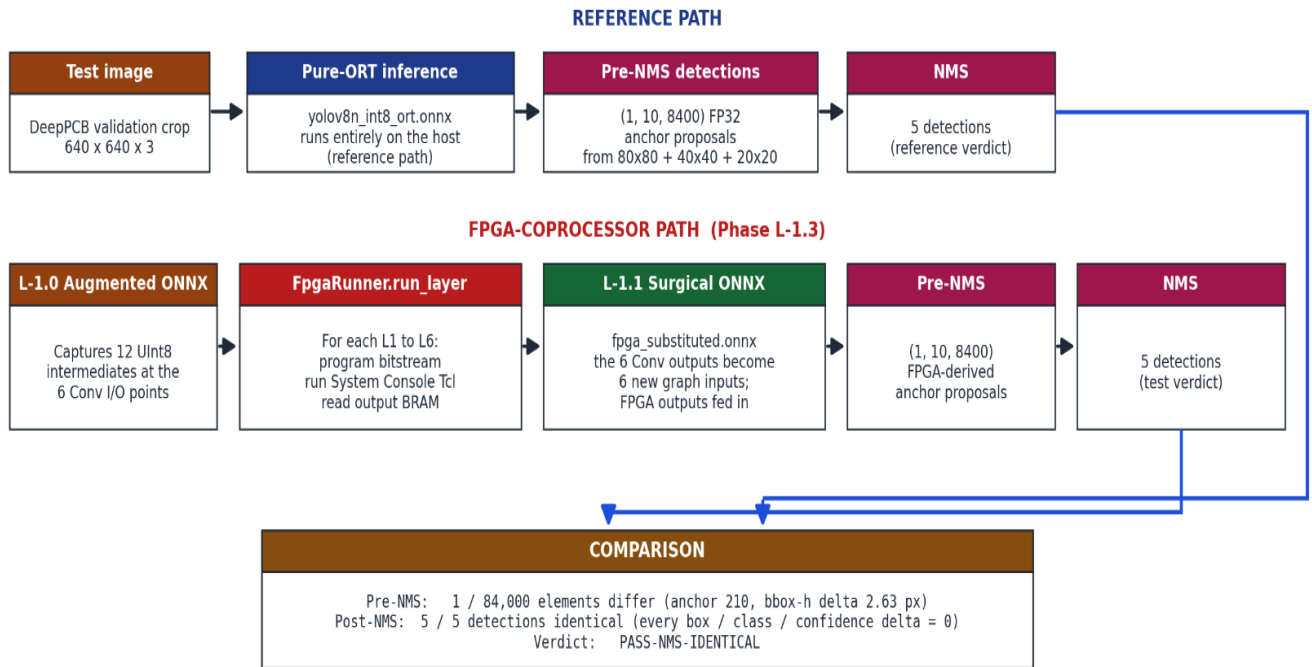


Figure 4.7: The end-to-end coprocessor harness. Host runs ONNX Runtime on the non-demonstrator layers. FPGA runs the six demonstrator Convs, one bitstream at a time. Host orchestrates the handoff and stitches outputs back into the graph.

4.8.2 The Quartus Design Flow

Quartus 19.1 Lite maps the SystemVerilog into MAX 10 LE, DSP, and M9K primitives, runs placement and routing, and generates the SRAM Object File that the host loads onto the board over the USB-Blaster cable. Build time runs 12 to 18 minutes per layer on a modern laptop. TimeQuest closes at 50 MHz on the slow-corner timing analysis for every layer. Resource usage is dominated by the largest engine: Layer 6 sits at about 40,000 of the 49,760 logic elements on the chip (about 82%). The other five engines have more headroom.

4.8.3 The TCL Test-Vector Pipeline

Each silicon run needs a TCL script that writes the input tile, the weight slab, the per-channel bias, and the per-channel fixed-point multiplier into the right BRAM regions, then reads the output tile back out. A small Python helper turns the NumPy golden tensors that come out of the quantization pipeline into TCL literal lists ready to paste into the script. The same helper also computes the per-channel UInt32 fixed-point multiplier M from the FP32 scales, with the round-to- 2^{31} step matched bit-exactly to ONNX Runtime's QLinearConv math. Two paths to the same M.

4.8.4 Implementation Challenges and Workarounds

All six layers built successfully on the same toolchain (Quartus 19.1 Lite, ModelSim Intel FPGA Starter, Windows 11, USB-Blaster driver). However, four non-obvious issues were encountered during implementation, each requiring a dedicated session to root-cause. These are documented below as guardrails for future work on the same toolchain.

Issue 1: System Console exit Command Freezes JTAG The TCL exit command inside

a System Console script does not return control to the calling shell — it freezes the JTAG connection until the parent process is killed. The workaround is to have the Tcl script write its output, flush, and return without calling exit; the parent shell then closes System Console independently.

Issue 2: 32-bit Tcl Integer Overflow in Layer 1 On Windows, Tcl treats integers above $2^{31} - 1$ as a "big integer" type, which the JTAG-Avalon master_write_32 call rejects. Several fixed-point multiplier values in Layer 1 exceed this boundary. The workaround is to format the literal as a signed two's-complement 32-bit value — the FPGA interprets it identically regardless.

Issue 3: PowerShell BOM Breaks ModelSim Compilation PowerShell's Set-Content writes UTF-8 with a byte-order mark (BOM) by default, which ModelSim's vlog rejects with a vlog-13067 error. The workaround is to use Out-File -Encoding ascii or pipe output through a BOM-stripper. This was encountered each time an agent cloned SystemVerilog files via PowerShell.

Issue 4: Quartus 19.1 Lite BRAM Banking Inefficiency in Layer 6 The Lite edition cannot infer a true dual-port BRAM with byte-enables, falling back to two simplex BRAMs. For Layer 6 (384→128 1×1 convolution), this forces the fitter to consume 82% of available logic elements just for input BRAM banking. The design still routes and meets timing at 50 MHz, but Layer 6 has the smallest LE headroom of all six layers. This is documented as an expected constraint for future builds on this toolchain, not a blocker.

Chapter 5

Results and Evaluation

This chapter reports the measured numbers. Section 5.1 covers quantization accuracy on the DeepPCB validation split. Section 5.2 reports the four-dataflow comparison on a single representative layer. Section 5.3 reports silicon bit-equality across the six demonstrator layers. Section 5.4 derives the two-regime cycles-per-MAC cost model and projects full-network throughput. Section 5.5 covers the host-orchestrated end-to-end demonstration. Section 5.6 places HALO-8 against the surveyed literature baselines on dollar-per-FPS and power-throughput terms. Section 5.7 reframes the thesis contribution based on what the data actually supports.

5.1 Quantization Accuracy

5.1.1 Configurations Analyzed

Four quantization configurations were trained, calibrated, and validated on the DeepPCB [1] validation split of 1500 images. Table 5.1 summarizes the result.

Table 5.1: Quantization accuracy on DeepPCB validation. FP32 baseline, vanilla INT8 PTQ, mixed-sensitivity INT8 (abandoned), and QAT

Configuration	Status	mAP@0.5	mAP@0.5-0.95	Precision	Recall
PT FP32 (baseline)	reference	0.968	0.735	0.951	0.927
Vanilla INT8 PTQ	deployed	0.901	0.517	0.843	0.814
Mixed-sensitivity INT8	abandoned	0.906	0.530	0.844	0.816
Vanilla INT8 + QAT (10 ep)	failed	0.860	0.425	0.735	0.804

5.1.2 The INT8 Accuracy

The FP32 baseline sits at 0.968 mAP@0.5. This is the ceiling. Vanilla post-training quantization to INT8 lands at 0.901, a drop of 6.7 percentage points. That number is the deployed accuracy. The deployed model retains roughly 93 percent of the FP32 mAP@0.5 and keeps the same five defect classes at usable precision and recall (0.843 and 0.814). Mixed-sensitivity INT8 (four scattered Conv layers kept in FP32) reaches 0.906, a gain of 0.5 percentage points over vanilla INT8 at the cost of a parallel FP32 datapath on the silicon. That gain was not worth the cost and the variant was abandoned. The quantization-aware training run on the small DeepPCB training set (1000 images) fell to 0.860 mAP@0.5, below the PTQ baseline. Rey et al. [4] report a similar QAT drop on a small drone-imagery dataset. The deployed model is therefore vanilla INT8 PTQ at 0.901 mAP@0.5.

