TÉLÉCOM PARIS · SETI MASTER · GUIDE HLS

From loops to a CNN accelerator

HLS

A short guide to HLS for Labs 6 and 7

Pipelining, unrolling, line buffers, fusion, DATAFLOW, the SoC around the IP

2026–2027 · about 20 minutes

What HLS builds: every loop becomes hardware, or time

```
for (oy = 0; oy < OUT_H; oy++) // 126
 for (ox = 0; ox < OUT_W; ox++) // 126
  for (oc = 0; oc < OUT_CH; oc++) // 64
   for (ic = 0; ic < IN_CH; ic++) // 3
    for (ky = 0; ky < K; ky++) // 3
     for (kx = 0; kx < K; kx++) // 3
       acc[oc] += in[ic][oy+ky][ox+kx]
                * w[oc][ic][ky][kx];
```

Lab 6 reference layer: $128 \times 128 \times 3 \to 126 \times 126 \times 64$, 27 MACs per output value, 27.4 M MACs per frame.

$$\text{cycles per frame} \approx \frac{\text{MACs per frame}}{\text{MACs per cycle}}$$

and FPS $= f_{clk}$ / cycles per frame.

PIPELINE

overlaps the iterations of a loop: a new one starts every II cycles, with one copy of the body.

UNROLL

copies the body: iterations run side by side, and need their operands in the same cycle.

Memory

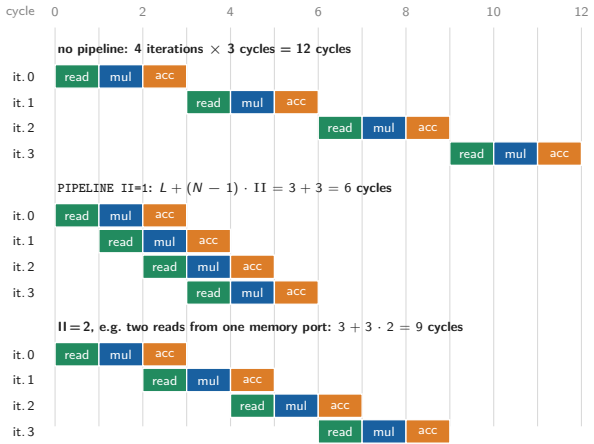
line buffers, partitioned arrays and FIFOs feed the multipliers on time.

DATAFLOW

overlaps whole functions: tasks run at once, connected by FIFOs.

The C++ fixes what is computed; the pragmas decide how much hardware computes it, and when.

Pipelining: start the next iteration before this one ends



Latency L and interval II

L: cycles of one iteration. II: cycles between two starts. N iterations take about $L + (N-1) \cdot \mathrm{II}$ cycles.

When II > 1

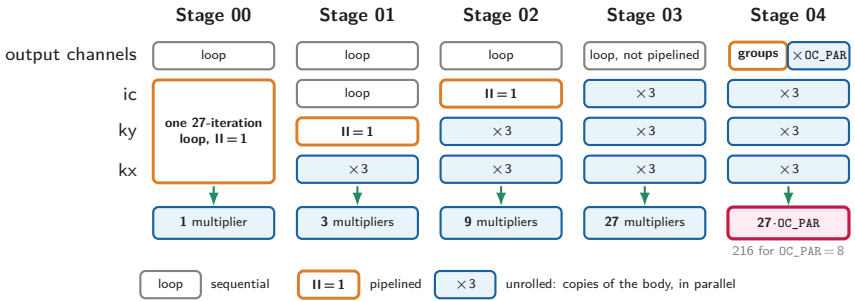
a resource is busy (one memory port for two reads), or an iteration needs a result that is not ready. The report says which.

In Lab 6

Stage 00 pipelines its 27-iteration MAC loop at II = 1 through one multiplier. Automatic pipelining is off: only the marked loops overlap.

Illustration: a three-step loop body (read, multiply, accumulate). The cycle counts of the lab come from the HLS reports.

Unrolling: from one multiplier to 216, on the same layer

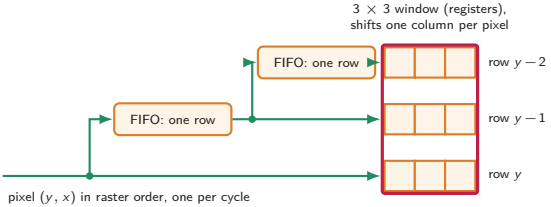


Unrolled copies need their operands *in the same cycle*: the window and the weights are split into registers (`ARRAY_PARTITION`), or the copies wait for a memory port.

The frugal question of Lab 6: the *smallest* `OC_PAR` that reaches 60 FPS, i.e. at most 1 666 666 cycles per frame at 100 MHz.

Lab 6 sources, `hls/0*/conv2d_top.cpp`: 27 MACs per output value (3 × 3 kernel, 3 input channels).

Line buffers: read every pixel once, keep only what the window still needs



- one cascade per input channel: $K-1=2$ FIFOs of W pixels;
- the layer keeps $C_{in}(K-1)W$ pixels on chip: $3 \times 2 \times 128 = 768$ in Lab 6, not the frame;
- the first complete window appears at $(y, x) = (K-1, K-1)$; from then on, one per pixel;
- Lab 6 declares each FIFO with depth $2W$: slack for scheduling, the logical delay stays one row.

Lab 6, Stages 01–04: `hls::stream<data_t> linebuf[IN_CH][K-1]`; Lab 7 uses the same idea in its convolution stage.

