CSEE 4840 Final Report - Spring 2026

Systolic Array based CNN Accelerator

Instructor:

Prof. Stephen A. Edwards

Group Members:

Cheng-wen Chu (cc5397)

Chengcheng Xu (cx2355)

Harvey Lu (hl3999)

Linxiao Wu (lw3227)

Mingyuan Zheng (mz3143)

May 2026

Abstract

This project implements a three-layer INT8 CNN accelerator on the DE1-SoC for classifying 64×64 grayscale ASL digit gestures. The design uses an offline training, quantization, and golden-model flow to generate hardware preload data, then executes inference on an FPGA datapath with reused on-chip SRAM buffers, systolic-array convolution, requantization, pooling, and a fully connected classifier. The accelerator is controlled by the HPS through an MMIO interface and demonstrated through a Flask web application. The compact 3,810-parameter CNN achieves 96.77% validation accuracy with no measured INT8 quantization loss. The FPGA build meets the 50 MHz target clock, reaches a 75.72 MHz accelerator Fmax, and uses 86 of 87 DSP blocks.

Contents

1 Introduction

This project presents a three-layer integer-quantized convolutional neural network (CNN) accelerator implemented on the DE1-SoC platform. The accelerator classifies hand signs for the digits 0–9 from the American Sign Language (ASL) Digits dataset, using 64×64 grayscale images as input, then performs end-to-end inference entirely in INT8 arithmetic. The network achieves 96.77% top-1 validation accuracy, and post-training quantization introduces zero accuracy degradation relative to the floating-point baseline.

The system is partitioned into two domains: an *offline software toolchain* that trains the model, applies full-integer post-training quantization, and packages the parameters into binary load streams; and a *synthesizable RTL pipeline* that receives those streams through a 32-bit streaming port, stores parameters in on-chip SRAM, and executes a pipelined convolution→quantization→pooling datapath followed by a fully-connected classifier and argmax reduction.

At a user level, the system works as a gesture-recognition demo. A user uploads or captures a hand image in the browser, the host software converts it into the same 64×64 INT8 input format used during training, and the DE1-SoC board returns the predicted digit class. Figure 1 shows the ten target gesture classes used by the demo.

Figure 1: American Sign Language (ASL) Digits

The remainder of the report follows the path of the project from model design to board demonstration. We first describe the gesture dataset, compact CNN architecture, training setup, and INT8 quantization flow. We then move from the algorithm into the deployed system, explaining how the host software, HPS Linux tools, Platform Designer interconnect, MMIO wrapper, and RTL accelerator fit together. The middle sections give the detailed hardware/software contract: where model and image data are stored, how the HPS writes and reads the accelerator, and how the internal FPGA memories are reused across CNN layers. The final sections summarize resource usage, timing, web and board validation results, and the operating constraints of the current im-

plementation.

2 Algorithm

2.1 Algorithm: CNN development and training

Dataset

We use the open *ASL Digits 0–9* dataset published on Kaggle by Victor Anthonywhich contains RGB photographs of American Sign Language hand gestures for the digits 0–9. To match the compact input format expected by the hardware accelerator, every image is converted to single-channel grayscale and resampled to 64×64 using `cv2.INTER_AREA`, giving an input tensor of shape $(64, 64, 1)$ with pixel values normalized to $[0, 1]$ via a $1/255$ rescaling.

The 2,062 processed images are partitioned per class into a 70%/15%/15% train / validation / test split, producing 1,444 training, 310 validation, and 308 test images. Class balance is preserved across all three splits (roughly 144 training and 31 validation images per digit).

Network architecture

The classifier is a deliberately small convolutional network chosen so that every layer maps cleanly onto the accelerator's MAC array and on-chip memory budget. It contains three 3×3, stride-1, `valid`-padding convolutions with ReLU activation, each followed by a 2×2 max-pooling stage, a flatten, and a single fully connected classification head (Table 1). The resulting model has only 3,810 trainable parameters, which makes it tractable for FPGA deployment.

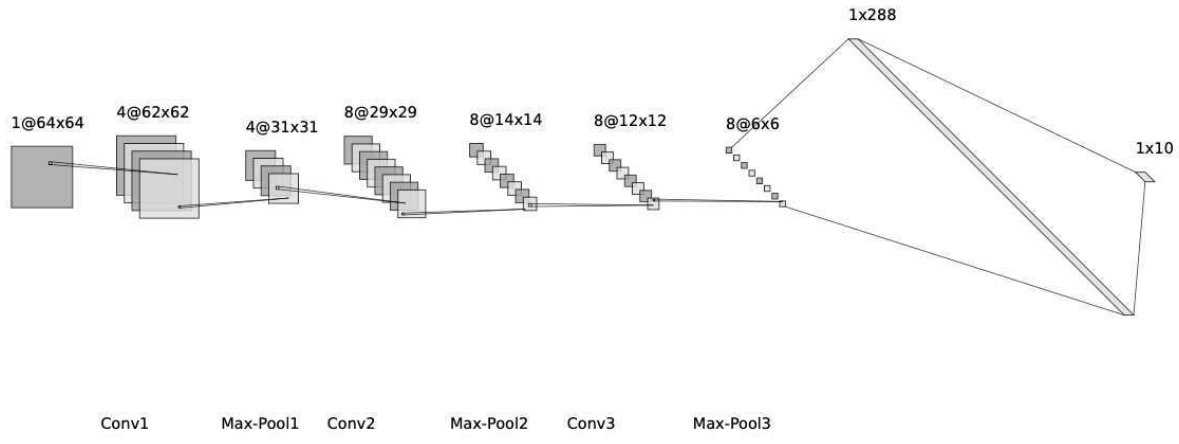


Figure 2: CNN Architecture

Table 1: CNN architecture for 64×64 grayscale digit gestures.

Layer	Kernel / Size	Output shape	Params
Input	–	$64 \times 64 \times 1$	0
Conv2D + ReLU	3×3, $C_{\text{out}} = 4$	$62 \times 62 \times 4$	40
MaxPool	2×2	$31 \times 31 \times 4$	0
Conv2D + ReLU	3×3, $C_{\text{out}} = 8$	$29 \times 29 \times 8$	296
MaxPool	2×2	$14 \times 14 \times 8$	0
Conv2D + ReLU	3×3, $C_{\text{out}} = 8$	$12 \times 12 \times 8$	584
MaxPool	2×2	$6 \times 6 \times 8$	0
Flatten	–	288	0
Dense + Softmax	$288 \to 10$	10	2,890
Total			**3,810**

Training procedure

The network is trained in TensorFlow/Keras with categorical cross-entropy loss and the Adam optimizer (default learning rate 10^{-3}). We use a batch size of 32 and train for up to 200 epochs. Each epoch iterates over $\lceil 1444/32 \rceil = 46$ training steps; validation uses all 310 images. After convergence the network reaches 96.77% top-1 accuracy on the validation split, with per-class F1 scores at or above 0.90 for every digit (Table 2).

Table 2: Per-class validation metrics for the float32 model (support = 31 per class).

Digit	Precision	Recall	F1	Support
0	0.939	1.000	0.969	31
1	0.969	1.000	0.984	31
2	1.000	0.968	0.984	31
3	0.968	0.968	0.968	31
4	0.968	0.968	0.968	31
5	1.000	1.000	1.000	31
6	0.903	0.903	0.903	31
7	0.968	0.968	0.968	31
8	0.966	0.903	0.933	31
9	1.000	1.000	1.000	31
Accuracy	**0.9677 (310 images)**			

Post-training quantization

Floating-point MAC units are costly to implement on FPGA — they require wide mantissa/exponent datapaths, normalization and rounding logic, and significantly more DSP and logic resources than their integer counterparts. To make the network hardware-friendly, every weight, activation,

and accumulator in the inference graph must therefore be reduced to fixed-width integer arithmetic. We achieve this with *full-integer post-training quantization* (PTQ) to `int8` using the TensorFlow Lite converter.

After PTQ, we evaluated the int8 model on the full 310-image validation split end-to-end in `int8` gives 96.77% top-1 accuracy, matching the float32 baseline (Δacc = 0.00) with a float/int8 prediction agreement of 98.39%, confirming that the move to integer arithmetic costs essentially no accuracy for this network. The quantized weights, biases, and per-layer scales are then exported and used by the MATLAB hardware-aligned reference model and the downstream FPGA bitstream.

3 System Block Diagram

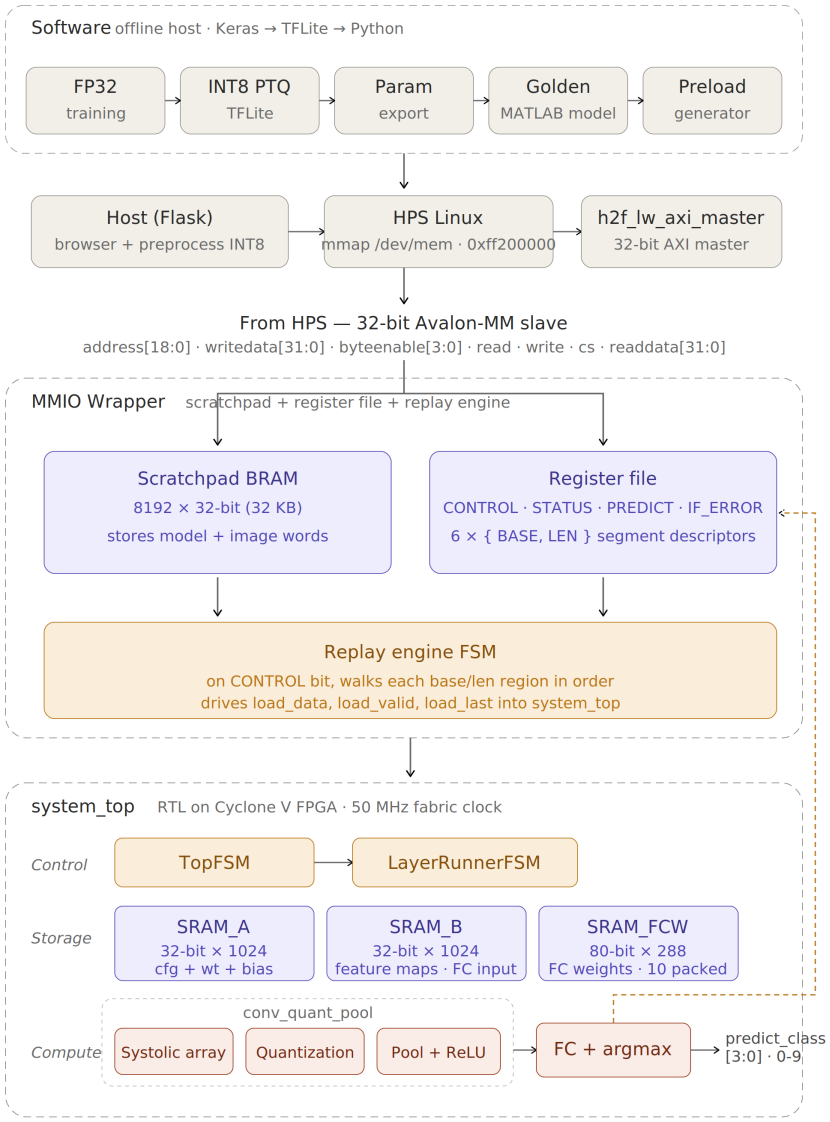


Figure 3: Block Diagram

Figure 3 shows the block diagram of our project, a three-layer INT8 CNN accelerator that performs end-to-end inference from a $64 \times 64 \times 1$ grayscale image to a 0–9 gesture class label. The system is partitioned into an *offline software toolchain* that runs on a development host (top region) and a *synthesizable RTL pipeline* that runs on the FPGA (bottom region). The HPS writes staged payloads through the MMIO wrapper, and the wrapper replays those 32-bit words into the accelerator's streaming load port.

3.1 Software Blocks

The software side is a sequential offline pipeline. We first train a floating-point model, quantize it to INT8, export the quantized parameters, run a bit-exact golden reference, and finally pack everything into the binary streams that the hardware expects.

FP32 training. A three-layer CNN, consisting of three convolution/pooling stages followed by a fully-connected classifier, is trained in a standard deep-learning framework on a 0–9 hand-gesture dataset. The output of this stage is a floating-point model checkpoint.

INT8 quantization. The trained floating-point model is converted into a fully INT8 model through post-training quantization. Weights use per-channel symmetric quantization and activations use per-tensor asymmetric quantization. For every layer this stage also produces the requantization multiplier and right-shift value that the hardware will later apply to each accumulator before saturating it back to the INT8 range.

Parameter export. The quantized model is parsed by a set of host scripts that extract the per-channel weights, biases, scales, and zero-points, and emit them as plain numerical parameter files that downstream stages can consume.

Golden reference model. A bit-exact software re-implementation of the quantized network is run on the host to produce per-layer reference tensors. These tensors are compared against the hardware outputs during RTL simulation so that every pipeline stage can be verified independently.

SRAM preload generator. The final software stage takes the exported parameters and the input images and reformats them for the hardware. Convolution weights are reordered into the column-interleaved layout expected by the systolic array, and the fully-connected weights are packed into the wide slot layout expected by the dedicated FC weight memory. The output is a sequence of 32-bit words that forms the load stream.

Software-internal communication. The software blocks are sequential host programs; they communicate strictly through files. There is no run-time communication between them.

3.2 Host and Runtime Path

The offline software pipeline described above only produces files; the runtime path is what carries one inference from a user click in the browser down to a single MMIO write seen by the FPGA. It spans three distinct machines in three distinct languages.

Browser front end. A small JavaScript page running in the user's browser handles image upload and webcam capture, posts the image to the host as a PNG/JPG blob, and renders the predicted class, timing breakdown, and confidence scores returned by the backend. The page does not emulate the accelerator: it never touches the model itself.

Flask host runtime. A Python Flask application on the development host receives the browser request, preprocesses the image into the same $64 \times 64 \times 1$ INT8 tensor used during training, and exports a temporary inference case directory. The host then transfers the case to the DE1-SoC board over SCP and triggers the board-side tool through SSH.

HPS Linux tool. A C program, `hps_mmio_predict`, runs on the embedded Linux system on the Cyclone V ARM core. It opens `/dev/mem` and `mmap`s the accelerator's 4 MiB Avalon-MM window at HPS physical address `0xff200000`, packs the model and image payloads into 32-bit words, writes them into the wrapper's scratchpad memory, programs the base/length descriptors, and finally sets the `CONTROL` start bit. After triggering inference it polls the `STATUS` register and reads back `PREDICT`, `IF_ERROR`, and the per-stage profile counters.

HPS-to-FPGA bridge. Every store from the C program is transported across the Cyclone V die by the `h2f_lw_axi_master` port of the HPS hard processor system, which exposes a 32-bit lightweight AXI master interface. Platform Designer automatically inserts the required AXI-to-Avalon-MM adapter between this master and our custom Avalon-MM slave, so the RTL only ever sees an ordinary Avalon-MM transaction.

Host-to-board communication. The host and the board run in different operating systems and on different CPUs. They communicate exclusively through SSH and SCP; there is no shared memory, no socket protocol, and no streaming RPC. This keeps the host code portable and the board code minimal, at the cost of a per-inference startup overhead that is reflected in the difference between the pure RTL cycle count and the HPS-observed wait time reported in Section 6.

3.3 MMIO Wrapper

The MMIO wrapper is the entire RTL interface between the Avalon-MM bus and the streaming load port that drives `system_top`. It contains a write-once scratchpad memory, a small register file, and a replay engine that turns the staged payload into the streaming protocol the accelerator core expects. Its purpose is to decouple HPS write timing from the deterministic per-cycle streaming required by the convolution datapath.

Address space split. The wrapper exposes a single 19-bit Avalon-MM word address. Bit `address[18]` selects between two logical spaces: `address[18] = 0` addresses the scratchpad BRAM,

and `address[18] = 1` addresses the configuration and status register file. From the Linux side this is a flat memory-mapped region; the partition is entirely an internal decoding decision.

Scratchpad BRAM. An 8192×32-bit on-chip BRAM (32 KB) stores the four model preload segments and the input image before inference begins. The HPS tool writes every word into this scratchpad through the Avalon-MM interface; the wrapper later replays them into the accelerator core word-by-word. Staging the payload in BRAM, rather than streaming it directly from the HPS, removes any timing dependence between the (asynchronous) software writes and the (strictly cycle-deterministic) streaming load protocol.

Register file. Fourteen 16-bit registers expose the control surface to the HPS. `CONTROL` kicks off model preload or inference; `STATUS` reports busy, model-loaded, inference-done, and the predicted class; `PREDICT` latches the final class and `IF_ERROR` reports interface-level errors such as a start request while the engine is still busy or an out-of-range scratchpad access. Six base/length pairs ({`CONV_CFG`, `CONV_WT`, `FC_BIAS`, `FCW`, `IMAGE`} each with `BASE` and `LEN`) describe where each preload segment lives in the scratchpad and how many 32-bit words it spans. The complete register map and bit-field definitions are listed in Section **??**.

Replay engine FSM. When the HPS sets `CONTROL[0]` or `CONTROL[1]`, the replay engine walks each configured base/length region in order, reads one 32-bit word per cycle from the scratchpad, and drives the three-signal streaming protocol (`load_data[31:0]`, `load_valid`, `load_last`) into `system_top`. During model load it walks the four model regions; during inference it walks only the image region, in which case `system_top` consumes the stream directly as the first convolution layer's input. When the accelerator signals completion, the replay engine returns to idle and sets the corresponding done flag in `STATUS`.

Readback path. Inference results travel back through the same wrapper. The `system_top` output `predict_class[3:0]` and the per-stage cycle counters are latched into the `STATUS`/`PREDICT`/`PROFILE` registers when the accelerator asserts `predict_done`. The HPS tool reads these registers through the Avalon-MM `readdata` channel and forwards them to the host, which renders them in the browser.

3.4 Hardware Blocks

Systolic Array (SA) structure The systolic array is organized as a 2-D grid of processing elements (PEs), where each PE performs a local MAC operation on streaming activations and weights [1]. Each PE receives input activations from one direction (from the left) and weights from an orthogonal direction (from the top), multiplies them, accumulates the result into a local register, and then forwards the activation and weight to its neighboring PEs in the next cycle.

For a convolution layer, input feature map tiles and kernel weights are scheduled into the array so that each PE computes partial sums for a subset of output channels and spatial locations over time. This dataflow reuses activations and weights efficiently across rows and columns of the array, which reduces external memory bandwidth and allows high throughput with a simple, regular hardware layout. After a fixed number of cycles corresponding to the kernel size and mapped channels, the partial sums in the PEs are ready and are forwarded to the quantization and pooling stages. The structure and single PE of the systolic array are shown in Fig. 4 and Fig. 5 as below.

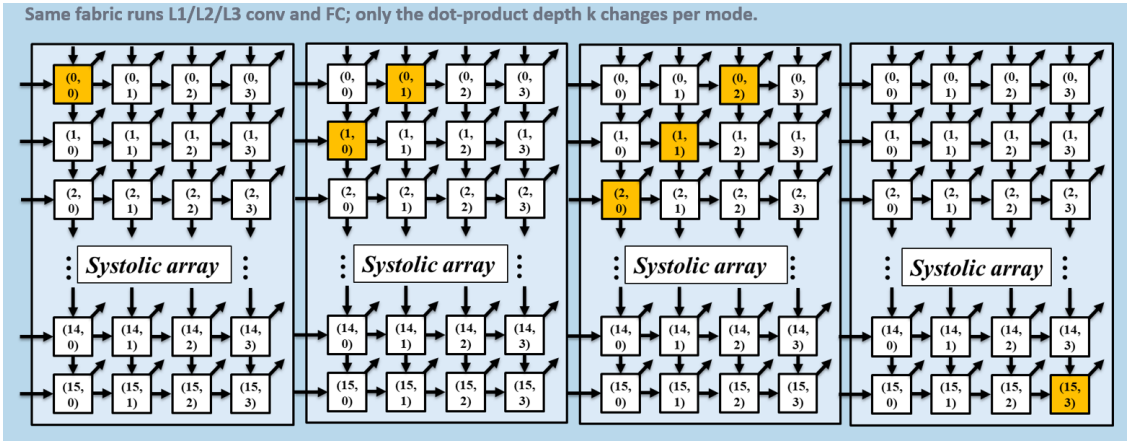


Figure 4: Systolic Array — 16×4 PE Grid, reused across all layers

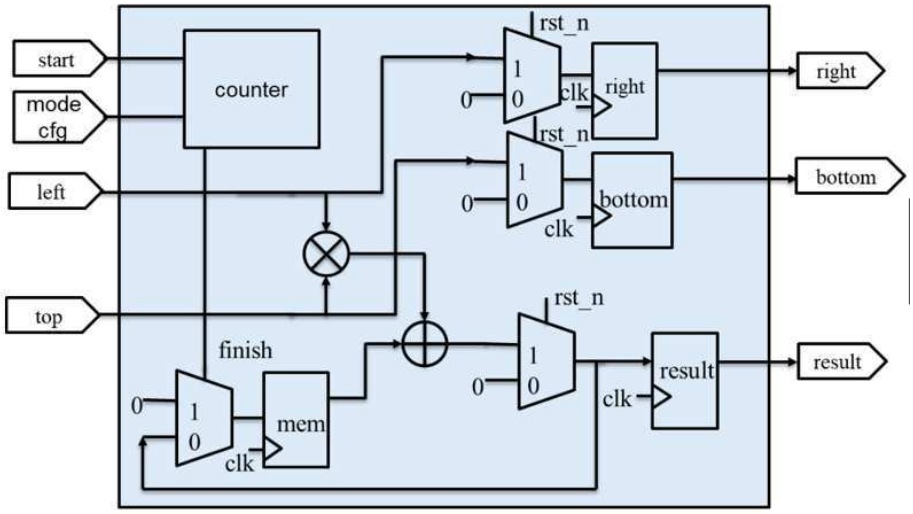


Figure 5: Single PE

3.5 Quantization internal computation

Quantization maps high-precision MAC results to low-precision fixed-point integers to save memory and bandwidth. Inside the quantization (QNT) block, each MAC output x (23-bits) is first scaled

by a learned or pre-defined factor s, then rounded and saturated into an 8-bit range. Concretely, the computation can be described as:

$$y_{\text{float}} = s \cdot x, \quad y_{\text{int}} = \text{round}(y_{\text{float}}), \quad y_{\text{q}} = \text{clip}(y_{\text{int}}, -128, 127)$$

where yq is the int8 activation written back to memory or passed to the next layer.

In hardware, this is implemented as a shift-and-add or multiplier stage for the scaling factor, followed by a rounding unit and a saturating adder/comparer block, which is shown as Fig. **??**. The QNT unit is fully pipelined so it can accept one MAC result per cycle from the systolic array, ensuring that quantization does not become a performance bottleneck for the accelerator dataflow.

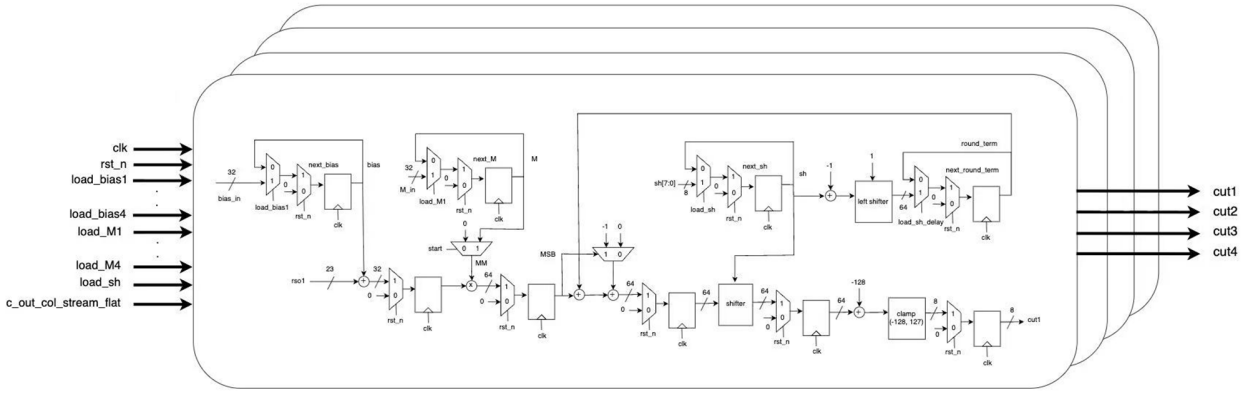


Figure 6: Quantization: 32-bit accumulator → 8-bit activation

3.6 Pooling pipeline

The pooling pipeline processes the quantized feature maps to reduce spatial resolution and increase invariance. After a convolution layer is completed and its outputs are quantized, the feature map is streamed into the pooling unit, which forms 2×2 spatial windows and computes the maximum value over each window.

This pooling operation is also pipelined: each stage performs part of the comparison reduction, so that a new window can enter the pipeline once the required inputs are available. The pooled outputs are then written back to on-chip memory and reused as inputs to the next convolution layer, forming a continuous conv → quantization → max-pooling pipeline across the 3-layer CNN.

Load port. The only data entry point of the chip. The host streams 32-bit beats with a `valid/ready/last` handshake. A mode signal selects between model-load traffic (configuration, convolution weights, FC bias, and FC weights) and image-load traffic, which also triggers inference.

Top-level FSM. A network-level sequencer. During model-load it routes the four preload segments to their destinations in the on-chip memories. During inference it drives the fixed layer schedule of the three convolutional/pooling stages followed by the fully-connected layer, and finally an argmax reduction over the ten output accumulators that produces the predicted class.

Layer-runner FSM. A per-layer transaction controller that loads the current layer's configuration and weights, streams data through the convolution datapath, and waits for the frame to finish. It decouples the network-level schedule from the per-layer streaming protocol and generates the control signals for the on-chip memories.

SRAM_A. A 32-bit memory that stores the convolution configuration words, the convolution weights, and the fully-connected bias vector. It also doubles as the writeback buffer for the second convolution layer, using a stride-2 interleaved address region.

SRAM_B. A 32-bit memory that holds the pooled feature maps between layers. The output of the first pooling stage and the output of the last pooling stage share the same physical memory; the later write simply overwrites the earlier data once it is no longer needed.

SRAM_FCW. A dedicated wide memory for the fully-connected weights. Each slot packs the per-output-channel weights for one flattened input position, so that all ten output channels can be fed in parallel in a single read. On the FPGA the internal storage can be mapped onto a block-memory IP.

All three memories receive control signals from the FSMs and expose streaming interfaces for data access.

3.7 Memory Resource Allocation

The memory subsystem supports both model parameter storage and intermediate activation buffering while enabling data reuse across CNN layers. The design adopts three logical on-chip memories: `SRAM_A`, `SRAM_B`, and `SRAM_FCW`. Although these modules follow SRAM-style interfaces, they are implemented in the FPGA using Verilog register arrays rather than physical SRAM macros.

SRAM_A (parameter and activation storage). `SRAM_A` is a 32-bit $\times$ 1024-word memory, giving a total capacity of 4 KB. During model preload, it stores convolution configuration words, convolution weights, and fully-connected bias values. During inference, it is also reused as an intermediate activation buffer. After the second convolution and pooling stage, the $14 \times 14 \times 8 = 1568$ byte output feature map is written into `SRAM_A` and later read as the input to the third convolution layer.

SRAM_B (activation buffer). `SRAM_B` is also a 32-bit × 1024-word memory. It is used mainly for intermediate feature maps between convolution layers. The first pooling stage writes a $31 \times 31 \times 4 = 3844$ byte feature map into `SRAM_B`, which is then reused as the input to the second convolution layer. Later, the third pooling stage writes the $6 \times 6 \times 8 = 288$ byte final feature map back into `SRAM_B`. This final feature vector is then consumed by the fully-connected classifier.

SRAM_FCW (fully-connected weight memory). `SRAM_FCW` is a dedicated wide memory for fully-connected weights. Each entry is 80 bits wide and contains ten INT8 weights, one for each output class. The implemented memory has 512 physical entries, with 288 entries used by the current $6 \times 6 \times 8$ feature vector. During execution, one 80-bit word is read per cycle, allowing one input activation to update all ten class accumulators in parallel.

Streaming input and dataflow. The original input image is not first stored in `SRAM_A` or `SRAM_B`. Instead, the HPS stages the packed image in the MMIO scratchpad, and the MMIO wrapper replays those 32-bit words directly into the first convolution layer. After that point, only pooled INT8 feature maps are written back to the local SRAM buffers for reuse by later layers.

Across layers, the feature map sizes are:

- Input: $64 \times 64 \times 1 = 4096$ bytes
- Layer 1 output: $31 \times 31 \times 4 = 3844$ bytes
- Layer 2 output: $14 \times 14 \times 8 = 1568$ bytes
- Layer 3 output: $6 \times 6 \times 8 = 288$ bytes

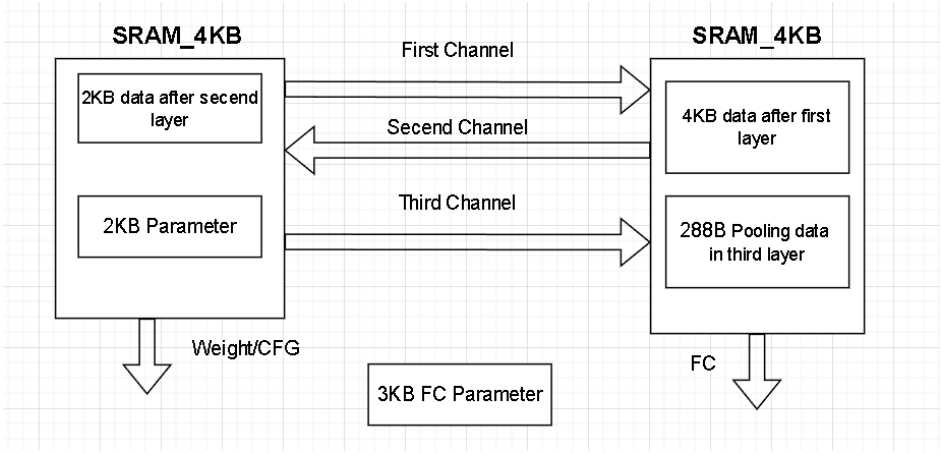


Figure 7: Memory organization and data reuse across CNN layers.

The overall dataflow follows the reuse pattern:

```
Input -> L1 -> SRAM_B -> L2 -> SRAM_A -> L3 -> SRAM_B -> FC
```

Design rationale. The memory architecture is optimized for three objectives: (1) minimizing on-chip memory usage through buffer reuse, (2) reducing external memory bandwidth by keeping intermediate feature maps on chip, and (3) improving fully-connected throughput through wide parallel weight reads. This balance allows the full CNN pipeline to fit within the limited FPGA memory budget.

3.8 Memory-to-Systolic-Array Datapath

After the feature maps are stored or replayed through the memory system, the convolution core consumes them as a packed 32-bit stream. This section describes how that stream is unpacked, buffered, converted into a 3×3 sliding window, and scheduled into the systolic array.

The activation source depends on the active layer. For layer 1, the input image is replayed directly from the MMIO scratchpad. For layer 2, the input feature map is read from `SRAM_B`. For layer 3, the input feature map is read from `SRAM_A`. In all three cases, the convolution core receives a 32-bit word stream and interprets the packed bytes according to the active layer configuration.

The line-buffer byte grouping changes because the number of channels per spatial position changes after each pooling stage. Layer 1 consumes a $64 \times 64 \times 1$ input, so each spatial position contains one INT8 byte. Layer 2 consumes a $31 \times 31 \times 4$ feature map, so each spatial position contains four INT8 channel values. Layer 3 consumes a $14 \times 14 \times 8$ feature map, so each spatial position contains eight INT8 channel values. The SRAM access width remains fixed at 32 bits, while the input adapter and line buffers reinterpret the byte stream for the current layer[2].

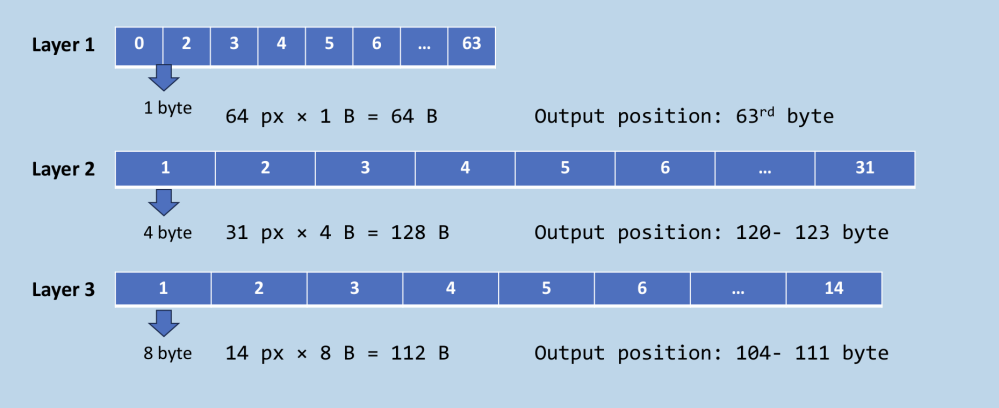


Figure 8: Layer-dependent row layout for the line-buffer input stream.

Figures 9 and 10 illustrate the sliding-window formation using a simplified 8×8, single-channel input. The same buffering principle is used in the actual accelerator, but the real row width and byte grouping are selected from the active layer configuration shown in Figure 8.

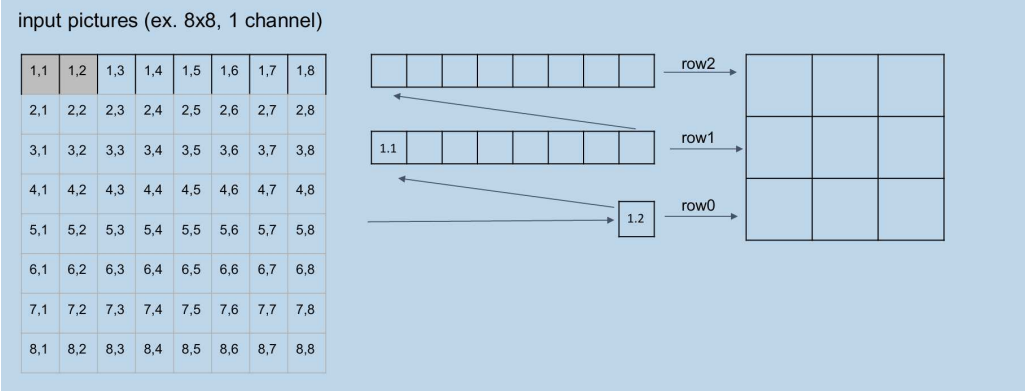


Figure 9: Initial line-buffer fill before a complete 3×3 convolution window is valid.

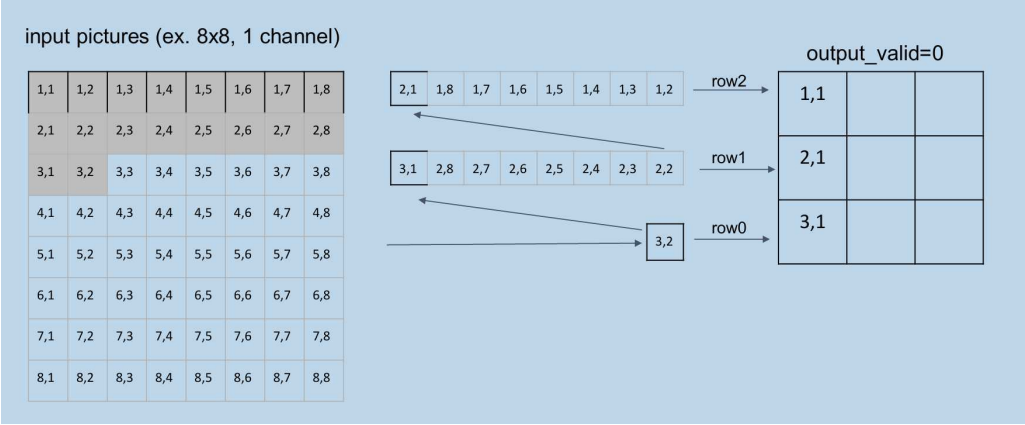


Figure 10: Formation of the three aligned rows used by the 3×3 sliding convolution window.

During the warm-up phase, the line buffers consume the packed memory stream and delay earlier rows so that three adjacent image rows can be presented at the same time. The convolution window is not marked valid until the datapath has received enough rows and columns to populate all nine positions of the 3×3 window. Once the window is valid, the input aligner and skew feeder schedule the INT8 activation values into the systolic array, where they are multiplied with reordered INT8 weights and accumulated into wider partial sums.

Convolution datapath (conv-quant-pool). The output of the systolic array continues through a three-stage streaming compute path:

- **Systolic array** — forms MAC results from the 3×3 sliding window using parallel INT8×INT8 multiply-accumulate operations.
- **Quantization stage** — rescales each partial sum using the per-channel multiplier, shift, and effective bias, then saturates the result back to INT8.

- **Pooling stage** — performs 2×2 stride-2 max-pooling and fuses the ReLU behavior before writeback.

The three stages are connected by streaming handshakes, so intermediate convolution sums are not written back to memory. Only the compact INT8 pooled feature maps are stored in `SRAM_A` or `SRAM_B` for reuse by the next layer.

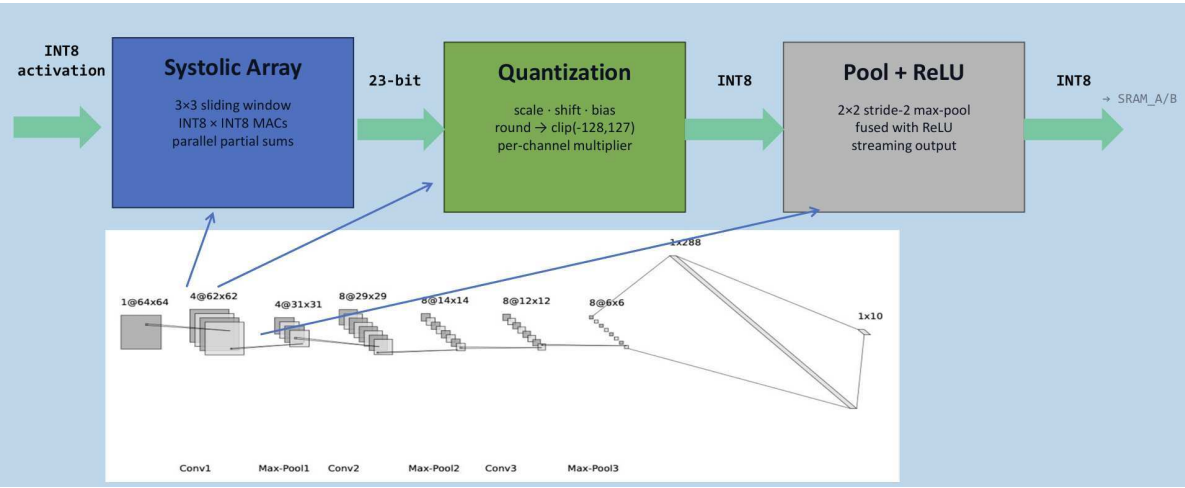


Figure 11: Streaming convolution datapath from the systolic array through quantization and pooling.

FC and argmax. A ten-way parallel multiply-accumulate array that computes all ten class scores simultaneously, reading the broadcast pixel from the feature-map memory, the ten per-channel weights from the wide FC weight memory, and the bias vector from the configuration memory. The ten final accumulators are reduced by the top-level FSM into the 4-bit predicted class, together with a one-cycle valid pulse on the output pins.

Memory	Width	Depth	Size	Function
`SRAM_A`	32-bit	1024	4 KB	Stores convolution configuration, convolution weights, FC bias, and layer-2 output feature maps
`SRAM_B`	32-bit	1024	4 KB	Stores intermediate feature maps (layer-1 and layer-3 outputs) and serves as FC input buffer
`SRAM_FCW`	80-bit	512 physical, 288 used	5 KB physical, 2.88 KB used	Stores fully-connected weights; each entry contains 10 INT8 weights for parallel computation

Table 3: On-chip memory allocation and functionality

3.9 Connection Path

One FPGA inference request crosses the host, HPS Linux, Platform Designer, and RTL boundaries shown in Table 4.

Table 4: End-to-end connection path for one FPGA inference request.

Step	Actor	Action	Data or interface
1	Browser	Upload or capture one gesture image.	PNG/JPG image bytes.
2	Host Flask runtime	Preprocess the image into the model input format.	$64 \times 64 \times 1$ INT8 tensor.
3	Host runtime	Export a temporary inference case and copy it to the board.	Case directory sent by SCP.
4	HPS Linux tool	Run `hps_mmio_predict`.	Board-side command over SSH.
5	HPS Linux tool	Map the accelerator address range.	`/dev/mem` at `0xff200000`.
6	HPS Linux tool	Write packed image/model words into the MMIO scratchpad.	32-bit H2F MMIO writes.
7	HPS Linux tool	Write base/length and control registers.	Avalon-MM config register space.
8	MMIO wrapper	Replay scratchpad words into the streaming accelerator core.	`load_data`, `load_valid`, `load_last`.
9	CNN accelerator	Run the fixed CNN schedule and latch outputs.	Predicted class and profile counters.
10	HPS Linux tool	Read back status, prediction, error, and profile registers.	MMIO reads returned to Flask/browser.

4 The Hardware/Software Interface

4.1 Platform Designer And Bus Integration

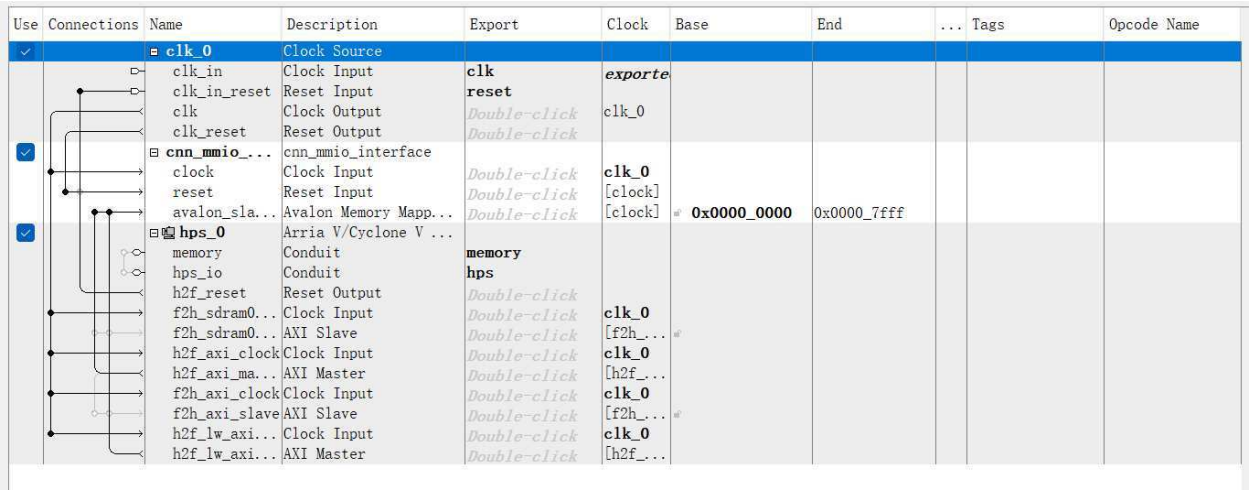


Figure 12: Screenshot of Platform Designer

As shown in Fig .12, the Platform Designer system is centered on three blocks: `clk_0`, `hps_0`, and `cnn_mmio_interface`. The `clk_0` block exports the board clock and reset to the rest of the system. The `hps_0` block is the Cyclone V hard processor system, including the ARM processor, HPS DDR interface, board I/O conduits, and HPS-to-FPGA bridge ports.

The active software path uses the lightweight HPS-to-FPGA bridge. In the HPS port name `h2f_lw_axi_master`, `h2f` means that Linux on the HPS initiates the transaction and the FPGA component responds as a peripheral. The custom CNN wrapper is mapped at offset `0x0000_0000` in this bridge window, so the HPS software accesses it at physical address `0xff200000`.

The CNN wrapper itself is packaged as a 32-bit Avalon-MM slave rather than as an AXI slave. Platform Designer automatically inserts the required interconnect and protocol adapter between the HPS AXI-side master and the custom Avalon-MM slave. This lets the RTL implement a simple memory-mapped register and scratchpad interface while Linux still sees an ordinary HPS physical address.

The `f2h` ports shown on the HPS block are the reverse direction: FPGA-to-HPS. They are available on the HPS IP, but this accelerator does not use them for inference. Data is pushed into the accelerator by HPS software over H2F, and intermediate feature maps remain in FPGA on-chip memories.

4.2 Signal-Level MMIO Interface

The custom IP exposes a 32-bit Avalon-MM slave. The key signals are shown in Table 5.

Table 5: Custom IP Avalon-MM signal interface.

Signal	Direction	Width	Meaning
`address`	input	13	Word address; `address[12]` selects scratchpad memory space.
`writedata`	input	32	Data written by the HPS/interconnect.
`byteenable`	input	4	Byte write enables.
`write`	input	1	Write transaction strobe.
`read`	input	1	Read transaction strobe.
`chipselect`	input	1	Selected transaction qualifier.
`readdata`	output	32	Data returned to the HPS/interconnect.

4.3 Address Space

The wrapper uses `address[12]` as a space selector. The low logical space is the 16 KiB scratchpad memory. The high logical space is the config/status register file. Software uses word addresses `0x0000..0x0FFF` for scratchpad writes and `0x1000..0x101F` for the 32 active register slots.

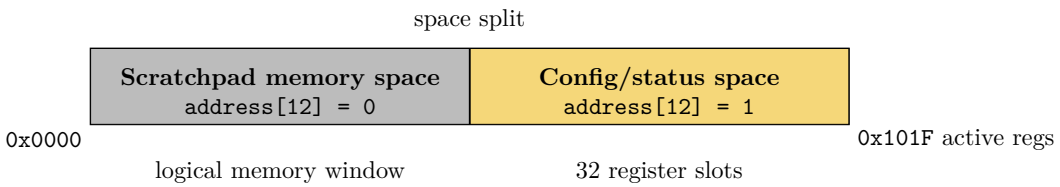


Figure 13: MMIO address-space split in the custom wrapper.

There are three sizes to distinguish:

Table 6: MMIO map sizes.

Layer	Size	Purpose
HPS userspace mapping	20 KiB	Convenient `/dev/mem` mapping window.
Scratchpad namespace	`0x0000..0x0FFF` word indices	4096 32-bit words selected by `address[12]=0`.
Config/status registers	`0x1000..0x101F` word indices	32 active register slots selected by `address[12]=1`.
Implemented scratchpad BRAM	4096 32-bit words, 16 KiB	Actual storage in `cnn_mmio_interface`.

4.4 Register Map

The Avalon-MM slave returns 32-bit read data. The compact control/status and base/length fields use the low 16 bits; profile counters are exposed as 32-bit values through the profile register slots. Table 7 summarizes the register interface exposed by the MMIO wrapper.

Register	Index	Access	Description
CONTROL	0	WO	start model load, start inference, clear status
STATUS	1	RO	busy flag, model-loaded flag, done flag, predicted class
CONV_CFG_BASE	2	RW	base word address of convolution configuration segment
CONV_CFG_LEN	3	RW	number of 32-bit words in convolution configuration segment
CONV_WT_BASE	4	RW	base word address of convolution weight segment
CONV_WT_LEN	5	RW	number of 32-bit words in convolution weight segment
FC_BIAS_BASE	6	RW	base word address of FC bias segment
FC_BIAS_LEN	7	RW	number of 32-bit words in FC bias segment
FCW_BASE	8	RW	base word address of FC weight segment
FCW_LEN	9	RW	number of 32-bit words in FC weight segment
IMAGE_BASE	10	RW	base word address of input image segment
IMAGE_LEN	11	RW	number of 32-bit words in input image segment
PREDICT	12	RO	latched predicted class
IF_ERROR	13	RO	interface error flags
PROFILE_*	14–29	RO	per-stage cycle counters; the low slots are reserved and the high slots hold the 32-bit counts
LAST_WRITE	30	RO	debug readback of the most recent write transaction
MAGIC	31	RO	debug constant 0x434E4E32

Table 7: MMIO register map.

4.5 Register Fields

Table 8 lists the meaning of each implemented control and status bit.

Register	Bits	Meaning
CONTROL	bit 0	start model preload replay
CONTROL	bit 1	start inference replay
CONTROL	bit 2	clear predict_done and IF_ERROR
CONTROL	bits 15:3	reserved, write zero
STATUS	bit 0	reserved
STATUS	bit 1	replay engine busy
STATUS	bit 2	model preload complete
STATUS	bit 3	inference complete
STATUS	bits 7:4	predicted class
STATUS	bits 15:8	reserved
PREDICT	bits 3:0	predicted class
PREDICT	bits 15:4	reserved
IF_ERROR	bit 0	model load requested while busy
IF_ERROR	bit 1	inference requested while busy
IF_ERROR	bit 2	inference requested before model load completed
IF_ERROR	bit 3	scratchpad replay read out of range
IF_ERROR	bits 15:4	reserved

Table 8: Implemented control and status bit definitions.

4.6 Scratchpad Layout

The HPS writes all model and image payloads into the scratchpad before triggering the replay engine. All base addresses are 32-bit word addresses.

Table 9: Default scratchpad layout for the active 10-class model.

Region	Base word	Words	Bytes	Content
conv_cfg	0	45	180	layer/pass configuration words
conv_wt	45	225	900	packed convolution weights
fc_bias	270	10	40	10 FC bias words
fcw	280	864	3456	packed fully connected weights
image	1144	1024	4096	packed 64×64 INT8 image

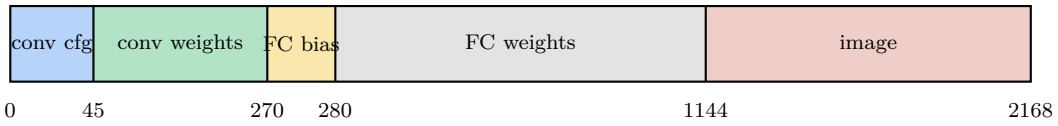


Figure 14: Contiguous 32-bit word layout used by the HPS tools. The figure is drawn for readability; exact word counts are listed in Table 9.

4.7 Model Preload Storage and Replay

The model preload directory contains four payloads:

- `preload_conv_cfg_45w.txt`: 45 signed 32-bit configuration words.
- `preload_conv_wt_225w_bytes.txt`: 225 packed words containing convolution weights.
- `preload_fc_bias_10w.txt`: 10 signed 32-bit FC bias words.
- `preload_fcw_864w.txt`: 864 signed 32-bit words carrying packed FC weights.

During model load, the HPS tool programs the default base/length registers, writes the four model regions into scratchpad memory, and writes `CONTROL[0]`. The wrapper then walks the configured regions in order, loads one 32-bit word at a time from scratchpad BRAM, and drives `load_valid`, `load_data`, and `load_last` into `system_top`. When the internal accelerator reports model-load completion, the wrapper sets `STATUS[2]`.

4.8 Image Storage And Inference Replay

The inference case directory contains the packed input source and expected class metadata. The important file for the FPGA path is `tb_conv1_in_i8_64x64x1.txt`. The board-side HPS code packs every four signed INT8 pixels into one 32-bit word. A 64×64 image therefore occupies 4096 bytes, or 1024 32-bit scratchpad words.

For inference, the HPS tool writes the image words into the image region, writes `CONTROL[1]`, and polls `STATUS[3]`. The wrapper replays the image region into the internal load stream with inference mode selected. Layer 1 consumes the image stream, later layers operate from the accelerator's on-chip SRAM dataflow, and the final argmax result is latched for MMIO readback.

4.9 Transaction Sequence

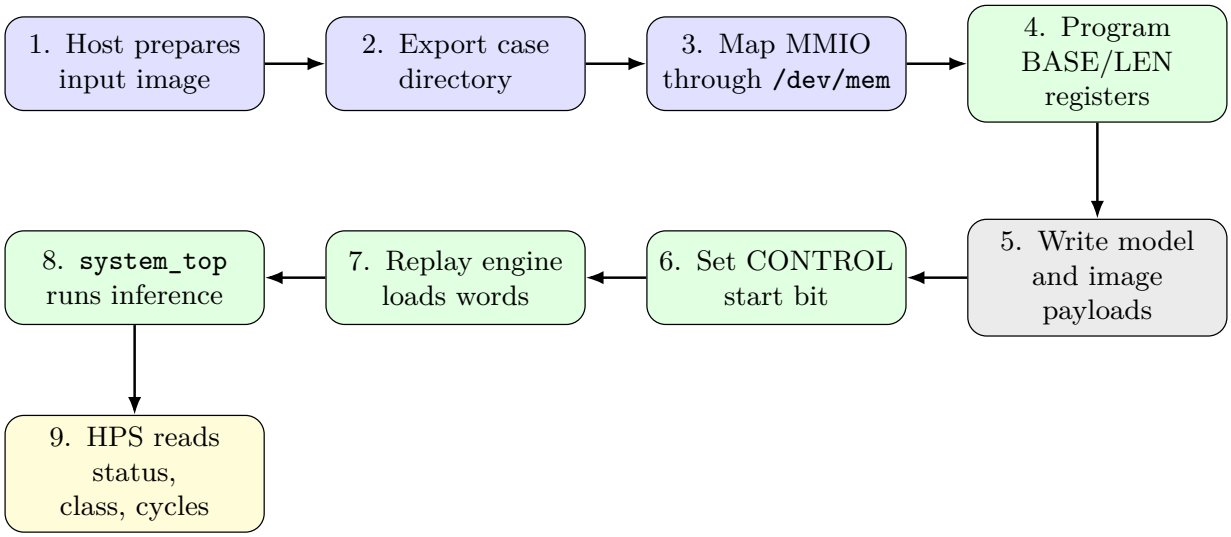


Figure 15: Software-triggered model-load and inference transaction state flow.

4.10 Readback Path

When inference completes, `system_top` provides the predicted class and profile counters to the MMIO wrapper. The wrapper exposes them through register reads:

- `STATUS[3]` tells software that prediction is done.
- `STATUS[7:4]` and `PREDICT[3:0]` contain the class.
- `IF_ERROR` reports incorrect command ordering or replay range errors.
- `PROFILE_*` registers report per-stage cycle counts.

The board tool prints these fields as text. The host parses the output and returns it to the Flask app, which then renders the result in the browser.

5 Internal Memory And Data Reuse

The accelerator avoids external DDR for intermediate feature maps. Once payloads have entered the FPGA wrapper, the CNN datapath uses local on-chip storage and deterministic streaming.

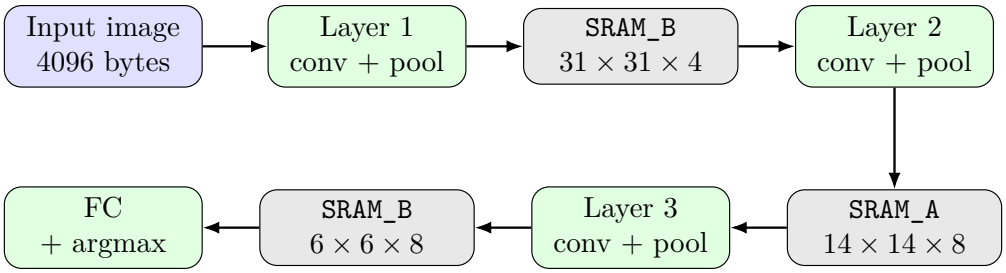


Figure 16: Internal feature-map memory reuse across CNN layers.

Table 10: Logical on-chip memories.

Memory	Width/depth	Main role	Reuse
`SRAM_A`	32-bit by 1024	configuration, conv weights, FC bias	layer-2 output buffer
`SRAM_B`	32-bit by 1024	activation feature maps	layer-1 and layer-3 pooled outputs
`SRAM_FCW`	80-bit by 512, 288 used	fully connected weights	ten class weights per FC input

The final feature vector has $6 \times 6 \times 8 = 288$ INT8 values. Each FC input position reads one packed 80-bit FC weight entry containing one INT8 weight for each of the ten classes. This lets the FC datapath update all ten class accumulators in parallel.

6 Performance And Resource Results

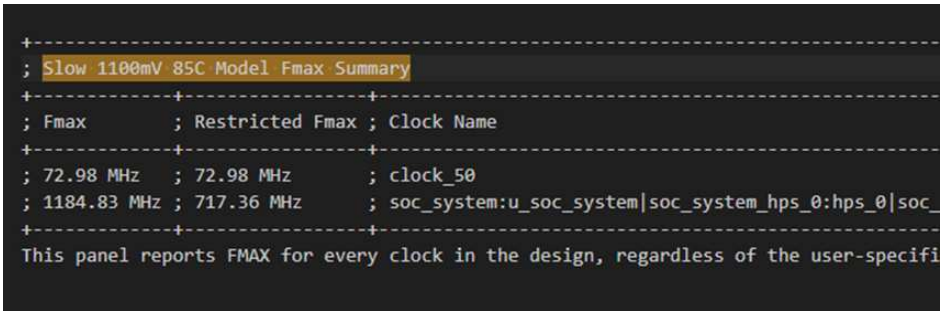
6.1 Timing

The target fabric clock is 50 MHz. The current build meets that target with positive slack on `clock_50`.

Table 11: Current timing summary for the user fabric clock.

Corner	Metric	Value	Meaning
Slow 1100 mV 85 C	setup slack	6.297 ns	Positive at 50 MHz.
Slow 1100 mV 85 C	hold slack	0.220 ns	Positive hold margin.
Slow 1100 mV 0 C	setup slack	6.661 ns	Positive at 50 MHz.
Slow 1100 mV 85 C	Fmax	72.98 MHz	Useful headroom above 50 MHz.
Slow 1100 mV 0 C	Fmax	74.97 MHz	Useful headroom above 50 MHz.

The full SoC timing report can still show HPS DDR-related paths as the whole-design worst paths because Linux and the HPS subsystem remain in the project. For accelerator-centric timing, the relevant clock is `clock_50`. The evidence of Fmax is shown in Fig. 17.

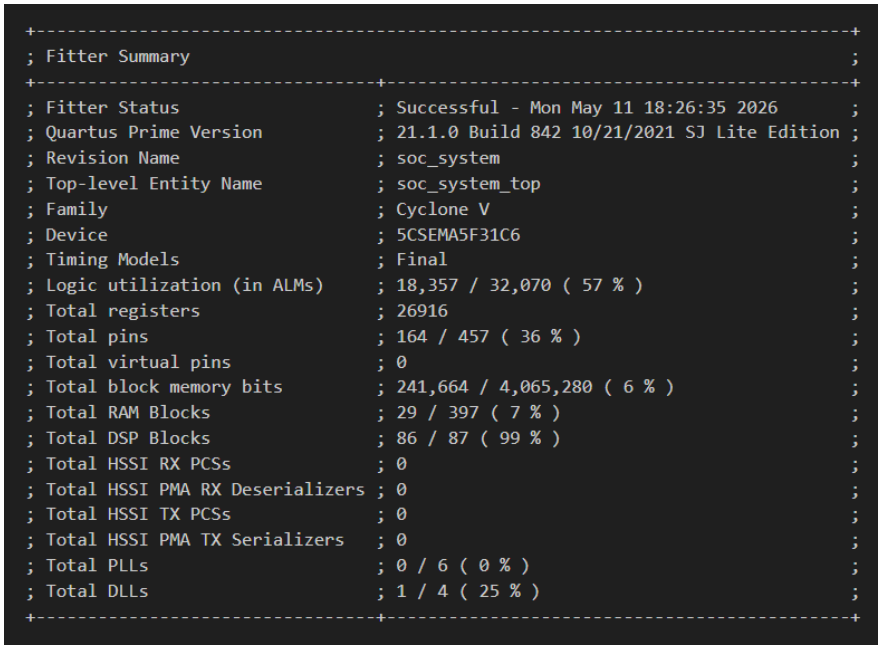


```
; Slow 1100mV 85C Model Fmax Summary
; Fmax        ; Restricted Fmax ; Clock Name
; 72.98 MHz   ; 72.98 MHz       ; clock_50
; 1184.83 MHz ; 717.36 MHz      ; soc_system:u_soc_system|soc_system_hps_0:hps_0|soc_
This panel reports FMAX for every clock in the design, regardless of the user-specifi
```

Figure 17: Evidence of Maximum Frequency

6.2 Resource Utilization

The current post-fit resource usage is shown as Fig.18. The design fits the Cyclone V device, but DSP usage is the tightest resource.



```
; Fitter Summary                                                              ;
; Fitter Status                  ; Successful - Mon May 11 18:26:35 2026       ;
; Quartus Prime Version          ; 21.1.0 Build 842 10/21/2021 SJ Lite Edition ;
; Revision Name                  ; soc_system                                  ;
; Top-level Entity Name          ; soc_system_top                              ;
; Family                         ; Cyclone V                                   ;
; Device                         ; 5CSEMA5F31C6                                ;
; Timing Models                  ; Final                                       ;
; Logic utilization (in ALMs)    ; 18,357 / 32,070 ( 57 % )                    ;
; Total registers                ; 26916                                       ;
; Total pins                     ; 164 / 457 ( 36 % )                          ;
; Total virtual pins             ; 0                                           ;
; Total block memory bits        ; 241,664 / 4,065,280 ( 6 % )                 ;
; Total RAM Blocks               ; 29 / 397 ( 7 % )                            ;
; Total DSP Blocks               ; 86 / 87 ( 99 % )                            ;
; Total HSSI RX PCSs             ; 0                                           ;
; Total HSSI PMA RX Deserializers ; 0                                          ;
; Total HSSI TX PCSs             ; 0                                           ;
; Total HSSI PMA TX Serializers  ; 0                                           ;
; Total PLLs                     ; 0 / 6 ( 0 % )                               ;
; Total DLLs                     ; 1 / 4 ( 25 % )                              ;
```

Figure 18: Resource Utilization from Quartus Report

7 Demo And Validation

7.1 Web Demo Screenshot

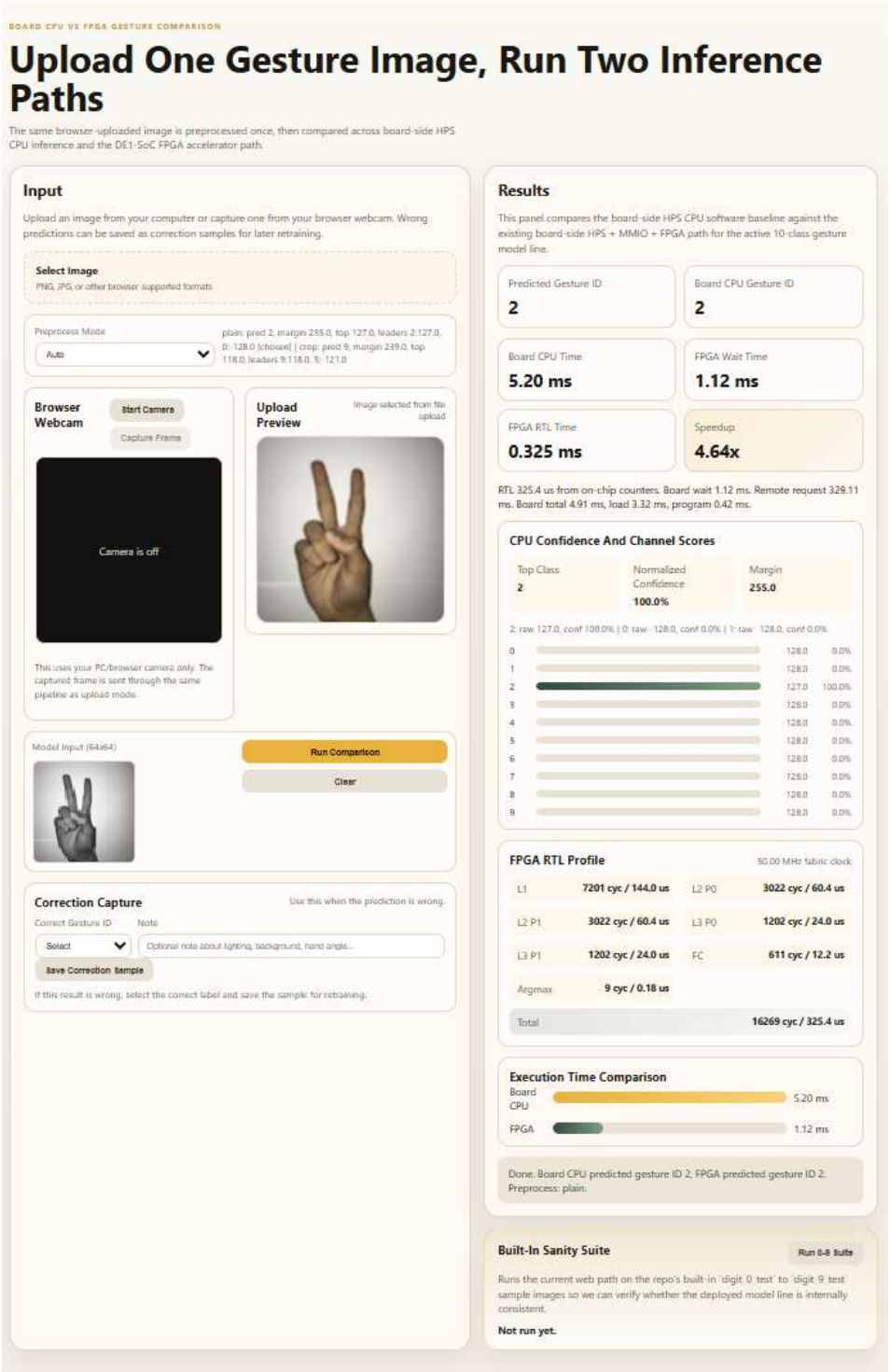


Figure 19: Screenshot of Web Demo

The web demo is the user-facing control and visualization layer of the final system. It is implemented as a Flask application running on the development host, with browser-side JavaScript for image selection, webcam capture, result rendering, and feedback collection. The page does not emulate the FPGA result: when the user presses `Run Comparison`, the backend creates a hardware-aligned inference case, transfers it to the DE1-SoC over SSH/SCP, runs the HPS CPU baseline and the FPGA MMIO path on the board, and returns the measured results to the browser.

Table 12: Main functions provided by the web demo.

Function	User-visible behavior	Backend/system behavior
Image upload	Select a PNG/JPG gesture image from the local computer.	The browser sends the file to `POST /api/infer`; the host preprocesses it into a $64 \times 64 \times 1$ INT8 tensor.
Webcam capture	Start the browser camera and capture one frame as the input image.	The captured frame is converted to a PNG blob and submitted through the same upload path as a file image.
Preprocess selection	Choose `Auto`, `Plain Resize`, or `Crop + Background Removal`.	The host evaluates the requested preprocessing branch. In auto mode, it compares candidate branches using the CPU model confidence and selects the stronger one.
Run comparison	Display board CPU prediction, FPGA prediction, timing, and speedup for the same input.	The backend exports a temporary case, copies it to the board, runs `hps_cpu_reference` and `hps_mmio_predict`, then parses their text output.
Input visualization	Show the original image and the exact 64×64 model input.	The backend returns a base64 preview of the final INT8 image that was packed into the FPGA scratchpad.
CPU confidence report	Show top classes, normalized confidence, raw channel scores, and margin.	The host TFLite model scores the same INT8 input and reports whether the top classes are close or tied.
FPGA profile report	Show per-stage cycle counts for L1, L2 passes, L3 passes, FC, argmax, and total cycles.	The HPS reads the MMIO `PROFILE_*` registers after inference and converts cycle counts to time using the configured 50 MHz fabric clock.
Built-in sanity suite	Run `digit_0_test` through `digit_9_test` from the repository and show pass/fail rows.	The server repeatedly invokes the same web comparison path on known sample images to check model/software/FPGA consistency.
Correction capture	Save a mispredicted input with the selected correct label and an optional note.	The Flask app stores the raw image and metadata under `web_demo/feedback_samples` as debugging evidence for the demonstrated system.

This interface is useful during bring-up because it puts several independent signals on one page: the uploaded image, the actual model input, the HPS CPU prediction, the FPGA prediction, the

HPS-observed accelerator wait time, the pure RTL cycle-counter time, and interface error flags. A mismatch between CPU and FPGA predictions usually points to a packing, preload, or MMIO replay problem, while a high `IF_ERROR` value points to command-ordering or scratchpad-range issues. The page therefore works as both a live demo and a debug dashboard for the hardware/software boundary.

7.2 Board Evidence

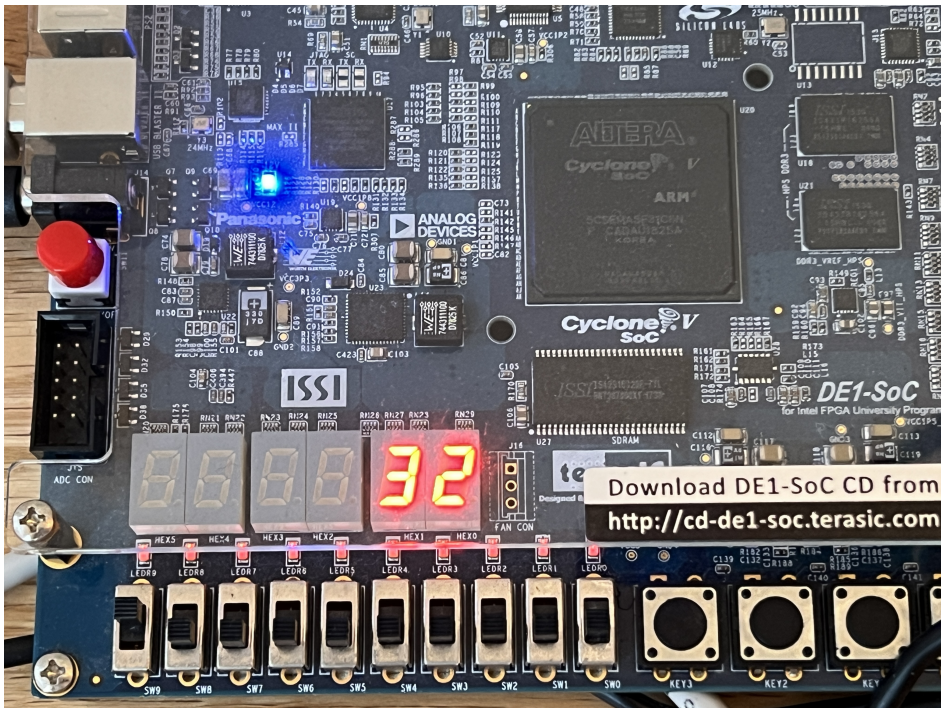


Figure 20: Status of MMIO and Results on Board

The DE1-SoC board also mirrors a small amount of accelerator state on the physical seven-segment displays. This is only a debug readout; the authoritative software result is still read from the MMIO `STATUS`, `PREDICT`, `IF_ERROR`, and `PROFILE_*` registers. The display is useful during demonstrations because it gives an immediate visual indication that the model was loaded and that inference completed.

The two main digits after a normal inference are `HEX1` and `HEX0`. `HEX0` shows the predicted gesture class, and remains blank until the accelerator asserts `predict_done`. `HEX1` shows a compact status code formed from the two debug bits `model_loaded` and `predict_done`.

Table 13: Meaning of the on-board seven-segment debug digits.

Display	Value source	Meaning
HEX0	predict_class[3:0]	Final predicted gesture ID, 0–9, shown after predict_done=1.
HEX1	{model_loaded, predict_done}	Status code: 0 = idle, 2 = model loaded, 3 = model loaded and inference done.
HEX2	interface_error[3:0]	Lowest 4 bits of the interface error code, shown only if an interface error is latched.
HEX5	constant E	Error marker, shown only if interface_error != 0.

For example, after a successful inference on a digit-2 input, the board shows 3 on HEX1 and 2 on HEX0. The 3 means that both model_loaded and predict_done are high, while the 2 is the accelerator's predicted class. If an interface error occurs, HEX5 displays E and HEX2 displays the low nibble of the error word.

8 Team Contributions

Table 14 lists the main implementation and integration responsibilities for the project. The work was collaborative, but the table identifies the areas where each member took the lead.

Table 14: Team contribution summary.

Team member	Main area	Work completed
Cheng-wen Chu	Quantization and pooling RTL	From theory to RTL for Quantization and pooling. Implemented the INT8 requantization path after convolution, including multiplier/shift handling, bias application, saturation back to INT8, and the pooling datapath. Helped verify the convolution-to-quantization-to-pooling streaming behavior against the hardware-aligned reference outputs.
Chengcheng Xu	SRAM subsystem	Implemented and debugged the SRAM-related hardware, including `SRAM_A`, `SRAM_B`, `SRAM_FCW`, address generation, read/write control, FC-weight packing support, activation-buffer reuse, and the memory layout used across the three convolution layers and FC stage.
Harvey Lu	HW/SW interface, system flow and integration	Implemented the board-facing software and hardware/-software interface flow: the MMIO access utilities, scratchpad programming, register-map usage, model-load replay, image-case replay, readback, and host-side SSH/SCP orchestration. Implemented a C-language version CNN model for timing measurement. Also integrated the Flask web demo with the HPS tools, brought up the full host-to-board-to-FPGA path, debugged interface and packing issues, and improved the runtime flow used for final validation.
Linxiao Wu	Golden model and accelerator RTL	Built the training/export lane for the compact CNN model and the hardware-aligned golden module, including reference tensors and SRAM preload artifacts. Also implemented and integrated major accelerator RTL pieces, including layer buffering, FC datapath support, systolic-array-related modules, top-level hardware integration, debug support, slides, and report documentation.
Mingyuan Zheng	Systolic-array, CNN Implementation on Board, and demo UI	Contributed to systolic-array design and simulation. Implemented the C language version CNN model for timing measurement and designed the user-facing web-demo layout used to present image upload, board execution, and prediction results during the final demonstration.

References

[1] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. A. Patterson, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[2] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 32–37.

A Web-To-Board Runtime Listings

This appendix lists the source files directly used by the implemented web-to-board demonstration path and the hardware-aligned golden model used to verify it. It focuses on the Python host flow, the HPS C commands executed on the DE1-SoC, the Verilog/SystemVerilog accelerator hierarchy, and the MATLAB / PyTorch resources that generate the reference tensors and SRAM preload bundles. Generated Quartus output, generated Platform Designer output, waveform/debug dumps, collected web feedback samples, and git history are intentionally excluded from both the printed listings and the machine-readable submission archive. Auxiliary board bring-up scripts, raw register debug tools, standalone batch runners, and alternate dataset-ingestion utilities are also excluded from the printed listings because they are not required for the final web-to-board runtime or the golden-reference verification path.