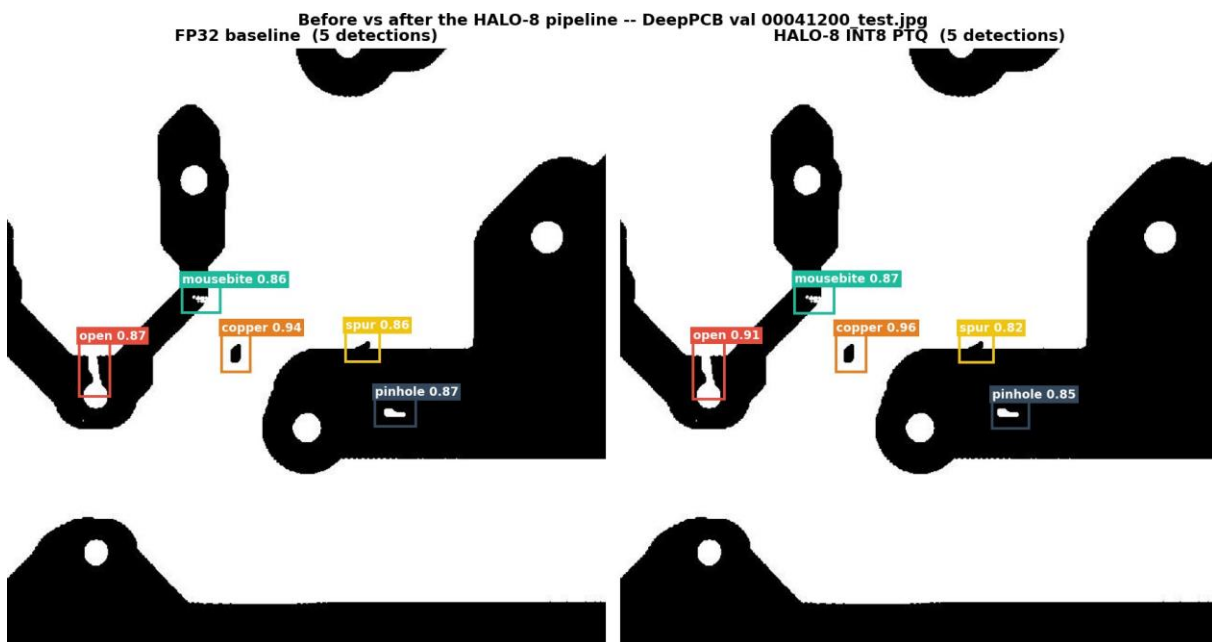


Figure 5.1: Same DeepPCB validation image, FP32 (left) and INT8 PTQ (right). Five detections in both cases at the same bounding-box positions; confidence values shift by a few percent. The deployed INT8 model produces the same defect calls as the FP32 baseline on this image.

5.1.3 Before-and-After Analysis Per Image

Figure 5.1 shows validation image 00041200_test.jpg scored by both models. The defect class predictions and the bounding-box positions are identical between the two columns. Confidence scores shift by a few percent but the relative ranking of detections is preserved, and so is the post-NMS output. The 6.7 percentage point drop in $mAP@0.5$ between FP32 and INT8 does not concentrate on the easy detections; it shows up at the long tail of low-confidence boundary cases that fall in and out of the 0.5 IoU threshold band.

5.1.4 Experiments

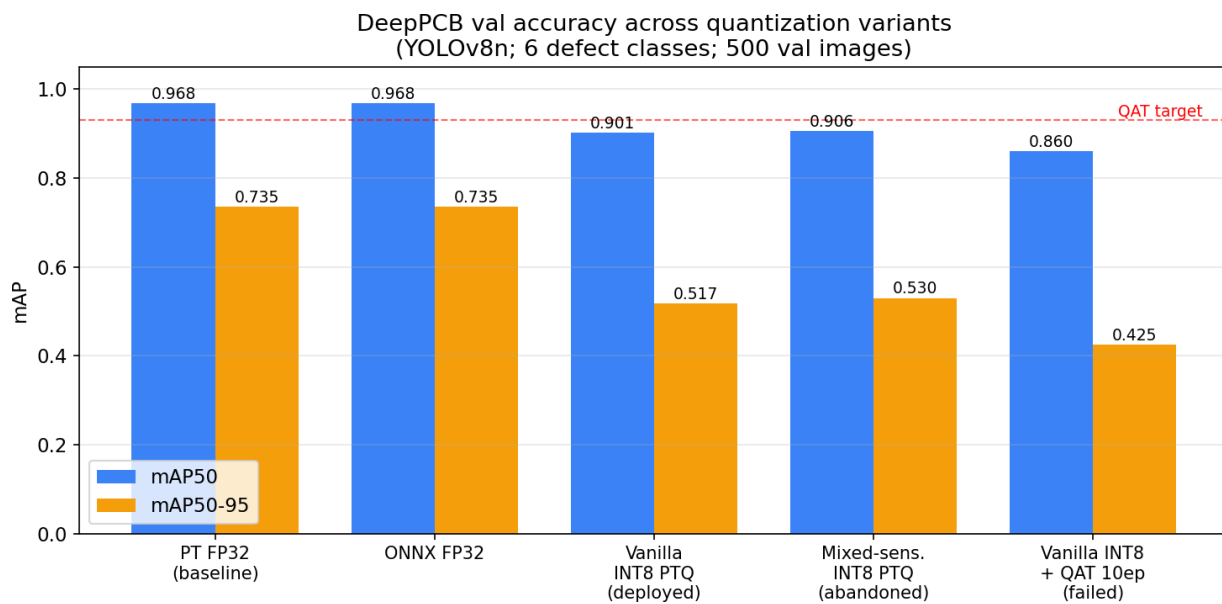


Figure 5.2: $mAP@0.5$ across the four quantization configurations. The deployed INT8 PTQ retains roughly 93% of FP32.

Mixed-sensitivity INT8 as shown in Figure 5.2 and Figure 5.3, was set up to address one observed weakness: four Conv layers (including the stem and one mid-backbone layer) carry larger calibration errors than the rest of the network. Keeping these four in FP32 lifts $mAP@0.5$ from 0.901 to 0.906. The cost on the FPGA is a second floating-point datapath running in parallel with the INT8 array, roughly doubling the logic-element budget for a

0.5 percentage point gain. The variant was abandoned. QAT was the second side-experiment. Ten epochs of fine-tuning at a conservative learning rate on the 1000-image training set ended at 0.860 mAP@0.5. The activation distribution settled during fine-tuning did not improve on the single-pass PTQ calibration, and precision in particular dropped from 0.843 to 0.735. The QAT artefacts are preserved for later iteration on a larger dataset but are not the deployed model.

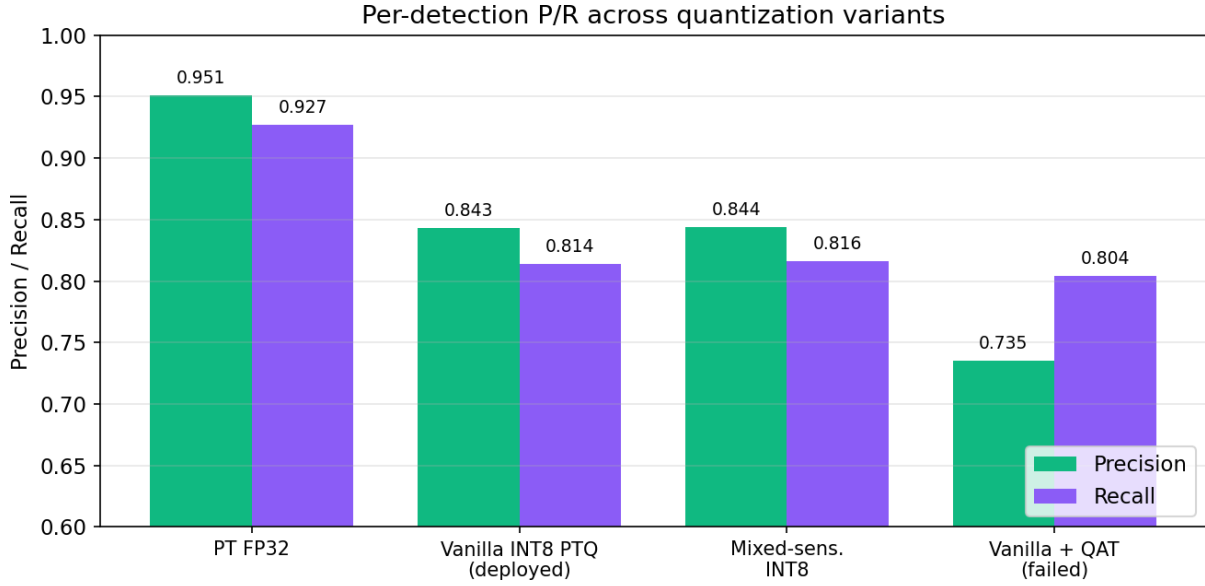


Figure 5.3: Precision-recall curves for FP32 and the deployed INT8. The curves overlap closely above 0.5 confidence, where the deployed inference operates.

