

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ



ĐỒ ÁN 1
KỸ THUẬT ĐIỆN TỬ- VIỄN THÔNG

ĐỀ TÀI
NHẬN DIỆN SỐ VIẾT TAY BẰNG FPGA
(FPGA-BASED HANDWRITTEN DIGIT RECOGNITION)

GVHD: THS. NGUYỄN TUẤN HÙNG
SV thực hiện: HOÀNG SÝ NHẤT 2111915

Thành phố Hồ Chí Minh, Tháng 05/2025



LỜI CẢM ƠN

Dầu tiên, em xin gửi lời cảm ơn đến tất cả các thầy cô trong Bộ Môn Điện Tử đã nhiệt tình dạy dỗ, truyền đạt kiến thức để em có thể tự tin làm và hoàn thành tốt đồ án này.

Tiếp theo, em cũng xin cảm ơn chân thành đến thầy Nguyễn Tuấn Hùng đã định hướng, giúp đỡ em nhiệt tình, hết mình để em có thể hoàn thành đồ án. Nhờ có sự giúp đỡ nhiệt tình của thầy mà em đã giải quyết được các vướng mắc xảy ra trong quá trình thực hiện. Những thắc mắc của em đều được thầy chỉ dạy tận tình, chu đáo.

Cuối cùng em cũng xin cảm ơn các bạn trong Khoa Điện-Điện Tử trường Đại Học Bách Khoa Thành Phố Hồ Chí Minh cũng đã giúp đỡ, trao đổi, thảo luận với em những gì mà em chưa biết để có thể hoàn thành tốt đồ án.

Dù đã cố gắng nhưng cũng không tránh khỏi sai sót, mong được Quý thầy cô xem qua và chỉ bảo thêm để em có thêm kinh nghiệm hoàn thành tốt hơn nữa các đề tài sau này.

Đây là hành trang kiến thức vô cùng quý báu mà em được Quý thầy cô khoa Điện-Điện Tử trường Đại Học Bách Khoa Thành Phố Hồ Chí Minh đã giúp đỡ, đã trang bị cho em. Để sau này, tốt nghiệp ra trường em có thể vận dụng những kiến thức trên vào trong công việc thực tế của em sau này.

Em xin chân thành cảm ơn



Mục lục

1 GIỚI THIỆU ĐỀ TÀI	8
1.1 Tổng quan	8
1.1.1 Bối cảnh nghiên cứu	8
1.1.2 Mục tiêu đề tài	8
1.2 Phương pháp tiếp cận	8
1.3 Ứng dụng mở rộng	9
1.3.1 Lĩnh vực tiềm năng	9
1.3.2 Hướng phát triển	9
1.4 Kế hoạch thực hiện	10
1.5 Kết luận	10
2 MÔ HÌNH CNN	11
2.1 Giới thiệu	11
2.1.1 Khái niệm cơ bản	11
2.1.2 Kiến trúc điển hình của CNN	11
2.1.3 Ưu điểm của CNN	12
2.1.4 Ứng dụng điển hình	12
2.1.5 Biến thể nâng cao của CNN	12
2.2 Xây dựng mô hình	13
2.2.1 Xử lý đầu vào thực tế	13
2.2.2 Thay thế và giảm bớt lớp fully connect	13
2.2.3 Sử dụng hàm kích hoạt ReLU	14
2.2.4 Binary Convolution (Tích chập nhị phân)	15
2.3 Kết quả	18
3 THIẾT KẾ PHẦN CỨNG	20
3.1 Tổng quan hệ thống	20
3.1.1 Giới thiệu	20
3.1.2 Sơ đồ thiết kế phần cứng	21
3.1.3 Mục tiêu thiết kế	22
3.1.4 Thông số kỹ thuật chính	22
3.1.5 Đặc Điểm Nổi Bật	22



3.2	Khối tích chập (Binary Convolution)	23
3.2.1	Ý Tưởng Thiết Kế	23
3.2.2	Thiết kế khối SIPO19	24
3.2.3	Thiết kế khối Conv	24
3.2.4	Kiểm Thủ & Xác Minh	25
3.2.5	Synthesize	26
3.3	Khối chuẩn hóa batch (Batch Nomalize)	27
3.3.1	Ý Tưởng Thiết Kế	27
3.3.2	Thiết kế khối Batch Normalize	28
3.3.3	Kiểm thử & Xác minh	29
3.3.4	Synthesize	30
3.4	Khối Maxpool	30
3.4.1	Ý Tưởng Thiết KẾ	30
3.4.2	Thiết kế khối Maxpool	31
3.4.3	Kiểm thử & Xác minh	32
3.4.4	Synthesize	33
3.5	Khối Fully Connect	34
3.5.1	Ý Tưởng Thiết KẾ	34
3.5.2	Thiết kế khối FullyConnect	36
3.5.3	Kiểm Thủ & Xác Minh	37
3.5.4	Synthesize	38
3.6	Khối BSRam	39
3.7	Các khói ROM chứa dữ liệu trọng số	41
3.8	Khối điều khiển (FSM)	43
3.8.1	Tổng quan	43
3.8.2	Giai đoạn LOAD	44
3.8.3	Giai đoạn CONV1	44
3.8.4	Giai đoạn CONV2	46
3.8.5	Giai đoạn MAXPOOL1	48
3.8.6	Giai đoạn CONV3	50
3.8.7	Giai đoạn CONV4	51
3.8.8	Giai đoạn MAXPOOL2	51
3.8.9	Giai đoạn CONV5	51



3.8.10 Giai đoạn GMPOOL	51
3.8.11 Giai đoạn FULLYCONN	52
3.9 CNN Top	53
3.9.1 Thiết kế khối CNN Top	53
3.9.2 Synthesize	54
3.9.3 Verification Plan	56
4 KIỂM TRA THIẾT KẾ	57
4.1 Simulation	57
4.2 Đưa lên kit thực tế	63
5 ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN	68
6 TÀI LIỆU THAM KHẢO	69



Danh sách hình vẽ

1.1	Biểu đồ Gantt thời gian dự án	10
2.1	Kiến trúc điển hình của CNN	11
2.2	Cấu trúc mô hình LeNet và AlexNet	13
2.3	Fully connect	14
2.4	Global max pooling	14
2.5	So sánh các phương pháp convolution	16
2.6	Model CNN	18
2.7	Kết quả training	18
2.8	Kết quả test data MNIST	19
3.1	Sơ đồ thiết kế phần cứng khối CNN	21
3.2	Sơ đồ khối Binary Conv	23
3.3	Dạng sóng ngõ ra với testcase mixed weight với input bằng -1 (Q4.8) . . .	26
3.4	Testbench Conv	26
3.5	Kết quả netlist synthesis khối Conv	27
3.6	Báo cáo timing khối Conv	27
3.7	Sơ đồ khối Batch Nomalize	28
3.8	Dạng sóng ngõ ra với testcase Scale only với $\theta = 1$ và ngõ vào bằng -1 . .	29
3.9	Testbench Bnorm	30
3.10	Kết quả netlist synthesis khối Bnorm	30
3.11	Maxpool flow chart	32
3.12	Dạng sóng ngõ ra cho testcase chức năng chính khối maxpool	33
3.13	Kết quả netlist synthesis khối Maxpool	33
3.14	Báo cáo timing khối Maxpool	34
3.15	Sơ đồ khối FullyConnect với pipeline 3 giai đoạn	35
3.16	Sơ đồ khối cây so sánh 5 tầng	35
3.17	Dạng sóng testbench với tất cả giá trị dense giống nhau	38
3.18	Testbench Fully Connect	38
3.19	Kết quả netlist synthesis khối FullyConnect	39
3.20	Báo cáo timing khối Fully Connect	39
3.21	BSRam ip core Gowin	41
3.22	Báo cáo trọng số từng lớp CNN	42



3.23 Sơ đồ trạng thái của khối điều khiển	43
3.24 Flow chart ở giai đoạn LOAD	44
3.25 Quá trình quét ảnh 2D	45
3.26 Padding cho lớp convolution	45
3.27 Flowchart CONV1	46
3.28 Tích chập với ngõ vào nhiều kênh	47
3.29 Thanh ghi đệm Bnorm	47
3.30 Flowchart CONV2	48
3.31 Maxpooling 2D	49
3.32 Flowchart MAXPOOL1	50
3.33 Kỹ thuật global max pool	51
3.34 Flowchart GlobalMaxPool	52
3.35 Flowchart FULLYCONN	53
4.1 Sơ đồ khối test simulation	57
4.2 Ảnh ngõ ra lớp Conv1	58
4.3 Ảnh ngõ ra lớp Conv2	58
4.4 Ảnh ngõ ra lớp Mpool1	59
4.5 Ảnh ngõ ra lớp Conv3	59
4.6 Ảnh ngõ ra lớp Conv4	60
4.7 Ảnh ngõ ra lớp Mpool2	60
4.8 Ảnh ngõ ra lớp Conv5	61
4.9 Kết quả ngõ ra Global max pooling	61
4.10 Giả lập dự đoán tập dữ liệu test MNIST trên Xcelium	62
4.11 Sơ đồ khối hệ thống thực tế	64
4.12 UART receiver fsm	64
4.13 Controller flow chart	65
4.14 LCD driver flow chart	65
4.15 Giao diện phần mềm	66
4.16 Kết quả nhận diện trên FPGA	66
4.17 Báo cáo timming hệ thống	67