A.1 Focused Source Tree

```
code/
|-- web_demo/
|   |-- app.py
|   |-- config.py
|   |-- static/
|   |   |-- app.js
|   |   `-- styles.css
|   `-- templates/
|       `-- index.html
|-- gesture_runtime/
|   |-- board_transport.py
|   |-- case_export.py
|   |-- cpu_inference.py
|   |-- demo_compare.py
|   |-- fpga_service.py
|   |-- preprocess.py
|   |-- preprocess_selection.py
|   |-- quantization.py
|   `-- ssh_transport.py
|-- include/
|   |-- cnn_mmio_host.h
|   `-- cnn_mmio_regs.h
|-- tools/
|   |-- cnn_mmio_host.c
```

```
|   |-- hps_cpu_reference.c
|   |-- hps_mmio_load_model.c
|   |-- hps_mmio_predict.c
|   `-- hps_mmio_status.c
|-- platform_designer/
|   `-- cnn_mmio_interface_hw.tcl
|-- de1_soc/
|   |-- build_soc_system.py
|   |-- soc_system_project.tcl
|   |-- soc_system.qsys
|   |-- soc_system.sdc
|   `-- soc_system_top.sv
|-- Golden-Module/
|   |-- models/
|   |   |-- v1.int8.tflite
|   |   `-- v1.int8.params.mat
|   |-- matlab/
|   |   |-- digit_0_test.png ... digit_9_test.png
|   |   `-- hardware_aligned/
|   |       |-- run_all.m
|   |       |-- export_case.m
|   |       |-- hw_forward.m
|   |       |-- gen_sram_preload.m
|   |       |-- load_params.m
|   |       |-- dump_txt.m
|   |       |-- flatten_nhwc_int8.m
|   |       |-- maxpool2x2_int8.m
|   |       |-- tflite_quantize_multiplier.m
|   |       `-- debug/
|   |           |-- sram_preload/
|   |           |   `-- digit_0_test/ ... digit_9_test/
|   |           `-- txt_cases/
|   |               `-- digit_0_test/ ... digit_9_test/
|   `-- pytorch/
|       |-- export_tflite_params_mat.py
|       `-- retrain_digits_cnn.py
`-- input/RTL/
    |-- system_top.v
    |-- interface/
```

```
|   `-- cnn_mmio_interface.v
|-- fc/
|   |-- FC.v
|   |-- fc_bias_loader.v
|   |-- fc_data_adapter.v
|   `-- mac.v
|-- fsm/
|   |-- conv_data_adapter.v
|   |-- layer_runner_fsm.v
|   |-- top_fsm.v
|   `-- wt_prepad_inserter.v
|-- conv_core/
|   |-- Conv_Buffer.v
|   |-- Line_Buffer.v
|   |-- conv_engine_ctrl.v
|   |-- conv_quant_pool.v
|   |-- conv_top.v
|   |-- input_row_aligner.v
|   |-- sa_skew_feeder.v
|   |-- systolic_array_top.v
|   `-- weight_buffer.v
|-- quant_pool/
|   |-- integration/
|   |   |-- conv_quant_adapter.v
|   |   |-- quant_param_loader.v
|   |   `-- quant_pool_adapter.v
|   |-- quant/
|   |   |-- Quantization_PE.v
|   |   `-- Quantization_Top.v
|   `-- pool/
|       |-- pool_core.v
|       `-- pool_stream_top.v
`-- SRAM/
    |-- Addr_Gen.v
    |-- fcw_preload_packer.v
    |-- sram_A_controller.v
    |-- sram_A_wrapper.v
    |-- sram_B_controller.v
    |-- sram_B_wrapper.v
```

```
|-- sram_FCW_wrapper.v
|-- top_sram_A.v
`-- top_sram_B.v
```

A.2 Included Resource Overview

Table 15: Source and resource groups printed in this appendix.

Group	Main language / format	Role in the web-to-board path
Web and host runtime	Python / HTML / CSS / JavaScript	Receives browser uploads, preprocesses images, exports board cases, chooses preprocessing mode, runs host CPU comparison, and calls the FPGA board service.
HPS board runtime	C	Maps the FPGA control window through `/dev/mem`, writes preload and image data, triggers model load or inference, polls status, and prints results for the web server.
FPGA board boundary	SystemVerilog / Verilog	Connects the HPS Platform Designer system to the custom MMIO wrapper and passes the replayed model/image stream into the accelerator top-level control logic.
Quartus and Platform Designer setup	Tcl / Qsys / SDC / Python	Contains the custom Platform Designer component descriptor, the Qsys system, and build wrapper scripts needed to regenerate and compile the board project.
Accelerator datapath RTL	Verilog	Implements the convolution engine, quantization/pooling path, fully-connected classifier, SRAM wrappers, and layer-control FSMs driven by the board boundary.
Golden model and export resources	MATLAB / Python / TFLite / MAT / TXT	Stores the quantized model, exported parameter file, hardware-aligned MATLAB golden forward pass, reference tensors, and SRAM preload bundles used by RTL simulation, HPS commands, and web-demo comparison.

Table 16: Golden module resources used by verification and board preload.

Resource	Function
code/Golden-Module/models/v1.int8.tflite	Deployed INT8 TFLite model used by the host-side CPU classifier and as the source for exported hardware parameters.
code/Golden-Module/models/v1.int8.params.mat	Quantized weights, biases, zero-points, scales, multipliers, and shifts imported by the MATLAB hardware-aligned flow.
code/Golden-Module/matlab/hardware_aligned/run_all.m	Batch entry point that regenerates all digit reference cases and preload bundles.
code/Golden-Module/matlab/hardware_aligned/export_case.m	Exports one image into layer-by-layer `tb_*.txt` reference tensors plus a manifest.
code/Golden-Module/matlab/hardware_aligned/hw_forward.m	Bit-exact hardware-style forward pass matching the RTL data ordering and INT8/INT32 arithmetic assumptions.
code/Golden-Module/matlab/hardware_aligned/gen_sram_preload.m	Packs convolution configuration, convolution weights, FC weights, and FC bias into the 32-bit preload streams consumed by the HPS loader.
code/Golden-Module/matlab/hardware_aligned/debug/txt_cases/	Per-digit input tensors and expected intermediate/output tensors used by RTL testbenches and CPU/FPGA comparison.
code/Golden-Module/matlab/hardware_aligned/debug/sram_preload/	Per-digit preload bundles copied to the board before model-load replay.
code/Golden-Module/pytorch/export_tflite_params_mat.py	Converts the TFLite model parameters into the MAT file consumed by the golden MATLAB flow.
code/Golden-Module/pytorch/retrain_digits_cnn.py	Training / retraining script for the compact digit CNN model family.

Table 17: Python files used by the web and host-side runtime.

Source file	Function
code/web_demo/app.py	Flask routes, service construction, upload endpoint, inference endpoint, and JSON response path.
code/web_demo/config.py	Board SSH settings, CSR base address, model paths, label names, and fabric-clock configuration.
code/gesture_runtime/demo_compare.py	End-to-end comparison flow combining preprocessing, host CPU scoring, HPS CPU baseline, FPGA run, and result formatting.
code/gesture_runtime/fpga_service.py	Board orchestration layer that copies cases, loads the model when needed, runs HPS commands, and parses command output.
code/gesture_runtime/ssh_transport.py	SSH/SCP command execution and directory transfer implementation.
code/gesture_runtime/board_transport.py	Transport interface shared by the FPGA service and any concrete board transport.
code/gesture_runtime/case_export.py	Exports the preprocessed tensor into the case directory consumed by board-side commands.
code/gesture_runtime/preprocess.py	Converts uploaded image bytes into the 64-by-64 INT8 input tensor and preview image.
code/gesture_runtime/preprocess_selection.py	Selects the preprocessing branch used by automatic mode.
code/gesture_runtime/quantization.py	INT8 helper routines used by preprocessing and export.
code/gesture_runtime/cpu_inference.py	Host-side TFLite inference used for confidence reporting and preprocessing selection.

Table 18: C files used by the HPS board runtime.

Source file	Function
code/include/cnn_mmio_regs.h	Register ABI shared by C and RTL: offsets, bit fields, control flags, status flags, and scratchpad layout.
code/include/cnn_mmio_host.h	Shared C structs and function declarations for board-side MMIO commands.
code/tools/cnn_mmio_host.c	Common implementation for memory mapping, scratchpad writes, control writes, polling, and profile readback.
code/tools/hps_mmio_status.c	Lightweight health check used by the web service before deciding whether to reload the model.
code/tools/hps_mmio_load_model.c	Loads the preload bundle into the scratchpad and starts model replay through the MMIO control register.
code/tools/hps_mmio_predict.c	Writes one image case, starts FPGA inference, waits for completion, and prints machine-readable prediction output.
code/tools/hps_cpu_reference.c	HPS ARM software baseline used for web-demo comparison against FPGA output.

Table 19: Verilog/SystemVerilog files at the FPGA board boundary.

Source file	Function
`code/de1_soc/soc_system_top.sv`	DE1-SoC board wrapper connecting the generated HPS system, board pins, clocks, resets, and seven-segment display.
`code/input/RTL/interface/cnn_mmio_interface.v`	Avalon-MM slave wrapper containing the software-visible register file, scratchpad, replay FSM, status readback, and profile readback.
`code/input/RTL/system_top.v`	Accelerator top-level connecting model load, CNN layer sequencing, SRAM datapath, FC classifier, argmax, and profiling outputs.
`code/input/RTL/fsm/top_fsm.v`	Network-level FSM for model load and inference scheduling across convolution, FC, and argmax phases.
`code/input/RTL/fsm/layer_runner_fsm.v`	Per-layer FSM that sequences layer configuration, weights, input data, output writes, and completion.

A.3 Web And Host Python Runtime

A.3.1 Flask Web Entry Point

Defines the web routes, service construction, image upload endpoint, and JSON response path.

code/web_demo/app.py

```python
#!/usr/bin/env python3
"""Flask app for the upload-first CPU-vs-FPGA comparison demo."""

from __future__ import annotations

import os
import sys
import json
from datetime import datetime
from pathlib import Path
from uuid import uuid4

from flask import Flask, jsonify, render_template, request

REPO_ROOT = Path(__file__).resolve().parents[1]
if str(REPO_ROOT) not in sys.path:
    sys.path.insert(0, str(REPO_ROOT))

from web_demo.config import load_demo_config
from gesture_runtime.cpu_inference import TFLiteCPUClassifier
from gesture_runtime.demo_compare import ComparisonDemoService, DemoLabels
from gesture_runtime.fpga_service import BoardCPUReferenceService, FPGABoardService, FPGAServiceConfig
from gesture_runtime.ssh_transport import SshBoardTransport


def build_service() -> ComparisonDemoService:
```

```
    cfg = load_demo_config()
    labels = DemoLabels(cfg.labels) if cfg.labels else DemoLabels.default_digits()
    cpu_classifier = TFLiteCPUClassifier(cfg.cpu_model)
    transport = SshBoardTransport(
        host=cfg.board_host,
        user=cfg.board_user,
        port=cfg.board_port,
        identity_file=cfg.ssh_key,
        known_hosts_file=cfg.ssh_known_hosts,
        bind_address=cfg.ssh_bind_address,
    )
    fpga_service = FPGABoardService(
        transport=transport,
        config=FPGAServiceConfig(
            remote_repo=cfg.remote_repo,
            csr_base=cfg.csr_base,
            remote_preload_root=cfg.remote_preload_root,
            remote_reference_case_root=cfg.remote_reference_case_root,
        ),
    )
    board_cpu_service = BoardCPUReferenceService(
        transport=transport,
        config=FPGAServiceConfig(
            remote_repo=cfg.remote_repo,
            csr_base=cfg.csr_base,
            remote_preload_root=cfg.remote_preload_root,
            remote_reference_case_root=cfg.remote_reference_case_root,
        ),
    )
    return ComparisonDemoService(
        cpu_classifier=cpu_classifier,
        board_cpu_service=board_cpu_service,
        fpga_service=fpga_service,
        labels=labels,
        fabric_mhz=cfg.fabric_mhz,
    )


app = Flask(__name__, template_folder="templates", static_folder="static")
SERVICE: ComparisonDemoService | None = None
SAMPLE_IMAGE_DIR = REPO_ROOT / "Golden-Module" / "matlab"
FEEDBACK_DIR = REPO_ROOT / "web_demo" / "feedback_samples"


def get_service() -> ComparisonDemoService:
    global SERVICE
    if SERVICE is None:
        SERVICE = build_service()
    return SERVICE


@app.route("/")
def index():
    return render_template("index.html")
```

```python


@app.get("/api/health")
def health():
    cfg = load_demo_config()
    return jsonify(
        {
            "ok": True,
            "board_host": cfg.board_host,
            "board_user": cfg.board_user,
            "board_port": cfg.board_port,
            "ssh_bind_address": cfg.ssh_bind_address,
            "remote_repo": cfg.remote_repo,
            "remote_preload_root": cfg.remote_preload_root,
            "remote_reference_case_root": cfg.remote_reference_case_root,
            "cpu_model": str(cfg.cpu_model),
            "fabric_mhz": cfg.fabric_mhz,
            "labels": cfg.labels,
            "model_lane": cfg.model_lane,
            "model_warning": cfg.model_warning,
            "task_semantics": "gesture-classification",
            "dataset_name": cfg.dataset_name,
            "dataset_source_hint": cfg.dataset_source_hint,
        }
    )


@app.post("/api/infer")
def infer():
    uploaded = request.files.get("image")
    if uploaded is None or not uploaded.filename:
        return jsonify({"error": "image upload is required"}), 400

    preprocess_mode = request.form.get("preprocess_mode", "auto").strip() or "auto"
    try:
        cfg = load_demo_config()
        result = get_service().compare_image_bytes(uploaded.read(), preferred_mode=preprocess_mode)
    except Exception as exc:  # pragma: no cover - integration path
        return jsonify({"error": str(exc)}), 500
    result["model_lane"] = cfg.model_lane
    result["model_warning"] = cfg.model_warning
    result["task_semantics"] = "gesture-classification"
    result["dataset_name"] = cfg.dataset_name
    result["dataset_source_hint"] = cfg.dataset_source_hint
    return jsonify(result)


@app.post("/api/feedback")
def save_feedback():
    uploaded = request.files.get("image")
    corrected_label = request.form.get("corrected_label", "").strip()
    predicted_label = request.form.get("predicted_label", "").strip()
    preprocess_mode = request.form.get("preprocess_mode", "").strip()
    note = request.form.get("note", "").strip()
```

```
135
136     if uploaded is None or not uploaded.filename:
137         return jsonify({"error": "image upload is required"}), 400
138     if corrected_label not in {str(i) for i in range(10)}:
139         return jsonify({"error": "corrected_label must be one of 0..9"}), 400
140
141     stamp = datetime.now().strftime("%Y%m%d_%H%M%S")
142     sample_id = f"{stamp}_{uuid4().hex[:8]}"
143     sample_dir = FEEDBACK_DIR / sample_id
144     sample_dir.mkdir(parents=True, exist_ok=True)
145
146     filename = Path(uploaded.filename).name or "capture.png"
147     raw_ext = Path(filename).suffix or ".png"
148     raw_path = sample_dir / f"input{raw_ext}"
149     uploaded.save(raw_path)
150
151     metadata = {
152         "id": sample_id,
153         "captured_at": datetime.now().isoformat(),
154         "source_filename": filename,
155         "raw_image": str(raw_path.relative_to(REPO_ROOT)),
156         "corrected_label": corrected_label,
157         "predicted_label": predicted_label,
158         "preprocess_mode": preprocess_mode,
159         "note": note,
160     }
161     metadata_path = sample_dir / "metadata.json"
162     metadata_path.write_text(json.dumps(metadata, indent=2), encoding="utf-8")
163
164     return jsonify(
165         {
166             "ok": True,
167             "sample_id": sample_id,
168             "saved_dir": str(sample_dir.relative_to(REPO_ROOT)),
169             "corrected_label": corrected_label,
170         }
171     )
172
173
174 @app.get("/api/sample-suite")
175 def sample_suite():
176     try:
177         cfg = load_demo_config()
178         sample_paths = sorted(SAMPLE_IMAGE_DIR.glob("digit_*_test.png"))
179         if not sample_paths:
180             return jsonify({"error": f"no sample images found under {SAMPLE_IMAGE_DIR}"}), 500
181
182         rows = []
183         passed = 0
184         service = get_service()
185         for sample_path in sample_paths:
186             stem = sample_path.stem
187             expected = int(stem.split("_")[1])
188             result = service.compare_image_path(sample_path)
```

```
            cpu_ok = int(result["cpu_predicted_class"]) == expected
            fpga_ok = int(result["predicted_class"]) == expected
            both_ok = cpu_ok and fpga_ok
            if both_ok:
                passed += 1
            rows.append(
                {
                    "case": stem,
                    "expected_class": expected,
                    "cpu_predicted_class": int(result["cpu_predicted_class"]),
                    "fpga_predicted_class": int(result["predicted_class"]),
                    "cpu_time_ms": result["cpu_time_ms"],
                    "fpga_time_ms": result["fpga_time_ms"],
                    "preprocess_mode": result.get("preprocess_mode", ""),
                    "cpu_confidence_margin": result.get("cpu_confidence_margin"),
                    "cpu_ok": cpu_ok,
                    "fpga_ok": fpga_ok,
                    "pass": both_ok,
                }
            )

        return jsonify(
            {
                "ok": True,
                "dataset_name": cfg.dataset_name,
                "model_lane": cfg.model_lane,
                "task_semantics": "gesture-classification",
                "total": len(rows),
                "passed": passed,
                "rows": rows,
            }
        )
    except Exception as exc:  # pragma: no cover - integration path
        return jsonify({"error": str(exc)}), 500


if __name__ == "__main__":
    cfg = load_demo_config()
    app.run(host=cfg.local_host, port=cfg.local_port, debug=cfg.debug)
```

A.3.2 Web Demo Configuration

Loads board SSH settings, CSR base address, model paths, labels, and fabric-clock configuration.

code/web_demo/config.py

```
"""Configuration loading for the web comparison demo."""

from __future__ import annotations

import json
```

```python
import os
from dataclasses import dataclass
from pathlib import Path
from typing import List, Optional


REPO_ROOT = Path(__file__).resolve().parents[1]
DEFAULT_MODEL = REPO_ROOT / "Golden-Module" / "models" / "v1.int8.tflite"
DEFAULT_CONFIG_PATH = Path(
    os.environ.get("CNN_ACC_DEMO_CONFIG", str(REPO_ROOT / "web_demo" / "demo_config.json"))
)


@dataclass
class DemoConfig:
    cpu_model: Path
    labels: List[str]
    board_host: str
    board_user: str
    board_port: int
    ssh_key: Optional[Path]
    ssh_known_hosts: Optional[Path]
    ssh_bind_address: Optional[str]
    remote_repo: str
    remote_preload_root: str
    remote_reference_case_root: str
    csr_base: str
    fabric_mhz: float
    local_host: str
    local_port: int
    debug: bool

    @property
    def model_lane(self) -> str:
        preload = self.remote_preload_root.lower()
        labels_are_numeric = self.labels and all(label.isdigit() for label in self.labels)
        if "digit_" in preload or labels_are_numeric:
            return "gesture-10class"
        if any(token in preload for token in ("paper", "rock", "scissors", "gesture")):
            return "gesture"
        return "unknown"

    @property
    def model_warning(self) -> Optional[str]:
        if self.model_lane == "unknown":
            return (
                "Current demo config does not clearly map to a gesture-specific deployed model. "
                "Predictions may not match the intended gesture task."
            )
        return None

    @property
    def dataset_name(self) -> str:
        if self.model_lane == "gesture-10class":
```

```
            return "Sign Language Digits Dataset"
        if self.model_lane == "gesture":
            return "Gesture dataset"
        return "Unknown dataset"

    @property
    def dataset_source_hint(self) -> Optional[str]:
        if self.model_lane == "gesture-10class":
            return "Kaggle sign-language digits dataset family; local training root expected as dataset_wlx/"
        return None


def _read_json_config(path: Path) -> dict:
    if not path.is_file():
        return {}
    return json.loads(path.read_text(encoding="utf-8"))


def _resolve_repo_path(value: Optional[str]) -> Optional[Path]:
    if not value:
        return None
    raw = Path(value)
    if raw.is_absolute():
        return raw
    return (REPO_ROOT / raw).resolve()


def load_demo_config(path: Optional[Path] = None) -> DemoConfig:
    cfg_path = path or DEFAULT_CONFIG_PATH
    raw = _read_json_config(cfg_path)

    labels = raw.get("labels") or os.environ.get("CNN_ACC_LABELS", "")
    label_list = labels if isinstance(labels, list) else [x.strip() for x in labels.split(",") if x.strip()]
    if not label_list:
        label_list = [str(i) for i in range(10)]

    ssh_key = raw.get("ssh_key") or os.environ.get("CNN_ACC_SSH_KEY")
    ssh_known_hosts = raw.get("ssh_known_hosts") or os.environ.get("CNN_ACC_SSH_KNOWN_HOSTS")
    ssh_bind_address = raw.get("ssh_bind_address") or os.environ.get("CNN_ACC_SSH_BIND_ADDRESS")

    return DemoConfig(
        cpu_model=_resolve_repo_path(
            raw.get("cpu_model") or os.environ.get("CNN_ACC_CPU_MODEL", str(DEFAULT_MODEL))
        )
        or DEFAULT_MODEL,
        labels=label_list,
        board_host=raw.get("board_host") or os.environ.get("CNN_ACC_BOARD_HOST", "192.168.0.2"),
        board_user=raw.get("board_user") or os.environ.get("CNN_ACC_BOARD_USER", "root"),
        board_port=int(raw.get("board_port") or os.environ.get("CNN_ACC_BOARD_PORT", "22")),
        ssh_key=_resolve_repo_path(ssh_key),
        ssh_known_hosts=_resolve_repo_path(ssh_known_hosts),
        ssh_bind_address=ssh_bind_address or None,
```

```
        remote_repo=raw.get("remote_repo") or os.environ.get("CNN_ACC_REMOTE_REPO", "/root/cnn_acc_hps"),
        remote_preload_root=raw.get("remote_preload_root")
        or os.environ.get(
            "CNN_ACC_REMOTE_PRELOAD_ROOT",
            "Golden-Module/matlab/hardware_aligned/debug/sram_preload/digit_0_test",
        ),
        remote_reference_case_root=raw.get("remote_reference_case_root")
        or os.environ.get(
            "CNN_ACC_REMOTE_REFERENCE_CASE_ROOT",
            "Golden-Module/matlab/hardware_aligned/debug/txt_cases/digit_0_test",
        ),
        csr_base=raw.get("csr_base") or os.environ.get("CNN_ACC_CSR_BASE", "0xff200000"),
        fabric_mhz=float(raw.get("fabric_mhz") or os.environ.get("CNN_ACC_FABRIC_MHZ", "50")),
        local_host=raw.get("local_host") or os.environ.get("HOST", "127.0.0.1"),
        local_port=int(raw.get("local_port") or os.environ.get("PORT", "5000")),
        debug=bool(raw.get("debug", os.environ.get("FLASK_DEBUG", "1") not in ("0", "false", "False"))),
    )
```

A.3.3 CPU-vs-FPGA Comparison Flow

Coordinates preprocessing, CPU scoring, board CPU reference execution, FPGA execution, and result formatting.

code/gesture_runtime/demo_compare.py

```python
"""Unified CPU-vs-FPGA comparison flow for the web demo."""

from __future__ import annotations

import math
import tempfile
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List

from .case_export import write_inference_case
from .cpu_inference import TFLiteCPUClassifier
from .fpga_service import BoardCPUReferenceService, FPGABoardService
from .preprocess import preprocess_image_bytes_variants
from .preprocess_selection import choose_best_variant


@dataclass
class DemoLabels:
    labels: List[str]

    @classmethod
    def default_digits(cls) -> "DemoLabels":
        return cls(labels=[str(i) for i in range(10)])

```

```
    def label_for(self, idx: int) -> str:
        if 0 <= idx < len(self.labels):
            return self.labels[idx]
        return str(idx)


class ComparisonDemoService:
    def __init__(
        self,
        cpu_classifier: TFLiteCPUClassifier,
        board_cpu_service: BoardCPUReferenceService,
        fpga_service: FPGABoardService,
        labels: DemoLabels,
        fabric_mhz: float = 25.0,
    ):
        self.cpu_classifier = cpu_classifier
        self.board_cpu_service = board_cpu_service
        self.fpga_service = fpga_service
        self.labels = labels
        self.fabric_mhz = fabric_mhz

    def _cycles_to_us(self, cycles: int) -> float:
        if self.fabric_mhz <= 0:
            return 0.0
        return float(cycles) / self.fabric_mhz

    def _score_variants(self, variants):
        scored = []
        for variant in variants:
            cpu_result = self.cpu_classifier.predict_int8_image(variant.values)
            scored.append((variant, cpu_result))
        return scored

    def _softmax_probabilities(self, scores: List[float]) -> List[float]:
        if not scores:
            return []
        max_score = max(scores)
        exp_scores = [math.exp(score - max_score) for score in scores]
        total = sum(exp_scores)
        if total <= 0:
            return [0.0 for _ in scores]
        return [value / total for value in exp_scores]

    def _build_cpu_score_report(self, cpu_result) -> Dict[str, object]:
        probabilities = self._softmax_probabilities(cpu_result.scores)
        channels = []
        for idx, (score, probability) in enumerate(zip(cpu_result.scores, probabilities)):
            channels.append(
                {
                    "class_id": idx,
                    "label": self.labels.label_for(idx),
                    "raw_score": float(score),
                    "probability": float(probability),
                    "probability_pct": float(probability * 100.0),
```

```
                }
            )

        top_channels = sorted(channels, key=lambda row: row["raw_score"], reverse=True)[:3]
        note = ""
        if len(top_channels) >= 2 and abs(top_channels[0]["raw_score"] - top_channels[1]["raw_score"]) <
    1e-6:
            note = "Top classes are tied, so this sample is currently low-confidence."
        elif float(cpu_result.margin) < 5.0:
            note = "Top classes are close, so this sample is currently low-confidence."

        return {
            "predicted_class": int(cpu_result.predicted_class),
            "predicted_label": self.labels.label_for(int(cpu_result.predicted_class)),
            "top_score": float(cpu_result.top_score),
            "margin": float(cpu_result.margin),
            "confidence": float(top_channels[0]["probability"]) if top_channels else 0.0,
            "confidence_pct": float(top_channels[0]["probability_pct"]) if top_channels else 0.0,
            "top_channels": top_channels,
            "channels": channels,
            "note": note,
        }

    def _build_branch_diagnostics(self, scored) -> List[Dict[str, object]]:
        diagnostics = []
        for variant, cpu_result in scored:
            top_channels = self._build_cpu_score_report(cpu_result)["top_channels"]
            diagnostics.append(
                {
                    "mode": variant.mode,
                    "predicted_class": int(cpu_result.predicted_class),
                    "predicted_label": self.labels.label_for(int(cpu_result.predicted_class)),
                    "margin": float(cpu_result.margin),
                    "top_score": float(cpu_result.top_score),
                    "top_channels": top_channels,
                }
            )
        return diagnostics

    def _build_fpga_profile(self, fpga_result: Dict[str, object]) -> Dict[str, object]:
        raw_profile = fpga_result.get("profile", {})
        stage_specs = [
            ("L1", "l1_cycles"),
            ("L2 P0", "l2_p0_cycles"),
            ("L2 P1", "l2_p1_cycles"),
            ("L3 P0", "l3_p0_cycles"),
            ("L3 P1", "l3_p1_cycles"),
            ("FC", "fc_cycles"),
            ("Argmax", "argmax_cycles"),
        ]
        stages = []
        for label, key in stage_specs:
            cycles = int(raw_profile.get(key, 0))
            stages.append(
```

```
                {
                    "label": label,
                    "cycles": cycles,
                    "time_us": self._cycles_to_us(cycles),
                }
            )
        total_cycles = int(raw_profile.get("total_cycles", 0))
        return {
            "fabric_mhz": self.fabric_mhz,
            "total_cycles": total_cycles,
            "rtl_time_us": self._cycles_to_us(total_cycles),
            "rtl_time_ms": self._cycles_to_us(total_cycles) / 1000.0,
            "stages": stages,
        }

    def _compare_preprocessed_image(self, image_values: List[int], preview_b64: str) -> Dict[str, object
    ]:
        cpu_result = self.cpu_classifier.predict_int8_image(image_values)

        with tempfile.TemporaryDirectory(prefix="cnn_acc_case_") as tmp_dir:
            case_root = Path(tmp_dir) / "gesture_upload_case"
            write_inference_case(case_root, image_values, case_name="gesture_upload", expected_class=-1)
            board_cpu_result = self.board_cpu_service.predict_case_dir(case_root)
            fpga_result = self.fpga_service.predict_case_dir(case_root)

        cpu_ms = float(board_cpu_result["board_cpu_infer_ms"])
        fpga_kernel_ms = float(fpga_result["board_infer_wait_ms"])
        fpga_request_ms = float(fpga_result["request_elapsed_ms"])
        fpga_profile = self._build_fpga_profile(fpga_result)
        speedup = (cpu_ms / fpga_kernel_ms) if fpga_kernel_ms > 0 else None

        cpu_class = int(board_cpu_result["predict_class"])
        fpga_class = int(fpga_result["predict_class"])
        return {
            "predicted_class": fpga_class,
            "predicted_label": self.labels.label_for(fpga_class),
            "cpu_predicted_class": cpu_class,
            "cpu_predicted_label": self.labels.label_for(cpu_class),
            "cpu_time_ms": cpu_ms,
            "fpga_time_ms": fpga_kernel_ms,
            "fpga_rtl_time_ms": fpga_profile["rtl_time_ms"],
            "fpga_rtl_time_us": fpga_profile["rtl_time_us"],
            "fpga_request_time_ms": fpga_request_ms,
            "fpga_board_total_time_ms": float(fpga_result["board_total_ms"]),
            "fpga_board_program_time_ms": float(fpga_result["board_program_ms"]),
            "fpga_board_case_load_time_ms": float(fpga_result["board_case_load_ms"]),
            "speedup": speedup,
            "fpga_status": fpga_result["status"],
            "fpga_error": fpga_result["error"],
            "chart": {
                "labels": ["CPU", "FPGA wait"],
                "values_ms": [cpu_ms, fpga_kernel_ms],
            },
            "fpga_profile": fpga_profile,
```

```
            "preprocessed_preview_url": f"{preview_b64}",
        }

    def compare_image_bytes(self, image_bytes: bytes, preferred_mode: str = "auto") -> Dict[str, object]:
        variants = preprocess_image_bytes_variants(image_bytes)
        scored = self._score_variants(variants)

        if preferred_mode != "auto":
            matches = [(variant, cpu_result) for variant, cpu_result in scored if variant.mode ==
 preferred_mode]
            if not matches:
                raise ValueError(f"unknown preprocess mode: {preferred_mode}")
            best_variant, best_cpu_result = matches[0]
        else:
            best_variant, best_cpu_result = choose_best_variant(scored)

        cpu_score_report = self._build_cpu_score_report(best_cpu_result)

        with tempfile.TemporaryDirectory(prefix="cnn_acc_case_") as tmp_dir:
            case_root = Path(tmp_dir) / "gesture_upload_case"
            write_inference_case(case_root, best_variant.values, case_name="gesture_upload",
 expected_class=-1)
            board_cpu_result = self.board_cpu_service.predict_case_dir(case_root)
            fpga_result = self.fpga_service.predict_case_dir(case_root)

        cpu_ms = float(board_cpu_result["board_cpu_infer_ms"])
        fpga_kernel_ms = float(fpga_result["board_infer_wait_ms"])
        fpga_request_ms = float(fpga_result["request_elapsed_ms"])
        fpga_profile = self._build_fpga_profile(fpga_result)
        speedup = (cpu_ms / fpga_kernel_ms) if fpga_kernel_ms > 0 else None

        cpu_class = int(board_cpu_result["predict_class"])
        fpga_class = int(fpga_result["predict_class"])
        return {
            "predicted_class": fpga_class,
            "predicted_label": self.labels.label_for(fpga_class),
            "cpu_predicted_class": cpu_class,
            "cpu_predicted_label": self.labels.label_for(cpu_class),
            "cpu_time_ms": cpu_ms,
            "fpga_time_ms": fpga_kernel_ms,
            "fpga_rtl_time_ms": fpga_profile["rtl_time_ms"],
            "fpga_rtl_time_us": fpga_profile["rtl_time_us"],
            "fpga_request_time_ms": fpga_request_ms,
            "fpga_board_total_time_ms": float(fpga_result["board_total_ms"]),
            "fpga_board_program_time_ms": float(fpga_result["board_program_ms"]),
            "fpga_board_case_load_time_ms": float(fpga_result["board_case_load_ms"]),
            "speedup": speedup,
            "fpga_status": fpga_result["status"],
            "fpga_error": fpga_result["error"],
            "chart": {
                "labels": ["CPU", "FPGA wait"],
                "values_ms": [cpu_ms, fpga_kernel_ms],
            },
            "fpga_profile": fpga_profile,
```

```
            "branch_diagnostics": self._build_branch_diagnostics(scored),
            "preprocessed_preview_url": f"{best_variant.preview_b64}",
            "preprocess_mode": best_variant.mode,
            "cpu_confidence_margin": best_cpu_result.margin,
            "cpu_score_report": cpu_score_report,
        }

    def compare_image_path(self, image_path: Path) -> Dict[str, object]:
        image_bytes = Path(image_path).read_bytes()
        return self.compare_image_bytes(image_bytes)
```

A.3.4 FPGA Board Service

Uses SSH/SCP to load the model when needed, copy an exported case to the board, run HPS tools, and parse key-value output.

code/gesture_runtime/fpga_service.py

```
"""Board-side FPGA inference service wrappers."""

from __future__ import annotations

import shlex
import tempfile
from dataclasses import dataclass
from pathlib import Path
from time import perf_counter
from typing import Dict

from .board_transport import BoardTransport


def parse_key_value_output(text: str) -> Dict[str, str]:
    result: Dict[str, str] = {}
    for raw_line in text.splitlines():
        line = raw_line.strip()
        if not line or "=" not in line:
            continue
        key, value = line.split("=", 1)
        result[key.strip()] = value.strip()
    return result


@dataclass
class FPGAServiceConfig:
    remote_repo: str = "/root/cnn_acc_hps"
    csr_base: str = "0xff200000"
    remote_preload_root: str = "Golden-Module/matlab/hardware_aligned/debug/sram_preload/digit_0_test"
    remote_reference_case_root: str = "Golden-Module/matlab/hardware_aligned/debug/txt_cases/digit_0_test
    "
```

```python
    remote_case_parent: str = "/tmp"


class FPGABoardService:
    def __init__(self, transport: BoardTransport, config: FPGAServiceConfig):
        self.transport = transport
        self.config = config
        self._model_loaded = False

    def _read_status(self) -> Dict[str, str]:
        cmd = (
            f"cd {shlex.quote(self.config.remote_repo)} && "
            f"./tools/hps_mmio_status {shlex.quote(self.config.csr_base)}"
        )
        output = self.transport.run(cmd, timeout_s=20.0)
        return parse_key_value_output(output)

    def ensure_model_loaded(self, force: bool = False) -> None:
        if not force and self._model_loaded:
            try:
                status = self._read_status()
                if status.get("model_loaded") == "1":
                    return
            except Exception:
                # Fall back to reloading the model if a status probe fails.
                pass
        cmd = (
            f"cd {shlex.quote(self.config.remote_repo)} && "
            f"./tools/hps_mmio_load_model {shlex.quote(self.config.csr_base)} "
            f"{shlex.quote(self.config.remote_preload_root)}"
        )
        self.transport.run(cmd, timeout_s=60.0)
        self._model_loaded = True

    def predict_case_dir(self, local_case_dir: Path) -> Dict[str, object]:
        self.ensure_model_loaded()
        self.transport.put_dir(local_case_dir, self.config.remote_case_parent, timeout_s=60.0)

        remote_case_dir = f"{self.config.remote_case_parent}/{local_case_dir.name}"
        cmd = (
            f"cd {shlex.quote(self.config.remote_repo)} && "
            f"./tools/hps_mmio_predict {shlex.quote(self.config.csr_base)} "
            f"{shlex.quote(remote_case_dir)}"
        )

        started = perf_counter()
        try:
            output = self.transport.run(cmd, timeout_s=60.0)
        except Exception as exc:
            message = str(exc)
            if "timeout waiting for predict_done" not in message:
                raise
            self.ensure_model_loaded(force=True)
            output = self.transport.run(cmd, timeout_s=60.0)
```

```
        elapsed_ms = (perf_counter() - started) * 1000.0
        parsed = parse_key_value_output(output)
        profile = {
            "l1_cycles": int(parsed.get("l1_cycles", "0") or 0),
            "l2_p0_cycles": int(parsed.get("l2_p0_cycles", "0") or 0),
            "l2_p1_cycles": int(parsed.get("l2_p1_cycles", "0") or 0),
            "l3_p0_cycles": int(parsed.get("l3_p0_cycles", "0") or 0),
            "l3_p1_cycles": int(parsed.get("l3_p1_cycles", "0") or 0),
            "fc_cycles": int(parsed.get("fc_cycles", "0") or 0),
            "argmax_cycles": int(parsed.get("argmax_cycles", "0") or 0),
            "total_cycles": int(parsed.get("total_cycles", "0") or 0),
        }
        return {
            "predict_class": int(parsed.get("predict_class", "0")),
            "status": parsed.get("status", ""),
            "error": parsed.get("error", ""),
            "request_elapsed_ms": elapsed_ms,
            "board_case_load_ms": float(parsed.get("board_case_load_ms", "0") or 0),
            "board_program_ms": float(parsed.get("board_program_ms", "0") or 0),
            "board_infer_wait_ms": float(parsed.get("board_infer_wait_ms", "0") or 0),
            "board_total_ms": float(parsed.get("board_total_ms", "0") or 0),
            "profile": profile,
            "raw_output": output,
        }


class BoardCPUReferenceService:
    def __init__(self, transport: BoardTransport, config: FPGAServiceConfig):
        self.transport = transport
        self.config = config

    def predict_case_dir(self, local_case_dir: Path) -> Dict[str, object]:
        self.transport.put_dir(local_case_dir, self.config.remote_case_parent, timeout_s=60.0)

        remote_case_dir = f"{self.config.remote_case_parent}/{local_case_dir.name}"
        cmd = (
            f"cd {shlex.quote(self.config.remote_repo)} && "
            f"./tools/hps_cpu_reference {shlex.quote(remote_case_dir)} "
            f"{shlex.quote(self.config.remote_reference_case_root)}"
        )
        output = self.transport.run(cmd, timeout_s=60.0)
        parsed = parse_key_value_output(output)
        return {
            "predict_class": int(parsed.get("predict_class", "0")),
            "board_cpu_infer_ms": float(parsed.get("board_cpu_infer_ms", "0") or 0),
            "raw_output": output,
        }
```

A.3.5 SSH Transport

Implements the command execution and directory transfer layer used by the board service.

code/gesture_runtime/ssh_transport.py

```python
"""SSH transport for host-to-board execution."""

from __future__ import annotations

import subprocess
from pathlib import Path
from typing import List, Optional

from .board_transport import BoardTransport, TransportError


class SshBoardTransport(BoardTransport):
    def __init__(
        self,
        host: str,
        user: str = "root",
        port: int = 22,
        identity_file: Optional[Path] = None,
        known_hosts_file: Optional[Path] = None,
        bind_address: Optional[str] = None,
    ):
        self.host = host
        self.user = user
        self.port = port
        self.identity_file = Path(identity_file) if identity_file else None
        self.known_hosts_file = Path(known_hosts_file) if known_hosts_file else None
        self.bind_address = bind_address

    @property
    def target(self) -> str:
        return f"{self.user}@{self.host}"

    def _base_args(self) -> List[str]:
        args = [
            "-p",
            str(self.port),
            "-o",
            "StrictHostKeyChecking=no",
        ]
        if self.identity_file:
            args.extend(["-i", str(self.identity_file)])
        if self.known_hosts_file:
            args.extend(["-o", f"UserKnownHostsFile={self.known_hosts_file}"])
        if self.bind_address:
            args.extend(["-o", f"BindAddress={self.bind_address}"])
        return args

    def run(self, command: str, timeout_s: float = 30.0) -> str:
        proc = subprocess.run(
            ["ssh", *self._base_args(), self.target, command],
            capture_output=True,
            text=True,
```

```
            encoding="utf-8",
            errors="replace",
            timeout=timeout_s,
        )
        if proc.returncode != 0:
            raise TransportError(
                "ssh command failed\n"
                f"command: {command}\n"
                f"stdout:\n{proc.stdout}\n"
                f"stderr:\n{proc.stderr}"
            )
        return proc.stdout

    def put_dir(self, local_dir: Path, remote_parent: str, timeout_s: float = 30.0) -> None:
        proc = subprocess.run(
            [
                "scp",
                "-r",
                "-P",
                str(self.port),
                "-o",
                "StrictHostKeyChecking=no",
                *(["-i", str(self.identity_file)] if self.identity_file else []),
                *(["-o", f"UserKnownHostsFile={self.known_hosts_file}"] if self.known_hosts_file else [])
,
                *(["-o", f"BindAddress={self.bind_address}"] if self.bind_address else []),
                str(local_dir),
                f"{self.target}:{remote_parent}",
            ],
            capture_output=True,
            text=True,
            encoding="utf-8",
            errors="replace",
            timeout=timeout_s,
        )
        if proc.returncode != 0:
            raise TransportError(
                "scp failed\n"
                f"local_dir: {local_dir}\n"
                f"remote_parent: {remote_parent}\n"
                f"stdout:\n{proc.stdout}\n"
                f"stderr:\n{proc.stderr}"
            )
```

A.3.6 Board Transport Interface

Defines the abstract transport contract shared by SSH and other possible board transports.

code/gesture_runtime/board_transport.py

```python
"""Transport abstraction for host-to-board execution."""

from __future__ import annotations

from abc import ABC, abstractmethod
from pathlib import Path
from typing import Optional


class BoardTransport(ABC):
    @abstractmethod
    def run(self, command: str, timeout_s: float = 30.0) -> str:
        raise NotImplementedError

    @abstractmethod
    def put_dir(self, local_dir: Path, remote_parent: str, timeout_s: float = 30.0) -> None:
        raise NotImplementedError


class TransportError(RuntimeError):
    pass
```

A.3.7 Inference Case Export

Writes a preprocessed image into the hardware-aligned case directory consumed by the HPS tools.

code/gesture_runtime/case_export.py

```python
"""Helpers for exporting one uploaded image into a board-consumable case folder."""

from pathlib import Path
from typing import Iterable

from .preprocess import write_txt_int8_image


def write_inference_case(
    case_root: Path,
    image_values: Iterable[int],
    case_name: str = "gesture_upload",
    expected_class: int = -1,
) -> Path:
    case_root.mkdir(parents=True, exist_ok=True)

    image_path = case_root / "tb_conv1_in_i8_64x64x1.txt"
    manifest_path = case_root / "manifest.txt"

    write_txt_int8_image(image_path, image_values)

```

```
    manifest_lines = [
        f"case={case_name}",
        "image_file=uploaded_image",
        f"conv1_input={image_path.name}",
        f"predict_class={expected_class}",
    ]
    manifest_path.write_text("\n".join(manifest_lines) + "\n", encoding="utf-8")
    return case_root
```

A.3.8 Image Preprocessing

Converts uploaded image bytes into the 64-by-64 INT8 input tensor and preview image used by the demo.

code/gesture_runtime/preprocess.py

```
"""Input preprocessing helpers.

The current RTL expects a quantized 64x64x1 INT8 tensor. For hardware-aligned
tests we support direct TXT loading; for image files we expose Pillow-based
helpers that work both from on-disk paths and in-memory upload bytes.
"""

import base64
from dataclasses import dataclass
from io import BytesIO
from pathlib import Path
from typing import Iterable, List

from .quantization import quantize_u8_to_i8


@dataclass
class PreprocessedImage:
    mode: str
    values: List[int]
    preview_b64: str


def load_txt_int8_image(path: Path) -> List[int]:
    with path.open("r", encoding="utf-8") as fp:
        return [int(line.strip()) for line in fp if line.strip()]


def _encode_image_b64(image) -> str:
    buf = BytesIO()
    image.save(buf, format="PNG")
    return base64.b64encode(buf.getvalue()).decode("ascii")


```

```python
def _otsu_threshold(arr) -> int:
    import numpy as np

    hist = np.bincount(arr.reshape(-1), minlength=256).astype(np.float64)
    total = arr.size
    cumulative = np.cumsum(hist)
    cumulative_mean = np.cumsum(hist * np.arange(256))
    global_mean = cumulative_mean[-1] / max(total, 1)

    numerator = (global_mean * cumulative - cumulative_mean) ** 2
    denominator = cumulative * (total - cumulative)
    with np.errstate(divide="ignore", invalid="ignore"):
        score = numerator / denominator
    score[~np.isfinite(score)] = 0.0
    return int(score.argmax())


def _select_foreground_mask(arr):
    import numpy as np

    threshold = _otsu_threshold(arr)
    dark_mask = arr <= threshold
    light_mask = arr >= threshold
    border = np.concatenate([arr[0, :], arr[-1, :], arr[:, 0], arr[:, -1]])
    bg_hint = int(np.median(border))

    def score(mask):
        area = float(mask.mean())
        border_frac = float(
            np.concatenate([mask[0, :], mask[-1, :], mask[:, 0], mask[:, -1]]).mean()
        )
        area_penalty = 0.0
        if area < 0.01:
            area_penalty += 1.0
        if area > 0.85:
            area_penalty += 1.0
        return border_frac + area_penalty

    if bg_hint >= threshold:
        preferred = dark_mask
        alternate = light_mask
    else:
        preferred = light_mask
        alternate = dark_mask

    return preferred if score(preferred) <= score(alternate) else alternate


def _estimate_background_level(arr) -> int:
    import numpy as np

    border = np.concatenate([arr[0, :], arr[-1, :], arr[:, 0], arr[:, -1]])
    return int(np.median(border))

```

```python

def _canonical_background_level(bg_level: int) -> int:
    # Snap the background to a clean binary pole so upload previews and model
    # input do not retain curtain / wall gradients after foreground extraction.
    return 255 if bg_level >= 128 else 0


def _largest_connected_component(mask):
    import numpy as np

    height, width = mask.shape
    visited = np.zeros((height, width), dtype=bool)
    best_component = []

    for y in range(height):
        for x in range(width):
            if not mask[y, x] or visited[y, x]:
                continue

            stack = [(y, x)]
            visited[y, x] = True
            component = []

            while stack:
                cy, cx = stack.pop()
                component.append((cy, cx))
                for ny in range(max(0, cy - 1), min(height, cy + 2)):
                    for nx in range(max(0, cx - 1), min(width, cx + 2)):
                        if visited[ny, nx] or not mask[ny, nx]:
                            continue
                        visited[ny, nx] = True
                        stack.append((ny, nx))

            if len(component) > len(best_component):
                best_component = component

    if not best_component:
        return mask

    refined = np.zeros_like(mask, dtype=bool)
    for y, x in best_component:
        refined[y, x] = True
    return refined


def _refine_foreground_mask(mask):
    import numpy as np
    from PIL import Image, ImageFilter

    mask_img = Image.fromarray((mask.astype(np.uint8) * 255), mode="L")
    # Close small gaps first, then trim isolated speckles.
    mask_img = mask_img.filter(ImageFilter.MaxFilter(5))
    mask_img = mask_img.filter(ImageFilter.MinFilter(5))
    mask_img = mask_img.filter(ImageFilter.MaxFilter(3))
```

```
    refined = np.array(mask_img, dtype=np.uint8) >= 128
    return _largest_connected_component(refined)


def _crop_and_square_image(image, width: int, height: int):
    import numpy as np
    from PIL import Image, ImageOps

    gray = ImageOps.autocontrast(image.convert("L"))
    arr = np.array(gray, dtype=np.uint8)
    bg_level = _canonical_background_level(_estimate_background_level(arr))
    mask = _refine_foreground_mask(_select_foreground_mask(arr))

    ys, xs = np.nonzero(mask)
    if len(xs) == 0 or len(ys) == 0:
        cropped = gray
    else:
        x0, x1 = xs.min(), xs.max()
        y0, y1 = ys.min(), ys.max()
        box_w = x1 - x0 + 1
        box_h = y1 - y0 + 1
        pad_x = max(2, int(box_w * 0.12))
        pad_y = max(2, int(box_h * 0.12))
        x0 = max(0, x0 - pad_x)
        y0 = max(0, y0 - pad_y)
        x1 = min(arr.shape[1] - 1, x1 + pad_x)
        y1 = min(arr.shape[0] - 1, y1 + pad_y)

        masked_arr = np.full_like(arr, bg_level)
        masked_arr[mask] = arr[mask]
        masked = Image.fromarray(masked_arr, mode="L")
        cropped = masked.crop((int(x0), int(y0), int(x1 + 1), int(y1 + 1)))

    side = max(cropped.size)
    square = Image.new("L", (side, side), color=bg_level)
    offset = ((side - cropped.size[0]) // 2, (side - cropped.size[1]) // 2)
    square.paste(cropped, offset)
    return square.resize((width, height))


def _plain_resize_image(image, width: int, height: int):
    from PIL import ImageOps

    return ImageOps.autocontrast(image.convert("L")).resize((width, height))


def _image_to_int8(image) -> List[int]:
    return quantize_u8_to_i8(list(image.getdata()), zero_point=-128)


def _preprocess_image(image, width: int, height: int, mode: str) -> PreprocessedImage:
    if mode == "plain":
        processed = _plain_resize_image(image, width, height)
    elif mode == "crop":
```

```
        processed = _crop_and_square_image(image, width, height)
    else:
        raise ValueError(f"unknown preprocess mode: {mode}")
    return PreprocessedImage(mode=mode, values=_image_to_int8(processed), preview_b64=_encode_image_b64(
     processed))


def preprocess_image_bytes_variants(image_bytes: bytes, width: int = 64, height: int = 64) -> List[
     PreprocessedImage]:
    try:
        from PIL import Image
    except ImportError as exc:  # pragma: no cover - optional dependency
        raise RuntimeError("Pillow is required for image preprocessing") from exc

    with Image.open(BytesIO(image_bytes)) as img:
        return [
            _preprocess_image(img, width, height, "plain"),
            _preprocess_image(img, width, height, "crop"),
        ]


def render_preprocessed_preview_b64(image_bytes: bytes, width: int = 64, height: int = 64) -> str:
    return preprocess_image_bytes_variants(image_bytes, width, height)[1].preview_b64


def load_image_bytes_to_int8(image_bytes: bytes, width: int = 64, height: int = 64) -> List[int]:
    return preprocess_image_bytes_variants(image_bytes, width, height)[1].values


def load_image_to_int8(path: Path, width: int = 64, height: int = 64) -> List[int]:
    try:
        from PIL import Image
    except ImportError as exc:  # pragma: no cover - optional dependency
        raise RuntimeError("Pillow is required for image preprocessing") from exc

    with Image.open(path) as img:
        return _preprocess_image(img, width, height, "crop").values


def write_txt_int8_image(path: Path, values: Iterable[int]) -> None:
    with path.open("w", encoding="utf-8") as fp:
        for value in values:
            fp.write(f"{int(value)}\n")
```

A.3.9 Preprocess Mode Selection

Chooses the preprocessing branch used by auto mode based on CPU-side scoring.

code/gesture_runtime/preprocess_selection.py

```python
"""Heuristics for choosing between preprocessing variants."""

from __future__ import annotations

from typing import Any, Iterable, Tuple


def choose_best_variant(scored: Iterable[Tuple[Any, Any]]) -> Tuple[Any, Any]:
    """Choose a preprocess branch using a conservative disagreement policy.

    The background-removed crop branch can occasionally become overconfident on
    the wrong class. We therefore keep the old "pick by margin" behavior when
    both branches agree, but when they disagree we only trust `crop` if it is
    *obviously* stronger while the `plain` branch looks weak.
    """

    scored = list(scored)
    if not scored:
        raise ValueError("scored variants cannot be empty")

    if len(scored) == 1:
        return scored[0]

    ordered = sorted(
        scored,
        key=lambda item: (item[1].margin, item[1].top_score),
        reverse=True,
    )
    best_variant, best_result = ordered[0]
    second_variant, second_result = ordered[1]

    if best_result.predicted_class == second_result.predicted_class:
        return best_variant, best_result

    plain_pair = next(((v, r) for v, r in scored if getattr(v, "mode", "") == "plain"), None)
    crop_pair = next(((v, r) for v, r in scored if getattr(v, "mode", "") == "crop"), None)
    if plain_pair is None or crop_pair is None:
        return best_variant, best_result

    plain_variant, plain_result = plain_pair
    crop_variant, crop_result = crop_pair

    # If crop saturates near the int8 ceiling while plain is weak, trust crop.
    if crop_result.top_score >= 120.0 and plain_result.top_score < 60.0:
        return crop_variant, crop_result

    # If plain is already reasonably confident, prefer its more literal view.
    if plain_result.margin >= 60.0:
        return plain_variant, plain_result

    # Otherwise fall back to the stronger-scoring branch.
    return best_variant, best_result
```

A.3.10 INT8 Quantization Helpers

Contains the small quantization helpers used by preprocessing and export.

code/gesture_runtime/quantization.py

```python
"""Shared quantization helpers for host/runtime code.

The current RTL datapath is INT8 activation/weight with INT32 accumulation.
The reference Gesture design used a simple Q7.8-style short transport for some
paths; we keep those helpers here for interoperability, but the project-wide
default remains the RTL-native INT8/INT32 format.
"""

from typing import Iterable, List


Q7_8_SCALE = 128.0


def saturate_int8(value: int) -> int:
    return max(-128, min(127, int(value)))


def saturate_int16(value: int) -> int:
    return max(-32768, min(32767, int(value)))


def saturate_int32(value: int) -> int:
    return max(-(2**31), min(2**31 - 1, int(value)))


def float_to_fixed_q7_8(value: float) -> int:
    return saturate_int16(round(value * Q7_8_SCALE))


def fixed_q7_8_to_float(value: int) -> float:
    return int(value) / Q7_8_SCALE


def quantize_u8_to_i8(samples: Iterable[int], zero_point: int = -128) -> List[int]:
    return [saturate_int8(int(v) + zero_point) for v in samples]


def dequantize_q7_8_buffer(samples: Iterable[int]) -> List[float]:
    return [fixed_q7_8_to_float(v) for v in samples]
```

A.3.11 Host CPU Classifier

Runs the TFLite CPU model on the host side for comparison and preprocessing selection.

code/gesture_runtime/cpu_inference.py

```python
"""CPU-side TFLite inference helpers for the web comparison demo."""

from dataclasses import dataclass
from pathlib import Path
from time import perf_counter
from typing import List, Optional


def _load_interpreter(model_path: Path):
    try:
        from tflite_runtime.interpreter import Interpreter  # type: ignore
    except ImportError:
        try:
            import tensorflow as tf  # type: ignore

            Interpreter = tf.lite.Interpreter
        except ImportError as exc:  # pragma: no cover - optional dependency
            raise RuntimeError(
                "A TFLite interpreter is required. Install tflite-runtime or tensorflow."
            ) from exc

    return Interpreter(model_path=str(model_path))


@dataclass
class CPUInferenceResult:
    predicted_class: int
    elapsed_ms: float
    scores: List[float]
    top_score: float
    margin: float


class TFLiteCPUClassifier:
    def __init__(self, model_path: Path):
        self.model_path = Path(model_path)
        self.interpreter = _load_interpreter(self.model_path)
        self.interpreter.allocate_tensors()
        self.input_details = self.interpreter.get_input_details()[0]
        self.output_details = self.interpreter.get_output_details()[0]

        import numpy as np

        self.np = np

    def _prepare_input(self, image_values: List[int]):
        np = self.np
        input_shape = self.input_details["shape"]
        input_dtype = self.input_details["dtype"]
        quant = self.input_details.get("quantization", (0.0, 0))

        arr = np.array(image_values, dtype=np.int16).reshape((1, 64, 64, 1))
```

```
53
54          if input_dtype == np.int8:
55              return arr.astype(np.int8)
56          if input_dtype == np.uint8:
57              return (arr + 128).astype(np.uint8)
58          if input_dtype in (np.float32, np.float64):
59              return ((arr.astype(np.float32) + 128.0) / 255.0).astype(input_dtype)
60
61          scale, zero_point = quant
62          if scale:
63              return ((arr.astype(np.float32) / scale) + zero_point).astype(input_dtype)
64          return arr.astype(input_dtype)
65
66      def predict_int8_image(self, image_values: List[int]) -> CPUInferenceResult:
67          np = self.np
68          input_tensor = self._prepare_input(image_values)
69
70          started = perf_counter()
71          self.interpreter.set_tensor(self.input_details["index"], input_tensor)
72          self.interpreter.invoke()
73          output = self.interpreter.get_tensor(self.output_details["index"])
74          elapsed_ms = (perf_counter() - started) * 1000.0
75
76          scores = np.array(output).reshape(-1).astype(np.float32).tolist()
77          predicted_class = int(np.argmax(scores))
78          ordered = sorted(scores, reverse=True)
79          top_score = float(ordered[0]) if ordered else 0.0
80          second_score = float(ordered[1]) if len(ordered) > 1 else float("-inf")
81          margin = top_score - second_score if ordered else 0.0
82          return CPUInferenceResult(
83              predicted_class=predicted_class,
84              elapsed_ms=elapsed_ms,
85              scores=scores,
86              top_score=top_score,
87              margin=margin,
88          )
```

A.3.12 Web Front-End Template

Defines the browser-side upload, preview, status, result, and feedback UI structure.

code/web_demo/templates/index.html

```
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>CNN_ACC Gesture Demo</title>
7      <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
8    </head>
```

```
<body>
  <main class="page">
    <section class="hero">
      <p class="eyebrow">Board CPU vs FPGA Gesture Comparison</p>
      <h1>Upload One Gesture Image, Run Two Inference Paths</h1>
      <p class="lede">
        The same browser-uploaded image is preprocessed once, then compared across
        board-side HPS CPU inference and the DE1-SoC FPGA accelerator path.
      </p>
      <div id="model-warning" class="model-warning" hidden></div>
    </section>

    <section class="layout">
      <div class="panel input-panel">
        <h2>Input</h2>
        <p class="panel-copy">Upload an image from your computer or capture one from your browser
webcam. Wrong predictions can be saved as correction samples for later retraining.</p>

        <label class="upload-box" for="image-input">
          <input id="image-input" type="file" accept="image/*">
          <span>Select Image</span>
          <small>PNG, JPG, or other browser-supported formats</small>
        </label>

        <div class="preprocess-card">
          <label class="preprocess-field">
            <span>Preprocess Mode</span>
            <select id="preprocess-mode">
              <option value="auto" selected>Auto</option>
              <option value="plain">Plain Resize</option>
              <option value="crop">Crop + Background Removal</option>
            </select>
          </label>
          <div id="branch-diagnostics" class="branch-diagnostics">Branch diagnostics will appear after
one run.</div>
        </div>

        <div class="input-top">
          <div class="webcam-card">
            <div class="webcam-head">
              <h3>Browser Webcam</h3>
              <div class="webcam-actions">
                <button id="start-camera-btn" class="secondary">Start Camera</button>
                <button id="capture-btn" class="secondary" disabled>Capture Frame</button>
              </div>
            </div>
            <div class="camera-stage">
              <video id="camera-preview" autoplay playsinline muted></video>
              <div id="camera-placeholder" class="camera-placeholder">Camera is off</div>
            </div>
            <p class="camera-copy">This uses your PC/browser camera only. The captured frame is sent
through the same pipeline as upload mode.</p>
          </div>

```

```
            <div class="preview-card">
              <div class="preview-head">
                <h3>Upload Preview</h3>
                <span class="preview-note">Image selected from file upload</span>
              </div>
              <div class="preview-wrap">
                <img id="preview" alt="Uploaded preview">
                <div id="preview-placeholder" class="preview-placeholder">No image selected yet</div>
              </div>
            </div>

            <div class="processed-wrap">
              <span class="processed-label">Model Input (64x64)</span>
              <img id="processed-preview" alt="Preprocessed preview">
              <div id="processed-placeholder" class="processed-placeholder">Will appear after inference</
    div>
              <div class="actions">
                <button id="run-btn" class="primary" disabled>Run Comparison</button>
                <button id="reset-btn" class="secondary">Clear</button>
              </div>
            </div>
          </div>

          <div class="feedback-card">
            <div class="feedback-head">
              <h3>Correction Capture</h3>
              <span class="feedback-note">Use this when the prediction is wrong.</span>
            </div>
            <div class="feedback-controls">
              <label class="feedback-field">
                <span>Correct Gesture ID</span>
                <select id="correct-label" disabled>
                  <option value="">Select</option>
                  <option value="0">0</option>
                  <option value="1">1</option>
                  <option value="2">2</option>
                  <option value="3">3</option>
                  <option value="4">4</option>
                  <option value="5">5</option>
                  <option value="6">6</option>
                  <option value="7">7</option>
                  <option value="8">8</option>
                  <option value="9">9</option>
                </select>
              </label>
              <label class="feedback-field feedback-field-wide">
                <span>Note</span>
                <input id="feedback-note" type="text" placeholder="Optional note about lighting,
    background, hand angle...">
              </label>
            </div>
            <button id="save-feedback-btn" class="secondary" disabled>Save Correction Sample</button>
            <div id="feedback-status" class="feedback-status">Run one inference first if you want to save
     a corrected sample.</div>
```

```
111           </div>
112
113         </div>
114
115         <div class="side-column">
116           <div class="panel result-panel">
117             <h2>Results</h2>
118             <p class="panel-copy">This panel compares the board-side HPS CPU software baseline against
    the existing board-side HPS + MMIO + FPGA path for the active 10-class gesture model line.</p>
119
120             <div class="prediction-strip">
121               <div class="metric">
122                 <span class="metric-label">Predicted Gesture ID</span>
123                 <strong id="predicted-class">-</strong>
124               </div>
125               <div class="metric">
126                 <span class="metric-label">Board CPU Gesture ID</span>
127                 <strong id="cpu-predicted-class">-</strong>
128               </div>
129             </div>
130
131             <div class="metrics">
132               <div class="metric">
133                 <span class="metric-label">Board CPU Time</span>
134                 <strong id="cpu-time">-</strong>
135               </div>
136               <div class="metric">
137                 <span class="metric-label">FPGA Wait Time</span>
138                 <strong id="fpga-time">-</strong>
139               </div>
140               <div class="metric">
141                 <span class="metric-label">FPGA RTL Time</span>
142                 <strong id="fpga-rtl-time">-</strong>
143               </div>
144               <div class="metric speedup">
145                 <span class="metric-label">Speedup</span>
146                 <strong id="speedup">-</strong>
147               </div>
148             </div>
149
150             <div class="timing-note">
151               <span id="timing-note">Board CPU and FPGA wait time are now same-platform metrics. Remote
    request time is reported separately.</span>
152             </div>
153
154             <div class="confidence-card">
155               <div class="confidence-head">
156                 <h3>CPU Confidence And Channel Scores</h3>
157                 <span id="confidence-note" class="confidence-note">Waiting for one run.</span>
158               </div>
159               <div class="confidence-summary">
160                 <div class="confidence-metric">
161                   <span>Top Class</span>
162                   <strong id="cpu-top-class">-</strong>
```

```
                </div>
                <div class="confidence-metric">
                  <span>Normalized Confidence</span>
                  <strong id="cpu-confidence">-</strong>
                </div>
                <div class="confidence-metric">
                  <span>Margin</span>
                  <strong id="cpu-margin">-</strong>
                </div>
              </div>
              <div id="cpu-top-channels" class="cpu-top-channels">Top channels will appear after one run.
        </div>
              <div id="channel-grid" class="channel-grid"></div>
            </div>

            <div class="profile-card">
              <div class="profile-head">
                <h3>FPGA RTL Profile</h3>
                <span id="fpga-profile-meta" class="profile-meta">Waiting for one run.</span>
              </div>
              <div id="profile-grid" class="profile-grid">
                <div class="profile-row">
                  <span>L1</span>
                  <strong id="profile-l1">-</strong>
                </div>
                <div class="profile-row">
                  <span>L2 P0</span>
                  <strong id="profile-l2-p0">-</strong>
                </div>
                <div class="profile-row">
                  <span>L2 P1</span>
                  <strong id="profile-l2-p1">-</strong>
                </div>
                <div class="profile-row">
                  <span>L3 P0</span>
                  <strong id="profile-l3-p0">-</strong>
                </div>
                <div class="profile-row">
                  <span>L3 P1</span>
                  <strong id="profile-l3-p1">-</strong>
                </div>
                <div class="profile-row">
                  <span>FC</span>
                  <strong id="profile-fc">-</strong>
                </div>
                <div class="profile-row">
                  <span>Argmax</span>
                  <strong id="profile-argmax">-</strong>
                </div>
                <div class="profile-row profile-row-total">
                  <span>Total</span>
                  <strong id="profile-total">-</strong>
                </div>
              </div>
```

```
            </div>

            <div class="chart-card">
              <h3>Execution Time Comparison</h3>
              <div class="chart">
                <div class="bar-group">
                  <span>Board CPU</span>
                  <div class="bar-track"><div id="cpu-bar" class="bar"></div></div>
                  <span id="cpu-bar-label">-</span>
                </div>
                <div class="bar-group">
                  <span>FPGA</span>
                  <div class="bar-track"><div id="fpga-bar" class="bar bar-alt"></div></div>
                  <span id="fpga-bar-label">-</span>
                </div>
              </div>
            </div>

            <div id="status-box" class="status-box">Waiting for an upload.</div>
          </div>

          <div class="panel suite-panel">
            <div class="suite-head">
              <h3>Built-In Sanity Suite</h3>
              <button id="suite-btn" class="secondary suite-btn">Run 0-9 Suite</button>
            </div>
            <p class="suite-copy">Runs the current web path on the repo's built-in ‘digit_0_test‘ to ‘
    digit_9_test‘ sample images so we can verify whether the deployed model line is internally
    consistent.</p>
            <div id="suite-summary" class="suite-summary">Not run yet.</div>
            <div id="suite-table-wrap" class="suite-table-wrap" hidden>
              <table class="suite-table">
                <thead>
                  <tr>
                    <th>Case</th>
                    <th>Exp</th>
                    <th>CPU</th>
                    <th>FPGA</th>
                    <th>Result</th>
                  </tr>
                </thead>
                <tbody id="suite-table-body"></tbody>
              </table>
            </div>
          </div>
        </div>
      </section>
    </main>
    <script src="{{ url_for('static', filename='app.js') }}"></script>
  </body>
</html>
```

A.3.13 Web Front-End Script

Implements browser-side upload handling, inference requests, progress state, and result rendering.

code/web_demo/static/app.js

```
const imageInput = document.getElementById("image-input");
const preview = document.getElementById("preview");
const previewPlaceholder = document.getElementById("preview-placeholder");
const processedPreview = document.getElementById("processed-preview");
const processedPlaceholder = document.getElementById("processed-placeholder");
const runBtn = document.getElementById("run-btn");
const resetBtn = document.getElementById("reset-btn");
const startCameraBtn = document.getElementById("start-camera-btn");
const captureBtn = document.getElementById("capture-btn");
const cameraPreview = document.getElementById("camera-preview");
const cameraPlaceholder = document.getElementById("camera-placeholder");
const statusBox = document.getElementById("status-box");
const modelWarningBox = document.getElementById("model-warning");
const suiteBtn = document.getElementById("suite-btn");
const suiteSummary = document.getElementById("suite-summary");
const suiteTableWrap = document.getElementById("suite-table-wrap");
const suiteTableBody = document.getElementById("suite-table-body");
const correctLabel = document.getElementById("correct-label");
const feedbackNote = document.getElementById("feedback-note");
const saveFeedbackBtn = document.getElementById("save-feedback-btn");
const feedbackStatus = document.getElementById("feedback-status");
const preprocessModeSelect = document.getElementById("preprocess-mode");
const branchDiagnosticsBox = document.getElementById("branch-diagnostics");

let selectedFile = null;
let cameraStream = null;
let lastResult = null;
let currentPreviewUrl = null;

function setStatus(message, isError = false) {
  statusBox.textContent = message;
  statusBox.classList.toggle("error", isError);
}

function setModelWarning(message) {
  if (!message) {
    modelWarningBox.hidden = true;
    modelWarningBox.textContent = "";
    return;
  }
  modelWarningBox.hidden = false;
  modelWarningBox.textContent = message;
}

function resetResults() {
  lastResult = null;
  document.getElementById("predicted-class").textContent = "-";
  document.getElementById("cpu-predicted-class").textContent = "-";
```

```
  document.getElementById("cpu-time").textContent = "-";
  document.getElementById("fpga-time").textContent = "-";
  document.getElementById("fpga-rtl-time").textContent = "-";
  document.getElementById("speedup").textContent = "-";
  document.getElementById("timing-note").textContent =
    "FPGA RTL time is derived from on-chip cycle counters. Board wait time is the HPS-observed elapsed time.";
  document.getElementById("fpga-profile-meta").textContent = "Waiting for one run.";
  document.getElementById("profile-l1").textContent = "-";
  document.getElementById("profile-l2-p0").textContent = "-";
  document.getElementById("profile-l2-p1").textContent = "-";
  document.getElementById("profile-l3-p0").textContent = "-";
  document.getElementById("profile-l3-p1").textContent = "-";
  document.getElementById("profile-fc").textContent = "-";
  document.getElementById("profile-argmax").textContent = "-";
  document.getElementById("profile-total").textContent = "-";
  document.getElementById("cpu-bar").style.width = "0%";
  document.getElementById("fpga-bar").style.width = "0%";
  document.getElementById("cpu-bar-label").textContent = "-";
  document.getElementById("fpga-bar-label").textContent = "-";
  document.getElementById("cpu-top-class").textContent = "-";
  document.getElementById("cpu-confidence").textContent = "-";
  document.getElementById("cpu-margin").textContent = "-";
  document.getElementById("cpu-top-channels").textContent = "Top channels will appear after one run.";
  document.getElementById("confidence-note").textContent = "Waiting for one run.";
  document.getElementById("channel-grid").innerHTML = "";
  branchDiagnosticsBox.textContent = "Branch diagnostics will appear after one run.";
  processedPreview.removeAttribute("src");
  processedPreview.style.display = "none";
  processedPlaceholder.style.display = "grid";
  correctLabel.value = "";
  correctLabel.disabled = true;
  saveFeedbackBtn.disabled = true;
  feedbackStatus.textContent = "Run one inference first if you want to save a corrected sample.";
}

function renderBranchDiagnostics(rows, chosenMode) {
  if (!rows || rows.length === 0) {
    branchDiagnosticsBox.textContent = "Branch diagnostics unavailable.";
    return;
  }
  const parts = rows.map((row) => {
    const chosenTag = row.mode === chosenMode ? " [chosen]" : "";
    const topSummary = (row.top_channels || [])
      .slice(0, 2)
      .map((entry) => `${entry.label}:${entry.raw_score.toFixed(1)}`)
      .join(", ");
    return `${row.mode}: pred ${row.predicted_label}, margin ${row.margin.toFixed(1)}, top ${row.top_score.toFixed(1)}${topSummary ? `, leaders ${topSummary}` : ""}${chosenTag}`;
  });
  branchDiagnosticsBox.textContent = parts.join(" | ");
}

function renderCpuScoreReport(report) {
```

```
  if (!report) {
    document.getElementById("cpu-top-class").textContent = "-";
    document.getElementById("cpu-confidence").textContent = "-";
    document.getElementById("cpu-margin").textContent = "-";
    document.getElementById("cpu-top-channels").textContent = "Top channels will appear after one run.";
    document.getElementById("confidence-note").textContent = "Waiting for one run.";
    document.getElementById("channel-grid").innerHTML = "";
    return;
  }

  document.getElementById("cpu-top-class").textContent = report.predicted_label;
  document.getElementById("cpu-confidence").textContent = `${report.confidence_pct.toFixed(1)}%`;
  document.getElementById("cpu-margin").textContent = report.margin.toFixed(1);
  document.getElementById("confidence-note").textContent = report.note || "";

  const topSummary = (report.top_channels || [])
    .map((entry) => `${entry.label}: raw ${entry.raw_score.toFixed(1)}, conf ${entry.probability_pct.
     toFixed(1)}%`)
    .join(" | ");
  document.getElementById("cpu-top-channels").textContent =
    topSummary || "Top channels unavailable.";

  const grid = document.getElementById("channel-grid");
  grid.innerHTML = "";
  for (const channel of report.channels || []) {
    const row = document.createElement("div");
    row.className = "channel-row";
    row.innerHTML = `
      <span class="channel-label">${channel.label}</span>
      <div class="channel-track"><div class="channel-fill" style="width:${channel.probability_pct.toFixed
     (2)}%"></div></div>
      <span class="channel-score">${channel.raw_score.toFixed(1)}</span>
      <span class="channel-prob">${channel.probability_pct.toFixed(1)}%</span>
    `;
    grid.appendChild(row);
  }
}

function formatProfileRow(stage) {
  if (!stage) {
    return "-";
  }
  const timeUs =
    stage.time_us < 0.1 ? stage.time_us.toFixed(2) :
    stage.time_us < 1 ? stage.time_us.toFixed(2) :
    stage.time_us.toFixed(1);
  return `${stage.cycles} cyc / ${timeUs} us`;
}

function renderProfile(profile) {
  if (!profile) {
    return;
  }
  const stageByLabel = Object.fromEntries(profile.stages.map((stage) => [stage.label, stage]));
```

```
  document.getElementById("fpga-profile-meta").textContent =
    `${profile.fabric_mhz.toFixed(2)} MHz fabric clock`;
  document.getElementById("profile-l1").textContent = formatProfileRow(stageByLabel["L1"]);
  document.getElementById("profile-l2-p0").textContent = formatProfileRow(stageByLabel["L2 P0"]);
  document.getElementById("profile-l2-p1").textContent = formatProfileRow(stageByLabel["L2 P1"]);
  document.getElementById("profile-l3-p0").textContent = formatProfileRow(stageByLabel["L3 P0"]);
  document.getElementById("profile-l3-p1").textContent = formatProfileRow(stageByLabel["L3 P1"]);
  document.getElementById("profile-fc").textContent = formatProfileRow(stageByLabel["FC"]);
  document.getElementById("profile-argmax").textContent = formatProfileRow(stageByLabel["Argmax"]);
  document.getElementById("profile-total").textContent =
    `${profile.total_cycles} cyc / ${profile.rtl_time_us.toFixed(1)} us`;
}

function setPreviewSource(url) {
  if (currentPreviewUrl && currentPreviewUrl.startsWith("blob:")) {
    URL.revokeObjectURL(currentPreviewUrl);
  }
  currentPreviewUrl = url;
  preview.src = url;
  preview.style.display = "block";
  previewPlaceholder.style.display = "none";
}

async function startCamera() {
  if (cameraStream) {
    return;
  }
  const stream = await navigator.mediaDevices.getUserMedia({
    video: {
      facingMode: "user",
      width: { ideal: 960 },
      height: { ideal: 960 },
    },
    audio: false,
  });
  cameraStream = stream;
  cameraPreview.srcObject = stream;
  cameraPreview.style.display = "block";
  cameraPlaceholder.style.display = "none";
  captureBtn.disabled = false;
  startCameraBtn.textContent = "Camera Ready";
  startCameraBtn.disabled = true;
}

async function captureFrame() {
  if (!cameraStream) {
    return;
  }
  const width = cameraPreview.videoWidth || 640;
  const height = cameraPreview.videoHeight || 640;
  const canvas = document.createElement("canvas");
  canvas.width = width;
  canvas.height = height;
  const context = canvas.getContext("2d");
```

```
  context.drawImage(cameraPreview, 0, 0, width, height);
  const blob = await new Promise((resolve) => canvas.toBlob(resolve, "image/png"));
  if (!blob) {
    throw new Error("Failed to capture webcam frame");
  }
  selectedFile = new File([blob], "webcam_capture.png", { type: "image/png" });
  imageInput.value = "";
  resetResults();
  setPreviewSource(URL.createObjectURL(selectedFile));
  runBtn.disabled = false;
  setStatus("Webcam frame captured. Run comparison when you are ready.");
}

function renderSuite(payload) {
  suiteSummary.textContent = `Built-in suite: ${payload.passed}/${payload.total} passed.`;
  suiteTableBody.innerHTML = "";
  for (const row of payload.rows) {
    const tr = document.createElement("tr");
    tr.innerHTML = `
      <td>${row.case}</td>
      <td>${row.expected_class}</td>
      <td>${row.cpu_predicted_class}</td>
      <td>${row.fpga_predicted_class}</td>
      <td class="${row.pass ? "suite-pass" : "suite-fail"}">${row.pass ? "PASS" : "FAIL"}</td>
    `;
    suiteTableBody.appendChild(tr);
  }
  suiteTableWrap.hidden = false;
}

function renderChart(cpuMs, fpgaMs) {
  const maxValue = Math.max(cpuMs, fpgaMs, 1);
  const cpuWidth = (cpuMs / maxValue) * 100;
  const fpgaWidth = (fpgaMs / maxValue) * 100;
  document.getElementById("cpu-bar").style.width = `${cpuWidth}%`;
  document.getElementById("fpga-bar").style.width = `${fpgaWidth}%`;
  document.getElementById("cpu-bar-label").textContent = `${cpuMs.toFixed(2)} ms`;
  document.getElementById("fpga-bar-label").textContent = `${fpgaMs.toFixed(2)} ms`;
}

async function loadHealth() {
  try {
    const response = await fetch("/api/health");
    const payload = await response.json();
    if (response.ok) {
      setModelWarning(payload.model_warning || "");
    }
  } catch {
    // Keep the page usable even if health probing fails.
  }
}

imageInput.addEventListener("change", () => {
  const [file] = imageInput.files;
```

```
  selectedFile = file || null;
  resetResults();

  if (!selectedFile) {
    preview.removeAttribute("src");
    preview.style.display = "none";
    previewPlaceholder.style.display = "grid";
    runBtn.disabled = true;
    setStatus("Waiting for an upload.");
    return;
  }

  setPreviewSource(URL.createObjectURL(selectedFile));
  runBtn.disabled = false;
  setStatus("Image ready. Run comparison when you are ready.");
});

resetBtn.addEventListener("click", () => {
  imageInput.value = "";
  selectedFile = null;
  preview.removeAttribute("src");
  preview.style.display = "none";
  previewPlaceholder.style.display = "grid";
  runBtn.disabled = true;
  resetResults();
  setStatus("Waiting for an upload.");
});

startCameraBtn.addEventListener("click", async () => {
  try {
    await startCamera();
    setStatus("Camera ready. Capture a frame when you are ready.");
  } catch (error) {
    setStatus(error.message || "Unable to start webcam.", true);
  }
});

captureBtn.addEventListener("click", async () => {
  try {
    await captureFrame();
  } catch (error) {
    setStatus(error.message || "Unable to capture webcam frame.", true);
  }
});

runBtn.addEventListener("click", async () => {
  if (!selectedFile) {
    return;
  }

  runBtn.disabled = true;
  setStatus("Running CPU and FPGA inference...");

  const formData = new FormData();
```

```
  formData.append("image", selectedFile);
  formData.append("preprocess_mode", preprocessModeSelect.value);

  try {
    const response = await fetch("/api/infer", {
      method: "POST",
      body: formData,
    });
    const payload = await response.json();
    if (!response.ok) {
      throw new Error(payload.error || "Inference failed");
    }

    lastResult = payload;
    document.getElementById("predicted-class").textContent = payload.predicted_label;
    document.getElementById("cpu-predicted-class").textContent = payload.cpu_predicted_label;
    document.getElementById("cpu-time").textContent = `${payload.cpu_time_ms.toFixed(2)} ms`;
    document.getElementById("fpga-time").textContent = `${payload.fpga_time_ms.toFixed(2)} ms`;
    document.getElementById("fpga-rtl-time").textContent = `${payload.fpga_rtl_time_ms.toFixed(3)} ms`;
    document.getElementById("speedup").textContent =
      payload.speedup ? `${payload.speedup.toFixed(2)}x` : "-";
    document.getElementById("timing-note").textContent =
      `RTL ${payload.fpga_rtl_time_us.toFixed(1)} us from on-chip counters. ` +
      `Board wait ${payload.fpga_time_ms.toFixed(2)} ms. Remote request ${payload.fpga_request_time_ms.toFixed(2)} ms. ` +
      `Board total ${payload.fpga_board_total_time_ms.toFixed(2)} ms, load ${payload.fpga_board_case_load_time_ms.toFixed(2)} ms, program ${payload.fpga_board_program_time_ms.toFixed(2)} ms.`;

    renderChart(payload.cpu_time_ms, payload.fpga_time_ms);
    renderProfile(payload.fpga_profile);
    renderBranchDiagnostics(payload.branch_diagnostics, payload.preprocess_mode);
    renderCpuScoreReport(payload.cpu_score_report);
    if (payload.preprocessed_preview_url) {
      processedPreview.src = payload.preprocessed_preview_url;
      processedPreview.style.display = "block";
      processedPlaceholder.style.display = "none";
    }
    correctLabel.disabled = false;
    saveFeedbackBtn.disabled = false;
    feedbackStatus.textContent = "If this result is wrong, select the correct label and save the sample for retraining.";
    setModelWarning(payload.model_warning || "");
    const preprocessMode = payload.preprocess_mode ? ` Preprocess: ${payload.preprocess_mode}.` : "";
    setStatus(`Done. Board CPU predicted gesture ID ${payload.cpu_predicted_label}, FPGA predicted gesture ID ${payload.predicted_label}.${preprocessMode}`);
  } catch (error) {
    setStatus(error.message, true);
  } finally {
    runBtn.disabled = false;
  }
});

saveFeedbackBtn.addEventListener("click", async () => {
```

```
  if (!selectedFile || !lastResult) {
    feedbackStatus.textContent = "Run one inference before saving feedback.";
    return;
  }
  if (!correctLabel.value) {
    feedbackStatus.textContent = "Choose the correct gesture ID first.";
    return;
  }

  saveFeedbackBtn.disabled = true;
  feedbackStatus.textContent = "Saving correction sample...";
  const formData = new FormData();
  formData.append("image", selectedFile);
  formData.append("corrected_label", correctLabel.value);
  formData.append("predicted_label", String(lastResult.predicted_label));
  formData.append("preprocess_mode", String(lastResult.preprocess_mode || ""));
  formData.append("note", feedbackNote.value || "");

  try {
    const response = await fetch("/api/feedback", {
      method: "POST",
      body: formData,
    });
    const payload = await response.json();
    if (!response.ok) {
      throw new Error(payload.error || "Unable to save correction sample");
    }
    feedbackStatus.textContent = `Saved correction sample ${payload.sample_id} under ${payload.saved_dir
    }.`;
  } catch (error) {
    feedbackStatus.textContent = error.message;
  } finally {
    saveFeedbackBtn.disabled = false;
  }
});

suiteBtn.addEventListener("click", async () => {
  suiteBtn.disabled = true;
  suiteSummary.textContent = "Running built-in suite...";
  try {
    const response = await fetch("/api/sample-suite");
    const payload = await response.json();
    if (!response.ok) {
      throw new Error(payload.error || "Suite failed");
    }
    renderSuite(payload);
  } catch (error) {
    suiteSummary.textContent = error.message;
    suiteTableWrap.hidden = true;
  } finally {
    suiteBtn.disabled = false;
  }
});
```

```
loadHealth();
```