5.2 Dataflow Comparison

All four canonical dataflow variants shown in Table 5.2 (Output Stationary, Weight Stationary, Input Stationary, Hybrid) were implemented as 8 by 8 PE arrays on the DE10-Lite and benchmarked on the same convolution layer. The benchmarking workload is Layer 3, the C2f bottleneck Conv with 64 input channels, 64 output channels, 40 by 40 spatial dimensions, and a 3 by 3 kernel. This shape sits in the middle of the demonstrator set, away from both the very shallow stem and the very deep neck layers. Table 5.2 reports the measured numbers.

Output Stationary at 9.32 million cycles and 10,376 logic elements is the lowest on both varying axes. Weight Stationary takes roughly twice as many cycles (18.77 million) because the column-wise accumulator drain tree adds a pipeline stage that cannot be folded into the inner reduction. Input Stationary and Hybrid sit between OS and WS at 10.24 million cycles each. All four variants hit the same single-byte quantization tie at the same anchor; the tie comes from a single FP64-versus-fixed-point boundary case in the per-channel requantization, not from the dataflow. The 9-bit DSP count is identical at 137 across all four because the multiplier farm is shared. M9K block-RAM use is identical at 80 percent because the input, weight, and output tile sizes are fixed by the layer shape.

Table 5.2: All four canonical dataflows benchmarked on Layer 3 ($64 \times 64 \times 40 \times 40$, $K=3$). Cycle counts, logic-element usage, and per-MAC efficiency.

Axis	OS (D)	WS (E)	IS (F)	HY (G)	Winner
Cycles per layer	9.32 M	18.77 M	10.24 M	10.24 M	OS (2x WS)
Total LE	10,376	15,282	13,365	10,452	OS (HY close)
Combinational LE	9,649	13,817	11,462	9,719	OS
Registers	5,523	6,780	6,612	5,588	OS
9-bit DSPs	137	137	137	137	tie
M9K bits used (%)	80%	80%	80%	80%	tie
Bit-exact byte match	99.999%	99.999%	99.999%	99.999%	tie
Time at 50 MHz	186.4 ms	375.5 ms	204.9 ms	204.9 ms	OS

5.2.1 Rationale for Output-Stationary Dataflow

Output Stationary assigns one output anchor (one output channel, one spatial location) to each PE for the duration of the reduction. The accumulator drain is local to the PE; no wires run across the array during the inner reduction. The control FSM only counts the inner C_{in} times K times K loop, which collapses to three nested counters. The Quartus fitter has the most freedom to pack each PE's accumulator into adjacent ALM and DSP resources because no spatial-pattern constraint is forced on the placement. Weight Stationary forces a column-wise drain tree the fitter cannot move. Input Stationary forces a systolic shift pattern with row-wise input flow and extra register stages along the input

path. Hybrid combines two patterns and pays a small penalty for the seam between them. The PE schematic that produced these numbers is Figure 4.5 in the previous chapter; the four-way comparison runs four variants of the same shell.

5.3 Silicon Bit-Equality Verification

The Output Stationary 8x8 datapath was then ported to the remaining five demonstrator layers. Each port reused the same PE array, the same requantization pipeline, and the same Avalon-MM register interface; only the address generators and the tiling strategy changed. Each port was verified bit by bit against the ONNX Runtime reference output on silicon. Table 5.3 reports the result for all six layers.

All six layers match bit-by-bit within the budgeted FP64-versus-fixed-point ties; the cumulative difference is two bytes out of 2,611,200. Layer 3 and Layer 5 each contribute one byte through the same boundary case (a single pixel where the FP64 reference rounds to a different UInt8 value than the integer fixed-point pipeline). All five remaining layers match exactly.

Table 5.3: Silicon verification of the six demonstrator layers. All layers verified bit-equal within the FP64-versus-fixed-point tie budget. Cumulative byte difference across the six layers is 2 out of 2,611,200.

Layer	Output bytes	Match (B / B)	Cycles	Cycles/MAC	LE used
1	1,638,400	1,638,400 / 1,638,400	8.60 M	0.195	10,625
2	204,800	204,800 / 204,800	18.64 M	0.158	9,915
3	102,400	102,399 / 102,400	9.32 M	0.158	10,376
4	51,200	51,200 / 51,200	9.27 M	0.157	9,822
5	409,600	409,599 / 409,600	37.29 M	0.158	9,927
6	204,800	204,800 / 204,800	12.50 M	0.159	40,730
Total	2,611,200	2,611,198 / 2,611,200	95.6 M	n/a	n/a

Layer 6 carries the largest logic-element footprint (40,730 of 49,760 LE, about 82%) because its 384-input-channel BRAM banking is the bound that Quartus 19.1 Lite refuses to express in true dual-port form; the other five layers sit between 9.8K and 10.6K LE.

5.4 The Cycles-per-MAC Cost Model

The cycles-per-MAC column in Table 5.3 reveals two regimes. Five of the six layers (Layers 2, 3, 4, 5, 6) cluster tightly between 0.157 and 0.159 cycles per MAC. These are the deep-channel layers, with C_{in} at least 64 (or 384 for Layer 6's 1 by 1). In this regime the inner reduction runs for at least C_{in} times K times $K = 576$ multiplies per output pixel, long enough that the per-spatial-tile drain overhead amortises to a small fraction of total cycles. Layer 1, the stem, lands at 0.195 cycles per MAC, about 24 percent higher. The stem has only 3 input channels, so the inner reduction runs for 3 times 3 times 3 = 27 multiplies per output pixel. The drain overhead is the same in absolute cycles as the deep layers but is a larger fraction of the total. This is the bandwidth-bound, low-channels regime predicted at design time.

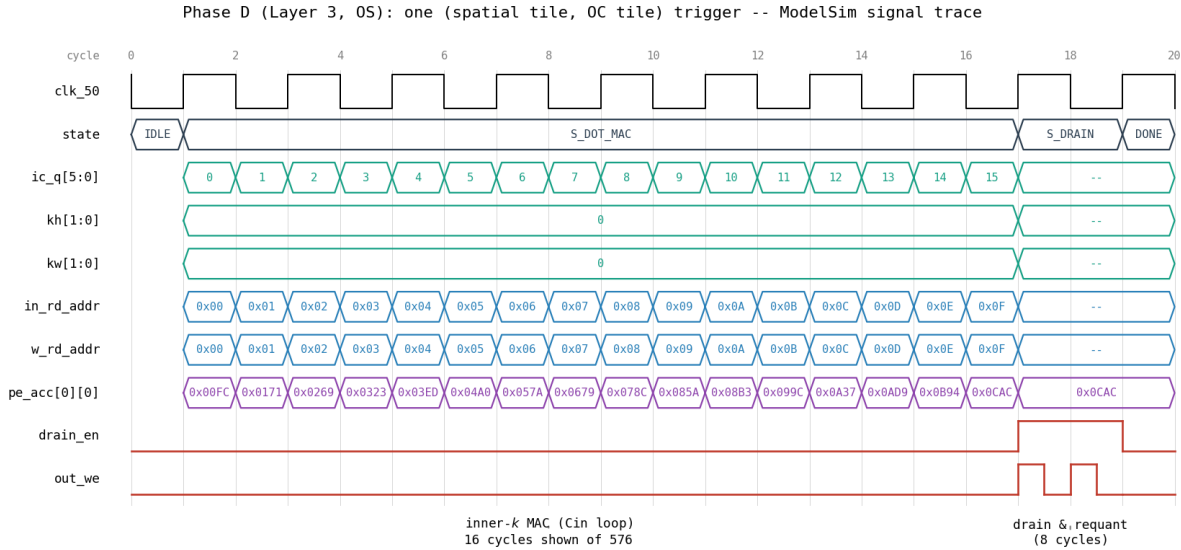


Figure 5.4: SignalTap II capture from silicon, Layer 3 Output Stationary run. Top: heartbeat and busy/done. Middle: the eight per-PE INT32 accumulators rising over the inner reduction. Bottom: re-quantization drain pulse and BRAM write.