Danh sách bảng

1	Module Binary Convolution Signal Definitions	24
2	Binary Convolution Verification Plan Table	25
3	Báo cáo sử dụng tài nguyên FPGA cho khối Conv	27
4	Module BNORM Signal Definitions	28
5	BNORM Module Verification Plan	29
6	Báo cáo sử dụng tài nguyên FPGA cho khối Bnorm	31
7	Module MaxPool Signal Definitions	31
8	MaxPool Module Verification Plan	32
9	Báo cáo sử dụng tài nguyên FPGA cho khối Maxpool	34
10	Module FullyConnect Signal Definitions	36
11	FullyConnect Verification Plan	37
12	Báo cáo tài nguyên FPGA cho FullyConnect	39
13	Bảng memory map hệ thống CNN (14-bit địa chỉ, 12-bit dữ liệu)	40
14	CNN Module Signal Definitions	53
15	Resource Usage Summary	54
16	Resource Utilization Summary	55
17	CNN Module Verification Plan	56
18	FPGA-based MNIST Recognition Models Comparison	63
19	Resource Utilization Summary	67

1 GIỚI THIỆU ĐỀ TÀI

Nhận diện chữ số viết tay (Handwritten Digit Recognition) là bài toán cơ bản trong lĩnh vực thị giác máy tính và học máy, với ứng dụng rộng rãi từ tự động hóa bưu chính đến nhập liệu thông minh. Đề tài này tập trung triển khai hệ thống nhận diện trên phần cứng FPGA (Field-Programmable Gate Array), nhằm tối ưu tốc độ xử lý, tiết kiệm năng lượng và đảm bảo độ chính xác cao.

1.1 Tổng quan

1.1.1 Bối cảnh nghiên cứu

- Nhu cầu thực tế:** Ứng dụng nhận diện chữ số viết tay ngày càng phổ biến trong các hệ thống tự động hóa (ví dụ: phân loại bưu kiện, đọc biển lai điện tử).
- Hạn chế của phần mềm truyền thống:** Các giải pháp dựa trên CPU/GPU tiêu tốn năng lượng và có độ trễ cao trong môi trường thời gian thực.
- Ưu thế của FPGA:** Khả năng xử lý song song, tiết kiệm điện năng (1/10 so với GPU) và độ linh hoạt trong thiết kế phần cứng.

1.1.2 Mục tiêu đề tài

- Xây dựng hệ thống nhận diện chữ số viết tay độ chính xác >90% trên bộ dữ liệu MNIST.
- Tối ưu tài nguyên FPGA (Logic Cells, BSRAM) để triển khai mô hình AI với độ trễ <100ms.
- Phát triển phần cứng tích hợp các ngoại vi để nhận diện số viết tay.

1.2 Phương pháp tiếp cận

- Dữ liệu:** Sử dụng bộ MNIST làm cơ sở.
- Mô hình:** Tối ưu hóa kiến trúc mạng neural, xây dựng mô hình dựa theo AlexNet và LeNet kết hợp với kiến trúc CNN binary-weight để giảm lương tài nguyên sử dụng trên fpga.



- **Công cụ:** Thư viện Tensorflow và Keras để tạo mô hình AI, ModelSim và Xcellium để giả lập thiết kế. Quartus hoặc Gowin để synthesis thiết kế phần cứng và nạp lên kit.

1.3 Ứng dụng mở rộng

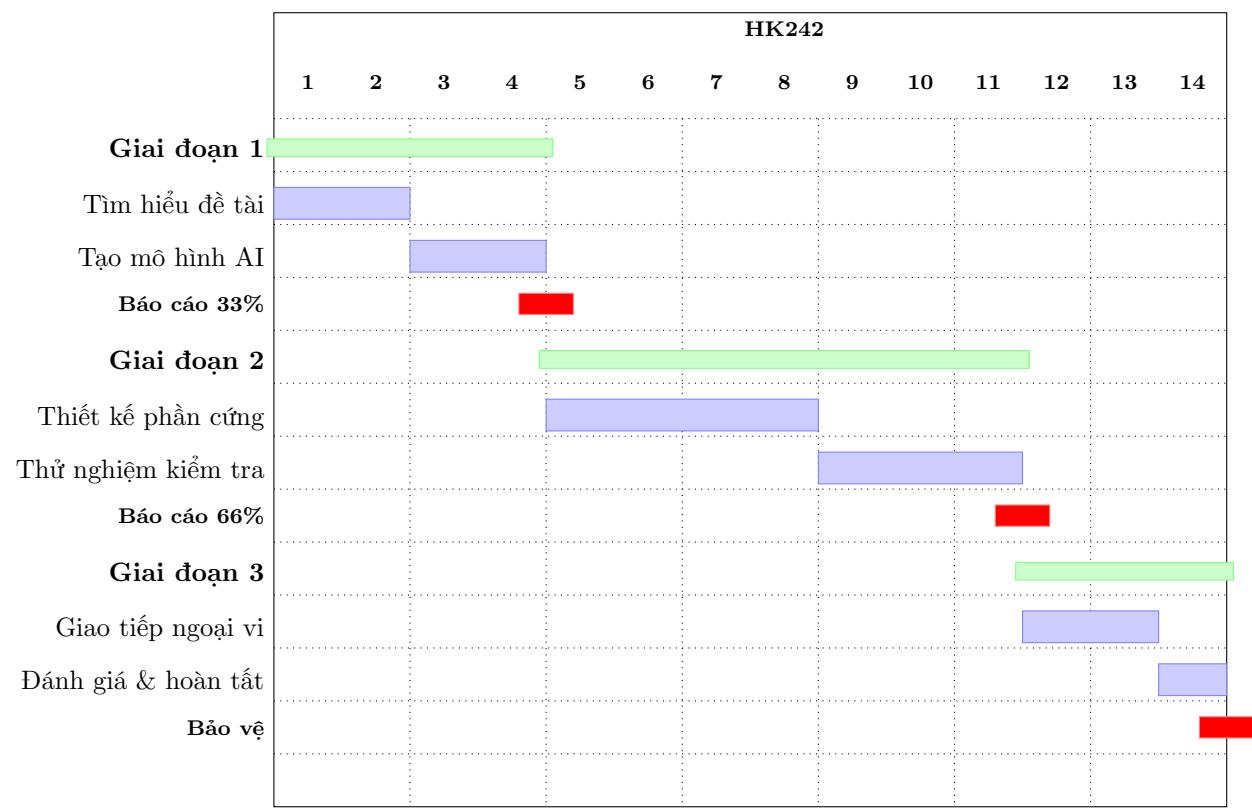
1.3.1 Lĩnh vực tiềm năng

- **Y tế:** Đọc chỉ số thiết bị đo tự động (máy ECG, máy xét nghiệm).
- **Giao thông:** Nhận diện biển số xe tại trạm thu phí không dừng.
- **Nông nghiệp:** Phân loại sản phẩm theo trọng lượng và ký hiệu viết tay.

1.3.2 Hướng phát triển

- **Tích hợp AI Edge:** Kết hợp FPGA với vi điều khiển (VD: Xilinx Zynq Ultra-Scale+).
- **Nâng cấp mô hình:** Thử nghiệm với kiến trúc MobileNetV3 hoặc Transformer tối ưu.
- **Pipeline hóa:** Chia các lớp CNN thành các stage độc lập để tăng throughput.

1.4 Kế hoạch thực hiện



Hình 1.1: Biểu đồ Gantt thời gian dự án

1.5 Kết luận

Đề tài kết hợp giữa **AI** và **thiết kế phần cứng**, mở ra hướng nghiên cứu tối ưu hiệu năng cho các bài toán nhận diện trong điều kiện tài nguyên hạn chế.