A.3.14 Web Front-End Styles

Defines the responsive browser UI layout, controls, result panels, and visual states.

code/web_demo/static/styles.css

```
:root {
  --bg: #f5efe4;
  --panel: #fffaf2;
  --ink: #1b1711;
  --muted: #655b4c;
  --accent: #f5b000;
  --accent-deep: #d89200;
  --line: #e3d7c2;
  --success: #173d2a;
  --error: #8f2d23;
}

* {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: "Segoe UI", "Helvetica Neue", Arial, sans-serif;
  background:
    radial-gradient(circle at top right, rgba(245, 176, 0, 0.22), transparent 28%),
    linear-gradient(180deg, #fbf8f1, var(--bg));
  color: var(--ink);
}

.page {
  max-width: 1400px;
  margin: 0 auto;
  padding: 32px 24px 48px;
}

.hero {
  margin-bottom: 22px;
}

.model-warning {
  margin-top: 16px;
  max-width: 860px;
  padding: 14px 16px;
  border: 1px solid rgba(245, 176, 0, 0.35);
  border-radius: 14px;
  background: rgba(245, 176, 0, 0.12);
  color: #6b4c00;
```

```
  line-height: 1.45;
}

.eyebrow {
  margin: 0 0 10px;
  text-transform: uppercase;
  letter-spacing: 0.12em;
  font-size: 0.8rem;
  color: var(--accent-deep);
  font-weight: 700;
}

.hero h1 {
  margin: 0 0 12px;
  font-size: clamp(2rem, 4vw, 3.6rem);
  line-height: 1.05;
}

.lede {
  margin: 0;
  max-width: 720px;
  color: var(--muted);
  font-size: 1.02rem;
}

.layout {
  display: grid;
  grid-template-columns: minmax(0, 1.08fr) minmax(380px, 0.92fr);
  gap: 20px;
  align-items: stretch;
}

.panel {
  background: var(--panel);
  border: 1px solid var(--line);
  border-radius: 24px;
  padding: 20px;
  box-shadow: 0 18px 40px rgba(34, 25, 7, 0.08);
}

.input-panel {
  display: grid;
  grid-template-columns: 1fr;
  gap: 16px;
  align-content: start;
  min-width: 0;
  height: 100%;
}

.input-panel > h2,
.input-panel > .panel-copy,
.input-panel > .upload-box,
.input-panel > .preprocess-card,
.input-panel > .input-top {
```

```
  grid-column: auto;
}

.result-panel {
  display: grid;
  gap: 16px;
  align-content: start;
  min-width: 0;
}

.side-column {
  display: flex;
  flex-direction: column;
  gap: 16px;
  min-width: 0;
  height: 100%;
}

.suite-panel {
  min-width: 0;
  display: grid;
  gap: 10px;
  border-color: rgba(245, 176, 0, 0.32);
  background:
    linear-gradient(180deg, rgba(245, 176, 0, 0.08), rgba(255, 255, 255, 0.74));
  margin-top: auto;
}

.feedback-card {
  grid-column: auto;
}

.input-top {
  display: grid;
  grid-template-columns: repeat(2, minmax(0, 1fr));
  grid-template-areas:
    "webcam preview"
    "processed processed";
  gap: 16px;
  align-items: start;
  min-width: 0;
}

.input-top > * {
  min-width: 0;
}

.panel h2,
.chart-card h3 {
  margin-top: 0;
  margin-bottom: 0;
}

.panel-copy {
```

```
  color: var(--muted);
  margin: 0;
  line-height: 1.45;
}

.upload-box {
  display: grid;
  gap: 6px;
  padding: 16px 18px;
  border: 2px dashed var(--line);
  border-radius: 18px;
  background: rgba(245, 176, 0, 0.05);
  cursor: pointer;
}

.upload-box input {
  display: none;
}

.upload-box span {
  font-weight: 700;
}

.upload-box small {
  color: var(--muted);
}

.preprocess-card {
  display: grid;
  grid-template-columns: minmax(220px, 280px) minmax(0, 1fr);
  gap: 12px;
  padding: 14px 16px;
  border: 1px solid var(--line);
  border-radius: 18px;
  background: rgba(255, 255, 255, 0.55);
  align-items: end;
}

.preprocess-field {
  display: grid;
  gap: 8px;
}

.preprocess-field span {
  color: var(--muted);
  font-size: 0.9rem;
}

.preprocess-field select {
  width: 100%;
  border: 1px solid var(--line);
  border-radius: 12px;
  padding: 10px 12px;
  background: #fff;
```

```
  color: var(--ink);
}

.branch-diagnostics {
  color: var(--muted);
  font-size: 0.9rem;
  line-height: 1.45;
  min-width: 0;
}

.webcam-card,
.feedback-card,
.preview-card,
.preview-wrap,
.processed-wrap {
  border: 1px solid var(--line);
  border-radius: 18px;
  padding: 16px;
  background: rgba(255, 255, 255, 0.55);
}

.webcam-card {
  display: flex;
  flex-direction: column;
  gap: 10px;
  grid-area: webcam;
}

.preview-card {
  display: grid;
  gap: 10px;
  grid-area: preview;
}

.webcam-head,
.feedback-head,
.preview-head {
  display: flex;
  align-items: flex-start;
  justify-content: space-between;
  gap: 10px;
}

.webcam-head h3,
.feedback-head h3,
.preview-head h3 {
  margin: 0;
}

.preview-note {
  color: var(--muted);
  font-size: 0.84rem;
  text-align: right;
}
```

```
260
261 .webcam-actions {
262   display: flex;
263   gap: 8px;
264   flex-wrap: wrap;
265 }
266
267 .camera-stage {
268   width: 100%;
269   aspect-ratio: 1 / 1;
270   max-width: 100%;
271   min-height: 280px;
272   border-radius: 14px;
273   overflow: hidden;
274   border: 1px solid var(--line);
275   background: #121212;
276   display: grid;
277   place-items: center;
278 }
279
280 #camera-preview {
281   width: 100%;
282   height: 100%;
283   object-fit: cover;
284   display: none;
285 }
286
287 .camera-placeholder {
288   color: #d6d6d6;
289   padding: 16px;
290   text-align: center;
291 }
292
293 .camera-copy,
294 .feedback-note {
295   color: var(--muted);
296   font-size: 0.9rem;
297   line-height: 1.45;
298 }
299
300 .preview-wrap {
301   overflow: hidden;
302   display: grid;
303   place-items: center;
304   min-width: 0;
305   width: 100%;
306   aspect-ratio: 1 / 1;
307   min-height: 280px;
308   background: #0c0c0c;
309   padding: 0;
310 }
311
312 #preview {
313   width: 100%;
```

```
  height: 100%;
  object-fit: cover;
  display: none;
}

.preview-placeholder {
  color: #d6d6d6;
  padding: 16px;
  text-align: center;
}

.processed-wrap {
  grid-area: processed;
  padding: 12px;
  display: grid;
  grid-template-columns: minmax(140px, 180px) minmax(0, 1fr);
  grid-template-areas:
    "label actions"
    "preview actions";
  align-items: start;
  min-width: 0;
  gap: 12px;
  overflow: hidden;
}

.processed-label {
  display: block;
  grid-area: label;
  margin-bottom: 0;
  color: var(--muted);
  font-size: 0.92rem;
}

#processed-preview,
.processed-placeholder {
  grid-area: preview;
  width: min(100%, 160px);
  aspect-ratio: 1 / 1;
  height: auto;
}

#processed-preview {
  image-rendering: pixelated;
  border-radius: 10px;
  border: 1px solid var(--line);
  display: none;
  background: #111;
}

.processed-placeholder {
  display: grid;
  place-items: center;
  border-radius: 10px;
  border: 1px dashed var(--line);
```

```
  color: var(--muted);
  text-align: center;
  font-size: 0.9rem;
  padding: 10px;
}

.actions {
  grid-area: actions;
  display: flex;
  flex-direction: column;
  gap: 10px;
  width: 100%;
  align-self: stretch;
  justify-self: end;
  max-width: 320px;
}

.actions button {
  width: 100%;
}

.feedback-controls {
  display: grid;
  grid-template-columns: 150px 1fr;
  gap: 10px;
  margin-top: 10px;
}

.feedback-field {
  display: grid;
  gap: 8px;
}

.feedback-field-wide {
  min-width: 0;
}

.feedback-field span {
  color: var(--muted);
  font-size: 0.92rem;
}

.feedback-field select,
.feedback-field input {
  width: 100%;
  border: 1px solid var(--line);
  border-radius: 12px;
  padding: 10px 12px;
  background: #fff;
  color: var(--ink);
}

.feedback-status {
  margin-top: 10px;
```

```
  color: var(--muted);
  font-size: 0.92rem;
}

.suite-head {
  display: grid;
  grid-template-columns: minmax(0, 1fr) auto;
  align-items: start;
  gap: 12px;
}

.suite-head h3 {
  margin: 0;
}

.suite-copy {
  color: var(--muted);
  margin: 0;
  line-height: 1.45;
}

.suite-summary {
  font-weight: 700;
}

.suite-table-wrap {
  margin-top: 8px;
  overflow-x: auto;
}

.suite-table {
  width: 100%;
  border-collapse: collapse;
  font-size: 0.92rem;
}

.suite-table th,
.suite-table td {
  text-align: left;
  padding: 8px 10px;
  border-top: 1px solid var(--line);
}

.suite-pass {
  color: #21553a;
  font-weight: 700;
}

.suite-fail {
  color: var(--error);
  font-weight: 700;
}

button {
```

```
  border: 0;
  border-radius: 12px;
  padding: 11px 16px;
  font-weight: 700;
  cursor: pointer;
}

button.primary {
  background: var(--accent);
  color: #201500;
}

button.secondary {
  background: #e9e2d5;
  color: #3f372b;
}

button:disabled {
  opacity: 0.55;
  cursor: not-allowed;
}

.prediction-strip {
  display: grid;
  grid-template-columns: repeat(2, minmax(0, 1fr));
  gap: 12px;
}

.metrics {
  display: grid;
  grid-template-columns: repeat(2, minmax(0, 1fr));
  gap: 12px;
}

.metric {
  border: 1px solid var(--line);
  border-radius: 18px;
  padding: 14px;
  background: rgba(255, 255, 255, 0.65);
  min-width: 0;
}

.metric-label {
  display: block;
  color: var(--muted);
  margin-bottom: 8px;
}

.metric strong {
  display: block;
  font-size: clamp(1.3rem, 1.9vw, 1.7rem);
  min-width: 0;
  overflow-wrap: anywhere;
}
```

```
.metric.speedup {
  background: linear-gradient(135deg, rgba(245, 176, 0, 0.16), rgba(245, 176, 0, 0.03));
}

.chart-card {
  border: 1px solid var(--line);
  border-radius: 18px;
  padding: 16px;
  min-width: 0;
}

.profile-card {
  border: 1px solid var(--line);
  border-radius: 18px;
  padding: 16px;
  background: rgba(255, 255, 255, 0.65);
  min-width: 0;
}

.confidence-card {
  border: 1px solid var(--line);
  border-radius: 18px;
  padding: 16px;
  background: rgba(255, 255, 255, 0.65);
  min-width: 0;
  display: grid;
  gap: 12px;
}

.confidence-head {
  display: flex;
  align-items: baseline;
  justify-content: space-between;
  gap: 12px;
}

.confidence-head h3 {
  margin: 0;
}

.confidence-note {
  color: var(--muted);
  font-size: 0.88rem;
}

.confidence-summary {
  display: grid;
  grid-template-columns: repeat(3, minmax(0, 1fr));
  gap: 10px;
}

.confidence-metric {
  padding: 10px 12px;
```

```
  border-radius: 14px;
  background: rgba(245, 176, 0, 0.07);
}

.confidence-metric span {
  display: block;
  color: var(--muted);
  margin-bottom: 6px;
}

.confidence-metric strong {
  font-size: 1rem;
}

.cpu-top-channels {
  color: var(--muted);
  font-size: 0.92rem;
  line-height: 1.45;
}

.channel-grid {
  display: grid;
  gap: 8px;
}

.channel-row {
  display: grid;
  grid-template-columns: 32px minmax(0, 1fr) 64px 58px;
  align-items: center;
  gap: 10px;
}

.channel-label,
.channel-score,
.channel-prob {
  font-size: 0.9rem;
}

.channel-score,
.channel-prob {
  text-align: right;
  color: var(--muted);
}

.channel-track {
  height: 12px;
  border-radius: 999px;
  background: #ebe2d3;
  overflow: hidden;
}

.channel-fill {
  height: 100%;
  background: linear-gradient(90deg, #2b4d3d, #75a082);
```

```
  border-radius: 999px;
}

.profile-head {
  display: flex;
  align-items: baseline;
  justify-content: space-between;
  gap: 12px;
}

.profile-head h3 {
  margin: 0;
}

.profile-meta {
  color: var(--muted);
  font-size: 0.92rem;
}

.profile-grid {
  display: grid;
  grid-template-columns: repeat(2, minmax(0, 1fr));
  gap: 10px 12px;
  margin-top: 12px;
}

.profile-row {
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 12px;
  padding: 10px 12px;
  border-radius: 14px;
  background: rgba(245, 176, 0, 0.07);
}

.profile-row span {
  color: var(--muted);
}

.profile-row strong {
  text-align: right;
  font-size: 0.94rem;
}

.profile-row-total {
  grid-column: 1 / -1;
  background: linear-gradient(135deg, rgba(43, 77, 61, 0.13), rgba(117, 160, 130, 0.05));
}

.chart {
  display: grid;
  gap: 14px;
}
```

```
692
693 .bar-group {
694   display: grid;
695   grid-template-columns: 56px 1fr 90px;
696   align-items: center;
697   gap: 12px;
698 }
699
700 .bar-track {
701   height: 18px;
702   border-radius: 999px;
703   background: #ebe2d3;
704   overflow: hidden;
705 }
706
707 .bar {
708   height: 100%;
709   width: 0;
710   border-radius: 999px;
711   background: linear-gradient(90deg, #f5b000, #ffd36d);
712 }
713
714 .bar-alt {
715   background: linear-gradient(90deg, #2b4d3d, #75a082);
716 }
717
718 .status-box {
719   padding: 14px 16px;
720   border-radius: 14px;
721   background: #efe6d6;
722   color: var(--success);
723   min-width: 0;
724   width: 100%;
725   max-width: 100%;
726   overflow: hidden;
727 }
728
729 .status-box.error {
730   background: #f7dbd5;
731   color: var(--error);
732 }
733
734 .suite-btn {
735   white-space: nowrap;
736   justify-self: end;
737 }
738
739 @media (max-width: 1180px) {
740   .layout {
741     grid-template-columns: 1fr;
742     align-items: start;
743   }
744
745   .side-column {
```

```
    gap: 14px;
    height: auto;
  }

  .input-panel {
    height: auto;
  }

  .suite-panel {
    margin-top: 0;
  }
}

@media (max-width: 980px) {
  .input-top {
    grid-template-columns: repeat(2, minmax(0, 1fr));
    grid-template-areas:
      "webcam preview"
      "processed processed";
  }

  .camera-stage,
  .preview-wrap {
    min-height: 0;
  }
}

@media (max-width: 760px) {
  .page {
    padding: 24px 16px 36px;
  }

  .input-top {
    grid-template-columns: 1fr;
    grid-template-areas:
      "webcam"
      "preview"
      "processed";
  }

  .processed-wrap {
    grid-template-columns: 1fr;
    grid-template-areas:
      "label"
      "preview"
      "actions";
  }

  .webcam-head,
  .feedback-head,
  .preview-head {
    flex-direction: column;
    align-items: flex-start;
  }
```

```
800
801   .suite-head {
802     grid-template-columns: 1fr;
803   }
804
805   .profile-grid {
806     grid-template-columns: 1fr;
807   }
808
809   .confidence-summary {
810     grid-template-columns: 1fr;
811   }
812
813   .metrics {
814     grid-template-columns: 1fr;
815   }
816
817   .prediction-strip {
818     grid-template-columns: 1fr;
819   }
820
821   .feedback-controls {
822     grid-template-columns: 1fr;
823   }
824
825   .preprocess-card {
826     grid-template-columns: 1fr;
827   }
828
829   .webcam-actions,
830   .actions {
831     width: 100%;
832   }
833
834   .webcam-actions button,
835   .actions button,
836   .suite-btn {
837     width: 100%;
838   }
839
840   .preview-note {
841     text-align: left;
842   }
843 }
```

A.4 HPS Board C Runtime

A.4.1 MMIO Register ABI

Defines the software-visible register numbers, bit fields, control flags, and default scratchpad layout.

code/include/cnn_mmio_regs.h

```
#ifndef CNN_MMIO_REGS_H
#define CNN_MMIO_REGS_H

#include <stdint.h>

/*
 * Shared register/memory map for cnn_mmio_interface.
 *
 * This header is the ABI contract between the HPS C programs and the FPGA RTL.
 * If a register index, bit field, or scratchpad address changes here, the
 * matching decode logic in cnn_mmio_interface.v must change in the same way.
 *
 * The RTL uses address[12] to split the space:
 *   0x0000..0x0FFF : 16 KiB 32-bit scratchpad memory space
 *   0x1000..0x101F : 32-bit config/status register space
 *
 * Userspace/HPS code should address 32-bit words, not bytes.
 */

#define CNN_MMIO_SCRATCHPAD_AW      12u
#define CNN_MMIO_SCRATCHPAD_WORDS   (1u << CNN_MMIO_SCRATCHPAD_AW)
#define CNN_MMIO_SCRATCHPAD_MASK    (CNN_MMIO_SCRATCHPAD_WORDS - 1u)

#define CNN_MMIO_MEM_SPACE_BIT      (0u << CNN_MMIO_SCRATCHPAD_AW)
#define CNN_MMIO_CFG_SPACE_BIT      (1u << CNN_MMIO_SCRATCHPAD_AW)

#define CNN_MMIO_REG_CONTROL       0u
#define CNN_MMIO_REG_STATUS        1u
#define CNN_MMIO_REG_CONV_CFG_BASE 2u
#define CNN_MMIO_REG_CONV_CFG_LEN  3u
#define CNN_MMIO_REG_CONV_WT_BASE  4u
#define CNN_MMIO_REG_CONV_WT_LEN   5u
#define CNN_MMIO_REG_FC_BIAS_BASE  6u
#define CNN_MMIO_REG_FC_BIAS_LEN   7u
#define CNN_MMIO_REG_FCW_BASE      8u
#define CNN_MMIO_REG_FCW_LEN       9u
#define CNN_MMIO_REG_IMAGE_BASE    10u
#define CNN_MMIO_REG_IMAGE_LEN     11u
#define CNN_MMIO_REG_PREDICT       12u
#define CNN_MMIO_REG_IF_ERROR      13u
#define CNN_MMIO_REG_PROFILE_L1_LO     14u
#define CNN_MMIO_REG_PROFILE_L1_HI     15u
#define CNN_MMIO_REG_PROFILE_L2_P0_LO  16u
#define CNN_MMIO_REG_PROFILE_L2_P0_HI  17u
#define CNN_MMIO_REG_PROFILE_L2_P1_LO  18u
#define CNN_MMIO_REG_PROFILE_L2_P1_HI  19u
#define CNN_MMIO_REG_PROFILE_L3_P0_LO  20u
#define CNN_MMIO_REG_PROFILE_L3_P0_HI  21u
#define CNN_MMIO_REG_PROFILE_L3_P1_LO  22u
#define CNN_MMIO_REG_PROFILE_L3_P1_HI  23u
#define CNN_MMIO_REG_PROFILE_FC_LO     24u
#define CNN_MMIO_REG_PROFILE_FC_HI     25u
```

```
#define CNN_MMIO_REG_PROFILE_ARGMAX_LO 26u
#define CNN_MMIO_REG_PROFILE_ARGMAX_HI 27u
#define CNN_MMIO_REG_PROFILE_TOTAL_LO  28u
#define CNN_MMIO_REG_PROFILE_TOTAL_HI  29u
/* Optional debug/readback helpers. Not used by the normal host flow. */
#define CNN_MMIO_REG_LAST_WRITE        30u
#define CNN_MMIO_REG_MAGIC            31u

#define CNN_MMIO_CTRL_MODEL_LOAD   0x0001u
#define CNN_MMIO_CTRL_INFER        0x0002u
#define CNN_MMIO_CTRL_CLEAR_STATUS 0x0004u

#define CNN_MMIO_STATUS_BUSY_SHIFT         1u
#define CNN_MMIO_STATUS_MODEL_LOADED_SHIFT 2u
#define CNN_MMIO_STATUS_PREDICT_DONE_SHIFT 3u
#define CNN_MMIO_STATUS_PREDICT_SHIFT      4u

/*
 * Default staged memory layout for the current 64x64x1 -> 10-class model.
 *
 * All *_WORDS values count 32-bit host words. All *_BASE_W values are
 * 32-bit word addresses because the MMIO scratchpad is word-indexed.
 *
 * Model structure behind these constants:
 *   - L1 conv: 3x3x1  -> DOT_K = 9
 *   - L2 conv: 3x3x4  -> DOT_K = 36, split into pass0/pass1
 *   - L3 conv: 3x3x8  -> DOT_K = 72, split into pass0/pass1
 *   - FC: 6x6x8 = 288 inputs -> 10 outputs
 *
 * Derived word counts:
 *   - CONV_CFG_WORDS = 5 passes x 9 cfg words/pass = 45
 *     (L1, L2p0, L2p1, L3p0, L3p1)
 *   - CONV_WT_WORDS  = 9 + 36 + 36 + 72 + 72 = 225
 *   - FC_BIAS_WORDS  = 10 output classes = 10
 *   - FCW_WORDS      = 288 FC positions x 3 host words/position = 864
 *     Each FC position stores 10x8b = 80b of weights, packed from 3x32b.
 *   - IMAGE_WORDS    = 64x64x1 int8 pixels / 4 pixels per 32-bit word = 1024
 *
 * Base addresses are packed contiguously in word units:
 *   conv_cfg @   0
 *   conv_wt  @   0 + 45     = 45
 *   fc_bias  @  45 + 225    = 270
 *   fcw      @ 270 + 10     = 280
 *   image    @ 280 + 864    = 1144
 */
#define CNN_MMIO_DEFAULT_CONV_CFG_BASE_W  0u
#define CNN_MMIO_DEFAULT_CONV_CFG_WORDS   45u
#define CNN_MMIO_DEFAULT_CONV_WT_BASE_W   45u
#define CNN_MMIO_DEFAULT_CONV_WT_WORDS    225u
#define CNN_MMIO_DEFAULT_FC_BIAS_BASE_W   270u
#define CNN_MMIO_DEFAULT_FC_BIAS_WORDS    10u
#define CNN_MMIO_DEFAULT_FCW_BASE_W       280u
#define CNN_MMIO_DEFAULT_FCW_WORDS        864u
#define CNN_MMIO_DEFAULT_IMAGE_BASE_W     1144u
```

```
#define CNN_MMIO_DEFAULT_IMAGE_WORDS      1024u

static inline uint32_t cnn_mmio_cfg_addr(uint32_t reg_idx) {
  /* Config/status registers live in the address[12] == 1 half of the window. */
  return CNN_MMIO_CFG_SPACE_BIT | (reg_idx & 0x1Fu);
}

static inline uint32_t cnn_mmio_mem_addr(uint32_t word_addr) {
  /* Scratchpad addresses are 32-bit word indices written by the HPS. */
  return CNN_MMIO_MEM_SPACE_BIT | (word_addr & CNN_MMIO_SCRATCHPAD_MASK);
}

static inline uint32_t cnn_mmio_pack_status_predict(uint16_t status_word) {
  return (status_word >> CNN_MMIO_STATUS_PREDICT_SHIFT) & 0xFu;
}

static inline uint32_t cnn_mmio_status_busy(uint16_t status_word) {
  return (status_word >> CNN_MMIO_STATUS_BUSY_SHIFT) & 0x1u;
}

static inline uint32_t cnn_mmio_status_model_loaded(uint16_t status_word) {
  return (status_word >> CNN_MMIO_STATUS_MODEL_LOADED_SHIFT) & 0x1u;
}

static inline uint32_t cnn_mmio_status_predict_done(uint16_t status_word) {
  return (status_word >> CNN_MMIO_STATUS_PREDICT_DONE_SHIFT) & 0x1u;
}

#endif
```

A.4.2 HPS MMIO Helper API

Declares the shared C structs and helper functions used by all board-side MMIO commands.

code/include/cnn_mmio_host.h

```
#ifndef CNN_MMIO_HOST_H
#define CNN_MMIO_HOST_H

#include <stdint.h>

#include "cnn_mmio_regs.h"

#ifdef __cplusplus
extern "C" {
#endif

/*
 * The HPS maps one small page-aligned lightweight-bridge window. The RTL uses
 * 16 KiB of scratchpad, and the highest 32-bit config/status register ends at
 * byte offset 0x407F. A 20 KiB span is the smallest 4 KiB page-aligned mapping
```

```
 * that covers the implemented scratchpad and register window.
 */
#define CNN_MMIO_MAP_SPAN_BYTES (20 * 1024)
#define CNN_MMIO_DEFAULT_TIMEOUT_MS 1000
#define CNN_MMIO_DEFAULT_FABRIC_MHZ 50.0

struct cnn_mmio_preload_bundle {
  int32_t conv_cfg[CNN_MMIO_DEFAULT_CONV_CFG_WORDS];
  uint32_t conv_wt[CNN_MMIO_DEFAULT_CONV_WT_WORDS];
  int32_t fc_bias[CNN_MMIO_DEFAULT_FC_BIAS_WORDS];
  int32_t fcw[CNN_MMIO_DEFAULT_FCW_WORDS];
};

struct cnn_mmio_inference_case {
  uint32_t image[CNN_MMIO_DEFAULT_IMAGE_WORDS];
  int expected_class;
};

struct cnn_mmio_device {
  int fd;
  void *map_base;
  volatile uint32_t *mmio_base;
  uintptr_t csr_base;
};

struct cnn_mmio_profile {
  uint32_t l1_cycles;
  uint32_t l2_p0_cycles;
  uint32_t l2_p1_cycles;
  uint32_t l3_p0_cycles;
  uint32_t l3_p1_cycles;
  uint32_t fc_cycles;
  uint32_t argmax_cycles;
  uint32_t total_cycles;
};

int cnn_mmio_load_preload_bundle(const char *preload_root, struct cnn_mmio_preload_bundle *bundle);
int cnn_mmio_load_inference_case(const char *case_root, struct cnn_mmio_inference_case *tc);

int cnn_mmio_open(struct cnn_mmio_device *dev, uintptr_t csr_base, const char *devmem_path);
void cnn_mmio_close(struct cnn_mmio_device *dev);

void cnn_mmio_program_default_registers(volatile uint32_t *mmio_base);
void cnn_mmio_write_preload_bundle(volatile uint32_t *mmio_base, const struct cnn_mmio_preload_bundle *
    bundle);
void cnn_mmio_write_inference_case(volatile uint32_t *mmio_base, const struct cnn_mmio_inference_case *tc
    );

uint16_t cnn_mmio_read_status(volatile uint32_t *mmio_base);
uint16_t cnn_mmio_read_error(volatile uint32_t *mmio_base);
uint16_t cnn_mmio_read_predict(volatile uint32_t *mmio_base);
void cnn_mmio_read_profile(volatile uint32_t *mmio_base, struct cnn_mmio_profile *profile);

void cnn_mmio_clear_status(volatile uint32_t *mmio_base);
```

```
void cnn_mmio_start_model_load(volatile uint32_t *mmio_base);
void cnn_mmio_start_infer(volatile uint32_t *mmio_base);

int cnn_mmio_wait_for_status_bit(
    volatile uint32_t *mmio_base,
    unsigned bit_idx,
    unsigned expected_value,
    int timeout_ms,
    uint16_t *last_status);

#ifdef __cplusplus
}
#endif

#endif
```

A.4.3 Common MMIO Implementation

Maps /dev/mem, writes preload/image words, starts model load or inference, polls status, and reads profile counters.

code/tools/cnn_mmio_host.c

```
#define _POSIX_C_SOURCE 200809L

#include "../include/cnn_mmio_host.h"

#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

static int read_i32_lines(const char *path, int32_t *dst, size_t count) {
  FILE *fp = fopen(path, "r");
  size_t i;
  if (!fp) {
    perror(path);
    return -1;
  }
  for (i = 0; i < count; ++i) {
    if (fscanf(fp, "%" SCNd32, &dst[i]) != 1) {
      fprintf(stderr, "short read in %s at index %zu\n", path, i);
      fclose(fp);
```

```
      return -1;
    }
  }
  fclose(fp);
  return 0;
}

static int read_packed_i8_words(const char *path, uint32_t *dst, size_t word_count) {
  FILE *fp = fopen(path, "r");
  size_t i;
  if (!fp) {
    perror(path);
    return -1;
  }
  for (i = 0; i < word_count; ++i) {
    int32_t b0, b1, b2, b3;
    if (fscanf(fp, "%" SCNd32, &b0) != 1 ||
        fscanf(fp, "%" SCNd32, &b1) != 1 ||
        fscanf(fp, "%" SCNd32, &b2) != 1 ||
        fscanf(fp, "%" SCNd32, &b3) != 1) {
      fprintf(stderr, "short packed read in %s at word %zu\n", path, i);
      fclose(fp);
      return -1;
    }
    dst[i] = ((uint32_t)(b0 & 0xFF)) |
             ((uint32_t)(b1 & 0xFF) << 8) |
             ((uint32_t)(b2 & 0xFF) << 16) |
             ((uint32_t)(b3 & 0xFF) << 24);
  }
  fclose(fp);
  return 0;
}

static int load_manifest_expected_class(const char *path, int *expected_class) {
  FILE *fp = fopen(path, "r");
  char line[256];
  if (!fp) {
    perror(path);
    return -1;
  }
  while (fgets(line, sizeof(line), fp)) {
    if (strncmp(line, "predict_class=", 14) == 0) {
      *expected_class = atoi(line + 14);
      fclose(fp);
      return 0;
    }
  }
  fclose(fp);
  fprintf(stderr, "predict_class not found in %s\n", path);
  return -1;
}

static void build_path(char *dst, size_t dst_len, const char *root, const char *leaf) {
  snprintf(dst, dst_len, "%s/%s", root, leaf);
```

```
}

static int try_read_i32_lines(const char *root, const char *const *leaves,
                              int32_t *dst, size_t count) {
  char path[1024];
  size_t i;
  for (i = 0; leaves[i] != NULL; ++i) {
    build_path(path, sizeof(path), root, leaves[i]);
    if (read_i32_lines(path, dst, count) == 0)
      return 0;
  }
  return -1;
}

static int try_read_packed_i8_words(const char *root, const char *const *leaves,
                                    uint32_t *dst, size_t word_count) {
  char path[1024];
  size_t i;
  for (i = 0; leaves[i] != NULL; ++i) {
    build_path(path, sizeof(path), root, leaves[i]);
    if (read_packed_i8_words(path, dst, word_count) == 0)
      return 0;
  }
  return -1;
}

static void mmio_write_cfg_reg(volatile uint32_t *base, uint32_t reg_idx, uint32_t value) {
  base[cnn_mmio_cfg_addr(reg_idx)] = value;
}

static uint32_t mmio_read_cfg_reg(volatile uint32_t *base, uint32_t reg_idx) {
  return base[cnn_mmio_cfg_addr(reg_idx)];
}

static void write_word_image(volatile uint32_t *base, uint32_t start_word,
                             const uint32_t *words, size_t word_count) {
  size_t i;

  for (i = 0; i < word_count; ++i) {
    base[cnn_mmio_mem_addr(start_word + (uint32_t)i)] = words[i];
  }
}

int cnn_mmio_load_preload_bundle(const char *preload_root, struct cnn_mmio_preload_bundle *bundle) {
  static const char *const conv_cfg_candidates[] = {
      "preload_conv_cfg_45w.txt",
      "conv_cfg_words.txt",
      NULL};
  static const char *const conv_wt_candidates[] = {
      "preload_conv_wt_225w_bytes.txt",
      "conv_wt_words.txt",
      NULL};
  static const char *const fc_bias_candidates[] = {
      "preload_fc_bias_10w.txt",
```

```
        "fc_bias_words.txt",
        NULL};
    static const char *const fcw_candidates[] = {
        "preload_fcw_864w.txt",
        "fcw_words.txt",
        NULL};

    if (try_read_i32_lines(preload_root, conv_cfg_candidates, bundle->conv_cfg,
                           CNN_MMIO_DEFAULT_CONV_CFG_WORDS) != 0)
      return -1;

    if (try_read_packed_i8_words(preload_root, conv_wt_candidates, bundle->conv_wt,
                                 CNN_MMIO_DEFAULT_CONV_WT_WORDS) != 0)
      return -1;

    if (try_read_i32_lines(preload_root, fc_bias_candidates, bundle->fc_bias,
                           CNN_MMIO_DEFAULT_FC_BIAS_WORDS) != 0)
      return -1;

    if (try_read_i32_lines(preload_root, fcw_candidates, (int32_t *)bundle->fcw,
                           CNN_MMIO_DEFAULT_FCW_WORDS) != 0)
      return -1;

    return 0;
}

int cnn_mmio_load_inference_case(const char *case_root, struct cnn_mmio_inference_case *tc) {
    char path[1024];
    static const char *const image_candidates[] = {
        "tb_conv1_in_i8_64x64x1.txt",
        "image_words.txt",
        NULL};

    if (try_read_packed_i8_words(case_root, image_candidates, tc->image,
                                 CNN_MMIO_DEFAULT_IMAGE_WORDS) != 0)
      return -1;

    build_path(path, sizeof(path), case_root, "manifest.txt");
    if (load_manifest_expected_class(path, &tc->expected_class) != 0)
      return -1;

    return 0;
}

int cnn_mmio_open(struct cnn_mmio_device *dev, uintptr_t csr_base, const char *devmem_path) {
    memset(dev, 0, sizeof(*dev));
    dev->fd = open(devmem_path, O_RDWR | O_SYNC);
    if (dev->fd < 0) {
      perror(devmem_path);
      return -1;
    }

    dev->map_base = mmap(NULL, CNN_MMIO_MAP_SPAN_BYTES, PROT_READ | PROT_WRITE,
                         MAP_SHARED, dev->fd, csr_base);
```

```c
  if (dev->map_base == MAP_FAILED) {
    perror("mmap");
    close(dev->fd);
    dev->fd = -1;
    return -1;
  }

  dev->mmio_base = (volatile uint32_t *)dev->map_base;
  dev->csr_base = csr_base;
  return 0;
}

void cnn_mmio_close(struct cnn_mmio_device *dev) {
  if (dev->map_base && dev->map_base != MAP_FAILED) {
    munmap(dev->map_base, CNN_MMIO_MAP_SPAN_BYTES);
  }
  if (dev->fd >= 0) {
    close(dev->fd);
  }
  dev->fd = -1;
  dev->map_base = NULL;
  dev->mmio_base = NULL;
  dev->csr_base = 0;
}

void cnn_mmio_program_default_registers(volatile uint32_t *mmio_base) {
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_CFG_BASE, CNN_MMIO_DEFAULT_CONV_CFG_BASE_W);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_CFG_LEN, CNN_MMIO_DEFAULT_CONV_CFG_WORDS);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_WT_BASE, CNN_MMIO_DEFAULT_CONV_WT_BASE_W);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_WT_LEN, CNN_MMIO_DEFAULT_CONV_WT_WORDS);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_FC_BIAS_BASE, CNN_MMIO_DEFAULT_FC_BIAS_BASE_W);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_FC_BIAS_LEN, CNN_MMIO_DEFAULT_FC_BIAS_WORDS);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_FCW_BASE, CNN_MMIO_DEFAULT_FCW_BASE_W);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_FCW_LEN, CNN_MMIO_DEFAULT_FCW_WORDS);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_IMAGE_BASE, CNN_MMIO_DEFAULT_IMAGE_BASE_W);
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_IMAGE_LEN, CNN_MMIO_DEFAULT_IMAGE_WORDS);
}

void cnn_mmio_write_preload_bundle(volatile uint32_t *mmio_base, const struct cnn_mmio_preload_bundle *
    bundle) {
  uint32_t conv_cfg_base = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_CFG_BASE);
  uint32_t conv_wt_base = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_CONV_WT_BASE);
  uint32_t fc_bias_base = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_FC_BIAS_BASE);
  uint32_t fcw_base = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_FCW_BASE);

  write_word_image(mmio_base, conv_cfg_base, (const uint32_t *)bundle->conv_cfg,
                   CNN_MMIO_DEFAULT_CONV_CFG_WORDS);
  write_word_image(mmio_base, conv_wt_base, bundle->conv_wt,
                   CNN_MMIO_DEFAULT_CONV_WT_WORDS);
  write_word_image(mmio_base, fc_bias_base, (const uint32_t *)bundle->fc_bias,
                   CNN_MMIO_DEFAULT_FC_BIAS_WORDS);
  write_word_image(mmio_base, fcw_base, (const uint32_t *)bundle->fcw,
                   CNN_MMIO_DEFAULT_FCW_WORDS);
}
```

```c

void cnn_mmio_write_inference_case(volatile uint32_t *mmio_base, const struct cnn_mmio_inference_case *tc
    ) {
  uint32_t image_base = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_IMAGE_BASE);

  write_word_image(mmio_base, image_base, tc->image,
                   CNN_MMIO_DEFAULT_IMAGE_WORDS);
}

uint16_t cnn_mmio_read_status(volatile uint32_t *mmio_base) {
  return (uint16_t)(mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_STATUS) & 0xFFFFu);
}

uint16_t cnn_mmio_read_error(volatile uint32_t *mmio_base) {
  return (uint16_t)(mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_IF_ERROR) & 0xFFFFu);
}

uint16_t cnn_mmio_read_predict(volatile uint32_t *mmio_base) {
  return (uint16_t)cnn_mmio_pack_status_predict(cnn_mmio_read_status(mmio_base));
}

void cnn_mmio_read_profile(volatile uint32_t *mmio_base, struct cnn_mmio_profile *profile) {
  if (!profile)
    return;

  profile->l1_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_L1_HI);
  profile->l2_p0_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_L2_P0_HI);
  profile->l2_p1_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_L2_P1_HI);
  profile->l3_p0_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_L3_P0_HI);
  profile->l3_p1_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_L3_P1_HI);
  profile->fc_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_FC_HI);
  profile->argmax_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_ARGMAX_HI);
  profile->total_cycles = mmio_read_cfg_reg(mmio_base, CNN_MMIO_REG_PROFILE_TOTAL_HI);
}

void cnn_mmio_clear_status(volatile uint32_t *mmio_base) {
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONTROL, CNN_MMIO_CTRL_CLEAR_STATUS);
}

void cnn_mmio_start_model_load(volatile uint32_t *mmio_base) {
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONTROL, CNN_MMIO_CTRL_MODEL_LOAD);
}

void cnn_mmio_start_infer(volatile uint32_t *mmio_base) {
  mmio_write_cfg_reg(mmio_base, CNN_MMIO_REG_CONTROL, CNN_MMIO_CTRL_INFER);
}

int cnn_mmio_wait_for_status_bit(
    volatile uint32_t *mmio_base,
    unsigned bit_idx,
    unsigned expected_value,
    int timeout_ms,
    uint16_t *last_status) {
  struct timespec req;
```

```
  int elapsed_ms = 0;
  if (bit_idx >= 16)
    return -1;

  req.tv_sec = 0;
  req.tv_nsec = 1000000L;

  while (elapsed_ms < timeout_ms) {
    uint16_t status = cnn_mmio_read_status(mmio_base);
    if (last_status)
      *last_status = status;
    if (((status >> bit_idx) & 0x1u) == (expected_value & 0x1u))
      return 0;
    nanosleep(&req, NULL);
    elapsed_ms += 1;
  }

  return -1;
}
```

A.4.4 Status Probe Command

Reads STATUS, PREDICT, IF_ERROR, and profile registers for health checks before web requests.

code/tools/hps_mmio_status.c

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "../include/cnn_mmio_host.h"

int main(int argc, char **argv) {
  const char *devmem_path = "/dev/mem";
  uintptr_t csr_base;
  struct cnn_mmio_device dev;
  uint16_t status;
  uint16_t predict;
  uint16_t error_reg;

  if (argc < 2 || argc > 3) {
    fprintf(stderr, "usage: %s <csr_base_hex> [devmem_path]\n", argv[0]);
    return 1;
  }

  csr_base = (uintptr_t)strtoull(argv[1], NULL, 0);
  if (argc > 2)
    devmem_path = argv[2];

  if (cnn_mmio_open(&dev, csr_base, devmem_path) != 0)
    return 1;
```

```

  status = cnn_mmio_read_status(dev.mmio_base);
  predict = cnn_mmio_read_predict(dev.mmio_base);
  error_reg = cnn_mmio_read_error(dev.mmio_base);

  printf("status=0x%04x\n", status);
  printf("model_loaded=%u\n", (unsigned)cnn_mmio_status_model_loaded(status));
  printf("predict_done=%u\n", (unsigned)cnn_mmio_status_predict_done(status));
  printf("predict_class=%u\n", (unsigned)(predict & 0xF));
  printf("error=0x%04x\n", error_reg);

  cnn_mmio_close(&dev);
  return 0;
}
```

A.4.5 Model Load Command

Loads the preload bundle into the scratchpad and triggers CONTROL[0] model replay.

code/tools/hps_mmio_load_model.c

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "../include/cnn_mmio_host.h"

int main(int argc, char **argv) {
  const char *devmem_path = "/dev/mem";
  const char *preload_root;
  uintptr_t csr_base;
  uint16_t status = 0;
  struct cnn_mmio_device dev;
  struct cnn_mmio_preload_bundle preload;

  if (argc < 3 || argc > 4) {
    fprintf(stderr,
            "usage: %s <csr_base_hex> <preload_root> [devmem_path]\n",
            argv[0]);
    return 1;
  }

  csr_base = (uintptr_t)strtoull(argv[1], NULL, 0);
  preload_root = argv[2];
  if (argc > 3)
    devmem_path = argv[3];

  if (cnn_mmio_load_preload_bundle(preload_root, &preload) != 0)
    return 1;
  if (cnn_mmio_open(&dev, csr_base, devmem_path) != 0)
    return 1;
```

```

  cnn_mmio_program_default_registers(dev.mmio_base);
  cnn_mmio_write_preload_bundle(dev.mmio_base, &preload);
  cnn_mmio_clear_status(dev.mmio_base);
  cnn_mmio_start_model_load(dev.mmio_base);

  if (cnn_mmio_wait_for_status_bit(
          dev.mmio_base,
          CNN_MMIO_STATUS_MODEL_LOADED_SHIFT,
          1,
          CNN_MMIO_DEFAULT_TIMEOUT_MS,
          &status) != 0) {
    fprintf(stderr, "timeout waiting for model_loaded, status=0x%04x\n", status);
    cnn_mmio_close(&dev);
    return 1;
  }

  printf("status=0x%04x\n", status);
  printf("model_loaded=%u\n", (unsigned)((status >> CNN_MMIO_STATUS_MODEL_LOADED_SHIFT) & 0x1));
  cnn_mmio_close(&dev);
  return 0;
}
```

A.4.6 FPGA Prediction Command

Writes one exported image case, triggers CONTROL[1] inference, waits for predict_done, and prints web-readable key-value output.

code/tools/hps_mmio_predict.c

```
#define _POSIX_C_SOURCE 200809L

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "../include/cnn_mmio_host.h"

static double monotonic_ms(void) {
  struct timespec ts;
  clock_gettime(CLOCK_MONOTONIC, &ts);
  return (double)ts.tv_sec * 1000.0 + (double)ts.tv_nsec / 1000000.0;
}

int main(int argc, char **argv) {
  const char *devmem_path = "/dev/mem";
  const char *case_root;
  uintptr_t csr_base;
  uint16_t status = 0;
```

```
  uint16_t error_reg = 0;
  struct cnn_mmio_device dev;
  struct cnn_mmio_inference_case tc;
  struct cnn_mmio_profile profile;
  double total_started_ms;
  double case_load_started_ms;
  double case_load_ended_ms;
  double mmio_program_started_ms;
  double mmio_program_ended_ms;
  double infer_started_ms;
  double infer_ended_ms;

  if (argc < 3 || argc > 4) {
    fprintf(stderr, "usage: %s <csr_base_hex> <case_root> [devmem_path]\n", argv[0]);
    return 1;
  }

  csr_base = (uintptr_t)strtoull(argv[1], NULL, 0);
  case_root = argv[2];
  if (argc > 3)
    devmem_path = argv[3];

  total_started_ms = monotonic_ms();
  case_load_started_ms = monotonic_ms();
  if (cnn_mmio_load_inference_case(case_root, &tc) != 0)
    return 1;
  case_load_ended_ms = monotonic_ms();
  if (cnn_mmio_open(&dev, csr_base, devmem_path) != 0)
    return 1;

  mmio_program_started_ms = monotonic_ms();
  cnn_mmio_program_default_registers(dev.mmio_base);
  cnn_mmio_write_inference_case(dev.mmio_base, &tc);
  mmio_program_ended_ms = monotonic_ms();

  infer_started_ms = monotonic_ms();
  cnn_mmio_start_infer(dev.mmio_base);

  if (cnn_mmio_wait_for_status_bit(
          dev.mmio_base,
          CNN_MMIO_STATUS_PREDICT_DONE_SHIFT,
          1,
          CNN_MMIO_DEFAULT_TIMEOUT_MS,
          &status) != 0) {
    fprintf(stderr, "timeout waiting for predict_done, status=0x%04x\n", status);
    cnn_mmio_close(&dev);
    return 1;
  }
  infer_ended_ms = monotonic_ms();

  error_reg = cnn_mmio_read_error(dev.mmio_base);
  cnn_mmio_read_profile(dev.mmio_base, &profile);
  printf("predict_class=%u\n", (unsigned)cnn_mmio_pack_status_predict(status));
  printf("status=0x%04x\n", status);
```

```
  printf("error=0x%04x\n", error_reg);
  printf("l1_cycles=%u\n", profile.l1_cycles);
  printf("l2_p0_cycles=%u\n", profile.l2_p0_cycles);
  printf("l2_p1_cycles=%u\n", profile.l2_p1_cycles);
  printf("l3_p0_cycles=%u\n", profile.l3_p0_cycles);
  printf("l3_p1_cycles=%u\n", profile.l3_p1_cycles);
  printf("fc_cycles=%u\n", profile.fc_cycles);
  printf("argmax_cycles=%u\n", profile.argmax_cycles);
  printf("total_cycles=%u\n", profile.total_cycles);
  printf("board_case_load_ms=%.3f\n", case_load_ended_ms - case_load_started_ms);
  printf("board_program_ms=%.3f\n", mmio_program_ended_ms - mmio_program_started_ms);
  printf("board_infer_wait_ms=%.3f\n", infer_ended_ms - infer_started_ms);
  printf("board_total_ms=%.3f\n", infer_ended_ms - total_started_ms);

  cnn_mmio_close(&dev);
  return (error_reg == 0) ? 0 : 1;
}
```

A.4.7 Board CPU Baseline

Runs the same exported case on the HPS CPU baseline so the web demo can compare CPU and FPGA results.

code/tools/hps_cpu_reference.c

```
#define _POSIX_C_SOURCE 200809L

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define IMG_H 64
#define IMG_W 64
#define C1_OUT 4
#define C2_OUT 8
#define C3_OUT 8
#define FC_OUT 10

#define C1_H 62
#define C1_W 62
#define P1_H 31
#define P1_W 31
#define C2_H 29
#define C2_W 29
#define P2_H 14
#define P2_W 14
#define C3_H 12
#define C3_W 12
```

```c
#define P3_H 6
#define P3_W 6
#define FC_IN 288

static double monotonic_ms(void) {
  struct timespec ts;
  clock_gettime(CLOCK_MONOTONIC, &ts);
  return (double)ts.tv_sec * 1000.0 + (double)ts.tv_nsec / 1000000.0;
}

static int read_i8_lines(const char *path, int8_t *dst, size_t count) {
  FILE *fp = fopen(path, "r");
  size_t i;
  long v;
  if (!fp) {
    perror(path);
    return -1;
  }
  for (i = 0; i < count; ++i) {
    if (fscanf(fp, "%ld", &v) != 1) {
      fprintf(stderr, "short read in %s at index %zu\n", path, i);
      fclose(fp);
      return -1;
    }
    if (v < -128 || v > 127) {
      fprintf(stderr, "i8 out of range in %s at index %zu: %ld\n", path, i, v);
      fclose(fp);
      return -1;
    }
    dst[i] = (int8_t)v;
  }
  fclose(fp);
  return 0;
}

static int read_i32_lines(const char *path, int32_t *dst, size_t count) {
  FILE *fp = fopen(path, "r");
  size_t i;
  long v;
  if (!fp) {
    perror(path);
    return -1;
  }
  for (i = 0; i < count; ++i) {
    if (fscanf(fp, "%ld", &v) != 1) {
      fprintf(stderr, "short read in %s at index %zu\n", path, i);
      fclose(fp);
      return -1;
    }
    dst[i] = (int32_t)v;
  }
  fclose(fp);
  return 0;
}
```

```c

static void build_path(char *dst, size_t dst_len, const char *root, const char *leaf) {
  snprintf(dst, dst_len, "%s/%s", root, leaf);
}

static int8_t clamp_int8(int64_t v) {
  if (v > 127)
    return 127;
  if (v < -128)
    return -128;
  return (int8_t)v;
}

static int idx_hwc(int h, int w, int c, int width, int channels) {
  return ((h * width) + w) * channels + c;
}

static int idx_oihw(int oc, int ic, int kh, int kw, int cin, int ksize) {
  return (((oc * cin + ic) * ksize + kh) * ksize + kw);
}

static int idx_ohw(int oc, int kh, int kw, int ksize) {
  return ((oc * ksize + kh) * ksize + kw);
}

static int32_t requantize_single(int32_t mac,
                                 int32_t eff_bias,
                                 int32_t mult,
                                 int32_t shift,
                                 int32_t zp_out) {
  int64_t acc = (int64_t)mac + (int64_t)eff_bias;
  int64_t prod = acc * (int64_t)mult;
  int64_t scaled;

  if (shift > 0) {
    int64_t round_term = (int64_t)1 << (shift - 1);
    int64_t prod_adj = prod + round_term - (prod < 0 ? 1 : 0);
    scaled = prod_adj >> shift;
  } else if (shift == 0) {
    scaled = prod;
  } else {
    scaled = prod << (-shift);
  }

  return (int32_t)scaled + zp_out;
}

static void conv1_forward(const int8_t *input,
                          const int8_t *weights,
                          const int32_t *bias,
                          const int32_t *mult,
                          const int32_t *shift,
                          int8_t *pool_out) {
  int8_t requant[C1_H * C1_W * C1_OUT];
```

```
  int h, w, oc, kh, kw;

  for (h = 0; h < C1_H; ++h) {
    for (w = 0; w < C1_W; ++w) {
      for (oc = 0; oc < C1_OUT; ++oc) {
        int32_t mac = 0;
        for (kh = 0; kh < 3; ++kh) {
          for (kw = 0; kw < 3; ++kw) {
            int32_t x = input[idx_hwc(h + kh, w + kw, 0, IMG_W, 1)];
            int32_t wt = weights[idx_ohw(oc, kh, kw, 3)];
            mac += x * wt;
          }
        }
        requant[idx_hwc(h, w, oc, C1_W, C1_OUT)] =
            clamp_int8(requantize_single(mac, bias[oc], mult[oc], shift[oc], -128));
      }
    }
  }

  for (h = 0; h < P1_H; ++h) {
    for (w = 0; w < P1_W; ++w) {
      for (oc = 0; oc < C1_OUT; ++oc) {
        int base_h = h * 2;
        int base_w = w * 2;
        int8_t a = requant[idx_hwc(base_h + 0, base_w + 0, oc, C1_W, C1_OUT)];
        int8_t b = requant[idx_hwc(base_h + 0, base_w + 1, oc, C1_W, C1_OUT)];
        int8_t c = requant[idx_hwc(base_h + 1, base_w + 0, oc, C1_W, C1_OUT)];
        int8_t d = requant[idx_hwc(base_h + 1, base_w + 1, oc, C1_W, C1_OUT)];
        int8_t m1 = a > b ? a : b;
        int8_t m2 = c > d ? c : d;
        pool_out[idx_hwc(h, w, oc, P1_W, C1_OUT)] = m1 > m2 ? m1 : m2;
      }
    }
  }
}

static void conv_generic_forward(const int8_t *input,
                                 int in_h,
                                 int in_w,
                                 int in_c,
                                 const int8_t *weights,
                                 int out_c,
                                 const int32_t *bias,
                                 const int32_t *mult,
                                 const int32_t *shift,
                                 int out_h,
                                 int out_w,
                                 int pool_h,
                                 int pool_w,
                                 int8_t *pool_out) {
  int8_t *requant = (int8_t *)malloc((size_t)out_h * (size_t)out_w * (size_t)out_c);
  int h, w, oc, ic, kh, kw;
  if (!requant) {
    fprintf(stderr, "alloc failed for requant buffer\n");
```

```
    exit(1);
  }

  for (h = 0; h < out_h; ++h) {
    for (w = 0; w < out_w; ++w) {
      for (oc = 0; oc < out_c; ++oc) {
        int32_t mac = 0;
        for (ic = 0; ic < in_c; ++ic) {
          for (kh = 0; kh < 3; ++kh) {
            for (kw = 0; kw < 3; ++kw) {
              int32_t x = input[idx_hwc(h + kh, w + kw, ic, in_w, in_c)];
              int32_t wt = weights[idx_oihw(oc, ic, kh, kw, in_c, 3)];
              mac += x * wt;
            }
          }
        }
        requant[idx_hwc(h, w, oc, out_w, out_c)] =
            clamp_int8(requantize_single(mac, bias[oc], mult[oc], shift[oc], -128));
      }
    }
  }

  for (h = 0; h < pool_h; ++h) {
    for (w = 0; w < pool_w; ++w) {
      for (oc = 0; oc < out_c; ++oc) {
        int base_h = h * 2;
        int base_w = w * 2;
        int8_t a = requant[idx_hwc(base_h + 0, base_w + 0, oc, out_w, out_c)];
        int8_t b = requant[idx_hwc(base_h + 0, base_w + 1, oc, out_w, out_c)];
        int8_t c = requant[idx_hwc(base_h + 1, base_w + 0, oc, out_w, out_c)];
        int8_t d = requant[idx_hwc(base_h + 1, base_w + 1, oc, out_w, out_c)];
        int8_t m1 = a > b ? a : b;
        int8_t m2 = c > d ? c : d;
        pool_out[idx_hwc(h, w, oc, pool_w, out_c)] = m1 > m2 ? m1 : m2;
      }
    }
  }

  free(requant);
}

static int fc_argmax(const int8_t *input, const int8_t *weights, const int32_t *bias) {
  int oc, k;
  int best_idx = 0;
  int32_t best_val = 0;

  for (oc = 0; oc < FC_OUT; ++oc) {
    int64_t acc = bias[oc];
    for (k = 0; k < FC_IN; ++k) {
      acc += (int64_t)input[k] * (int64_t)weights[oc * FC_IN + k];
    }
    if (oc == 0 || acc > best_val) {
      best_val = (int32_t)acc;
      best_idx = oc;
```

```
    }
  }

  return best_idx;
}

int main(int argc, char **argv) {
  char path[1024];
  const char *image_case_root;
  const char *reference_case_root;
  int8_t input[IMG_H * IMG_W];
  int8_t pool1[P1_H * P1_W * C1_OUT];
  int8_t pool2[P2_H * P2_W * C2_OUT];
  int8_t pool3[P3_H * P3_W * C3_OUT];
  int8_t conv1_w[C1_OUT * 3 * 3];
  int8_t conv2_w[C2_OUT * C1_OUT * 3 * 3];
  int8_t conv3_w[C3_OUT * C2_OUT * 3 * 3];
  int8_t fc_w[FC_OUT * FC_IN];
  int32_t conv1_bias[C1_OUT], conv2_bias[C2_OUT], conv3_bias[C3_OUT], fc_bias[FC_OUT];
  int32_t conv1_m[C1_OUT], conv2_m[C2_OUT], conv3_m[C3_OUT];
  int32_t conv1_sh[C1_OUT], conv2_sh[C2_OUT], conv3_sh[C3_OUT];
  int predicted_class;
  double started_ms, ended_ms;

  if (argc != 3) {
    fprintf(stderr, "usage: %s <image_case_root> <reference_case_root>\n", argv[0]);
    return 1;
  }

  image_case_root = argv[1];
  reference_case_root = argv[2];

  build_path(path, sizeof(path), image_case_root, "tb_conv1_in_i8_64x64x1.txt");
  if (read_i8_lines(path, input, IMG_H * IMG_W) != 0)
    return 1;

  build_path(path, sizeof(path), reference_case_root, "tb_conv1_w_i8_3x3x4.txt");
  if (read_i8_lines(path, conv1_w, C1_OUT * 3 * 3) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv2_w_i8_3x3x4x8.txt");
  if (read_i8_lines(path, conv2_w, C2_OUT * C1_OUT * 3 * 3) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv3_w_i8_3x3x8x8.txt");
  if (read_i8_lines(path, conv3_w, C3_OUT * C2_OUT * 3 * 3) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_fc_w_i8_10x288.txt");
  if (read_i8_lines(path, fc_w, FC_OUT * FC_IN) != 0)
    return 1;

  build_path(path, sizeof(path), reference_case_root, "tb_conv1_quant_bias_eff_i32_4.txt");
  if (read_i32_lines(path, conv1_bias, C1_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv2_quant_bias_eff_i32_8.txt");
  if (read_i32_lines(path, conv2_bias, C2_OUT) != 0)
```

```
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv3_quant_bias_eff_i32_8.txt");
  if (read_i32_lines(path, conv3_bias, C3_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_fc_bias_eff_i32_10.txt");
  if (read_i32_lines(path, fc_bias, FC_OUT) != 0)
    return 1;

  build_path(path, sizeof(path), reference_case_root, "tb_conv1_quant_M_i32_4.txt");
  if (read_i32_lines(path, conv1_m, C1_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv2_quant_M_i32_8.txt");
  if (read_i32_lines(path, conv2_m, C2_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv3_quant_M_i32_8.txt");
  if (read_i32_lines(path, conv3_m, C3_OUT) != 0)
    return 1;

  build_path(path, sizeof(path), reference_case_root, "tb_conv1_quant_sh_i32_4.txt");
  if (read_i32_lines(path, conv1_sh, C1_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv2_quant_sh_i32_8.txt");
  if (read_i32_lines(path, conv2_sh, C2_OUT) != 0)
    return 1;
  build_path(path, sizeof(path), reference_case_root, "tb_conv3_quant_sh_i32_8.txt");
  if (read_i32_lines(path, conv3_sh, C3_OUT) != 0)
    return 1;

  started_ms = monotonic_ms();
  conv1_forward(input, conv1_w, conv1_bias, conv1_m, conv1_sh, pool1);
  conv_generic_forward(pool1, P1_H, P1_W, C1_OUT, conv2_w, C2_OUT, conv2_bias, conv2_m, conv2_sh,
                       C2_H, C2_W, P2_H, P2_W, pool2);
  conv_generic_forward(pool2, P2_H, P2_W, C2_OUT, conv3_w, C3_OUT, conv3_bias, conv3_m, conv3_sh,
                       C3_H, C3_W, P3_H, P3_W, pool3);
  predicted_class = fc_argmax(pool3, fc_w, fc_bias);
  ended_ms = monotonic_ms();

  printf("predict_class=%d\n", predicted_class);
  printf("board_cpu_infer_ms=%.3f\n", ended_ms - started_ms);
  return 0;
}
```

A.5 Golden Model And Export Code

A.5.1 Golden Batch Entry Point

Regenerates all hardware-aligned digit cases and SRAM preload bundles.

code/Golden-Module/matlab/hardware_aligned/run_all.m

```
% RUN_ALL  Multi-case golden TXT generator.
%
%   Loads the model once, then for every PNG matching `image_glob` runs:
%     export_case  -> hardware_aligned/debug/txt_cases/<case>/
%
%   By default this folder is code-only. It reads the model from
%   <repo>/models/v1.int8.params.mat and images from <repo>/matlab/digit_*.png.

clc; clear; close all;

hw_dir = fileparts(mfilename('fullpath'));
matlab_dir = fileparts(hw_dir);
repo_root = fileparts(matlab_dir);
addpath(hw_dir);

% ---------------------------- USER CONFIG ------------------------------
params_candidates = {
    fullfile(repo_root, 'models', 'v1.int8.params.mat')
    fullfile(matlab_dir, 'main', 'v1.int8.params.mat')
    fullfile(hw_dir, 'v1.int8.params.mat')
};

% Glob all `digit_<n>_*.png` images in matlab/. Adjust as you add cases.
image_glob       = fullfile(matlab_dir, 'digit_*.png');
out_root         = fullfile(hw_dir, 'debug', 'txt_cases');
sram_out_root    = fullfile(hw_dir, 'debug', 'sram_preload');
% -----------------------------------------------------------------------

params_path = "";
for k = 1:numel(params_candidates)
    if exist(params_candidates{k}, 'file')
        params_path = string(params_candidates{k});
        break;
    end
end

if ~exist(params_path, 'file')
    error('run_all: params .mat not found. Tried repo models/, matlab/main/, and hardware_aligned/.');
end

P = load_params(params_path);
imgs = dir(image_glob);
if isempty(imgs)
    error('run_all: no images match %s', image_glob);
end

fprintf('Loaded params from %s\n', params_path);
fprintf('Processing %d image(s) matching %s\n', numel(imgs), image_glob);

for i = 1:numel(imgs)
    img_path = fullfile(imgs(i).folder, imgs(i).name);
    [~, base, ~] = fileparts(imgs(i).name);
```

```
    case_name = sanitize_case_name(base);

    R = export_case(img_path, P, case_name, out_root);
    gen_sram_preload(case_name, R, P, sram_out_root);
end

fprintf('run_all: done.\n');


function s = sanitize_case_name(s)
% Make a filesystem-safe case tag from the image basename.
    s = lower(s);
    s = regexprep(s, '[^a-z0-9_]+', '_');
    s = regexprep(s, '^_+|_+$', '');
end
```

A.5.2 Golden Case Exporter

Exports one source image into input tensors, intermediate tensors, final logits, and manifest data.

