CNN Image Classifier on FPGA

Che Li & Owen Wang

https://github.com/Owen-yd-Wang/Design-Project.git

Table of Contents

Problem Statement

Edge AI needs: warehouses, factories, drones need real-time image classification

GPU solutions cost $500-$2000+, draw 200W+, need internet or bulky servers

Small businesses can't justify this cost for simple classification tasks

Solutions:

Design a custom tiny CNN (~48K params) that fits entirely in on-chip BRAM

Quantize to int8 (47 KB) -- only 29% of available BRAM, leaving room for control logic

Verify with behavioral FPGA simulation before writing RTL

Pipeline

Reference — our pipeline stages:

1. Train a self designed CNN on CIFAR-10 dataset
2. Export to ONNX
3. Int8 quantization
4. Extract weights to .mem files
5. Behavioral FPGA simulation
6. RTL verification — MAC unit vs Python <- we are here
7. Synthesize on Basys 3 board fpga

Modified MobileNetV2

Why don't we just use MobileNetV2 ?:

- 3.5M params, 13.3 MB float32 → 15x too large for Basys 3(the board for the project)
- Depth wise separable convolutions need complex control logic

Our modified CNN (~48K params, 47 KB int8):

Conv1: 3→16, 3×3, pad=1 → ReLU → MaxPool2×2 → 16×16×16

Conv2: 16→32, 3×3, pad=1 → ReLU → MaxPool2×2 → 8×8×32

Conv3: 32→32, 3×3, pad=1 → ReLU → MaxPool2×2 → 4×4×32

FC1: 512→64 → ReLU

FC2: 64→10 (10 CIFAR-10 classes)

Conclusion

Total: 47,818 params | Int8 size: ~47 KB | BRAM usage: 29.4%

Working with Cognichip

Cognichip generated the following RTL modules:

- mac_uint8_int32.sv — MAC unit (uint8 × uint8 → uint32)
- line_buffer_3x3.sv — 3-line buffer for sliding window
- depthwise_conv3x3_engine.sv — 3×3 depthwise convolution engine
- pointwise_conv1x1_engine.sv — 1×1 pointwise convolution engine
- inverted_residual_block.sv — MobileNetV2 inverted residual block

Corresponding testbenches:

- tb_mac_uint8_int32.sv
- tb_depthwise_conv3x3_engine.sv
- tb_pointwise_conv1x1_engine.sv
- tb_inverted_residual_block.sv
- tb_complete_inference.sv — full integrated test
- Generated sim.vvp simulation binary (Icarus Verilog)

We used Cognichip to generate functional modules (MAC, line buffer, conv engines, etc.) and numerous testbenches. Individual module tests pass, but the integrated inference test is not yet working — we willdebug and resolve this in the next phase.

FPGA Considerations

Memory (BRAM):

- 225 KB total, no external DRAM
  - Weights: 47 KB (Only takes about 30% memory on the board)
  - Activations: largest intermediate = 16 KB
- Peak usage: 66.1 KB / 225 KB

Arithmetic:

- MAC unit uses unsigned uint8 - Accumulator is 32-bit to avoid overflow
- Time-multiplexed: one MAC unit shared across all layers

Results

Accuracy (CIFAR-10 test set, 10,000 images):