2 MÔ HÌNH CNN

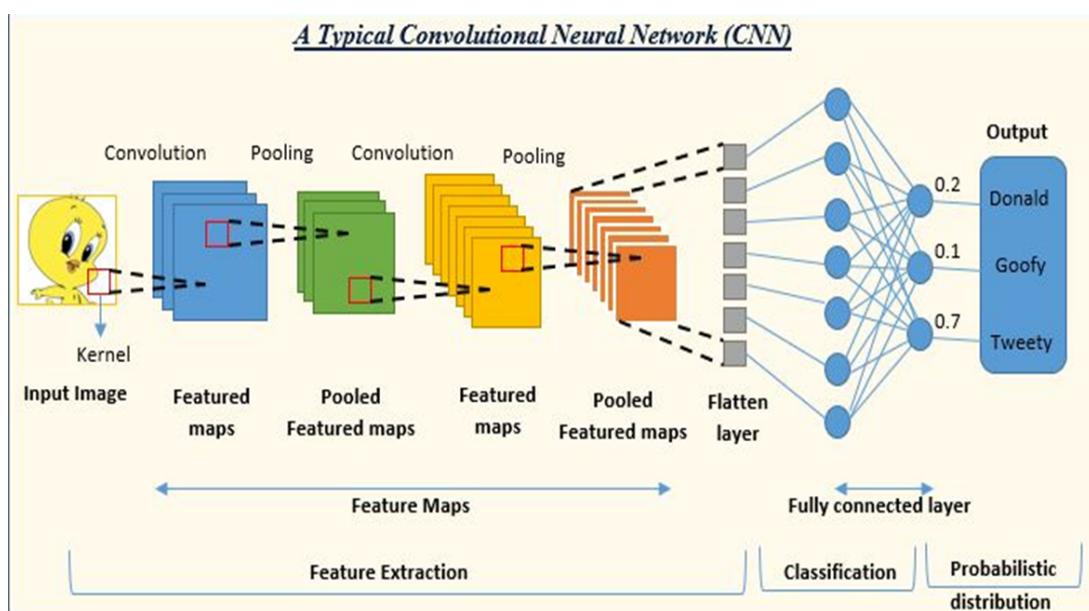
2.1 Giới thiệu

2.1.1 Khái niệm cơ bản

CNN (Convolutional Neural Network) là một kiến trúc mạng neural chuyên dụng cho xử lý dữ liệu có cấu trúc không gian như ảnh, video hoặc tín hiệu đa chiều. Khác với mạng neural truyền thống (DNN), CNN tập trung vào việc tự động trích xuất đặc trưng cục bộ thông qua các phép toán tích chập (convolution), giúp giảm đáng kể tham số mô hình mà vẫn đạt độ chính xác cao.

2.1.2 Kiến trúc điển hình của CNN

Một mô hình CNN cơ bản bao gồm các lớp chính sau:



Hình 2.1: Kiến trúc điển hình của CNN

• Lớp Convolution (Conv):

- Sử dụng bộ lọc (kernel) quét qua ảnh để phát hiện đặc trưng (ví dụ: cạnh, texture).
- Ví dụ: Kernel 3×3 với stride=1 và padding="same" giữ nguyên kích thước ảnh.

• Lớp Activation Function:

- ReLU (Rectified Linear Unit) là hàm kích hoạt phổ biến nhất, giúp mô hình hội tụ nhanh hơn.



- Sigmoid là một hàm kích hoạt phi tuyến có dạng chữ "S", biến đổi đầu vào thành khoảng giá trị từ 0 đến 1, thường được sử dụng trong các mô hình phân loại nhị phân để dự đoán xác suất.

- **Lớp Pooling (Max/Average):**

- Giảm kích thước không gian (downsampling), tăng tính bất biến với nhiễu.
- Ví dụ: Max Pooling 2×2 giảm 50

- **Lớp Fully Connected (FC):** Kết nối toàn bộ đặc trưng để phân loại (thường dùng ở cuối mô hình).

2.1.3 Ưu điểm của CNN

- **Hiệu quả với dữ liệu ảnh:** Tận dụng tính chất cục bộ (local connectivity) và trọng số chia sẻ (shared weights), giảm overfitting.
- **Tiết kiệm tài nguyên:** Ít tham số hơn so với mạng DNN nhờ cơ chế convolution.
- **Khả năng mở rộng:** Dễ dàng kết hợp với các kiến trúc hiện đại (ResNet, Transformer).

2.1.4 Ứng dụng điển hình

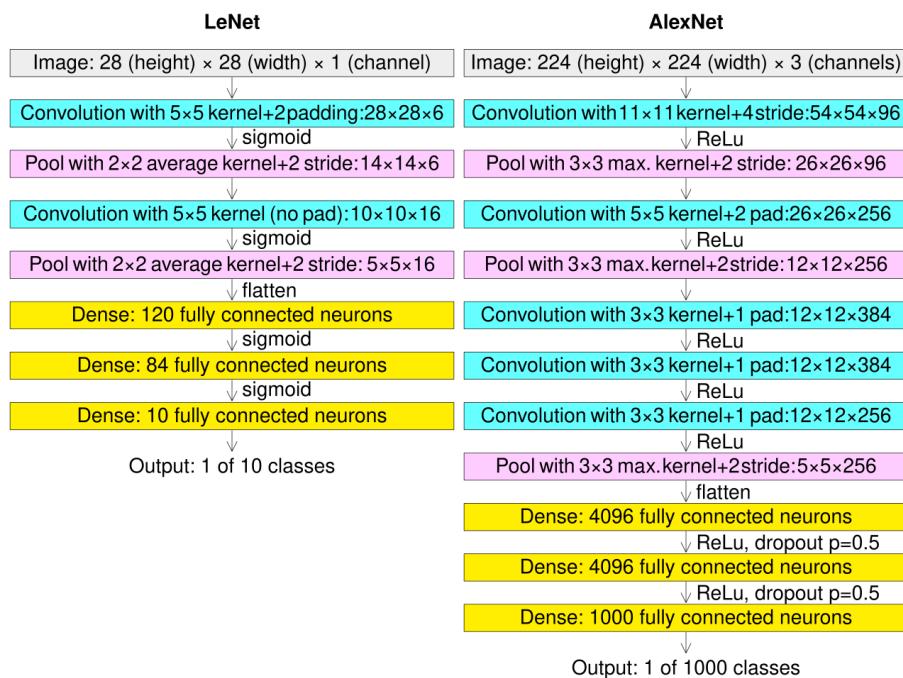
- **Nhận diện đối tượng:** Phát hiện khuôn mặt, chữ số viết tay (MNIST).
- **Phân đoạn ảnh y tế:** Xác định khối u trong ảnh MRI.
- **Xử lý video:** Theo dõi chuyển động trong camera an ninh.

2.1.5 Biến thể nâng cao của CNN

- **ResNet:** Giải quyết vấn đề vanishing gradient bằng kết nối tắt (skip connection).
- **MobileNet:** Tối ưu cho thiết bị di động nhờ convolution tách rời (depthwise separable conv).
- **EfficientNet:** Cân bằng giữa độ chính xác và kích thước mô hình qua scaling coefficient.

2.2 Xây dựng mô hình

Nền tảng thiết kế: Mô hình CNN được đề xuất kế thừa các nguyên tắc cốt lõi từ AlexNet (khả năng trích xuất đặc trưng đa lớp với kernel lớn) và LeNet (kiến trúc đơn giản, phù hợp phần cứng), đồng thời để tối ưu hóa cho triển khai trên FPGA ta cần phải thay đổi mô hình theo các kỹ thuật sau.



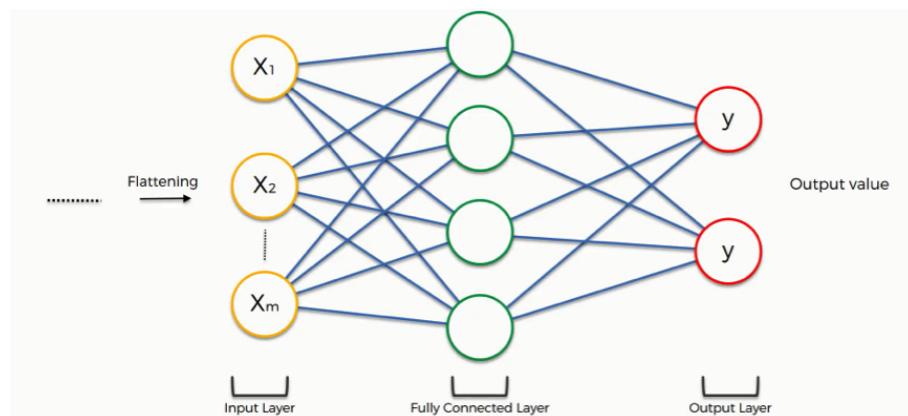
Hình 2.2: Cấu trúc mô hình LeNet và AlexNet

2.2.1 Xử lý đầu vào thực tế

Tích hợp tiền xử lý ảnh trực tiếp trên FPGA (grayscale, resize về 28x28 như LeNet) để giảm latency. Và cũng phù hợp với Dataset MNIST đồ án này sử dụng để đào tạo mạng CNN.