code/Golden-Module/matlab/hardware_aligned/export_case.m

```
function R = export_case(image_path, P, case_name, out_root)
% EXPORT_CASE  Run hardware-aligned forward + write all golden TXT for one image.
%
%   R = export_case(image_path, P, case_name, out_root)
%
%   image_path : path to source PNG (any size; converted to 64x64 grayscale)
%   P          : params struct from load_params()
%   case_name  : short tag for the case directory (e.g. 'digit_0')
%   out_root   : root directory; case files go to <out_root>/<case_name>/
%
%   Returns the hw_forward result struct R for further inspection.
%
%   Files written to <out_root>/<case_name>/ follow GOLDEN_FORMAT.md naming
%   `tb_<layer>_<role>_<dtype>_<shape>.txt` plus a `manifest.txt`.

    if nargin < 4 || isempty(out_root)
        matlab_dir = fileparts(mfilename('fullpath'));
        out_root   = fullfile(matlab_dir, 'debug', 'txt_cases');
    end

    case_dir = fullfile(out_root, case_name);
    if ~exist(case_dir, 'dir'), mkdir(case_dir); end

    % ---------------------------------------------------------------------
    %  Load + quantize image to INT8
    % ---------------------------------------------------------------------
    img_uint8 = imread(image_path);
    if ndims(img_uint8) == 3
        img_uint8 = rgb2gray(img_uint8);
```

```
    end
    if size(img_uint8, 1) ~= 64 || size(img_uint8, 2) ~= 64
        img_uint8 = imresize(img_uint8, [64, 64]);
    end
    img_float = single(img_uint8) / 255.0;
    img_q     = int32(round(img_float / P.s_in)) + P.in_zp;
    img_q     = max(min(img_q, int32(127)), int32(-128));
    img_int8  = int8(img_q);

    % ------------------------------------------------------------------
    %  Hardware-aligned forward
    % ------------------------------------------------------------------
    R = hw_forward(img_int8, P);

    % ------------------------------------------------------------------
    %  Write all golden TXT files
    % ------------------------------------------------------------------
    cd_ = @(name) fullfile(case_dir, name);

    % Conv1
    dump_txt(cd_('tb_conv1_in_i8_64x64x1.txt'),         R.input_int8,          'i8',  'hwc');
    dump_txt(cd_('tb_conv1_w_i8_3x3x4.txt'),            squeeze(R.conv1_w),    'i8',  'ohw');   % Cin=1
     squeezed
    dump_txt(cd_('tb_conv1_out_i32_62x62x4.txt'),       R.conv1_mac,           'i32', 'hwc');
    dump_txt(cd_('tb_conv1_quant_bias_eff_i32_4.txt'),  R.conv1_eff_bias,      'i32', 'flat');
    dump_txt(cd_('tb_conv1_quant_M_i32_4.txt'),         R.conv1_M,             'i32', 'flat');
    dump_txt(cd_('tb_conv1_quant_sh_i32_4.txt'),        R.conv1_sh,            'i32', 'flat');
    dump_txt(cd_('tb_conv1_requant_i8_62x62x4.txt'),    R.conv1_requant,       'i8',  'hwc');
    dump_txt(cd_('tb_conv1_pool_i8_31x31x4.txt'),       R.pool1,               'i8',  'hwc');

    % Conv2
    dump_txt(cd_('tb_conv2_in_i8_31x31x4.txt'),         R.pool1,               'i8',  'hwc');
    dump_txt(cd_('tb_conv2_w_i8_3x3x4x8.txt'),          R.conv2_w,             'i8',  'oihw');
    dump_txt(cd_('tb_conv2_out_i32_29x29x8.txt'),       R.conv2_mac,           'i32', 'hwc');
    dump_txt(cd_('tb_conv2_quant_bias_eff_i32_8.txt'),  R.conv2_eff_bias,      'i32', 'flat');
    dump_txt(cd_('tb_conv2_quant_M_i32_8.txt'),         R.conv2_M,             'i32', 'flat');
    dump_txt(cd_('tb_conv2_quant_sh_i32_8.txt'),        R.conv2_sh,            'i32', 'flat');
    dump_txt(cd_('tb_conv2_requant_i8_29x29x8.txt'),    R.conv2_requant,       'i8',  'hwc');
    dump_txt(cd_('tb_conv2_pool_i8_14x14x8.txt'),       R.pool2,               'i8',  'hwc');

    % Conv3
    dump_txt(cd_('tb_conv3_in_i8_14x14x8.txt'),         R.pool2,               'i8',  'hwc');
    dump_txt(cd_('tb_conv3_w_i8_3x3x8x8.txt'),          R.conv3_w,             'i8',  'oihw');
    dump_txt(cd_('tb_conv3_out_i32_12x12x8.txt'),       R.conv3_mac,           'i32', 'hwc');
    dump_txt(cd_('tb_conv3_quant_bias_eff_i32_8.txt'),  R.conv3_eff_bias,      'i32', 'flat');
    dump_txt(cd_('tb_conv3_quant_M_i32_8.txt'),         R.conv3_M,             'i32', 'flat');
    dump_txt(cd_('tb_conv3_quant_sh_i32_8.txt'),        R.conv3_sh,            'i32', 'flat');
    dump_txt(cd_('tb_conv3_requant_i8_12x12x8.txt'),    R.conv3_requant,       'i8',  'hwc');
    dump_txt(cd_('tb_conv3_pool_i8_6x6x8.txt'),         R.pool3,               'i8',  'hwc');

    % FC (no quant -- raw INT32 accumulator with eff_bias, argmax-ready)
    dump_txt(cd_('tb_fc_in_i8_288.txt'),                R.fc_in,               'i8',  'flat');
    dump_txt(cd_('tb_fc_w_i8_10x288.txt'),              R.fc_w,                'i8',  'fc_oik');
    dump_txt(cd_('tb_fc_w_interleaved_i8_288x10.txt'),  R.fc_w,                'i8',  'fc_int_kxo');
```

```
    dump_txt(cd_('tb_fc_bias_eff_i32_10.txt'),          R.fc_eff_bias,          'i32', 'flat');
    dump_txt(cd_('tb_fc_out_i32_10.txt'),               R.fc_acc,               'i32', 'flat');

    % ---------------------------------------------------------------------
    %  Manifest (key=value, GOLDEN_FORMAT.md compliant)
    % ---------------------------------------------------------------------
    manifest_path = fullfile(case_dir, 'manifest.txt');
    fid = fopen(manifest_path, 'w');
    if fid < 0, error('export_case:io', 'cannot open %s', manifest_path); end
    cu = onCleanup(@() fclose(fid));

    [~, img_name, img_ext] = fileparts(image_path);
    fprintf(fid, 'case=%s\n',                           case_name);
    fprintf(fid, 'image_file=%s%s\n',                   img_name, img_ext);

    fprintf(fid, 'conv1_input=tb_conv1_in_i8_64x64x1.txt\n');
    fprintf(fid, 'conv1_weight=tb_conv1_w_i8_3x3x4.txt\n');
    fprintf(fid, 'conv1_output=tb_conv1_out_i32_62x62x4.txt\n');
    fprintf(fid, 'conv1_requant=tb_conv1_requant_i8_62x62x4.txt\n');
    fprintf(fid, 'conv1_pool=tb_conv1_pool_i8_31x31x4.txt\n');
    fprintf(fid, 'conv2_input=tb_conv2_in_i8_31x31x4.txt\n');
    fprintf(fid, 'conv2_weight=tb_conv2_w_i8_3x3x4x8.txt\n');
    fprintf(fid, 'conv2_output=tb_conv2_out_i32_29x29x8.txt\n');
    fprintf(fid, 'conv2_requant=tb_conv2_requant_i8_29x29x8.txt\n');
    fprintf(fid, 'conv2_pool=tb_conv2_pool_i8_14x14x8.txt\n');
    fprintf(fid, 'conv3_input=tb_conv3_in_i8_14x14x8.txt\n');
    fprintf(fid, 'conv3_weight=tb_conv3_w_i8_3x3x8x8.txt\n');
    fprintf(fid, 'conv3_output=tb_conv3_out_i32_12x12x8.txt\n');
    fprintf(fid, 'conv3_requant=tb_conv3_requant_i8_12x12x8.txt\n');
    fprintf(fid, 'conv3_pool=tb_conv3_pool_i8_6x6x8.txt\n');
    fprintf(fid, 'fc_input=tb_fc_in_i8_288.txt\n');
    fprintf(fid, 'fc_weight=tb_fc_w_i8_10x288.txt\n');
    fprintf(fid, 'fc_weight_interleaved=tb_fc_w_interleaved_i8_288x10.txt\n');
    fprintf(fid, 'fc_bias_eff=tb_fc_bias_eff_i32_10.txt\n');
    fprintf(fid, 'fc_output=tb_fc_out_i32_10.txt\n');
    fprintf(fid, 'predict_class=%d\n',                  R.predict_class);

    fprintf(fid, 'input_zero_point=%d\n',               R.zp.in);
    fprintf(fid, 'conv1_output_zero_point=%d\n',        R.zp.conv1_out);
    fprintf(fid, 'conv2_output_zero_point=%d\n',        R.zp.conv2_out);
    fprintf(fid, 'conv3_output_zero_point=%d\n',        R.zp.conv3_out);
    fprintf(fid, 'fc_input_zero_point=%d\n',            R.zp.fc_in);

    fprintf(fid, ['conv_rule=layer input keeps full upstream tensor values; ' ...
                  'current-layer conv MAC uses x_zp=0,w_zp=0,bias=0; ' ...
                  'eff_bias = bias - x_zp*sum(W) added in Quant stage; ' ...
                  'no standalone ReLU (subsumed by Quant clamp when out_zp=-128); ' ...
                  'FC has no requant (argmax on raw INT32 acc)\n']);

    fprintf(fid, 'conv1_input_shape=64x64x1\n');
    fprintf(fid, 'conv2_input_shape=31x31x4\n');
    fprintf(fid, 'conv3_input_shape=14x14x8\n');
    fprintf(fid, 'conv1_output_shape=62x62x4\n');
    fprintf(fid, 'conv2_output_shape=29x29x8\n');
```

```
    fprintf(fid, 'conv3_output_shape=12x12x8\n');
    fprintf(fid, 'fc_output_shape=10\n');

    fprintf(fid, 'conv1_output_range=[%d,%d]\n', min(R.conv1_mac(:)), max(R.conv1_mac(:)));
    fprintf(fid, 'conv2_output_range=[%d,%d]\n', min(R.conv2_mac(:)), max(R.conv2_mac(:)));
    fprintf(fid, 'conv3_output_range=[%d,%d]\n', min(R.conv3_mac(:)), max(R.conv3_mac(:)));
    fprintf(fid, 'fc_acc_range=[%d,%d]\n',       min(R.fc_acc),       max(R.fc_acc));

    fprintf('export_case: wrote case=%s -> %s (predict_class=%d)\n', ...
            case_name, case_dir, R.predict_class);
end
```

A.5.3 Hardware-Aligned Forward Pass

Runs the integer CNN reference using RTL-compatible layer ordering and quantization behavior.

code/Golden-Module/matlab/hardware_aligned/hw_forward.m

```
function R = hw_forward(img_int8, P)
% HW_FORWARD  Hardware-aligned forward pass for one 64x64x1 image.
%
%   R = hw_forward(img_int8, P)
%
%   img_int8 : int8 [64, 64] or [64, 64, 1] -- already quantized input
%   P        : params struct (see load_params helper below)
%
%   Returns struct R with all intermediates needed by the testbench.
%
%   ---------------------------------------------------------------
%   Hardware conventions (vs the TFLite reference rps_conv2.m):
%
%     1. **Conv MAC stage**: x_zp = 0, w_zp = 0, bias = 0.
%        Output is raw INT32 MAC (sum(x_q * w_q)) with no zp folding.
%        These goldens go to tb_convN_out_i32_*.txt.
%
%     2. **Quant stage**: receives eff_bias (NOT raw bias):
%           eff_bias = b - x_zp * sum(W_per_oc)
%        Then: y = clamp(((MAC + eff_bias) * M >> sh) + zp_out, -128, 127)
%        These eff_bias / M / shift goldens go to tb_convN_quant_*_C.txt.
%
%     3. **No standalone ReLU**. With out_zp = -128 the Quant clamp itself
%        is the ReLU; we do NOT call relu() before pooling.
%
%     4. **Pool**: pure 2x2 max, no ReLU fold (use maxpool2x2_int8.m).
%
%     5. **FC**: NO requantization. Raw INT32 accumulator with eff_bias is
%        what the hardware feeds straight into argmax. Golden output is
%        tb_fc_out_i32_<OUT_CHANNELS>.txt; predicted class = argmax(fc_acc).
%   ---------------------------------------------------------------
%
%   Returned fields (every tensor matches the format the TB will read):
```

```
%     R.input_int8                      int8 [64,64,1]
%     R.conv1_w / R.conv2_w / R.conv3_w  int8 weight tensors (OIHW logical)
%     R.conv1_mac / R.conv2_mac / R.conv3_mac
%                                       int32 raw MAC outputs (no bias, no zp)
%     R.conv1_eff_bias / ..._M / ..._sh    int32 quant params per channel
%     R.conv1_requant / R.conv2_requant / R.conv3_requant  int8 quantized
%     R.pool1 / R.pool2 / R.pool3       int8 maxpool outputs
%     R.fc_in                           int8 [288] (= flatten of pool3)
%     R.fc_w                            int8 [10, 288]
%     R.fc_eff_bias                     int32 [10]
%     R.fc_acc                          int32 [10]   <-- HW final, argmax-ready
%     R.predict_class                   0..9
%     R.zp                              struct of zero points used

    if ndims(img_int8) == 2
        img_int8 = reshape(img_int8, size(img_int8,1), size(img_int8,2), 1);
    end
    [imgH, imgW, imgC] = size3(img_int8);
    assert(isa(img_int8, 'int8') && imgH == 64 && imgW == 64 && imgC == 1, ...
           'hw_forward: img must be int8 [64,64] or [64,64,1]');

    R.input_int8 = img_int8;
    R.conv1_w = P.W1;     % [Cout, Cin, H, W] logical (OIHW)
    R.conv2_w = P.W2;
    R.conv3_w = P.W3;
    R.fc_w    = P.W_fc;   % [Cout, K]

    R.zp = struct( ...
        'in',         P.in_zp, ...
        'conv1_out',  P.z_conv1_out, ...
        'conv2_out',  P.z_conv2_out, ...
        'conv3_out',  P.z_conv3_out, ...
        'fc_in',      P.z_fc_in);

    % --- Layer 1 -----------------------------------------------------------
    % R.conv*_sh is the HARDWARE-facing right-shift amount consumed by
    % Quantization_PE.v (`correct >>> sh`), so we pre-convert from the
    % TFLite exponent `shift` to `31 - shift` = total right-shift count.
    R.conv1_mac      = conv2d_mac(img_int8, P.W1);                      % int32 [62,62,4]
    R.conv1_eff_bias = compute_eff_bias(P.b1, P.W1, P.in_zp);           % int32 [4]
    R.conv1_M        = int32(P.qm1);
    R.conv1_sh       = int32(int32(31) - int32(P.shift1));
    R.conv1_requant  = quant_int32_to_int8(R.conv1_mac, R.conv1_eff_bias, ...
                                           R.conv1_M, R.conv1_sh, P.z_conv1_out);
    R.pool1          = maxpool2x2_int8(R.conv1_requant);                % int8 [31,31,4]

    % --- Layer 2 -----------------------------------------------------------
    R.conv2_mac      = conv2d_mac(R.pool1, P.W2);                       % int32 [29,29,8]
    R.conv2_eff_bias = compute_eff_bias(P.b2, P.W2, P.z_conv1_out);     % int32 [8]
    R.conv2_M        = int32(P.qm2);
    R.conv2_sh       = int32(int32(31) - int32(P.shift2));
    R.conv2_requant  = quant_int32_to_int8(R.conv2_mac, R.conv2_eff_bias, ...
                                           R.conv2_M, R.conv2_sh, P.z_conv2_out);
    R.pool2          = maxpool2x2_int8(R.conv2_requant);                % int8 [14,14,8]
```

```
88
89      % --- Layer 3 ------------------------------------------------------------
90      R.conv3_mac      = conv2d_mac(R.pool2, P.W3);                        % int32 [12,12,8]
91      R.conv3_eff_bias = compute_eff_bias(P.b3, P.W3, P.z_conv2_out);      % int32 [8]
92      R.conv3_M        = int32(P.qm3);
93      R.conv3_sh       = int32(int32(31) - int32(P.shift3));
94      R.conv3_requant  = quant_int32_to_int8(R.conv3_mac, R.conv3_eff_bias, ...
95                                             R.conv3_M, R.conv3_sh, P.z_conv3_out);
96      R.pool3          = maxpool2x2_int8(R.conv3_requant);                 % int8 [6,6,8]
97
98      % --- Flatten + FC -------------------------------------------------------
99      R.fc_in = flatten_nhwc_int8(R.pool3);   % int8 [288]
100     K       = numel(R.fc_in);
101     Cout    = size(P.W_fc, 1);
102     assert(size(P.W_fc, 2) == K, 'FC weight K (%d) must match flatten (%d)', size(P.W_fc,2), K);
103     assert(numel(P.b_fc)   == Cout, 'FC bias must have length %d', Cout);
104
105     R.fc_eff_bias = compute_fc_eff_bias(P.b_fc, P.W_fc, P.z_fc_in);   % int32 [Cout]
106
107     % FC accumulator (hardware path): acc[oc] = eff_bias[oc] + sum(x * w[oc,:])
108     % where x is the raw INT8 stored in SRAM_B (zp already baked in).
109     x_i32 = int32(R.fc_in(:));
110     acc   = zeros(Cout, 1, 'int32');
111     for oc = 1:Cout
112         w_i32 = int32(P.W_fc(oc, :)).';
113         % Use int64 accumulation internally to avoid intermediate overflow,
114         % then saturate to int32. RTL accumulator is 32 bits; saturation here
115         % matches what the hardware would silently wrap, so flag any case
116         % where it actually exceeds 32-bit range (would indicate a model bug).
117         s64 = int64(R.fc_eff_bias(oc)) + sum(int64(x_i32) .* int64(w_i32));
118         if s64 > intmax('int32') || s64 < intmin('int32')
119             warning('hw_forward:fc_overflow', ...
120                     'FC oc=%d accumulator %d out of int32; saturating', oc, s64);
121         end
122         acc(oc) = int32(max(min(s64, int64(intmax('int32'))), int64(intmin('int32'))));
123     end
124     R.fc_acc = acc;
125
126     % Argmax: strict-greater so ties go to the lower index (matches RTL ST_ARGMAX).
127     best_idx = int32(0);   % 0-based class id (matches predict_class[3:0])
128     best_val = R.fc_acc(1);
129     for oc = 2:Cout
130         if R.fc_acc(oc) > best_val
131             best_val = R.fc_acc(oc);
132             best_idx = int32(oc - 1);
133         end
134     end
135     R.predict_class = best_idx;
136 end
137
138
139 % ========================================================================
140 %  Helpers
141 % ========================================================================
```

```
142
143 function mac = conv2d_mac(x_int8, W_oihw)
144 % Raw MAC convolution: x_zp = 0, w_zp = 0, bias = 0; INT32 output.
145 % W_oihw: int8 [Cout, Cin, H, W] (OIHW).
146 % Stride 1, valid padding.
147
148     [Cout, Cin, kH, kW] = size_oihw(W_oihw);
149     [Hi, Wi, Ci] = size3(x_int8);
150     assert(Cin == Ci, 'conv2d_mac: weight Cin (%d) != input C (%d)', Cin, Ci);
151
152     Ho = Hi - kH + 1;
153     Wo = Wi - kW + 1;
154     mac = zeros(Ho, Wo, Cout, 'int32');
155
156     x32 = int32(x_int8);
157     W32 = int32(W_oihw);
158
159     for oc = 1:Cout
160         for r = 1:Ho
161             for c = 1:Wo
162                 s = int64(0);
163                 for ic = 1:Cin
164                     patch = x32(r:r+kH-1, c:c+kW-1, ic);
165                     ker   = squeeze(W32(oc, ic, :, :));   % [kH, kW]
166                     s     = s + sum(int64(patch(:)) .* int64(ker(:)));
167                 end
168                 if s > intmax('int32') || s < intmin('int32')
169                     warning('hw_forward:mac_overflow', ...
170                             'MAC overflow at oc=%d r=%d c=%d : %d', oc, r, c, s);
171                 end
172                 mac(r, c, oc) = int32(s);
173             end
174         end
175     end
176 end
177
178
179 function eb = compute_eff_bias(bias, W_oihw, x_zp)
180 % eff_bias[oc] = bias[oc] - x_zp * sum(W[oc, :, :, :])
181 % W_oihw: int8 [Cout, Cin, H, W]
182     Cout = size(W_oihw, 1);
183     eb = zeros(Cout, 1, 'int32');
184     xz = int64(x_zp);
185     for oc = 1:Cout
186         sw = sum(int64(W_oihw(oc, :, :, :)), 'all');
187         eb(oc) = int32(int64(bias(oc)) - xz * sw);
188     end
189 end
190
191
192 function eb = compute_fc_eff_bias(bias, W_fc, x_zp)
193 % eff_bias[oc] = bias[oc] - x_zp * sum(W_fc[oc, :])
194 % W_fc: int8 [Cout, K]
195     Cout = size(W_fc, 1);
```

```
    eb = zeros(Cout, 1, 'int32');
    xz = int64(x_zp);
    for oc = 1:Cout
        sw = sum(int64(W_fc(oc, :)));
        eb(oc) = int32(int64(bias(oc)) - xz * sw);
    end
end


function out = quant_int32_to_int8(mac_i32, eff_bias_i32, M_i32, sh_i32, zp_out)
% Mirror of RTL Quantization_PE.v requant:
%   acc     = mac + eff_bias                // signed 32b
%   mul     = acc * M                       // signed 64b
%   correct = mul + (1<<(sh-1)) - (mul<0?1:0)
%   shiftt  = correct >>> sh                // arithmetic right shift
%   y       = shiftt - 128                  // zp_out = -128 baked in
%   out     = clamp(y, -128, 127)
%
% sh_i32 is the HARDWARE-native shift (= 31 - TFLite_exp). Passed through
% verbatim. For typical conv layers this is a positive value ~39..41.
%
% mac_i32:      int32 [H, W, C]
% eff_bias_i32: int32 [C]
% M_i32:        int32 [C]
% sh_i32:       int32 [C]   (RTL-style right-shift amount)
% zp_out:       int32 scalar
    [H, W, C] = size3(mac_i32);
    out = zeros(H, W, C, 'int8');
    for c = 1:C
        acc64 = int64(mac_i32(:, :, c)) + int64(eff_bias_i32(c));
        prod  = acc64 * int64(M_i32(c));
        sh    = int64(sh_i32(c));
        if sh > 0
            round_term = bitshift(int64(1), sh - 1);
            prod_adj   = prod + round_term - int64(prod < 0);
            scaled     = bitshift(prod_adj, -sh);
        elseif sh == 0
            scaled = prod;
        else
            scaled = bitshift(prod, -sh);
        end
        scaled = scaled + int64(zp_out);
        scaled(scaled >  127) =  127;
        scaled(scaled < -128) = -128;
        out(:, :, c) = int8(scaled);
    end
end


function [Cout, Cin, kH, kW] = size_oihw(W)
    sz = size(W);
    Cout = sz(1); Cin = sz(2); kH = sz(3); kW = sz(4);
end

```

```
250
251 function [H, W, C] = size3(x)
252     if ismatrix(x)
253         x = reshape(x, size(x,1), size(x,2), 1);
254     end
255     sz = size(x);
256     H = sz(1); W = sz(2);
257     if numel(sz) >= 3, C = sz(3); else, C = 1; end
258 end
```

A.5.4 SRAM Preload Packer

Packs convolution configuration, convolution weights, FC weights, and FC bias for HPS model replay.

code/Golden-Module/matlab/hardware_aligned/gen_sram_preload.m

```
1  function gen_sram_preload(case_name, R, P, out_root)
2  % GEN_SRAM_PRELOAD  Produce the four host preload streams the TB drives in.
3  %
4  %   gen_sram_preload(case_name, R, P, out_root)
5  %
6  %   case_name : tag (e.g. 'digit_0')
7  %   R         : forward result struct from hw_forward / export_case
8  %   P         : params struct from load_params
9  %   out_root  : root output dir (default: matlab/main/debug/sram_preload/)
10 %
11 %   Writes to <out_root>/<case_name>/:
12 %     preload_conv_cfg_<NCFG>w.txt     conv L1/L2/L3 cfg (45 host words to SRAM_A@0x000)
13 %     preload_conv_wt_225w_bytes.txt   conv L1/L2/L3 weights (900 INT8 bytes = 225 packed words)
14 %     preload_fc_bias_10w.txt          FC bias eff (10 host words to SRAM_A@0x111 via LAYER_FC+SEL_CFG)
15 %     preload_fcw_864w.txt             FC weight host stream (864 words -> packer -> 288 x 80b)
16 %
17 %   Each host word is one INT32 line (signed decimal) -- the TB feeds these
18 %   directly into load_data[31:0] one per beat. Host words are signed but the
19 %   bit-pattern is what the wrapper writes to SRAM.
20 %
21 %   ---------------------------------------------------------------
22 %   STATUS NOTE (read before relying on this script):
23 %
24 %     * FC bias  (10w)   : FULLY IMPLEMENTED
25 %     * FC weight (864w) : FULLY IMPLEMENTED -- matches RTL fcw_preload_packer
26 %                          spec (3 host words -> 80-bit slot, ch0 in LSB)
27 %     * Conv weight (900B) : IMPLEMENTED via the reorder ported from
28 %                            matlab_old/gen_sram_preload.py. TB packs this
29 %                            byte stream into 225 host words.
30 %     * Conv cfg (45w)   : SCAFFOLD ONLY. Each layer-pass uses 9 cfg words,
31 %                          but the bit-packing inside each word depends on
32 %                          how Quantization_Top.v unpacks them. See the TODO
33 %                          inside conv_cfg_stream() below; consult the per-
```

```
%                       layer cfg writer in the standalone TBs to verify.
%   ------------------------------------------------------------

    if nargin < 4 || isempty(out_root)
        matlab_dir = fileparts(mfilename('fullpath'));
        out_root   = fullfile(matlab_dir, 'debug', 'sram_preload');
    end
    case_dir = fullfile(out_root, case_name);
    if ~exist(case_dir, 'dir'), mkdir(case_dir); end

    % ---------------------------------------------------------------------
    %  conv CFG: 45 words = 9*5 (L1, L2_p0, L2_p1, L3_p0, L3_p1)
    % ---------------------------------------------------------------------
    cfg_stream = [conv_cfg_stream(R.conv1_eff_bias, R.conv1_M, R.conv1_sh, R.zp.conv1_out, 'L1');
                  conv_cfg_stream(R.conv2_eff_bias(1:4), R.conv2_M(1:4), R.conv2_sh(1:4), R.zp.conv2_out,
      'L2p0');
                  conv_cfg_stream(R.conv2_eff_bias(5:8), R.conv2_M(5:8), R.conv2_sh(5:8), R.zp.conv2_out,
      'L2p1');
                  conv_cfg_stream(R.conv3_eff_bias(1:4), R.conv3_M(1:4), R.conv3_sh(1:4), R.zp.conv3_out,
      'L3p0');
                  conv_cfg_stream(R.conv3_eff_bias(5:8), R.conv3_M(5:8), R.conv3_sh(5:8), R.zp.conv3_out,
      'L3p1')];
    assert(numel(cfg_stream) == 45, 'conv_cfg stream length %d != 45', numel(cfg_stream));
    dump_txt(fullfile(case_dir, 'preload_conv_cfg_45w.txt'), cfg_stream, 'i32', 'flat');

    % ---------------------------------------------------------------------
    %  conv WT: 900 bytes = 4*(9 + 36*2 + 72*2), later packed to 225 host words
    % ---------------------------------------------------------------------
    wt_stream = [conv_wt_pass(P.W1, 9,  1, 1, 4);   % L1: dot_k=9,  Cin=1, 4 channels (single pass)
                 conv_wt_pass(P.W2, 36, 4, 1, 4);   % L2 pass 0: dot_k=36, Cin=4, ch 1..4
                 conv_wt_pass(P.W2, 36, 4, 5, 4);   % L2 pass 1: ch 5..8
                 conv_wt_pass(P.W3, 72, 8, 1, 4);   % L3 pass 0: dot_k=72, Cin=8, ch 1..4
                 conv_wt_pass(P.W3, 72, 8, 5, 4)];  % L3 pass 1: ch 5..8
    assert(numel(wt_stream) == 4 * (9 + 36*2 + 72*2), 'conv wt stream length %d != 900 bytes', numel(
     wt_stream));
    % wt_stream is byte-stream (each entry one INT8 byte); host word packs 4 bytes,
    % so the file ends up as 225 host beats only after 4-byte packing in the TB.
    % Until the TB's packer is hooked here, dump byte-by-byte for transparency.
    dump_txt(fullfile(case_dir, 'preload_conv_wt_225w_bytes.txt'), wt_stream, 'i8', 'flat');

    % ---------------------------------------------------------------------
    %  FC bias: 10 INT32 words. Goes to SRAM_A @ 0x111 via LAYER_FC+SEL_CFG.
    % ---------------------------------------------------------------------
    dump_txt(fullfile(case_dir, 'preload_fc_bias_10w.txt'), R.fc_eff_bias, 'i32', 'flat');

    % ---------------------------------------------------------------------
    %  FC weight: 864 host words for fcw_preload_packer (3 host words per
    %  80-bit slot). For each k in 0..287, emit 3 host words:
    %    word0 [31:0] = {k3, k2, k1, k0}     (LSB byte = ch 0)
    %    word1 [31:0] = {k7, k6, k5, k4}
    %    word2 [15:0] = {k9, k8}            (high 16b zero-padded)
    %  This matches input/RTL/SRAM/fcw_preload_packer.v.
    % ---------------------------------------------------------------------
    Cout = size(P.W_fc, 1);
```

```
    K    = size(P.W_fc, 2);
    assert(Cout == 10 && K == 288, 'FC weight expected [10,288], got [%d,%d]', Cout, K);
    fcw_stream = zeros(K * 3, 1, 'int32');
    for k = 1:K
        % bytes for 10 channels at this k position
        bytes = int32(P.W_fc(:, k));   % [10, 1]
        bytes = bitand(bytes, int32(255));   % treat as unsigned for bit-packing
        word0 = bitor(bitor(bitor(bytes(1), bitshift(bytes(2), 8)), ...
                            bitshift(bytes(3), 16)), bitshift(bytes(4), 24));
        word1 = bitor(bitor(bytes(5), bitshift(bytes(6), 8)), ...
                            bitshift(bytes(7), 16)), bitshift(bytes(8), 24));
        word2 = bitor(bytes(9), bitshift(bytes(10), 8));   % upper 16b = 0
        % word0/1/2 are already int32 with the correct bit pattern.
        % Do NOT pass through uint32() - MATLAB saturates negatives to 0.
        fcw_stream(3*(k-1)+1) = word0;
        fcw_stream(3*(k-1)+2) = word1;
        fcw_stream(3*(k-1)+3) = word2;
    end
    assert(numel(fcw_stream) == 864, 'fcw stream length %d != 864', numel(fcw_stream));
    dump_txt(fullfile(case_dir, 'preload_fcw_864w.txt'), fcw_stream, 'i32', 'flat');

    fprintf('gen_sram_preload: wrote 4 streams to %s\n', case_dir);
end


% ========================================================================
%  Helpers
% ========================================================================

function out = conv_cfg_stream(eff_bias, M, sh, zp_out, tag)   %#ok<INUSD>
% Pack 9 host words for one conv pass cfg, matching input/RTL/quant_pool/
% integration/quant_param_loader.v (lines 6-8 header comment):
%
%   word 0..3 : bias0..3 (per-channel eff_bias, one INT32 per word)
%   word 4..7 : M0..3    (per-channel TFLite multiplier, one INT32 per word)
%   word 8    : sh_packed = {sh1[31:24], sh2[23:16], sh3[15:8], sh4[7:0]}
%               i.e. ch1 is MSB, ch4 is LSB (Verilog concat order).
%
% Verified against Quantization_Top.v:32-53 which slices sh_in[31:24]->PE1,
% [23:16]->PE2, [15:8]->PE3, [7:0]->PE4.
    assert(numel(eff_bias) == 4 && numel(M) == 4 && numel(sh) == 4, ...
           'conv_cfg_stream: expected 4 channels per pass, got [%d,%d,%d] (%s)', ...
           numel(eff_bias), numel(M), numel(sh), tag);

    sh8 = bitand(int32(sh(:)), int32(255));   % truncate each shift to 8 bits
    sh_packed = bitor(bitor(bitor( ...
        bitshift(sh8(1), 24), ...
        bitshift(sh8(2), 16)), ...
        bitshift(sh8(3),  8)), ...
                    sh8(4));

    out       = int32(zeros(9, 1));
    out(1:4)  = int32(eff_bias(:));
    out(5:8)  = int32(M(:));
```

```
    out(9)    = int32(sh_packed);
end


function bytes = conv_wt_pass(W_oihw, dot_k, Cin, ch_start, cols)
% Reorder + interleave one pass worth of weights for the systolic array.
% Ported from matlab_old/gen_sram_preload.py (reorder_weights + interleave_for_sram).
%
% W_oihw : int8 [Cout_total, Cin, 3, 3]
% dot_k  : 9 (L1) / 36 (L2) / 72 (L3) -- inner MAC sequence length
% Cin    : 1 / 4 / 8
% ch_start, cols : take ch_start..ch_start+cols-1 output channels for this pass
%
% Returns INT8 column vector of length cols*dot_k, byte-interleaved as the
% SRAM stores them: for each k position, emit col0_k, col1_k, ..., col(cols-1)_k.

    assert(size(W_oihw, 2) == Cin, 'conv_wt_pass: W Cin %d != expected %d', size(W_oihw,2), Cin);

    % Build w_tap[ch][rd_col_idx] in the SA traversal order.
    w_tap = zeros(cols, dot_k, 'int8');
    for ch = 0:cols-1
        oc = ch_start + ch;        % 1-based oc index
        for rd_col_idx = 0:dot_k-1
            c_in_i = mod(rd_col_idx, Cin);
            kw     = mod(floor(rd_col_idx / Cin), 3);
            kh     = 2 - floor(rd_col_idx / (3 * Cin));
            % MATLAB indices are 1-based and stored as OIHW.
            w_tap(ch+1, rd_col_idx+1) = W_oihw(oc, c_in_i+1, kh+1, kw+1);
        end
    end

    % Interleave: for each k, emit cols bytes (col0..col_{cols-1}).
    bytes = zeros(cols * dot_k, 1, 'int8');
    idx = 1;
    for k = 1:dot_k
        for ch = 1:cols
            bytes(idx) = w_tap(ch, k);
            idx = idx + 1;
        end
    end
end
```

A.5.5 Golden Parameter Loader

Loads the exported quantized parameters consumed by the MATLAB golden flow.

code/Golden-Module/matlab/hardware_aligned/load_params.m

```
function P = load_params(params_mat_path)
% LOAD_PARAMS  Load TFLite-exported INT8 model into hw_forward-ready struct.
%
```

```
%   P = load_params(params_mat_path)
%
%   Reads .mat produced by pytorch/export_tflite_params_mat.py with the
%   conventions documented in matlab/main/rps_conv2.m:
%
%     values{11}/{10} : conv1 weight / bias  (weight TFLite OHWI -> we permute to OIHW)
%     values{9}/{8}   : conv2 weight / bias
%     values{7}/{6}   : conv3 weight / bias
%     values{5}/{4}   : fc weight / bias  (already [Cout, K])
%
%   Output struct P:
%     P.W1, P.W2, P.W3   int8 weights in OIHW [Cout, Cin, H, W]
%     P.b1, P.b2, P.b3   int32 raw biases
%     P.W_fc             int8 [Cout=10, K=288]
%     P.b_fc             int32 [10]
%     P.in_zp                                 int32 input zero point
%     P.z_conv1_out / P.z_conv2_out / P.z_conv3_out  int32 quant zps
%     P.z_fc_in                               int32
%     P.qm1 / P.shift1                        int32 (per-channel from TFLite multiplier)
%     P.qm2 / P.shift2
%     P.qm3 / P.shift3
%     P.s_in                                   single (input scale, used for image quant)
%
%   FC quantization (qm_fc / shift_fc / z_fc_out) is intentionally OMITTED:
%   the hardware FC path has no requantization stage, argmax runs on the raw
%   INT32 accumulator. See matlab/main/hw_forward.m header for the rule.

    if ~exist(params_mat_path, 'file')
        error('load_params:missing', 'params .mat not found: %s', params_mat_path);
    end
    S = load(params_mat_path);

    % --- Conv weights: TFLite OHWI -> OIHW [Cout, Cin, H, W] -------------
    P.W1 = int8(permute(S.values{11}, [1, 4, 2, 3]));   % OHWI -> OIHW
    P.W2 = int8(permute(S.values{9},  [1, 4, 2, 3]));
    P.W3 = int8(permute(S.values{7},  [1, 4, 2, 3]));

    P.b1 = int32(S.values{10}(:));
    P.b2 = int32(S.values{8}(:));
    P.b3 = int32(S.values{6}(:));

    % --- FC weight / bias ------------------------------------------------
    P.W_fc = int8(S.values{5});      % [Cout, K]
    P.b_fc = int32(S.values{4}(:));

    % --- Zero points / scales --------------------------------------------
    P.in_zp        = int32(S.input_zero_points{1}(1));
    P.s_in         = single(S.input_scales{1}(1));
    P.z_conv1_out  = int32(S.activation_zero_points{1}(1));
    P.z_conv2_out  = int32(S.activation_zero_points{3}(1));
    P.z_conv3_out  = int32(S.activation_zero_points{5}(1));
    P.z_fc_in      = int32(S.activation_zero_points{10}(1));

    s_conv1_out = S.activation_scales{1};
```

```
    s_conv2_out = S.activation_scales{3};
    s_conv3_out = S.activation_scales{5};

    sw1 = S.scales{11};
    sw2 = S.scales{9};
    sw3 = S.scales{7};

    % Per-channel multiplier/shift via TFLite formula.
    [~, qm1, sh1] = tflite_quantize_multiplier(P.s_in,        sw1, s_conv1_out);
    [~, qm2, sh2] = tflite_quantize_multiplier(s_conv1_out,   sw2, s_conv2_out);
    [~, qm3, sh3] = tflite_quantize_multiplier(s_conv2_out,   sw3, s_conv3_out);

    P.qm1    = int32(qm1(:));    P.shift1 = int32(sh1(:));
    P.qm2    = int32(qm2(:));    P.shift2 = int32(sh2(:));
    P.qm3    = int32(qm3(:));    P.shift3 = int32(sh3(:));
end
```

A.5.6 Golden TXT Writer

Writes integer tensors in the one-value-per-line format consumed by RTL testbenches and HPS tools.

code/Golden-Module/matlab/hardware_aligned/dump_txt.m

```
function dump_txt(filepath, tensor, dtype, layout)
% DUMP_TXT  Write a tensor as one decimal int per line (GOLDEN_FORMAT.md).
%
%   dump_txt(filepath, tensor, dtype, layout)
%
%   filepath : output path (will overwrite)
%   tensor   : numeric array (any size)
%   dtype    : 'i8' or 'i32' (validates element range)
%   layout   : (optional) traversal hint:
%       'hwc'   - 3D activation [H, W, C], outer->inner = H, W, C (default for ndims=3)
%       'oihw'  - 4D weight [Cout, Cin, H, W] (default for ndims=4)
%       'ohw'   - 3D weight  [Cout, H, W]    (e.g. conv1 with Cin=1, after squeeze)
%       'flat'  - just iterate column-major (default for vectors / matrices)
%       'fc_oik'- 2D FC weight [Cout, K], outer=Cout, inner=K (default for 2D)
%       'fc_int_kxo' - SRAM-interleaved FC weight: outer=K, inner=Cout
%                      (writes Cout bytes per K position, LSB=ch0)
%
%   The tensor argument is expected to already be in the *logical* layout
%   listed above (e.g. an OIHW tensor must be passed as a [Cout, Cin, H, W]
%   numeric array). The function reorders elements into the file's
%   slow->fast order before writing.
%
%   GOLDEN_FORMAT.md compliance:
%     * one signed decimal integer per line, LF terminator
%     * no header, no comments, no blank lines
%     * file ends with a trailing newline
```

```
%     * line count == numel(tensor)

    if nargin < 4 || isempty(layout)
        if ndims(tensor) == 4
            layout = 'oihw';
        elseif ndims(tensor) == 3
            layout = 'hwc';
        elseif ismatrix(tensor) && ~isvector(tensor)
            layout = 'fc_oik';
        else
            layout = 'flat';
        end
    end

    switch lower(dtype)
        case 'i8'
            lo = -128; hi = 127;
        case 'i32'
            lo = -2147483648; hi = 2147483647;
        otherwise
            error('dump_txt:dtype', 'dtype must be ''i8'' or ''i32'', got "%s"', dtype);
    end

    % Reorder into the file emission sequence.
    switch lower(layout)
        case 'hwc'
            assert(ndims(tensor) <= 3, 'hwc layout expects up to 3D tensor');
            % MATLAB stores [H, W, C] in column-major (H fastest, then W, then C).
            % File order is H outer, W middle, C fastest -> permute then linearize.
            t = permute(tensor, [3, 2, 1]);   % now [C, W, H], column-major == file order
            seq = t(:);

        case 'oihw'
            assert(ndims(tensor) == 4, 'oihw layout expects 4D tensor');
            % File order: Cout outer, Cin, H, W fastest.
            t = permute(tensor, [4, 3, 2, 1]); % [W, H, Cin, Cout] in column-major
            seq = t(:);

        case 'ohw'
            assert(ndims(tensor) == 3, 'ohw layout expects 3D tensor');
            % File order: Cout outer, H, W fastest.
            t = permute(tensor, [3, 2, 1]);   % [W, H, Cout] in column-major
            seq = t(:);

        case 'fc_oik'
            assert(ismatrix(tensor), 'fc_oik layout expects 2D [Cout, K] tensor');
            % File order: Cout outer, K fastest.
            t = tensor.';   % [K, Cout]; column-major == [Cout][K] with K fast
            seq = t(:);

        case 'fc_int_kxo'
            assert(ismatrix(tensor), 'fc_int_kxo layout expects 2D [Cout, K] tensor');
            % File order: K outer, Cout fastest. SRAM_FCW has one 80-bit slot
            % per K position; bytes within slot are ch0 (LSB) .. ch9 (MSB).
```

```
            % Writing column-major over the original [Cout, K] = ch0,ch1,..,ch9
            % per K position is exactly what we want.
            seq = tensor(:);

        case 'flat'
            seq = tensor(:);

        otherwise
            error('dump_txt:layout', 'unknown layout "%s"', layout);
    end

    % Range check (after cast to a safe wide type).
    seq_i64 = int64(seq);
    if any(seq_i64 < lo) || any(seq_i64 > hi)
        bad_idx = find(seq_i64 < lo | seq_i64 > hi, 1, 'first');
        error('dump_txt:range', ...
              'value %d at element %d out of %s range [%d, %d] (file %s)', ...
              seq_i64(bad_idx), bad_idx, dtype, lo, hi, filepath);
    end

    % Make sure parent directory exists.
    parent = fileparts(filepath);
    if ~isempty(parent) && ~exist(parent, 'dir')
        mkdir(parent);
    end

    % Write: one decimal int per line, LF, trailing newline.
    fid = fopen(filepath, 'w');
    if fid < 0
        error('dump_txt:io', 'cannot open %s for write', filepath);
    end
    cleanup = onCleanup(@() fclose(fid));
    fprintf(fid, '%d\n', seq_i64);
end
```

A.5.7 Flatten Helper

Flattens NHWC feature maps into the FC input order expected by the RTL datapath.

code/Golden-Module/matlab/hardware_aligned/flatten_nhwc_int8.m

```
function y = flatten_nhwc_int8(x)
%FLATTEN_NHWC_INT8 Flatten HxWxC int8 tensor with NHWC logical order.
%   x: int8 [H, W, C]
%   y: int8 [1, H*W*C]
%
% TFLite Flatten on [1,H,W,C] follows NHWC contiguous order:
%   h major, then w, and c is the fastest-changing index.

    assert(isa(x, 'int8'), 'x must be int8');
    assert(ndims(x) == 3, 'x must be HxWxC');
```

```

    [H, W, C] = size(x);
    y = zeros(1, H * W * C, 'int8');

    k = 1;
    for h = 1:H
        for w = 1:W
            for c = 1:C
                y(k) = x(h, w, c);
                k = k + 1;
            end
        end
    end
end
```

A.5.8 Golden Max-Pool Helper

Implements the 2-by-2 INT8 max-pooling operation used by the golden CNN path.

code/Golden-Module/matlab/hardware_aligned/maxpool2x2_int8.m

```
function out = maxpool2x2_int8(x)
% MAXPOOL2X2_INT8  Pure 2x2 stride-2 max pooling (no ReLU fold).
%
%   out = maxpool2x2_int8(x)
%
%   x   : int8 [H, W, C]
%   out : int8 [floor(H/2), floor(W/2), C]
%
%   Hardware alignment:
%     * The accelerator's pool stage is plain max over a 2x2 window.
%     * ReLU is implicit in the upstream Quantization stage when out_zp = -128
%       (clamp(.., -128, 127) == max(.., zp_neg)). We do NOT clamp again here.
%
%   Use this in the hardware-aligned forward pass instead of
%   `relu_maxpool2x2_int8` (which fuses an explicit ReLU into the window).

    assert(isa(x, 'int8'), 'maxpool2x2_int8: input must be int8');
    [H, W, C] = size(x);
    Ho = floor(H / 2);
    Wo = floor(W / 2);
    out = zeros(Ho, Wo, C, 'int8');

    for c = 1:C
        for r = 1:Ho
            for cc = 1:Wo
                window = x(2*r-1:2*r, 2*cc-1:2*cc, c);
                out(r, cc, c) = max(window(:));
            end
        end
    end
```

```
end
```

A.5.9 TFLite Multiplier Helper

Computes TFLite-style fixed-point quantized multipliers and shifts.

code/Golden-Module/matlab/hardware_aligned/tflite_quantize_multiplier.m

```
function [M, qm, shift] = tflite_quantize_multiplier(S_in, S_w, S_out)
%TFLITE_QUANTIZE_MULTIPLIER  Compute per-channel real multiplier and
%TFLite-style (quantized_multiplier, shift) for int32->int8 requant.
%
%   M[c]  = (S_in * S_w[c]) / S_out[c]
%   M ~= (qm / 2^31) * 2^shift
%
% Inputs:
%   S_in  : single/double scalar
%   S_w   : single/double scalar or vector (per-channel)
%   S_out : single/double scalar or vector (per-channel)
%
% Outputs:
%   M     : double vector of real multipliers
%   qm    : int32 vector (Q0.31 quantized multiplier, non-negative)
%   shift : int32 vector (can be negative/positive)
%
% Notes:
%   - This matches the common TFLite meaning used in MultiplyByQuantizedMultiplier:
%       total_shift = 31 - shift
%       y = (x * qm + round) >> total_shift
%   - For typical conv/FC requant, M is in (0,1), shift usually <= 0,
%     but we support general M >= 0.

    % Make everything double for stable computation
    S_in  = double(S_in);
    S_w   = double(S_w);
    S_out = double(S_out);

    % Broadcast scalars to vector length
    n = max([numel(S_w), numel(S_out), 1]);
    if isscalar(S_w),   S_w   = repmat(S_w,   1, n); end
    if isscalar(S_out), S_out = repmat(S_out, 1, n); end

    % Real multipliers
    M = (S_in .* S_w) ./ S_out;

    qm    = zeros(1, n, 'int32');
    shift = zeros(1, n, 'int32');

    for i = 1:n
        r = M(i);

```

```
        if r == 0 || ~isfinite(r) || r < 0
            qm(i) = int32(0);
            shift(i) = int32(0);
            continue;
        end

        % log2 gives: r = significand * 2^exp, significand in [0.5, 1)
        [significand, exp] = log2(r);

        % Convert significand to Q31
        q = round(significand * 2^31);

        % Edge case: rounding to exactly 2^31
        if q == 2^31
            q = q / 2;
            exp = exp + 1;
        end

        % Clamp into [0, 2^31-1] (TFLite expects non-negative here)
        if q < 0, q = 0; end
        if q > (2^31 - 1), q = (2^31 - 1); end

        qm(i) = int32(q);
        shift(i) =  int32(exp);
    end
end
```

A.5.10 TFLite Parameter Exporter

Extracts TFLite weights, biases, scales, zero-points, multipliers, and shifts into a MAT file.

code/Golden-Module/pytorch/export_tflite_params_mat.py

```
#!/usr/bin/env python3
from __future__ import annotations

import argparse
from pathlib import Path

import numpy as np
import scipy.io as sio
import tensorflow as tf


def _to_cell(items: list[object]) -> np.ndarray:
    arr = np.empty((len(items), 1), dtype=object)
    for i, v in enumerate(items):
        arr[i, 0] = v
    return arr


```

```
def collect_param_tensor_indices(interpreter: tf.lite.Interpreter) -> list[int]:
    details = interpreter.get_tensor_details()
    all_indices = {d["index"] for d in details}
    model_inputs = {d["index"] for d in interpreter.get_input_details()}

    op_inputs: set[int] = set()
    op_outputs: set[int] = set()
    for op in interpreter._get_ops_details():
        op_inputs.update(i for i in op["inputs"] if i >= 0)
        op_outputs.update(i for i in op["outputs"] if i >= 0)

    candidates = (op_inputs - op_outputs - model_inputs) & all_indices
    return sorted(candidates)


def collect_graph_tensor_indices(interpreter: tf.lite.Interpreter) -> list[int]:
    details = interpreter.get_tensor_details()
    all_indices = {d["index"] for d in details}
    model_inputs = {d["index"] for d in interpreter.get_input_details()}
    model_outputs = {d["index"] for d in interpreter.get_output_details()}

    op_inputs: set[int] = set()
    op_outputs: set[int] = set()
    for op in interpreter._get_ops_details():
        op_inputs.update(i for i in op["inputs"] if i >= 0)
        op_outputs.update(i for i in op["outputs"] if i >= 0)

    graph_indices = (op_inputs | op_outputs | model_inputs | model_outputs) & all_indices
    return sorted(graph_indices)


def collect_activation_tensor_indices(interpreter: tf.lite.Interpreter) -> list[int]:
    details = interpreter.get_tensor_details()
    all_indices = {d["index"] for d in details}

    op_outputs: set[int] = set()
    for op in interpreter._get_ops_details():
        op_outputs.update(i for i in op["outputs"] if i >= 0)

    return sorted(op_outputs & all_indices)


def export_params_to_mat(model_path: Path, out_path: Path) -> None:
    interpreter = tf.lite.Interpreter(model_path=str(model_path))
    interpreter.allocate_tensors()

    tensor_details = {d["index"]: d for d in interpreter.get_tensor_details()}
    op_details = interpreter._get_ops_details()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    param_indices = collect_param_tensor_indices(interpreter)
    graph_indices = collect_graph_tensor_indices(interpreter)
    activation_indices = collect_activation_tensor_indices(interpreter)
```

```
73
74      # Parameter tensors (weights/bias/const tensors consumed by ops)
75      names: list[str] = []
76      tensor_indices: list[int] = []
77      dtypes: list[str] = []
78      shapes: list[np.ndarray] = []
79      shape_signatures: list[np.ndarray] = []
80      quantized_dims: list[int] = []
81      values: list[np.ndarray] = []
82      scales: list[np.ndarray] = []
83      zero_points: list[np.ndarray] = []
84
85      for idx in param_indices:
86          d = tensor_details[idx]
87          q = d["quantization_parameters"]
88          v = interpreter.get_tensor(idx)
89
90          names.append(d["name"])
91          tensor_indices.append(int(idx))
92          dtypes.append(str(v.dtype))
93          shapes.append(np.asarray(d["shape"], dtype=np.int64))
94          shape_signatures.append(np.asarray(d["shape_signature"], dtype=np.int64))
95          quantized_dims.append(int(q["quantized_dimension"]))
96          values.append(v)
97          scales.append(np.asarray(q["scales"], dtype=np.float32))
98          zero_points.append(np.asarray(q["zero_points"], dtype=np.int32))
99
100     # Activation tensors (op outputs)
101     act_names: list[str] = []
102     act_indices: list[int] = []
103     act_dtypes: list[str] = []
104     act_shapes: list[np.ndarray] = []
105     act_shape_signatures: list[np.ndarray] = []
106     act_quantized_dims: list[int] = []
107     act_scales: list[np.ndarray] = []
108     act_zero_points: list[np.ndarray] = []
109
110     for idx in activation_indices:
111         d = tensor_details[idx]
112         q = d["quantization_parameters"]
113
114         act_names.append(d["name"])
115         act_indices.append(int(idx))
116         act_dtypes.append(str(np.dtype(d["dtype"])))
117         act_shapes.append(np.asarray(d["shape"], dtype=np.int64))
118         act_shape_signatures.append(np.asarray(d["shape_signature"], dtype=np.int64))
119         act_quantized_dims.append(int(q["quantized_dimension"]))
120         act_scales.append(np.asarray(q["scales"], dtype=np.float32))
121         act_zero_points.append(np.asarray(q["zero_points"], dtype=np.int32))
122
123     # Model IO tensor specs
124     input_names: list[str] = []
125     input_indices: list[int] = []
126     input_dtypes: list[str] = []
```

```
    input_shapes: list[np.ndarray] = []
    input_shape_signatures: list[np.ndarray] = []
    input_quantized_dims: list[int] = []
    input_scales: list[np.ndarray] = []
    input_zero_points: list[np.ndarray] = []

    for d in input_details:
        q = d["quantization_parameters"]
        input_names.append(d["name"])
        input_indices.append(int(d["index"]))
        input_dtypes.append(str(np.dtype(d["dtype"])))
        input_shapes.append(np.asarray(d["shape"], dtype=np.int64))
        input_shape_signatures.append(np.asarray(d["shape_signature"], dtype=np.int64))
        input_quantized_dims.append(int(q["quantized_dimension"]))
        input_scales.append(np.asarray(q["scales"], dtype=np.float32))
        input_zero_points.append(np.asarray(q["zero_points"], dtype=np.int32))

    output_names: list[str] = []
    output_indices: list[int] = []
    output_dtypes: list[str] = []
    output_shapes: list[np.ndarray] = []
    output_shape_signatures: list[np.ndarray] = []
    output_quantized_dims: list[int] = []
    output_scales: list[np.ndarray] = []
    output_zero_points: list[np.ndarray] = []

    for d in output_details:
        q = d["quantization_parameters"]
        output_names.append(d["name"])
        output_indices.append(int(d["index"]))
        output_dtypes.append(str(np.dtype(d["dtype"])))
        output_shapes.append(np.asarray(d["shape"], dtype=np.int64))
        output_shape_signatures.append(np.asarray(d["shape_signature"], dtype=np.int64))
        output_quantized_dims.append(int(q["quantized_dimension"]))
        output_scales.append(np.asarray(q["scales"], dtype=np.float32))
        output_zero_points.append(np.asarray(q["zero_points"], dtype=np.int32))

    # Operator graph connectivity
    op_indices: list[int] = []
    op_names: list[str] = []
    op_inputs: list[np.ndarray] = []
    op_outputs: list[np.ndarray] = []

    for op in op_details:
        op_indices.append(int(op["index"]))
        op_names.append(op["op_name"])
        op_inputs.append(np.asarray(op["inputs"], dtype=np.int32))
        op_outputs.append(np.asarray(op["outputs"], dtype=np.int32))

    out_path.parent.mkdir(parents=True, exist_ok=True)
    sio.savemat(
        str(out_path),
        {
            "model_path": str(model_path),
```

```
            "num_graph_tensors": np.int32(len(graph_indices)),
            "graph_tensor_indices": np.asarray(graph_indices, dtype=np.int32).reshape(-1, 1),
            "num_params": np.int32(len(param_indices)),
            "names": _to_cell(names),
            "tensor_indices": np.asarray(tensor_indices, dtype=np.int32).reshape(-1, 1),
            "dtypes": _to_cell(dtypes),
            "shapes": _to_cell(shapes),
            "shape_signatures": _to_cell(shape_signatures),
            "quantized_dimensions": np.asarray(quantized_dims, dtype=np.int32).reshape(-1, 1),
            "values": _to_cell(values),
            "scales": _to_cell(scales),
            "zero_points": _to_cell(zero_points),
            "num_inputs": np.int32(len(input_details)),
            "input_names": _to_cell(input_names),
            "input_indices": np.asarray(input_indices, dtype=np.int32).reshape(-1, 1),
            "input_dtypes": _to_cell(input_dtypes),
            "input_shapes": _to_cell(input_shapes),
            "input_shape_signatures": _to_cell(input_shape_signatures),
            "input_quantized_dimensions": np.asarray(input_quantized_dims, dtype=np.int32).reshape(-1, 1)
        ,
            "input_scales": _to_cell(input_scales),
            "input_zero_points": _to_cell(input_zero_points),
            "num_outputs": np.int32(len(output_details)),
            "output_names": _to_cell(output_names),
            "output_indices": np.asarray(output_indices, dtype=np.int32).reshape(-1, 1),
            "output_dtypes": _to_cell(output_dtypes),
            "output_shapes": _to_cell(output_shapes),
            "output_shape_signatures": _to_cell(output_shape_signatures),
            "output_quantized_dimensions": np.asarray(output_quantized_dims, dtype=np.int32).reshape(-1,
        1),
            "output_scales": _to_cell(output_scales),
            "output_zero_points": _to_cell(output_zero_points),
            "num_activations": np.int32(len(activation_indices)),
            "activation_names": _to_cell(act_names),
            "activation_indices": np.asarray(act_indices, dtype=np.int32).reshape(-1, 1),
            "activation_dtypes": _to_cell(act_dtypes),
            "activation_shapes": _to_cell(act_shapes),
            "activation_shape_signatures": _to_cell(act_shape_signatures),
            "activation_quantized_dimensions": np.asarray(act_quantized_dims, dtype=np.int32).reshape(-1,
         1),
            "activation_scales": _to_cell(act_scales),
            "activation_zero_points": _to_cell(act_zero_points),
            "num_ops": np.int32(len(op_details)),
            "op_indices": np.asarray(op_indices, dtype=np.int32).reshape(-1, 1),
            "op_names": _to_cell(op_names),
            "op_inputs": _to_cell(op_inputs),
            "op_outputs": _to_cell(op_outputs),
        },
        do_compression=True,
    )

    print(f"[OK] Exported {len(param_indices)} parameter tensors")
    print(f"[OK] Exported {len(input_details)} input tensor specs")
    print(f"[OK] Exported {len(output_details)} output tensor specs")
```

```
    print(f"[OK] Exported {len(activation_indices)} activation tensor specs")
    print(f"[OK] Exported {len(op_details)} operator nodes")
    print(f"[OK] Saved: {out_path}")


def main() -> None:
    parser = argparse.ArgumentParser(
        description="Export int8 TFLite parameter tensors to a MATLAB .mat file."
    )
    parser.add_argument(
        "--model",
        type=Path,
        default=Path("models/v2.int8.tflite"),
        help="Path to .tflite model",
    )
    parser.add_argument(
        "--out",
        type=Path,
        default=Path("models/v2.int8.params.mat"),
        help="Output .mat path",
    )
    args = parser.parse_args()

    if not args.model.exists():
        raise FileNotFoundError(f"Model not found: {args.model}")

    export_params_to_mat(args.model.resolve(), args.out.resolve())


if __name__ == "__main__":
    main()
```

A.5.11 Digit CNN Retraining Script

Trains and exports the compact digit CNN model family used by the accelerator flow.

code/Golden-Module/pytorch/retrain_digits_cnn.py

```
#!/usr/bin/env python3
from __future__ import annotations

import argparse
import math
import subprocess
import sys
import textwrap
from pathlib import Path

from tensorflow.keras import callbacks
from tensorflow.keras.layers import Conv2D, Dense, Flatten, InputLayer, MaxPooling2D
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


def build_model(num_classes: int = 10) -> Sequential:
    model = Sequential(
        [
            InputLayer(input_shape=(64, 64, 1)),
            Conv2D(filters=4, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation="relu"),
            MaxPooling2D(pool_size=(2, 2)),
            Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation="relu"),
            MaxPooling2D(pool_size=(2, 2)),
            Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation="relu"),
            MaxPooling2D(pool_size=(2, 2)),
            Flatten(),
            Dense(units=num_classes, activation="softmax"),
        ]
    )
    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model


def main() -> None:
    parser = argparse.ArgumentParser(description="Retrain the 64x64 grayscale 0-9 sign-language CNN and export INT8 TFLite.")
    parser.add_argument("--dataset", type=Path, required=True, help="Merged dataset root with train/validation/test splits.")
    parser.add_argument("--models-dir", type=Path, default=Path("../models"), help="Model output directory (default: ../models).")
    parser.add_argument("--name", type=str, default="v3_merged", help="Base model name (default: v3_merged).")
    parser.add_argument("--epochs", type=int, default=60, help="Training epochs (default: 60).")
    parser.add_argument("--batch-size", type=int, default=32, help="Batch size (default: 32).")
    parser.add_argument("--rep-per-class", type=int, default=120, help="Representative samples per class for INT8 export.")
    args = parser.parse_args()

    dataset_root = args.dataset.resolve()
    train_folder = dataset_root / "train"
    validation_folder = dataset_root / "validation"
    test_folder = dataset_root / "test"
    if not train_folder.exists():
        raise FileNotFoundError(f"Missing train split: {train_folder}")
    if not validation_folder.exists():
        raise FileNotFoundError(f"Missing validation split: {validation_folder}")
    if not test_folder.exists():
        raise FileNotFoundError(f"Missing test split: {test_folder}")

    train_gen_cfg = ImageDataGenerator(
        rescale=1.0 / 255.0,
        rotation_range=12,
        width_shift_range=0.10,
        height_shift_range=0.10,
        shear_range=0.10,
        zoom_range=0.10,
```

```
        horizontal_flip=False,
        vertical_flip=False,
        fill_mode="nearest",
    )
    eval_gen_cfg = ImageDataGenerator(rescale=1.0 / 255.0)

    train_gen = train_gen_cfg.flow_from_directory(
        train_folder,
        target_size=(64, 64),
        color_mode="grayscale",
        batch_size=args.batch_size,
        class_mode="categorical",
        shuffle=True,
    )
    validation_gen = eval_gen_cfg.flow_from_directory(
        validation_folder,
        target_size=(64, 64),
        color_mode="grayscale",
        batch_size=args.batch_size,
        class_mode="categorical",
        shuffle=False,
    )
    test_gen = eval_gen_cfg.flow_from_directory(
        test_folder,
        target_size=(64, 64),
        color_mode="grayscale",
        batch_size=args.batch_size,
        class_mode="categorical",
        shuffle=False,
    )

    expected_mapping = {str(i): i for i in range(10)}
    if train_gen.class_indices != expected_mapping:
        raise ValueError(f"Unexpected class mapping: {train_gen.class_indices}")

    models_dir = args.models_dir.resolve()
    models_dir.mkdir(parents=True, exist_ok=True)
    h5_path = models_dir / f"{args.name}.h5"
    tflite_path = models_dir / f"{args.name}.int8.tflite"

    model = build_model(num_classes=10)
    checkpoint = callbacks.ModelCheckpoint(str(h5_path), monitor="val_accuracy", save_best_only=True,
     verbose=1)
    early_stop = callbacks.EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)

    history = model.fit(
        train_gen,
        epochs=args.epochs,
        steps_per_epoch=math.ceil(train_gen.samples / train_gen.batch_size),
        validation_data=validation_gen,
        validation_steps=math.ceil(validation_gen.samples / validation_gen.batch_size),
        callbacks=[checkpoint, early_stop],
        verbose=2,
    )
```

```
    test_loss, test_acc = model.evaluate(test_gen, verbose=0)
    print(f"[TEST] loss={test_loss:.4f} acc={test_acc:.4f}")

    export_script = textwrap.dedent(
        """
        import sys
        from pathlib import Path

        import numpy as np
        import tensorflow as tf

        model_path = Path(sys.argv[1])
        rep_dir = Path(sys.argv[2])
        out_path = Path(sys.argv[3])
        per_class_samples = int(sys.argv[4])

        model = tf.keras.models.load_model(str(model_path), compile=False)
        model(np.zeros((1, 64, 64, 1), dtype=np.float32))

        def representative_dataset_gen():
            exts = {'.jpg', '.jpeg', '.png', '.bmp', '.webp'}
            class_dirs = sorted([d for d in rep_dir.iterdir() if d.is_dir()], key=lambda p: int(p.name))
            picked = []
            for d in class_dirs:
                files = [p for p in sorted(d.rglob('*')) if p.suffix.lower() in exts]
                picked.extend(files[: min(per_class_samples, len(files))])
            if not picked:
                raise FileNotFoundError(f'No representative images found in {rep_dir}')
            for p in picked:
                img = tf.keras.utils.load_img(p, color_mode='grayscale', target_size=(64, 64))
                arr = tf.keras.utils.img_to_array(img).astype(np.float32) / 255.0
                arr = np.expand_dims(arr, axis=0)
                yield [arr]

        serving_fn = tf.function(lambda x: model(x))
        concrete = serving_fn.get_concrete_function(tf.TensorSpec([1, 64, 64, 1], tf.float32))
        converter = tf.lite.TFLiteConverter.from_concrete_functions([concrete], model)
        converter.optimizations = [tf.lite.Optimize.DEFAULT]
        converter.representative_dataset = representative_dataset_gen
        converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
        converter.inference_input_type = tf.int8
        converter.inference_output_type = tf.int8
        out_path.write_bytes(converter.convert())

        print(f'[OK] wrote {out_path}')
        """
    )
    cmd = [sys.executable, "-c", export_script, str(h5_path), str(train_folder), str(tflite_path), str(
     args.rep_per_class)]
    subprocess.run(cmd, check=True)

    summary = {
        "name": args.name,
```

```
        "dataset": str(dataset_root),
        "train_samples": int(train_gen.samples),
        "validation_samples": int(validation_gen.samples),
        "test_samples": int(test_gen.samples),
        "best_val_accuracy": max(history.history.get("val_accuracy", [0.0])),
        "test_accuracy": float(test_acc),
        "h5_path": str(h5_path),
        "tflite_path": str(tflite_path),
    }
    print(summary)


if __name__ == "__main__":
    main()
```

A.6 Quartus And Platform Designer Setup

The Quartus and Platform Designer GUI was used only as an editing and inspection front end. The reproducible project state is captured by the saved source configuration files listed below: the `.qsys` Platform Designer system, the custom component `*_hw.tcl`, the Quartus project-generation Tcl script, and the timing constraint `.sdc`. Generated HDL, QIP files, reports, programming files, and build databases are intentionally excluded from the submission archive because they can be regenerated from these sources.

A.6.1 Platform Designer Component Descriptor

Registers the custom CNN MMIO interface as a Platform Designer component.

code/platform_designer/cnn_mmio_interface_hw.tcl

```
# TCL File adapted for Component Editor / Platform Designer 21.1
#
# cnn_mmio_interface "cnn_mmio_interface" v1.0

package require -exact qsys 16.1

# ------------------------------------------------------------------------
# module cnn_mmio_interface
# ------------------------------------------------------------------------
set_module_property DESCRIPTION "Memory-mapped wrapper for the CNN_ACC gesture accelerator"
set_module_property NAME cnn_mmio_interface
set_module_property VERSION 1.0
set_module_property GROUP "Project"
set_module_property INTERNAL false
set_module_property HIDE_FROM_SOPC false
set_module_property HIDE_FROM_QUARTUS false
set_module_property OPAQUE_ADDRESS_MAP true
```

```
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME cnn_mmio_interface
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false

# ------------------------------------------------------------------------
# file sets
# ------------------------------------------------------------------------
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL cnn_mmio_interface
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file ../input/RTL/interface/cnn_mmio_interface.v VERILOG PATH ../input/RTL/interface/
    cnn_mmio_interface.v TOP_LEVEL_FILE
add_fileset_file ../input/RTL/system_top.v VERILOG PATH ../input/RTL/system_top.v
add_fileset_file ../input/RTL/fsm/top_fsm.v VERILOG PATH ../input/RTL/fsm/top_fsm.v
add_fileset_file ../input/RTL/fsm/layer_runner_fsm.v VERILOG PATH ../input/RTL/fsm/layer_runner_fsm.v
add_fileset_file ../input/RTL/fsm/conv_data_adapter.v VERILOG PATH ../input/RTL/fsm/conv_data_adapter.v
add_fileset_file ../input/RTL/fsm/wt_prepad_inserter.v VERILOG PATH ../input/RTL/fsm/wt_prepad_inserter.v
add_fileset_file ../input/RTL/conv_core/conv_quant_pool.v VERILOG PATH ../input/RTL/conv_core/
    conv_quant_pool.v
add_fileset_file ../input/RTL/conv_core/conv_top.v VERILOG PATH ../input/RTL/conv_core/conv_top.v
add_fileset_file ../input/RTL/conv_core/conv_engine_ctrl.v VERILOG PATH ../input/RTL/conv_core/
    conv_engine_ctrl.v
add_fileset_file ../input/RTL/conv_core/weight_buffer.v VERILOG PATH ../input/RTL/conv_core/
    weight_buffer.v
add_fileset_file ../input/RTL/conv_core/Line_Buffer.v VERILOG PATH ../input/RTL/conv_core/Line_Buffer.v
add_fileset_file ../input/RTL/conv_core/input_row_aligner.v VERILOG PATH ../input/RTL/conv_core/
    input_row_aligner.v
add_fileset_file ../input/RTL/conv_core/Conv_Buffer.v VERILOG PATH ../input/RTL/conv_core/Conv_Buffer.v
add_fileset_file ../input/RTL/conv_core/sa_skew_feeder.v VERILOG PATH ../input/RTL/conv_core/
    sa_skew_feeder.v
add_fileset_file ../input/RTL/conv_core/systolic_array_top.v VERILOG PATH ../input/RTL/conv_core/
    systolic_array_top.v
add_fileset_file ../input/RTL/quant_pool/integration/quant_param_loader.v VERILOG PATH ../input/RTL/
    quant_pool/integration/quant_param_loader.v
add_fileset_file ../input/RTL/quant_pool/integration/conv_quant_adapter.v VERILOG PATH ../input/RTL/
    quant_pool/integration/conv_quant_adapter.v
add_fileset_file ../input/RTL/quant_pool/integration/quant_pool_adapter.v VERILOG PATH ../input/RTL/
    quant_pool/integration/quant_pool_adapter.v
add_fileset_file ../input/RTL/quant_pool/quant/Quantization_PE.v VERILOG PATH ../input/RTL/quant_pool/
    quant/Quantization_PE.v
add_fileset_file ../input/RTL/quant_pool/quant/Quantization_Top.v VERILOG PATH ../input/RTL/quant_pool/
    quant/Quantization_Top.v
add_fileset_file ../input/RTL/quant_pool/pool/pool_core.v VERILOG PATH ../input/RTL/quant_pool/pool/
    pool_core.v
add_fileset_file ../input/RTL/quant_pool/pool/pool_stream_top.v VERILOG PATH ../input/RTL/quant_pool/pool
    /pool_stream_top.v
add_fileset_file ../input/RTL/fc/mac.v VERILOG PATH ../input/RTL/fc/mac.v
add_fileset_file ../input/RTL/fc/fc_bias_loader.v VERILOG PATH ../input/RTL/fc/fc_bias_loader.v
add_fileset_file ../input/RTL/fc/fc_data_adapter.v VERILOG PATH ../input/RTL/fc/fc_data_adapter.v
```

```
add_fileset_file ../input/RTL/fc/FC.v VERILOG PATH ../input/RTL/fc/FC.v
add_fileset_file ../input/RTL/SRAM/Addr_Gen.v VERILOG PATH ../input/RTL/SRAM/Addr_Gen.v
add_fileset_file ../input/RTL/SRAM/sram_A_controller.v VERILOG PATH ../input/RTL/SRAM/sram_A_controller.v
add_fileset_file ../input/RTL/SRAM/sram_B_controller.v VERILOG PATH ../input/RTL/SRAM/sram_B_controller.v
add_fileset_file ../input/RTL/SRAM/sram_A_wrapper.v VERILOG PATH ../input/RTL/SRAM/sram_A_wrapper.v
add_fileset_file ../input/RTL/SRAM/sram_B_wrapper.v VERILOG PATH ../input/RTL/SRAM/sram_B_wrapper.v
add_fileset_file ../input/RTL/SRAM/sram_FCW_wrapper.v VERILOG PATH ../input/RTL/SRAM/sram_FCW_wrapper.v
add_fileset_file ../input/RTL/SRAM/fcw_preload_packer.v VERILOG PATH ../input/RTL/SRAM/
    fcw_preload_packer.v
add_fileset_file ../input/RTL/SRAM/top_sram_A.v VERILOG PATH ../input/RTL/SRAM/top_sram_A.v
add_fileset_file ../input/RTL/SRAM/top_sram_B.v VERILOG PATH ../input/RTL/SRAM/top_sram_B.v

# ------------------------------------------------------------------------
# module assignments
# ------------------------------------------------------------------------
set_module_assignment embeddedsw.dts.group cnn_mmio_interface
set_module_assignment embeddedsw.dts.name cnn_mmio_interface
set_module_assignment embeddedsw.dts.vendor csee4840

# ------------------------------------------------------------------------
# connection point clock
# ------------------------------------------------------------------------
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""
add_interface_port clock clk clk Input 1

# ------------------------------------------------------------------------
# connection point reset
# ------------------------------------------------------------------------
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""
add_interface_port reset reset reset Input 1

# ------------------------------------------------------------------------
# connection point avalon_slave_0
# ------------------------------------------------------------------------
add_interface avalon_slave_0 avalon end
set_interface_property avalon_slave_0 addressUnits WORDS
set_interface_property avalon_slave_0 associatedClock clock
set_interface_property avalon_slave_0 associatedReset reset
set_interface_property avalon_slave_0 bitsPerSymbol 8
set_interface_property avalon_slave_0 burstOnBurstBoundariesOnly false
set_interface_property avalon_slave_0 burstcountUnits WORDS
```

```
set_interface_property avalon_slave_0 explicitAddressSpan 0
set_interface_property avalon_slave_0 holdTime 0
set_interface_property avalon_slave_0 linewrapBursts false
set_interface_property avalon_slave_0 maximumPendingReadTransactions 0
set_interface_property avalon_slave_0 maximumPendingWriteTransactions 0
set_interface_property avalon_slave_0 readLatency 2
set_interface_property avalon_slave_0 readWaitTime 0
set_interface_property avalon_slave_0 setupTime 0
set_interface_property avalon_slave_0 timingUnits Cycles
set_interface_property avalon_slave_0 writeWaitTime 0
set_interface_property avalon_slave_0 ENABLED true
set_interface_property avalon_slave_0 EXPORT_OF ""
set_interface_property avalon_slave_0 PORT_NAME_MAP ""
set_interface_property avalon_slave_0 CMSIS_SVD_VARIABLES ""
set_interface_property avalon_slave_0 SVD_ADDRESS_GROUP ""
add_interface_port avalon_slave_0 writedata writedata Input 32
add_interface_port avalon_slave_0 byteenable byteenable Input 4
add_interface_port avalon_slave_0 write write Input 1
add_interface_port avalon_slave_0 read read Input 1
add_interface_port avalon_slave_0 chipselect chipselect Input 1
add_interface_port avalon_slave_0 address address Input 13
add_interface_port avalon_slave_0 readdata readdata Output 32
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isFlash 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isMemoryDevice 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isNonVolatileStorage 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isPrintableDevice 0
```

A.6.2 Quartus Build Wrapper

Discovers Quartus tools, regenerates Qsys HDL, patches generated debug ports, and runs the compile flow.