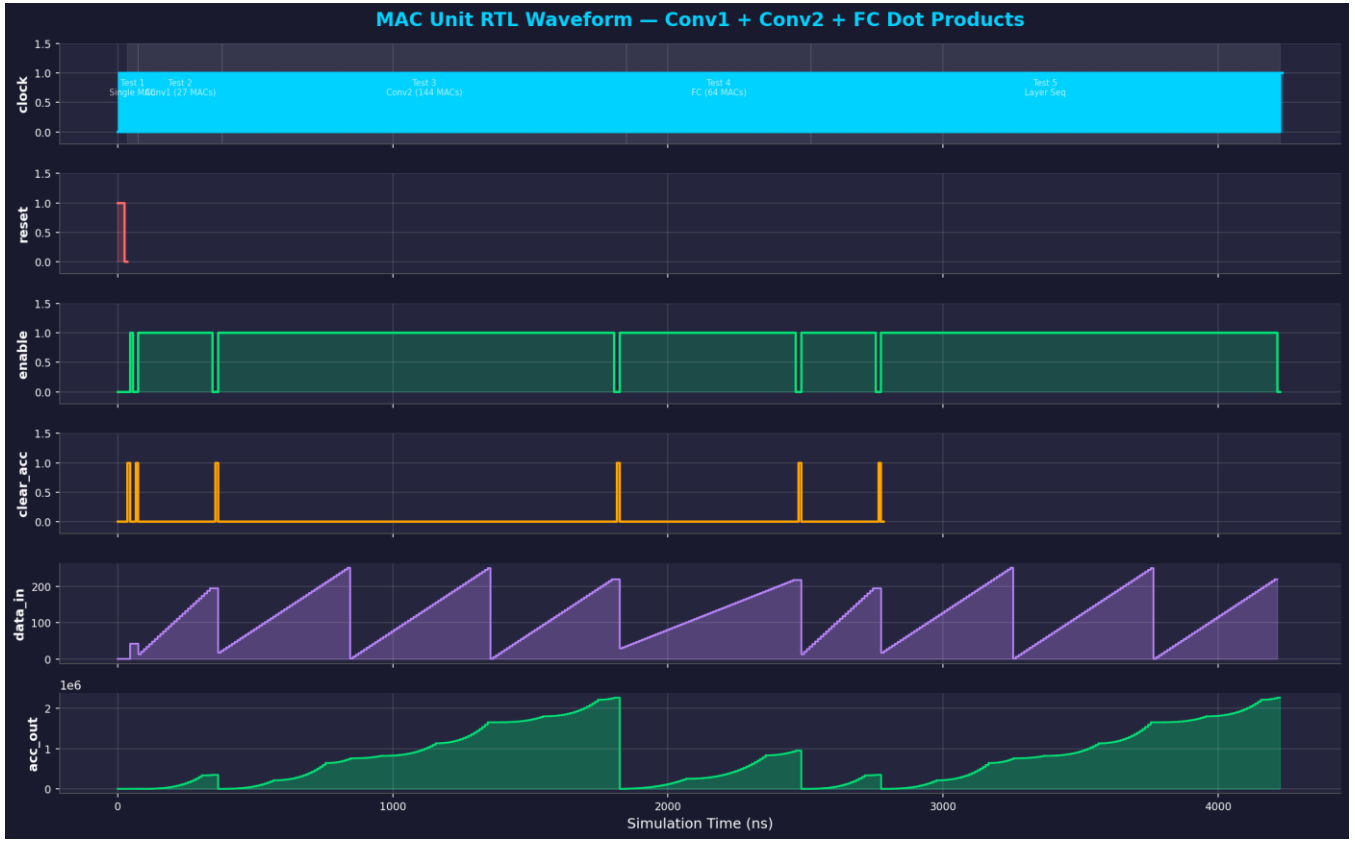
- Float32: F1 = 0.7400
- Int8: F1 = 0.7395 (delta: -0.0005)

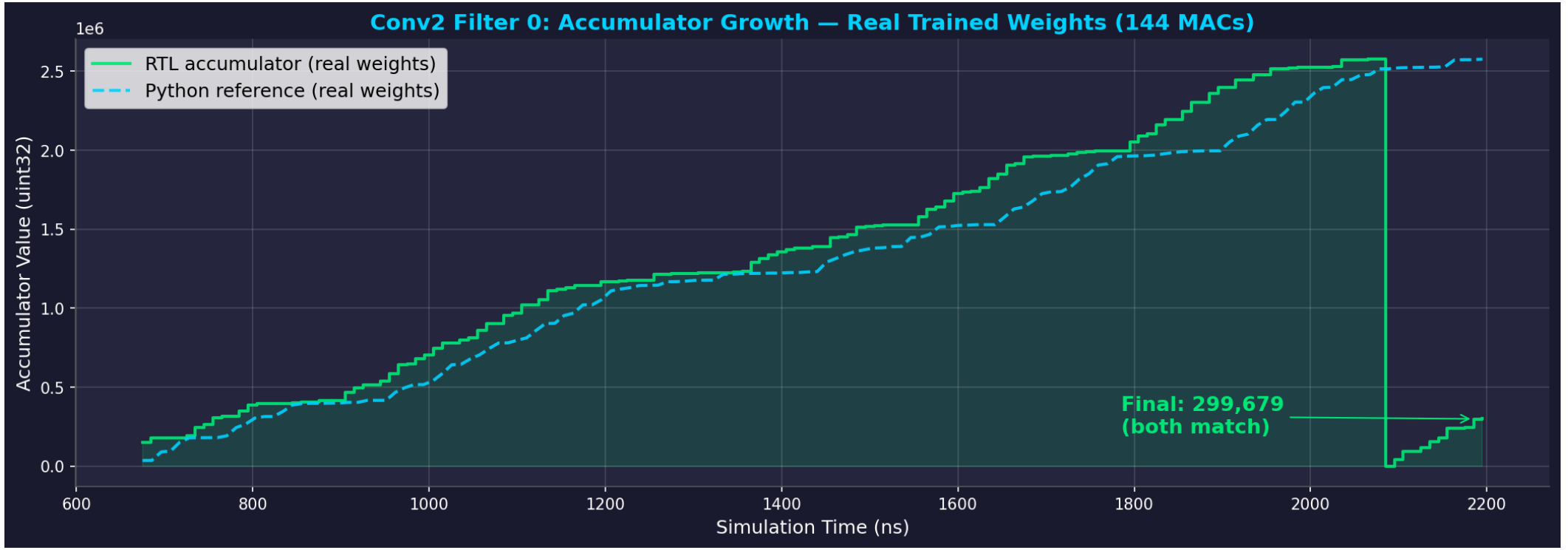
Performance (100 MHz):