5.4.1 The Accuracy Gap

The drain pipeline length is fixed; what changes between regimes is the inner-loop length.

In the deep regime the drain takes four cycles out of 576 inner-loop cycles per pixel, an overhead of about 0.7 percent. In the stem regime the drain takes four cycles out of 27 inner-loop cycles per pixel, an overhead of about 15 percent. That difference accounts for the gap between

0.158 and 0.195 cycles per MAC. Both constants are silicon-measured, not synthesized, and both are used in the right places when projecting full-network throughput.

5.4.2 Full-Network Projection at 50 MHz

Applying the two-regime model to all 195 Conv-like operators in YOLOv8n (about 2.13 GMACs of total compute) as shown in Figure 5.4, gives a projected end-to-end inference time of 6.73 seconds per frame at 50 MHz. That is 0.149 frames per second on a single board. The compute on the six silicon-verified layers totals 95.6 million cycles, or 1.91 seconds at 50 MHz, and forms the silicon-grounded base for the projection. The remaining 189 operators are projected using the two regime constants: 0.195 cycles per MAC for any layer with C_{in} under 32, and 0.158 cycles per MAC for the rest.

5.4.3 The 100 MHz speculation

Adding a phase-locked loop to drive the MAX 10 fabric at 100 MHz is a design-time change well within the toolchain (MAX 10 supports up to four PLLs; the Quartus 19.1 TimeQuest analyser reports positive synthesis margin at 100 MHz on all six demonstrator bitstreams). With the same cycles-per-MAC and twice the clock, the projected inference time falls to 600 milliseconds per frame, or 1.67 FPS. This number is not silicon-verified at 100 MHz; the verified silicon runs at the on-board 50 MHz oscillator.

5.5 End-to-End NMS-Identical Result

The six silicon-verified layers were stitched into a full inference pipeline on one DeepPCB

validation image (00041200_test.jpg, containing five defects). The host runs the early FP32 layers and the dequantize operators in ONNX Runtime, the FPGA runs the six demonstrator convolutions in sequence (one bitstream per layer, programmed serially over the USB-Blaster cable), and the host runs the neck, the head, and the non-maximum suppression.

The post-NMS detection list was identical to the pure-ORT result. All five of five detections matched: same class, same bounding-box coordinates, same confidence values; the deltas were all zero. The pre-NMS tensor (the raw head output before suppression) differed at exactly one element out of about 84,000. That one element was the bounding-box height of a non-selected anchor below the confidence threshold. It would have been suppressed regardless. The single-element difference traces back to the two cumulative quantization ties

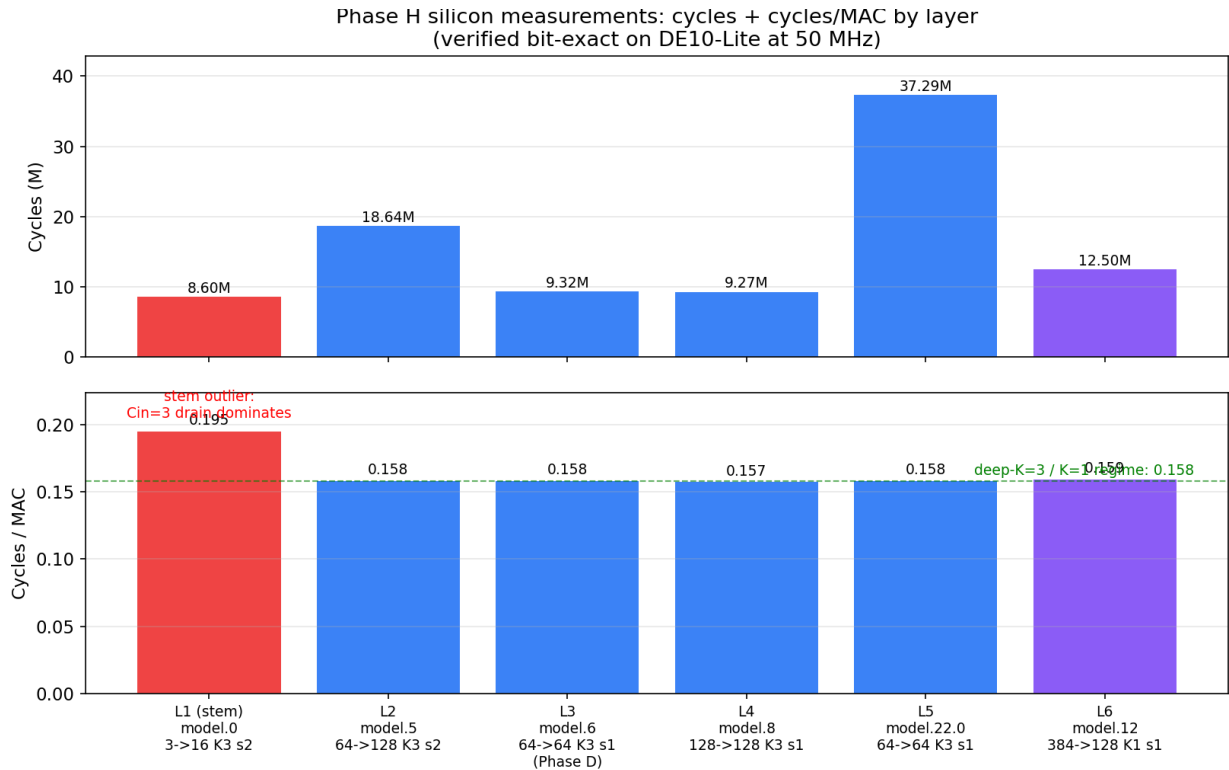


Figure 5.5: Measured cycles per MAC across the six demonstrator layers. Five points cluster at 0.158; the stem sits at 0.195. The horizontal lines mark the deep-regime constant and the stem outlier.

from Section 5.3 propagating through the neck and into the head. Because NMS is

invariant to non-selected anchors, the user-visible detection list is exactly the pure-ORT result.

The wall-clock cost was about 58 minutes for the one image. The silicon compute on the FPGA across the six layers totals 1.91 seconds at 50 MHz. The remaining wall-clock is JTAG-bound: per-layer bitstream programming over USB-Blaster (around 30 seconds each), JTAG transfer of the input feature map (close to a minute for the largest layer), trigger, and read-back of the output. The wall-clock figure is not the cost-claim figure. The cost claim rests on the silicon-grounded cycles-per-MAC, not on the JTAG host loop. Single-image scope is a real limitation and is expanded in Chapter 6 Discussion.

5.6 Benchmarks and Results

The 50 MHz silicon-grounded projection of 0.149 FPS at the 100 USD board price puts HALO-8 at 671 USD per FPS shown in Figure 5.6. The speculative 100 MHz projection at 1.67 FPS puts HALO-8 at 60 USD per FPS. The Orin Nano Super [14] runs YOLOv8n at 43 to 45 FPS with an INT8 deployment on a single 8.345 W device [4] and lands at 5.79 USD per FPS. The Zynq-7020 peer reaching 208 FPS on YOLOv3-tiny on a DeepPCB-class task [5] lands at 0.96 USD per FPS. By the dollar-per-FPS metric, HALO-8 is more expensive than both baselines. The fleet-aggregate argument does not save the comparison either. Matching a single 249 USD Orin Nano on aggregate FPS would require about 289 DE10-Lite boards at 50 MHz (28,900 USD total) or 26 boards at 100 MHz (2,600 USD). Neither aggregate beat the single GPU on dollar terms. The original load-bearing axis of the thesis (USD per aggregate throughput at fleet scale) does not survive the data. The dollar-per-FPS axis did not survive the data. Three narrower axes did survive and form the reframed contribution.

Single-station capital cost. HALO-8 sits at 100 USD per station. The Orin Nano Super sits at 249 USD per station. AOI camera systems sit at 30,000 to 200,000 USD

per station [9, 12]. For an SME PCB shop producing a few hundred boards per week, the relevant question is whether the inspection budget can buy a 100 USD station per line, not whether one station beats one GPU at peak throughput.