code/de1_soc/build_soc_system.py

```
#!/usr/bin/env python3
"""Cross-platform build wrapper for the DE1-SoC Quartus project.

Usage:
    python de1_soc/build_soc_system.py
"""

from __future__ import annotations

import argparse
import os
import platform
import shutil
import subprocess
import sys
from pathlib import Path
```

```
17
18
19  PROJECT_DIR = Path(__file__).resolve().parent
20  PROJECT_NAME = "soc_system"
21  QSYS_FILE = PROJECT_DIR / f"{PROJECT_NAME}.qsys"
22  SOF_FILE = PROJECT_DIR / "output_files" / f"{PROJECT_NAME}.sof"
23  RBF_FILE = PROJECT_DIR / "output_files" / f"{PROJECT_NAME}.rbf"
24  GENERATED_SOC_SYSTEM_V = PROJECT_DIR / "soc_system" / "synthesis" / "soc_system.v"
25
26
27  def candidate_roots() -> list[Path]:
28      env_root = os.environ.get("QUARTUS_ROOTDIR")
29      roots: list[Path] = []
30      if env_root:
31          roots.append(Path(env_root))
32
33      system = platform.system()
34      if system == "Windows":
35          roots.extend(
36              [
37                  Path(r"C:\intelFPGA_lite"),
38                  Path(r"C:\intelFPGA"),
39                  Path(r"C:\Program Files\intelFPGA_lite"),
40                  Path(r"C:\Program Files\intelFPGA"),
41              ]
42          )
43      elif system == "Darwin":
44          roots.extend(
45              [
46                  Path("/Applications/intelFPGA_lite"),
47                  Path("/Applications/intelFPGA"),
48                  Path.home() / "intelFPGA_lite",
49                  Path.home() / "intelFPGA",
50              ]
51          )
52      else:
53          roots.extend(
54              [
55                  Path("/opt/intelFPGA_lite"),
56                  Path("/opt/intelFPGA"),
57                  Path.home() / "intelFPGA_lite",
58                  Path.home() / "intelFPGA",
59              ]
60          )
61      return roots
62
63
64  def tool_names(base: str) -> list[str]:
65      system = platform.system()
66      if system == "Windows":
67          return [f"{base}.exe", base]
68      return [base]
69
70
```

```
def env_var_name(base: str) -> str:
    return base.replace("-", "_").upper()


def find_tool(base: str, verbose: bool = False) -> tuple[str | None, list[str]]:
    searched: list[str] = []
    override_var = env_var_name(base)
    override = os.environ.get(override_var)
    if override:
        searched.append(f"{override_var}={override}")
        if Path(override).exists():
            return override, searched

    for name in tool_names(base):
        searched.append(f"PATH:{name}")
        found = shutil.which(name)
        if found:
            return found, searched

    for root in candidate_roots():
        searched.append(f"ROOT:{root}")
        if not root.exists():
            continue

        if root.name == "quartus":
            search_bins = [
                root / "bin64",
                root / "bin",
                root / "sopc_builder" / "bin",
            ]
        else:
            search_bins = []
            for version_dir in sorted(root.glob("*")):
                if not version_dir.is_dir():
                    continue
                quartus_dir = version_dir / "quartus"
                if quartus_dir.exists():
                    search_bins.extend(
                        [
                            quartus_dir / "bin64",
                            quartus_dir / "bin",
                            quartus_dir / "sopc_builder" / "bin",
                        ]
                    )

        for bin_dir in search_bins:
            for name in tool_names(base):
                candidate = bin_dir / name
                searched.append(str(candidate))
                if candidate.exists():
                    return str(candidate), searched

    if verbose:
        for item in searched:
```

```
            print(f"  searched {item}")

    return None, searched


def build_env_for_subprocesses(quartus_sh: str | None) -> dict[str, str]:
    env = os.environ.copy()

    if platform.system() != "Windows" or not quartus_sh:
        return env

    quartus_path = Path(quartus_sh).resolve()
    acds_root = quartus_path.parent.parent.parent
    nios_gnu_bin = acds_root / "nios2eds" / "bin" / "gnu" / "H-x86_64-mingw32" / "bin"

    if nios_gnu_bin.exists():
        env["PATH"] = str(nios_gnu_bin) + os.pathsep + env.get("PATH", "")

    return env


def run(cmd: list[str], cwd: Path, env: dict[str, str] | None = None) -> None:
    print("+", " ".join(f'"{part}"' if " " in part else part for part in cmd))
    subprocess.run(cmd, cwd=str(cwd), env=env, check=True)


def run_capture(cmd: list[str], cwd: Path, env: dict[str, str] | None = None) -> subprocess.
    CompletedProcess[str]:
    print("+", " ".join(f'"{part}"' if " " in part else part for part in cmd))
    return subprocess.run(
        cmd,
        cwd=str(cwd),
        env=env,
        check=False,
        text=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
    )


def is_recoverable_windows_qsys_failure(output: str) -> bool:
    markers = [
        "generate_hps_sdram.tcl",
        "Nios II Command Shell.bat",
        "child process exited abnormally",
        "Error: border: Error during execution of script generate_hps_sdram.tcl: seq:",
    ]
    return platform.system() == "Windows" and all(marker in output for marker in markers)


def generated_hdl_looks_usable() -> bool:
    required = [
        PROJECT_DIR / "soc_system" / "synthesis" / "soc_system.v",
        PROJECT_DIR / "soc_system" / "synthesis" / "submodules" / "hps_sdram.v",
```

```
        PROJECT_DIR / "soc_system" / "synthesis" / "submodules" / "hps_sdram_p0.sv",
    ]
    return all(path.exists() for path in required)


def patch_generated_soc_system_debug_ports() -> None:
    if not GENERATED_SOC_SYSTEM_V.exists():
        raise FileNotFoundError(f"generated top not found: {GENERATED_SOC_SYSTEM_V}")

    text = GENERATED_SOC_SYSTEM_V.read_text(encoding="utf-8")

    if "cnn_debug_predict_class" in text:
        return

    module_old = "module soc_system (\n\t\tinput  wire        clk_clk,"
    module_new = (
        "module soc_system (\n"
        "\t\tinput  wire        clk_clk,\n"
        "\t\toutput wire        cnn_debug_model_loaded,\n"
        "\t\toutput wire        cnn_debug_predict_done,\n"
        "\t\toutput wire [3:0]  cnn_debug_predict_class,\n"
        "\t\toutput wire [15:0] cnn_debug_interface_error,"
    )

    inst_old = (
        "\t\t.address    (mm_interconnect_0_cnn_mmio_interface_0_avalon_slave_0_address),    //
          .address\n"
        "\t\t.readdata   (mm_interconnect_0_cnn_mmio_interface_0_avalon_slave_0_readdata)    //
          .readdata\n"
        "\t);"
    )
    inst_new = (
        "\t\t.address    (mm_interconnect_0_cnn_mmio_interface_0_avalon_slave_0_address),    //
          .address\n"
        "\t\t.readdata   (mm_interconnect_0_cnn_mmio_interface_0_avalon_slave_0_readdata),   //
          .readdata\n"
        "\t\t.debug_model_loaded   (cnn_debug_model_loaded),                                  //
           .debug_model_loaded\n"
        "\t\t.debug_predict_done   (cnn_debug_predict_done),                                  //
           .debug_predict_done\n"
        "\t\t.debug_predict_class  (cnn_debug_predict_class),                                 //
           .debug_predict_class\n"
        "\t\t.debug_interface_error(cnn_debug_interface_error)                               //
           .debug_interface_error\n"
        "\t);"
    )

    if module_old not in text:
        raise RuntimeError("could not find soc_system module header to patch debug ports")
    if inst_old not in text:
        raise RuntimeError("could not find cnn_mmio_interface instantiation to patch debug ports")

    text = text.replace(module_old, module_new, 1)
    text = text.replace(inst_old, inst_new, 1)
```

```
    GENERATED_SOC_SYSTEM_V.write_text(text, encoding="utf-8")


def check_windows_qsys_prereqs(quartus_sh: str | None) -> tuple[bool, list[str]]:
    if platform.system() != "Windows" or not quartus_sh:
        return True, []

    messages: list[str] = []
    quartus_path = Path(quartus_sh).resolve()
    quartus_root = quartus_path.parent.parent
    nios2_shell = quartus_root.parent / "nios2eds" / "Nios II Command Shell.bat"
    mingw_bin = quartus_root.parent / "nios2eds" / "bin" / "gnu" / "H-x86_64-mingw32" / "bin"

    ok = True
    if not nios2_shell.exists():
        messages.append(f"missing Nios II Command Shell: {nios2_shell}")
        ok = False
    if not mingw_bin.exists():
        messages.append(f"missing Nios II GCC toolchain directory: {mingw_bin}")
        ok = False

    try:
        subprocess.run(
            ["wsl.exe", "bash", "-lc", "command -v dos2unix"],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL,
            check=True,
        )
    except subprocess.CalledProcessError:
        messages.append("WSL is missing dos2unix (install with: wsl -u root bash -lc \"apt-get install -y
      dos2unix\")")
        ok = False
    except FileNotFoundError:
        messages.append("WSL is not available on this machine")
        ok = False

    return ok, messages


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Build the DE1-SoC Quartus project.")
    parser.add_argument(
        "--check",
        action="store_true",
        help="Only check tool discovery and print what would be used.",
    )
    parser.add_argument(
        "--verbose",
        action="store_true",
        help="Print searched locations while resolving Quartus tools.",
    )
    parser.add_argument(
        "--skip-qsys",
        action="store_true",
```

```
        help="Skip qsys-generate and compile using the committed generated HDL.",
    )
    return parser.parse_args()


def main() -> int:
    args = parse_args()
    qsys_generate, qsys_search = find_tool("qsys-generate", verbose=args.verbose)
    quartus_sh, quartus_sh_search = find_tool("quartus_sh", verbose=args.verbose)
    quartus_cpf, quartus_cpf_search = find_tool("quartus_cpf", verbose=args.verbose)

    missing = []
    if not qsys_generate and not args.skip_qsys:
        missing.append("qsys-generate")
    if not quartus_sh:
        missing.append("quartus_sh")

    if missing:
        print("Required Quartus tools were not found:", ", ".join(missing), file=sys.stderr)
        print(
            "Tried PATH, tool-specific env vars, QUARTUS_ROOTDIR, and common install folders.",
            file=sys.stderr,
        )
        print("Optional overrides:", file=sys.stderr)
        print("  QSYS_GENERATE=/path/to/qsys-generate", file=sys.stderr)
        print("  QUARTUS_SH=/path/to/quartus_sh", file=sys.stderr)
        print("  QUARTUS_CPF=/path/to/quartus_cpf", file=sys.stderr)
        return 1

    prereq_ok, prereq_messages = check_windows_qsys_prereqs(quartus_sh)
    build_env = build_env_for_subprocesses(quartus_sh)
    if args.check:
        print("Quartus tool check passed.")
        if args.skip_qsys:
            print("  qsys-generate: skipped by request")
        else:
            print(f"  qsys-generate: {qsys_generate}")
        print(f"  quartus_sh:    {quartus_sh}")
        if quartus_cpf:
            print(f"  quartus_cpf:   {quartus_cpf}")
        else:
            print("  quartus_cpf:   not found (RBF conversion will be skipped)")
        if platform.system() == "Windows":
            if prereq_ok:
                print("  windows qsys prereqs: OK")
                quartus_path = Path(quartus_sh).resolve()
                acds_root = quartus_path.parent.parent.parent
                mingw_bin = acds_root / "nios2eds" / "bin" / "gnu" / "H-x86_64-mingw32" / "bin"
                print(f"  injected Nios GCC path: {mingw_bin}")
            else:
                print("  windows qsys prereqs: MISSING")
                for message in prereq_messages:
                    print(f"    - {message}")
                return 1
```

```
331         return 0
332
333     if not prereq_ok:
334         print("Windows Qsys prerequisites are incomplete:", file=sys.stderr)
335         for message in prereq_messages:
336             print(f"  - {message}", file=sys.stderr)
337         return 1
338
339     try:
340         if args.skip_qsys:
341             print("[1/3] Skipping Qsys generation and using committed generated HDL...")
342         else:
343             print("[1/3] Generating Qsys HDL...")
344             qsys_result = run_capture(
345                 [qsys_generate, str(QSYS_FILE), "--synthesis=VERILOG"],
346                 cwd=PROJECT_DIR,
347                 env=build_env,
348             )
349             sys.stdout.write(qsys_result.stdout)
350             if qsys_result.returncode != 0:
351                 if is_recoverable_windows_qsys_failure(qsys_result.stdout) and generated_hdl_looks_usable
     ():
352                     print()
353                     print(
354                         "Warning: qsys-generate hit the known Windows/WSL HPS SDRAM "
355                         "sequencer-build failure, but the HDL outputs were still emitted."
356                     )
357                     print("Continuing to Quartus compile with the freshly generated HDL.")
358                 else:
359                     print(f"Build failed with exit code {qsys_result.returncode}.", file=sys.stderr)
360                     return qsys_result.returncode
361
362         patch_generated_soc_system_debug_ports()
363
364         print("[2/3] Compiling Quartus project...")
365         run([quartus_sh, "--flow", "compile", PROJECT_NAME], cwd=PROJECT_DIR, env=build_env)
366
367         if not SOF_FILE.exists():
368             print(f"Expected SOF not found: {SOF_FILE}", file=sys.stderr)
369             return 1
370
371         if quartus_cpf:
372             print("[3/3] Converting SOF to RBF...")
373             run([quartus_cpf, "-c", str(SOF_FILE), str(RBF_FILE)], cwd=PROJECT_DIR, env=build_env)
374         else:
375             print("[3/3] quartus_cpf not found, skipping RBF conversion.")
376     except subprocess.CalledProcessError as exc:
377         print(f"Build failed with exit code {exc.returncode}.", file=sys.stderr)
378         return exc.returncode
379
380     print()
381     print(f"Build complete.")
382     print(f"SOF: {SOF_FILE}")
383     if RBF_FILE.exists():
```

```
        print(f"RBF: {RBF_FILE}")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

A.6.3 Quartus Project Script

Creates the DE1-SoC Quartus project and assigns board-level files, SDC constraints, and QIP references.

code/de1_soc/soc_system_project.tcl

```
# Generate Quartus project files for the DE1-SoC board.
#
# Adapted for CNN_ACC:
# - expects a generated Platform Designer system under soc_system/
# - expects a board-level top named soc_system_top in soc_system_top.sv
# - pulls in soc_system/synthesis/soc_system.qip
#
# Invoke as:
#   quartus_sh -t de1_soc/soc_system_project.tcl

set script_dir [file dirname [file normalize [info script]]]
set project_name "soc_system"
set project_path [file join $script_dir $project_name]

set systemVerilogSource [file join $script_dir "${project_name}_top.sv"]
set qip [file join $script_dir $project_name "synthesis" "${project_name}.qip"]

project_new $project_path -overwrite

# Clean out any stale assignments from previous iterations of this project.
foreach stale_file {
    "input/SRAM_macro/sram_A/sram_A.v"
    "input/SRAM_macro/sram_B/sram_B.v"
    "/homes/user/stud/fall25/lw3227/CNN_ACC/input/SRAM_macro/sram_A/sram_A.v"
    "/homes/user/stud/fall25/lw3227/CNN_ACC/input/SRAM_macro/sram_B/sram_B.v"
} {
    catch { remove_global_assignment -name VERILOG_FILE $stale_file }
}

foreach {name value} {
    FAMILY "Cyclone V"
    DEVICE 5CSEMA5F31C6
    PROJECT_OUTPUT_DIRECTORY output_files
    CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"
    NUM_PARALLEL_PROCESSORS 4
} {
```

```
    set_global_assignment -name $name $value
}

set_global_assignment -name TOP_LEVEL_ENTITY "${project_name}_top"
set_global_assignment -name SYSTEMVERILOG_FILE $systemVerilogSource
set_global_assignment -name QIP_FILE $qip
# Match the original Quartus project: use synthesizable behavioral SRAM
# models instead of the foundry macro models, which contain sim-only code.
set_global_assignment -name VERILOG_FILE [file join [file dirname $script_dir] "input" "TB" "sram_behav.v
    "]

# Minimal board-level assignment set for the generated HPS/MMIO system.
# Follow the lab3-hw pattern: export the HPS DDR3 interface using the board-
# level HPS_DDR3_* port names and let the generated HPS SDRAM assignment Tcl
# handle the dedicated DDR3 pin standards/termination.
set_location_assignment PIN_AF14 -to CLOCK_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50

set sdcFilename [file join $script_dir "${project_name}.sdc"]
set_global_assignment -name SDC_FILE $sdcFilename

set sdcf [open $sdcFilename "w"]
puts $sdcf {
    create_clock -name clock_50 -period 20ns [get_ports CLOCK_50]
    derive_pll_clocks -create_base_clocks
    derive_clock_uncertainty
}
close $sdcf

project_close
```

A.6.4 Platform Designer System

Stores the HPS, clock, reset, interconnect, and CNN MMIO component connections used by qsys-generate.

code/de1_soc/soc_system.qsys

```
<?xml version="1.0" encoding="UTF-8"?>
<system name="$${FILENAME}">
 <component
   name="$${FILENAME}"
   displayName="$${FILENAME}"
   version="1.0"
   description=""
   tags=""
   categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
   element clk_0
```

```
   {
      datum _sortIndex
      {
         value = "0";
         type = "int";
      }
   }
   element cnn_mmio_interface_0
   {
      datum _sortIndex
      {
         value = "1";
         type = "int";
      }
   }
   element hps_0
   {
      datum _sortIndex
      {
         value = "2";
         type = "int";
      }
   }
}
]]></parameter>
 <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="deviceFamily" value="Cyclone V" />
 <parameter name="deviceSpeedGrade" value="6" />
 <parameter name="fabricMode" value="QSYS" />
 <parameter name="generateLegacySim" value="false" />
 <parameter name="generationId" value="0" />
 <parameter name="globalResetBus" value="false" />
 <parameter name="hdlLanguage" value="VERILOG" />
 <parameter name="hideFromIPCatalog" value="false" />
 <parameter name="lockedInterfaceDefinition" value="" />
 <parameter name="maxAdditionalLatency" value="1" />
 <parameter name="projectName" value="soc_system.qpf" />
 <parameter name="sopcBorderPoints" value="false" />
 <parameter name="systemHash" value="0" />
 <parameter name="testBenchDutName" value="" />
 <parameter name="timeStamp" value="0" />
 <parameter name="useTestBenchNamingPattern" value="false" />
 <instanceScript></instanceScript>
 <interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
 <interface name="hps" internal="hps_0.hps_io" type="conduit" dir="end" />
 <interface name="memory" internal="hps_0.memory" type="conduit" dir="end" />
 <interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
 <module name="clk_0" kind="clock_source" version="21.1" enabled="1">
  <parameter name="clockFrequency" value="50000000" />
  <parameter name="clockFrequencyKnown" value="true" />
  <parameter name="inputClockFrequency" value="0" />
  <parameter name="resetSynchronousEdges" value="NONE" />
 </module>
```

```
<module
  name="cnn_mmio_interface_0"
  kind="cnn_mmio_interface"
  version="1.0"
  enabled="1" />
<module name="hps_0" kind="altera_hps" version="21.1" enabled="1">
 <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
 <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
 <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
 <parameter name="AC_PACKAGE_DESKEW" value="false" />
 <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
 <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
 <parameter name="ADDR_ORDER" value="0" />
 <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
 <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
 <parameter name="ADVANCED_CK_PHASES" value="false" />
 <parameter name="ADVERTIZE_SEQUENCER_SW_BUILD_FILES" value="false" />
 <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
 <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
 <parameter name="AP_MODE" value="false" />
 <parameter name="AP_MODE_EN" value="0" />
 <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
 <parameter name="AUTO_PD_CYCLES" value="0" />
 <parameter name="AUTO_POWERDN_EN" value="false" />
 <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
 <parameter name="AVL_MAX_SIZE" value="4" />
 <parameter name="BONDING_OUT_ENABLED" value="false" />
 <parameter name="BOOTFROMFPGA_Enable" value="false" />
 <parameter name="BSEL" value="1" />
 <parameter name="BSEL_EN" value="false" />
 <parameter name="BYTE_ENABLE" value="true" />
 <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
 <parameter name="CALIBRATION_MODE" value="Skip" />
 <parameter name="CALIB_REG_WIDTH" value="8" />
 <parameter name="CAN0_Mode" value="N/A" />
 <parameter name="CAN0_PinMuxing" value="Unused" />
 <parameter name="CAN1_Mode" value="N/A" />
 <parameter name="CAN1_PinMuxing" value="Unused" />
 <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
 <parameter name="CFG_REORDER_DATA" value="true" />
 <parameter name="CFG_TCCD_NS" value="2.5" />
 <parameter name="COMMAND_PHASE" value="0.0" />
 <parameter name="CONTROLLER_LATENCY" value="5" />
 <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
 <parameter name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional
    ,Bidirectional</parameter>
 <parameter name="CSEL" value="0" />
 <parameter name="CSEL_EN" value="false" />
 <parameter name="CTI_Enable" value="false" />
 <parameter name="CTL_AUTOPCH_EN" value="false" />
 <parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
 <parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
 <parameter name="CTL_CSR_ENABLED" value="false" />
 <parameter name="CTL_CSR_READ_ONLY" value="1" />
```

```
    <parameter name="CTL_DEEP_POWERDN_EN" value="false" />
    <parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
    <parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
    <parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
    <parameter name="CTL_ECC_ENABLED" value="false" />
    <parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />
    <parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
    <parameter name="CTL_HRB_ENABLED" value="false" />
    <parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
    <parameter name="CTL_SELF_REFRESH_EN" value="false" />
    <parameter name="CTL_USR_REFRESH_EN" value="false" />
    <parameter name="CTL_ZQCAL_EN" value="false" />
    <parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
    <parameter name="DAT_DATA_WIDTH" value="32" />
    <parameter name="DEBUGAPB_Enable" value="false" />
    <parameter name="DEBUG_MODE" value="false" />
    <parameter name="DEVICE_DEPTH" value="1" />
    <parameter name="DEVICE_FAMILY_PARAM" value="" />
    <parameter name="DISABLE_CHILD_MESSAGING" value="false" />
    <parameter name="DISCRETE_FLY_BY" value="true" />
    <parameter name="DLL_SHARING_MODE" value="None" />
    <parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
    <parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
    <parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
    <parameter name="DUPLICATE_AC" value="false" />
    <parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
    <parameter name="EMAC0_Mode" value="N/A" />
    <parameter name="EMAC0_PTP" value="false" />
    <parameter name="EMAC0_PinMuxing" value="Unused" />
    <parameter name="EMAC1_Mode" value="RGMII" />
    <parameter name="EMAC1_PTP" value="false" />
    <parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
    <parameter name="ENABLE_BONDING" value="false" />
    <parameter name="ENABLE_BURST_MERGE" value="false" />
    <parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
    <parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
    <parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
    <parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
    <parameter name="ENABLE_EXTRA_REPORTING" value="false" />
    <parameter name="ENABLE_ISS_PROBES" value="false" />
    <parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
    <parameter name="ENABLE_NON_DES_CAL" value="false" />
    <parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
    <parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
    <parameter name="ENABLE_USER_ECC" value="false" />
    <parameter name="EXPORT_AFI_HALF_CLK" value="false" />
    <parameter name="EXTRA_SETTINGS" value="" />
    <parameter name="F2H_AXI_CLOCK_FREQ" value="50000000" />
    <parameter name="F2H_SDRAM0_CLOCK_FREQ" value="50000000" />
    <parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
    <parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
    <parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
    <parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
```

```
   <parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
   <parameter name="F2SCLK_COLDRST_Enable" value="false" />
   <parameter name="F2SCLK_DBGRST_Enable" value="false" />
   <parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
   <parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
   <parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
   <parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
   <parameter name="F2SCLK_WARMRST_Enable" value="false" />
   <parameter name="F2SDRAM_Type" value="AXI-3" />
   <parameter name="F2SDRAM_Width" value="64" />
   <parameter name="F2SINTERRUPT_Enable" value="false" />
   <parameter name="F2S_Width" value="2" />
   <parameter name="FIX_READ_LATENCY" value="8" />
   <parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
   <parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
   <parameter name="FORCE_DQS_TRACKING" value="AUTO" />
   <parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
   <parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
   <parameter name="FORCE_SHADOW_REGS" value="AUTO" />
   <parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN" value="100" />
   <parameter
      name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"
      value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK" value="125" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK" value="2.5" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK" value="125" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK" value="2.5" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT" value="100" />
   <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT" value="100" />
   <parameter name="GPIO_Enable">No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No,No,No,No,
      No,No,No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,Yes,No
      ,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No
      ,No,No,No,No,No,No,No,No,No,No,No</parameter>
   <parameter name="GP_Enable" value="false" />
   <parameter name="H2F_AXI_CLOCK_FREQ" value="25000000" />
   <parameter name="H2F_CTI_CLOCK_FREQ" value="100" />
```

```
    <parameter name="H2F_DEBUG_APB_CLOCK_FREQ" value="100" />
    <parameter name="H2F_LW_AXI_CLOCK_FREQ" value="25000000" />
    <parameter name="H2F_TPIU_CLOCK_IN_FREQ" value="100" />
    <parameter name="HARD_EMIF" value="true" />
    <parameter name="HCX_COMPAT_MODE" value="false" />
    <parameter name="HHP_HPS" value="true" />
    <parameter name="HHP_HPS_SIMULATION" value="false" />
    <parameter name="HHP_HPS_VERIFICATION" value="false" />
    <parameter name="HLGPI_Enable" value="false" />
    <parameter name="HPS_PROTOCOL" value="DDR3" />
    <parameter name="I2C0_Mode" value="I2C" />
    <parameter name="I2C0_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="I2C1_Mode" value="I2C" />
    <parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="I2C2_Mode" value="N/A" />
    <parameter name="I2C2_PinMuxing" value="Unused" />
    <parameter name="I2C3_Mode" value="N/A" />
    <parameter name="I2C3_PinMuxing" value="Unused" />
    <parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
    <parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
    <parameter name="IS_ES_DEVICE" value="false" />
    <parameter name="LOANIO_Enable">No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No
       ,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
       No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
       No,No,No,No,No,No,No,No,No,No</parameter>
    <parameter name="LOCAL_ID_WIDTH" value="8" />
    <parameter name="LRDIMM_EXTENDED_CONFIG">0x00000000000000000000</parameter>
    <parameter name="LWH2F_Enable" value="true" />
    <parameter name="MARGIN_VARIATION_TEST" value="false" />
    <parameter name="MAX_PENDING_RD_CMD" value="32" />
    <parameter name="MAX_PENDING_WR_CMD" value="16" />
    <parameter name="MEM_ASR" value="Manual" />
    <parameter name="MEM_ATCL" value="Disabled" />
    <parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
    <parameter name="MEM_BANKADDR_WIDTH" value="3" />
    <parameter name="MEM_BL" value="OTF" />
    <parameter name="MEM_BT" value="Sequential" />
    <parameter name="MEM_CK_PHASE" value="0.0" />
    <parameter name="MEM_CK_WIDTH" value="1" />
    <parameter name="MEM_CLK_EN_WIDTH" value="1" />
    <parameter name="MEM_CLK_FREQ" value="300.0" />
    <parameter name="MEM_CLK_FREQ_MAX" value="400.0" />
    <parameter name="MEM_COL_ADDR_WIDTH" value="10" />
    <parameter name="MEM_CS_WIDTH" value="1" />
    <parameter name="MEM_DEVICE" value="MISSING_MODEL" />
    <parameter name="MEM_DLL_EN" value="true" />
    <parameter name="MEM_DQ_PER_DQS" value="8" />
    <parameter name="MEM_DQ_WIDTH" value="32" />
    <parameter name="MEM_DRV_STR" value="RZQ/6" />
    <parameter name="MEM_FORMAT" value="DISCRETE" />
    <parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
    <parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
    <parameter name="MEM_IF_DM_PINS_EN" value="true" />
    <parameter name="MEM_IF_DQSN_EN" value="true" />
```

```
    <parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
    <parameter name="MEM_INIT_EN" value="false" />
    <parameter name="MEM_INIT_FILE" value="" />
    <parameter name="MEM_MIRROR_ADDRESSING" value="0" />
    <parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
    <parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
    <parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
    <parameter name="MEM_PD" value="DLL off" />
    <parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
    <parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
    <parameter name="MEM_RTT_NOM" value="ODT Disabled" />
    <parameter name="MEM_RTT_WR" value="Dynamic ODT off" />
    <parameter name="MEM_SRT" value="Normal" />
    <parameter name="MEM_TCL" value="7" />
    <parameter name="MEM_TFAW_NS" value="40.0" />
    <parameter name="MEM_TINIT_US" value="500" />
    <parameter name="MEM_TMRD_CK" value="4" />
    <parameter name="MEM_TRAS_NS" value="37.5" />
    <parameter name="MEM_TRCD_NS" value="12.5" />
    <parameter name="MEM_TREFI_US" value="7.8" />
    <parameter name="MEM_TRFC_NS" value="90.0" />
    <parameter name="MEM_TRP_NS" value="12.5" />
    <parameter name="MEM_TRRD_NS" value="13.2" />
    <parameter name="MEM_TRTP_NS" value="13.2" />
    <parameter name="MEM_TWR_NS" value="15.0" />
    <parameter name="MEM_TWTR" value="4" />
    <parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
    <parameter name="MEM_VENDOR" value="JEDEC" />
    <parameter name="MEM_VERBOSE" value="true" />
    <parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
    <parameter name="MEM_WTCL" value="6" />
    <parameter name="MPU_EVENTS_Enable" value="false" />
    <parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
    <parameter name="MULTICAST_EN" value="false" />
    <parameter name="NAND_Mode" value="N/A" />
    <parameter name="NAND_PinMuxing" value="Unused" />
    <parameter name="NEXTGEN" value="true" />
    <parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
    <parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
    <parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
    <parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
    <parameter name="NUM_OF_PORTS" value="1" />
    <parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
    <parameter name="OCT_SHARING_MODE" value="None" />
    <parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
    <parameter name="PACKAGE_DESKEW" value="false" />
    <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
    <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
    <parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
    <parameter name="PHY_CSR_ENABLED" value="false" />
    <parameter name="PHY_ONLY" value="false" />
    <parameter name="PINGPONGPHY_EN" value="false" />
    <parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
```

```
    <parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_CLK_PARAM_VALID" value="false" />
    <parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_LOCATION" value="Top_Bottom" />
    <parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
```

```
    <parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="PLL_SHARING_MODE" value="None" />
    <parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
    <parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
    <parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
    <parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
    <parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
    <parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
    <parameter name="POWER_OF_TWO_BUS" value="false" />
    <parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
    <parameter name="QSPI_Mode" value="N/A" />
    <parameter name="QSPI_PinMuxing" value="Unused" />
    <parameter name="RATE" value="Full" />
    <parameter name="RDIMM_CONFIG" value="0" />
    <parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
    <parameter name="READ_FIFO_SIZE" value="8" />
    <parameter name="REFRESH_BURST_VALIDATION" value="false" />
    <parameter name="REFRESH_INTERVAL" value="15000" />
    <parameter name="REF_CLK_FREQ" value="125.0" />
    <parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
    <parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
    <parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
    <parameter name="S2FCLK_COLDRST_Enable" value="false" />
    <parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
    <parameter name="S2FCLK_USER0CLK_Enable" value="false" />
    <parameter name="S2FCLK_USER1CLK_Enable" value="false" />
    <parameter name="S2FCLK_USER1CLK_FREQ" value="100.0" />
    <parameter name="S2FCLK_USER2CLK" value="5" />
    <parameter name="S2FCLK_USER2CLK_Enable" value="false" />
    <parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
    <parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
    <parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
    <parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
    <parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
    <parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
    <parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
    <parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
    <parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
    <parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
    <parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
    <parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
    <parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
    <parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
```

```
    <parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
    <parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
    <parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
    <parameter name="S2FINTERRUPT_UART_Enable" value="false" />
    <parameter name="S2FINTERRUPT_USB_Enable" value="false" />
    <parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
    <parameter name="S2F_Width" value="2" />
    <parameter name="SDIO_Mode" value="4-bit Data" />
    <parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="SEQUENCER_TYPE" value="NIOS" />
    <parameter name="SEQ_MODE" value="0" />
    <parameter name="SKIP_MEM_INIT" value="true" />
    <parameter name="SOPC_COMPAT_RESET" value="false" />
    <parameter name="SPEED_GRADE" value="7" />
    <parameter name="SPIM0_Mode" value="N/A" />
    <parameter name="SPIM0_PinMuxing" value="Unused" />
    <parameter name="SPIM1_Mode" value="Single Slave Select" />
    <parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="SPIS0_Mode" value="N/A" />
    <parameter name="SPIS0_PinMuxing" value="Unused" />
    <parameter name="SPIS1_Mode" value="N/A" />
    <parameter name="SPIS1_PinMuxing" value="Unused" />
    <parameter name="STARVE_LIMIT" value="10" />
    <parameter name="STM_Enable" value="false" />
    <parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
    <parameter name="TEST_Enable" value="false" />
    <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
    <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
    <parameter name="TIMING_BOARD_AC_SKEW" value="0.02" />
    <parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
    <parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
    <parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
    <parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
    <parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
    <parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
    <parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
    <parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
    <parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
    <parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
    <parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
    <parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.6" />
    <parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.6" />
    <parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
    <parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
    <parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.02" />
    <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.01" />
    <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="-0.01" />
    <parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.02" />
    <parameter name="TIMING_BOARD_TDH" value="0.0" />
    <parameter name="TIMING_BOARD_TDS" value="0.0" />
    <parameter name="TIMING_BOARD_TIH" value="0.0" />
    <parameter name="TIMING_BOARD_TIS" value="0.0" />
    <parameter name="TIMING_TDH" value="150" />
    <parameter name="TIMING_TDQSCK" value="400" />
```

```
    <parameter name="TIMING_TDQSCKDL" value="1200" />
    <parameter name="TIMING_TDQSCKDM" value="900" />
    <parameter name="TIMING_TDQSCKDS" value="450" />
    <parameter name="TIMING_TDQSH" value="0.35" />
    <parameter name="TIMING_TDQSQ" value="200" />
    <parameter name="TIMING_TDQSS" value="0.25" />
    <parameter name="TIMING_TDS" value="75" />
    <parameter name="TIMING_TDSH" value="0.2" />
    <parameter name="TIMING_TDSS" value="0.2" />
    <parameter name="TIMING_TIH" value="275" />
    <parameter name="TIMING_TIS" value="200" />
    <parameter name="TIMING_TQH" value="0.38" />
    <parameter name="TIMING_TQHS" value="300" />
    <parameter name="TIMING_TQSH" value="0.38" />
    <parameter name="TPIUFPGA_Enable" value="false" />
    <parameter name="TPIUFPGA_alt" value="false" />
    <parameter name="TRACE_Mode" value="N/A" />
    <parameter name="TRACE_PinMuxing" value="Unused" />
    <parameter name="TRACKING_ERROR_TEST" value="false" />
    <parameter name="TRACKING_WATCH_TEST" value="false" />
    <parameter name="TREFI" value="35100" />
    <parameter name="TRFC" value="350" />
    <parameter name="UART0_Mode" value="No Flow Control" />
    <parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="UART1_Mode" value="N/A" />
    <parameter name="UART1_PinMuxing" value="Unused" />
    <parameter name="USB0_Mode" value="N/A" />
    <parameter name="USB0_PinMuxing" value="Unused" />
    <parameter name="USB1_Mode" value="SDR" />
    <parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="USER_DEBUG_LEVEL" value="1" />
    <parameter name="USE_AXI_ADAPTOR" value="false" />
    <parameter name="USE_FAKE_PHY" value="false" />
    <parameter name="USE_MEM_CLK_FREQ" value="false" />
    <parameter name="USE_MM_ADAPTOR" value="true" />
    <parameter name="USE_SEQUENCER_BFM" value="false" />
    <parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
    <parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
    <parameter name="can0_clk_div" value="1" />
    <parameter name="can1_clk_div" value="1" />
    <parameter name="configure_advanced_parameters" value="false" />
    <parameter name="customize_device_pll_info" value="false" />
    <parameter name="dbctrl_stayosc1" value="true" />
    <parameter name="dbg_at_clk_div" value="0" />
    <parameter name="dbg_clk_div" value="1" />
    <parameter name="dbg_trace_clk_div" value="0" />
    <parameter name="desired_can0_clk_mhz" value="100.0" />
    <parameter name="desired_can1_clk_mhz" value="100.0" />
    <parameter name="desired_cfg_clk_mhz" value="100.0" />
    <parameter name="desired_emac0_clk_mhz" value="250.0" />
    <parameter name="desired_emac1_clk_mhz" value="250.0" />
    <parameter name="desired_gpio_db_clk_hz" value="32000" />
    <parameter name="desired_l4_mp_clk_mhz" value="100.0" />
    <parameter name="desired_l4_sp_clk_mhz" value="100.0" />
```

```
546   <parameter name="desired_mpu_clk_mhz" value="800.0" />
547   <parameter name="desired_nand_clk_mhz" value="12.5" />
548   <parameter name="desired_qspi_clk_mhz" value="400.0" />
549   <parameter name="desired_sdmmc_clk_mhz" value="200.0" />
550   <parameter name="desired_spi_m_clk_mhz" value="200.0" />
551   <parameter name="desired_usb_mp_clk_mhz" value="200.0" />
552   <parameter name="device_name" value="5CSEMA5F31C6" />
553   <parameter name="device_pll_info_manual">{320000000 1600000000} {320000000 1000000000} {800000000
        400000000 400000000}</parameter>
554   <parameter name="eosc1_clk_mhz" value="25.0" />
555   <parameter name="eosc2_clk_mhz" value="25.0" />
556   <parameter name="gpio_db_clk_div" value="6249" />
557   <parameter name="l3_mp_clk_div" value="1" />
558   <parameter name="l3_sp_clk_div" value="1" />
559   <parameter name="l4_mp_clk_div" value="1" />
560   <parameter name="l4_mp_clk_source" value="1" />
561   <parameter name="l4_sp_clk_div" value="1" />
562   <parameter name="l4_sp_clk_source" value="1" />
563   <parameter name="main_pll_c3" value="3" />
564   <parameter name="main_pll_c4" value="3" />
565   <parameter name="main_pll_c5" value="15" />
566   <parameter name="main_pll_m" value="63" />
567   <parameter name="main_pll_n" value="0" />
568   <parameter name="nand_clk_source" value="2" />
569   <parameter name="periph_pll_c0" value="3" />
570   <parameter name="periph_pll_c1" value="3" />
571   <parameter name="periph_pll_c2" value="1" />
572   <parameter name="periph_pll_c3" value="19" />
573   <parameter name="periph_pll_c4" value="4" />
574   <parameter name="periph_pll_c5" value="9" />
575   <parameter name="periph_pll_m" value="79" />
576   <parameter name="periph_pll_n" value="1" />
577   <parameter name="periph_pll_source" value="0" />
578   <parameter name="qspi_clk_source" value="1" />
579   <parameter name="quartus_ini_hps_emif_pll" value="false" />
580   <parameter
581      name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
582      value="false" />
583   <parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
584   <parameter
585      name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
586      value="false" />
587   <parameter
588      name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
589      value="false" />
590   <parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
591   <parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false" />
592   <parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model" value="false" />
593   <parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false" />
594   <parameter name="sdmmc_clk_source" value="2" />
595   <parameter name="show_advanced_parameters" value="false" />
596   <parameter name="show_debug_info_as_warning_msg" value="false" />
597   <parameter name="show_warning_as_error_msg" value="false" />
598   <parameter name="spi_m_clk_div" value="0" />
```

```
   <parameter name="usb_mp_clk_div" value="0" />
   <parameter name="use_default_mpu_clk" value="true" />
  </module>
  <connection
    kind="avalon"
    version="21.1"
    start="hps_0.h2f_axi_master"
    end="cnn_mmio_interface_0.avalon_slave_0">
   <parameter name="arbitrationPriority" value="1" />
   <parameter name="baseAddress" value="0x0000" />
   <parameter name="defaultConnection" value="false" />
  </connection>
  <connection
    kind="avalon"
    version="21.1"
    start="hps_0.h2f_lw_axi_master"
    end="cnn_mmio_interface_0.avalon_slave_0">
   <parameter name="arbitrationPriority" value="1" />
   <parameter name="baseAddress" value="0x0000" />
   <parameter name="defaultConnection" value="false" />
  </connection>
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="cnn_mmio_interface_0.clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.f2h_axi_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.f2h_sdram0_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.h2f_axi_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.h2f_lw_axi_clock" />
  <connection
    kind="reset"
    version="21.1"
    start="clk_0.clk_reset"
    end="cnn_mmio_interface_0.reset" />
  <connection
    kind="reset"
    version="21.1"
```

```
    start="hps_0.h2f_reset"
    end="clk_0.clk_in_reset" />
  <interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
  <interconnectRequirement for="$system" name="qsys_mm.enableEccProtection" value="FALSE" />
  <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave" value="FALSE" />
  <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="1" />
</system>
```

A.6.5 Timing Constraint File

Defines the board clock, derived PLL clock handling, and clock uncertainty constraints used by Quartus timing analysis.

code/de1_soc/soc_system.sdc

```
1
2     create_clock -name clock_50 -period 20ns [get_ports CLOCK_50]
3     derive_pll_clocks -create_base_clocks
4     derive_clock_uncertainty
```

A.7 FPGA Board Boundary And Accelerator Top

A.7.1 DE1-SoC Board Wrapper

Connects the generated HPS system, board pins, clocks/resets, and seven-segment debug display.

code/de1_soc/soc_system_top.sv

```
1  module soc_system_top (
2      input  wire        CLOCK_50,
3      output wire [6:0]  HEX0,
4      output wire [6:0]  HEX1,
5      output wire [6:0]  HEX2,
6      output wire [6:0]  HEX3,
7      output wire [6:0]  HEX4,
8      output wire [6:0]  HEX5,
9      input  wire        HPS_UART_RX,
10     output wire        HPS_UART_TX,
11     output wire        HPS_SD_CLK,
12     inout  wire        HPS_SD_CMD,
13     inout  wire [3:0]  HPS_SD_DATA,
14     inout  wire        HPS_I2C1_SCLK,
15     inout  wire        HPS_I2C1_SDAT,
16     inout  wire        HPS_I2C2_SCLK,
17     inout  wire        HPS_I2C2_SDAT,
18     inout  wire        HPS_I2C_CONTROL,
```

```
    output wire        HPS_ENET_GTX_CLK,
    inout  wire        HPS_ENET_INT_N,
    output wire        HPS_ENET_MDC,
    inout  wire        HPS_ENET_MDIO,
    input  wire        HPS_ENET_RX_CLK,
    input  wire [3:0]  HPS_ENET_RX_DATA,
    input  wire        HPS_ENET_RX_DV,
    output wire [3:0]  HPS_ENET_TX_DATA,
    output wire        HPS_ENET_TX_EN,
    output wire        HPS_SPIM_CLK,
    input  wire        HPS_SPIM_MISO,
    output wire        HPS_SPIM_MOSI,
    inout  wire        HPS_SPIM_SS,
    input  wire        HPS_USB_CLKOUT,
    inout  wire [7:0]  HPS_USB_DATA,
    input  wire        HPS_USB_DIR,
    input  wire        HPS_USB_NXT,
    output wire        HPS_USB_STP,
    inout  wire        HPS_CONV_USB_N,
    inout  wire        HPS_GSENSOR_INT,
    inout  wire        HPS_KEY,
    inout  wire        HPS_LED,
    inout  wire        HPS_LTC_GPIO,

    output wire [14:0] HPS_DDR3_ADDR,
    output wire [2:0]  HPS_DDR3_BA,
    output wire        HPS_DDR3_CAS_N,
    output wire        HPS_DDR3_CKE,
    output wire        HPS_DDR3_CK_N,
    output wire        HPS_DDR3_CK_P,
    output wire        HPS_DDR3_CS_N,
    output wire [3:0]  HPS_DDR3_DM,
    inout  wire [31:0] HPS_DDR3_DQ,
    inout  wire [3:0]  HPS_DDR3_DQS_N,
    inout  wire [3:0]  HPS_DDR3_DQS_P,
    output wire        HPS_DDR3_ODT,
    output wire        HPS_DDR3_RAS_N,
    output wire        HPS_DDR3_RESET_N,
    input  wire        HPS_DDR3_RZQ,
    output wire        HPS_DDR3_WE_N
);
    wire [3:0] predict_class_dbg;
    wire       predict_done_dbg;
    wire       model_loaded_dbg;
    wire [15:0] interface_error_dbg;

    localparam [6:0] SEVEN_SEG_BLANK = 7'b1111111;

    function automatic [6:0] seven_seg_decode;
        input [3:0] value;
        begin
            case (value)
                4'h0: seven_seg_decode = 7'b1000000;
                4'h1: seven_seg_decode = 7'b1111001;
```

```
                4'h2: seven_seg_decode = 7'b0100100;
                4'h3: seven_seg_decode = 7'b0110000;
                4'h4: seven_seg_decode = 7'b0011001;
                4'h5: seven_seg_decode = 7'b0010010;
                4'h6: seven_seg_decode = 7'b0000010;
                4'h7: seven_seg_decode = 7'b1111000;
                4'h8: seven_seg_decode = 7'b0000000;
                4'h9: seven_seg_decode = 7'b0010000;
                4'hA: seven_seg_decode = 7'b0001000; // A
                4'hB: seven_seg_decode = 7'b0000011; // b
                4'hC: seven_seg_decode = 7'b1000110; // C
                4'hD: seven_seg_decode = 7'b0100001; // d
                4'hE: seven_seg_decode = 7'b0000110; // E
                default: seven_seg_decode = SEVEN_SEG_BLANK;
            endcase
        end
    endfunction

    soc_system u_soc_system (
        .clk_clk            (CLOCK_50),
        .cnn_debug_model_loaded(model_loaded_dbg),
        .cnn_debug_predict_done(predict_done_dbg),
        .cnn_debug_predict_class(predict_class_dbg),
        .cnn_debug_interface_error(interface_error_dbg),
        .reset_reset_n      (1'b1),
        .memory_mem_a       (HPS_DDR3_ADDR),
        .memory_mem_ba      (HPS_DDR3_BA),
        .memory_mem_ck      (HPS_DDR3_CK_P),
        .memory_mem_ck_n    (HPS_DDR3_CK_N),
        .memory_mem_cke     (HPS_DDR3_CKE),
        .memory_mem_cs_n    (HPS_DDR3_CS_N),
        .memory_mem_ras_n   (HPS_DDR3_RAS_N),
        .memory_mem_cas_n   (HPS_DDR3_CAS_N),
        .memory_mem_we_n    (HPS_DDR3_WE_N),
        .memory_mem_reset_n (HPS_DDR3_RESET_N),
        .memory_mem_dq      (HPS_DDR3_DQ),
        .memory_mem_dqs     (HPS_DDR3_DQS_P),
        .memory_mem_dqs_n   (HPS_DDR3_DQS_N),
        .memory_mem_odt     (HPS_DDR3_ODT),
        .memory_mem_dm      (HPS_DDR3_DM),
        .memory_oct_rzqin   (HPS_DDR3_RZQ),
        .hps_hps_io_emac1_inst_TX_CLK (HPS_ENET_GTX_CLK),
        .hps_hps_io_emac1_inst_TXD0   (HPS_ENET_TX_DATA[0]),
        .hps_hps_io_emac1_inst_TXD1   (HPS_ENET_TX_DATA[1]),
        .hps_hps_io_emac1_inst_TXD2   (HPS_ENET_TX_DATA[2]),
        .hps_hps_io_emac1_inst_TXD3   (HPS_ENET_TX_DATA[3]),
        .hps_hps_io_emac1_inst_RXD0   (HPS_ENET_RX_DATA[0]),
        .hps_hps_io_emac1_inst_MDIO   (HPS_ENET_MDIO),
        .hps_hps_io_emac1_inst_MDC    (HPS_ENET_MDC),
        .hps_hps_io_emac1_inst_RX_CTL (HPS_ENET_RX_DV),
        .hps_hps_io_emac1_inst_TX_CTL (HPS_ENET_TX_EN),
        .hps_hps_io_emac1_inst_RX_CLK (HPS_ENET_RX_CLK),
        .hps_hps_io_emac1_inst_RXD1   (HPS_ENET_RX_DATA[1]),
        .hps_hps_io_emac1_inst_RXD2   (HPS_ENET_RX_DATA[2]),
```

```
        .hps_hps_io_emac1_inst_RXD3   (HPS_ENET_RX_DATA[3]),
        .hps_hps_io_sdio_inst_CMD     (HPS_SD_CMD),
        .hps_hps_io_sdio_inst_D0      (HPS_SD_DATA[0]),
        .hps_hps_io_sdio_inst_D1      (HPS_SD_DATA[1]),
        .hps_hps_io_sdio_inst_CLK     (HPS_SD_CLK),
        .hps_hps_io_sdio_inst_D2      (HPS_SD_DATA[2]),
        .hps_hps_io_sdio_inst_D3      (HPS_SD_DATA[3]),
        .hps_hps_io_usb1_inst_D0      (HPS_USB_DATA[0]),
        .hps_hps_io_usb1_inst_D1      (HPS_USB_DATA[1]),
        .hps_hps_io_usb1_inst_D2      (HPS_USB_DATA[2]),
        .hps_hps_io_usb1_inst_D3      (HPS_USB_DATA[3]),
        .hps_hps_io_usb1_inst_D4      (HPS_USB_DATA[4]),
        .hps_hps_io_usb1_inst_D5      (HPS_USB_DATA[5]),
        .hps_hps_io_usb1_inst_D6      (HPS_USB_DATA[6]),
        .hps_hps_io_usb1_inst_D7      (HPS_USB_DATA[7]),
        .hps_hps_io_usb1_inst_CLK     (HPS_USB_CLKOUT),
        .hps_hps_io_usb1_inst_STP     (HPS_USB_STP),
        .hps_hps_io_usb1_inst_DIR     (HPS_USB_DIR),
        .hps_hps_io_usb1_inst_NXT     (HPS_USB_NXT),
        .hps_hps_io_spim1_inst_CLK    (HPS_SPIM_CLK),
        .hps_hps_io_spim1_inst_MOSI   (HPS_SPIM_MOSI),
        .hps_hps_io_spim1_inst_MISO   (HPS_SPIM_MISO),
        .hps_hps_io_spim1_inst_SS0    (HPS_SPIM_SS),
        .hps_hps_io_uart0_inst_RX     (HPS_UART_RX),
        .hps_hps_io_uart0_inst_TX     (HPS_UART_TX),
        .hps_hps_io_i2c0_inst_SDA     (HPS_I2C1_SDAT),
        .hps_hps_io_i2c0_inst_SCL     (HPS_I2C1_SCLK),
        .hps_hps_io_i2c1_inst_SDA     (HPS_I2C2_SDAT),
        .hps_hps_io_i2c1_inst_SCL     (HPS_I2C2_SCLK),
        .hps_hps_io_gpio_inst_GPIO09  (HPS_CONV_USB_N),
        .hps_hps_io_gpio_inst_GPIO35  (HPS_ENET_INT_N),
        .hps_hps_io_gpio_inst_GPIO40  (HPS_LTC_GPIO),
        .hps_hps_io_gpio_inst_GPIO48  (HPS_I2C_CONTROL),
        .hps_hps_io_gpio_inst_GPIO53  (HPS_LED),
        .hps_hps_io_gpio_inst_GPIO54  (HPS_KEY),
        .hps_hps_io_gpio_inst_GPIO61  (HPS_GSENSOR_INT)
    );

    // Current seven-segment policy:
    // - HEX0 shows the predicted class once predict_done is asserted
    // - HEX1 shows model_loaded / predict_done as two debug bits
    // - HEX2 shows the low nibble of interface_error when non-zero
    // - HEX3/HEX4 stay blank
    // - HEX5 shows "E" only when an interface error is latched
    assign HEX0 = predict_done_dbg ? seven_seg_decode(predict_class_dbg) : SEVEN_SEG_BLANK;
    assign HEX1 = seven_seg_decode({2'b00, model_loaded_dbg, predict_done_dbg});
    assign HEX2 = (interface_error_dbg != 16'h0000) ? seven_seg_decode(interface_error_dbg[3:0]) :
     SEVEN_SEG_BLANK;
    assign HEX3 = SEVEN_SEG_BLANK;
    assign HEX4 = SEVEN_SEG_BLANK;
    assign HEX5 = (interface_error_dbg != 16'h0000) ? seven_seg_decode(4'hE) : SEVEN_SEG_BLANK;

endmodule
```

A.7.2 Avalon-MM MMIO Wrapper

Implements the scratchpad, register file, replay FSM, status/profile readback, and stream bridge into the accelerator.

code/input/RTL/interface/cnn_mmio_interface.v

```
`timescale 1ns / 1ps

// cnn_mmio_interface
// ------------------
// A thin MMIO/BRAM wrapper around system_top.
//
// address[12] = 0 : memory space   (16 KiB 32-bit word BRAM)
// address[12] = 1 : config space   (32-bit MMIO registers, low 16 bits keep
//                                   the original control/status ABI)
//
// The wrapper keeps the existing accelerator datapath intact and only adds:
//   1) a stable memory image interface for model/image staging
//   2) a small register block for base/length programming
//   3) a sequencer that replays staged 32-bit words into system_top
//
// This keeps the current CNN_ACC architecture and borrows only the transaction
// pattern from the reference cnn_interface design.
//
// Notes on the slightly "mixed-width" looking register map:
//   - Control/status bits stay in the low 16 bits so the existing HPS helpers
//     and docs keep working after the move to a 32-bit Avalon-MM slave.
//   - Profile counters are already 32-bit values, so the legacy *_HI slots are
//     treated as the canonical full-width readback locations. The matching
//     *_LO slots are kept as zero/reserved padding for compatibility.

module cnn_mmio_interface #(
  parameter integer MEM_AW       = 12,
  parameter integer MEM_WORDS    = (1 << MEM_AW)
) (
  input  wire        clk,
  input  wire        reset,
  input  wire [31:0] writedata,
  input  wire [3:0]  byteenable,
  input  wire        write,
  input  wire        read,
  input  wire        chipselect,
  input  wire [MEM_AW:0] address,
  output reg  [31:0] readdata,
  output wire        debug_model_loaded,
  output wire        debug_predict_done,
  output wire [3:0]  debug_predict_class,
  output wire [15:0] debug_interface_error
);

  localparam [4:0] REG_CONTROL       = 5'd0;
  localparam [4:0] REG_STATUS        = 5'd1;
```

```
    localparam [4:0] REG_CONV_CFG_BASE = 5'd2;
    localparam [4:0] REG_CONV_CFG_LEN  = 5'd3;
    localparam [4:0] REG_CONV_WT_BASE  = 5'd4;
    localparam [4:0] REG_CONV_WT_LEN   = 5'd5;
    localparam [4:0] REG_FC_BIAS_BASE  = 5'd6;
    localparam [4:0] REG_FC_BIAS_LEN   = 5'd7;
    localparam [4:0] REG_FCW_BASE      = 5'd8;
    localparam [4:0] REG_FCW_LEN       = 5'd9;
    localparam [4:0] REG_IMAGE_BASE    = 5'd10;
    localparam [4:0] REG_IMAGE_LEN     = 5'd11;
    localparam [4:0] REG_PREDICT       = 5'd12;
    localparam [4:0] REG_IF_ERROR      = 5'd13;
    localparam [4:0] REG_PROFILE_L1_LO     = 5'd14;
    localparam [4:0] REG_PROFILE_L1_HI     = 5'd15;
    localparam [4:0] REG_PROFILE_L2_P0_LO  = 5'd16;
    localparam [4:0] REG_PROFILE_L2_P0_HI  = 5'd17;
    localparam [4:0] REG_PROFILE_L2_P1_LO  = 5'd18;
    localparam [4:0] REG_PROFILE_L2_P1_HI  = 5'd19;
    localparam [4:0] REG_PROFILE_L3_P0_LO  = 5'd20;
    localparam [4:0] REG_PROFILE_L3_P0_HI  = 5'd21;
    localparam [4:0] REG_PROFILE_L3_P1_LO  = 5'd22;
    localparam [4:0] REG_PROFILE_L3_P1_HI  = 5'd23;
    localparam [4:0] REG_PROFILE_FC_LO     = 5'd24;
    localparam [4:0] REG_PROFILE_FC_HI     = 5'd25;
    localparam [4:0] REG_PROFILE_ARGMAX_LO = 5'd26;
    localparam [4:0] REG_PROFILE_ARGMAX_HI = 5'd27;
    localparam [4:0] REG_PROFILE_TOTAL_LO  = 5'd28;
    localparam [4:0] REG_PROFILE_TOTAL_HI  = 5'd29;
    localparam [4:0] REG_LAST_WRITE        = 5'd30;
    localparam [4:0] REG_MAGIC             = 5'd31;

    localparam [31:0] CFG_MAGIC_WORD = 32'h434E4E32; // "CNN2"

    localparam [1:0] ENG_IDLE    = 2'd0;
    localparam [1:0] ENG_MODEL   = 2'd1;
    localparam [1:0] ENG_INFER   = 2'd2;

    localparam [2:0] ST_IDLE     = 3'd0;
    localparam [2:0] ST_RD       = 3'd1;
    localparam [2:0] ST_WAIT_RD  = 3'd2;
    localparam [2:0] ST_SEND     = 3'd3;
    localparam [2:0] ST_GAP      = 3'd4;

    localparam [1:0] SEG_CONV_CFG = 2'd0;
    localparam [1:0] SEG_CONV_WT  = 2'd1;
    localparam [1:0] SEG_FC_BIAS  = 2'd2;
    localparam [1:0] SEG_FCW      = 2'd3;

    // ------------------------------------------------------------------------
    // 32-bit staging BRAM
    // ------------------------------------------------------------------------
    (* ramstyle = "M10K" *)
    reg [31:0] mem [0:MEM_WORDS-1];

```

```
  reg [31:0]       mem_rdata_q;
  reg [31:0]       host_rdata_q;
  reg [31:0]       last_writedata_q;
  reg [3:0]        last_byteenable_q;
  reg              host_mem_rd_q;
  reg [MEM_AW-1:0] mem_rd_addr_q;
  reg              mem_rd_pending_q;
  reg              mem_rd_host_q;

  wire mem_sel    = chipselect && !address[MEM_AW];
  wire cfg_sel    = chipselect &&  address[MEM_AW];
  wire mem_wr_req = mem_sel &&  write;
  wire mem_rd_req = mem_sel &&  read;
  wire cfg_wr_req = cfg_sel &&  write;
  wire cfg_rd_req = cfg_sel &&  read;
  wire mem_addr_ok = (address[MEM_AW-1:0] < MEM_WORDS);
  wire mem_any_byte_en = |byteenable;
  wire cfg_lo_byte_en = |byteenable[1:0];

  // ------------------------------------------------------------------------
  // Config registers
  // ------------------------------------------------------------------------
  reg [15:0] conv_cfg_base_w;
  reg [15:0] conv_cfg_len_w;
  reg [15:0] conv_wt_base_w;
  reg [15:0] conv_wt_len_w;
  reg [15:0] fc_bias_base_w;
  reg [15:0] fc_bias_len_w;
  reg [15:0] fcw_base_w;
  reg [15:0] fcw_len_w;
  reg [15:0] image_base_w;
  reg [15:0] image_len_w;

  reg        model_loaded;
  reg        predict_done;
  reg [3:0]  predict_class_latched;
  reg [15:0] interface_error;
  reg [1:0]  eng_mode;
  wire       eng_busy = (eng_mode != ENG_IDLE);

  wire [15:0] cfg_status_word = {
    8'd0,
    predict_class_latched,
    predict_done,
    model_loaded,
    eng_busy,
    1'b0
  };

  reg [31:0] cfg_read_data;
  always @(*) begin
    cfg_read_data = 32'd0;
    case (address[4:0])
      REG_STATUS:        cfg_read_data = cfg_status_word;
```

```
155         REG_CONV_CFG_BASE: cfg_read_data = conv_cfg_base_w;
156         REG_CONV_CFG_LEN:  cfg_read_data = conv_cfg_len_w;
157         REG_CONV_WT_BASE:  cfg_read_data = conv_wt_base_w;
158         REG_CONV_WT_LEN:   cfg_read_data = conv_wt_len_w;
159         REG_FC_BIAS_BASE:  cfg_read_data = fc_bias_base_w;
160         REG_FC_BIAS_LEN:   cfg_read_data = fc_bias_len_w;
161         REG_FCW_BASE:      cfg_read_data = fcw_base_w;
162         REG_FCW_LEN:       cfg_read_data = fcw_len_w;
163         REG_IMAGE_BASE:    cfg_read_data = image_base_w;
164         REG_IMAGE_LEN:     cfg_read_data = image_len_w;
165         REG_PREDICT:       cfg_read_data = {12'd0, predict_class_latched};
166         REG_IF_ERROR:      cfg_read_data = interface_error;
167         // Keep the legacy low-half register numbers reserved. Software should
168         // read the *_HI aliases for the full 32-bit counters.
169         REG_PROFILE_L1_LO:     cfg_read_data = 32'd0;
170         REG_PROFILE_L1_HI:     cfg_read_data = accel_profile_l1_cycles;
171         REG_PROFILE_L2_P0_LO:  cfg_read_data = 32'd0;
172         REG_PROFILE_L2_P0_HI:  cfg_read_data = accel_profile_l2_p0_cycles;
173         REG_PROFILE_L2_P1_LO:  cfg_read_data = 32'd0;
174         REG_PROFILE_L2_P1_HI:  cfg_read_data = accel_profile_l2_p1_cycles;
175         REG_PROFILE_L3_P0_LO:  cfg_read_data = 32'd0;
176         REG_PROFILE_L3_P0_HI:  cfg_read_data = accel_profile_l3_p0_cycles;
177         REG_PROFILE_L3_P1_LO:  cfg_read_data = 32'd0;
178         REG_PROFILE_L3_P1_HI:  cfg_read_data = accel_profile_l3_p1_cycles;
179         REG_PROFILE_FC_LO:     cfg_read_data = 32'd0;
180         REG_PROFILE_FC_HI:     cfg_read_data = accel_profile_fc_cycles;
181         REG_PROFILE_ARGMAX_LO: cfg_read_data = 32'd0;
182         REG_PROFILE_ARGMAX_HI: cfg_read_data = accel_profile_argmax_cycles;
183         REG_PROFILE_TOTAL_LO:  cfg_read_data = 32'd0;
184         REG_PROFILE_TOTAL_HI:  cfg_read_data = accel_profile_total_cycles;
185         REG_LAST_WRITE:     cfg_read_data = { last_byteenable_q, last_writedata_q[27:0] };
186         REG_MAGIC:          cfg_read_data = CFG_MAGIC_WORD;
187         default:           cfg_read_data = 32'd0;
188       endcase
189     end
190
191     // ------------------------------------------------------------------------
192     // system_top host-stream interface
193     // ------------------------------------------------------------------------
194     reg         accel_load_sel;
195     reg         accel_load_valid;
196     reg  [31:0] accel_load_data;
197     reg         accel_load_last;
198     wire        accel_load_ready;
199     wire        accel_busy;
200     wire        accel_predict_valid;
201     wire [3:0]  accel_predict_class;
202     wire [31:0] accel_profile_l1_cycles;
203     wire [31:0] accel_profile_l2_p0_cycles;
204     wire [31:0] accel_profile_l2_p1_cycles;
205     wire [31:0] accel_profile_l3_p0_cycles;
206     wire [31:0] accel_profile_l3_p1_cycles;
207     wire [31:0] accel_profile_fc_cycles;
208     wire [31:0] accel_profile_argmax_cycles;
```

```
  wire [31:0] accel_profile_total_cycles;

  system_top u_system_top (
    .clk(clk),
    .rst_n(~reset),
    .load_sel(accel_load_sel),
    .load_valid(accel_load_valid),
    .load_data(accel_load_data),
    .load_last(accel_load_last),
    .load_ready(accel_load_ready),
    .busy(accel_busy),
    .predict_valid(accel_predict_valid),
    .predict_class(accel_predict_class),
    .profile_l1_cycles(accel_profile_l1_cycles),
    .profile_l2_p0_cycles(accel_profile_l2_p0_cycles),
    .profile_l2_p1_cycles(accel_profile_l2_p1_cycles),
    .profile_l3_p0_cycles(accel_profile_l3_p0_cycles),
    .profile_l3_p1_cycles(accel_profile_l3_p1_cycles),
    .profile_fc_cycles(accel_profile_fc_cycles),
    .profile_argmax_cycles(accel_profile_argmax_cycles),
    .profile_total_cycles(accel_profile_total_cycles)
  );

  // ---------------------------------------------------------------------------
  // Replay engine
  // ---------------------------------------------------------------------------
  reg [2:0] eng_state;
  reg [1:0] model_seg_idx;
  reg [15:0] seg_word_idx;
  reg [15:0] seg_word_count;

  function [15:0] model_seg_base;
    input [1:0] seg;
    begin
      case (seg)
        SEG_CONV_CFG: model_seg_base = conv_cfg_base_w;
        SEG_CONV_WT:  model_seg_base = conv_wt_base_w;
        SEG_FC_BIAS:  model_seg_base = fc_bias_base_w;
        default:      model_seg_base = fcw_base_w;
      endcase
    end
  endfunction

  function [15:0] model_seg_len;
    input [1:0] seg;
    begin
      case (seg)
        SEG_CONV_CFG: model_seg_len = conv_cfg_len_w;
        SEG_CONV_WT:  model_seg_len = conv_wt_len_w;
        SEG_FC_BIAS:  model_seg_len = fc_bias_len_w;
        default:      model_seg_len = fcw_len_w;
      endcase
    end
  endfunction
```

```
263
264   // Memory read arbitration: the engine owns BRAM while active.
265   wire issue_eng_read = (eng_state == ST_RD);
266   wire [15:0] current_seg_base = (eng_mode == ENG_INFER) ? image_base_w
267                                                          : model_seg_base(model_seg_idx);
268   wire [15:0] current_seg_len  = (eng_mode == ENG_INFER) ? image_len_w
269                                                          : model_seg_len(model_seg_idx);
270   wire [15:0] current_word_addr = current_seg_base + seg_word_idx;
271   wire eng_read_in_range = (current_word_addr < MEM_WORDS);
272
273   always @(posedge clk) begin
274     if (reset) begin
275       mem_rdata_q      <= 32'd0;
276       host_rdata_q     <= 32'd0;
277       last_writedata_q  <= 32'd0;
278       last_byteenable_q <= 4'd0;
279       host_mem_rd_q    <= 1'b0;
280       mem_rd_addr_q    <= {MEM_AW{1'b0}};
281       mem_rd_pending_q <= 1'b0;
282       mem_rd_host_q    <= 1'b0;
283     end else begin
284       host_mem_rd_q <= 1'b0;
285
286       if (cfg_rd_req) begin
287         host_rdata_q <= cfg_read_data;
288       end
289
290       if (host_mem_rd_q)
291         host_rdata_q <= mem_rdata_q;
292
293       if (mem_wr_req || cfg_wr_req) begin
294         last_writedata_q  <= writedata;
295         last_byteenable_q <= byteenable;
296       end
297
298       // The staged scratchpad expects full-word host writes. We only use
299       // byteenable as a write-present qualifier here to keep the memory inferable
300       // as block RAM on Cyclone V.
301       if (mem_wr_req && !eng_busy && mem_addr_ok && mem_any_byte_en)
302         mem[address[MEM_AW-1:0]] <= writedata;
303
304       if (mem_rd_pending_q) begin
305         mem_rdata_q <= mem[mem_rd_addr_q];
306         host_mem_rd_q <= mem_rd_host_q;
307       end
308
309       mem_rd_pending_q <= 1'b0;
310
311       if (issue_eng_read && eng_read_in_range) begin
312         mem_rd_addr_q    <= current_word_addr[MEM_AW-1:0];
313         mem_rd_pending_q <= 1'b1;
314         mem_rd_host_q    <= 1'b0;
315       end else if (mem_rd_req && !eng_busy && mem_addr_ok) begin
316         mem_rd_addr_q    <= address[MEM_AW-1:0];
```

```
        mem_rd_pending_q <= 1'b1;
        mem_rd_host_q    <= 1'b1;
      end
    end
  end

  // ---------------------------------------------------------------------------
  // Register block, status latches, and replay FSM
  // ---------------------------------------------------------------------------
  always @(posedge clk) begin
    if (reset) begin
      conv_cfg_base_w       <= 16'd0;
      conv_cfg_len_w        <= 16'd45;
      conv_wt_base_w        <= 16'd45;
      conv_wt_len_w         <= 16'd225;
      fc_bias_base_w        <= 16'd270;
      fc_bias_len_w         <= 16'd10;
      fcw_base_w            <= 16'd280;
      fcw_len_w             <= 16'd864;
      image_base_w          <= 16'd1144;
      image_len_w           <= 16'd1024;
      model_loaded          <= 1'b0;
      predict_done          <= 1'b0;
      predict_class_latched <= 4'd0;
      interface_error       <= 16'd0;
      eng_mode              <= ENG_IDLE;
      eng_state             <= ST_IDLE;
      model_seg_idx         <= SEG_CONV_CFG;
      seg_word_idx          <= 16'd0;
      seg_word_count        <= 16'd0;
    end else begin
      if (accel_predict_valid) begin
        predict_done          <= 1'b1;
        predict_class_latched <= accel_predict_class;
      end

      if (cfg_wr_req && cfg_lo_byte_en) begin
        case (address[4:0])
          REG_CONTROL: begin
            if (writedata[2]) begin
              predict_done    <= 1'b0;
              interface_error <= 16'd0;
            end

            if (writedata[0]) begin
              if (eng_busy || accel_busy) begin
                interface_error[0] <= 1'b1;
              end else begin
                eng_mode       <= ENG_MODEL;
                eng_state      <= ST_RD;
                model_seg_idx  <= SEG_CONV_CFG;
                seg_word_idx   <= 16'd0;
                seg_word_count <= conv_cfg_len_w;
                predict_done   <= 1'b0;
```

```
371                 model_loaded   <= 1'b0;
372               end
373             end
374
375             if (writedata[1]) begin
376               if (eng_busy || accel_busy) begin
377                 interface_error[1] <= 1'b1;
378               end else if (!model_loaded) begin
379                 interface_error[2] <= 1'b1;
380               end else begin
381                 eng_mode       <= ENG_INFER;
382                 eng_state      <= ST_RD;
383                 seg_word_idx   <= 16'd0;
384                 seg_word_count <= image_len_w;
385                 predict_done   <= 1'b0;
386               end
387             end
388           end
389
390           REG_CONV_CFG_BASE: conv_cfg_base_w  <= writedata[15:0];
391           REG_CONV_CFG_LEN:  conv_cfg_len_w   <= writedata[15:0];
392           REG_CONV_WT_BASE:  conv_wt_base_w   <= writedata[15:0];
393           REG_CONV_WT_LEN:   conv_wt_len_w    <= writedata[15:0];
394           REG_FC_BIAS_BASE:  fc_bias_base_w   <= writedata[15:0];
395           REG_FC_BIAS_LEN:   fc_bias_len_w    <= writedata[15:0];
396           REG_FCW_BASE:      fcw_base_w       <= writedata[15:0];
397           REG_FCW_LEN:       fcw_len_w        <= writedata[15:0];
398           REG_IMAGE_BASE:    image_base_w     <= writedata[15:0];
399           REG_IMAGE_LEN:     image_len_w      <= writedata[15:0];
400           default: begin
401           end
402         endcase
403       end
404
405       case (eng_state)
406         ST_IDLE: begin
407         end
408
409         ST_RD: begin
410           if (!eng_read_in_range) begin
411             interface_error[3] <= 1'b1;
412             eng_mode           <= ENG_IDLE;
413             eng_state          <= ST_IDLE;
414           end else begin
415             eng_state <= ST_WAIT_RD;
416           end
417         end
418
419         ST_WAIT_RD: begin
420           eng_state <= ST_SEND;
421         end
422
423         ST_SEND: begin
424           if (accel_load_ready) begin
```

```
            if (seg_word_idx == (seg_word_count - 16'd1)) begin
              if (eng_mode == ENG_MODEL) begin
                if (model_seg_idx == SEG_FCW) begin
                  model_loaded <= 1'b1;
                  eng_mode     <= ENG_IDLE;
                  eng_state    <= ST_IDLE;
                end else begin
                  model_seg_idx <= model_seg_idx + 2'd1;
                  seg_word_idx  <= 16'd0;
                  eng_state     <= ST_GAP;
                end
              end else begin
                eng_mode    <= ENG_IDLE;
                eng_state   <= ST_IDLE;
                seg_word_idx <= 16'd0;
              end
            end else begin
              seg_word_idx <= seg_word_idx + 16'd1;
              eng_state    <= ST_RD;
            end
          end
        end

        ST_GAP: begin
          seg_word_count <= model_seg_len(model_seg_idx);
          eng_state      <= ST_RD;
        end

        default: begin
          eng_state <= ST_IDLE;
          eng_mode  <= ENG_IDLE;
        end
      endcase
    end
  end

  always @(*) begin
    accel_load_valid = (eng_state == ST_SEND);
    accel_load_data  = mem_rdata_q;
    accel_load_sel   = (eng_mode == ENG_INFER);
    accel_load_last  = (eng_state == ST_SEND) && (seg_word_idx == (seg_word_count - 16'd1));
  end

  // ---------------------------------------------------------------------------
  // Read response
  // ---------------------------------------------------------------------------
  always @(*) begin
    if (host_mem_rd_q)
      readdata = mem_rdata_q;
    else
      readdata = host_rdata_q;
  end

  assign debug_model_loaded   = model_loaded;
```

```
  assign debug_predict_done   = predict_done;
  assign debug_predict_class  = predict_class_latched;
  assign debug_interface_error = interface_error;

endmodule
```

A.7.3 Accelerator Top

Integrates model-load, CNN layer sequencing, SRAM datapath, FC classifier, argmax, and profiling outputs.

code/input/RTL/system_top.v

```
// system_top: ASIC top-level module
//
// Instantiates:
//   TopFSM + LayerRunnerFSM    (control)
//   top_sram_A + top_sram_B    (storage, from teammate)
//   conv_quant_pool          (Conv-Quant-Pool)
//   FC + fc_data_adapter       (FC classifier)
//   wt_prepad_inserter         (3-zero-beat inserter for conv weights)
//
// Chip pins: clk, rst_n, load_data[31:0], load_valid, load_sel,
//            load_last, load_ready, busy, predict_valid, predict_class[1:0]

module system_top #(
  parameter integer OUT_CHANNELS = 10
) (
  input  wire        clk,
  input  wire        rst_n,

  // --- External data port ---
  input  wire        load_sel,
  input  wire        load_valid,
  input  wire [31:0] load_data,
  input  wire        load_last,
  output wire        load_ready,

  // --- Status ---
  output wire        busy,
  output wire        predict_valid,
  output wire [3:0]  predict_class,
  output wire [31:0] profile_l1_cycles,
  output wire [31:0] profile_l2_p0_cycles,
  output wire [31:0] profile_l2_p1_cycles,
  output wire [31:0] profile_l3_p0_cycles,
  output wire [31:0] profile_l3_p1_cycles,
  output wire [31:0] profile_fc_cycles,
  output wire [31:0] profile_argmax_cycles,
  output wire [31:0] profile_total_cycles
```

```
);

  // ============================================================
  // Internal wires
  // ============================================================

  // --- TopFSM <-> LayerRunnerFSM ---
  wire        runner_start, runner_done;
  wire [1:0]  runner_layer_sel;
  wire        runner_pass_id, runner_is_fc;

  // --- TopFSM -> SRAM_A preload control ---
  wire        top_sram_a_start;
  wire [2:0]  top_sram_a_layer_sel;
  wire [1:0]  top_sram_a_data_sel;
  wire        top_sram_a_done;
  wire        preload_wr_valid;
  wire [31:0] preload_wr_data;

  // --- LayerRunnerFSM -> SRAM_A inference control ---
  wire        run_sram_a_start;
  wire [2:0]  run_sram_a_layer_sel;
  wire [1:0]  run_sram_a_data_sel;
  wire        run_sram_a_pass_id;

  // --- LayerRunnerFSM -> SRAM_B control ---
  wire        run_sram_b_start;
  wire [2:0]  run_sram_b_layer_sel;
  wire [1:0]  run_sram_b_data_sel;
  wire        run_sram_b_pass_id;

  // --- SRAM_A muxed control ---
  wire        sram_a_start;
  wire [2:0]  sram_a_layer_sel;
  wire [1:0]  sram_a_data_sel;
  wire        sram_a_pass_id;
  wire        sram_a_done;

  // --- SRAM_A streams ---
  wire        sram_a_data_valid, sram_a_data_last;
  wire [31:0] sram_a_read_data;
  wire        sram_a_data_ready;
  wire        sram_a_pool_ready;

  // --- SRAM_B streams ---
  wire        sram_b_data_valid, sram_b_data_last;
  wire [31:0] sram_b_read_data;
  wire        sram_b_data_ready;
  wire        sram_b_pool_ready;
  wire        sram_b_done;

  // --- conv_quant_pool signals ---
  wire        conv_cfg_ready, conv_wt_ready, conv_in_ready;
  wire signed [31:0] conv_pool_data;
```

```
92  wire        conv_pool_valid, conv_pool_last;
93  wire        conv_cfg_load_done, conv_wt_load_done, conv_pool_frame_done;
94  wire        conv_frame_rearm_done;
95
96  // --- wt_prepad_inserter ---
97  wire        wt_prepad_dn_valid, wt_prepad_dn_last;
98  wire [31:0] wt_prepad_dn_data;
99  wire        wt_prepad_dn_ready;
100 wire        wt_prepad_up_ready;
101
102 // --- conv_data_adapter ---
103 wire        conv_in_adapter_up_ready;
104
105 // --- L1 pixel bypass control ---
106 wire        pixel_stream_active;
107
108 // --- FC signals ---
109 wire        fc_cfg_ready, fc_param_load_done;
110 wire [OUT_CHANNELS*32-1:0] fc_acc_vec;
111 wire [OUT_CHANNELS-1:0]    fc_mul_done_vec;
112 wire        fc_all_done;
113
114 // --- fc_data_adapter signals ---
115 wire        fc_mul_en;
116 wire [OUT_CHANNELS*8-1:0]  fc_pixel_vec;
117 wire [OUT_CHANNELS*8-1:0]  fc_kernel_vec;
118 wire        fc_adapter_wt_ready, fc_adapter_data_ready;
119 wire        fc_done;
120
121 // --- FC weight dedicated 80-bit stream (from SRAM_FCW via top_sram_A) ---
122 wire [OUT_CHANNELS*8-1:0]  fc_wt_read_data;
123 wire                       fc_wt_stream_valid;
124 wire                       fc_wt_stream_last;
125
126 // ============================================================
127 // MUX control: who owns SRAM_A?
128 // ============================================================
129 // TopFSM drives SRAM_A during preload, LayerRunnerFSM during inference.
130 // preload_mode latches on start pulse and HOLDS until the next start,
131 // so layer_sel/data_sel/pass_id remain stable throughout the transaction
132 // (wrapper requires stability until done).
133
134 // preload_mode: latches on start pulse, determines MUX routing.
135 // sram_a_start is delayed 1 cycle so that preload_mode is stable
136 // when the SRAM wrapper samples layer_sel / data_sel / pass_id.
137 reg preload_mode;
138 always @(posedge clk or negedge rst_n) begin
139   if (!rst_n)               preload_mode <= 1'b1;
140   else if (top_sram_a_start) preload_mode <= 1'b1;
141   else if (run_sram_a_start) preload_mode <= 1'b0;
142 end
143
144 reg sram_a_start_d;
145 always @(posedge clk or negedge rst_n) begin
```