- Throughput: 4,008 images/sec
- Latency: 0.25 ms per image

RTL Verification:

- 4/4 tests PASS — bit-exact match with Python




Conv2 Filter 0: Accumulator Growth — Real Trained Weights (144 MACs)
1e6
RTL accumulator (real weights)
Python reference (real weights)
Accumulator Value (uint32)
2.5
2.0
1.5
1.0
0.5
0.0
600
800
1000
1200
1400
1600
1800
2000
2200
Simulation Time (ns)
Final: 299,679
(both match)

Challenge and Lessons Learned

1.MobileNetV2 was 15x too large

- Original plan failed immediately

- Had to redesign from scratch for BRAM constraints

2.FC layers dominate weight memory

- FC1 alone: 32K of 48K total parameters (68%)

- Conv layers are surprisingly lightweight

3.Quantization tooling issues

- ONNX export needed extra dependencies (onnxscript)

- Int8 quantization requires calibration data

Name	CNN CIFAR-10 Accelerator on Basys 3 FPGA
Sponsor	N/A
Advisor	Ramesh Karri, Weihua Xiao

Legend		
	C	Complete
	S	Started
	NS	Not Started

	Month	FEB			MAR				APR					MAY
	Wednesday's Date	11	18	25	4	11	18	25	1	8	15	22	29	6
	Week #	1	2	3	4	5	6	7	8	9	10	11	12	13
Category	Milestone	Primary Owner	Support											
ch & Plannin	review: FPGA CNN accelerators	Che Li	Owen Wang	C										
h & Planning	Select target model architecture	Owen Wang	Che Li	S										
h & Planning	Basys 3 resource constraints analysis	Che Li	Owen Wang	C	S									
evelopment	Design tiny CNN architecture (~48K params	Che Li	Owen Wang		S	NS								
Development	Train CNN on CIFAR-10 (F1 >= 0.70)	Che Li	Owen Wang			NS	NS							
evelopment	Export to ONNX (float32 + int8 quantized)	Che Li	Owen Wang				NS							
evelopment	Verify int8 accuracy (minimal quantization los	Owen Wang	Che Li				NS	NS						
imulation	Build behavioral FPGA simulator (int8 arithme	Che Li	Owen Wang				NS	NS						
Simulation	Clock cycle & throughput analysis	Che Li	Owen Wang					NS						
imulation	BRAM/DSP resource utilization report	Owen Wang	Che Li					NS	NS					
sign	Design MAC unit (int8 multiply-accumulate)	Owen Wang	Che Li					NS	NS	NS				
sign	Design convolution datapath	Owen Wang	Che Li					NS	NS	NS				
sign	Design FC layer datapath	Che Li	Owen Wang						NS	NS				
sign	Design control FSM & memory interface	Owen Wang	Che Li							NS	NS			
sign	Integrate full CNN pipeline in Verilog	Owen Wang	Che Li								NS	NS		
ion	Testbench: MAC unit verification	Che Li	Owen Wang						NS	NS				
ion	Testbench: layer-level verification	Che Li	Owen Wang							NS	NS			
ion	Full system RTL simulation	Che Li	Owen Wang									NS	NS	
	ynthesis for XC7A35T	Owen Wang	Che Li										NS	
		Owen Wang	Che Li											