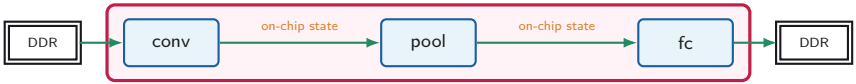
A line buffer turns a whole-image operation into a stream: memory drops from the frame to a few rows.

Fusion: the intermediate maps never go back to DDR

layer by layer



fused: one accelerator, maps on chip



Per image, Lab 7 compact network	DDR traffic	
layer by layer: every map written, then read again	7 584 B	≈7.6 kB
Conv–ReLU–Pool fused (as Lab 6 Stage 05), the FC separate	2 176 B	≈2.2 kB
whole network fused (Lab 7): the image in, 10 logits out	824 B	

Lab 6, one layer

the $126 \times 126 \times 64$ int8 map is 1 016 064 B: a DDR round trip costs 2.0 MB per image. Fusion changes no MAC; it removes traffic.

From the tensor shapes: 28×28 input, $26 \times 26 \times 4$ conv, $13 \times 13 \times 4$ pool, 10 logits. The weights stay on chip in all three cases.

DATAFLOW: the layers become tasks that run at the same time

`#pragma HLS DATAFLOW`: one process per layer, FIFOs of depth 16 between them



Three kinds of overlap

`UNROLL`: copies; `PIPELINE`: iterations; `DATAFLOW`: whole functions.

FIFOs, not maps

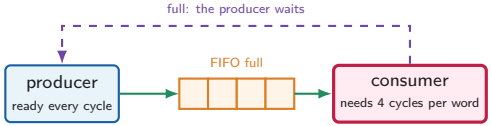
a FIFO holds a few values in flight; no stage stores a whole feature map it does not need.

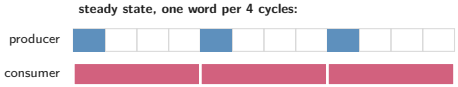
Count the words

each stage reads and writes exactly its count per image: one word too many or too few, and the pipeline blocks.

Lab 7 compact network, `compact_mnist_top.cpp`: the FIFOs are declared; you write `fc1` and connect the stages. Lab 6 Stage 06 is the same pattern on one layer.

Back-pressure: the slowest task sets the pace



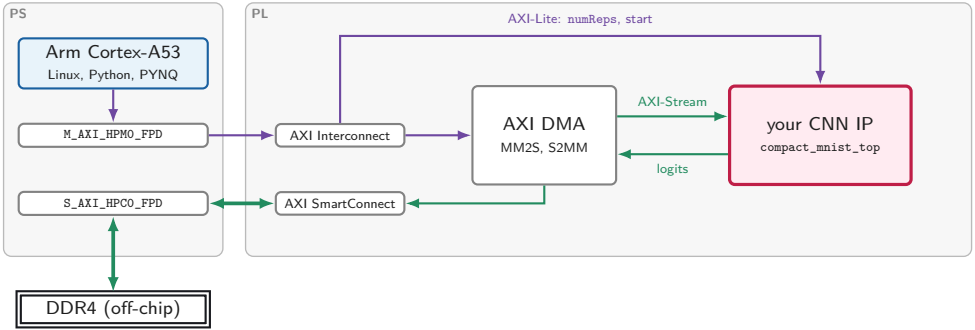


Lab 7 compact, `PAR=1`	**cycles/image**
read input	784
conv1	**3 164**
maxpool1	676
fc1	1 700
write logits	10

A full FIFO stalls its producer; an empty one stalls its consumer. The interval of the network is that of its slowest task: here `conv1`. With `conv1=2` the estimate moves the bottleneck to `fc1` (1 700 cycles): speeding up a faster task gains nothing.

`tools/cost_model.py` of Lab 7, first-order model (one loop iteration per cycle): an estimate, not a measurement; the HLS report and the board measure.

The SoC around the IP: the Lab 7 network on the KV260



Control
PYNQ programs the DMA (address, length) and the IP (`numReps`, start) through AXI-Lite.

Data
the DMA reads the images from DDR into a stream (one 32-bit word per pixel) and writes the 10 int32 logits of each image back.

TLAST
only on the last word of the whole call: it ends the DMA transfer. The weights live inside the IP.

Lab 7 block design (`vivado/build_overlay.tcl`). Lab 6 Version 04 instead masters DDR itself through four `m_axi` ports and needs no DMA.

What you do in Labs 6 and 7 – and how to read the evidence

Lab 6	Versions 00–04 of one Conv2D layer: unrolling and pipelining, one table the `OC_PAR` sweep: the smallest design that reaches 60 FPS your code: Conv, ReLU and MaxPool fused (two TODOs) optional: Stage 06 (`DATAFLOW`, AXI-Stream), Version 04 on the KV260
Lab 7	a trained MNIST CNN to one streaming IP: you write `fc1` and connect the `DATAFLOW` top bit-exact against the integer reference, then the parallelism of each layer optional: the accelerator on the KV260, with a USB camera

C simulation
is the C++ right? Integer results compared with a reference: `PASS` or not.

Synthesis report
latency, II, DSP, LUT, BRAM: estimates for the target clock, before any board.

Cost model
a first-order prediction to compare with the report, never a result.

Board
correctness first, then a measured time, with its method; no FPS without a measurement.

Choose the smallest hardware that meets the target, prove it is bit-exact, and say which number is a measurement.