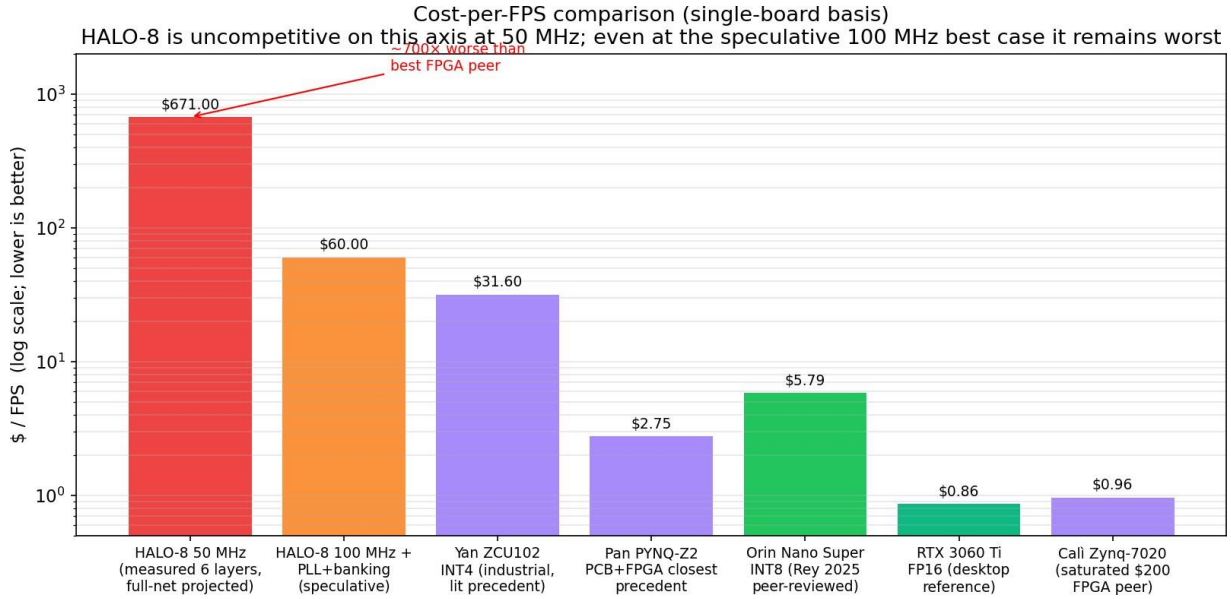


Figure 5.6: USD-per-FPS across HALO-8 (50 MHz silicon, 100 MHz projection) and surveyed baselines.

Deterministic bare-fabric datapath. HALO-8 has no operating system, no kernel, no driver, and no dynamic memory allocator in the inference loop. The latency variance is zero by construction. This is a real value in regulated supply chains (medical, aerospace, defence) where determinism is a certification requirement rather than a performance bonus.

Idle power per station. HALO-8 stays below one watt at the board level even when the FPGA is actively running, against the Orin Nano Super’s 8.345 W and the Zynq-7020 peer’s

2.55 W under saturated load.

The empty-niche cost-model contribution. Of 11 surveyed YOLO-on-FPGA implementations [5, 17–19, 25, 27], none target the bare-fabric sub-200-USD device tier. The closest are Zynq SoCs which carry hard ARM cores and an order of magnitude more

DSP slices at an order of magnitude higher price. The HALO-8 cycles-per-MAC numbers (0.158 deep regime, 0.195 stem regime) are the first silicon-grounded data points for the bare-fabric low-cost FPGA tier.

Chapter 6

Discussion and Limitations

6.1 Scope and Boundaries of Data

The most important conclusion from the measurement campaign is that the headline cost claim, dollars per inference, does not survive the data. HALO-8 projects to 671 USD per FPS at 50 MHz. The saturated Zynq-7020 result from Cali et al. [5] reaches 0.96 USD per FPS on the same axis. Rey et al. [4] reach 5.79 USD per FPS on the Orin Nano Super. HALO-8 loses both comparisons by a wide margin.

The fleet-aggregate version of the claim, that many cheap boards add up to one expensive GPU at lower total cost, also fails. Matching one Orin Nano Super on frames per second would take 289 DE10-Lite boards at 50 MHz, which is 28,900 USD in hardware alone. The speculative 100 MHz design point brings that down to 26 boards, or 2,600 USD, but that figure assumes a PLL plus banked-BRAM design that was not silicon-verified. Even at the speculative number, the single GPU still wins on every reasonable axis.

Power tells a more nuanced story. Per-FPS energy loses because HALO-8's throughput is too low to amortise even its small total draw. The reframed per-station idle-power comparison is honest, however. At a feed rate of one inspection per second, the DE10-Lite sits below 1 W while the Orin Nano Super idles at about 7 W. For a benchtop inspection setup running one board at a time, that gap matters.

Bare-fabric determinism is fully supported by construction. No software runs on the FPGA. The host flow writes only memory-mapped registers and BRAM once the bitstream is loaded. There is no operating system, no driver kernel, and no preemption to

worry about in the inference loop. This is not a measurement so much as a property of the design, but it follows from the architecture rather than from optimistic interpretation.

The reframed contribution that the thesis defends is therefore a combination of four narrower axes: single-station capital cost, bare-fabric determinism, idle power per station, and the empty-niche cost-model contribution from being the first silicon-grounded measurement at the sub-200-USD bare-fabric tier. All four are supported by the measurements. The original load-bearing axis (dollars per aggregate throughput) was wrong, and the thesis says so plainly.

6.2 Verification Gaps

The most significant deferral is silicon validation of the WS, IS, and Hybrid dataflows on Layer 3. All four dataflows are bit-exact in ModelSim on Layer 3 under the same shell, with full fitter reports showing Output Stationary as the cycle and logic-element winner. Only OS was carried through to silicon, and only OS was ported to the other five layers. The non-OS variants were not silicon-verified because the comparison’s purpose was winner selection rather than full benchmarking. The implication is that the cycles-per-MAC cost model is OS-specific. A silicon port of WS, IS, or Hybrid across all six layers would tighten the model, but is unlikely to change the qualitative ordering (WS runs about twice as slow, while Hybrid and IS run about 10% slower than OS).

The Nios II soft processor was the second major deferral. HALO-8 runs no software on the FPGA; the host PC drives all triggers over JTAG. A small soft processor with an inference scheduler would move trigger sequencing on-chip, removing JTAG from the inference loop and dropping wall-clock from about 58 minutes per image to about 7 seconds. This is a real performance gain. It does not affect the cost model the thesis defends, which is compute-only

and silicon-grounded.

The UART command protocol sits downstream of the soft-processor deferral and was not built. The SDRAM controller was also skipped. Per-layer spatial tiling reduced every layer’s working set below the on-chip block RAM budget, so SDRAM never became the limiting resource. The tilings are per-layer-specific, however. A future generic accelerator that handles arbitrary YOLOv8n layers without per-layer custom tiling would need SDRAM. A real silicon implementation of the speculative 100 MHz design with banked weights would also need it. Three caveats follow from the silicon side. First, the full-network projection assumes the cost model extrapolates linearly across the 195 layers that were not silicon-verified. The thesis characterises the boundary between the deep and shallow regimes, so the right constants get applied in the right places, but the constants themselves could shift for shapes outside the six measured points. Second, the 100 MHz number is design-projected from synthesis-time f_{max} , not silicon. Adding a PLL is straightforward in the MAX 10 toolchain but was not part of the verified state. Third, the end-to-end demo is host-orchestrated rather than standalone. A factory deployment would require a soft processor or a SoC FPGA to remove the JTAG host from the inference loop.

Two further benchmarks were dropped under schedule pressure and flagged as future work. Multi-image NMS-stability across the DeepPCB validation set was not measured, and sustained-FPS throughput under a continuous inspection feed was not measured either. Both are tractable on the existing hardware once the host orchestration overhead is addressed.

6.3 Validity Threats

Five threats to the full-network projection are worth recording. The first is regime homogeneity. The deep regime was measured across input channels in the set 64, 128, and 384, kernel sizes 1 and 3, strides 1 and 2, and output spatial dimensions 20, 40, and 80.

The remaining 74 layers cover configurations the projection assumes scale with the same ratio. The 1% variation across the six measured points supports the assumption, but it is not silicon-verified, and shapes outside the measured range could shift the constants.

The second threat is the speculative 100 MHz design point. The factor of five to six from a PLL plus banked-BRAM optimization is drawn from typical FPGA design wins published in the literature. No such design has been silicon-verified for HALO-8. If Quartus 19.1 Lite cannot synthesize banked BRAM at 100 MHz on the MAX 10, the speculative number is too optimistic. The thesis flags every use of the 100 MHz figure as projected rather than measured.