```
    if (!rst_n) sram_a_start_d <= 1'b0;
    else        sram_a_start_d <= top_sram_a_start | run_sram_a_start;
  end

  assign sram_a_start     = sram_a_start_d;
  assign sram_a_layer_sel = preload_mode ? top_sram_a_layer_sel : run_sram_a_layer_sel;
  assign sram_a_data_sel  = preload_mode ? top_sram_a_data_sel  : run_sram_a_data_sel;
  assign sram_a_pass_id   = preload_mode ? 1'b0                 : run_sram_a_pass_id;

  // ============================================================
  // SRAM_A write port MUX
  // ============================================================
  // Preload: load_data from external host
  // Inference: pool_data from Conv (L2 writeback to SRAM_A)
  wire [31:0] sram_a_pool_data  = preload_mode ? load_data      : conv_pool_data;
  wire        sram_a_pool_valid = preload_mode ? (preload_wr_valid && load_valid)
                                               : conv_pool_valid;
  wire        sram_a_pool_last  = preload_mode ? load_last       : conv_pool_last;

  // SRAM_A preload_wr_ready = sram_a_pool_ready (when in preload mode)
  wire        preload_wr_ready  = preload_mode ? sram_a_pool_ready : 1'b0;

  // ============================================================
  // SRAM_A read stream routing
  // ============================================================
  // Latched data_sel and is_fc to know where to route read data.
  reg [1:0]  active_data_sel;
  reg        active_is_fc;
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      active_data_sel <= 2'd0;
      active_is_fc    <= 1'b0;
    end else if (run_sram_a_start) begin
      active_data_sel <= run_sram_a_data_sel;
      active_is_fc    <= runner_is_fc;
    end
  end

  localparam [1:0] SEL_CFG = 2'd0, SEL_WT = 2'd1, SEL_DATA = 2'd2;

  // Route SRAM_A read to correct consumer:
  //   CFG + conv -> conv_quant_pool.cfg
  //   CFG + FC   -> FC.cfg
  //   WT  + conv -> wt_prepad_inserter -> conv_quant_pool.wt
  //   WT  + FC   -> fc_data_adapter.wt (no prepad for FC)
  //   DATA + conv (L1/L3) -> conv_quant_pool.in
  //   DATA + FC   -> never (FC data comes from SRAM_B)

  wire route_conv_cfg = !active_is_fc && (active_data_sel == SEL_CFG);
  wire route_conv_wt  = !active_is_fc && (active_data_sel == SEL_WT);
  wire route_conv_in  = !active_is_fc && (active_data_sel == SEL_DATA);
  wire route_fc_cfg   =  active_is_fc && (active_data_sel == SEL_CFG);
  // FC weights no longer flow through the 32-bit SRAM_A read bus; they travel
  // via the dedicated 80-bit fc_wt_read_data port out of top_sram_A.
```

```
200
201   // Conv cfg port
202   wire        conv_cfg_valid = route_conv_cfg ? sram_a_data_valid : 1'b0;
203   wire [31:0] conv_cfg_data  = sram_a_read_data;
204   wire        conv_cfg_last  = route_conv_cfg ? sram_a_data_last  : 1'b0;
205
206   // Conv wt port (through prepad inserter)
207   wire        conv_wt_sram_valid = route_conv_wt ? sram_a_data_valid : 1'b0;
208   wire        conv_wt_sram_last  = route_conv_wt ? sram_a_data_last  : 1'b0;
209
210   // Conv in port (L1/L3 data from SRAM_A; L2 data from SRAM_B handled below)
211   wire        conv_in_from_a_valid = route_conv_in ? sram_a_data_valid : 1'b0;
212   wire        conv_in_from_a_last  = route_conv_in ? sram_a_data_last  : 1'b0;
213
214   // FC cfg port
215   wire        fc_cfg_valid = route_fc_cfg ? sram_a_data_valid : 1'b0;
216   wire [31:0] fc_cfg_data  = sram_a_read_data;
217   wire        fc_cfg_last  = route_fc_cfg ? sram_a_data_last  : 1'b0;
218
219   // SRAM_A data_ready: OR of all active consumers on the shared 32-bit bus.
220   assign sram_a_data_ready = preload_mode ? 1'b0 :
221                              (route_conv_cfg ? conv_cfg_ready            : 1'b0) |
222                              (route_conv_wt  ? wt_prepad_up_ready        : 1'b0) |
223                              (route_conv_in  ? conv_in_adapter_up_ready  : 1'b0) |
224                              (route_fc_cfg   ? fc_cfg_ready              : 1'b0);
225
226   // ============================================================
227   // SRAM_B read stream routing
228   // ============================================================
229   // L2: SRAM_B -> conv in port
230   // FC: SRAM_B -> fc_data_adapter data port
231   // L1/L3 must not see SRAM_B read-side signals while SRAM_B is the pool sink.
232   wire route_conv_in_b = !active_is_fc && (runner_layer_sel == 2'b01);
233   wire route_fc_data_b =  active_is_fc;
234
235   wire        conv_in_from_b_valid = route_conv_in_b ? sram_b_data_valid : 1'b0;
236   wire        conv_in_from_b_last  = route_conv_in_b ? sram_b_data_last  : 1'b0;
237
238   wire        fc_data_valid = route_fc_data_b ? sram_b_data_valid : 1'b0;
239   wire        fc_data_last  = route_fc_data_b ? sram_b_data_last  : 1'b0;
240
241   assign sram_b_data_ready = (route_conv_in_b ? conv_in_adapter_up_ready : 1'b0) |
242                              (route_fc_data_b ? fc_adapter_data_ready    : 1'b0);
243
244   // ============================================================
245   // Conv input MUX: L1 from external load_data (bypass SRAM_A),
246   //                 L2 from SRAM_B, L3 from SRAM_A
247   // ============================================================
248   wire        conv_in_from_ext_valid = pixel_stream_active ? load_valid : 1'b0;
249   wire        conv_in_from_ext_last  = pixel_stream_active ? load_last  : 1'b0;
250
251   wire        conv_in_raw_valid = conv_in_from_ext_valid
252                                 | conv_in_from_a_valid
253                                 | conv_in_from_b_valid;
```

```
254   wire [31:0] conv_in_raw_data  = conv_in_from_ext_valid ? load_data        :
255                                   conv_in_from_a_valid   ? sram_a_read_data :
256                                                            sram_b_read_data;
257   wire        conv_in_raw_last  = conv_in_from_ext_last
258                                 | conv_in_from_a_last
259                                 | conv_in_from_b_last;
260
261   // conv_data_adapter: L1 byte unpack (1 word -> 4 beats), L2/L3 pass-through
262   wire        conv_in_valid;
263   wire [31:0] conv_in_data;
264   wire [3:0]  conv_in_byte_en;
265   wire        conv_in_last;
266
267   // ============================================================
268   // SRAM_B write port: pool output from Conv (L1/L3 writeback)
269   // ============================================================
270   // Only connected during L1/L3 STREAM phase; SRAM_B controller
271   // ignores pool_valid when not in write mode.
272   wire [31:0] sram_b_pool_data  = conv_pool_data;
273   wire        sram_b_pool_valid = conv_pool_valid;
274   wire        sram_b_pool_last  = conv_pool_last;
275
276   // Conv pool_ready: from whichever SRAM is the write sink
277   // L1 (00) / L3 (10): SRAM_B is sink -> pool_ready from SRAM_B
278   // L2 (01): SRAM_A is sink -> pool_ready from SRAM_A (when not in preload)
279   wire        pool_sink_is_b  = (runner_layer_sel != 2'b01);  // L1/L3 -> B; L2 -> A
280   wire        conv_pool_ready = preload_mode ? 1'b0 :
281                                 (pool_sink_is_b ? sram_b_pool_ready
282                                                 : sram_a_pool_ready);
283
284   // ============================================================
285   // Module instantiations
286   // ============================================================
287
288   // --- TopFSM ---
289   top_fsm #(.ACC_W(32), .OUT_CHANNELS(OUT_CHANNELS)) u_top_fsm (
290     .clk(clk), .rst_n(rst_n),
291     .load_sel(load_sel), .load_valid(load_valid),
292     .load_data(load_data), .load_last(load_last),
293     .load_ready(load_ready),
294     .busy(busy), .predict_valid(predict_valid), .predict_class(predict_class),
295     .profile_l1_cycles(profile_l1_cycles),
296     .profile_l2_p0_cycles(profile_l2_p0_cycles),
297     .profile_l2_p1_cycles(profile_l2_p1_cycles),
298     .profile_l3_p0_cycles(profile_l3_p0_cycles),
299     .profile_l3_p1_cycles(profile_l3_p1_cycles),
300     .profile_fc_cycles(profile_fc_cycles),
301     .profile_argmax_cycles(profile_argmax_cycles),
302     .profile_total_cycles(profile_total_cycles),
303     .runner_start(runner_start),
304     .runner_layer_sel(runner_layer_sel),
305     .runner_pass_id(runner_pass_id),
306     .runner_is_fc(runner_is_fc),
307     .runner_done(runner_done),
```

```
    .sram_a_start(top_sram_a_start),
    .sram_a_layer_sel(top_sram_a_layer_sel),
    .sram_a_data_sel(top_sram_a_data_sel),
    .sram_a_done(sram_a_done),
    .preload_wr_valid(preload_wr_valid),
    .preload_wr_data(preload_wr_data),
    .preload_wr_ready(preload_wr_ready),
    .pixel_stream_active(pixel_stream_active),
    .conv_adapter_up_ready(conv_in_adapter_up_ready),
    .conv_frame_rearm(conv_frame_rearm_done),
    .fc_acc_vec(fc_acc_vec)
  );

  // --- LayerRunnerFSM ---
  layer_runner_fsm u_runner (
    .clk(clk), .rst_n(rst_n),
    .start(runner_start),
    .layer_sel(runner_layer_sel),
    .pass_id(runner_pass_id),
    .is_fc(runner_is_fc),
    .busy(), .done(runner_done),
    .sram_a_start(run_sram_a_start),
    .sram_a_layer_sel(run_sram_a_layer_sel),
    .sram_a_data_sel(run_sram_a_data_sel),
    .sram_a_pass_id(run_sram_a_pass_id),
    .sram_a_done(sram_a_done),
    .sram_b_start(run_sram_b_start),
    .sram_b_layer_sel(run_sram_b_layer_sel),
    .sram_b_data_sel(run_sram_b_data_sel),
    .sram_b_pass_id(run_sram_b_pass_id),
    .sram_b_done(sram_b_done),
    .cfg_load_done(runner_is_fc ? fc_param_load_done : conv_cfg_load_done),
    .wt_load_done(conv_wt_load_done),
    .pool_frame_done(conv_pool_frame_done),
    .conv_frame_rearm(conv_frame_rearm_done),
    .fc_done(fc_done)
  );

  // --- SRAM_A (shared 32-bit pool + SRAM_FCW 80-bit peer for FC weights) ---
  top_sram_A u_sram_a (
    .clk(clk), .rst_n(rst_n),
    .layer_sel(sram_a_layer_sel),
    .data_sel(sram_a_data_sel),
    .pass_id(sram_a_pass_id),
    .start(sram_a_start),
    .busy(), .done(sram_a_done),
    .data_ready(sram_a_data_ready),
    .read_data(sram_a_read_data),
    .data_valid(sram_a_data_valid),
    .data_last(sram_a_data_last),
    .pool_ready(sram_a_pool_ready),
    .pool_data(sram_a_pool_data),
    .pool_valid(sram_a_pool_valid),
    .pool_last(sram_a_pool_last),
```

```
    .fc_wt_ready(fc_adapter_wt_ready),
    .fc_wt_read_data(fc_wt_read_data),
    .fc_wt_valid(fc_wt_stream_valid),
    .fc_wt_last(fc_wt_stream_last)
  );

  // --- SRAM_B (from teammate) ---
  top_sram_B u_sram_b (
    .clk(clk), .rst_n(rst_n),
    .layer_sel(run_sram_b_layer_sel),
    .data_sel(run_sram_b_data_sel),
    .pass_id(run_sram_b_pass_id),
    .start(run_sram_b_start),
    .busy(), .done(sram_b_done),
    .data_ready(sram_b_data_ready),
    .read_data(sram_b_read_data),
    .data_valid(sram_b_data_valid),
    .data_last(sram_b_data_last),
    .pool_ready(sram_b_pool_ready),
    .pool_data(sram_b_pool_data),
    .pool_valid(sram_b_pool_valid),
    .pool_last(sram_b_pool_last)
  );

  // --- WT Prepad Inserter (between SRAM_A read and Conv wt port) ---
  wt_prepad_inserter u_wt_prepad (
    .clk(clk), .rst_n(rst_n),
    .is_wt_read(route_conv_wt),
    .up_valid(conv_wt_sram_valid),
    .up_data(sram_a_read_data),
    .up_last(conv_wt_sram_last),
    .up_ready(wt_prepad_up_ready),
    .dn_valid(wt_prepad_dn_valid),
    .dn_data(wt_prepad_dn_data),
    .dn_last(wt_prepad_dn_last),
    .dn_ready(wt_prepad_dn_ready)
  );

  // --- Conv Data Adapter (L1 byte unpack, L2/L3 pass-through) ---
  conv_data_adapter u_conv_data_adapter (
    .clk(clk), .rst_n(rst_n),
    .layer_sel(runner_layer_sel),
    .up_valid(conv_in_raw_valid),
    .up_data(conv_in_raw_data),
    .up_last(conv_in_raw_last),
    .up_ready(conv_in_adapter_up_ready),
    .dn_valid(conv_in_valid),
    .dn_data(conv_in_data),
    .dn_byte_en(conv_in_byte_en),
    .dn_last(conv_in_last),
    .dn_ready(conv_in_ready)
  );

  // --- Conv-Quant-Pool ---
```

```
  conv_quant_pool u_conv (
    .layer_sel(runner_layer_sel),
    .clk(clk), .rst_n(rst_n),
    // Image stream (from conv_data_adapter)
    .in_valid(conv_in_valid),
    .in_data(conv_in_data),
    .in_byte_en(conv_in_byte_en),
    .in_last(conv_in_last),
    .in_ready(conv_in_ready),
    // Weight stream (from prepad inserter)
    .wt_valid(wt_prepad_dn_valid),
    .wt_ready(wt_prepad_dn_ready),
    .wt_last(wt_prepad_dn_last),
    .wt_data(wt_prepad_dn_data),
    // Cfg stream
    .cfg_valid(conv_cfg_valid),
    .cfg_ready(conv_cfg_ready),
    .cfg_data(conv_cfg_data),
    .cfg_last(conv_cfg_last),
    // Pool output
    .pool_ready(conv_pool_ready),
    .pool_data(conv_pool_data),
    .pool_valid(conv_pool_valid),
    .pool_last(conv_pool_last),
    // Control observation
    .cfg_load_done(conv_cfg_load_done),
    .wt_load_done(conv_wt_load_done),
    .pool_frame_done(conv_pool_frame_done),
    .conv_frame_rearm_out(conv_frame_rearm_done)
  );

  // --- FC Data Adapter ---
  fc_data_adapter #(
    .OUT_CHANNELS(OUT_CHANNELS),
    .PIX_W(8), .KER_W(8)
  ) u_fc_adapter (
    .clk(clk), .rst_n(rst_n),
    // 80-bit weight stream from SRAM_FCW (via top_sram_A FCW peer)
    .wt_valid(fc_wt_stream_valid),
    .wt_data(fc_wt_read_data),
    .wt_last(fc_wt_stream_last),
    .wt_ready(fc_adapter_wt_ready),
    // Data stream from SRAM_B (packed pixels, 32b)
    .data_valid(fc_data_valid),
    .data_data(sram_b_read_data),
    .data_last(fc_data_last),
    .data_ready(fc_adapter_data_ready),
    // To FC
    .mul_en(fc_mul_en),
    .pixel_vec(fc_pixel_vec),
    .kernel_vec(fc_kernel_vec),
    .all_done(fc_all_done),
    .fc_done(fc_done)
  );
```

```
470
471   // --- FC ---
472   FC #(
473     .PIX_W(8), .KER_W(8), .K(288), .ACC_W(32),
474     .OUT_CHANNELS(OUT_CHANNELS)
475   ) u_fc (
476     .clk(clk), .rst_n(rst_n),
477     .cfg_valid(fc_cfg_valid),
478     .cfg_ready(fc_cfg_ready),
479     .cfg_data(fc_cfg_data),
480     .cfg_last(fc_cfg_last),
481     .param_load_done(fc_param_load_done),
482     .mul_en(fc_mul_en),
483     .pixel_vec(fc_pixel_vec),
484     .kernel_vec(fc_kernel_vec),
485     .acc_vec(fc_acc_vec),
486     .mul_done_vec(fc_mul_done_vec),
487     .all_done(fc_all_done)
488   );
489
490 endmodule
```

A.7.4 Network-Level FSM

Controls the high-level model-load and inference schedule across convolution, FC, and argmax phases.

code/input/RTL/fsm/top_fsm.v

```
1  // TopFSM: network-level sequencer.
2  //
3  // MODEL_LOAD (one-time): host sends cfg -> weight -> image via load_data bus
4  // INFER (repeatable): L1 -> L2p0 -> L2p1 -> L3p0 -> L3p1 -> FC -> ARGMAX -> READY
5  //
6  // Preload follows SRAM_CONNECTION_NOTES layer0 convention:
7  //   PL_CFG   -> sram_a_start(layer0, CFG_READ)
8  //   PL_WT    -> sram_a_start(layer0, WT_READ)
9  //   PL_PIXEL -> sram_a_start(layer0, DATA_READ)
10 // Each segment ends when host asserts load_last on the valid&&ready beat.
11
12 module top_fsm #(
13   parameter integer ACC_W        = 32,  // FC accumulator width (matches SRAM/cfg 32-bit path)
14   parameter integer OUT_CHANNELS = 10   // FC output channels (0-9 gesture classes)
15 )(
16   input  wire        clk,
17   input  wire        rst_n,
18
19   // --- External data port (32-bit parallel, valid/ready) ---
20   input  wire        load_sel,       // 0=model, 1=image
21   input  wire        load_valid,
```

```
    input  wire [31:0] load_data,
    input  wire        load_last,
    output wire        load_ready,

    // --- External status ---
    output wire        busy,
    output reg         predict_valid,
    output reg  [3:0]  predict_class,  // 10-class gesture id (0-9)
    output reg  [31:0] profile_l1_cycles,
    output reg  [31:0] profile_l2_p0_cycles,
    output reg  [31:0] profile_l2_p1_cycles,
    output reg  [31:0] profile_l3_p0_cycles,
    output reg  [31:0] profile_l3_p1_cycles,
    output reg  [31:0] profile_fc_cycles,
    output reg  [31:0] profile_argmax_cycles,
    output reg  [31:0] profile_total_cycles,

    // --- LayerRunnerFSM control ---
    output reg         runner_start,
    output reg  [1:0]  runner_layer_sel,
    output reg         runner_pass_id,
    output reg         runner_is_fc,
    input  wire        runner_done,

    // --- SRAM_A preload control (active during MODEL_LOAD / LOAD_IMAGE) ---
    // During inference, SRAM_A is driven by LayerRunnerFSM via system_top MUX.
    output reg         sram_a_start,
    output reg  [2:0]  sram_a_layer_sel,
    output reg  [1:0]  sram_a_data_sel,   // 0=CFG, 1=WT, 2=DATA
    input  wire        sram_a_done,

    // --- Preload write data (directly forwarded to SRAM_A wrapper write port) ---
    output wire        preload_wr_valid,
    output wire [31:0] preload_wr_data,
    input  wire        preload_wr_ready,  // backpressure from SRAM_A wrapper

    // --- L1 pixel bypass: load_data streams directly to conv ---
    output reg         pixel_stream_active,  // high during L1 pixel streaming
    input  wire        conv_adapter_up_ready, // backpressure from conv_data_adapter
    input  wire        conv_frame_rearm,      // from Conv1_top: all pixels processed

    // --- FC accumulator results (packed vector, LSB = channel 0) ---
    input  wire [OUT_CHANNELS*ACC_W-1:0] fc_acc_vec
);

    // ------------------------------------------------------------
    // State encoding
    // ------------------------------------------------------------
    localparam [4:0] ST_IDLE       = 5'd0,
                     ST_PL_CFG     = 5'd1,   // MODEL_LOAD sub-state: conv cfg
                     ST_PL_CFG_W   = 5'd2,
                     ST_PL_WT      = 5'd3,   // MODEL_LOAD sub-state: conv weight
                     ST_PL_WT_W    = 5'd4,
                     ST_PL_PIXEL   = 5'd5,
```

```
                  ST_PL_PIXEL_W  = 5'd6,
                  ST_READY       = 5'd7,
                  ST_L1          = 5'd8,
                  ST_L2_P0       = 5'd9,
                  ST_L2_P1       = 5'd10,
                  ST_L3_P0       = 5'd11,
                  ST_L3_P1       = 5'd12,
                  ST_FC          = 5'd13,
                  ST_ARGMAX      = 5'd14,
                  ST_PL_FC_CFG   = 5'd15,  // MODEL_LOAD: 10 FC bias words into SRAM_A@0x111
                  ST_PL_FC_CFG_W = 5'd16,
                  ST_PL_FCW      = 5'd17,  // MODEL_LOAD: 864 host words into SRAM_FCW packer
                  ST_PL_FCW_W    = 5'd18;

// SRAM_A layer0 preload encodings
localparam [2:0] LSEL_LAYER0 = 3'b000;
localparam [1:0] SEL_CFG  = 2'd0,
                 SEL_WT   = 2'd1,
                 SEL_DATA = 2'd2,
                 SEL_FCW  = 2'd3;

reg [4:0] state;
reg       preload_active;   // data forwarding enabled
reg       load_last_seen;   // current segment's load_last received
reg       model_loaded;     // model has been loaded at least once
reg       preload_sram_done; // latched sram_a_done during preload wait
reg       runner_started;    // prevents repeated runner_start pulses
reg       runner_prepared;   // gives downstream logic one cycle to latch layer config

// ------------------------------------------------------------
// Preload data forwarding / L1 pixel bypass
// ------------------------------------------------------------
// CFG/WT preload: forward load_data to SRAM_A write port
// L1 pixel streaming: load_data bypasses SRAM_A, goes directly to conv
assign preload_wr_valid = preload_active && load_valid;
assign preload_wr_data  = load_data;
assign load_ready       = (preload_active      && preload_wr_ready)
                        | (pixel_stream_active && conv_adapter_up_ready);

assign busy = (state != ST_IDLE) && (state != ST_READY);

// ------------------------------------------------------------
// Iterative argmax state
// ------------------------------------------------------------
reg [3:0]              argmax_scan_idx;
reg [3:0]              argmax_best_idx;
reg signed [ACC_W-1:0] argmax_best_val;
wire signed [ACC_W-1:0] argmax_scan_val =
    $signed(fc_acc_vec[argmax_scan_idx*ACC_W +: ACC_W]);

// ------------------------------------------------------------
// FSM
// ------------------------------------------------------------
always @(posedge clk or negedge rst_n) begin
```

```
130     if (!rst_n) begin
131       state            <= ST_IDLE;
132       predict_valid    <= 1'b0;
133       predict_class    <= 4'd0;
134       runner_start     <= 1'b0;
135       runner_layer_sel <= 2'b00;
136       runner_pass_id   <= 1'b0;
137       runner_is_fc     <= 1'b0;
138       profile_l1_cycles <= 32'd0;
139       profile_l2_p0_cycles <= 32'd0;
140       profile_l2_p1_cycles <= 32'd0;
141       profile_l3_p0_cycles <= 32'd0;
142       profile_l3_p1_cycles <= 32'd0;
143       profile_fc_cycles <= 32'd0;
144       profile_argmax_cycles <= 32'd0;
145       profile_total_cycles <= 32'd0;
146       sram_a_start     <= 1'b0;
147       sram_a_layer_sel <= 3'd0;
148       sram_a_data_sel   <= 2'b00;
149       preload_active   <= 1'b0;
150       load_last_seen   <= 1'b0;
151       model_loaded     <= 1'b0;
152       preload_sram_done <= 1'b0;
153       runner_started   <= 1'b0;
154       runner_prepared  <= 1'b0;
155       pixel_stream_active <= 1'b0;
156       argmax_scan_idx  <= 4'd0;
157       argmax_best_idx  <= 4'd0;
158       argmax_best_val  <= {ACC_W{1'b0}};
159     end else begin
160       // Default: clear one-cycle pulses
161       sram_a_start <= 1'b0;
162       runner_start <= 1'b0;
163       predict_valid <= 1'b0;
164
165       // Track load_last during preload (fires on valid && load_last)
166       if (preload_active && load_valid && load_last)
167         load_last_seen <= 1'b1;
168
169       // Latch sram_a_done during preload wait (done is a 1-cycle pulse)
170       if (preload_active && sram_a_done)
171         preload_sram_done <= 1'b1;
172
173       if (state != ST_IDLE && state != ST_READY &&
174           state != ST_PL_CFG && state != ST_PL_CFG_W &&
175           state != ST_PL_WT && state != ST_PL_WT_W &&
176           state != ST_PL_PIXEL && state != ST_PL_PIXEL_W &&
177           state != ST_PL_FC_CFG && state != ST_PL_FC_CFG_W &&
178           state != ST_PL_FCW && state != ST_PL_FCW_W) begin
179         profile_total_cycles <= profile_total_cycles + 32'd1;
180       end
181
182       case (state)
183         ST_L1:     profile_l1_cycles <= profile_l1_cycles + 32'd1;
```

```
        ST_L2_P0:  profile_l2_p0_cycles <= profile_l2_p0_cycles + 32'd1;
        ST_L2_P1:  profile_l2_p1_cycles <= profile_l2_p1_cycles + 32'd1;
        ST_L3_P0:  profile_l3_p0_cycles <= profile_l3_p0_cycles + 32'd1;
        ST_L3_P1:  profile_l3_p1_cycles <= profile_l3_p1_cycles + 32'd1;
        ST_FC:     profile_fc_cycles <= profile_fc_cycles + 32'd1;
        ST_ARGMAX: profile_argmax_cycles <= profile_argmax_cycles + 32'd1;
        default: begin
        end
      endcase

      case (state)
        // ------------------------------------------------------------
        ST_IDLE: begin
          preload_active <= 1'b0;
          if (load_valid && load_sel == 1'b0) begin
            // Host wants to load model -> start MODEL_LOAD
            state <= ST_PL_CFG;
          end
        end

        // ------------------------------------------------------------
        // MODEL_LOAD: 3 sub-states (PL_CFG -> PL_WT -> PL_PIXEL)
        // ------------------------------------------------------------
        ST_PL_CFG: begin
          sram_a_start      <= 1'b1;
          sram_a_layer_sel  <= LSEL_LAYER0;
          sram_a_data_sel   <= SEL_CFG;
          preload_active    <= 1'b1;
          load_last_seen    <= 1'b0;
          preload_sram_done <= 1'b0;
          state             <= ST_PL_CFG_W;
        end

        ST_PL_CFG_W: begin
          if (load_last_seen && (sram_a_done || preload_sram_done)) begin
            preload_active <= 1'b0;
            if (!load_valid) begin
              load_last_seen <= 1'b0;
              state          <= ST_PL_WT;
            end
          end
        end

        ST_PL_WT: begin
          sram_a_start      <= 1'b1;
          sram_a_layer_sel  <= LSEL_LAYER0;
          sram_a_data_sel   <= SEL_WT;
          preload_active    <= 1'b1;
          load_last_seen    <= 1'b0;
          preload_sram_done <= 1'b0;
          state             <= ST_PL_WT_W;
        end

        ST_PL_WT_W: begin
```

```
238           if (load_last_seen && (sram_a_done || preload_sram_done)) begin
239             preload_active <= 1'b0;
240             if (!load_valid) begin
241               load_last_seen <= 1'b0;
242               // Continue preload: FC bias words, then FCW weight packed stream.
243               state          <= ST_PL_FC_CFG;
244             end
245           end
246         end
247
248         // FC bias: 10 x 32-bit words written to SRAM_A at FC_CFG_BASE (0x111).
249         // Uses LAYER_FC + SEL_CFG; controller enables both read/write modes,
250         // and preload_mode in system_top gates the write path while blocking
251         // conv_pool_valid so only the host write actually fires.
252         ST_PL_FC_CFG: begin
253           sram_a_start      <= 1'b1;
254           sram_a_layer_sel  <= 3'd4;          // LAYER_FC
255           sram_a_data_sel   <= SEL_CFG;
256           preload_active    <= 1'b1;
257           load_last_seen    <= 1'b0;
258           preload_sram_done <= 1'b0;
259           state             <= ST_PL_FC_CFG_W;
260         end
261
262         ST_PL_FC_CFG_W: begin
263           if (load_last_seen && (sram_a_done || preload_sram_done)) begin
264             preload_active <= 1'b0;
265             if (!load_valid) begin
266               load_last_seen <= 1'b0;
267               state          <= ST_PL_FCW;
268             end
269           end
270         end
271
272         // FC weight: 864 x 32-bit words packed by fcw_preload_packer into
273         // 288 x 80-bit words inside SRAM_FCW.
274         ST_PL_FCW: begin
275           sram_a_start      <= 1'b1;
276           sram_a_layer_sel  <= LSEL_LAYER0;   // LAYER_PRELOAD
277           sram_a_data_sel   <= SEL_FCW;
278           preload_active    <= 1'b1;
279           load_last_seen    <= 1'b0;
280           preload_sram_done <= 1'b0;
281           state             <= ST_PL_FCW_W;
282         end
283
284         ST_PL_FCW_W: begin
285           if (load_last_seen && (sram_a_done || preload_sram_done)) begin
286             preload_active <= 1'b0;
287             if (!load_valid) begin
288               load_last_seen <= 1'b0;
289               // MODEL_LOAD complete.
290               model_loaded   <= 1'b1;
291               state          <= ST_READY;
```

```
292             end
293           end
294         end
295
296         // ST_PL_PIXEL / ST_PL_PIXEL_W: removed - pixels now stream directly
297         // to conv_data_adapter via system_top during ST_L1.
298
299         // ----------------------------------------------------------
300         ST_READY: begin
301           preload_active <= 1'b0;
302           if (load_valid && !load_last && load_sel == 1'b1) begin
303             profile_l1_cycles <= 32'd0;
304             profile_l2_p0_cycles <= 32'd0;
305             profile_l2_p1_cycles <= 32'd0;
306             profile_l3_p0_cycles <= 32'd0;
307             profile_l3_p1_cycles <= 32'd0;
308             profile_fc_cycles <= 32'd0;
309             profile_argmax_cycles <= 32'd0;
310             profile_total_cycles <= 32'd0;
311             // Host wants to send image -> enter L1 (pixels stream directly)
312             state <= ST_L1;
313           end else if (load_valid && !load_last && load_sel == 1'b0) begin
314             // Host wants to reload model
315             model_loaded <= 1'b0;
316             state <= ST_PL_CFG;
317           end
318         end
319
320         // ----------------------------------------------------------
321         // Inference: L1 -> L2p0 -> L2p1 -> L3p0 -> L3p1 -> FC -> ARGMAX
322         // Each state: pulse runner_start once, then wait for runner_done.
323         // runner_started prevents repeated start pulses while waiting.
324         // ----------------------------------------------------------
325         ST_L1: begin
326           if (!runner_prepared) begin
327             runner_layer_sel <= 2'b00;
328             runner_pass_id   <= 1'b0;
329             runner_is_fc     <= 1'b0;
330             runner_prepared  <= 1'b1;
331           end else if (!runner_started) begin
332             runner_start     <= 1'b1;
333             runner_started   <= 1'b1;
334             pixel_stream_active <= 1'b1;  // route load_data directly to conv
335           end
336           if (runner_done) begin
337             state <= ST_L2_P0;
338             runner_started <= 1'b0;
339             runner_prepared <= 1'b0;
340             pixel_stream_active <= 1'b0;
341           end
342         end
343
344         ST_L2_P0: begin
345           if (!runner_prepared) begin
```

```
            runner_layer_sel <= 2'b01;
            runner_pass_id   <= 1'b0;
            runner_is_fc     <= 1'b0;
            runner_prepared  <= 1'b1;
          end else if (!runner_started) begin
            runner_start     <= 1'b1;
            runner_started   <= 1'b1;
          end
          if (runner_done) begin state <= ST_L2_P1; runner_started <= 1'b0; runner_prepared <= 1'b0; end
        end

        ST_L2_P1: begin
          if (!runner_prepared) begin
            runner_layer_sel <= 2'b01;
            runner_pass_id   <= 1'b1;
            runner_is_fc     <= 1'b0;
            runner_prepared  <= 1'b1;
          end else if (!runner_started) begin
            runner_start     <= 1'b1;
            runner_started   <= 1'b1;
          end
          if (runner_done) begin state <= ST_L3_P0; runner_started <= 1'b0; runner_prepared <= 1'b0; end
        end

        ST_L3_P0: begin
          if (!runner_prepared) begin
            runner_layer_sel <= 2'b10;
            runner_pass_id   <= 1'b0;
            runner_is_fc     <= 1'b0;
            runner_prepared  <= 1'b1;
          end else if (!runner_started) begin
            runner_start     <= 1'b1;
            runner_started   <= 1'b1;
          end
          if (runner_done) begin state <= ST_L3_P1; runner_started <= 1'b0; runner_prepared <= 1'b0; end
        end

        ST_L3_P1: begin
          if (!runner_prepared) begin
            runner_layer_sel <= 2'b10;
            runner_pass_id   <= 1'b1;
            runner_is_fc     <= 1'b0;
            runner_prepared  <= 1'b1;
          end else if (!runner_started) begin
            runner_start     <= 1'b1;
            runner_started   <= 1'b1;
          end
          if (runner_done) begin state <= ST_FC; runner_started <= 1'b0; runner_prepared <= 1'b0; end
        end

        ST_FC: begin
          if (!runner_prepared) begin
            runner_layer_sel <= 2'b00;
            runner_pass_id   <= 1'b0;
```

```
            runner_is_fc     <= 1'b1;
            runner_prepared  <= 1'b1;
          end else if (!runner_started) begin
            runner_start     <= 1'b1;
            runner_started   <= 1'b1;
          end
          if (runner_done) begin
            state           <= ST_ARGMAX;
            runner_started  <= 1'b0;
            runner_prepared <= 1'b0;
            argmax_scan_idx <= 4'd1;
            argmax_best_idx <= 4'd0;
            argmax_best_val <= $signed(fc_acc_vec[0 +: ACC_W]);
          end
        end

        // ------------------------------------------------------
        ST_ARGMAX: begin
          // Compare one class per cycle. This removes the long 10-way
          // comparator chain from the fabric critical path.
          if (argmax_scan_val > argmax_best_val) begin
            argmax_best_val <= argmax_scan_val;
            argmax_best_idx <= argmax_scan_idx;
          end

          if (argmax_scan_idx == OUT_CHANNELS - 1) begin
            predict_class <= (argmax_scan_val > argmax_best_val)
                             ? argmax_scan_idx
                             : argmax_best_idx;
            predict_valid <= 1'b1;
            state         <= ST_READY;
          end else begin
            argmax_scan_idx <= argmax_scan_idx + 4'd1;
          end
        end

        default: state <= ST_IDLE;
      endcase
    end
  end

endmodule
```

A.7.5 Layer Runner FSM

Sequences per-layer configuration, weights, input data, output writes, and completion for the CNN layers.

code/input/RTL/fsm/layer_runner_fsm.v

```
// LayerRunnerFSM: single-layer / single-pass transaction controller.
//
// Conv mode (is_fc=0):
//   IDLE -> LOAD_CFG -> WAIT_CFG -> LOAD_WT -> WAIT_WT -> STREAM -> WAIT_DONE -> DONE_PULSE
//
// FC mode (is_fc=1):
//   IDLE -> LOAD_CFG -> WAIT_CFG -> STREAM -> WAIT_DONE -> DONE_PULSE
//
// SRAM_A interface uses data_sel (not op_type):
//   data_sel = 0 (CFG), 1 (WT), 2 (DATA)
//   Read/write direction derived by wrapper from layer_sel + data_sel.
//
// All state transitions driven by SRAM wrapper done signals.

module layer_runner_fsm (
  input  wire        clk,
  input  wire        rst_n,

  // --- TopFSM control ---
  input  wire        start,
  input  wire [1:0]  layer_sel,      // 00=L1, 01=L2, 10=L3
  input  wire        pass_id,
  input  wire        is_fc,
  output wire        busy,
  output reg         done,

  // --- SRAM_A wrapper control ---
  output reg         sram_a_start,
  output reg  [2:0]  sram_a_layer_sel,
  output reg  [1:0]  sram_a_data_sel,    // 0=CFG, 1=WT, 2=DATA
  output reg         sram_a_pass_id,
  input  wire        sram_a_done,

  // --- SRAM_B wrapper control ---
  output reg         sram_b_start,
  output reg  [2:0]  sram_b_layer_sel,
  output reg  [1:0]  sram_b_data_sel,
  output reg         sram_b_pass_id,
  input  wire        sram_b_done,

  // --- Conv path observation (auxiliary, NOT for transitions) ---
  input  wire        cfg_load_done,
  input  wire        wt_load_done,
  input  wire        pool_frame_done,
  input  wire        conv_frame_rearm,

  // --- FC path feedback ---
  input  wire        fc_done
);

  localparam [2:0] ST_IDLE      = 3'd0,
                   ST_LOAD_CFG  = 3'd1,
```

```
                  ST_WAIT_CFG  = 3'd2,
                  ST_LOAD_WT   = 3'd3,
                  ST_WAIT_WT   = 3'd4,
                  ST_STREAM    = 3'd5,
                  ST_WAIT_DONE = 3'd6,
                  ST_DONE      = 3'd7;

  localparam [1:0] SEL_CFG  = 2'd0,
                  SEL_WT   = 2'd1,
                  SEL_DATA = 2'd2;

  localparam [2:0] LSEL_L1 = 3'd1,
                  LSEL_L2 = 3'd2,
                  LSEL_L3 = 3'd3,
                  LSEL_FC = 3'd4;

  reg [2:0] state;

  // Debug-visible mirrors kept for TB/wave compatibility. They are no longer
  // used to drive transactions.
  reg [1:0] layer_sel_r;
  reg       pass_id_r;
  reg       is_fc_r;

  // TopFSM keeps layer_sel / pass_id / is_fc stable while a layer is running.
  // Use the live inputs for transaction control so the runner cannot start the
  // next layer with stale latched parameters.

  function [2:0] to_lsel;
    input [1:0] ls;
    input       fc;
    begin
      if (fc) to_lsel = LSEL_FC;
      else case (ls)
        2'b00:   to_lsel = LSEL_L1;
        2'b01:   to_lsel = LSEL_L2;
        2'b10:   to_lsel = LSEL_L3;
        default: to_lsel = LSEL_L1;
      endcase
    end
  endfunction

  // WAIT_DONE: accumulate done signals from both wrappers + pool
  reg sram_a_done_seen, sram_b_done_seen;
  reg pool_done_seen;
  reg frame_rearm_seen;

  assign busy = (state != ST_IDLE);

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      state            <= ST_IDLE;
      done             <= 1'b0;
      sram_a_start     <= 1'b0;
```

```
      sram_a_layer_sel <= 3'd0;
      sram_a_data_sel  <= 2'd0;
      sram_a_pass_id   <= 1'b0;
      sram_b_start     <= 1'b0;
      sram_b_layer_sel <= 3'd0;
      sram_b_data_sel  <= 2'd0;
      sram_b_pass_id   <= 1'b0;
      layer_sel_r      <= 2'b00;
      pass_id_r        <= 1'b0;
      is_fc_r          <= 1'b0;
      sram_a_done_seen <= 1'b0;
      sram_b_done_seen <= 1'b0;
      pool_done_seen   <= 1'b0;
      frame_rearm_seen <= 1'b0;
    end else begin
      // Clear one-cycle pulses
      sram_a_start <= 1'b0;
      sram_b_start <= 1'b0;
      done         <= 1'b0;

      case (state)
        // ------------------------------------------------------------
        ST_IDLE: begin
          if (start) begin
            layer_sel_r <= layer_sel;
            pass_id_r   <= pass_id;
            is_fc_r     <= is_fc;
            state       <= ST_LOAD_CFG;
          end
        end

        // ------------------------------------------------------------
        ST_LOAD_CFG: begin
          sram_a_start     <= 1'b1;
          sram_a_layer_sel <= to_lsel(layer_sel, is_fc);
          sram_a_data_sel  <= SEL_CFG;
          sram_a_pass_id   <= pass_id;
          sram_a_done_seen <= 1'b0;
          state            <= ST_WAIT_CFG;
        end

        ST_WAIT_CFG: begin
          if (sram_a_done) sram_a_done_seen <= 1'b1;
          if (is_fc) begin
            if ((sram_a_done_seen || sram_a_done) && cfg_load_done) begin
              sram_a_done_seen <= 1'b0;
              state <= ST_STREAM;   // FC skips LOAD_WT
            end
          end else begin
            if (sram_a_done)
              state <= ST_LOAD_WT;
          end
        end

```

```
161         // ------------------------------------------------------
162         ST_LOAD_WT: begin
163           sram_a_start     <= 1'b1;
164           sram_a_layer_sel <= to_lsel(layer_sel, 1'b0);
165           sram_a_data_sel  <= SEL_WT;
166           sram_a_pass_id   <= pass_id;
167           state            <= ST_WAIT_WT;
168         end
169
170         ST_WAIT_WT: begin
171           if (sram_a_done)
172             state <= ST_STREAM;
173         end
174
175         // ------------------------------------------------------
176         ST_STREAM: begin
177           sram_a_done_seen <= 1'b0;
178           sram_b_done_seen <= 1'b0;
179           pool_done_seen   <= 1'b0;
180           frame_rearm_seen <= 1'b0;
181
182           if (is_fc) begin
183             // FC: SRAM_A reads interleaved weight, SRAM_B reads packed data
184             sram_a_start     <= 1'b1;
185             sram_a_layer_sel <= LSEL_FC;
186             sram_a_data_sel  <= SEL_WT;
187             sram_a_pass_id   <= 1'b0;
188
189             sram_b_start     <= 1'b1;
190             sram_b_layer_sel <= LSEL_FC;
191             sram_b_data_sel  <= SEL_DATA;
192             sram_b_pass_id   <= 1'b0;
193           end else begin
194             // Conv: start DATA transaction on both SRAMs
195             // L1: pixels stream directly from external load_data (no SRAM_A read)
196             // L2/L3: read DATA from SRAM_A (or SRAM_B)
197             if (layer_sel != 2'b00) begin
198               sram_a_start     <= 1'b1;
199               sram_a_layer_sel <= to_lsel(layer_sel, 1'b0);
200               sram_a_data_sel  <= SEL_DATA;
201               sram_a_pass_id   <= pass_id;
202             end
203
204             sram_b_start     <= 1'b1;
205             sram_b_layer_sel <= to_lsel(layer_sel, 1'b0);
206             sram_b_data_sel  <= SEL_DATA;
207             sram_b_pass_id   <= pass_id;
208           end
209
210           state <= ST_WAIT_DONE;
211         end
212
213         // ------------------------------------------------------
214         ST_WAIT_DONE: begin
```

```
          if (sram_a_done) sram_a_done_seen <= 1'b1;
          if (sram_b_done) sram_b_done_seen <= 1'b1;
          if (pool_frame_done) pool_done_seen <= 1'b1;
          if (conv_frame_rearm) frame_rearm_seen <= 1'b1;

          // Both wrapper dones must arrive
          if (is_fc) begin
            if (fc_done &&
                (sram_a_done_seen || sram_a_done) &&
                (sram_b_done_seen || sram_b_done))
              state <= ST_DONE;
          end else if (layer_sel == 2'b00) begin
            // L1: pixels stream from external (no SRAM_A read), only wait
            // for SRAM_B (pool writeback) and conv_frame_rearm.
            if ((sram_b_done_seen || sram_b_done) &&
                (frame_rearm_seen || conv_frame_rearm))
              state <= ST_DONE;
          end else begin
            if ((sram_a_done_seen || sram_a_done) &&
                (sram_b_done_seen || sram_b_done) &&
                (frame_rearm_seen || conv_frame_rearm))
              state <= ST_DONE;
            end
        end

        // ------------------------------------------------------------
        ST_DONE: begin
          done  <= 1'b1;
          state <= ST_IDLE;
        end

        default: state <= ST_IDLE;
      endcase
    end
  end

endmodule
```

A.7.6 Convolution Data Adapter

Adapts SRAM and MMIO image words into the byte/word stream expected by convolution layers.

code/input/RTL/fsm/conv_data_adapter.v

```
// conv_data_adapter: unpack 32-bit SRAM words into Conv input beats.
//
// L1 (layer_sel=00): 1 SRAM word -> 4 byte beats (1-word holding register)
//   SRAM outputs 1024 words, Conv receives 4096 beats x 8-bit, in_byte_en=0001
// L2/L3 also use a 1-word holding register so the adapter can safely absorb
// the SRAM wrapper's 1-cycle read pipeline under downstream backpressure.

```

```
module conv_data_adapter (
  input  wire        clk,
  input  wire        rst_n,
  input  wire [1:0]  layer_sel,

  // Upstream: from SRAM read stream
  input  wire        up_valid,
  input  wire [31:0] up_data,
  input  wire        up_last,
  output wire        up_ready,

  // Downstream: to Conv in port
  output wire        dn_valid,
  output wire [31:0] dn_data,
  output wire [3:0]  dn_byte_en,
  output wire        dn_last,
  input  wire        dn_ready
);

  wire is_l1 = (layer_sel == 2'b00);

  // ---------------------------------------------------------------
  // L1: byte unpack with 1-word holding register
  // L2/L3: buffered word pass-through
  // ---------------------------------------------------------------

  reg [31:0] hold_reg;
  reg        hold_valid;
  reg [1:0]  byte_sel;     // 0,1,2,3
  reg        hold_is_last; // the held word was the last SRAM word

  wire [31:0] cur_word_l1   = hold_valid ? hold_reg : up_data;
  wire [1:0]  cur_byte_sel  = hold_valid ? byte_sel : 2'd0;
  wire        cur_is_last_l1 = hold_valid ? hold_is_last : up_last;

  // Current pixel byte from active word (held word or fresh bypass)
  wire [7:0] pixel_byte = (cur_byte_sel == 2'd0) ? cur_word_l1[ 7: 0] :
                          (cur_byte_sel == 2'd1) ? cur_word_l1[15: 8] :
                          (cur_byte_sel == 2'd2) ? cur_word_l1[23:16] :
                                                   cur_word_l1[31:24];

  // L1 downstream signals
  wire l1_dn_valid = hold_valid || up_valid;
  wire l1_dn_last  = l1_dn_valid && cur_is_last_l1 && (cur_byte_sel == 2'd3);

  // L1 up_ready: assert only when adapter WILL BE free next cycle.
  // SRAM wrapper has 1-cycle read pipeline (read_en -> data_valid next cycle),
  // so up_ready must predict availability 1 cycle ahead.
  //   Case A: hold_valid=1, last byte being consumed -> free next cycle
  //   Case B: hold_valid=0 AND no data arriving -> already free
  // When data is arriving (up_valid=1, hold_valid=0), the adapter is latching
  // this cycle - NOT free for another read next cycle.
  wire l1_will_be_free = (hold_valid && dn_ready && byte_sel == 2'd3)
                       || (!hold_valid && !up_valid);
```

```
  wire l1_up_ready = l1_will_be_free;

  // L2/L3: one-word buffer to absorb the upstream read pipeline.
  // Byte-swap: pool_data stores {ch0,ch1,ch2,ch3} with ch0 at MSB,
  // but Conv pixel_serializer expects ch0 at LSB (in_data[7:0]=ch0).
  wire [31:0] nonl1_raw  = hold_valid ? hold_reg : up_data;
  wire [31:0] nonl1_word = {nonl1_raw[7:0], nonl1_raw[15:8],
                            nonl1_raw[23:16], nonl1_raw[31:24]};
  wire        nonl1_is_last = hold_valid ? hold_is_last : up_last;
  wire nonl1_dn_valid = hold_valid || up_valid;
  wire nonl1_dn_last  = nonl1_dn_valid && nonl1_is_last;
  wire nonl1_will_be_free = (hold_valid && dn_ready)
                         || (!hold_valid && (!up_valid || dn_ready));
  wire nonl1_up_ready = nonl1_will_be_free;

  // Muxed outputs
  assign dn_valid   = is_l1 ? l1_dn_valid : nonl1_dn_valid;
  assign dn_data    = is_l1 ? {24'd0, pixel_byte} : nonl1_word;
  assign dn_byte_en = is_l1 ? 4'b0001 : 4'b1111;
  assign dn_last    = is_l1 ? l1_dn_last : nonl1_dn_last;
  assign up_ready   = is_l1 ? l1_up_ready : nonl1_up_ready;

  // ------------------------------------------------------------
  // L1 holding register and byte_sel control
  // ------------------------------------------------------------
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      hold_reg    <= 32'd0;
      hold_valid  <= 1'b0;
      byte_sel    <= 2'd0;
      hold_is_last <= 1'b0;
    end else if (is_l1) begin
      if (!hold_valid) begin
        if (up_valid) begin
          hold_reg    <= up_data;
          hold_is_last <= up_last;
          hold_valid  <= 1'b1;
          // Fresh word can emit byte0 immediately; keep byte1 queued next.
          byte_sel    <= dn_ready ? 2'd1 : 2'd0;
        end
      end else if (dn_ready) begin
        if (byte_sel == 2'd3) begin
          // Finished current word
          hold_valid <= 1'b0;
          byte_sel   <= 2'd0;
        end else begin
          byte_sel <= byte_sel + 2'd1;
        end
      end
    end else begin
      if (!hold_valid) begin
        if (up_valid) begin
          if (!dn_ready) begin
            hold_reg     <= up_data;
```

```
            hold_valid   <= 1'b1;
            hold_is_last <= up_last;
          end
          byte_sel     <= 2'd0;
        end
      end else if (dn_ready) begin
        hold_valid <= 1'b0;
        byte_sel   <= 2'd0;
      end
    end
  end

endmodule
```

A.7.7 Weight Prepad Inserter

Pads and aligns convolution weight streams before they enter the convolution datapath.

code/input/RTL/fsm/wt_prepad_inserter.v

```
// wt_prepad_inserter: insert 3 zero words before real weight data.
//
// Sits between SRAM_A read stream and Conv wt port.
// When enabled (is_wt_read=1), inserts 3 zero beats then forwards real data.
// When disabled (is_wt_read=0), passes through transparently.
//
// Conv weight_buffer expects DOT_K+3 beats: 3 prepad zeros + DOT_K real weights.
// SRAM only stores DOT_K real weights. This module bridges the gap.

module wt_prepad_inserter (
  input  wire        clk,
  input  wire        rst_n,

  // Control: assert when current SRAM_A transaction is WT_READ
  input  wire        is_wt_read,

  // Upstream: SRAM_A read stream
  input  wire        up_valid,
  input  wire [31:0] up_data,
  input  wire        up_last,
  output wire        up_ready,

  // Downstream: to Conv wt port
  output wire        dn_valid,
  output wire [31:0] dn_data,
  output wire        dn_last,
  input  wire        dn_ready
);

  localparam [1:0] ST_IDLE   = 2'd0,
                   ST_PREPAD = 2'd1,
```

```
                  ST_PASS   = 2'd2;

  reg [1:0] state;
  reg [1:0] pad_cnt;  // counts 0,1,2 (3 zero beats)

  // Rising-edge detection on is_wt_read to avoid re-entering prepad
  reg is_wt_read_d;
  wire is_wt_read_rise = is_wt_read && !is_wt_read_d;
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) is_wt_read_d <= 1'b0;
    else        is_wt_read_d <= is_wt_read;
  end

  // Pass-through when not in WT_READ mode
  wire passthrough = !is_wt_read || (state == ST_PASS);

  // SRAM wrapper read is 1-cycle pipelined. On the last zero-prepad beat,
  // request the first real weight so ST_PASS can start without a bubble.
  assign up_ready = passthrough ? dn_ready
                                : ((state == ST_PREPAD) && (pad_cnt == 2'd2) && dn_ready);
  assign dn_valid = passthrough ? up_valid : (state == ST_PREPAD);
  assign dn_data  = passthrough ? up_data  : 32'd0;
  assign dn_last  = passthrough ? up_last  : 1'b0;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      state   <= ST_IDLE;
      pad_cnt <= 2'd0;
    end else begin
      case (state)
        ST_IDLE: begin
          if (is_wt_read_rise) begin
            // is_wt_read just asserted: insert 3 prepad zeros before real data
            state   <= ST_PREPAD;
            pad_cnt <= 2'd0;
          end
        end

        ST_PREPAD: begin
          if (dn_ready) begin
            if (pad_cnt == 2'd2) begin
              state <= ST_PASS;  // prepad done, now forward real data
            end else begin
              pad_cnt <= pad_cnt + 2'd1;
            end
          end
        end

        ST_PASS: begin
          // Forward until upstream last beat is consumed
          if (up_valid && up_ready && up_last) begin
            state <= ST_IDLE;
          end
        end
```

```
86
87          default: state <= ST_IDLE;
88        endcase
89      end
90    end
91
92  endmodule
```

A.8 Convolution Core RTL

A.8.1 Convolution Top

Connects input alignment, line buffering, convolution buffering, weight buffering, SA launch, and output streaming.

code/input/RTL/conv_core/conv_top.v

```
1  module conv_top #(
2    parameter integer W      = 64,    // max line buffer width (L1)
3    parameter integer DATA_W = 8,
4    parameter integer ROWS   = 16,
5    parameter integer COLS   = 4,
6    parameter integer DOT_K  = 72,    // max dot product length (L3: 9*8)
7    parameter integer ACC_W  = 23,
8    parameter integer IMG_H  = 64,
9    parameter integer IMG_W  = 64,
10   parameter integer C_IN   = 8      // max input channels (L3)
11 )(
12   input  wire [1:0]                layer_sel,
13   input  wire clk,
14   input  wire rst_n,
15
16   // Pixel stream input (ready/valid handshake)
17   input  wire                      in_valid,
18   input  wire        [31:0]        in_data,
19   input  wire        [3:0]         in_byte_en,
20   input  wire                      in_last,
21   output wire                      in_ready,
22
23   // Weight load channel
24   input  wire                      wt_valid,
25   output wire                      wt_ready,
26   input  wire                      wt_last,
27   input  wire signed [COLS*DATA_W-1:0] wt_data,
28
29   // Weight load done (1-cycle pulse on weights_loaded rising edge)
30   output wire                        wt_load_done,
31
32   // Systolic-array output
33   output wire                        sa_done,
```

```
    // Per-column real-time stream output
    output wire signed [COLS*ACC_W-1:0] c_out_col_stream_flat,// 4*23 bits, one output row
    output wire        [COLS-1:0]       c_out_col_valid,
    output wire        [COLS-1:0]       c_out_col_last,

    // Frame rearm: Conv backend fully drained, ready for next frame/pass
    output wire                         frame_rearm_out,
    // Frame done: all input pixels received, SA may still be draining
    output wire                         frame_done_out
);

    // Debug-visible state encoding kept in top hierarchy for TB/wave compatibility.
    localparam [1:0] ST_IDLE     = 2'd0;

    // Runtime IMG_PIXELS decode from layer_sel
    // Counts input BEATS (not bytes). Each beat is one pix_accept.
    //   L1: 4096 beats (1 byte/beat, 64*64*1 bytes)
    //   L2: 961 beats  (4 bytes/beat, 31*31 wide pixels)
    //   L3: 392 beats  (4 bytes/beat, 14*14*8/4 half-pixels)
    reg [1:0] active_layer_sel_q;
    reg [31:0] eff_img_pixels;
    always @(posedge clk or negedge rst_n) begin
      if (!rst_n)
        active_layer_sel_q <= 2'b00;
      else
        active_layer_sel_q <= layer_sel;
    end

    always @* begin
      case (active_layer_sel_q)
        2'b01:   eff_img_pixels = 32'd961;   // 31*31
        2'b10:   eff_img_pixels = 32'd392;   // 14*14*2
        default: eff_img_pixels = 32'd4096;  // 64*64
      endcase
    end

    reg [31:0] pix_cnt;
    reg        frame_done;

    wire [1:0] state;
    wire [8:0] feed_cnt;
    wire [8:0] wt_load_cnt;
    wire [8:0] rd_col_r;
    wire       rd_en_r;
    wire       rd_bank_r;
    wire       wb_ring_r;
    wire       start_pulse_r;
    wire       weights_loaded;
    reg        weights_loaded_d;
    assign     wt_load_done = weights_loaded && !weights_loaded_d;
    always @(posedge clk or negedge rst_n) begin
      if (!rst_n) weights_loaded_d <= 1'b0;
      else        weights_loaded_d <= weights_loaded;
    end
```

```
    wire       wt_load_active;
    wire       consume_ready_bank;
    wire       consume_bank_sel;
    wire [4:0] valid_rows_A, valid_rows_B;
    wire       partial_pending;
    // Frame re-arm: declared here, driven below after backend idle detection.
    wire frame_rearm;

    // Internal observation signals kept hierarchical for TB and wave scripts.
    wire                         pix3_valid;
    wire signed [C_IN*DATA_W-1:0] row0_pix;
    wire signed [C_IN*DATA_W-1:0] row1_pix;
    wire signed [C_IN*DATA_W-1:0] row2_pix;
    wire signed [DATA_W-1:0] wb_out0, wb_out1, wb_out2, wb_out3;
    wire [COLS-1:0] wb_out_valid;
    wire                        bank_can_write;

    // Input-side flow control.
    wire pix_accept = in_valid && in_ready;
    assign in_ready = !frame_done && bank_can_write;

    // Bank write-side control.
    wire rd_active = (state != ST_IDLE);

    // Launch arbitration and weight-side handshake.
    wire have_ready_block;
    wire wt_fire = wt_valid && wt_ready;
    wire launch_from_A;
    wire have_launchable_block = have_ready_block && weights_loaded;
    // Prevent SA launch after frame_done (avoid stale bank data between layers/passes)
    wire can_launch = have_launchable_block && !wt_fire && !frame_done;
    wire [1:0] sa_mode_cfg = active_layer_sel_q;
    reg  [1:0] sa_mode_cfg_q;
    reg  [4:0] sa_valid_rows_cfg_q;
    reg        sa_start_pulse_q;
    reg  signed [ROWS*DATA_W-1:0] sa_a_in_flat_q;
    reg  signed [COLS*DATA_W-1:0] sa_b_in_flat_q;

    // SA input gating.
    wire signed [ROWS*DATA_W-1:0] a_in_flat;
    wire signed [ROWS*DATA_W-1:0] a_in_from_bank;
    wire signed [ROWS*DATA_W-1:0] raw_firstcol_flat;
    wire signed [ROWS*COLS*ACC_W-1:0] c_out_raw_flat;
    wire signed [DATA_W-1:0] wb_col0 = wb_out_valid[0] ? wb_out0 : {DATA_W{1'b0}};
    wire signed [DATA_W-1:0] wb_col1 = wb_out_valid[1] ? wb_out1 : {DATA_W{1'b0}};
    wire signed [DATA_W-1:0] wb_col2 = wb_out_valid[2] ? wb_out2 : {DATA_W{1'b0}};
    wire signed [DATA_W-1:0] wb_col3 = wb_out_valid[3] ? wb_out3 : {DATA_W{1'b0}};
    wire sa_a_gate = rd_en_r || ((state == ST_IDLE) && can_launch);
    wire rd_bank_sel = ((state == ST_IDLE) && can_launch) ? (launch_from_A ? 1'b0 : 1'b1) : rd_bank_r;
    wire signed [COLS*DATA_W-1:0] b_in_to_sa =
        rd_en_r ? {wb_col3, wb_col2, wb_col1, wb_col0} : {(COLS*DATA_W){1'b0}};
    // in_last is consumed by frame_done logic below (no longer unused).

    // Gate A inputs to zero while idle to avoid accumulating stale values.
```

```
142   assign a_in_flat = sa_a_gate ? a_in_from_bank : {ROWS*DATA_W{1'b0}};
143
144   // One-cycle input pipeline into the systolic array. This cuts the long
145   // layer-select-dependent Conv_Buffer -> PE MAC path.
146   always @(posedge clk or negedge rst_n) begin
147     if (!rst_n) begin
148       sa_mode_cfg_q       <= 2'b00;
149       sa_valid_rows_cfg_q <= 5'd0;
150       sa_start_pulse_q    <= 1'b0;
151       sa_a_in_flat_q      <= {ROWS*DATA_W{1'b0}};
152       sa_b_in_flat_q      <= {COLS*DATA_W{1'b0}};
153     end else begin
154       sa_mode_cfg_q       <= sa_mode_cfg;
155       sa_valid_rows_cfg_q <= rd_bank_sel ? valid_rows_B : valid_rows_A;
156       sa_start_pulse_q    <= start_pulse_r;
157       sa_a_in_flat_q      <= a_in_flat;
158       sa_b_in_flat_q      <= b_in_to_sa;
159     end
160   end
161
162   input_row_aligner #(
163     .W   (W),
164     .DW  (DATA_W),
165     .C_IN(C_IN)
166   ) u_input_row_aligner (
167     .layer_sel  (active_layer_sel_q),
168     .clk        (clk),
169     .rst_n      (rst_n),
170     .frame_rearm(frame_rearm),
171     .in_valid   (pix_accept),
172     .in_data    (in_data),
173     .in_byte_en (in_byte_en),
174     .pix3_valid (pix3_valid),
175     .row0_out   (row0_pix),
176     .row1_out   (row1_pix),
177     .row2_out   (row2_pix)
178   );
179
180   weight_buffer #(
181     .DATA_W(COLS*DATA_W),
182     .COLS  (COLS),
183     .L0    (DOT_K),
184     .L1    (DOT_K + 1),
185     .L2    (DOT_K + 2),
186     .L3    (DOT_K + 3)
187   ) u_weight_buffer (
188     .layer_sel(active_layer_sel_q),
189     .clk      (clk),
190     .rst_n    (rst_n),
191     .ring     (wb_ring_r),
192     .wr_en    (wt_fire),
193     .in_data  (wt_data),
194     .out_ready(rd_en_r),
195     .out0     (wb_out0),
```

```
    .out1     (wb_out1),
    .out2     (wb_out2),
    .out3     (wb_out3),
    .out_valid(wb_out_valid)
  );

  Conv_Buffer #(
    .DATA_W(DATA_W),
    .ROWS  (ROWS),
    .DOT_K (DOT_K),
    .C_IN  (C_IN)
  ) u_conv_buffer (
    .layer_sel         (active_layer_sel_q),
    .clk               (clk),
    .rst_n             (rst_n),
    .frame_rearm       (frame_rearm),
    .pix3_valid        (pix3_valid),
    .row0_pix          (row0_pix),
    .row1_pix          (row1_pix),
    .row2_pix          (row2_pix),
    .rd_active         (rd_active),
    .rd_en             (rd_en_r),
    .rd_bank           (rd_bank_sel),
    .rd_col            (rd_col_r),
    .consume_ready_bank(consume_ready_bank),
    .consume_bank_sel  (consume_bank_sel),
    .bank_can_write    (bank_can_write),
    .have_ready_block  (have_ready_block),
    .launch_from_A     (launch_from_A),
    .valid_rows_A      (valid_rows_A),
    .valid_rows_B      (valid_rows_B),
    .partial_pending   (partial_pending),
    .raw_firstcol_flat (raw_firstcol_flat)
  );

  sa_skew_feeder #(
    .ROWS(ROWS)
  ) u_sa_skew_feeder (
    .clk             (clk),
    .rst_n           (rst_n),
    .frame_rearm     (frame_rearm),
    .rd_en           (rd_en_r),
    .in_firstcol_flat(raw_firstcol_flat),
    .out_firstcol_flat(a_in_from_bank)
  );

  conv_engine_ctrl #(
    .DOT_K(DOT_K),
    .COLS (COLS),
    .ROWS (ROWS)
  ) u_conv_engine_ctrl (
    .layer_sel         (active_layer_sel_q),
    .clk               (clk),
    .rst_n             (rst_n),
```

```
    .wt_valid          (wt_valid),
    .wt_last           (wt_last),
    .have_ready_block  (have_ready_block),
    .launch_from_A     (launch_from_A),
    .launch_valid_rows (launch_from_A ? valid_rows_A : valid_rows_B),
    .frame_rearm       (frame_rearm),
    .sa_done           (sa_done),
    .wt_ready          (wt_ready),
    .weights_loaded    (weights_loaded),
    .wt_load_active    (wt_load_active),
    .wt_load_cnt       (wt_load_cnt),
    .state_dbg         (state),
    .feed_cnt_dbg      (feed_cnt),
    .consume_ready_bank(consume_ready_bank),
    .consume_bank_sel  (consume_bank_sel),
    .rd_en             (rd_en_r),
    .rd_bank           (rd_bank_r),
    .rd_col            (rd_col_r),
    .wb_ring           (wb_ring_r),
    .start_pulse       (start_pulse_r)
  );

  // Backend drain detection: all SA work complete, window FIFO truly empty,
  // and no ready/partial block pending.
  wire backend_idle = frame_done
                  && (state == ST_IDLE)
                  && !have_ready_block
                  && !partial_pending;

  // Re-arm: one cycle after backend_idle, clear frame state for next transaction.
  reg backend_idle_d;
  assign frame_rearm = backend_idle && !backend_idle_d;
  assign frame_rearm_out = frame_rearm;
  assign frame_done_out  = frame_done;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
      backend_idle_d <= 1'b0;
    else
      backend_idle_d <= backend_idle;
  end

  // Frame ends when the beat count reaches eff_img_pixels or in_last is asserted.
  // frame_rearm clears frame state after the backend drains, allowing back-to-back transactions.
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      pix_cnt <= 32'd0;
      frame_done <= 1'b0;
    end else begin
      if (frame_rearm) begin
        pix_cnt <= 32'd0;
        frame_done <= 1'b0;
      end else if (pix_accept && !frame_done) begin
        if (pix_cnt == (eff_img_pixels - 32'd1) || in_last) begin
```

```
          frame_done <= 1'b1;
        end
        pix_cnt <= pix_cnt + 32'd1;
      end
    end
  end

  systolic_array_top #(
    .DATA_W(DATA_W),
    .ROWS  (ROWS),
    .COLS  (COLS),
    .DOT_K (DOT_K),
    .ACC_W (ACC_W)
  ) u_systolic_array_top (
    .clk        (clk),
    .rst_n      (rst_n),
    .start_pulse(sa_start_pulse_q),
    .mode_cfg   (sa_mode_cfg_q),
    .valid_rows_cfg(sa_valid_rows_cfg_q),
    .a_in_flat  (sa_a_in_flat_q),
    .b_in_flat  (sa_b_in_flat_q),
    .c_out_flat (c_out_raw_flat),
    .done       (sa_done),
    .col_stream_data_flat(c_out_col_stream_flat),
    .col_stream_valid    (c_out_col_valid),
    .col_stream_last     (c_out_col_last)
  );

endmodule
```

A.8.2 Convolution Engine Controller

Controls weight loading, ready-bank consumption, systolic-array launch, feed, and drain sequencing.

code/input/RTL/conv_core/conv_engine_ctrl.v

```
module conv_engine_ctrl #(
  parameter integer DOT_K = 72,
  parameter integer COLS  = 4,
  parameter integer ROWS  = 16
)(
  input  wire [1:0] layer_sel,
  input  wire       clk,
  input  wire       rst_n,
  input  wire       wt_valid,
  input  wire       wt_last,
  input  wire       have_ready_block,
  input  wire       launch_from_A,
  input  wire [4:0] launch_valid_rows,
  input  wire       frame_rearm,
  input  wire       sa_done,
```

```
    output wire       wt_ready,
    output wire       weights_loaded,
    output wire       wt_load_active,
    output wire [8:0] wt_load_cnt,
    output wire [1:0] state_dbg,
    output wire [8:0] feed_cnt_dbg,
    output wire       consume_ready_bank,
    output wire       consume_bank_sel,
    output wire       rd_en,
    output wire       rd_bank,
    output wire [8:0] rd_col,
    output wire       wb_ring,
    output wire       start_pulse
);

    localparam [1:0] ST_IDLE     = 2'd0;
    localparam [1:0] ST_PRERING  = 2'd1;
    localparam [1:0] ST_FEED     = 2'd2;
    localparam [1:0] ST_WAITDONE = 2'd3;

    // Runtime decode: eff_dot_k, FEED_CYCLES, WT_LOAD_CYCLES
    reg [8:0] eff_dot_k;
    reg [8:0] eff_feed_last_base;
    reg [8:0] eff_wt_load_cycles;
    always @* begin
      case (layer_sel)
        2'b01:   begin eff_dot_k = 9'd36; eff_feed_last_base = 9'd50;  eff_wt_load_cycles = 9'd39; end //
      36+16-1-1=50, 36+4-1=39
        2'b10:   begin eff_dot_k = 9'd72; eff_feed_last_base = 9'd86;  eff_wt_load_cycles = 9'd75; end //
      72+16-1-1=86, 72+4-1=75
        default: begin eff_dot_k = 9'd9;  eff_feed_last_base = 9'd23;  eff_wt_load_cycles = 9'd12; end //
      9+16-1-1=23, 9+4-1=12
      endcase
    end

    reg [1:0] state_r;
    reg [8:0] feed_cnt_r;
    reg [8:0] wt_load_cnt_r;
    reg       rd_en_r;
    reg       rd_bank_r;
    reg [8:0] rd_col_r;
    reg       wb_ring_r;
    reg       start_pulse_r;
    reg       weights_loaded_r;
    reg       wt_load_active_r;
    reg [4:0] active_valid_rows_r;
    reg [8:0] active_feed_last_r;

    wire wt_fire = wt_valid && wt_ready;
    wire have_launchable_block = have_ready_block && weights_loaded_r;
    wire can_launch = have_launchable_block && !wt_fire;

    assign wt_ready           = (state_r == ST_IDLE);
    assign weights_loaded     = weights_loaded_r;
```

```
  assign wt_load_active     = wt_load_active_r;
  assign wt_load_cnt        = wt_load_cnt_r;
  assign state_dbg          = state_r;
  assign feed_cnt_dbg       = feed_cnt_r;
  assign rd_en              = rd_en_r;
  assign rd_bank            = rd_bank_r;
  assign rd_col             = rd_col_r;
  assign wb_ring            = wb_ring_r;
  assign start_pulse        = start_pulse_r;
  assign consume_ready_bank =
      ((state_r == ST_IDLE) && can_launch) ||
      ((state_r == ST_FEED) && (feed_cnt_r == active_feed_last_r) && have_launchable_block) ||
      ((state_r == ST_WAITDONE) && sa_done && have_launchable_block);
  assign consume_bank_sel   = launch_from_A ? 1'b0 : 1'b1;

  // Weight load protocol:
  // 1) After wt_valid/wt_ready, weight_buffer accepts one new weight beat.
  // 2) wt_last marks the last beat in the current weight group.
  // 3) SA launch is allowed from a ready bank only after weights_loaded=1.
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      weights_loaded_r <= 1'b0;
      wt_load_active_r <= 1'b0;
      wt_load_cnt_r    <= 9'd0;
    end else if (frame_rearm) begin
      weights_loaded_r <= 1'b0;
      wt_load_active_r <= 1'b0;
      wt_load_cnt_r    <= 9'd0;
    end else if (wt_fire) begin
      if (!wt_load_active_r) begin
        weights_loaded_r <= wt_last;
        wt_load_active_r <= !wt_last;
        wt_load_cnt_r    <= 9'd1;
      end else begin
        weights_loaded_r <= wt_last;
        wt_load_active_r <= !wt_last;
        if (wt_load_cnt_r < eff_wt_load_cycles)
          wt_load_cnt_r <= wt_load_cnt_r + 9'd1;
        else
          wt_load_cnt_r <= wt_load_cnt_r;
      end
    end
  end

  // SA launch control:
  // ST_IDLE, the ST_FEED tail beat, or ST_WAITDONE can enter ST_PRERING
  // when a ready block is available.
  // The next ST_PRERING cycle asserts start_pulse and rd_en together.
  // ST_FEED sends columns 0..DOT_K-1, then sends out-of-range columns to flush.
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      state_r       <= ST_IDLE;
      feed_cnt_r    <= 9'd0;
      rd_col_r      <= 9'd0;
```

```
      rd_en_r       <= 1'b0;
      rd_bank_r     <= 1'b0;
      wb_ring_r     <= 1'b0;
      start_pulse_r <= 1'b0;
      active_valid_rows_r <= 5'd0;
      active_feed_last_r  <= 9'd0;
    end else if (frame_rearm) begin
      state_r       <= ST_IDLE;
      feed_cnt_r    <= 9'd0;
      rd_col_r      <= 9'd0;
      rd_en_r       <= 1'b0;
      rd_bank_r     <= 1'b0;
      wb_ring_r     <= 1'b0;
      start_pulse_r <= 1'b0;
      active_valid_rows_r <= 5'd0;
      active_feed_last_r  <= 9'd0;
    end else begin
      rd_en_r       <= 1'b0;
      wb_ring_r     <= 1'b0;
      start_pulse_r <= 1'b0;

      case (state_r)
        ST_IDLE: begin
          feed_cnt_r <= 9'd0;
          rd_col_r   <= 9'd0;
          if (can_launch) begin
            rd_bank_r <= launch_from_A ? 1'b0 : 1'b1;
            active_valid_rows_r <= launch_valid_rows;
            if (launch_valid_rows > 5'd0)
              active_feed_last_r <= eff_dot_k + {{4{1'b0}}, launch_valid_rows} - 9'd2;
            else
              active_feed_last_r <= eff_feed_last_base;
            wb_ring_r <= 1'b1;
            state_r   <= ST_PRERING;
          end
        end

        ST_PRERING: begin
          rd_col_r      <= 9'd0;
          rd_en_r       <= 1'b1;
          start_pulse_r <= 1'b1;
          feed_cnt_r    <= 9'd1;
          state_r       <= ST_FEED;
        end

        ST_FEED: begin
          rd_en_r <= 1'b1;
          if (feed_cnt_r < eff_dot_k) begin
            rd_col_r <= feed_cnt_r;
          end else begin
            rd_col_r <= 9'd511;  // out-of-range -> get_byte returns 0 (flush)
          end

          if (feed_cnt_r == active_feed_last_r) begin
```

```
            feed_cnt_r <= 9'd0;
            rd_col_r   <= 9'd0;
            if (have_launchable_block) begin
              rd_bank_r <= launch_from_A ? 1'b0 : 1'b1;
              active_valid_rows_r <= launch_valid_rows;
              if (launch_valid_rows > 5'd0)
                active_feed_last_r <= eff_dot_k + {{4{1'b0}}, launch_valid_rows} - 9'd2;
              else
                active_feed_last_r <= eff_feed_last_base;
              wb_ring_r <= 1'b1;
              state_r   <= ST_PRERING;
            end else begin
              state_r <= ST_WAITDONE;
            end
          end else begin
            feed_cnt_r <= feed_cnt_r + 9'd1;
          end
        end