Milestones Tracker

Risks Management

#	Risk Category/Impacted Milestone	Risk Description	Impact	Mitigation Strategy	Primary Owner	Support	Type	Date Open	Target Date Close	Actual Date Close	% Complete
1	Model Development	Int8 quantization may degrade accuracy below acceptable threshold (F1 < 0.60)	2	Test multiple quantization strategies (symmetric, asymmetric); fall back to int16 for	Che Li	Owen Wang	Risk	2/11/2026	3/18/2026		0.25
2	RTL Design	Model may not fit in Basys 3 BRAM (225 KB) after adding control logic overhead	4	Keep model under 50% BRAM to leave headroom; behavioral simulation will estimate	Owen Wang	Che Li	Risk	2/11/2026	4/15/2026		0.25
3	RTL Design	Timing closure failure at 100 MHz on XC7A35T	3	Design with conservative clock target (50-75 MHz); optimize critical path if needed	Owen Wang	Che Li	Risk	2/11/2026	5/6/2026		0
4	Verification	RTL simulation results may not match behavioral Python simulation	3	Use Python golden model outputs for layer-by-layer comparison in testbench	Che Li	Owen Wang	Risk	2/11/2026	4/22/2026		0
5	Synthesis & Demo	Vivado synthesis may exceed available LUTs or flip-flops for control logic	3	Simplify FSM; share resources between layers; estimate LUT usage early	Owen Wang	Che Li	Risk	2/11/2026	5/6/2026		0
6	Model Development	MobileNetV2 (original candidate) is 15x too large for Basys 3 BRAM	5	Need to design a custom tiny CNN from scratch instead of using off-the-shelf model	Che Li	Owen Wang	Issue	2/11/2026	2/25/2026		0.25
7	Research & Planning	Survey existing FPGA CNN accelerator designs for reference	1	Review FINN framework documentation and academic papers	Che Li	Owen Wang	Task	2/11/2026	2/25/2026		0.25
8	Research & Planning	Document Basys 3 resource limits (BRAM, DSP, LUT, FF, clock)	1	Read XC7A35T datasheet and compile resource table	Owen Wang	Che Li	Task	2/11/2026	2/18/2026		0.5
9	RTL Design	Implement MAC unit in SystemVerilog (int8 inputs, int32 accumulator)	2	Use existing mac_uint8_int32.sv as starting point	Owen Wang	Che Li	Task	2/18/2026	3/18/2026		0
10	RTL Design	Design weight BRAM controller for sequential layer loading	3	Store all weights in BRAM at init; use address offsets per layer	Owen Wang	Che Li	Task	3/4/2026	4/1/2026		0
11	Verification	Create testbench comparing RTL output vs Python golden model per layer	2	Export Python int8 intermediate tensors as hex files for Verilog $readmemh	Che Li	Owen Wang	Task	3/18/2026	4/15/2026		0
12	Synthesis & Demo	Prepare demo: classify sample images on FPGA and display result	2	Pre-load test images via UART; display class on 7-segment display	Che Li	Owen Wang	Task	4/22/2026	5/6/2026		0

Looking ahead

Next steps:

- Synthesize full CNN

- FPGA board wiring actualize

- Multi-filter parallelism to improve throughput

- Live demo: camera → FPGA → classification result

References

- [1] CIFAR-10 Dataset — A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009
- [2] PyTorch — A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," NeurIPS 2019
- [3] ONNX Runtime — github.com/microsoft/onnxruntime
- [4] Xilinx Artix-7 FPGA Family Data Sheet (DS180)
- [5] Basys 3 Reference Manual — Digilent, digilent.com/reference/programmable-logic/basys-3/reference-manual
- [6] MobileNetV2 — M. Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," CVPR 2018
- [7] FINN Framework — M. Blott et al., "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," ACM TRETS 2018
- [8] Cognichip Co-Design Platform — github.com/Owen-yd-Wang/Design-Project
- [9] Icarus Verilog — S. Williams, iverilog.icarus.com
- [10] Integer Quantization for Deep Learning Inference — NVIDIA, arXiv:2004.09602, 2020