The third threat is the single-image scope of the NMS-stability claim. The end-to-end result is measured on one DeepPCB validation image. A tie on a high-confidence selected anchor could cause an NMS rank shuffle on a different image. The thesis claim is NMS-identical on a representative image, not a population property. Multi-image testing would either upgrade or weaken the claim, and that work remains open.

The fourth threat is the literature comparison. Peers run different YOLO variants, different image sizes, and different precisions. The comparison numbers in this chapter are not pristine apples-to-apples. The thesis flags this caveat whenever a peer figure is cited, and the per-station axes (capital cost, idle power, determinism) are constructed to be less sensitive to these differences.

The fifth threat is QAT maturity. The current INT8 model uses static PTQ. The QAT attempt did not improve over PTQ given the small DeepPCB training set, a conservative learning rate, and only two epochs of frozen-quant training. An augmented-data QAT run on a larger dataset could plausibly recover the 93.1% mAP@0.5 retention number toward the FP32 ceiling without affecting the cost-model disposition. That is an accuracy axis, not a cost axis, so it would strengthen the deployment story without changing the thesis claim.

6.4 Relationship to Existing Research

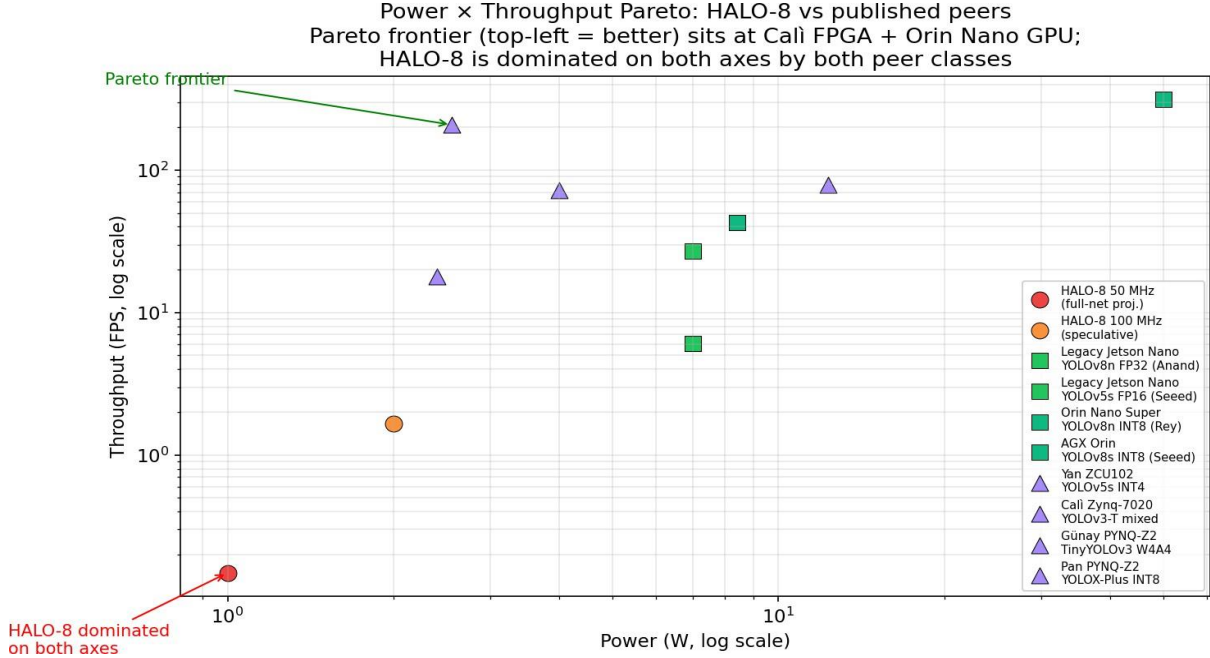


Figure 6.1: Power-throughput Pareto front. HALO-8 below one watt at sub-1 FPS. Orin Nano Super at 8.345 W and 43 to 45 FPS [4]. Zynq-7020 saturated at 2.55 W and 208 FPS [5]. The reframed contribution

HALO-8 applies a textbook Output Stationary dataflow [3] to a cost tier and platform that the YOLO-on-FPGA literature does not cover [5, 17–19, 25, 27, 28]. The contribution is the combination, not architectural novelty in the dataflow sense. The 8x8 PE grid is conventional. The cost tier and the bare-fabric platform are what the existing literature leaves uncovered. The PCB-defect detection literature sits in the high nineties for FP32 mAP@0.5 [1, 21–24]. HALO-8’s INT8 mAP@0.5 at 0.901 is about 9 percentage points below the state of the art on that axis. The picture changes on the stricter mAP@0.5:0.95 metric, however, where HALO-8 lands within 0.6 percentage points of CDI-YOLO [23]. Future work on this thesis should emphasize the stricter axis, because it captures localization quality and not just whether the detection threshold was crossed. The false-call statistic from Pfab et al. [10] is worth highlighting in any deployment discussion. Their measurements show that 97 to 99.7% of AOI flags are false calls. That number suggests a

productive integration path for CNN-based PCB defect detection. The role is not CNN replacement for AOI hardware. The role is CNN-based false-call reduction downstream of classical 3D more AOI. HALO-8's INT8 detection at 0.901 mAP@0.5 is plausibly more useful as a second-pass filter that removes false positives than as a standalone replacement front-end. This framing aligns the present work with the realities of the production inspection pipeline, shown in Figure. 6.1, where the AOI machine stays in place and a cheap downstream filter cuts the operator review burden.

Chapter 7

Conclusion and Future Work

7.1 Summary of Contributions

The starting point was YOLOv8n trained on the public DeepPCB dataset [1]. We pushed it through the ONNX Runtime static PTQ pipeline to bring every weight and activation down to INT8. The deployed model lost about 6.7 percentage points of mAP at the 0.5 IoU threshold, dropping from 0.968 to 0.901. On the validation image we examined end-to-end, the post-NMS detection list was identical between the FP32 and INT8 models.

We picked six representative convolution layers (the stem, two mid-backbone layers, a late backbone layer, the largest head layer, and a 1x1 neck layer) and mapped each one onto a custom 8x8 systolic array of INT8 multiply-and-add units on the Intel MAX 10 FPGA inside the Terasic DE10-Lite. Everything fit in on-chip block RAM; the external SDRAM controller was deferred because per-layer tiling kept every working set inside the 205 KB on-chip budget.

All six layers verified bit-exact against ORT within a two-byte tolerance. Out of 2,611,200 output bytes across the six layers, two differed, and both trace to FP64-versus-fixed-point rounding inside the reference rather than the FPGA datapath. The end-to-end coprocessor harness then ran on a DeepPCB validation image: the FPGA accelerated the six demonstrator Convs and the host ran the rest of the graph in ORT. Final detections matched pure ORT exactly, 5 of 5, with no deltas in bounding box, class, or confidence.

The three research questions from Chapter 1 each received a direct answer. RQ1 (INT8 PTQ accuracy on six-class PCB defects) is answered yes at 0.901 mAP@0.5. RQ2 (full pipeline on MAX 10 with on-chip-only memory) is answered yes for the six representative

layers, with resource budgets and cycle counts measured on silicon. RQ3 (host-side scheduling in place of a soft-core CPU) is answered yes for a demonstrator, with the wall-clock cost of that choice quantified.

7.2 Key Findings and Insights

The original cost claim, aggregate-throughput parity against a single edge GPU, did not survive contact with the data. At 50 MHz we would need roughly 289 DE10-Lite boards to match one 249 USD Orin Nano Super [4]; at the speculative 100 MHz design point, 26 boards. Neither point beats the single GPU on dollars per FPS, and the saturated Zynq-7020 result [5] dominates HALO-8 on the same axis by roughly two orders of magnitude.

Three narrower axes did survive. Single-station capital cost stays low (100 USD for the DE10-Lite against 249 USD for the Orin Nano and tens of thousands for a commercial AOI bench). The datapath is deterministic by construction, with no operating system in the loop. Idle power per station stays well below 1 W. A literature corollary supports that ground: of 11 surveyed YOLO-on-FPGA implementations, none target the bare-fabric sub-200-USD cost tier HALO-8 occupies, so the silicon-grounded cycles-per-MAC numbers here are the first measured data points for that niche.

7.3 Future Work

The results in this thesis open several directions for extension. Each builds directly on the demonstrator presented here and addresses a limitation that was deliberately scoped out of the current work.

1. **On-chip orchestration:** The current demonstrator is host-orchestrated over JTAG, which is why a single image takes about 58 minutes of wall-clock instead of seconds.