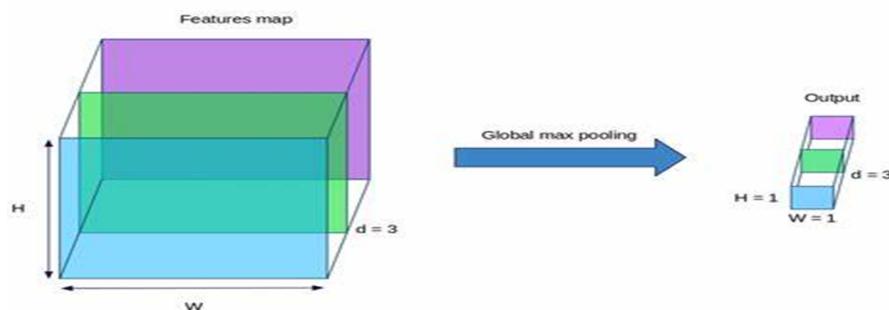
2.2.2 Thay thế và giảm bớt lớp fully connect

Các mạng phỏ biến cho nhận dạng chữ số viết tay bao gồm LeNet và AlexNet, những mạng này thường được sử dụng cho các tác vụ phân loại hình ảnh. Tuy nhiên, trong triển khai phần cứng, các mạng này có thể không phù hợp lắm do số lượng tham số lớn, vì vậy cần phải điều chỉnh các mạng này. Đặc biệt, các lớp fully connected yêu cầu rất nhiều trọng số (weight) và các bộ DSP. Ví dụ nếu số input fully connect là 100, output là 10 thì tổng số trọng số là 100x10 và 10 bias cho lớp fully connect này. Do đó để tiết kiệm tài nguyên triển khai, chúng ta cần giảm bớt trọng số của các lớp này. Một phương pháp



Hình 2.3: Fully connect

phổ biến là sử dụng global max pooling để thay thế một phần trọng số của các lớp fully connected.



Hình 2.4: Global max pooling

2.2.3 Sử dụng hàm kích hoạt ReLU

Hàm kích hoạt **ReLU** (Rectified Linear Unit) được định nghĩa:

$$\text{ReLU}(x) = \max(0, x) \quad (1)$$

Lợi ích:

- **Tính toán hiệu quả:**

$$\text{Phép toán} \Rightarrow \begin{cases} 0 & \text{nếu } x < 0 \\ x & \text{nếu } x \geq 0 \end{cases} \quad (2)$$

Chỉ cần bộ so sánh (comparator) và multiplexer đơn giản

- **Tiết kiệm tài nguyên:**

- Không dùng phép nhân/phép chia phức tạp



- Chiếm ít LUTs (Look-Up Tables) trên FPGA
- **Tận dụng pipeline:** Throughput \uparrow nhờ latency thấp
- **Tương thích lượng tử hóa:** ReLU quan hệ tuyến tính \Rightarrow dễ dàng INT8/BNN hóa

2.2.4 Binary Convolution (Tích chập nhị phân)

Binary Weight Networks lượng tử hóa trọng số thành 2 giá trị $\{-1, +1\}$ nhưng giữ nguyên đầu vào full-precision. Công thức cơ bản:

$$\mathbf{Y} = \mathbf{X} * \text{sign}(\mathbf{W}) \quad (3)$$

với:

- **X:** Đầu vào full-precision (float32)
- **W:** Trọng số nhị phân hóa qua hàm $\text{sign}(\cdot)$
- $*$: Phép tích chập thông thường

Quá trình nhị phân hóa

$$\text{sign}(W_{ij}) = \begin{cases} +1 & \text{nếu } W_{ij} \geq 0 \\ -1 & \text{nếu } W_{ij} < 0 \end{cases} \quad (4)$$

Kèm theo scaling factor α để bù sai số:

$$\alpha = \frac{1}{n} \|\mathbf{W}\|_1 = \frac{1}{n} \sum_{i,j} |W_{ij}| \quad (5)$$

Ưu điểm phần cứng

- **Tiết kiệm bộ nhớ:** Giảm 32x so với float32
- **Tối ưu phép nhân:**

$$x \times \text{sign}(w) = \begin{cases} +x & \text{nếu } w \geq 0 \\ -x & \text{nếu } w < 0 \end{cases} \quad (6)$$

- Triển khai trên FPGA chỉ cần:

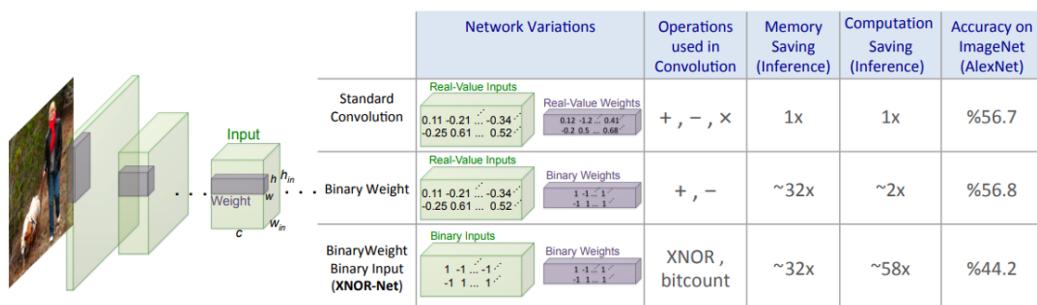
- Multiplexer chọn giữa $+x$ và $-x$
- Không cần DSP blocks cho phép nhân

Triển khai trên FPGA

```
module binary_weight_layer (
    input [31:0] input_data,
    input binary_weight,
    output reg [31:0] result
);
    always @(*) begin
        result = (binary_weight == 1'b1) ? input_data : -input_data;
    end
endmodule
```

So sánh với Full-Precision và Xnor-net

Thông số	Full-Precision	Binary Weight	XNOR-Net
Bộ nhớ trọng số	32 bit/parameter	1 bit/parameter	1 bit/parameter
Bộ nhớ activations	32 bit	32 bit	1 bit
Phép toán nhân	Float multiplier	MUX + inverter	XNOR + popcount
Tốc độ (FPGA)	1x	2.7x	58x
Dộ chính xác	Baseline	Giảm 1-3%	Giảm 8-12%
Năng lượng tiêu thụ	100%	35%	12%
Scaling factor	Không cần	α cho weights	α, β cho activations
Triển khai phần cứng	DSP blocks	LUTs + MUX	Pure LUTs



Hình 2.5: So sánh các phương pháp convolution

Xây dựng lớp Binary Convolution trên Tensorflow Vì Tensorflow không hỗ trợ sẵn nên ta cần phải dựa vào cấu trúc lớp Convolution mặc định để xây dựng lớp Binary Convolution. Với quá trình *Forward* của lớp Convolution mặc định ta sẽ nhị phân hóa trọng số bằng cách sử dụng hàm *tf.sign()*, giữ nguyên trọng số và *Gradient* cho quá trình *Backpropagation* để hội tụ nhanh hơn.



```
1 @tf.custom_gradient
2 def binarize(weights):
3     """Function binarize weights with Straight-Through Estimator gradient"""
4     def grad(dy, variables=None):
5         return dy # Keep input gradient
6     return tf.sign(weights), grad # Binarize + custom gradient
```

Giải thích:

- `@tf.custom_gradient`: Decorator định nghĩa gradient tùy chỉnh
- `tf.sign(weights)`: Chuyển weights thành giá trị nhị phân $\{-1, 1\}$
- `grad(dy)`: Triển khai Straight-Through Estimator (STE) - truyền nguyên gradient đầu vào khi backpropagation

Với scaling-factor α để đơn giản phần lập trình ta đơn giản thay thế bằng cách thêm một lớp Batch Normalize sau lớp Binary Convolution.