        ST_WAITDONE: begin
          if (sa_done) begin
            if (have_launchable_block) begin
              rd_bank_r <= launch_from_A ? 1'b0 : 1'b1;
              active_valid_rows_r <= launch_valid_rows;
              if (launch_valid_rows > 5'd0)
                active_feed_last_r <= eff_dot_k + {{4{1'b0}}, launch_valid_rows} - 9'd2;
              else
                active_feed_last_r <= eff_feed_last_base;
              wb_ring_r <= 1'b1;
              state_r   <= ST_PRERING;
            end else begin
              state_r <= ST_IDLE;
            end
          end
        end

        default: begin
          state_r <= ST_IDLE;
        end
      endcase
    end
  end

endmodule
```

A.8.3 Convolution Buffer

Stores incoming convolution windows, tracks ready banks, and feeds rows into the systolic array.

code/input/RTL/conv_core/Conv_Buffer.v

```
// Conv_Buffer - dual-bank stripe buffer for conv_top.
//
// Storage: flat 8-bit regs (2 x 432 entries) - cheap write-enable per register.
// Read path: hardwired concat into 24 x 144-bit "group" wires (zero gate cost),
//   then a single 24:1 MUX + 3:1 barrel shift extract 16 consecutive bytes.
// Address decode: counter-based (ch_r, dx_r, dy_r) advances with rd_col - no
//   runtime division/modulo.
// zero_fill_tail: eliminated; read path masks out unfilled columns via fill_cols.
//
// Synthesis: ~84K um^2 (vs ~114K baseline with 16x 432:1 MUX + div/mod).

module Conv_Buffer #(
  parameter integer DATA_W = 8,
  parameter integer ROWS   = 16,
  parameter integer DOT_K  = 72,
  parameter integer C_IN   = 8
)(
  input  wire [1:0]                    layer_sel,
  input  wire                          clk,
  input  wire                          rst_n,
  input  wire                          frame_rearm,
  input  wire                          pix3_valid,
  input  wire signed [C_IN*DATA_W-1:0] row0_pix,
  input  wire signed [C_IN*DATA_W-1:0] row1_pix,
  input  wire signed [C_IN*DATA_W-1:0] row2_pix,
  input  wire                          rd_active,
  input  wire                          rd_en,
  input  wire                          rd_bank,
  input  wire [8:0]                    rd_col,
  input  wire                          consume_ready_bank,
  input  wire                          consume_bank_sel,
  output wire                          bank_can_write,
  output wire                          have_ready_block,
  output wire                          launch_from_A,
  output wire [4:0]                    valid_rows_A,
  output wire [4:0]                    valid_rows_B,
  output wire                          partial_pending,
  output wire signed [ROWS*DATA_W-1:0] raw_firstcol_flat
);

  // ------------------------------------------------------------------
  // Constants
  // ------------------------------------------------------------------
  localparam integer PIX_W        = C_IN * DATA_W;
  localparam integer STRIPE_ROWS  = 3;
  localparam integer STRIPE_COLS  = 18;
  localparam integer STRIPE_STEP  = 16;
  localparam integer STRIPE_BYTES = STRIPE_ROWS * STRIPE_COLS * C_IN;
  localparam integer WORD_W       = STRIPE_COLS * DATA_W;  // 144
  localparam integer MAX_WORDS    = STRIPE_ROWS * C_IN;    // 24

  // ------------------------------------------------------------------
```

```
53   // Flat 8-bit storage (same as Conv_Buffer - cheap writes)
54   // ------------------------------------------------------------------
55   reg signed [7:0] bank_A [0:STRIPE_BYTES-1];
56   reg signed [7:0] bank_B [0:STRIPE_BYTES-1];
57
58   // ------------------------------------------------------------------
59   // Control registers
60   // ------------------------------------------------------------------
61   reg [6:0] x_pos_r;
62   reg [6:0] fill_base_x_r;
63   reg [4:0] fill_cols_r;
64   reg       wr_bank_r;
65   reg       ready_A_r;
66   reg       ready_B_r;
67   reg [4:0] valid_rows_A_r;
68   reg [4:0] valid_rows_B_r;
69   reg       seed_pending_r;
70   reg       seed_src_bank_r;
71   // Per-bank fill column count (replaces zero_fill_tail)
72   reg [4:0] fill_cols_A_r;
73   reg [4:0] fill_cols_B_r;
74
75   // ------------------------------------------------------------------
76   // Layer-dependent parameters
77   // ------------------------------------------------------------------
78   reg [6:0] eff_w;
79   reg [3:0] eff_c;
80   reg [6:0] eff_out_w;
81   reg [8:0] eff_dot_k;
82
83   always @* begin
84     case (layer_sel)
85       2'b01: begin
86         eff_w     = 7'd31;
87         eff_c     = 4'd4;
88         eff_out_w = 7'd29;
89         eff_dot_k = 9'd36;
90       end
91       2'b10: begin
92         eff_w     = 7'd14;
93         eff_c     = 4'd8;
94         eff_out_w = 7'd12;
95         eff_dot_k = 9'd72;
96       end
97       default: begin
98         eff_w     = 7'd64;
99         eff_c     = 4'd1;
100        eff_out_w = 7'd62;
101        eff_dot_k = 9'd9;
102      end
103    endcase
104  end
105
106  // ------------------------------------------------------------------
```

```
107   // TB-visible diagnostic wires (same as Conv_Buffer)
108   // ------------------------------------------------------------------
109   wire stripe_window_open =
110       (x_pos_r >= fill_base_x_r) && (x_pos_r < (fill_base_x_r + 7'd18));
111   wire write_bank_blocked = ready_A_r && ready_B_r;
112   wire write_read_conflict = rd_active && (wr_bank_r == rd_bank);
113   wire input_stall = write_bank_blocked || write_read_conflict;
114   wire accepting_input_col = pix3_valid && !input_stall && stripe_window_open;
115   wire signed [DOT_K*DATA_W-1:0] write_sample_flat =
116       {{(DOT_K*DATA_W - 3*PIX_W){1'b0}}, row2_pix, row1_pix, row0_pix};
117
118   // ------------------------------------------------------------------
119   // Functions
120   // ------------------------------------------------------------------
121   function integer stripe_idx;
122     input integer row_idx;
123     input integer col_idx;
124     input integer ch_idx;
125     begin
126       stripe_idx = ((row_idx * STRIPE_COLS) + col_idx) * C_IN + ch_idx;
127     end
128   endfunction
129
130   function [4:0] calc_valid_rows;
131     input [6:0] base_x;
132     input [6:0] out_w;
133     reg [6:0] remaining;
134     begin
135       if (base_x >= out_w)
136         calc_valid_rows = 5'd0;
137       else begin
138         remaining = out_w - base_x;
139         if (remaining >= ROWS[6:0])
140           calc_valid_rows = ROWS[4:0];
141         else
142           calc_valid_rows = {1'b0, remaining[3:0]};
143       end
144     end
145   endfunction
146
147   // ------------------------------------------------------------------
148   // Write tasks (from Conv_Buffer - simple flat-reg writes)
149   // ------------------------------------------------------------------
150   integer col_off_i;
151   reg [4:0] new_valid_rows;
152   integer real_cols_i;
153
154   task write_input_column;
155     input dst_bank;
156     input integer col_idx;
157     integer ch_i;
158     begin
159       for (ch_i = 0; ch_i < C_IN; ch_i = ch_i + 1) begin
160         if (dst_bank) begin
```

```
          bank_B[stripe_idx(0, col_idx, ch_i)] <= row0_pix[ch_i*DATA_W +: DATA_W];
          bank_B[stripe_idx(1, col_idx, ch_i)] <= row1_pix[ch_i*DATA_W +: DATA_W];
          bank_B[stripe_idx(2, col_idx, ch_i)] <= row2_pix[ch_i*DATA_W +: DATA_W];
        end else begin
          bank_A[stripe_idx(0, col_idx, ch_i)] <= row0_pix[ch_i*DATA_W +: DATA_W];
          bank_A[stripe_idx(1, col_idx, ch_i)] <= row1_pix[ch_i*DATA_W +: DATA_W];
          bank_A[stripe_idx(2, col_idx, ch_i)] <= row2_pix[ch_i*DATA_W +: DATA_W];
        end
      end
    end
  endtask

  task copy_overlap_cols;
    input dst_bank;
    input src_bank;
    integer row_i;
    integer ch_i;
    begin
      for (row_i = 0; row_i < STRIPE_ROWS; row_i = row_i + 1) begin
        for (ch_i = 0; ch_i < C_IN; ch_i = ch_i + 1) begin
          if (dst_bank) begin
            if (src_bank) begin
              bank_B[stripe_idx(row_i, 0, ch_i)] <= bank_B[stripe_idx(row_i, 16, ch_i)];
              bank_B[stripe_idx(row_i, 1, ch_i)] <= bank_B[stripe_idx(row_i, 17, ch_i)];
            end else begin
              bank_B[stripe_idx(row_i, 0, ch_i)] <= bank_A[stripe_idx(row_i, 16, ch_i)];
              bank_B[stripe_idx(row_i, 1, ch_i)] <= bank_A[stripe_idx(row_i, 17, ch_i)];
            end
          end else begin
            if (src_bank) begin
              bank_A[stripe_idx(row_i, 0, ch_i)] <= bank_B[stripe_idx(row_i, 16, ch_i)];
              bank_A[stripe_idx(row_i, 1, ch_i)] <= bank_B[stripe_idx(row_i, 17, ch_i)];
            end else begin
              bank_A[stripe_idx(row_i, 0, ch_i)] <= bank_A[stripe_idx(row_i, 16, ch_i)];
              bank_A[stripe_idx(row_i, 1, ch_i)] <= bank_A[stripe_idx(row_i, 17, ch_i)];
            end
          end
        end
      end
    end
  endtask

  // zero_fill_tail: ELIMINATED - read-side fill_cols mask handles this

  task mark_ready_bank;
    input dst_bank;
    input [4:0] valid_rows;
    input [4:0] fill_cols;
    begin
      if (dst_bank) begin
        ready_B_r      <= (valid_rows != 5'd0);
        valid_rows_B_r <= valid_rows;
        fill_cols_B_r  <= fill_cols;
      end else begin
```

```
        ready_A_r      <= (valid_rows != 5'd0);
        valid_rows_A_r <= valid_rows;
        fill_cols_A_r  <= fill_cols;
      end
    end
  endtask

  // ------------------------------------------------------------------
  // Output assignments
  // ------------------------------------------------------------------
  assign bank_can_write   = !input_stall;
  assign have_ready_block = ready_A_r || ready_B_r;
  assign launch_from_A    = ready_A_r;
  assign valid_rows_A     = valid_rows_A_r;
  assign valid_rows_B     = valid_rows_B_r;
  assign partial_pending  = seed_pending_r || (fill_cols_r != 5'd0);

  // ==================================================================
  // READ PATH - counter-based decode + hardwired concatenation
  // ==================================================================

  // Hardwired 144-bit group wires: zero gate cost, pure wiring.
  // group_X[dy*C_IN+ch] concatenates 18 flat regs into a 144-bit word.
  wire [WORD_W-1:0] group_A [0:MAX_WORDS-1];
  wire [WORD_W-1:0] group_B [0:MAX_WORDS-1];

  genvar g_dy, g_ch, g_col;
  generate
    for (g_dy = 0; g_dy < STRIPE_ROWS; g_dy = g_dy + 1) begin : gen_dy
      for (g_ch = 0; g_ch < C_IN; g_ch = g_ch + 1) begin : gen_ch
        for (g_col = 0; g_col < STRIPE_COLS; g_col = g_col + 1) begin : gen_col
          localparam integer WADDR = g_dy * C_IN + g_ch;
          localparam integer SIDX  = (g_dy * STRIPE_COLS + g_col) * C_IN + g_ch;
          assign group_A[WADDR][g_col*DATA_W +: DATA_W] = bank_A[SIDX];
          assign group_B[WADDR][g_col*DATA_W +: DATA_W] = bank_B[SIDX];
        end
      end
    end
  endgenerate

  // ------------------------------------------------------------------
  // Counter-based im2col decode (replaces runtime div/mod on rd_col)
  // ------------------------------------------------------------------
  reg [3:0] ch_r;
  reg [1:0] dx_r;
  reg [1:0] dy_r;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      ch_r <= 4'd0;
      dx_r <= 2'd0;
      dy_r <= 2'd0;
    end else if (frame_rearm) begin
      ch_r <= 4'd0;
```

```
      dx_r <= 2'd0;
      dy_r <= 2'd0;
    end else if (rd_en && rd_col < eff_dot_k) begin
      if (rd_col == 9'd0) begin
        if (eff_c == 4'd1) begin
          ch_r <= 4'd0;
          dx_r <= 2'd1;
          dy_r <= 2'd0;
        end else begin
          ch_r <= 4'd1;
          dx_r <= 2'd0;
          dy_r <= 2'd0;
        end
      end else begin
        if (ch_r == eff_c - 4'd1) begin
          ch_r <= 4'd0;
          if (dx_r == 2'd2) begin
            dx_r <= 2'd0;
            dy_r <= dy_r + 2'd1;
          end else begin
            dx_r <= dx_r + 2'd1;
          end
        end else begin
          ch_r <= ch_r + 4'd1;
        end
      end
    end else if (!rd_en) begin
      ch_r <= 4'd0;
      dx_r <= 2'd0;
      dy_r <= 2'd0;
    end
  end

  // Current-cycle combinational decode
  reg [3:0] cur_ch;
  reg [1:0] cur_dx;
  reg [1:0] cur_dy;
  always @* begin
    if (!rd_en || rd_col >= eff_dot_k) begin
      cur_ch = 4'd0;
      cur_dx = 2'd0;
      cur_dy = 2'd0;
    end else if (rd_col == 9'd0) begin
      cur_ch = 4'd0;
      cur_dx = 2'd0;
      cur_dy = 2'd0;
    end else begin
      cur_ch = ch_r;
      cur_dx = dx_r;
      cur_dy = dy_r;
    end
  end

  // ----------------------------------------------------------------
```

```
  // Read MUX: 24:1 group select -> shift extract -> fill_cols mask
  // ------------------------------------------------------------------
  reg [WORD_W-1:0] rd_word;
  reg [4:0] selected_valid_rows;
  reg [4:0] rd_fill_cols;
  reg signed [ROWS*DATA_W-1:0] raw_col_r;
  integer ri;

  // Word address - always use C_IN stride to match generate block indexing
  reg [4:0] rd_word_addr;
  always @* begin
    rd_word_addr = {cur_dy, 3'b000} + {1'b0, cur_ch};  // dy * 8 + ch
  end

  always @* begin
    selected_valid_rows = rd_bank ? valid_rows_B_r : valid_rows_A_r;
    rd_fill_cols        = rd_bank ? fill_cols_B_r  : fill_cols_A_r;
    rd_word             = rd_bank ? group_B[rd_word_addr] : group_A[rd_word_addr];

    raw_col_r = {(ROWS*DATA_W){1'b0}};
    if (rd_en && rd_col < eff_dot_k) begin
      for (ri = 0; ri < ROWS; ri = ri + 1) begin
        if (ri < selected_valid_rows && (ri + cur_dx) < rd_fill_cols)
          raw_col_r[ri*DATA_W +: DATA_W] =
              rd_word[(ri + cur_dx) * DATA_W +: DATA_W];
      end
    end
  end

  assign raw_firstcol_flat = raw_col_r;

  // ------------------------------------------------------------------
  // Main sequential block - stripe fill FSM
  // ------------------------------------------------------------------
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      x_pos_r         <= 7'd0;
      fill_base_x_r   <= 7'd0;
      fill_cols_r     <= 5'd0;
      wr_bank_r       <= 1'b0;
      ready_A_r       <= 1'b0;
      ready_B_r       <= 1'b0;
      valid_rows_A_r  <= 5'd0;
      valid_rows_B_r  <= 5'd0;
      seed_pending_r  <= 1'b0;
      seed_src_bank_r <= 1'b0;
      fill_cols_A_r   <= 5'd18;
      fill_cols_B_r   <= 5'd18;
    end else if (frame_rearm) begin
      x_pos_r         <= 7'd0;
      fill_base_x_r   <= 7'd0;
      fill_cols_r     <= 5'd0;
      wr_bank_r       <= 1'b0;
      ready_A_r       <= 1'b0;
```

```
      ready_B_r       <= 1'b0;
      valid_rows_A_r  <= 5'd0;
      valid_rows_B_r  <= 5'd0;
      seed_pending_r  <= 1'b0;
      seed_src_bank_r <= 1'b0;
      fill_cols_A_r   <= 5'd18;
      fill_cols_B_r   <= 5'd18;
    end else begin
      if (consume_ready_bank) begin
        if (consume_bank_sel == 1'b0)
          ready_A_r <= 1'b0;
        else
          ready_B_r <= 1'b0;
      end

      if (seed_pending_r && !pix3_valid && !input_stall) begin
        copy_overlap_cols(wr_bank_r, seed_src_bank_r);
        fill_cols_r    <= 5'd2;
        seed_pending_r <= 1'b0;
      end else if (pix3_valid && !input_stall) begin
        if (seed_pending_r) begin
          copy_overlap_cols(wr_bank_r, seed_src_bank_r);
          fill_cols_r    <= 5'd2;
          seed_pending_r <= 1'b0;
        end

        col_off_i = x_pos_r - fill_base_x_r;
        if (stripe_window_open) begin
          write_input_column(wr_bank_r, col_off_i);
          fill_cols_r <= col_off_i[4:0] + 5'd1;
        end

        if (x_pos_r == eff_w - 7'd1) begin
          if (fill_base_x_r < eff_out_w) begin
            real_cols_i = x_pos_r - fill_base_x_r + 1;
            // zero_fill_tail REMOVED - read-side fill_cols mask handles it
            new_valid_rows = calc_valid_rows(fill_base_x_r, eff_out_w);
            mark_ready_bank(wr_bank_r, new_valid_rows, real_cols_i[4:0]);
            wr_bank_r <= ~wr_bank_r;
          end

          x_pos_r         <= 7'd0;
          fill_base_x_r   <= 7'd0;
          fill_cols_r     <= 5'd0;
          seed_pending_r  <= 1'b0;
        end else if (x_pos_r == fill_base_x_r + 7'd17) begin
          new_valid_rows = calc_valid_rows(fill_base_x_r, eff_out_w);
          mark_ready_bank(wr_bank_r, new_valid_rows, 5'd18);

          seed_src_bank_r <= wr_bank_r;
          wr_bank_r       <= ~wr_bank_r;
          fill_base_x_r   <= fill_base_x_r + STRIPE_STEP[6:0];
          fill_cols_r     <= 5'd0;
          seed_pending_r  <= ((fill_base_x_r + STRIPE_STEP[6:0]) < eff_out_w);
```

```
          x_pos_r         <= x_pos_r + 7'd1;
        end else begin
          x_pos_r <= x_pos_r + 7'd1;
        end
      end
    end
  end

endmodule
```

A.8.4 Line Buffer

Implements the layer-dependent row delay needed to assemble 3-row convolution windows.

code/input/RTL/conv_core/Line_Buffer.v

```
`timescale 1ns/1ns

// Line_Buffer: runtime-compatible version.
// Fixed 124-byte shift register (992 bits).
// layer_sel selects effective depth and stride:
//   L1 (2'b00): stride=1 byte,  depth=64 -> tap at byte 63
//   L2 (2'b01): stride=4 bytes, depth=31 -> tap at byte 120..123
//   L3 (2'b10): stride=8 bytes, depth=14 -> tap at byte 104..111
module Line_Buffer #(
    parameter integer W  = 64,
    parameter integer DW = 64   // max pixel bit-width
)(
    input  wire        [1:0]  layer_sel,
    input  wire               clk,
    input  wire               rst_n,
    input  wire               frame_rearm,// clears the shift register and fill counter
    input  wire               in_valid,
    input  wire signed [DW-1:0] in_data,
    output wire               out_valid,
    output wire signed [DW-1:0] out_data// selected shift-register tap for layer_sel
);

    localparam integer N = 124;  // total bytes

    reg [7:0] shreg [0:N-1];

    // Runtime parameters
    reg [6:0] eff_w;        // effective depth (pixels)
    reg [3:0] eff_stride;   // bytes per pixel
    always @* begin
      case (layer_sel)
        2'b01:   begin eff_w = 7'd31; eff_stride = 4'd4; end//L2
        2'b10:   begin eff_w = 7'd14; eff_stride = 4'd8; end//L3
        default: begin eff_w = 7'd64; eff_stride = 4'd1; end//L1
      endcase
```

```
    end

    // Fill counter
    reg [6:0] fill_cnt;
    assign out_valid = in_valid && (fill_cnt == eff_w);

    // Output tap: read eff_stride bytes from the oldest position.
    // Use byte-level MUX to avoid negative replication for any DW.
    reg signed [DW-1:0] out_r;
    integer ob;
    always @* begin
      out_r = {DW{1'b0}};
      case (layer_sel)
        2'b01: begin // L2: 4 bytes at [120..123]
          for (ob = 0; ob < DW/8 && ob < 4; ob = ob + 1)
            out_r[ob*8 +: 8] = shreg[120 + ob];
        end
        2'b10: begin // L3: 8 bytes at [104..111]
          for (ob = 0; ob < DW/8 && ob < 8; ob = ob + 1)
            out_r[ob*8 +: 8] = shreg[104 + ob];
        end
        default: begin // L1: 1 byte at [63]
          out_r[7:0] = shreg[63];
        end
      endcase
    end
    assign out_data = out_r;

    // Shift register: on each in_valid, shift by eff_stride bytes.
    integer i;
    always @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
        fill_cnt <= 7'd0;
        for (i = 0; i < N; i = i + 1)
          shreg[i] <= 8'd0;
      end else if (frame_rearm) begin
        fill_cnt <= 7'd0;
        for (i = 0; i < N; i = i + 1)
          shreg[i] <= 8'd0;
      end else if (in_valid) begin
        case (eff_stride)
          4'd4: begin
            for (i = N-1; i >= 4; i = i - 1) shreg[i] <= shreg[i-4];
            shreg[3] <= in_data[31:24];
            shreg[2] <= in_data[23:16];
            shreg[1] <= in_data[15:8];
            shreg[0] <= in_data[7:0];
          end
          4'd8: begin
            for (i = N-1; i >= 8; i = i - 1) shreg[i] <= shreg[i-8];
            for (ob = 0; ob < 8; ob = ob + 1) begin
              if (ob < DW/8)
                shreg[ob] <= in_data[ob*8 +: 8];
              else
```

```
                shreg[ob] <= 8'd0;
            end
          end
          default: begin  // stride=1
            for (i = N-1; i >= 1; i = i - 1) shreg[i] <= shreg[i-1];
            shreg[0] <= in_data[7:0];
          end
        endcase

        if (fill_cnt < eff_w)
          fill_cnt <= fill_cnt + 7'd1;
      end
    end

endmodule
```

A.8.5 Input Row Aligner

Unpacks and aligns incoming pixels into the row streams consumed by the convolution buffer.

code/input/RTL/conv_core/input_row_aligner.v

```
`timescale 1ns/1ns

module input_row_aligner #(
    parameter integer W    = 64,
    parameter integer DW   = 8,
    parameter integer C_IN = 8
)(
    input  wire                      [1:0]         layer_sel,
    input  wire                                     clk,
    input  wire                                     rst_n,
    input  wire                                     frame_rearm,
    input  wire                                     in_valid,
    input  wire                      [31:0]         in_data,
    input  wire                      [3:0]          in_byte_en,
    output wire                                     pix3_valid,
    output wire signed              [C_IN*DW-1:0] row0_out,
    output wire signed              [C_IN*DW-1:0] row1_out,
    output wire signed              [C_IN*DW-1:0] row2_out
);

    localparam integer PIX_W = C_IN * DW;

    reg        [31:0]           saved_lo;
    reg                          half_r;
    reg                          ser_valid_r;
    reg signed [PIX_W-1:0]       ser_data_r;

    wire [31:0] in_masked = { in_byte_en[3] ? in_data[31:24] : 8'd0,
                              in_byte_en[2] ? in_data[23:16] : 8'd0,
```

```
                              in_byte_en[1] ? in_data[15:8]  : 8'd0,
                              in_byte_en[0] ? in_data[7:0]   : 8'd0 };

    integer pb;
    always @* begin
      ser_valid_r = 1'b0;
      ser_data_r  = {PIX_W{1'b0}};
      case (layer_sel)
        2'b00: begin
          ser_valid_r   = in_valid;
          ser_data_r[7:0] = in_masked[7:0];
        end
        2'b01: begin
          ser_valid_r = in_valid;
          for (pb = 0; pb < 4 && pb < PIX_W/8; pb = pb + 1)
            ser_data_r[pb*8 +: 8] = in_masked[pb*8 +: 8];
        end
        default: begin
          ser_valid_r = in_valid && half_r;
          for (pb = 0; pb < 4 && pb < PIX_W/8; pb = pb + 1)
            ser_data_r[pb*8 +: 8] = saved_lo[pb*8 +: 8];
          for (pb = 4; pb < 8 && pb < PIX_W/8; pb = pb + 1)
            ser_data_r[pb*8 +: 8] = in_masked[(pb-4)*8 +: 8];
        end
      endcase
    end

    always @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
        saved_lo <= 32'd0;
        half_r   <= 1'b0;
      end else if (frame_rearm) begin
        saved_lo <= 32'd0;
        half_r   <= 1'b0;
      end else if (in_valid) begin
        case (layer_sel)
          2'b10: begin
            if (!half_r)
              saved_lo <= in_masked;
            half_r <= ~half_r;
          end
          default: begin
            half_r <= 1'b0;
          end
        endcase
      end
    end

    wire signed [PIX_W-1:0] row0;
    wire signed [PIX_W-1:0] row1;
    wire signed [PIX_W-1:0] row2;
    wire row0_valid;
    wire row1_valid;
    wire row2_valid;
```

```
84
85      assign row0       = ser_data_r;
86      assign row0_valid = ser_valid_r;
87
88      Line_Buffer #(
89          .W(W),
90          .DW(PIX_W)
91      ) Line_Buffer_row1 (
92          .layer_sel  (layer_sel),
93          .clk        (clk),
94          .rst_n      (rst_n),
95          .frame_rearm(frame_rearm),
96          .in_valid   (row0_valid),
97          .in_data    (row0),
98          .out_valid  (row1_valid),
99          .out_data   (row1)
100     );
101
102     Line_Buffer #(
103         .W(W),
104         .DW(PIX_W)
105     ) Line_Buffer_row2 (
106         .layer_sel  (layer_sel),
107         .clk        (clk),
108         .rst_n      (rst_n),
109         .frame_rearm(frame_rearm),
110         .in_valid   (row1_valid),
111         .in_data    (row1),
112         .out_valid  (row2_valid),
113         .out_data   (row2)
114     );
115
116     assign pix3_valid = row2_valid & row1_valid & row0_valid;
117     assign row0_out   = row0;
118     assign row1_out   = row1;
119     assign row2_out   = row2;
120
121 endmodule
```

A.8.6 Systolic-Array Skew Feeder

Applies row-wise skewing so activation rows arrive at the systolic array with the required timing.

code/input/RTL/conv_core/sa_skew_feeder.v

```
1  module sa_skew_feeder #(
2      parameter integer ROWS = 16
3  )(
4      input  wire                clk,
5      input  wire                rst_n,
6      input  wire                frame_rearm,
```

```
    input  wire                 rd_en,
    input  wire signed [ROWS*8-1:0]    in_firstcol_flat,
    output wire signed [ROWS*8-1:0]    out_firstcol_flat
);

  // Skew-only stage: row r is delayed by r cycles to create the systolic
  // wavefront. The large full-window banks now live in Conv_Buffer instead.
  reg signed [7:0] delay [0:119];
  reg       rd_en_d;

  function integer base;
    input integer r;
    begin
      base = (r * (r - 1)) / 2;
    end
  endfunction

  wire rd_start = rd_en & ~rd_en_d;

  wire signed [7:0] raw0  = in_firstcol_flat[7:0];
  wire signed [7:0] raw1  = in_firstcol_flat[15:8];
  wire signed [7:0] raw2  = in_firstcol_flat[23:16];
  wire signed [7:0] raw3  = in_firstcol_flat[31:24];
  wire signed [7:0] raw4  = in_firstcol_flat[39:32];
  wire signed [7:0] raw5  = in_firstcol_flat[47:40];
  wire signed [7:0] raw6  = in_firstcol_flat[55:48];
  wire signed [7:0] raw7  = in_firstcol_flat[63:56];
  wire signed [7:0] raw8  = in_firstcol_flat[71:64];
  wire signed [7:0] raw9  = in_firstcol_flat[79:72];
  wire signed [7:0] raw10 = in_firstcol_flat[87:80];
  wire signed [7:0] raw11 = in_firstcol_flat[95:88];
  wire signed [7:0] raw12 = in_firstcol_flat[103:96];
  wire signed [7:0] raw13 = in_firstcol_flat[111:104];
  wire signed [7:0] raw14 = in_firstcol_flat[119:112];
  wire signed [7:0] raw15 = in_firstcol_flat[127:120];

  integer x;
  integer s;
  integer b;
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      rd_en_d <= 1'b0;
      for (x = 0; x < 120; x = x + 1)
        delay[x] <= 8'h00;
    end else if (frame_rearm) begin
      rd_en_d <= 1'b0;
      for (x = 0; x < 120; x = x + 1)
        delay[x] <= 8'h00;
    end else begin
      rd_en_d <= rd_en;
      if (rd_en) begin
        for (x = 1; x <= 15; x = x + 1) begin
          b = base(x);
          if (rd_start) begin
```

```
            for (s = x - 1; s >= 1; s = s - 1)
              delay[b + s] <= 8'h00;
          end else begin
            for (s = x - 1; s >= 1; s = s - 1)
              delay[b + s] <= delay[b + s - 1];
          end

          case (x)
            1:  delay[b] <= raw1;
            2:  delay[b] <= raw2;
            3:  delay[b] <= raw3;
            4:  delay[b] <= raw4;
            5:  delay[b] <= raw5;
            6:  delay[b] <= raw6;
            7:  delay[b] <= raw7;
            8:  delay[b] <= raw8;
            9:  delay[b] <= raw9;
            10: delay[b] <= raw10;
            11: delay[b] <= raw11;
            12: delay[b] <= raw12;
            13: delay[b] <= raw13;
            14: delay[b] <= raw14;
            15: delay[b] <= raw15;
            default: delay[b] <= 8'h00;
          endcase
        end
      end
    end
  end

  wire signed [7:0] out0  = rd_en ? raw0 : 8'h00;
  wire signed [7:0] out1  = delay[base(1)  + 0];
  wire signed [7:0] out2  = delay[base(2)  + 1];
  wire signed [7:0] out3  = delay[base(3)  + 2];
  wire signed [7:0] out4  = delay[base(4)  + 3];
  wire signed [7:0] out5  = delay[base(5)  + 4];
  wire signed [7:0] out6  = delay[base(6)  + 5];
  wire signed [7:0] out7  = delay[base(7)  + 6];
  wire signed [7:0] out8  = delay[base(8)  + 7];
  wire signed [7:0] out9  = delay[base(9)  + 8];
  wire signed [7:0] out10 = delay[base(10) + 9];
  wire signed [7:0] out11 = delay[base(11) + 10];
  wire signed [7:0] out12 = delay[base(12) + 11];
  wire signed [7:0] out13 = delay[base(13) + 12];
  wire signed [7:0] out14 = delay[base(14) + 13];
  wire signed [7:0] out15 = delay[base(15) + 14];

  assign out_firstcol_flat = {
      out15, out14, out13, out12, out11, out10, out9, out8,
      out7,  out6,  out5,  out4,  out3,  out2,  out1, out0
  };

endmodule
```

A.8.7 Systolic Array Top

Implements the 16-by-4 MAC array and per-column output stream used by convolution and FC-compatible modes.

code/input/RTL/conv_core/systolic_array_top.v

```
// ============================================================
// 16x4 systolic array top.
// - Keeps the flat input, flat output, and column-stream interface.
// - Uses mode_cfg and runtime cur_k for layer-dependent MAC length.
// - Conv1-compatible mode is selected with mode_cfg=2'b00.
// ============================================================

module systolic_array_top #(
  parameter integer DATA_W = 8,
  parameter integer ROWS   = 16,
  parameter integer COLS   = 4,
  parameter integer DOT_K  = 9,   // legacy compatibility only
  parameter integer ACC_W  = 23
)(
  input  wire clk,
  input  wire rst_n,
  input  wire start_pulse,
  input  wire [1:0] mode_cfg,
  input  wire [4:0] valid_rows_cfg,

  // Flat inputs:
  // a_in_flat[(r+1)*DATA_W-1 : r*DATA_W] is the left-edge input for row r.
  input  wire signed [ROWS*DATA_W-1:0] a_in_flat,
  // b_in_flat[(c+1)*DATA_W-1 : c*DATA_W] is the top-edge input for column c.
  input  wire signed [COLS*DATA_W-1:0] b_in_flat,

  // Flat outputs:
  // idx = r*COLS + c
  // c_out_flat[(idx+1)*ACC_W-1 : idx*ACC_W] maps to c_out[r][c].
  output wire signed [ROWS*COLS*ACC_W-1:0] c_out_flat,
  output wire done,

  // Per-column real-time stream output.
  output reg  signed [COLS*ACC_W-1:0] col_stream_data_flat,
  output reg         [COLS-1:0]       col_stream_valid,
  output reg         [COLS-1:0]       col_stream_last
);

  localparam [1:0] MODE_CONV1 = 2'b00;
  localparam [1:0] MODE_CONV2 = 2'b01;
  localparam [1:0] MODE_CONV3 = 2'b10;
  localparam [1:0] MODE_FC    = 2'b11;
  localparam integer MAX_DOT_K    = 288;
  localparam integer CNT_W        = $clog2(MAX_DOT_K + 1);
  localparam integer START_STAGES = ROWS + COLS - 1;

```

```
  integer k;
  integer sc;
  integer data_lo;

  reg  [START_STAGES-1:0] start_pipe;
  reg  [CNT_W-1:0]        cur_k;
  wire                    fc_mode;

  reg  [COLS-1:0] stream_active;
  reg  [COLS-1:0] stream_arm;
  reg  [7:0]      stream_row [0:COLS-1];
  reg  [4:0]      stream_rows_limit [0:COLS-1];
  reg  [4:0]      arm_rows_limit [0:COLS-1];
  reg  [4:0]      col_rows_cfg [0:COLS-1];
  reg  [4:0]      valid_rows_pipe [0:COLS-1];
  reg             valid_rows_pipe_v [0:COLS-1];

  wire signed [DATA_W-1:0] a_in [0:ROWS-1];
  wire signed [DATA_W-1:0] b_in [0:COLS-1];
  wire                    start_ij [0:ROWS-1][0:COLS-1];
  wire                    finish_ij [0:ROWS-1][0:COLS-1];

  wire signed [DATA_W-1:0] pe_left   [0:ROWS-1][0:COLS-1];
  wire signed [DATA_W-1:0] pe_up     [0:ROWS-1][0:COLS-1];
  wire signed [DATA_W-1:0] pe_right  [0:ROWS-1][0:COLS-1];
  wire signed [DATA_W-1:0] pe_bottom [0:ROWS-1][0:COLS-1];
  wire signed [ACC_W-1:0]  pe_result [0:ROWS-1][0:COLS-1];
  reg                      conv_done_r;

  assign fc_mode = (mode_cfg == MODE_FC);

  always @(*) begin
    case (mode_cfg)
      MODE_CONV1: cur_k = 9;
      MODE_CONV2: cur_k = 36;
      MODE_CONV3: cur_k = 72;
      default:    cur_k = 288;
    endcase
  end

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      start_pipe <= {START_STAGES{1'b0}};
    end else begin
      start_pipe[0] <= start_pulse;
      for (k=1; k<START_STAGES; k=k+1) begin
        start_pipe[k] <= start_pipe[k-1];
      end
    end
  end

  // FC mode uses row 0 PEs; convolution mode completes on the active block rows.
  always @* begin
    case (valid_rows_cfg)
```

```
      5'd0:    conv_done_r = 1'b0;
      5'd1:    conv_done_r = finish_ij[0][COLS-1];
      5'd2:    conv_done_r = finish_ij[1][COLS-1];
      5'd3:    conv_done_r = finish_ij[2][COLS-1];
      5'd4:    conv_done_r = finish_ij[3][COLS-1];
      5'd5:    conv_done_r = finish_ij[4][COLS-1];
      5'd6:    conv_done_r = finish_ij[5][COLS-1];
      5'd7:    conv_done_r = finish_ij[6][COLS-1];
      5'd8:    conv_done_r = finish_ij[7][COLS-1];
      5'd9:    conv_done_r = finish_ij[8][COLS-1];
      5'd10:   conv_done_r = finish_ij[9][COLS-1];
      5'd11:   conv_done_r = finish_ij[10][COLS-1];
      5'd12:   conv_done_r = finish_ij[11][COLS-1];
      5'd13:   conv_done_r = finish_ij[12][COLS-1];
      5'd14:   conv_done_r = finish_ij[13][COLS-1];
      5'd15:   conv_done_r = finish_ij[14][COLS-1];
      default: conv_done_r = finish_ij[15][COLS-1];
    endcase
  end
  assign done = fc_mode ? finish_ij[0][2] : conv_done_r;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      col_stream_data_flat <= {(COLS*ACC_W){1'b0}};
      col_stream_valid     <= {COLS{1'b0}};
      col_stream_last      <= {COLS{1'b0}};
      stream_active        <= {COLS{1'b0}};
      stream_arm           <= {COLS{1'b0}};
      for (sc=0; sc<COLS; sc=sc+1) begin
        stream_row[sc] <= 8'd0;
        stream_rows_limit[sc] <= 5'd0;
        arm_rows_limit[sc]    <= 5'd0;
        col_rows_cfg[sc]      <= 5'd0;
        valid_rows_pipe[sc]   <= 5'd0;
        valid_rows_pipe_v[sc] <= 1'b0;
      end
    end else begin
      col_stream_valid <= {COLS{1'b0}};
      col_stream_last  <= {COLS{1'b0}};
      for (sc=0; sc<COLS; sc=sc+1) begin
        data_lo = sc*ACC_W;
        col_stream_data_flat[data_lo +: ACC_W] <= {ACC_W{1'b0}};
      end

      valid_rows_pipe[0]   <= valid_rows_cfg;
      valid_rows_pipe_v[0] <= start_pulse;
      for (sc=1; sc<COLS; sc=sc+1) begin
        valid_rows_pipe[sc]   <= valid_rows_pipe[sc-1];
        valid_rows_pipe_v[sc] <= valid_rows_pipe_v[sc-1];
      end

      if (start_pulse)
        col_rows_cfg[0] <= valid_rows_cfg;
      for (sc=1; sc<COLS; sc=sc+1) begin
```

```
        if (valid_rows_pipe_v[sc-1])
          col_rows_cfg[sc] <= valid_rows_pipe[sc-1];
      end

      for (sc=0; sc<COLS; sc=sc+1) begin
        if (!stream_arm[sc] && finish_ij[0][sc]) begin
          stream_arm[sc] <= 1'b1;
          arm_rows_limit[sc] <= col_rows_cfg[sc];
        end

        data_lo = sc*ACC_W;
        if (!stream_active[sc] && stream_arm[sc]) begin
          stream_arm[sc] <= 1'b0;
          stream_rows_limit[sc] <= arm_rows_limit[sc];
          if (arm_rows_limit[sc] != 5'd0) begin
            col_stream_valid[sc] <= 1'b1;
            col_stream_last[sc]  <= (arm_rows_limit[sc] == 5'd1);
            col_stream_data_flat[data_lo +: ACC_W] <= pe_result[0][sc];
          end

          if (arm_rows_limit[sc] <= 5'd1) begin
            stream_active[sc] <= 1'b0;
            stream_row[sc]    <= 8'd0;
          end else begin
            stream_active[sc] <= 1'b1;
            stream_row[sc]    <= 8'd1;
          end
        end else if (stream_active[sc]) begin
          col_stream_valid[sc] <= (stream_row[sc] < stream_rows_limit[sc]);
          if (stream_row[sc] < stream_rows_limit[sc])
            col_stream_data_flat[data_lo +: ACC_W] <= pe_result[stream_row[sc]][sc];

          if (stream_row[sc] == (stream_rows_limit[sc]-1'b1)) begin
            col_stream_last[sc] <= 1'b1;
            stream_active[sc]   <= 1'b0;
            stream_row[sc]      <= 8'd0;
          end else begin
            stream_row[sc]      <= stream_row[sc] + 8'd1;
          end
        end
      end
    end
  end

  genvar ar, bc;
  generate
    for (ar=0; ar<ROWS; ar=ar+1) begin: GEN_A_IN
      localparam integer A_LO = ar*DATA_W;
      localparam integer A_HI = ar*DATA_W + (DATA_W-1);
      assign a_in[ar] = a_in_flat[A_HI:A_LO];
    end
    for (bc=0; bc<COLS; bc=bc+1) begin: GEN_B_IN
      localparam integer B_LO = bc*DATA_W;
      localparam integer B_HI = bc*DATA_W + (DATA_W-1);
```

```
      assign b_in[bc] = b_in_flat[B_HI:B_LO];
    end
  endgenerate

  genvar r, c;
  generate
    for (r=0; r<ROWS; r=r+1) begin: GEN_ROW
      for (c=0; c<COLS; c=c+1) begin: GEN_COL
        localparam integer O_LO = (r*COLS + c)*ACC_W;
        localparam integer O_HI = (r*COLS + c)*ACC_W + (ACC_W-1);

        if (c == 0) begin: LEFT_EDGE
          assign pe_left[r][c] = a_in[r];
        end else begin: LEFT_INNER
          assign pe_left[r][c] = pe_right[r][c-1];
        end

        if (r == 0) begin: UP_EDGE
          assign pe_up[r][c] = b_in[c];
        end else begin: UP_INNER
          assign pe_up[r][c] = pe_bottom[r-1][c];
        end

        assign start_ij[r][c] = fc_mode ?
                                (((r == 0) && (c < 3)) ? ((c == 0) ? start_pulse : start_pipe[c-1]) : 1'
    b0) :
                                (((r == 0) && (c == 0)) ? start_pulse : start_pipe[r+c-1]);

        single_PE #(
          .DATA_W(DATA_W),
          .CNT_W (CNT_W),
          .ACC_W (ACC_W)
        ) PE_ij (
          .clk   (clk),
          .rst_n (rst_n),
          .start (start_ij[r][c]),
          .cur_k (cur_k),
          .finish(finish_ij[r][c]),
          .left  (pe_left[r][c]),
          .up    (pe_up[r][c]),
          .right (pe_right[r][c]),
          .bottom(pe_bottom[r][c]),
          .result(pe_result[r][c])
        );

        assign c_out_flat[O_HI:O_LO] = pe_result[r][c];
      end
    end
  endgenerate

endmodule


module single_PE #(
```

```
  parameter integer DATA_W = 8,
  parameter integer CNT_W  = 9,
  parameter integer ACC_W  = 23
)(
  input  wire clk,
  input  wire rst_n,
  input  wire start,
  input  wire [CNT_W-1:0] cur_k,
  output reg  finish,
  input  wire signed [DATA_W-1:0] left,
  input  wire signed [DATA_W-1:0] up,

  output reg  signed [DATA_W-1:0] right,
  output reg  signed [DATA_W-1:0] bottom,
  output reg  signed [ACC_W-1:0] result
);

  reg  signed [ACC_W-1:0]    mem;
  wire signed [2*DATA_W-1:0] product;
  reg  [CNT_W-1:0]           cnt;

  assign product = left * up;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      right  <= {DATA_W{1'b0}};
      bottom <= {DATA_W{1'b0}};
      mem    <= {ACC_W{1'b0}};
      result <= {ACC_W{1'b0}};
      finish <= 1'b0;
      cnt    <= {CNT_W{1'b0}};
    end else begin
      right  <= left;
      bottom <= up;
      finish <= 1'b0;

      if (start) begin
        if (cur_k <= 1) begin
          result <= {{(ACC_W-2*DATA_W){product[2*DATA_W-1]}}, product};
          mem    <= {ACC_W{1'b0}};
          cnt    <= {CNT_W{1'b0}};
          finish <= 1'b1;
        end else begin
          mem <= {{(ACC_W-2*DATA_W){product[2*DATA_W-1]}}, product};
          cnt <= {{(CNT_W-1){1'b0}}, 1'b1};
        end
      end else if (cnt != {CNT_W{1'b0}}) begin
        if (cnt == cur_k - 1'b1) begin
          result <= mem + {{(ACC_W-2*DATA_W){product[2*DATA_W-1]}}, product};
          mem    <= {ACC_W{1'b0}};
          cnt    <= {CNT_W{1'b0}};
          finish <= 1'b1;
        end else begin
          mem <= mem + {{(ACC_W-2*DATA_W){product[2*DATA_W-1]}}, product};
```

```
316             cnt <= cnt + 1'b1;
317           end
318         end
319       end
320     end
321
322 endmodule
```

A.8.8 Weight Buffer

Buffers and rotates convolution weights for the systolic-array columns.

code/input/RTL/conv_core/weight_buffer.v

```
1
2
3  module ring_shift_reg #(
4    parameter Length  = 75,   // max channel depth (DOT_K_MAX + 3)
5    parameter DATA_W  = 8,
6    parameter VALID_LEN = 72, // max valid window (DOT_K_MAX)
7    parameter PERIOD    = 75  // max period (= Length)
8  ) (
9    input wire [1:0] layer_sel,
10   input wire clk,
11   input wire rst_n,
12   input wire wr_en,
13   input wire signed [DATA_W-1:0] in_data,
14   input wire ring,
15   input wire out_ready,
16   output wire signed [DATA_W-1:0] out_data,
17   output wire out_valid
18 );
19
20 reg signed [DATA_W-1:0] sr [Length-1:0];
21 reg signed [DATA_W-1:0] saved_sr [Length-1:0];
22
23 // Runtime decode: effective length, valid_len, period per channel
24 // These are set by the parent weight_buffer based on which channel instance this is.
25 // However, since ring_shift_reg is generic, we decode from layer_sel + compile-time
26 // Length offset. The key relationship:
27 //   eff_len = eff_dot_k + (Length - VALID_LEN)  -- channel offset preserved
28 //   eff_valid = eff_dot_k
29 //   eff_period = eff_dot_k + 3  -- always equal to max channel length
30 reg [6:0] eff_dot_k;
31 reg [6:0] eff_len;
32 reg [6:0] eff_valid;
33 reg [6:0] eff_period;
34
35 localparam integer CH_OFFSET = Length - VALID_LEN;  // 0, 1, 2, or 3
36 localparam [6:0] LENGTH_U7 = Length;
37
```

```
always @* begin
  case (layer_sel)
    2'b01:   eff_dot_k = 7'd36;
    2'b10:   eff_dot_k = 7'd72;
    default: eff_dot_k = 7'd9;
  endcase
  eff_valid  = eff_dot_k;
  eff_len    = eff_dot_k + CH_OFFSET[6:0];
  eff_period = eff_dot_k + 7'd3;
end

reg [$clog2(PERIOD)-1:0] phase;
reg [$clog2(PERIOD+1)-1:0] burst_cnt;
reg burst_active;
reg ring_d;

localparam integer ALIGN_MAX = Length - VALID_LEN;
wire [6:0] eff_align = eff_len - eff_valid;
wire [6:0] feedback_idx = eff_len - 7'd1;
wire feedback_idx_valid = (eff_len != 7'd0) && (eff_len <= LENGTH_U7);
wire in_valid_window = (phase >= eff_align) && (phase < eff_align + eff_valid);
wire ring_rise = ring && !ring_d;
wire step_en = burst_active && (!in_valid_window || out_ready);

// Runtime MUX: read from the effective end of the ring, not the physical end.
// When DOT_K < Length (e.g. L1: eff_len=9, Length=72), the data lives in
// sr[0..eff_len-1] and never reaches sr[Length-1] within the burst period.
reg signed [DATA_W-1:0] out_tap;
integer oi;
always @* begin
  out_tap = 0;
  for (oi = 0; oi < Length; oi = oi + 1)
    if (oi[6:0] == eff_len - 7'd1)
      out_tap = sr[oi];
end
assign out_data = (burst_active && in_valid_window) ? out_tap : 0;
assign out_valid = burst_active && in_valid_window;

integer i;
always@(posedge clk or negedge rst_n)begin
  if(!rst_n)begin
    for(i=0;i<Length;i=i+1) begin
      sr[i] <= 0;
      saved_sr[i] <= 0;
    end
    phase <= 0;
    burst_cnt <= 0;
    burst_active <= 1'b0;
    ring_d <= 1'b0;
  end
  else begin
    ring_d <= ring;
    if(ring_rise) begin
      for (i=0; i<Length; i=i+1) saved_sr[i] <= sr[i];
```

```
      burst_active <= 1'b1;
      burst_cnt <= 0;
      phase <= 0;
    end
    else if(burst_active && step_en) begin
      if(burst_cnt == eff_period) begin
        for (i=0; i<Length; i=i+1) sr[i] <= saved_sr[i];
        phase <= 0;
        burst_cnt <= 0;
        burst_active <= 1'b0;
      end else begin
        // Runtime-select the feedback tap so smaller Length instances
        // never statically reference out-of-range sr[] entries.
        if (feedback_idx_valid) sr[0] <= sr[feedback_idx];
        else                    sr[0] <= 0;
        for (i=1; i<Length; i=i+1) sr[i] <= sr[i-1];
        phase <= phase + 1'b1;
        burst_cnt <= burst_cnt + 1'b1;
      end
    end
    else if(wr_en) begin
      sr[0] <= in_data;
      for (i=1; i<Length; i=i+1) sr[i] <= sr[i-1];
      phase <= 0;
      burst_cnt <= 0;
    end
  end
end
endmodule


module weight_buffer #(
  parameter integer DATA_W = 32,
  parameter integer COLS = 4,
  parameter integer L0 = 72, L1 = 73, L2 = 74, L3 = 75  // max depths
) (
  input wire [1:0] layer_sel,
  input wire clk,
  input wire rst_n,
  input wire ring,
  input wire wr_en,
  input wire signed [DATA_W-1:0] in_data,
  input wire out_ready,

  output wire signed [DATA_W/COLS-1:0] out0, out1, out2, out3,
  output wire [COLS-1:0] out_valid
);

  localparam integer CH_W = DATA_W/COLS;
  wire signed [CH_W-1:0] in0, in1, in2, in3;
  assign in0 = in_data[0*CH_W +: CH_W];
  assign in1 = in_data[1*CH_W +: CH_W];
  assign in2 = in_data[2*CH_W +: CH_W];
  assign in3 = in_data[3*CH_W +: CH_W];
```

```
146
147   ring_shift_reg #(.Length(L0), .DATA_W(CH_W), .VALID_LEN(L0), .PERIOD(L3))
148     u0_ring_shift_reg (
149       .layer_sel(layer_sel), .clk(clk), .rst_n(rst_n),
150       .wr_en(wr_en), .in_data(in0), .ring(ring), .out_ready(out_ready),
151       .out_data(out0), .out_valid(out_valid[0])
152     );
153
154   ring_shift_reg #(.Length(L1), .DATA_W(CH_W), .VALID_LEN(L0), .PERIOD(L3))
155     u1_ring_shift_reg (
156       .layer_sel(layer_sel), .clk(clk), .rst_n(rst_n),
157       .wr_en(wr_en), .in_data(in1), .ring(ring), .out_ready(out_ready),
158       .out_data(out1), .out_valid(out_valid[1])
159     );
160
161   ring_shift_reg #(.Length(L2), .DATA_W(CH_W), .VALID_LEN(L0), .PERIOD(L3))
162     u2_ring_shift_reg (
163       .layer_sel(layer_sel), .clk(clk), .rst_n(rst_n),
164       .wr_en(wr_en), .in_data(in2), .ring(ring), .out_ready(out_ready),
165       .out_data(out2), .out_valid(out_valid[2])
166     );
167
168   ring_shift_reg #(.Length(L3), .DATA_W(CH_W), .VALID_LEN(L0), .PERIOD(L3))
169     u3_ring_shift_reg (
170       .layer_sel(layer_sel), .clk(clk), .rst_n(rst_n),
171       .wr_en(wr_en), .in_data(in3), .ring(ring), .out_ready(out_ready),
172       .out_data(out3), .out_valid(out_valid[3])
173     );
174
175 endmodule
```

A.9 Quantization, Pooling, And FC RTL

A.9.1 Convolution-Quantization-Pooling Wrapper

Connects convolution output streams to quantization, pooling, and SRAM writeback flow control.

code/input/RTL/conv_core/conv_quant_pool.v

```
1  module conv_quant_pool #(
2    parameter integer W          = 64,
3    parameter integer DATA_W     = 8,
4    parameter integer ROWS       = 16,
5    parameter integer COLS       = 4,
6    parameter integer DOT_K      = 72,   // max (L3: 9*8)
7    parameter integer ACC_W      = 23,
8    parameter integer IMG_H      = 64,
9    parameter integer IMG_W      = 64,
10   parameter integer C_IN       = 8,    // max (L3)
11   parameter integer OUT_H      = IMG_H - 2,
```

```
  parameter integer OUT_W      = IMG_W - 2,
  parameter integer POOL_DEPTH = 31    // max (L1)
)(
  input  wire [1:0]                layer_sel,
  input  wire                      clk,
  input  wire                      rst_n,
  input  wire                      in_valid,
  input  wire        [31:0]        in_data,
  input  wire        [3:0]         in_byte_en,
  input  wire                      in_last,
  output wire                      in_ready,
  input  wire                      wt_valid,
  output wire                      wt_ready,
  input  wire                      wt_last,
  input  wire signed [COLS*DATA_W-1:0] wt_data,
  input  wire                      cfg_valid,
  output wire                      cfg_ready,
  input  wire [31:0]               cfg_data,
  input  wire                      cfg_last,
  input  wire                      pool_ready,
  output wire signed [31:0]        pool_data,
  output wire                      pool_valid,
  output wire                      pool_last,
  // Control observation signals for FSM integration
  output wire                      cfg_load_done,
  output wire                      wt_load_done,
  output wire                      pool_frame_done,
  output wire                      conv_frame_rearm_out
);

  wire                          sa_done;
  wire signed [COLS*ACC_W-1:0] c_out_col_stream_flat;
  wire [COLS-1:0]               c_out_col_valid;
  wire [COLS-1:0]               c_out_col_last;
  wire                          conv_frame_rearm;
  wire                          conv_frame_done;

  wire signed [ACC_W-1:0]      qp_rso0;
  wire signed [ACC_W-1:0]      qp_rso1;
  wire signed [ACC_W-1:0]      qp_rso2;
  wire signed [ACC_W-1:0]      qp_rso3;
  wire                          in_valid1;
  wire                          in_valid2;
  wire                          in_valid3;
  wire                          in_valid4;
  wire                          start1;
  wire                          start2;
  wire                          start3;
  wire                          start4;

  wire signed [31:0]           bias_in_w;
  wire signed [31:0]           M_in_w;
  wire [31:0]                  sh_in_w;
  wire                          load_bias1;
```

```
    wire                        load_bias2;
    wire                        load_bias3;
    wire                        load_bias4;
    wire                        load_M1;
    wire                        load_M2;
    wire                        load_M3;
    wire                        load_M4;
    wire                        load_sh;

    wire signed [7:0]           cut1;
    wire signed [7:0]           cut2;
    wire signed [7:0]           cut3;
    wire signed [7:0]           cut4;
    wire                        cut_valid1;
    wire                        cut_valid2;
    wire                        cut_valid3;
    wire                        cut_valid4;

    wire signed [7:0]           cut1_out;
    wire signed [7:0]           cut2_out;
    wire signed [7:0]           cut3_out;
    wire signed [7:0]           cut4_out;
    wire                        cut_valid1_out;
    wire                        cut_valid2_out;
    wire                        cut_valid3_out;
    wire                        cut_valid4_out;

    conv_top #(
      .W(W),
      .DATA_W(DATA_W),
      .ROWS(ROWS),
      .COLS(COLS),
      .DOT_K(DOT_K),
      .ACC_W(ACC_W),
      .IMG_H(IMG_H),
      .IMG_W(IMG_W),
      .C_IN(C_IN)
    ) u_conv1 (
      .layer_sel(layer_sel),
      .clk(clk),
      .rst_n(rst_n),
      .in_valid(in_valid),
      .in_data(in_data),
      .in_byte_en(in_byte_en),
      .in_last(in_last),
      .in_ready(in_ready),
      .wt_valid(wt_valid),
      .wt_ready(wt_ready),
      .wt_last(wt_last),
      .wt_data(wt_data),
      .wt_load_done(wt_load_done),
      .sa_done(sa_done),
      .c_out_col_stream_flat(c_out_col_stream_flat),
      .c_out_col_valid(c_out_col_valid),
```

```
    .c_out_col_last(c_out_col_last),
    .frame_rearm_out(conv_frame_rearm),
    .frame_done_out(conv_frame_done)
  );

  conv_quant_adapter #(
    .ACC_W(ACC_W),
    .COLS(COLS),
    .ROWS(ROWS)
  ) u_conv_quant_adapter (
    .clk(clk),
    .rst_n(rst_n),
    .c_out_col_stream_flat(c_out_col_stream_flat),
    .c_out_col_valid(c_out_col_valid),
    .c_out_col_last(c_out_col_last),
    .sa_done(sa_done),
    .qp_rso0(qp_rso0),
    .qp_rso1(qp_rso1),
    .qp_rso2(qp_rso2),
    .qp_rso3(qp_rso3),
    .in_valid1(in_valid1),
    .in_valid2(in_valid2),
    .in_valid3(in_valid3),
    .in_valid4(in_valid4),
    .start1(start1),
    .start2(start2),
    .start3(start3),
    .start4(start4),
    .en1(),
    .en2(),
    .en3(),
    .en4(),
    .bias_in(),
    .M_in(),
    .sh_in(),
    .load_bias1(),
    .load_bias2(),
    .load_bias3(),
    .load_bias4(),
    .load_M1(),
    .load_M2(),
    .load_M3(),
    .load_M4(),
    .load_sh(),
    .param_bias0(32'sd0),
    .param_bias1(32'sd0),
    .param_bias2(32'sd0),
    .param_bias3(32'sd0),
    .param_M0(32'sd0),
    .param_M1(32'sd0),
    .param_M2(32'sd0),
    .param_M3(32'sd0),
    .param_sh0(8'd0),
    .param_sh1(8'd0),
```

```
    .param_sh2(8'd0),
    .param_sh3(8'd0),
    .param_load_start(1'b0),
    .param_load_done()
  );

  quant_param_loader u_quant_param_loader (
    .clk(clk),
    .rst_n(rst_n),
    .cfg_valid(cfg_valid),
    .cfg_ready(cfg_ready),
    .cfg_data(cfg_data),
    .cfg_last(cfg_last),
    .param_load_done(cfg_load_done),
    .bias_in(bias_in_w),
    .M_in(M_in_w),
    .sh_in(sh_in_w),
    .load_bias1(load_bias1),
    .load_bias2(load_bias2),
    .load_bias3(load_bias3),
    .load_bias4(load_bias4),
    .load_M1(load_M1),
    .load_M2(load_M2),
    .load_M3(load_M3),
    .load_M4(load_M4),
    .load_sh(load_sh)
  );

  Quantization_Top u_quant (
    .clk(clk),
    .rst_n(rst_n),
    .start1(start1),
    .start2(start2),
    .start3(start3),
    .start4(start4),
    .in_valid1(in_valid1),
    .in_valid2(in_valid2),
    .in_valid3(in_valid3),
    .in_valid4(in_valid4),
    .load_bias1(load_bias1),
    .load_bias2(load_bias2),
    .load_bias3(load_bias3),
    .load_bias4(load_bias4),
    .load_M1(load_M1),
    .load_M2(load_M2),
    .load_M3(load_M3),
    .load_M4(load_M4),
    .load_sh(load_sh),
    .bias_in(bias_in_w),
    .M_in(M_in_w),
    .sh_in(sh_in_w),
    .rso0(qp_rso0),
    .rso1(qp_rso1),
    .rso2(qp_rso2),
```

```
    .rso3(qp_rso3),
    .cut1(cut1),
    .cut2(cut2),
    .cut3(cut3),
    .cut4(cut4),
    .cut_valid1(cut_valid1),
    .cut_valid2(cut_valid2),
    .cut_valid3(cut_valid3),
    .cut_valid4(cut_valid4)
  );

  quant_pool_adapter u_quant_pool_adapter (
    .clk(clk),
    .rst_n(rst_n),
    .cut1_in(cut1),
    .cut2_in(cut2),
    .cut3_in(cut3),
    .cut4_in(cut4),
    .cut_valid1(cut_valid1),
    .cut_valid2(cut_valid2),
    .cut_valid3(cut_valid3),
    .cut_valid4(cut_valid4),
    .cut1_out(cut1_out),
    .cut2_out(cut2_out),
    .cut3_out(cut3_out),
    .cut4_out(cut4_out),
    .cut_valid1_out(cut_valid1_out),
    .cut_valid2_out(cut_valid2_out),
    .cut_valid3_out(cut_valid3_out),
    .cut_valid4_out(cut_valid4_out)
  );

  pool_stream_top #(
    .POOL_DEPTH(POOL_DEPTH)
  ) u_pool (
    .layer_sel(layer_sel),
    .clk(clk),
    .rst_n(rst_n),
    .cut1(cut1_out),
    .cut2(cut2_out),
    .cut3(cut3_out),
    .cut4(cut4_out),
    .cut_valid1(cut_valid1_out),
    .cut_valid2(cut_valid2_out),
    .cut_valid3(cut_valid3_out),
    .cut_valid4(cut_valid4_out),
    .frame_rearm(conv_frame_rearm),
    .pool_ready(pool_ready),
    .pool_data(pool_data),
    .pool_valid(pool_valid),
    .pool_last(pool_last),
    .pool_frame_done(pool_frame_done)
  );

```

```
  assign conv_frame_rearm_out = conv_frame_rearm;

endmodule
```

A.9.2 Convolution-Quantization Adapter

Adapts convolution stream timing and layout into the quantization block interface.

code/input/RTL/quant_pool/integration/conv_quant_adapter.v

```
module conv_quant_adapter #(
  parameter integer ACC_W = 23,
  parameter integer COLS  = 4,
  parameter integer ROWS  = 16
)(
  input  wire clk,
  input  wire rst_n,

  // From Conv1_top
  input  wire signed [COLS*ACC_W-1:0] c_out_col_stream_flat,
  input  wire [COLS-1:0]              c_out_col_valid,
  input  wire [COLS-1:0]              c_out_col_last,
  input  wire                         sa_done,

  // To Quantization_Top
  output wire signed [ACC_W-1:0]      qp_rso0,
  output wire signed [ACC_W-1:0]      qp_rso1,
  output wire signed [ACC_W-1:0]      qp_rso2,
  output wire signed [ACC_W-1:0]      qp_rso3,
  output wire                         in_valid1,
  output wire                         in_valid2,
  output wire                         in_valid3,
  output wire                         in_valid4,
  output reg                          start1,
  output reg                          start2,
  output reg                          start3,
  output reg                          start4,
  // Deprecated: en1..4 are not consumed by any active-path module.
  // Retained only for legacy TB compatibility; do not rely on in new logic.
  output wire                         en1,
  output wire                         en2,
  output wire                         en3,
  output wire                         en4,

  // Quant parameter loading
  output wire signed [31:0]           bias_in,
  output wire signed [31:0]           M_in,
  output wire        [31:0]           sh_in,
  output wire                         load_bias1,
  output wire                         load_bias2,
  output wire                         load_bias3,
```

```
    output wire                          load_bias4,
    output wire                          load_M1,
    output wire                          load_M2,
    output wire                          load_M3,
    output wire                          load_M4,
    output wire                          load_sh,
    input  wire signed [31:0]            param_bias0,
    input  wire signed [31:0]            param_bias1,
    input  wire signed [31:0]            param_bias2,
    input  wire signed [31:0]            param_bias3,
    input  wire signed [31:0]            param_M0,
    input  wire signed [31:0]            param_M1,
    input  wire signed [31:0]            param_M2,
    input  wire signed [31:0]            param_M3,
    input  wire        [7:0]             param_sh0,
    input  wire        [7:0]             param_sh1,
    input  wire        [7:0]             param_sh2,
    input  wire        [7:0]             param_sh3,
    input  wire                          param_load_start,
    output wire                          param_load_done
);

    // Reserved for future boundary cleanup when c_out_col_last / sa_done become
    // part of the explicit Conv->Quant transaction contract.
    wire unused_last  = ^c_out_col_last;
    wire unused_done  = sa_done;
    wire _unused_ok   = unused_last ^ unused_done;

    wire signed [ACC_W-1:0] col0 = c_out_col_stream_flat[0*ACC_W +: ACC_W];
    wire signed [ACC_W-1:0] col1 = c_out_col_stream_flat[1*ACC_W +: ACC_W];
    wire signed [ACC_W-1:0] col2 = c_out_col_stream_flat[2*ACC_W +: ACC_W];
    wire signed [ACC_W-1:0] col3 = c_out_col_stream_flat[3*ACC_W +: ACC_W];

    reg signed [ACC_W-1:0] qp_rso0_r;
    reg signed [ACC_W-1:0] qp_rso1_r;
    reg signed [ACC_W-1:0] qp_rso2_r;
    reg signed [ACC_W-1:0] qp_rso3_r;
    reg [COLS-1:0] col_valid_d1;
    reg [COLS-1:0] col_valid_d2;

    reg        cfg_compat_active;
    reg [3:0]  cfg_compat_idx;
    wire       cfg_valid_i;
    wire       cfg_ready_i;
    wire [31:0] cfg_data_i;
    wire       cfg_last_i;

    reg [3:0] en1_sr, en2_sr, en3_sr, en4_sr;
    reg       en1_raw, en2_raw, en3_raw, en4_raw;

    assign qp_rso0   = qp_rso0_r;
    assign qp_rso1   = qp_rso1_r;
    assign qp_rso2   = qp_rso2_r;
    assign qp_rso3   = qp_rso3_r;
```

```
  assign in_valid1 = col_valid_d1[0];
  assign in_valid2 = col_valid_d1[1];
  assign in_valid3 = col_valid_d1[2];
  assign in_valid4 = col_valid_d1[3];
  assign en1       = en1_raw;
  assign en2       = en2_raw;
  assign en3       = en3_raw;
  assign en4       = en4_raw;

  assign cfg_valid_i = cfg_compat_active;
  assign cfg_last_i  = (cfg_compat_idx == 4'd8);
  assign cfg_data_i  =
      (cfg_compat_idx == 4'd0) ? param_bias0 :
      (cfg_compat_idx == 4'd1) ? param_bias1 :
      (cfg_compat_idx == 4'd2) ? param_bias2 :
      (cfg_compat_idx == 4'd3) ? param_bias3 :
      (cfg_compat_idx == 4'd4) ? param_M0    :
      (cfg_compat_idx == 4'd5) ? param_M1    :
      (cfg_compat_idx == 4'd6) ? param_M2    :
      (cfg_compat_idx == 4'd7) ? param_M3    :
                                 {param_sh0, param_sh1, param_sh2, param_sh3};

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      qp_rso0_r    <= {ACC_W{1'b0}};
      qp_rso1_r    <= {ACC_W{1'b0}};
      qp_rso2_r    <= {ACC_W{1'b0}};
      qp_rso3_r    <= {ACC_W{1'b0}};
      col_valid_d1 <= {COLS{1'b0}};
      col_valid_d2 <= {COLS{1'b0}};
      start1       <= 1'b0;
      start2       <= 1'b0;
      start3       <= 1'b0;
      start4       <= 1'b0;
      en1_sr       <= 4'b0;
      en2_sr       <= 4'b0;
      en3_sr       <= 4'b0;
      en4_sr       <= 4'b0;
      en1_raw      <= 1'b0;
      en2_raw      <= 1'b0;
      en3_raw      <= 1'b0;
      en4_raw      <= 1'b0;
      cfg_compat_active <= 1'b0;
      cfg_compat_idx    <= 4'd0;
    end else begin
      qp_rso0_r <= col0;
      qp_rso1_r <= col1;
      qp_rso2_r <= col2;
      qp_rso3_r <= col3;

      col_valid_d1 <= c_out_col_valid;
      col_valid_d2 <= col_valid_d1;
      start1 <= col_valid_d1[0] | col_valid_d2[0];
      start2 <= col_valid_d1[1] | col_valid_d2[1];
```

```
      start3 <= col_valid_d1[2] | col_valid_d2[2];
      start4 <= col_valid_d1[3] | col_valid_d2[3];

      en1_sr  <= {en1_sr[2:0], col_valid_d1[0]};
      en2_sr  <= {en2_sr[2:0], col_valid_d1[1]};
      en3_sr  <= {en3_sr[2:0], col_valid_d1[2]};
      en4_sr  <= {en4_sr[2:0], col_valid_d1[3]};
      en1_raw <= en1_sr[3];
      en2_raw <= en2_sr[3];
      en3_raw <= en3_sr[3];
      en4_raw <= en4_sr[3];

      if (!cfg_compat_active) begin
        if (param_load_start) begin
          cfg_compat_active <= 1'b1;
          cfg_compat_idx    <= 4'd0;
        end
      end else if (cfg_valid_i && cfg_ready_i) begin
        if (cfg_compat_idx == 4'd8) begin
          cfg_compat_active <= 1'b0;
          cfg_compat_idx    <= 4'd0;
        end else begin
          cfg_compat_idx <= cfg_compat_idx + 4'd1;
        end
      end
    end
  end

  quant_param_loader u_quant_param_loader (
    .clk(clk),
    .rst_n(rst_n),
    .cfg_valid(cfg_valid_i),
    .cfg_ready(cfg_ready_i),
    .cfg_data(cfg_data_i),
    .cfg_last(cfg_last_i),
    .param_load_done(param_load_done),
    .bias_in(bias_in),
    .M_in(M_in),
    .sh_in(sh_in),
    .load_bias1(load_bias1),
    .load_bias2(load_bias2),
    .load_bias3(load_bias3),
    .load_bias4(load_bias4),
    .load_M1(load_M1),
    .load_M2(load_M2),
    .load_M3(load_M3),
    .load_M4(load_M4),
    .load_sh(load_sh)
  );

endmodule
```

A.9.3 Quantization Parameter Loader

Loads per-layer multiplier, shift, bias, and zero-point parameters into the quantization path.

code/input/RTL/quant_pool/integration/quant_param_loader.v

```
module quant_param_loader(
  input  wire               clk,
  input  wire               rst_n,

  // Handshake config stream: fixed 9-word packet
  //   0..3: bias0..3
  //   4..7: M0..3
  //   8   : {sh0, sh1, sh2, sh3}
  input  wire               cfg_valid,
  output wire               cfg_ready,
  input  wire [31:0]        cfg_data,
  input  wire               cfg_last,

  output wire               param_load_done,
  output reg signed [31:0]  bias_in,
  output reg signed [31:0]  M_in,
  output reg        [31:0]  sh_in,
  output reg                load_bias1,
  output reg                load_bias2,
  output reg                load_bias3,
  output reg                load_bias4,
  output reg                load_M1,
  output reg                load_M2,
  output reg                load_M3,
  output reg                load_M4,
  output reg                load_sh
);

  localparam integer CFG_WORDS = 9;

  localparam [3:0] PL_IDLE  = 4'd0,
                   PL_BIAS0 = 4'd1,
                   PL_BIAS1 = 4'd2,
                   PL_BIAS2 = 4'd3,
                   PL_BIAS3 = 4'd4,
                   PL_M0    = 4'd5,
                   PL_M1    = 4'd6,
                   PL_M2    = 4'd7,
                   PL_M3    = 4'd8,
                   PL_SH    = 4'd9;

  reg signed [31:0] bias_reg0, bias_reg1, bias_reg2, bias_reg3;
  reg signed [31:0] M_reg0,    M_reg1,    M_reg2,    M_reg3;
  reg        [31:0] sh_reg;

  reg [3:0] rx_idx;
  reg [3:0] pl_state;
  reg       preload_active;
```

```
  reg       load_done_r;

  wire cfg_fire = cfg_valid && cfg_ready;

  assign cfg_ready = !preload_active && (rx_idx < CFG_WORDS);
  assign param_load_done = load_done_r;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      bias_reg0 <= 32'sd0;
      bias_reg1 <= 32'sd0;
      bias_reg2 <= 32'sd0;
      bias_reg3 <= 32'sd0;
      M_reg0    <= 32'sd0;
      M_reg1    <= 32'sd0;
      M_reg2    <= 32'sd0;
      M_reg3    <= 32'sd0;
      sh_reg    <= 32'd0;
      bias_in   <= 32'sd0;
      M_in      <= 32'sd0;
      sh_in     <= 32'd0;
      load_bias1 <= 1'b0;
      load_bias2 <= 1'b0;
      load_bias3 <= 1'b0;
      load_bias4 <= 1'b0;
      load_M1    <= 1'b0;
      load_M2    <= 1'b0;
      load_M3    <= 1'b0;
      load_M4    <= 1'b0;
      load_sh    <= 1'b0;
      rx_idx        <= 4'd0;
      pl_state      <= PL_IDLE;
      preload_active <= 1'b0;
      load_done_r   <= 1'b0;
    end else begin
      load_bias1 <= 1'b0;
      load_bias2 <= 1'b0;
      load_bias3 <= 1'b0;
      load_bias4 <= 1'b0;
      load_M1    <= 1'b0;
      load_M2    <= 1'b0;
      load_M3    <= 1'b0;
      load_M4    <= 1'b0;
      load_sh    <= 1'b0;

      if (cfg_fire) begin
        load_done_r <= 1'b0;
        case (rx_idx)
          4'd0: bias_reg0 <= $signed(cfg_data);
          4'd1: bias_reg1 <= $signed(cfg_data);
          4'd2: bias_reg2 <= $signed(cfg_data);
          4'd3: bias_reg3 <= $signed(cfg_data);
          4'd4: M_reg0    <= $signed(cfg_data);
          4'd5: M_reg1    <= $signed(cfg_data);
```

```
          4'd6: M_reg2    <= $signed(cfg_data);
          4'd7: M_reg3    <= $signed(cfg_data);
          4'd8: sh_reg    <= cfg_data;
          default: ;
        endcase

        if (rx_idx == CFG_WORDS - 1) begin
          rx_idx         <= 4'd0;
          preload_active <= 1'b1;
          pl_state       <= PL_BIAS0;
        end else begin
          rx_idx <= rx_idx + 4'd1;
        end
      end

      if (preload_active) begin
        case (pl_state)
          PL_BIAS0: begin
            bias_in    <= bias_reg0;
            load_bias1 <= 1'b1;
            pl_state   <= PL_BIAS1;
          end
          PL_BIAS1: begin
            bias_in    <= bias_reg1;
            load_bias2 <= 1'b1;
            pl_state   <= PL_BIAS2;
          end
          PL_BIAS2: begin
            bias_in    <= bias_reg2;
            load_bias3 <= 1'b1;
            pl_state   <= PL_BIAS3;
          end
          PL_BIAS3: begin
            bias_in    <= bias_reg3;
            load_bias4 <= 1'b1;
            pl_state   <= PL_M0;
          end
          PL_M0: begin
            M_in      <= M_reg0;
            load_M1   <= 1'b1;
            pl_state  <= PL_M1;
          end
          PL_M1: begin
            M_in      <= M_reg1;
            load_M2   <= 1'b1;
            pl_state  <= PL_M2;
          end
          PL_M2: begin
            M_in      <= M_reg2;
            load_M3   <= 1'b1;
            pl_state  <= PL_M3;
          end
          PL_M3: begin
            M_in      <= M_reg3;
```