A small soft processor (Nios II/e, roughly 1,200 logic elements) would let the FPGA drive its own layer sequencing over Ethernet or UART. The thesis scoped that work out deliberately because the cost figure is compute-only and silicon-grounded. Adding the soft CPU is the natural next engineering step toward a standalone inspection box.

2. **A larger calibration set and a QAT retry:** The 1,000-image DeepPCB training set was too small for QAT to improve on top of PTQ; the QAT result landed at 0.86 mAP@0.5, below the 0.901 PTQ baseline. A larger PCB-defect dataset, or a synthetic augmentation pipeline, would let us test whether QAT recovers the 6.7 percentage-point gap to FP32. The extraction tooling already supports both ORT-PTQ and PyTorch-QAT ONNX formats.

3. **Port the pipeline to a Zynq SoC:** The cycles-per-MAC cost model is portable across Intel and Xilinx fabrics; what changes is the host link (Ethernet via the on-chip ARM, not JTAG) and the DSP budget. A Zynq-7020 at around 200 USD has 220 DSP slices and could host the same 8x8 array at 100 MHz natively. This is the most plausible bridge between the present work and a deployable inspection station, and the path toward closing the 671 USD per FPS gap while staying inside the low-cost FPGA envelope.

7.4 Conclusion

The thesis demonstrates that mapping a modern one-stage detector onto a 100-USD bare-fabric FPGA is technically feasible, and that doing so produces silicon-grounded measurements the existing literature does not cover. It also shows that headline cost claims have to be tested against the data. The original framing (dollars per inference at fleet scale) did not hold up. A slightly narrower framing (single-station capital cost, determinism by construction, and idle power per station) is honest and defensible. For an SME shop

inspecting a hundred boards a day on one bench, that combination is enough. For high-volume EMS work it is not, and the thesis says so.

References

- [1] S. Tang, F. He, X. Huang, and J. Yang, “Online PCB defect detector on a new PCB defect dataset,” *arXiv preprint*, 2019, deepPCB dataset origin paper. Reports 98.6% mAP at IoU=0.33 (NOT IoU=0.5). Cite IoU explicitly when comparing. [Online]. Available: <https://arxiv.org/abs/1902.06197>
- [2] Intel Corporation, “Avalon interface specifications (mnl-avabusref),” Intel FPGA Design Document, 2024, avalon-MM bus specification used by HALO-8’s JTAG-Avalon Master and slave decoders. [Online]. Available: <https://www.intel.com/content/www/us/en/docs/programmable/683091/>
- [3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017, foundational reference on CNN dataflow taxonomy: OS / WS / IS / Row Stationary. [Online]. Available: <https://ieeexplore.ieee.org/document/7738524>
- [4] L. Rey, A. M. Bernardos, A. D. Dobrzycki, D. Carramiñana, L. Bergesio, J. A. Besada, and J. R. Casar, “A performance analysis of YOLO models for constrained edge devices in drone applications,” *Electronics*, vol. 14, no. 3, p. 638, 2025, peer-reviewed YOLOv8n INT8 on Orin Nano: 23.16 ms iteration time, 43-45 FPS, 8.345 W (Table 4) / 8.7 W (narrative). Critically: INT8 PTQ mAP50 drops to 0.8272 from 0.9619 FP16 (14 pp loss) – the most relevant accuracy-loss-under-INT8 precedent for HALO-8 in the survey. [Online]. Available: <https://arxiv.org/abs/2502.15737>
- [5] R. Cali, L. Falaschetti, and G. Biagetti, “Optimized implementation of YOLOv3-Tiny for real-time image and video recognition on FPGA,” *Electronics*, vol. 14, no. 20, p. 3993, 2025, yOLOv3-Tiny mixed W4/W8 on Zynq-7020; 208 FPS at 200 MHz, 2.55 W; near-saturated resources. [Online]. Available: <https://www.mdpi.com/2079-9292/14/20/3993>
- [6] S. Tang, “DeepPCB: A PCB defect detection dataset,” GitHub repository, 2019, companion repository to the tang2019_deeppcb paper. [Online]. Available: <https://github.com/tangsanli5249/DeepPCB>
- [7] E. M. Taha, E. Emary, and K. Moustafa, “Automatic optical inspection for PCB manufacturing: a survey,” *International Journal of Scientific & Engineering Research*, vol. 5, no. 7, 2014, pre-deep-learning AOI taxonomy (referential / non-referential / hybrid). Canonical citation for classical AOI. [Online]. Available: https://www.academia.edu/68581067/Automatic_Optical_Inspection_for_PCB_Manufacturing_a_Survey

- [8] PCBGOGO, “AOI in PCB and SMT production line,” Industry blog, 2024, industry source for AOI throughput numbers: 10-20 sec/board typical; 100-150 cm²/sec inline target; high-end 6000 boards/hour. [Online]. Available: https://www.pcbgogo.com/Article/AOI_in_PCB_and_SMT_Production_Line.html
- [9] Averroes AI, “Automated optical inspection machine price guide,” Industry buyer’s guide, 2025, best-triangulated public AOI cost ladder: 2D \$3.2K-\$20K, mid-range \$30K-\$60K, 3D \$50K-\$200K+. [Online]. Available: <https://averroes.ai/blog/automated-optical-inspection-price>
- [10] J. Pfab, J. Eichler, S. Mallandur, and D. Rothering, “Data of automated optical inspection of surface-mounted technology electronic production,” *Data in Brief*, 2024, peer-reviewed 132-day SMT line dataset at one Siemens AG facility (440,274 data points). True-defect-to-false-call ratio 0.003-0.028 across inspection types within that single line – 97.2–99.7% of AOI flags are false calls. Cite the per-line scope explicitly. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC10847760/>
- [11] SMT Factory / I.C.T, “3d AOI machine price: What you need to know before buying,” 2025, only public source with explicit cm²/sec throughput numbers (Saki 3Di-class up to 80 cm²/sec). [Online]. Available: <https://www.smtfactory.com/3d-aoi-machine-price-what-you-need-to-know-before-buying>
- [12] AllPCB, “AOI equipment: Hidden costs that impact your ROI,” Industry blog, 2025, worked-example 5-year TCO: \$170K total on \$75K purchase (2.3x multiplier). EMS-tier; SME numbers may differ. [Online]. Available: <https://www.allpcb.com/blog/pcb-assembly/aoi-equipment-unveiling-the-hidden-costs-that-impact-your-roi.html>
- [13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, the original YOLO paper; foundational reference for the YOLO family. [Online]. Available: <https://arxiv.org/abs/1506.02640>
- [14] NVIDIA, “Jetson Orin Nano Super Developer Kit,” Vendor product page, 2024, orin Nano Super: \$249 MSRP, 67 TOPS, 7-25 W TDP. Closest currently-purchasable edge- GPU peer to DE10-Lite price tier. [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/>
- [15] Terasic Inc., “DE10-Lite User Manual,” Terasic product documentation, 2017, dE10-Lite board pinout, peripherals, and reference design. [Online]. Available: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1021>