Batch Normalization (BN) là kỹ thuật chuẩn hóa dữ liệu theo batch trong quá trình huấn luyện mạng neural, giúp ổn định và tăng tốc độ hội tụ.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (7)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (8)$$

Trong đó:

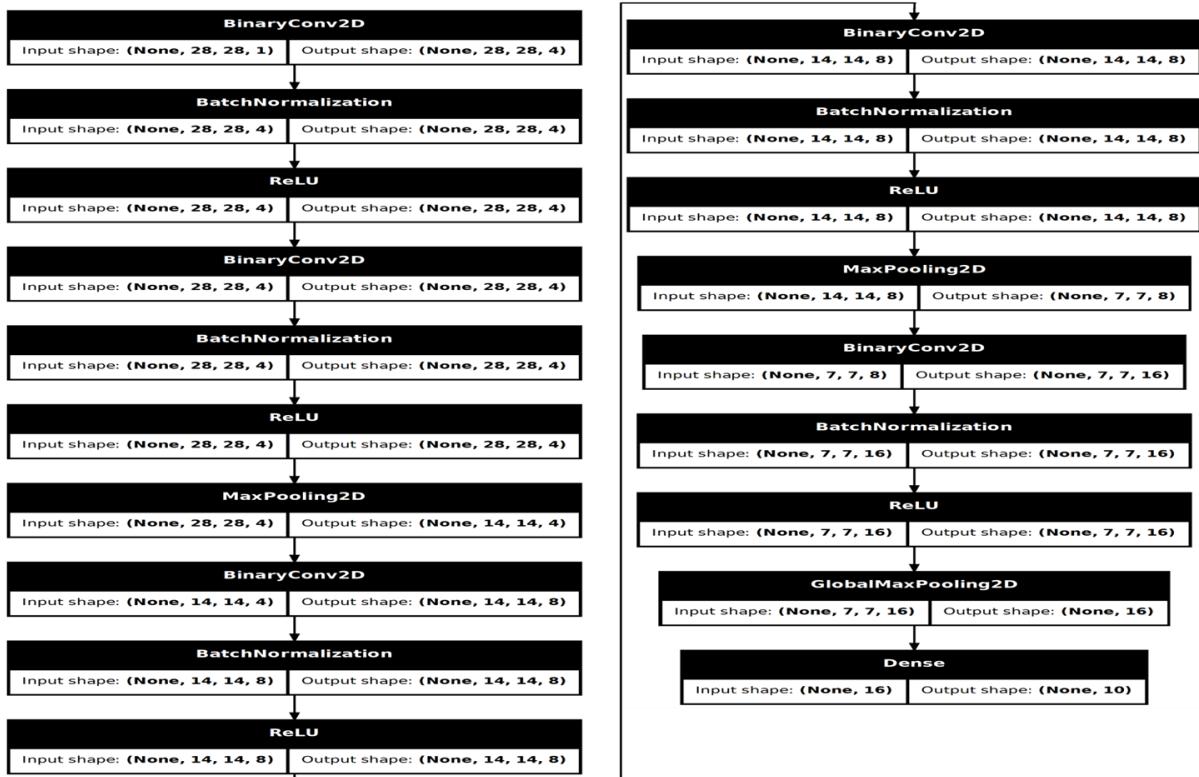
- μ_B, σ_B^2 : Trung bình và phương sai của batch
- γ, β : Tham số học được (scale và offset)
- ϵ : Hằng số tránh chia cho 0 (thường 10^{-5})

Công dụng chính

- **Ổn định huấn luyện**: Giảm hiện tượng "internal covariate shift"
- **Hỗ trợ learning rate lớn**: Gradient ổn định hơn
- **Giảm phụ thuộc khởi tạo**: Ít nhạy cảm với giá trị ban đầu
- **Regularization ẩn**: Noise từ thống kê batch giảm overfitting
- **Tối ưu inference**: Sử dụng mean/variance cố định khi dự đoán

2.3 Kết quả

Từ cở sở trên ta xây dựng mô hình gồm 5 lớp Binary Convolution với sau mỗi lớp tích chập là khối Batch Normalize và Activation ReLU, 2 khối Max Pooling, 1 lớp Global Max Pooling và cuối cùng là 1 lớp Fully Connect.



Hình 2.6: Model CNN

Kết quả training

```

Epoch 10: val_loss did not improve from 0.39759
468/468 1s 3ms/step - accuracy: 0.9297 - loss: 0.2401 - val_accuracy: 0.7621 - val_loss: 0.7581
Epoch 11/30
468/468 0s 32ms/step - accuracy: 0.9166 - loss: 0.2837
Epoch 11: val_loss improved from 0.39759 to 0.25419, saving model to training_1/cp6.keras
468/468 16s 34ms/step - accuracy: 0.9166 - loss: 0.2837 - val_accuracy: 0.9260 - val_loss: 0.2542
Epoch 12/30
1/468 12s 27ms/step - accuracy: 0.9297 - loss: 0.2184
Epoch 12: val_loss did not improve from 0.25419
468/468 1s 3ms/step - accuracy: 0.9297 - loss: 0.2184 - val_accuracy: 0.8455 - val_loss: 0.4825
Epoch 13/30
467/468 0s 32ms/step - accuracy: 0.9211 - loss: 0.2668
Epoch 13: val_loss did not improve from 0.25419
468/468 16s 35ms/step - accuracy: 0.9211 - loss: 0.2667 - val_accuracy: 0.8748 - val_loss: 0.3925
Epoch 14/30
1/468 13s 30ms/step - accuracy: 0.9609 - loss: 0.1611
Epoch 14: val_loss did not improve from 0.25419
468/468 1s 3ms/step - accuracy: 0.9609 - loss: 0.1611 - val_accuracy: 0.8685 - val_loss: 0.4034
Epoch 15/30
467/468 0s 33ms/step - accuracy: 0.9282 - loss: 0.2414
Epoch 15: val_loss did not improve from 0.25419
468/468 17s 37ms/step - accuracy: 0.9282 - loss: 0.2414 - val_accuracy: 0.7314 - val_loss: 0.8058
Epoch 16/30
1/468 12s 26ms/step - accuracy: 0.9609 - loss: 0.1259
Epoch 16: val_loss did not improve from 0.25419
468/468 1s 3ms/step - accuracy: 0.9609 - loss: 0.1259 - val_accuracy: 0.7422 - val_loss: 0.8337

```

Hình 2.7: Kết quả training



Quá trình training model sử dụng Adam optimizer với learning rate 10^{-4} có early stop ta có kết quả nhận diện đúng trên tập data *Train* là 91.66% loss 0.2873 và kết quả trên tập data *Evaluate* là 92.6% loss 0.2542

```
313/313 - 2s - 5ms/step - accuracy: 0.9236 - loss: 0.2533
Restored model, accuracy: 92.36%
```

Hình 2.8: Kết quả test data MNIST

Kết quả test trên tập data *Test* của MNIST có độ chính xác khoảng 92.36%.