```
            load_M4   <= 1'b1;
            pl_state  <= PL_SH;
          end
          PL_SH: begin
            sh_in          <= sh_reg;
            load_sh        <= 1'b1;
            pl_state       <= PL_IDLE;
            preload_active <= 1'b0;
            load_done_r    <= 1'b1;
          end
          default: begin
            pl_state       <= PL_IDLE;
            preload_active <= 1'b0;
          end
        endcase
      end

      // Keep cfg_last in the interface contract even though the current
      // fixed 9-word packet format is ordered by index.
      if (cfg_fire && cfg_last && rx_idx != CFG_WORDS - 1) begin
        rx_idx      <= 4'd0;
        load_done_r <= 1'b0;
      end
    end
  end

endmodule
```

A.9.4 Quantization-Pooling Adapter

Bridges quantized outputs into the pooling stream and output writeback path.

code/input/RTL/quant_pool/integration/quant_pool_adapter.v

```
module quant_pool_adapter #(
  parameter integer OUT_W      = 62,
  parameter integer POOL_DEPTH = 31
)(
  input  wire              clk,
  input  wire              rst_n,
  input  wire signed [7:0] cut1_in,
  input  wire signed [7:0] cut2_in,
  input  wire signed [7:0] cut3_in,
  input  wire signed [7:0] cut4_in,
  input  wire              cut_valid1,
  input  wire              cut_valid2,
  input  wire              cut_valid3,
  input  wire              cut_valid4,
  output wire signed [7:0] cut1_out,
  output wire signed [7:0] cut2_out,
  output wire signed [7:0] cut3_out,
```

```
    output wire signed [7:0] cut4_out,
    output wire              cut_valid1_out,
    output wire              cut_valid2_out,
    output wire              cut_valid3_out,
    output wire              cut_valid4_out
);

    // Reserved for future continuity smoothing or skid buffering if a later
    // streaming pool revision adds input backpressure. The active stream-top
    // path currently accepts continuous lane traffic directly.
    wire unused_clk = clk;
    wire unused_rst = rst_n;
    wire _unused_ok = unused_clk ^ unused_rst;

    assign cut1_out       = cut1_in;
    assign cut2_out       = cut2_in;
    assign cut3_out       = cut3_in;
    assign cut4_out       = cut4_in;
    assign cut_valid1_out = cut_valid1;
    assign cut_valid2_out = cut_valid2;
    assign cut_valid3_out = cut_valid3;
    assign cut_valid4_out = cut_valid4;

endmodule
```

A.9.5 Quantization Processing Element

Implements one per-channel fixed-point requantization lane.

code/input/RTL/quant_pool/quant/Quantization_PE.v

```
module Quantization_PE(
    input clk, rst_n,
    input start,
    input in_valid,
    input load_bias, load_M, load_sh,
    input signed [31:0] bias_in,
    input signed [31:0] M_in,
    input [7:0] sh_in,
    input signed [22:0] rso,
    output reg signed [7:0] cut,
    output reg cut_valid
);

reg load_delay; // created by load_sh by one cycle delay to activate round_term calculation
reg [3:0] valid_sr;
reg signed [31:0] bias;
reg signed [31:0] next_bias;
reg signed [31:0] M;
reg signed [31:0] next_M;
reg [7:0] sh;
```

```
reg [7:0] next_sh;
reg signed [63:0] round_term;
reg signed [63:0] next_round_term;
reg signed [31:0] acc;
reg signed [31:0] next_acc;
reg signed [63:0] mul;
reg signed [63:0] next_mul;
reg signed [63:0] correct;
reg signed [63:0] next_correct;
reg signed [63:0] shiftt;
reg signed [63:0] next_shiftt;
reg signed [63:0] y;
reg signed [7:0] next_cut;
wire signed [31:0] MM;

// create load_delay by one cycle delay from load_sh
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) load_delay <= 1'd0;
    else load_delay <= load_sh;
end

// Output-valid tracks true input samples, not the tail-extended start signal.
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        valid_sr  <= 4'b0;
        cut_valid <= 1'b0;
    end
    else if (load_sh) begin
        valid_sr  <= 4'b0;
        cut_valid <= 1'b0;
    end
    else begin
        valid_sr  <= {valid_sr[2:0], in_valid};
        cut_valid <= valid_sr[3];
    end
end

// Bias(32-bits) loaded from outside to bias branch
always @* begin
    next_bias = bias;
    if (load_bias) begin
        next_bias = bias_in;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        bias <= 32'sd0;
    end
    else begin
        bias <= next_bias;
    end
end

```

```
// M(32-bits) loaded from outside to M branch
always @* begin
    next_M = M;
    if (load_M) begin
        next_M = M_in;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        M <= 32'sd0;
    end
    else begin
        M <= next_M;
    end
end

assign MM = start ? M : 32'sd0;

// sh loaded from outside to sh(shifter) branch
always @* begin
    next_sh = sh;
    if (load_sh) begin
        next_sh = sh_in;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        sh <= 8'd0;
    end
    else begin
        sh <= next_sh;
    end
end

// one cycle delay to generate round_term
always @* begin
    if (!load_delay) begin
        next_round_term = round_term;
    end
    else begin
        next_round_term = 64'sd1 << (sh - 8'd1);
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        round_term <= 64'sd0;
    end
    else if (load_sh) begin
        round_term <= 64'sd0;
    end
    else begin
```

```
        round_term <= next_round_term;
    end
end

// main branch
// step 1. adding bias
always @* begin
    next_acc = rso + bias;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        acc <= 32'sd0;
    end
    else if (load_sh) begin
        acc <= 32'sd0;
    end
    else begin
        acc <= next_acc;
    end
end

// step 2. acc * M
always @* begin
    next_mul = acc * MM;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        mul <= 64'sd0;
    end
    else if (load_sh) begin
        mul <= 64'sd0;
    end
    else begin
        mul <= next_mul;
    end
end

// step 3. Correctness: adding round_term & negative correctness
always @* begin
    next_correct = mul + round_term - mul[63];
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        correct <= 64'sd0;
    end
    else if (load_sh) begin
        correct <= 64'sd0;
    end
    else begin
        correct <= next_correct;
    end
```

```
end

// step 4. shifting sh-bits
always @* begin
    next_shiftt = correct >>> sh;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        shiftt <= 64'sd0;
    end
    else if (load_sh) begin
        shiftt <= 64'sd0;
    end
    else begin
        shiftt <= next_shiftt;
    end
end

// step 5.  sdding zeropoint & cutting LSB 8 bits
always @* begin
    y = shiftt - 64'sd128;  // 128 is the zp

    // saturating clamp
    if (y > 64'sd127) next_cut = 8'sd127;
    else if (y < -64'sd128) next_cut = -8'sd128;
    else next_cut = y[7:0];
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cut <= 8'sd0;
    end
    else if (load_sh) begin
        cut <= 8'sd0;
    end
    else begin
        cut <= next_cut;
    end
end

endmodule
```

A.9.6 Quantization Top

Applies per-channel bias, multiplier, shift, zero-point, and saturation to convolution accumulators.

code/input/RTL/quant_pool/quant/Quantization_Top.v

```
module Quantization_Top(
    input clk, rst_n,

    // triggered by its owm layer's finish signal by 2 cycles delay.
    input start1, start2, start3, start4,
    input in_valid1, in_valid2, in_valid3, in_valid4,

    // bias 32 bits, only one bias_in is loaded at a cycle; same M.
    input load_bias1, load_bias2, load_bias3, load_bias4,
    input load_M1, load_M2, load_M3, load_M4,

    // sh1, sh2, sh3, sh4 are 8 bits, can loaded all of them at a same cycle as 32 bits.
    input load_sh,

    input signed [31:0] bias_in,
    input signed [31:0] M_in,

    // sh_in 32 bits [31:24] is sh_in1, [23:16] is sh_in2, ...
    input [31:0] sh_in,

    input signed [22:0] rso0,
    input signed [22:0] rso1,
    input signed [22:0] rso2,
    input signed [22:0] rso3,
    output signed [7:0] cut1, cut2, cut3, cut4,
    output cut_valid1, cut_valid2, cut_valid3, cut_valid4
);

Quantization_PE u_QPE1(
    .clk(clk), .rst_n(rst_n), .start(start1), .in_valid(in_valid1),
    .load_bias(load_bias1), .load_M(load_M1), .load_sh(load_sh),
    .bias_in(bias_in), .M_in(M_in), .sh_in(sh_in[31:24]),
    .rso(rso0), .cut(cut1), .cut_valid(cut_valid1)
);

Quantization_PE u_QPE2(
    .clk(clk), .rst_n(rst_n), .start(start2), .in_valid(in_valid2),
    .load_bias(load_bias2), .load_M(load_M2), .load_sh(load_sh),
    .bias_in(bias_in), .M_in(M_in), .sh_in(sh_in[23:16]),
    .rso(rso1), .cut(cut2), .cut_valid(cut_valid2)
);

Quantization_PE u_QPE3(
    .clk(clk), .rst_n(rst_n), .start(start3), .in_valid(in_valid3),
    .load_bias(load_bias3), .load_M(load_M3), .load_sh(load_sh),
    .bias_in(bias_in), .M_in(M_in), .sh_in(sh_in[15:8]),
    .rso(rso2), .cut(cut3), .cut_valid(cut_valid3)
);

Quantization_PE u_QPE4(
    .clk(clk), .rst_n(rst_n), .start(start4), .in_valid(in_valid4),
    .load_bias(load_bias4), .load_M(load_M4), .load_sh(load_sh),
```

```
    .bias_in(bias_in), .M_in(M_in), .sh_in(sh_in[7:0]),
    .rso(rso3), .cut(cut4), .cut_valid(cut_valid4)
);

endmodule
```

A.9.7 Pooling Core

Implements the streaming 2-by-2 max-pooling datapath.

code/input/RTL/quant_pool/pool/pool_core.v

```
module pool_core_lane #(
  parameter integer POOL_DEPTH = 31   // max across all layers
)(
  input  wire        [1:0] layer_sel,
  input  wire              clk,
  input  wire              rst_n,
  input  wire              lane_reset,
  input  wire signed [7:0] cut,
  input  wire              cut_valid,
  input  wire              emit_step,
  input  wire              emit_done,
  output wire              bank_ready,
  output wire signed [7:0] emit_head
);

  // --- runtime decode from layer_sel ---
  reg [4:0] eff_pool_depth;
  reg [5:0] eff_in_w;
  reg [5:0] eff_in_h;
  always @* begin
    case (layer_sel)
      2'b01:   begin eff_pool_depth = 5'd14; eff_in_w = 6'd29; eff_in_h = 6'd29; end
      2'b10:   begin eff_pool_depth = 5'd6;  eff_in_w = 6'd12; eff_in_h = 6'd12; end
      default: begin eff_pool_depth = 5'd31; eff_in_w = 6'd62; eff_in_h = 6'd62; end
    endcase
  end

  reg              cnt1;
  reg        [4:0] cnt2;
  reg              cnt3;
  reg              ready_r;
  reg signed [7:0] accum_bank [0:POOL_DEPTH-1];
  reg signed [7:0] emit_bank  [0:POOL_DEPTH-1];
  reg signed [7:0] next_accum_val;
  integer i;

  // --- orphan pixel suppression counters ---
  reg [5:0] col_in_row;
  reg [5:0] row_cnt;
```

```
  wire pixel_ok = (col_in_row < {1'b0, eff_pool_depth} << 1)
              && (row_cnt    < {1'b0, eff_pool_depth} << 1);

  wire tile_complete = cut_valid && pixel_ok
                   && cnt1 && (cnt2 == eff_pool_depth - 5'd1) && cnt3;

  wire [4:0] cnt2_next =
      (cnt1 && (cnt2 != eff_pool_depth - 5'd1)) ? (cnt2 + 5'd1) :
      (cnt1 && (cnt2 == eff_pool_depth - 5'd1)) ? 5'd0 :
                                                  cnt2;
  wire cnt3_next = (cnt1 && (cnt2 == eff_pool_depth - 5'd1)) ? ~cnt3 : cnt3;

  always @* begin
    if (cnt3)
      next_accum_val = (cut > accum_bank[cnt2]) ? cut : accum_bank[cnt2];
    else if (cnt1)
      next_accum_val = (cut > accum_bank[cnt2]) ? cut : accum_bank[cnt2];
    else
      next_accum_val = cut;
  end

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      cnt1       <= 1'b0;
      cnt2       <= 5'd0;
      cnt3       <= 1'b0;
      ready_r    <= 1'b0;
      col_in_row <= 6'd0;
      row_cnt    <= 6'd0;
      for (i = 0; i < POOL_DEPTH; i = i + 1) begin
        accum_bank[i] <= 8'sd0;
        emit_bank[i]  <= 8'sd0;
      end
    end else if (lane_reset) begin
      cnt1       <= 1'b0;
      cnt2       <= 5'd0;
      cnt3       <= 1'b0;
      ready_r    <= 1'b0;
      col_in_row <= 6'd0;
      row_cnt    <= 6'd0;
      for (i = 0; i < POOL_DEPTH; i = i + 1) begin
        accum_bank[i] <= 8'sd0;
        emit_bank[i]  <= 8'sd0;
      end
    end else begin
      if (emit_step) begin
        for (i = 0; i < POOL_DEPTH-1; i = i + 1)
          emit_bank[i] <= emit_bank[i+1];
        emit_bank[POOL_DEPTH-1] <= 8'sd0;
      end

      if (emit_done)
        ready_r <= 1'b0;

```

```
      if (cut_valid) begin
        // advance orphan counters; auto-wrap at frame boundary
        if (col_in_row == eff_in_w - 6'd1) begin
          col_in_row <= 6'd0;
          if (row_cnt == eff_in_h - 6'd1)
            row_cnt <= 6'd0;          // frame complete -> ready for next
          else
            row_cnt <= row_cnt + 6'd1;
        end else begin
          col_in_row <= col_in_row + 6'd1;
        end

        if (pixel_ok) begin
          if (tile_complete) begin
            for (i = 0; i < POOL_DEPTH; i = i + 1) begin
              if (i == cnt2)
                emit_bank[i] <= next_accum_val;
              else
                emit_bank[i] <= accum_bank[i];
              accum_bank[i] <= 8'sd0;
            end
            ready_r <= 1'b1;
            cnt1    <= 1'b0;
            cnt2    <= 5'd0;
            cnt3    <= 1'b0;
          end else begin
            accum_bank[cnt2] <= next_accum_val;
            cnt1             <= ~cnt1;
            cnt2             <= cnt2_next;
            cnt3             <= cnt3_next;
          end
        end
        // !pixel_ok -> orphan pixel, only counters advanced, accum/cnt untouched
      end
    end
  end

  assign bank_ready = ready_r;
  assign emit_head  = emit_bank[0];

endmodule

module pool_core #(
  parameter integer POOL_DEPTH = 31
)(
  input  wire        [1:0]  layer_sel,
  input  wire               clk,
  input  wire               rst_n,
  input  wire signed [7:0]  cut1,
  input  wire signed [7:0]  cut2,
  input  wire signed [7:0]  cut3,
  input  wire signed [7:0]  cut4,
  input  wire               en1,
  input  wire               en2,
```

```
  input  wire               en3,
  input  wire               en4,
  input  wire               frame_rearm,
  input  wire               pool_ready,
  output wire signed [31:0] pool_data,
  output wire               pool_valid,
  output wire               pool_last,
  output wire               pool_frame_done
);

  // runtime decode for emit_last
  reg [4:0] eff_pool_depth;
  always @* begin
    case (layer_sel)
      2'b01:   eff_pool_depth = 5'd14;
      2'b10:   eff_pool_depth = 5'd6;
      default: eff_pool_depth = 5'd31;
    endcase
  end

  // Delay frame_rearm by 8 cycles so quant pipeline residual is flushed.
  // Reset on layer_changed so stale frame_rearm from previous layer doesn't
  // prematurely clear drop_until_idle in the new layer context.
  reg        emit_active;
  reg  [5:0] emit_count;
  reg        lane_reset_r;
  reg        drop_until_idle;
  reg [1:0]  layer_sel_d;
  wire       layer_changed;
  wire       l2_to_l3_change = (layer_sel_d == 2'b01) && (layer_sel == 2'b10);
  reg [7:0]  frame_rearm_sr;
  reg        frame_rearm_hold_active;
  reg        frame_rearm_pending;
  wire       frame_rearm_delayed = frame_rearm_sr[7];
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n)             frame_rearm_sr <= 8'd0;
    else                    frame_rearm_sr <= {frame_rearm_sr[6:0], frame_rearm};
  end

  assign layer_changed = (layer_sel != layer_sel_d);
  wire       lane1_ready, lane2_ready, lane3_ready, lane4_ready;
  wire signed [7:0] lane1_emit, lane2_emit, lane3_emit, lane4_emit;
  wire       all_ready = lane1_ready & lane2_ready & lane3_ready & lane4_ready;
  wire       emit_last = (emit_count == {1'b0, eff_pool_depth} - 6'd1);
  wire       emit_fire = emit_active && pool_ready;
  wire       emit_step = emit_fire;
  wire       emit_done = emit_fire && emit_last;
  wire       any_cut_valid = en1 | en2 | en3 | en4;
  wire       lane_cut_valid1 = drop_until_idle ? 1'b0 : en1;
  wire       lane_cut_valid2 = drop_until_idle ? 1'b0 : en2;
  wire       lane_cut_valid3 = drop_until_idle ? 1'b0 : en3;
  wire       lane_cut_valid4 = drop_until_idle ? 1'b0 : en4;

  pool_core_lane #(.POOL_DEPTH(POOL_DEPTH)) u_lane1 (
```

```
    .layer_sel(layer_sel),
    .clk(clk), .rst_n(rst_n),
    .lane_reset(lane_reset_r),
    .cut(cut1), .cut_valid(lane_cut_valid1),
    .emit_step(emit_step), .emit_done(emit_done),
    .bank_ready(lane1_ready), .emit_head(lane1_emit)
  );

  pool_core_lane #(.POOL_DEPTH(POOL_DEPTH)) u_lane2 (
    .layer_sel(layer_sel),
    .clk(clk), .rst_n(rst_n),
    .lane_reset(lane_reset_r),
    .cut(cut2), .cut_valid(lane_cut_valid2),
    .emit_step(emit_step), .emit_done(emit_done),
    .bank_ready(lane2_ready), .emit_head(lane2_emit)
  );

  pool_core_lane #(.POOL_DEPTH(POOL_DEPTH)) u_lane3 (
    .layer_sel(layer_sel),
    .clk(clk), .rst_n(rst_n),
    .lane_reset(lane_reset_r),
    .cut(cut3), .cut_valid(lane_cut_valid3),
    .emit_step(emit_step), .emit_done(emit_done),
    .bank_ready(lane3_ready), .emit_head(lane3_emit)
  );

  pool_core_lane #(.POOL_DEPTH(POOL_DEPTH)) u_lane4 (
    .layer_sel(layer_sel),
    .clk(clk), .rst_n(rst_n),
    .lane_reset(lane_reset_r),
    .cut(cut4), .cut_valid(lane_cut_valid4),
    .emit_step(emit_step), .emit_done(emit_done),
    .bank_ready(lane4_ready), .emit_head(lane4_emit)
  );

  assign pool_data  = {lane1_emit, lane2_emit, lane3_emit, lane4_emit};
  assign pool_valid = emit_active;
  assign pool_last  = emit_active && emit_last;

  // --- frame-level burst counter for pool_frame_done ---
  reg [4:0] eff_burst_total;
  always @* begin
    case (layer_sel)
      2'b01:   eff_burst_total = 5'd14;  // L2
      2'b10:   eff_burst_total = 5'd6;   // L3
      default: eff_burst_total = 5'd31;  // L1
    endcase
  end

  reg [4:0] burst_cnt;
  reg       frame_done_r;

  assign pool_frame_done = frame_done_r;

```

```
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      emit_active  <= 1'b0;
      emit_count   <= 6'd0;
      burst_cnt    <= 5'd0;
      frame_done_r <= 1'b0;
      lane_reset_r <= 1'b0;
      drop_until_idle <= 1'b0;
      layer_sel_d    <= 2'b00;
      frame_rearm_hold_active <= 1'b0;
      frame_rearm_pending <= 1'b0;
    end else begin
      frame_done_r <= 1'b0;
      lane_reset_r <= 1'b0;
      layer_sel_d  <= layer_sel;

      if (frame_rearm)
        frame_rearm_pending <= 1'b1;

      if (layer_changed) begin
        emit_active     <= 1'b0;
        emit_count      <= 6'd0;
        burst_cnt       <= 5'd0;
        lane_reset_r    <= 1'b1;
        drop_until_idle <= l2_to_l3_change;
        frame_rearm_hold_active <= l2_to_l3_change;
        frame_rearm_pending <= 1'b0;
      end else begin
        if (frame_rearm_hold_active && frame_rearm_delayed) begin
          drop_until_idle <= 1'b0;
          lane_reset_r    <= 1'b1;
          frame_rearm_hold_active <= 1'b0;
          frame_rearm_pending <= 1'b0;
        end

        if (!emit_active) begin
          emit_count <= 6'd0;
          if (all_ready)
            emit_active <= 1'b1;
        end else if (emit_fire) begin
          if (emit_last) begin
            emit_active <= 1'b0;
            emit_count  <= 6'd0;
            if (burst_cnt == eff_burst_total - 5'd1) begin
              burst_cnt       <= 5'd0;
              frame_done_r    <= 1'b1;
              lane_reset_r    <= 1'b1;
              if (frame_rearm_pending) begin
                frame_rearm_pending <= 1'b0;
              end else begin
                drop_until_idle         <= 1'b1;
                frame_rearm_hold_active <= 1'b1;
              end
            end else begin
```

```
              burst_cnt <= burst_cnt + 5'd1;
            end
          end else begin
            emit_count <= emit_count + 6'd1;
          end
        end
      end
    end
  end

endmodule
```

A.9.8 Pooling Stream Top

Wraps the pooling core with stream-level valid, ready, and last handling.

code/input/RTL/quant_pool/pool/pool_stream_top.v

```
module pool_stream_top #(
  parameter integer POOL_DEPTH = 31
)(
  input  wire        [1:0]  layer_sel,
  input  wire               clk,
  input  wire               rst_n,
  input  wire signed [7:0]  cut1,
  input  wire signed [7:0]  cut2,
  input  wire signed [7:0]  cut3,
  input  wire signed [7:0]  cut4,
  input  wire               cut_valid1,
  input  wire               cut_valid2,
  input  wire               cut_valid3,
  input  wire               cut_valid4,
  input  wire               frame_rearm,
  input  wire               pool_ready,
  output wire signed [31:0] pool_data,
  output wire               pool_valid,
  output wire               pool_last,
  output wire               pool_frame_done
);

  pool_core #(
    .POOL_DEPTH(POOL_DEPTH)
  ) u_pool_core (
    .layer_sel(layer_sel),
    .clk(clk),
    .rst_n(rst_n),
    .cut1(cut1),
    .cut2(cut2),
    .cut3(cut3),
    .cut4(cut4),
    .en1(cut_valid1),
```

```
    .en2(cut_valid2),
    .en3(cut_valid3),
    .en4(cut_valid4),
    .frame_rearm(frame_rearm),
    .pool_ready(pool_ready),
    .pool_data(pool_data),
    .pool_valid(pool_valid),
    .pool_last(pool_last),
    .pool_frame_done(pool_frame_done)
  );

endmodule
```

A.9.9 Fully-Connected Top

Implements the final classifier datapath and output logits.

code/input/RTL/fc/FC.v

```
module FC #(
  parameter PIX_W        = 8,
  parameter KER_W        = 8,
  parameter K            = 288,
  parameter ACC_W        = 32,
  parameter OUT_CHANNELS = 10
)(
  input  wire clk,
  input  wire rst_n,

  // cfg handshake for eff_bias loading (OUT_CHANNELS-word packet)
  input  wire        cfg_valid,
  output wire        cfg_ready,
  input  wire [31:0] cfg_data,
  input  wire        cfg_last,
  output wire        param_load_done,

  // MAC data path (packed vectors, LSB = channel 0)
  input  wire                              mul_en,
  input  wire [OUT_CHANNELS*PIX_W-1:0]     pixel_vec,
  input  wire [OUT_CHANNELS*KER_W-1:0]     kernel_vec,
  output wire [OUT_CHANNELS*ACC_W-1:0]     acc_vec,
  output wire [OUT_CHANNELS-1:0]           mul_done_vec,
  output wire                              all_done
);

  // Bias loader -> MAC wires
  wire signed [31:0]              bias_in_w;
  wire [OUT_CHANNELS-1:0]         load_bias_vec;

  fc_bias_loader #(
    .OUT_CHANNELS(OUT_CHANNELS)
```

```
  ) u_bias_loader (
    .clk(clk),
    .rst_n(rst_n),
    .cfg_valid(cfg_valid),
    .cfg_ready(cfg_ready),
    .cfg_data(cfg_data),
    .cfg_last(cfg_last),
    .param_load_done(param_load_done),
    .bias_in(bias_in_w),
    .load_bias_vec(load_bias_vec)
  );

  // Per-channel MAC array
  genvar gi;
  generate
    for (gi = 0; gi < OUT_CHANNELS; gi = gi + 1) begin : g_mac
      wire signed [PIX_W-1:0]  pix_i = pixel_vec [gi*PIX_W +: PIX_W];
      wire signed [KER_W-1:0]  ker_i = kernel_vec[gi*KER_W +: KER_W];
      wire signed [ACC_W-1:0]  acc_i;
      wire                     done_i;

      mac #(
        .PIX_W(PIX_W),
        .KER_W(KER_W),
        .K(K),
        .ACC_W(ACC_W)
      ) u_mac (
        .clk(clk),
        .rst_n(rst_n),
        .mul_en(mul_en),
        .pixel(pix_i),
        .kernel(ker_i),
        .load_bias(load_bias_vec[gi]),
        .bias_in(bias_in_w[ACC_W-1:0]),
        .acc(acc_i),
        .mul_done(done_i)
      );

      assign acc_vec[gi*ACC_W +: ACC_W] = acc_i;
      assign mul_done_vec[gi]           = done_i;
    end
  endgenerate

  assign all_done = &mul_done_vec;

endmodule
```

A.9.10 FC Bias Loader

Loads and presents FC bias terms for classifier accumulation.

code/input/RTL/fc/fc_bias_loader.v

```
module fc_bias_loader #(
  parameter OUT_CHANNELS = 10
)(
  input  wire        clk,
  input  wire        rst_n,

  // 4-wire cfg handshake: OUT_CHANNELS-word packet, one bias per word.
  input  wire        cfg_valid,
  output wire        cfg_ready,
  input  wire [31:0] cfg_data,
  input  wire        cfg_last,

  output wire                         param_load_done,
  output reg  signed [31:0]           bias_in,
  output reg  [OUT_CHANNELS-1:0]      load_bias_vec
);

  localparam integer CFG_WORDS = OUT_CHANNELS;
  localparam integer IDX_W     = (OUT_CHANNELS <= 1) ? 1 : $clog2(OUT_CHANNELS);

  localparam [1:0] PL_IDLE       = 2'd0,
                   PL_DISTRIBUTE = 2'd1;

  reg signed [31:0]       bias_reg [0:OUT_CHANNELS-1];
  reg [IDX_W:0]           rx_idx;    // one extra bit to compare against CFG_WORDS
  reg [IDX_W:0]           pl_idx;
  reg [1:0]               pl_state;
  reg                     preload_active;
  reg                     load_done_r;

  wire cfg_fire = cfg_valid && cfg_ready;

  assign cfg_ready       = !preload_active && (rx_idx < CFG_WORDS);
  assign param_load_done = load_done_r;

  always @(posedge clk or negedge rst_n) begin : bias_fsm
    integer i;
    if (!rst_n) begin
      for (i = 0; i < OUT_CHANNELS; i = i + 1)
        bias_reg[i] <= 32'sd0;
      bias_in        <= 32'sd0;
      load_bias_vec  <= {OUT_CHANNELS{1'b0}};
      rx_idx         <= {(IDX_W+1){1'b0}};
      pl_idx         <= {(IDX_W+1){1'b0}};
      pl_state       <= PL_IDLE;
      preload_active <= 1'b0;
      load_done_r    <= 1'b0;
    end else begin
      // Default: clear one-hot pulse
      load_bias_vec <= {OUT_CHANNELS{1'b0}};

      // ---- RX phase: accept OUT_CHANNELS cfg words ----
```

```
      if (cfg_fire) begin
        load_done_r <= 1'b0;
        bias_reg[rx_idx[IDX_W-1:0]] <= $signed(cfg_data);

        if (rx_idx == CFG_WORDS - 1) begin
          rx_idx         <= {(IDX_W+1){1'b0}};
          preload_active <= 1'b1;
          pl_state       <= PL_DISTRIBUTE;
          pl_idx         <= {(IDX_W+1){1'b0}};
        end else begin
          rx_idx <= rx_idx + 1'b1;
        end
      end

      // ---- Preload phase: one channel per cycle ----
      if (preload_active && pl_state == PL_DISTRIBUTE) begin
        bias_in                      <= bias_reg[pl_idx[IDX_W-1:0]];
        load_bias_vec[pl_idx[IDX_W-1:0]] <= 1'b1;

        if (pl_idx == OUT_CHANNELS - 1) begin
          pl_state       <= PL_IDLE;
          preload_active <= 1'b0;
          load_done_r    <= 1'b1;
          pl_idx         <= {(IDX_W+1){1'b0}};
        end else begin
          pl_idx <= pl_idx + 1'b1;
        end
      end

      // Early cfg_last resets RX if packet is short
      if (cfg_fire && cfg_last && rx_idx != CFG_WORDS - 1) begin
        rx_idx      <= {(IDX_W+1){1'b0}};
        load_done_r <= 1'b0;
      end
    end
  end

endmodule
```

A.9.11 FC Data Adapter

Adapts SRAM_B feature data and FC weight data into the classifier MAC schedule.

code/input/RTL/fc/fc_data_adapter.v

```
// FC Data Adapter: synchronize SRAM_FCW weight stream and SRAM_B pixel stream
// into FC pixel_vec/kernel_vec + mul_en.
//
// SRAM_FCW (weight): 288 words x 80b, one word per MAC cycle.
//   Packing (LSB = channel 0): {k9, k8, k7, k6, k5, k4, k3, k2, k1, k0}
// SRAM_B (data):    72 words x 32b, one word per 4 MAC cycles (byte-serial).
```

```
//
// The same pixel byte is broadcast to all OUT_CHANNELS MACs in the same cycle;
// per-channel differentiation lives entirely in the 80-bit weight word.

module fc_data_adapter #(
  parameter OUT_CHANNELS = 10,
  parameter PIX_W        = 8,
  parameter KER_W        = 8
)(
  input  wire        clk,
  input  wire        rst_n,

  // FC weight stream (from SRAM_FCW, 288 words x 80b)
  input  wire                                wt_valid,
  input  wire [OUT_CHANNELS*KER_W-1:0]       wt_data,
  input  wire                                wt_last,
  output wire                                wt_ready,

  // FC data stream (from SRAM_B, 72 packed words)
  input  wire                                data_valid,
  input  wire [31:0]                         data_data,
  input  wire                                data_last,
  output wire                                data_ready,

  // FC data path (packed vectors, LSB = channel 0)
  output wire                                mul_en,
  output wire [OUT_CHANNELS*PIX_W-1:0]       pixel_vec,
  output wire [OUT_CHANNELS*KER_W-1:0]       kernel_vec,

  // FC completion
  input  wire        all_done,
  output wire        fc_done
);
  localparam integer WT_W = OUT_CHANNELS*KER_W;

  // --- Holding registers ---
  reg [WT_W-1:0]  wt_reg;
  reg             wt_held;
  reg [31:0]      data_reg;
  reg             data_held;
  reg [1:0]       byte_sel;

  wire wt_avail    = wt_held;
  wire pixel_avail = data_held;

  assign mul_en     = wt_avail && pixel_avail && !all_done;
  assign wt_ready   = (wt_held   && mul_en) || (!wt_held   && !wt_valid);
  assign data_ready = (data_held && mul_en && (byte_sel == 2'd3))
                   || (!data_held && !data_valid);

  // Kernel vector: directly drive from held 80-bit weight word
  assign kernel_vec = wt_reg;

  // Pixel byte selection (SRAM_B packs high byte first, matching current behavior)
```

```
  wire [7:0] pixel_byte = (byte_sel == 2'd0) ? data_reg[31:24] :
                          (byte_sel == 2'd1) ? data_reg[23:16] :
                          (byte_sel == 2'd2) ? data_reg[15: 8] :
                                               data_reg[ 7: 0];

  // Broadcast one pixel byte to all OUT_CHANNELS slots
  genvar gi;
  generate
    for (gi = 0; gi < OUT_CHANNELS; gi = gi + 1) begin : g_pix
      assign pixel_vec[gi*PIX_W +: PIX_W] = $signed(pixel_byte);
    end
  endgenerate

  assign fc_done = all_done;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      wt_reg    <= {WT_W{1'b0}};
      wt_held   <= 1'b0;
      data_reg  <= 32'd0;
      data_held <= 1'b0;
      byte_sel  <= 2'd0;
    end else begin
      if (!wt_held) begin
        if (wt_valid) begin
          wt_reg  <= wt_data;
          wt_held <= 1'b1;
        end
      end else if (mul_en) begin
        wt_held <= 1'b0;
      end

      if (!data_held) begin
        if (data_valid) begin
          data_reg  <= data_data;
          data_held <= 1'b1;
          byte_sel  <= 2'd0;
        end
      end else if (mul_en) begin
        if (byte_sel == 2'd3) begin
          byte_sel  <= 2'd0;
          data_held <= 1'b0;
        end else begin
          byte_sel <= byte_sel + 2'd1;
        end
      end
    end
  end

endmodule
```

A.9.12 FC MAC

Implements the multiply-accumulate primitive used by the fully-connected classifier.

code/input/RTL/fc/mac.v

```
module mac #(
  parameter PIX_W = 8,
  parameter KER_W = 8,
  parameter K = 288,
  parameter ACC_W = 32
)(
  input  wire clk,
  input  wire rst_n,
  input  wire mul_en,
  input  wire signed [PIX_W-1:0] pixel,
  input  wire signed [KER_W-1:0] kernel,
  // Bias preload
  input  wire               load_bias,
  input  wire signed [ACC_W-1:0] bias_in,
  output reg  signed [ACC_W-1:0] acc,
  output reg  mul_done
);

  wire signed [PIX_W+KER_W-1:0] product;
  wire signed [ACC_W-1:0] product_ext;

  reg [$clog2(K)-1:0] cnt;
  reg signed [ACC_W-1:0] bias_reg;

  assign product = $signed(pixel) * $signed(kernel);
  assign product_ext = {{(ACC_W-(PIX_W+KER_W)){product[PIX_W+KER_W-1]}}, product};

  // Bias register: latch on load_bias pulse
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
      bias_reg <= {ACC_W{1'b0}};
    else if (load_bias)
      bias_reg <= bias_in;
  end

  // MAC accumulator
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      acc      <= {ACC_W{1'b0}};
      mul_done <= 1'b0;
      cnt      <= {$clog2(K){1'b0}};
    end else if (load_bias) begin
      // Reset for new inference: clear accumulator state so mul_en can fire
      acc      <= {ACC_W{1'b0}};
      mul_done <= 1'b0;
      cnt      <= {$clog2(K){1'b0}};
    end else if (mul_en) begin
      if (cnt == 0) begin
```

```
        acc      <= bias_reg + product_ext;
        mul_done <= 1'b0;
        cnt      <= cnt + 1'b1;
      end else if (cnt == K - 1) begin
        acc      <= acc + product_ext;
        mul_done <= 1'b1;
        cnt      <= {$clog2(K){1'b0}};
      end else begin
        acc      <= acc + product_ext;
        mul_done <= 1'b0;
        cnt      <= cnt + 1'b1;
      end
    end
  end

endmodule
```

A.10 SRAM Controller RTL

A.10.1 SRAM Address Generator

Generates layer-dependent SRAM addresses for configuration, weights, activations, and writeback.

code/input/RTL/SRAM/Addr_Gen.v

```
module Addr_Gen #(
    parameter ADDR_WIDTH = 11
)(
    input  clk,
    input  rst_n,

    input  start,
    input  enable,
    input  [ADDR_WIDTH-1:0] base_addr,
    input  [ADDR_WIDTH-1:0] length,

    output [ADDR_WIDTH-1:0] addr,
    output done
);

    // ----------------------------------------
    // Internal Registers
    // ----------------------------------------
    reg [ADDR_WIDTH-1:0] counter;
    reg                  running;

    // ----------------------------------------
    // Running Control
    // ----------------------------------------
    always @(posedge clk or negedge rst_n) begin
```

```
        if (!rst_n)
            running <= 1'b0;
        else if (start)
            running <= 1'b1;
        else if (done)
            running <= 1'b0;
    end

    // ----------------------------------------
    // Counter
    // ----------------------------------------
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            counter <= {ADDR_WIDTH{1'b0}};
        else if (start)
            counter <= {ADDR_WIDTH{1'b0}};
        else if (running && enable && !done)
            counter <= counter + 1'b1;
    end

    // ----------------------------------------
    // Combinational Outputs
    // ----------------------------------------
    assign addr = base_addr + counter;

    assign done = running && enable &&
                  (counter == (length - 1'b1));

endmodule
```

A.10.2 FC Weight Preload Packer

Packs incoming FC weight preload words into the SRAM_FCW layout.

code/input/RTL/SRAM/fcw_preload_packer.v

```
`timescale 1ns/1ps

// fcw_preload_packer: pack three 32-bit host words into one 80-bit FCW write.
//
// Host preload stream carries FC weights at 32b/word. Each FCW slot holds
// 10 kernels x 8b = 80b. The packer consumes 3 host words per FCW slot:
//   word0 bits[31:0]  -> fcw_word[31:0]   = {k3, k2, k1, k0}
//   word1 bits[31:0]  -> fcw_word[63:32]  = {k7, k6, k5, k4}
//   word2 bits[15:0]  -> fcw_word[79:64]  = {k9, k8}    (upper 16b ignored)
//
// On the third beat the packer emits a single wea pulse at the current
// FCW address, then advances the address. 'start' resets the address
// counter and phase counter.

module fcw_preload_packer # (
```

```
    parameter AW = 9,
    parameter DW = 80
) (
    input               clk,
    input               rst_n,

    input               start,         // resets address/phase counters

    // 32b host preload stream
    input               up_valid,
    input      [31:0]   up_data,
    output              up_ready,

    // 80b write port to sram_FCW_wrapper
    output                  wea,
    output     [AW-1:0]     addra,
    output     [DW-1:0]     dina
);
    reg  [1:0]     phase;
    reg  [AW-1:0]  wr_cnt;
    reg  [31:0]    word0_r;
    reg  [31:0]    word1_r;

    wire fire = up_valid && up_ready;

    assign up_ready = 1'b1;

    // Combinationally emit wea on the cycle the third host word arrives.
    assign wea   = fire && (phase == 2'd2);
    assign addra = wr_cnt;
    assign dina  = {up_data[15:0], word1_r, word0_r};

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            phase   <= 2'd0;
            wr_cnt  <= {AW{1'b0}};
            word0_r <= 32'd0;
            word1_r <= 32'd0;
        end else if (start) begin
            phase  <= 2'd0;
            wr_cnt <= {AW{1'b0}};
        end else if (fire) begin
            case (phase)
                2'd0: begin
                    word0_r <= up_data;
                    phase   <= 2'd1;
                end
                2'd1: begin
                    word1_r <= up_data;
                    phase   <= 2'd2;
                end
                2'd2: begin
                    phase  <= 2'd0;
                    wr_cnt <= wr_cnt + {{(AW-1){1'b0}}, 1'b1};
```

```
                end
                default: phase <= 2'd0;
            endcase
        end
    end

endmodule
```

A.10.3 SRAM A Controller

Controls SRAM_A reads and writes for preload data, convolution input, and intermediate feature maps.

code/input/RTL/SRAM/sram_A_controller.v

```
module sram_A_controller #(
    parameter ADDR_WIDTH = 10
)(
    input [2:0] layer_sel,
    input [1:0] data_sel,
    input       pass_id,

    output reg [ADDR_WIDTH-1:0] base_addr,
    output reg [ADDR_WIDTH-1:0] length,
    output reg                  read_mode,
    output reg                  write_mode
);

localparam [2:0] LAYER_PRELOAD = 3'd0;
localparam [2:0] LAYER_L1      = 3'd1;
localparam [2:0] LAYER_L2      = 3'd2;
localparam [2:0] LAYER_L3      = 3'd3;
localparam [2:0] LAYER_FC      = 3'd4;

localparam [1:0] SEL_CFG       = 2'd0;
localparam [1:0] SEL_WT        = 2'd1;
localparam [1:0] SEL_DATA      = 2'd2;
localparam [1:0] SEL_FCW       = 2'd3;  // FC weights in dedicated 80b SRAM_FCW

localparam [ADDR_WIDTH-1:0] PRELOAD_CFG_BASE  = 10'h000;
localparam [ADDR_WIDTH-1:0] PRELOAD_CFG_LEN   = 10'd45; // conv L1/L2/L3 cfg only; FC bias moved to
    dedicated slot
localparam [ADDR_WIDTH-1:0] PRELOAD_WT_BASE   = 10'h030;
localparam [ADDR_WIDTH-1:0] PRELOAD_WT_LEN    = 10'd225; // conv L1/L2/L3 weights only; FC weights moved
    to SRAM_FCW

// FC bias preload reuses SRAM_A (32b words) but at its own base outside
// the conv region. Preload writes 10 bias words to the free zone that used
// to hold the 3-class FC weight table.
localparam [ADDR_WIDTH-1:0] PRELOAD_FC_CFG_BASE = 10'h111;
```

```
localparam [ADDR_WIDTH-1:0] PRELOAD_FC_CFG_LEN  = 10'd10;

// FC weight preload streams through SRAM_A write port (host 32b) into the
// SRAM_FCW packer: 288 FCW slots x 3 host words = 864 beats.
localparam [ADDR_WIDTH-1:0] PRELOAD_FCW_LEN   = 10'd864;
// DATA preload is no longer stored in SRAM_A.
localparam [ADDR_WIDTH-1:0] PRELOAD_DATA_BASE = 10'h231;
localparam [ADDR_WIDTH-1:0] PRELOAD_DATA_LEN  = 10'd0;

localparam [ADDR_WIDTH-1:0] L1_CFG_BASE       = 10'h000;
localparam [ADDR_WIDTH-1:0] L1_CFG_LEN        = 10'd9;
localparam [ADDR_WIDTH-1:0] L1_WT_BASE        = 10'h030;
localparam [ADDR_WIDTH-1:0] L1_WT_LEN         = 10'd9;
// L1 input data is no longer read from SRAM_A.
localparam [ADDR_WIDTH-1:0] L1_DATA_BASE      = 10'h231;
localparam [ADDR_WIDTH-1:0] L1_DATA_LEN       = 10'd0;

localparam [ADDR_WIDTH-1:0] L2_PASS0_CFG_BASE = 10'h009;
localparam [ADDR_WIDTH-1:0] L2_PASS0_CFG_LEN  = 10'd9;
localparam [ADDR_WIDTH-1:0] L2_PASS1_CFG_BASE = 10'h012;
localparam [ADDR_WIDTH-1:0] L2_PASS1_CFG_LEN  = 10'd9;
localparam [ADDR_WIDTH-1:0] L2_PASS0_WT_BASE  = 10'h039;
localparam [ADDR_WIDTH-1:0] L2_PASS0_WT_LEN   = 10'd36;
localparam [ADDR_WIDTH-1:0] L2_PASS1_WT_BASE  = 10'h05D;
localparam [ADDR_WIDTH-1:0] L2_PASS1_WT_LEN   = 10'd36;
localparam [ADDR_WIDTH-1:0] L2_PASS0_OUT_BASE = 10'h231;
localparam [ADDR_WIDTH-1:0] L2_PASS0_OUT_LEN  = 10'd196;
// L2 pass1 uses same base as pass0; stride-2 interleave done in top_sram_A
localparam [ADDR_WIDTH-1:0] L2_PASS1_OUT_BASE = 10'h231;
localparam [ADDR_WIDTH-1:0] L2_PASS1_OUT_LEN  = 10'd196;

localparam [ADDR_WIDTH-1:0] L3_PASS0_CFG_BASE = 10'h01B;
localparam [ADDR_WIDTH-1:0] L3_PASS0_CFG_LEN  = 10'd9;
localparam [ADDR_WIDTH-1:0] L3_PASS1_CFG_BASE = 10'h024;
localparam [ADDR_WIDTH-1:0] L3_PASS1_CFG_LEN  = 10'd9;
localparam [ADDR_WIDTH-1:0] L3_PASS0_WT_BASE  = 10'h081;
localparam [ADDR_WIDTH-1:0] L3_PASS0_WT_LEN   = 10'd72;
localparam [ADDR_WIDTH-1:0] L3_PASS1_WT_BASE  = 10'h0C9;
localparam [ADDR_WIDTH-1:0] L3_PASS1_WT_LEN   = 10'd72;
localparam [ADDR_WIDTH-1:0] L3_DATA_BASE      = 10'h231;
localparam [ADDR_WIDTH-1:0] L3_DATA_LEN       = 10'd392;

// FC bias lives at 0x111 (the region freed by moving FC weights to SRAM_FCW).
// FC weights no longer live in SRAM_A at all -- LAYER_FC+SEL_WT is a no-op
// here; top_sram_A routes that transaction to the SRAM_FCW peer instead.
localparam [ADDR_WIDTH-1:0] FC_CFG_BASE       = 10'h111;
localparam [ADDR_WIDTH-1:0] FC_CFG_LEN        = 10'd10;

always @(*) begin
    base_addr  = {ADDR_WIDTH{1'b0}};
    length     = {ADDR_WIDTH{1'b0}};
    read_mode  = 1'b0;
    write_mode = 1'b0;

```

```
    case (layer_sel)
        LAYER_PRELOAD: begin
            write_mode = 1'b1;
            case (data_sel)
                SEL_CFG: begin
                    base_addr = PRELOAD_CFG_BASE;
                    length    = PRELOAD_CFG_LEN;
                end
                SEL_WT: begin
                    base_addr = PRELOAD_WT_BASE;
                    length    = PRELOAD_WT_LEN;
                end
                SEL_DATA: begin
                    // Preload DATA no longer uses SRAM_A.
                    write_mode = 1'b0;
                end
                SEL_FCW: begin
                    // FC weight preload: top_sram_A redirects the write stream
                    // to the SRAM_FCW packer. length counts host 32b beats.
                    base_addr = {ADDR_WIDTH{1'b0}};
                    length    = PRELOAD_FCW_LEN;
                end
                default: begin
                    write_mode = 1'b0;
                end
            endcase
        end

        LAYER_L1: begin
            read_mode = 1'b1;
            case (data_sel)
                SEL_CFG: begin
                    base_addr = L1_CFG_BASE;
                    length    = L1_CFG_LEN;
                end
                SEL_WT: begin
                    base_addr = L1_WT_BASE;
                    length    = L1_WT_LEN;
                end
                SEL_DATA: begin
                    // L1 DATA no longer reads from SRAM_A.
                    read_mode = 1'b0;
                end
                default: begin
                    read_mode = 1'b0;
                end
            endcase
        end

        LAYER_L2: begin
            case (data_sel)
                SEL_CFG: begin
                    read_mode = 1'b1;
                    base_addr = pass_id ? L2_PASS1_CFG_BASE : L2_PASS0_CFG_BASE;
```

```
                    length    = pass_id ? L2_PASS1_CFG_LEN  : L2_PASS0_CFG_LEN;
                end
                SEL_WT: begin
                    read_mode = 1'b1;
                    base_addr = pass_id ? L2_PASS1_WT_BASE : L2_PASS0_WT_BASE;
                    length    = pass_id ? L2_PASS1_WT_LEN  : L2_PASS0_WT_LEN;
                end
                SEL_DATA: begin
                    write_mode = 1'b1;
                    base_addr  = pass_id ? L2_PASS1_OUT_BASE : L2_PASS0_OUT_BASE;
                    length     = pass_id ? L2_PASS1_OUT_LEN  : L2_PASS0_OUT_LEN;
                end
                default: begin
                end
            endcase
        end

        LAYER_L3: begin
            read_mode = 1'b1;
            case (data_sel)
                SEL_CFG: begin
                    base_addr = pass_id ? L3_PASS1_CFG_BASE : L3_PASS0_CFG_BASE;
                    length    = pass_id ? L3_PASS1_CFG_LEN  : L3_PASS0_CFG_LEN;
                end
                SEL_WT: begin
                    base_addr = pass_id ? L3_PASS1_WT_BASE : L3_PASS0_WT_BASE;
                    length    = pass_id ? L3_PASS1_WT_LEN  : L3_PASS0_WT_LEN;
                end
                SEL_DATA: begin
                    base_addr = L3_DATA_BASE;
                    length    = L3_DATA_LEN;
                end
                default: begin
                    read_mode = 1'b0;
                end
            endcase
        end

        LAYER_FC: begin
            case (data_sel)
                SEL_CFG: begin
                    // FC bias lives in the freed old-FC-WT region of SRAM_A.
                    // Enable BOTH read and write so the same transaction can
                    // preload (write via pool_valid from host load_data) or
                    // fetch at inference time (read via data_ready from FC).
                    // Only one of read_en/write_en actually fires per cycle
                    // because pool_valid and data_ready are mutually exclusive
                    // in their respective modes.
                    read_mode  = 1'b1;
                    write_mode = 1'b1;
                    base_addr  = FC_CFG_BASE;
                    length     = FC_CFG_LEN;
                end
                SEL_WT: begin
```

```
                    // FC weights live in SRAM_FCW, not here.
                    // top_sram_A routes this transaction to the FCW peer and
                    // leaves the 32b SRAM_A idle; length=0 so Addr_Gen never
                    // asserts done on this path.
                end
                default: begin
                end
            endcase
        end

        default: begin
        end
    endcase
end

endmodule
```

A.10.4 SRAM A Wrapper

Wraps the SRAM_A model with active-low chip/write enables and read-valid timing.

code/input/RTL/SRAM/sram_A_wrapper.v

```
`timescale 1ns/1ps
`define DELAY #1

module sram_A_wrapper # (
    parameter AW = 10
) (
    input clk,
    input rst_n,
    input write_en,
    input read_en,
    input [AW-1:0] addr,
    input [31:0] write_data,

    output [31:0] read_data,
    output reg read_valid
);
    wire do_write;
    wire do_read;

    wire CEN;
    wire WEN;
    wire [AW-1:0] addr_delayed;
    wire [31:0] write_data_delayed;

    assign do_write = write_en & ~read_en;
    assign do_read = read_en & ~write_en;

    assign `DELAY CEN = ~(do_write | do_read);
```

```
    assign `DELAY WEN = ~do_write;
    assign `DELAY addr_delayed = addr;
    assign `DELAY write_data_delayed = write_data;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            read_valid <= 1'b0;
        else
            read_valid <= do_read;
    end

    sram_A_model sram_A_inst (
        .CLK(clk),
        .CEN(CEN),
        .WEN(WEN),
        .A(addr_delayed),
        .D(write_data_delayed),
        .EMA(3'b000),
        .RETN(1'b1),
        .Q(read_data)
    );


endmodule
```

A.10.5 SRAM B Controller

Controls SRAM_B read/write traffic for pooled convolution outputs and downstream layer inputs.

code/input/RTL/SRAM/sram_B_controller.v

```
module sram_B_controller #(
    parameter ADDR_WIDTH = 10
)(
    input [2:0] layer_sel,
    input [1:0] data_sel,
    input       pass_id,

    output reg [ADDR_WIDTH-1:0] base_addr,
    output reg [ADDR_WIDTH-1:0] length,
    output reg                  read_mode,
    output reg                  write_mode
);

localparam [2:0] LAYER_PRELOAD = 3'd0;
localparam [2:0] LAYER_L1      = 3'd1;
localparam [2:0] LAYER_L2      = 3'd2;
localparam [2:0] LAYER_L3      = 3'd3;
localparam [2:0] LAYER_FC      = 3'd4;

localparam [1:0] SEL_CFG       = 2'd0;
```

```
localparam [1:0] SEL_WT        = 2'd1;
localparam [1:0] SEL_DATA      = 2'd2;

// SRAM_B only stores runtime feature maps in the current top-level memory plan.
// There is no preload transaction on SRAM_B.
// It is reused in time:
// - L1 pooled output shares the same 31x31x4 buffer used as L2 input.
// - L3/FC data traffic is routed by top_sram_B into a dedicated FC-order
//   buffer, so the FC stage no longer needs interleaved reads out of SRAM_B.

localparam [ADDR_WIDTH-1:0] L1_OUT_BASE       = 10'h000;
localparam [ADDR_WIDTH-1:0] L1_OUT_LEN        = 10'd961; // 31*31*4 bytes / 4

localparam [ADDR_WIDTH-1:0] L2_DATA_BASE      = 10'h000;
localparam [ADDR_WIDTH-1:0] L2_DATA_LEN       = 10'd961; // 31*31*4 bytes / 4

localparam [ADDR_WIDTH-1:0] L3_PASS0_OUT_BASE = 10'h000;
localparam [ADDR_WIDTH-1:0] L3_PASS0_OUT_LEN  = 10'd36;  // 6*6*4 bytes / 4
localparam [ADDR_WIDTH-1:0] L3_PASS1_OUT_BASE = 10'h024;
localparam [ADDR_WIDTH-1:0] L3_PASS1_OUT_LEN  = 10'd36;  // 6*6*4 bytes / 4

localparam [ADDR_WIDTH-1:0] FC_DATA_BASE      = 10'h000;
localparam [ADDR_WIDTH-1:0] FC_DATA_LEN       = 10'd72;  // 6*6*8 bytes / 4

always @(*) begin
    base_addr  = {ADDR_WIDTH{1'b0}};
    length     = {ADDR_WIDTH{1'b0}};
    read_mode  = 1'b0;
    write_mode = 1'b0;

    case (layer_sel)
        LAYER_L1: begin
            if (data_sel == SEL_DATA) begin
                write_mode = 1'b1;
                base_addr  = L1_OUT_BASE;
                length     = L1_OUT_LEN;
            end
        end

        LAYER_L2: begin
            if (data_sel == SEL_DATA) begin
                read_mode = 1'b1;
                base_addr = L2_DATA_BASE;
                length    = L2_DATA_LEN;
            end
        end

        LAYER_L3: begin
            if (data_sel == SEL_DATA) begin
                write_mode = 1'b1;
                base_addr  = pass_id ? L3_PASS1_OUT_BASE : L3_PASS0_OUT_BASE;
                length     = pass_id ? L3_PASS1_OUT_LEN  : L3_PASS0_OUT_LEN;
            end
        end
```

```
75
76          LAYER_FC: begin
77              if (data_sel == SEL_DATA) begin
78                  read_mode = 1'b1;
79                  base_addr = FC_DATA_BASE;
80                  length    = FC_DATA_LEN;
81              end
82          end
83
84          default: begin
85          end
86      endcase
87  end
88
89  endmodule
```

A.10.6 SRAM B Wrapper

Wraps the SRAM_B model with the project-level SRAM control convention.

code/input/RTL/SRAM/sram_B_wrapper.v

```
1  `timescale 1ns/1ps
2  `define DELAY #1
3
4  module sram_B_wrapper # (
5      parameter AW = 11
6  ) (
7      input clk,
8      input rst_n,
9      input write_en,
10     input read_en,
11     input [AW-1:0] addr,
12     input [31:0] write_data,
13
14     output [31:0] read_data,
15     output reg read_valid
16 );
17     wire do_write;
18     wire do_read;
19
20     wire CEN;
21     wire WEN;
22     wire [AW-1:0] addr_delayed;
23     wire [31:0] write_data_delayed;
24
25     assign do_write = write_en & ~read_en;
26     assign do_read = read_en & ~write_en;
27
28     assign `DELAY CEN = ~(do_write | do_read);
29     assign `DELAY WEN = ~do_write;
```

```
    assign `DELAY addr_delayed = addr;
    assign `DELAY write_data_delayed = write_data;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            read_valid <= 1'b0;
        else
            read_valid <= do_read;
    end

    sram_B_model sram_B_inst (
        .CLK(clk),
        .CEN(CEN),
        .WEN(WEN),
        .A(addr_delayed),
        .D(write_data_delayed),
        .EMA(3'b000),
        .RETN(1'b1),
        .Q(read_data)
    );


endmodule
```

A.10.7 SRAM FCW Wrapper

Wraps the wide FC weight SRAM used by the classifier.

code/input/RTL/SRAM/sram_FCW_wrapper.v

```
`timescale 1ns/1ps

// sram_FCW_wrapper: dedicated FC-weight memory.
//   Depth  = 288 words (FC input K = 6*6*8)
//   Width  = 80 bits  (10 output channels x 8-bit INT8 kernel)
//
// Written so Quartus (Cyclone V M10K) and Xilinx Vivado both infer
// BRAM. Keys to inference:
//   (1) memory read/write in ONE tiny always block, no reset.
//   (2) no reset of `douta` inside the RAM block -- the M10K output
//       register cannot be reset and still map to BRAM. Any reset of
//       `douta` forces the tool into FFs.
//   (3) `douta_valid` tracking is in a SEPARATE always block so it
//       does not contaminate the RAM template.
//   (4) 80-bit width > M10K's 40-bit ceiling -> Quartus automatically
//       concatenates two M10K blocks (fine as long as (1)-(3) hold).

module sram_FCW_wrapper # (
    parameter AW = 9,         // 2^9 = 512 >= 288
    parameter DW = 80
) (
```

```
    input                clka,
    input                rsta,       // sync active-high reset (valid flag only)
    input                ena,
    input                wea,
    input  [AW-1:0]      addra,
    input  [DW-1:0]      dina,

    output reg [DW-1:0]  douta,
    output reg           douta_valid
);
    // ----------------------------------------------------------------
    //  RAM block: plain sync write / sync read, no reset (BRAM template)
    // ----------------------------------------------------------------
    (* ramstyle = "M10K" *)
    reg [DW-1:0] mem [0:(1<<AW)-1];

    always @(posedge clka) begin
        if (ena) begin
            if (wea)
                mem[addra] <= dina;
            else
                douta <= mem[addra];
        end
    end

    // ----------------------------------------------------------------
    //  Valid flag: separate always block so it does NOT inhibit
    //  BRAM inference of ‘mem‘ / ‘douta‘.
    // ----------------------------------------------------------------
    always @(posedge clka) begin
        if (rsta)
            douta_valid <= 1’b0;
        else if (ena && !wea)
            douta_valid <= 1’b1;
        else
            douta_valid <= 1’b0;
    end

endmodule
```

A.10.8 Top SRAM A Integration

Integrates SRAM_A controller, address generation, wrapper, and stream-facing control.

code/input/RTL/SRAM/top_sram_A.v

```
module top_sram_A(
    input clk,
    input rst_n,

    input [2:0] layer_sel,
```

```
    input [1:0] data_sel,
    input       pass_id,
    input       start,
    output      busy,
    output      done,

    // Read path from SRAM_A to conv / pool pipeline (32b, shared).
    input        data_ready,
    output [31:0] read_data,
    output        data_valid,
    output        data_last,

    // Write path from pool pipeline back to SRAM_A (32b, shared).
    output               pool_ready,
    input signed [31:0]  pool_data,
    input                pool_valid,
    input                pool_last,

    // FC weight read path (80b, dedicated SRAM_FCW).
    input                fc_wt_ready,
    output [79:0]        fc_wt_read_data,
    output               fc_wt_valid,
    output               fc_wt_last
);

localparam ADDR_WIDTH    = 10;
localparam FCW_ADDR_WIDTH = 9;     // 512 depth covers 288 entries
localparam FCW_DATA_WIDTH = 80;    // 10 x 8-bit kernels
localparam FCW_LEN        = 10'd288;

// SRAM_A controller localparams -- mirror sram_A_controller encodings.
localparam [2:0] LAYER_PRELOAD = 3'd0;
localparam [2:0] LAYER_FC      = 3'd4;
localparam [1:0] SEL_CFG       = 2'd0;
localparam [1:0] SEL_WT        = 2'd1;
localparam [1:0] SEL_FCW       = 2'd3;

// ---------------------------------------------------------------
// Classify incoming transaction
// ---------------------------------------------------------------
wire fcw_preload_txn = (layer_sel == LAYER_PRELOAD) && (data_sel == SEL_FCW);
wire fcw_read_txn    = (layer_sel == LAYER_FC)      && (data_sel == SEL_WT);
wire fcw_txn         = fcw_preload_txn | fcw_read_txn;

// ===============================================================
// Existing 32b SRAM_A path (conv L1/L2/L3, FC CFG, preload CFG/WT)
// ===============================================================

wire [ADDR_WIDTH-1:0] addr;
wire [ADDR_WIDTH-1:0] base_addr;
wire [ADDR_WIDTH-1:0] length;
wire        read_mode;
wire        write_mode;
wire        read_en;
```

```
wire        write_en;
wire        addr_step;
wire        txn_done;
wire        pool_ready_int;
wire        unused_pool_last;

reg         read_last_d;
reg         txn_active;

// Active-transaction flag is shared: only ONE of (32b SRAM_A, SRAM_FCW) runs
// at a time. fcw_txn diverts start/done to the FCW datapath below.
wire        a32_start = start && !fcw_txn;

assign addr_step  = read_en | write_en;
assign read_en    = txn_active && read_mode  && data_ready;
assign write_en   = txn_active && write_mode && pool_valid;
assign pool_ready_int = txn_active && write_mode;
assign data_last  = read_last_d & data_valid;
assign unused_pool_last = pool_last;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        read_last_d <= 1'b0;
    else
        read_last_d <= txn_done && read_en;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        txn_active <= 1'b0;
    else if (a32_start)
        txn_active <= (length != {ADDR_WIDTH{1'b0}}) && (read_mode || write_mode);
    else if (txn_done)
        txn_active <= 1'b0;
end

Addr_Gen #(
    .ADDR_WIDTH(ADDR_WIDTH)
) u_addr_gen (
    .clk(clk),
    .rst_n(rst_n),
    .start(a32_start),
    .enable(addr_step),
    .base_addr(base_addr),
    .length(length),
    .addr(addr),
    .done(txn_done)
);

// L2 DATA write interleave: pass0 writes even offsets, pass1 writes odd offsets.
wire l2_data_write = (layer_sel == 3'd2) && (data_sel == 2'd2) && write_mode;
wire [ADDR_WIDTH-1:0] counter_val = addr - base_addr;
wire [ADDR_WIDTH-1:0] addr_final  = l2_data_write
    ? (base_addr + {counter_val[ADDR_WIDTH-2:0], pass_id})
```

```
    : addr;

sram_A_wrapper #(
    .AW(ADDR_WIDTH)
) u_sram_A_wrapper (
    .clk(clk),
    .rst_n(rst_n),
    .write_en(write_en),
    .read_en(read_en),
    .addr(addr_final),
    .write_data(pool_data),
    .read_data(read_data),
    .read_valid(data_valid)
);

sram_A_controller #(
    .ADDR_WIDTH(ADDR_WIDTH)
) u_sram_A_controller(
    .layer_sel(layer_sel),
    .data_sel(data_sel),
    .pass_id(pass_id),
    .base_addr(base_addr),
    .length(length),
    .read_mode(read_mode),
    .write_mode(write_mode)
);

// ============================================================================
// SRAM_FCW dedicated 80b path (FC weight preload + inference read)
// ============================================================================

wire                        fcw_start   = start && fcw_txn;
reg                         fcw_active;
reg                         fcw_rd_mode;   // 1 = inference read, 0 = preload write

wire                        fcw_rd_done;
wire [FCW_ADDR_WIDTH-1:0]   fcw_rd_addr;
wire [FCW_ADDR_WIDTH-1:0]   fcw_rd_base = {FCW_ADDR_WIDTH{1'b0}};
wire [FCW_ADDR_WIDTH-1:0]   fcw_rd_len  = FCW_LEN[FCW_ADDR_WIDTH-1:0];

// Read-side enable: on inference, advance when adapter is ready.
wire fcw_read_en = fcw_active && fcw_rd_mode && fc_wt_ready;

// Read address generator
Addr_Gen #(
    .ADDR_WIDTH(FCW_ADDR_WIDTH)
) u_fcw_read_addr (
    .clk(clk),
    .rst_n(rst_n),
    .start(fcw_start && fcw_read_txn),
    .enable(fcw_read_en),
    .base_addr(fcw_rd_base),
    .length(fcw_rd_len),
    .addr(fcw_rd_addr),
```

```
    .done(fcw_rd_done)
);

// Track last flag for 80b reads: pulses in the cycle the registered data
// for the final FCW entry becomes valid on the wrapper output.
reg fcw_read_last_d;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        fcw_read_last_d <= 1'b0;
    else
        fcw_read_last_d <= fcw_rd_done && fcw_read_en;
end

// Preload path: fcw_preload_packer accepts 32b host words (from pool_data)
// and emits 80b writes to SRAM_FCW. Beat count is driven by length=864 from
// the controller, but we manage it locally here via pool_valid beats.
reg [9:0] fcw_pl_beat_cnt;  // up to 864 beats
wire      fcw_pl_en = fcw_active && !fcw_rd_mode && pool_valid;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        fcw_pl_beat_cnt <= 10'd0;
    end else if (fcw_start && fcw_preload_txn) begin
        fcw_pl_beat_cnt <= 10'd0;
    end else if (fcw_pl_en) begin
        fcw_pl_beat_cnt <= fcw_pl_beat_cnt + 10'd1;
    end
end

wire fcw_pl_done = fcw_active && !fcw_rd_mode && (fcw_pl_beat_cnt == 10'd863) && pool_valid;

// fcw_active tracks the FCW transaction
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        fcw_active <= 1'b0;
    else if (fcw_start)
        fcw_active <= 1'b1;
    else if (fcw_rd_done || fcw_pl_done)
        fcw_active <= 1'b0;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        fcw_rd_mode <= 1'b0;
    else if (fcw_start)
        fcw_rd_mode <= fcw_read_txn;
end

// Packer instance (only driven during preload)
wire                        packer_wea;
wire [FCW_ADDR_WIDTH-1:0]   packer_addr;
wire [FCW_DATA_WIDTH-1:0]   packer_dina;
wire                        packer_up_ready;
```

```
222
223 fcw_preload_packer #(
224     .AW(FCW_ADDR_WIDTH),
225     .DW(FCW_DATA_WIDTH)
226 ) u_fcw_packer (
227     .clk(clk),
228     .rst_n(rst_n),
229     .start(fcw_start && fcw_preload_txn),
230     .up_valid(fcw_pl_en),
231     .up_data(pool_data),
232     .up_ready(packer_up_ready),
233     .wea(packer_wea),
234     .addra(packer_addr),
235     .dina(packer_dina)
236 );
237
238 // SRAM_FCW access port: mux read vs. write
239 wire                       fcw_ena   = fcw_read_en | packer_wea;
240 wire                       fcw_wea   = packer_wea;
241 wire [FCW_ADDR_WIDTH-1:0]  fcw_addr  = packer_wea ? packer_addr : fcw_rd_addr;
242 wire [FCW_DATA_WIDTH-1:0]  fcw_dina  = packer_dina;
243
244 wire [FCW_DATA_WIDTH-1:0]  fcw_douta;
245 wire                       fcw_douta_valid;
246
247 sram_FCW_wrapper #(
248     .AW(FCW_ADDR_WIDTH),
249     .DW(FCW_DATA_WIDTH)
250 ) u_sram_FCW (
251     .clka(clk),
252     .rsta(~rst_n),
253     .ena(fcw_ena),
254     .wea(fcw_wea),
255     .addra(fcw_addr),
256     .dina(fcw_dina),
257     .douta(fcw_douta),
258     .douta_valid(fcw_douta_valid)
259 );
260
261 assign fc_wt_read_data = fcw_douta;
262 assign fc_wt_valid     = fcw_douta_valid & fcw_rd_mode;
263 assign fc_wt_last      = fcw_read_last_d;
264
265 // ================================================================
266 // Shared done / busy
267 // ================================================================
268 assign busy      = txn_active | fcw_active;
269 // For FCW reads, use Addr_Gen.done pulse (fires on the last read enable) so
270 // done aligns with the existing 32b SRAM_A convention (done coincident with
271 // the last valid data beat being consumed).
272 wire fcw_done_pulse = fcw_active
273                       ? (fcw_rd_mode ? fcw_rd_done : fcw_pl_done)
274                       : 1'b0;
275 wire a32_write_done = write_en && txn_done;
```

```
wire a32_read_done  = read_mode && data_last;
assign done      = fcw_done_pulse
                   | (!fcw_active && (a32_write_done | a32_read_done));

// pool_ready: during non-FCW transactions behaves as before; during FCW
// preload the packer always accepts (up_ready=1) so we echo that through.
assign pool_ready = fcw_active
                    ? (!fcw_rd_mode ? packer_up_ready : 1'b0)
                    : pool_ready_int;

endmodule
```

A.10.9 Top SRAM B Integration

Integrates SRAM_B controller, address generation, wrapper, and pool/input stream routing.

code/input/RTL/SRAM/top_sram_B.v

```
module top_sram_B(
    input clk,
    input rst_n,

    input [2:0] layer_sel,
    input [1:0] data_sel,
    input       pass_id,
    input       start,
    output      busy,
    output      done,

    // Read path from SRAM_B to conv / FC pipeline.
    input         data_ready,
    output [31:0] read_data,
    output        data_valid,
    output        data_last,

    // Write path from pool pipeline back to SRAM_B.
    output               pool_ready,
    input signed [31:0]  pool_data,
    input                pool_valid,
    input                pool_last
);

localparam [2:0] LAYER_L3  = 3'd3;
localparam [2:0] LAYER_FC  = 3'd4;
localparam [1:0] SEL_DATA  = 2'd2;
localparam integer FC_BUF_AW = 7;   // 2^7 = 128 >= 72 words
localparam [5:0] FC_PASS_LEN = 6'd36;
localparam [6:0] FC_FULL_LEN = 7'd72;

wire [9:0] addr;
wire [9:0] base_addr;
```

```
wire [9:0] length;
wire       read_mode;
wire       write_mode;
wire       read_en;
wire       write_en;
wire       addr_step;
wire       txn_done;
wire       pool_ready_int;
wire       unused_pool_last;
wire [31:0] main_read_data;
wire        main_data_valid;
wire        main_data_last;

reg        read_last_d;
reg        txn_active;

// L3 output -> FC input reorder buffer.
// Instead of storing pass0[0..35], pass1[0..35] and fixing the order on the
// FC read side, write pass0 to even addresses and pass1 to odd addresses.
// This trades a little extra BRAM for a simpler FC read path.
wire fc_buf_write_mirror = (layer_sel == LAYER_L3) && (data_sel == SEL_DATA) && write_mode;
wire fc_buf_read_txn     = (layer_sel == LAYER_FC) && (data_sel == SEL_DATA) && read_mode;
wire main_start          = start && !fc_buf_read_txn;

assign addr_step        = read_en | write_en;
assign unused_pool_last = pool_last;
assign read_en          = txn_active && read_mode  && data_ready;
assign write_en         = txn_active && write_mode && pool_valid;
assign pool_ready_int  = txn_active && write_mode;
assign main_data_last  = read_last_d & main_data_valid;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        read_last_d <= 1'b0;
    else
        read_last_d <= txn_done && read_en;
end

// Assumes the FSM holds layer_sel / data_sel / pass_id stable until txn_done.
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        txn_active <= 1'b0;
    else if (main_start)
        txn_active <= (length != 10'd0) && (read_mode || write_mode);
    else if (txn_done)
        txn_active <= 1'b0;
end

Addr_Gen #(
    .ADDR_WIDTH(10)
) u_addr_gen (
    .clk(clk),
    .rst_n(rst_n),
    .start(main_start),
```

```
    .enable(addr_step),
    .base_addr(base_addr),
    .length(length),
    .addr(addr),
    .done(txn_done)
);

sram_B_wrapper #(
    .AW(10)
) u_sram_B_wrapper (
    .clk(clk),
    .rst_n(rst_n),
    .write_en(write_en),
    .read_en(read_en),
    .addr(addr),
    .write_data(pool_data),
    .read_data(main_read_data),
    .read_valid(main_data_valid)
);

sram_B_controller u_sram_B_controller(
    .layer_sel(layer_sel),
    .data_sel(data_sel),
    .pass_id(pass_id),
    .base_addr(base_addr),
    .length(length),
    .read_mode(read_mode),
    .write_mode(write_mode)
);

// Dedicated FC-order buffer. L3 still writes the original SRAM_B path; we
// mirror those writes into a second BRAM organized for FC sequential reads.
// pass0 writes even slots, pass1 writes odd slots, so FC can read 0..71
// with no address interleave logic.
(* ramstyle = "M10K" *)
reg [31:0] fc_input_mem [0:(1<<FC_BUF_AW)-1];

reg        fc_buf_rd_active;
reg [6:0]  fc_buf_rd_count;
reg [31:0] fc_buf_read_data_q;
reg        fc_buf_data_valid_q;
reg        fc_buf_read_last_d;

wire [9:0] counter_val = addr - base_addr;
wire       fc_buf_write_en = write_en && fc_buf_write_mirror;
wire       fc_buf_read_en  = fc_buf_rd_active && data_ready;
wire [6:0] fc_buf_wr_addr  = {counter_val[5:0], 1'b0} + pass_id;
wire       fc_buf_rd_done  = fc_buf_read_en  && (fc_buf_rd_count == (FC_FULL_LEN - 1'b1));
wire       fc_buf_data_last = fc_buf_data_valid_q && fc_buf_read_last_d;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        fc_buf_rd_active    <= 1'b0;
        fc_buf_rd_count     <= 7'd0;
```

```
        fc_buf_read_data_q  <= 32'd0;
        fc_buf_data_valid_q <= 1'b0;
        fc_buf_read_last_d  <= 1'b0;
    end else begin
        fc_buf_data_valid_q <= 1'b0;
        fc_buf_read_last_d  <= 1'b0;

        if (fc_buf_write_en)
            fc_input_mem[fc_buf_wr_addr] <= pool_data;

        if (start && fc_buf_read_txn) begin
            fc_buf_rd_active    <= 1'b1;
            fc_buf_rd_count     <= 7'd0;
        end else if (fc_buf_read_en) begin
            fc_buf_read_data_q  <= fc_input_mem[fc_buf_rd_count];
            fc_buf_data_valid_q <= 1'b1;
            fc_buf_read_last_d  <= fc_buf_rd_done;
            if (fc_buf_rd_done)
                fc_buf_rd_active <= 1'b0;
            else
                fc_buf_rd_count <= fc_buf_rd_count + 7'd1;
        end
    end
end

assign busy       = fc_buf_read_txn ? fc_buf_rd_active : txn_active;
assign done       = fc_buf_read_txn ? fc_buf_data_last
                                    : (read_mode ? main_data_last : (write_en && txn_done));
assign pool_ready = pool_ready_int;
assign read_data  = fc_buf_read_txn ? fc_buf_read_data_q : main_read_data;
assign data_valid = fc_buf_read_txn ? fc_buf_data_valid_q : main_data_valid;
assign data_last  = fc_buf_read_txn ? fc_buf_data_last : main_data_last;

endmodule
```