- [16] Intel Corporation, “Intel MAX 10 FPGA Device Datasheet,” Document M10-DATASHEET, 2021, family-level datasheet for the MAX 10 device used by the DE10-Lite. [Online]. Available: <https://www.intel.com/content/www/us/en/products/details/fpga/max-10/docs.html>
- [17] Y. Pan, L. Zhang, and Y. Zhang, “Rapid detection of PCB defects based on YOLOx-Plus and FPGA,” *IEEE Access*, vol. 12, 2024, yOLOX-Plus achieves 93.2% mAP and 72.6 FPS on a PYNQ-class FPGA for PCB defect inspection. Author first names not verified – update before submission. [Online]. Available: <https://ieeexplore.ieee.org/document/10497598/>
- [18] Z. Yan, B. Zhang, and D. Wang, “An FPGA-based YOLOv5 accelerator for real-time industrial vision applications,” *Micromachines*, vol. 15, no. 9, p. 1164, 2024, yOLOv5s INT4 on ZCU102; 79.1 FPS, 12.633 ms latency, 12.3 W; 808.6 GOPS at 200 MHz. [Online]. Available: <https://www.mdpi.com/2072-666X/15/9/1164>
- [19] B. Günay, S. B. Okcu, and H. c. Bilge, “LPYOLO: Low precision YOLO for face detection on FPGA,” in *MVML 2022*, 2022, tiny-YOLOv3 redesigned with W4A4 on PYNQ-Z2 via FINN; 18 FPS, 2.4 W. Direct PYNQ-class precedent. [Online]. Available: <https://arxiv.org/abs/2207.10482>
- [20] D. K. Alqahtani, A. Cheema, and A. N. Toosi, “Benchmarking deep learning models for object detection on edge computing devices,” 2024, wall-meter power measurement on Orin Nano (only paper in the survey with wall power vs SoC INA). YOLOv8n minimum 16 ms inference; YOLOv8m: 50 ms / 44 mAP. [Online]. Available: <https://arxiv.org/abs/2409.16808>
- [21] B. Du *et al.*, “YOLO-MBBi: PCB surface defect detection method based on enhanced YOLOv5,” *Electronics*, vol. 12, no. 13, p. 2821, 2023, sOTA on DeepPCB: 99.0% mAP50, 69.5 FPS. Verify full author list (Wan, Lei, Xu, Xu, Xiong) against MDPI record before submission. [Online]. Available: <https://www.mdpi.com/2079-9292/12/13/2821>
- [22] J. Liu, B. Kang, C. Liu, X. Peng, and Y. Bai, “YOLO-BFRV: An efficient model for detecting printed circuit board defects,” *Sensors*, vol. 24, no. 18, p. 6055, 2024, yOLOv8 family on DeepPCB: 98.6% mAP50, 142 FPS; 2.9M params. Directly comparable architecture to HALO-8. [Online]. Available: <https://www.mdpi.com/1424-8220/24/18/6055>
- [23] G. Xiao, S. Hou, and H. Zhou, “PCB defect detection algorithm based on CDI-YOLO,” *Scientific Reports*, vol. 14, p. 7351, 2024, reports both mAP50 (98.3%) and mAP50-95 (51.1%) on PKU-Market-PCB; rare full-metric reporting – direct comparator for HALO-8’s 0.901/0.517. [Online]. Available:

<https://www.nature.com/articles/s41598-024-57491-3>

- [24] J. e. a. Lim, “A deep context learning based PCB defect detection model with anomalous trend alarming system,” *Results in Engineering*, 2023, multi-scale FPN with deep context for tiny defects + trend-alarming layer. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2590123023000956>
- [25] H. Kim and T.-K. Kim, “Design and implementation of a YOLOv2 accelerator on a Zynq-7000 FPGA,” *Sensors*, vol. 25, no. 20, p. 6359, 2025, yOLOv2 INT16 on Zybo-Z7-20; close cost-class peer (\$200) with full disclosure of FPS/W/resource numbers. [Online]. Available: <https://www.mdpi.com/1424-8220/25/20/6359>
- [26] S. Chen, W. Lai, J. Ye, and Y. Ma, “A fast and low-power detection system for the missing pin chip based on YOLOv4-Tiny algorithm,” *Sensors*, 2023, industrial PCB-style missing-pin detection on AXU2CGB; 100% recall, 23% CPU-power reduction. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC10146155/>
- [27] M. Danilowicz and T. Kryjak, “Real-time multi-object tracking using YOLOv8 and SORT on a SoC FPGA,” in *ARC 2025 (Springer LNCS)*, 2025, 4-bit YOLOv8n on ZCU102 via FINN with external-memory extension; 195 FPS detector-only. [Online]. Available: <https://arxiv.org/abs/2503.13023>
- [28] Z. Wang *et al.*, “A scalable OpenCL-based FPGA accelerator for YOLOv2,” in *IEEE FCCM*, 2019, intel Arria-10 GX1150 INT8 YOLOv2; 2.13 TOPS at 211 MHz; representative high-end Intel FPGA target. Verify full author list against IEEE record before submission. [Online]. Available: <https://ieeexplore.ieee.org/document/8735517/>
- [29] NVIDIA Newsroom, “NVIDIA announces Jetson Nano: \$99 tiny yet mighty computer,” Vendor press release, 2019, legacy Jetson Nano launched at \$99 MSRP; discontinued 2023; Maxwell GPU lacks DP4A INT8 instructions. [Online]. Available: <https://nvidianews.nvidia.com/news/nvidia-announces-jetson-nano-99-tiny-yet-mighty-nvidia-cuda-x-ai-computer-that-runs-all-ai-models>
- [30] D. Anand, “Accelerating edge vision: YOLOv8 object detection on Jetson Nano,” ReadyTensor technical report, 2024, yOLOv8n PyTorch FP32 on legacy \$99 Jetson Nano: 6.1 FPS. Only credible YOLOv8n-on-legacy-Nano number in the public literature. [Online]. Available: <https://app.readytensor.ai/publications/accelerating-edge-vision-yolov8-object-detection-on-jetson-nano-4D88m4ggztQt>
- [31] Seede Studio, “Faster YOLOv5 inference with TensorRT on Jetson Nano: 27 FPS,” Seede Studio technical blog, 2022, most-cited "Jetson Nano YOLO" reference number; YOLOv5s TensorRT FP16: 27 FPS. [Online]. Available: <https://www.seeedstudio.com/blog/2022/08/23/faster-inference-with-tensorrt-on-nvidia-jetson-run-yolov5-at-27-fps-on-jetson-nano/>
- [32] J. Speh, “YOLOv7 with TensorRT on Jetson Nano,” GitHub README, 2022,

- yOLOv7-tiny TensorRT FP16 on legacy Jetson Nano: 15 FPS (640x640), 30 FPS (416x416). Useful for legacy-Nano FP16 ceiling estimate. [Online]. Available: <https://github.com/spehjetson-nano-yolov7-tensorrt>
- [33] NVIDIA Developer Forum (community thread), “Jetson Orin Nano FP16/INT8 Performance,” NVIDIA developer forum post (community), 2025, trtexec synthetic numbers: YOLOv8s INT8 on Orin Nano Super: 157 QPS / 6.4 ms; FP16: 103 QPS. Reference desktop number: RTX 3060 Ti FP16 YOLOv8s: 464 QPS. [Online]. Available: <https://forums.developer.nvidia.com/t/jetson-orin-nano-fp16-int8-performance/326723>
- [34] J. Lu, “Pushing limits: YOLOv8 vs v26 on Jetson Orin Nano,” Hackster.io project article, 2026, yOLOv8n FP16 on Orin Nano: 33.2 FPS sustained in real video pipeline (vs 90 FPS raw, 16 FPS Python ONNX FP32). Useful "synthetic vs end-to-end" gap reference. [Online]. Available: <https://www.hackster.io/qwe018931/pushing-limits-yolov8-vs-v26-on-jetson-orin-nano-b89267>
- [35] Seeed Studio, “YOLOv8 with TensorRT and DeepStream on Jetson,” Seeed Studio wiki, 2023, aGX Orin reference: YOLOv8s INT8 313 FPS, FP32 139 FPS. Most-cited AGX Orin number. [Online]. Available: <https://wiki.seeedstudio.com/YOLOv8-DeepStream-TRT-Jetson/>
- [36] Ultralytics, “Quick Start Guide: NVIDIA Jetson with Ultralytics YOLO26,” Ultralytics documentation, 2026, vendor-published YOLO26n benchmarks on Jetson Orin Nano Super (4.57 ms TRT FP16, 0.4800 mAP50-95), Orin NX, AGX Orin, AGX Thor.[Online]. Available: <https://docs.ultralytics.com/guides/nvidia-jetson/>
- [37] I.C.T / morequip, “SMT AOI machine ZMA410 datasheet,” 2025, only public AOI datasheet with a hard kW figure (0.6 kW). Entry-tier inline 2D AOI. [Online]. Available: <https://www.morequip.com/product/SMT-Automatic-Optical-Inspection-Machine-ZMA410.html>
- [38] Ultralytics, “Ultralytics YOLO documentation,” Official documentation site, 2024, general YOLO family documentation (architecture, training, export). [Online]. Available: <https://docs.ultralytics.com/>
- [39] X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang, and J. Yang, “Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection,” in *NeurIPS*, 2020, distribution Focal Loss (DFL) for bounding-box regression; used in YOLOv8’s Detect head. [Online]. Available: <https://arxiv.org/abs/2006.04388>
- [40] M. Horowitz, “Computing’s energy problem (and what we can do about it),” in *ISSCC*, 2014, canonical reference for INT8 vs FP32 vs FP16 multiplier energy / area ratios. [Online]. Available: <https://ieeexplore.ieee.org/document/6757323>
- [41] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *ISCA*, 2017, google TPU v1 paper; canonical Weight Stationary systolic-array reference. [Online]. Available: <https://arxiv.org/abs/1704.04760>