3 THIẾT KẾ PHẦN CỨNG

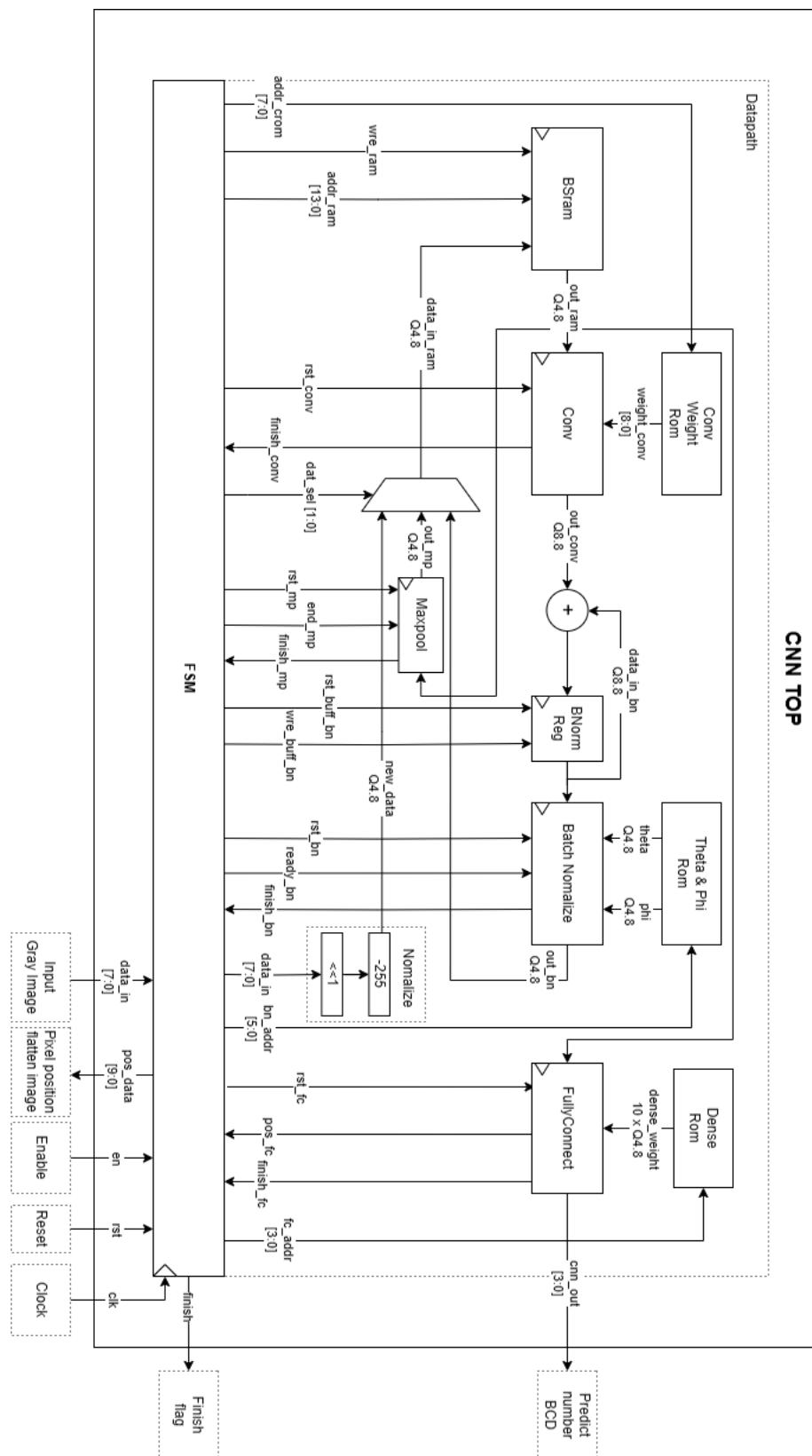
3.1 Tổng quan hệ thống

3.1.1 Giới thiệu

Hệ thống CNN (Convolutional Neural Network) được thiết kế trong tài liệu này tập trung vào việc phân tích ảnh xám và dự đoán chữ số từ 0 đến 9 dưới dạng mã BCD (Binary-Coded Decimal), kết hợp giữa kiến trúc phần cứng tối ưu và thuật toán học sâu. Mô-đun CNN áp dụng ý tưởng thiết kế hệ thống số về mô-đun điều khiển/xử lý dữ liệu, chia toàn bộ mạng nhận dạng thành hai khối chính:

- **Khối điều khiển (máy trạng thái FSM)** Phát lệnh điều khiển dựa trên tín hiệu và trạng thái từ khối xử lý dữ liệu.
- **Khối xử lý dữ liệu (Datapath)** Nhận lệnh từ khối điều khiển để thực hiện tính toán và phản hồi trạng thái hiện tại.

3.1.2 Sơ đồ thiết kế phần cứng



Hình 3.1: Sơ đồ thiết kế phần cứng khối CNN



3.1.3 Mục tiêu thiết kế

- **Xử lý ảnh hiệu quả:** Tiếp nhận đầu vào là điểm ảnh xám 8-bit, kết hợp theo dõi vị trí để tối ưu bộ nhớ.
- **Tối ưu tài nguyên:** Sử dụng định dạng dấu phẩy tĩnh (Q4.8, Q8.8) để cân bằng độ chính xác và chi phí tính toán. Việc sử dụng nhiều các khối chuẩn hóa giúp cho dữ liệu sau khi tính toán giao động từ $[-1; 1]$ do đó chọn định dạng số có dấu Q4.8 với tầm từ $[-8; 7.99609375]$ lượng tử với độ phân giải $\frac{1}{256}$. Dữ liệu trung gian Q8.8 để dự phòng cho trường hợp ngõ vào của khối Batch Normalize là tổng tích chập nhiều Window (Tối đa là 8 Window với mô hình của đề tài) với nhau, ngoài ra Q8.8 còn sử dụng trong ruột của khối Fully Connect.

3.1.4 Thông số kỹ thuật chính

Thiết kế pipeline xử lý ảnh xám đầu vào qua 4 giai đoạn chính: Convolution → Batch Normalization → Max Pooling → Fully Connected.

- **Định dạng dữ liệu:**

- Đầu vào: 8-bit grayscale + 10-bit vị trí (pos_data[9:0]).
- Trung gian: Q4.8/Q8.8 fixed-point.
- Đầu ra: 4-bit BCD (cnn_out[3:0])

- **Bộ nhớ:**

- BRAM 14-bit address (addr_ram[13:0])
- 3 ROM chứa:
 - * Conv weights (9-bit)
 - * Theta/Phi params (Q4.8) (Đề cập sau)
 - * Dense layer weights (Q4.8)

3.1.5 Đặc Điểm Nổi Bật

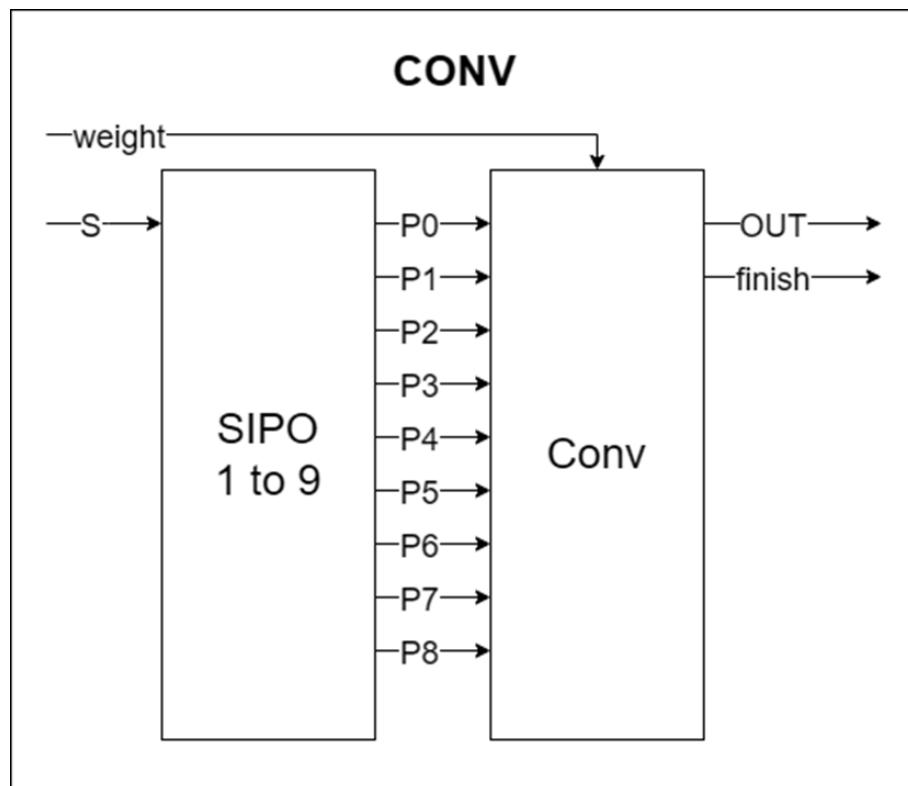
- **Kiến trúc phần cứng rõ ràng:** ách biệt khối xử lý (Convolution, BN, Pooling, FC) và điều khiển (FSM), đồng bộ qua các tín hiệu *finish* và *rst*.
- **Bộ nhớ chuyên biệt:** ROM lưu trọng số tích chập (9-bit) và tham số chuẩn hóa (Q4.8), BRAM lưu dữ liệu trung gian với địa chỉ 14-bit.

- **Đầu ra linh hoạt:** Kết quả dự đoán 4-bit BCD, phù hợp với các hệ thống nhúng hoặc giao tiếp vi điều khiển.

3.2 Khối tích chập (Binary Convolution)

3.2.1 Ý Tưởng Thiết Kế

- Thiết kế một bộ tích chập (convolution unit) tối ưu cho xử lý tín hiệu số với:
 - Đầu vào dạng fixed-point Q4.8 (12-bit)
 - Đầu ra dạng Q8.8 (16-bit)
 - Hỗ trợ kernel 3x3 (9 trọng số)
- Kết hợp 2 module chính:
 - *SIPO19*: Chuyển đổi serial-to-parallel 1 to 9 cho từng bit dữ liệu
 - *conv*: Thực hiện phép tích chập bằng cách ghép 12 bộ SIPO19 độc lập và thực hiện phép cộng trừ dựa theo trọng số nhị phân.



Hình 3.2: Sơ đồ khối Binary Conv

- Sử dụng counter 4-bit để điều khiển luồng dữ liệu.



- Triển khai phép nhân qua thao tác XOR và cộng đơn giản.

$$d_w[i] = (n_data[i] \wedge \{12\{w[i]\}\}) + \{11'b0, w[i]\}$$

3.2.2 Thiết kế khối SIPO19

```
module SIPO19 (
    input clk, reset, serial_in, shift,
    output reg [8:0] parallel_out
);
```

- Chức năng:
 - Dịch 9 bit nối tiếp → song song
 - Giữ kết quả ở *parallel_out* khi *shift=1*
- Đặc điểm thiết kế:
 - Thanh ghi 9-bit *shift_register* dịch trái mỗi chu kỳ clock
 - Reset đồng bộ xóa cả thanh ghi và đầu ra

3.2.3 Thiết kế khối Conv

Bảng 1: Module Binary Convolution Signal Definitions

Signal Name	Direction	Width	Description
clk	Input	1	System clock
rst	Input	1	Active-high reset
en_conv	Input	1	Convolution enable
data_in	Input	12	Input data (Q4.8 format)
w	Input	9	Weight vector (0=add, 1=subtract)
finish	Output	1	Operation completion flag
out	Output	16	Result (Q8.8 format)

- Cơ chế hoạt động:
 - Giai đoạn load dữ liệu:
 - * 12 SIPO19 thu nhận từng bit của *data_in* khi *sync=1*
 - * Mỗi SIPO19 tạo 9 bản sao song song của 1 bit dữ liệu
 - Giai đoạn tính toán:

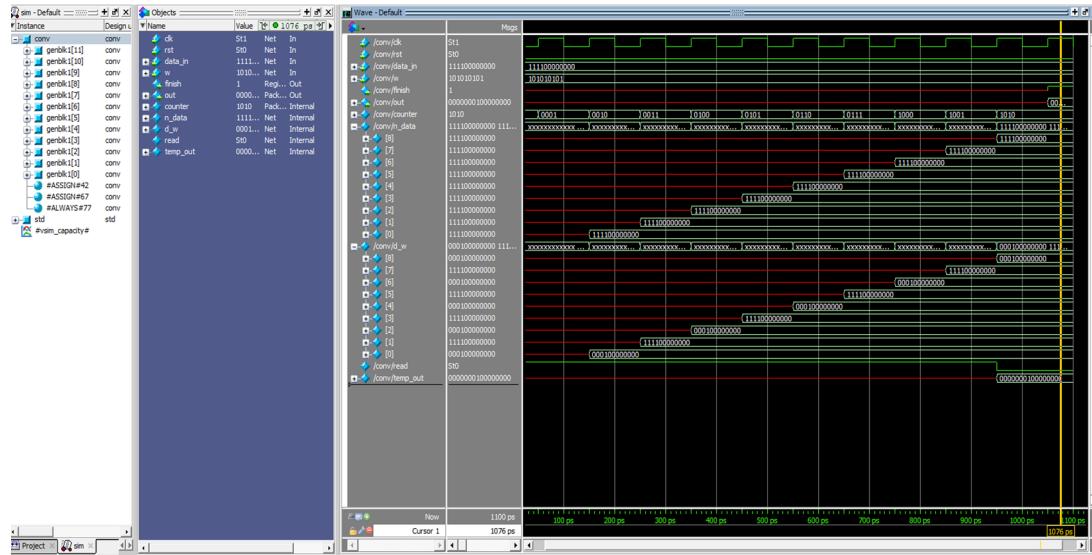


- * Tạo 9 phiên bản dữ liệu đã chuẩn hóa ($d_w[0:8]$)
- * Tính tổng có dấu với sign extension (bit mở rộng dấu)
- Giai đoạn hoàn tất conv: Counter đếm đến 10 chu kỳ → báo *finish*
- Phép Toán Tích Chập
 - Cơ chế nhân nhị phân: Với mỗi trọng số $w[i]$
 - * Nếu $w[i]=1$: Thực hiện phép đảo bit và cộng 1 (bù 2)
 - * Nếu $w[i]=0$: Giữ nguyên giá trị
 - Mở rộng dấu 4 bit trước khi tính tổng (từ 12-bit → 16-bit)

3.2.4 Kiểm Thử & Xác Minh

Bảng 2: Binary Convolution Verification Plan Table

Test Category	Test Case	Verification Method	Expected Result
Reset Verification	Power-on reset	Assert reset, then release	All registers cleared, outputs zero
SIPO Functionality	Serial data shifting	Input known pattern, verify parallel output	Correct parallel output after 9 cycles
Basic Convolution	All weights=0 (pure add)	Input fixed value, check output	Output = $9 \times$ input
Basic Convolution	All weights=1 (pure subtract)	Input fixed value, check output	Output = $-9 \times$ input
Mixed Operation	Alternating weights	Input fixed value, mixed weights	Correct weighted sum
Timing Control	Enable/disable during operation	Toggle en_conv at various times	Proper operation only when enabled
Finish Signal	Operation completion	Monitor finish signal	Asserted after 10 cycles
Boundary Cases	Maximum input value	Input 12'h7FF	Correct saturated output
Boundary Cases	Minimum input value	Input 12'h800	Correct negative output
Reset Recovery	Reset during operation	Assert reset mid-calculation	Immediate termination, outputs zero



Hình 3.3: Dạng sóng ngõ ra với testcase mixed weight với input bằng -1 (Q4.8)

Quan sát thấy khí tín hiệu $finish=1$ kết quả ngõ ra bằng 1 (Q8.8) do ngõ vào cỗ định bằng -1 và với $weight = 9'b101010101$ sẽ thực hiện 5 phép trừ và 4 phép cộng kết quả bằng 1 là chính xác.

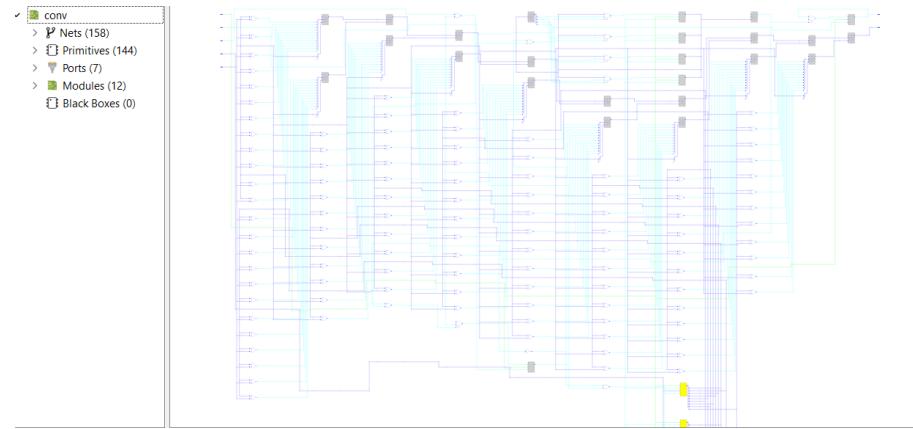
Viết code testbench và sử dụng chức năng monitor của Modelsim để kiểm tra các testcase còn lại.

```
# Test Case 1: All weights 0 (pure addition)
# Output: 0900 (expected ~9.0 in Q8.8)
#
# Test Case 2: All weights 1 (pure subtraction)
# Output: f700 (expected ~-9.0 in Q8.8)
#
# Test Case 3: Mixed weights
# Output: 0200 (expected ~16.0 in Q8.8)
#
# Test Case 4: Multiple cycles
# Output: 0180
#
# Test Case 5: Reset during operation
# Output after reset: 0000 (should be 0)
#
# Test Case 6: Edge cases
# Max input output: 07ff
# Min input output: b800
#
# All tests completed
# ** Note: $finish : C:/Users/ADMIN/Desktop/HK242/DA1/cnverilog/tb/tb_conv.sv(131)
# Time: 845 ns Iteration: 0 Instance: /tb_conv
# l
```

Hình 3.4: Testbench Conv

3.2.5 Synthesize

Sử dụng phần mềm Gowin FPGA Designer để chạy synthesis ta có kết quả Netlist như hình 3.5.



Hình 3.5: Kết quả netlist synthesis khối Conv

Ta có được báo cáo tài nguyên sử dụng và timing như sau.

Bảng 3: Báo cáo tài nguyên FPGA cho khối Conv

Tài nguyên	Sử dụng/Tổng (đơn vị)	Utilization
Logic	335 (189 LUT, 128 ALU, 3 RAM16) / 8640	4.0%
Register	105 / 6693	2.0%
• Register as Latch	0 / 6693	0.0%
• Register as FF	105 / 6693	2.0%
BSRAM	0 / 26	0.0%

Max Frequency Summary:

NO.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	clk	50.000(MHz)	176.699(MHz)	3	TOP

Hình 3.6: Báo cáo timing khối Conv

3.3 Khối chuẩn hóa batch (Batch Normalize)

3.3.1 Ý Tưởng Thiết Kế

Nhắc lại phương trình tính Batch Normalize, mỗi quan hệ vào ra của dữ liệu được thể hiện qua công thức:

$$y_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta \quad (9)$$

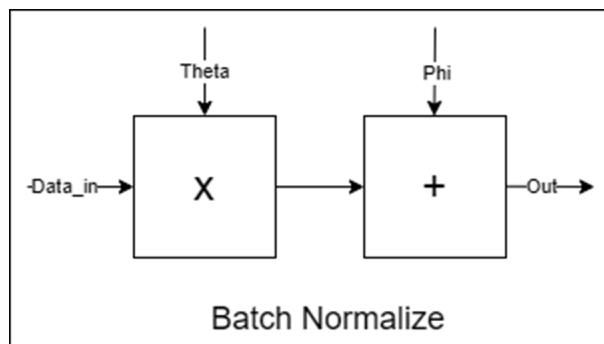
Ta thấy khối này sử dụng 4 thông số trọng số, để giảm khói lượng tính toán và lưu trữ ta đơn giản phương trình như sau:

$$y_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2}} + \beta = \theta x_i + \phi \quad (10)$$

với:

- $\theta = \gamma/\sqrt{\sigma^2}$ (hệ số tỷ lệ)
- $\phi = -\mu\gamma/\sqrt{\sigma^2} + \beta$ (hệ số dịch chuyển)

Với 2 hệ số θ và ϕ sẽ được tính toán trước và lượng tử hóa về kiểu dữ liệu Q4.8 do dựa theo khảo sát thì hệ số θ luôn dương và bé hơn 1 và hệ số ϕ của mô hình nằm trong khoảng kiểu dữ liệu Q4.8.



Hình 3.7: Sơ đồ khối Batch Normalize

3.3.2 Thiết kế khối Batch Normalize

Bảng 4: Module BNORM Signal Definitions

Signal Name	Direction	Width	Description
clk	Input	1	System clock
rst	Input	1	Active-high reset
ready	Input	1	Data processing enable
data_in	Input	16	Input data (Q8.8 format)
theta	Input	12	Scale factor $\gamma/\sqrt{\sigma^2}$
phi	Input	12	Shift factor $-\mu\gamma/\sqrt{\sigma^2} + \beta$
finish	Output	1	Operation completion flag
out	Output	12	Result (Q4.8 format)

Bên trong khối này sẽ thực hiện tính toán ở kiểu dữ liệu Q8.8 sau đó cắt giảm 4 bit đầu để về kiểu dữ liệu Q4.8 và thực hiện luôn nhiệm vụ cuat lớp ReLU bằng cách kiểm tra bit MSB để quyết định kết quả ngõ ra.

Các luồng tính toán chính

- Nhân với hệ số θ :
 - Mở rộng dấu ngõ vào 16-bit \rightarrow 28-bit (Đảm bảo kết quả cho phép nhân có dấu 16-bit x 12-bit)
 - Nhân với θ (12-bit) sau khi mở rộng zero (θ luôn dương)

- Cộng với hệ số ϕ :
 - Lấy 12 bit trung gian (bit 19-8) của kết quả nhân
 - Cộng với hệ số dịch chuyển ϕ
- ReLU
 - Phát hiện giá trị âm qua bit dấu (bit 11)
 - Tự động cắt về 0 nếu âm (ReLU đơn giản)

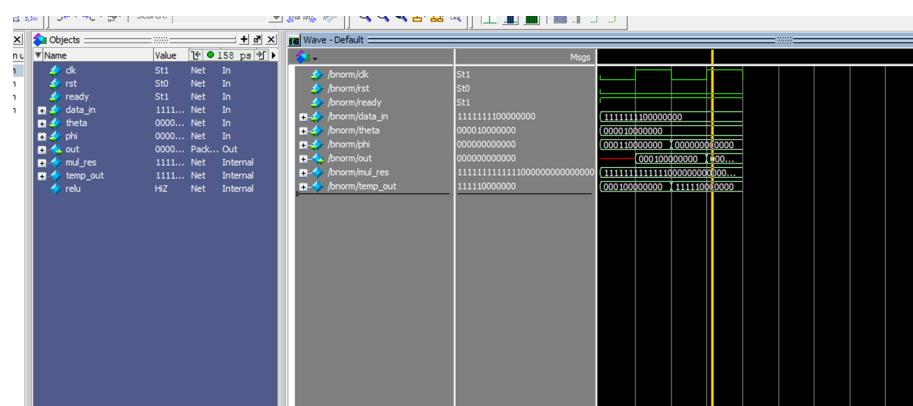
Điều khiển trạng thái

- Tín hiệu *finish* được set sau 1 chu kỳ khi *ready* active
- Reset đồng bộ thiết lập lại trạng thái

3.3.3 Kiểm thử & Xác minh

Bảng 5: BNORM Module Verification Plan

Test Category	Verification Method	Expected Result
Reset Verification		
Power-on reset	Assert reset, then release	All registers cleared, outputs zero
Basic Functionality		
Identity transform	Set $\theta = 1.0, \phi = 0.0$	Output = Input (Q4.8 scaled)
Scale only	Set $\theta = 2.0, \phi = 0.0$	Output = $2 \times$ Input
Shift only	Set $\theta = 1.0, \phi = 0.5$	Output = Input + 0.5
Boundary Cases		
Max input value	Input 16'h7FFF (Q8.8)	Correct saturated output
Min input value	Input 16'h8000 (Q8.8)	Correct negative handling
Zero variance	Set $\theta = 0.0, \phi = 0.5$	Output = 0.5 (constant)
Timing Control		
Ready signal timing	Toggle ready during operation	Proper output only when ready=1
Finish signal	Monitor finish signal	Asserted 2 cycles after ready
Reset Recovery		
Mid-operation reset	Assert reset during calculation	Immediate termination, outputs zero



Hình 3.8: Dạng sóng ngõ ra với testcase Scale only với $\theta = 1$ và ngõ vào bằng -1

Quan sát khi tín hiệu $finish=1$ kết quả ngõ ra bằng -1 (Q4.8) do ngõ vào bằng -1 (Q8.8) với hệ số $\theta = 1$ và $\phi = 0$.

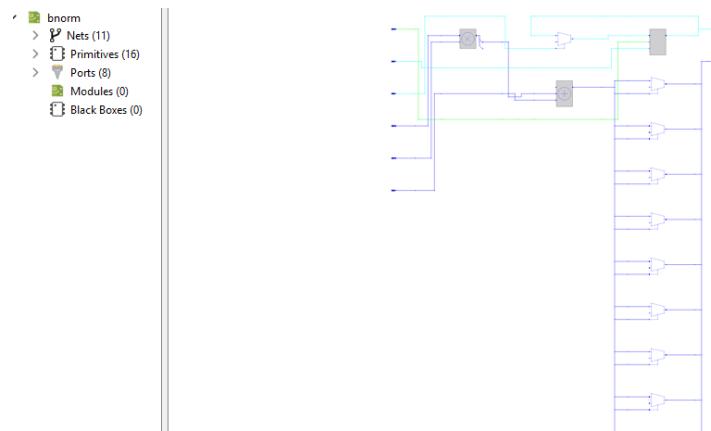
Viết code testbench và sử dụng chức năng monitor của Modelsim để kiểm tra các testcase còn lại.

```
VSIM 14> run
# At 0: data_in=0000 theta=100 phi=000 -> out=000 finish=0
# At 30000: data_in=0100 theta=100 phi=000 -> out=100 finish=0
# At 35000: data_in=0100 theta=100 phi=000 -> out=100 finish=1
#   ty transform passed
# At 41000: data_in=0100 theta=200 phi=000 -> out=200 finish=1
#   Scale only passed
# At 52000: data_in=0100 theta=100 phi=080 -> out=180 finish=1
#   Shift only passed
# At 63000: data_in=7fff theta=100 phi=000 -> out=000 finish=1
#   Max input passed
# At 74000: data_in=ff00 theta=100 phi=000 -> out=000 finish=1
#   input (ReLU) passed
# At 115000: data_in=ff00 theta=100 phi=000 -> out=000 finish=0
# At 125000: data_in=ff00 theta=100 phi=000 -> out=000 finish=1
# All tests completed
# ** Note: $finish : C:/Users/ADMIN/Desktop/HK242/DA1/cnnverilog/tb/tb_bnorm.sv(80)
#   Time: 225 ns Iteration: 0 Instance: /bnorm_tb
```

Hình 3.9: Testbench Bnorm

3.3.4 Synthesize

Sử dụng phần mềm Gowin FPGA Designer để chạy synthesis ta có kết quả Netlist như hình 3.10.



Hình 3.10: Kết quả netlist synthesis khối Bnorm

Ta có được báo cáo tài nguyên sử dụng như sau.

3.4 Khối Maxpool

3.4.1 Ý Tưởng Thiết Kế

So sánh lần lượt các giá trị 12-bit (Q4.8) không cần để ý đến dấu. Nếu có giá trị lớn hơn giá trị temp thì sẽ thay thế temp bằng giá trị đó. Nếu có tín hiệu *end* sẽ kết thúc quá



Bảng 6: Báo cáo sử dụng tài nguyên FPGA cho khối Bnorm

Tài nguyên	Sử dụng/Tổng (đơn vị)	Utilization
Logic	23 (11 LUT, 12 ALU) / 8640	<1.0%
Register	1 / 6693	<1.0%
• Register as Latch	0 / 6693	0.0%
• Register as FF	1 / 6693	<1.0%
BSRAM	0 / 26	0.0%
DSP		
• MULT18X18	1	

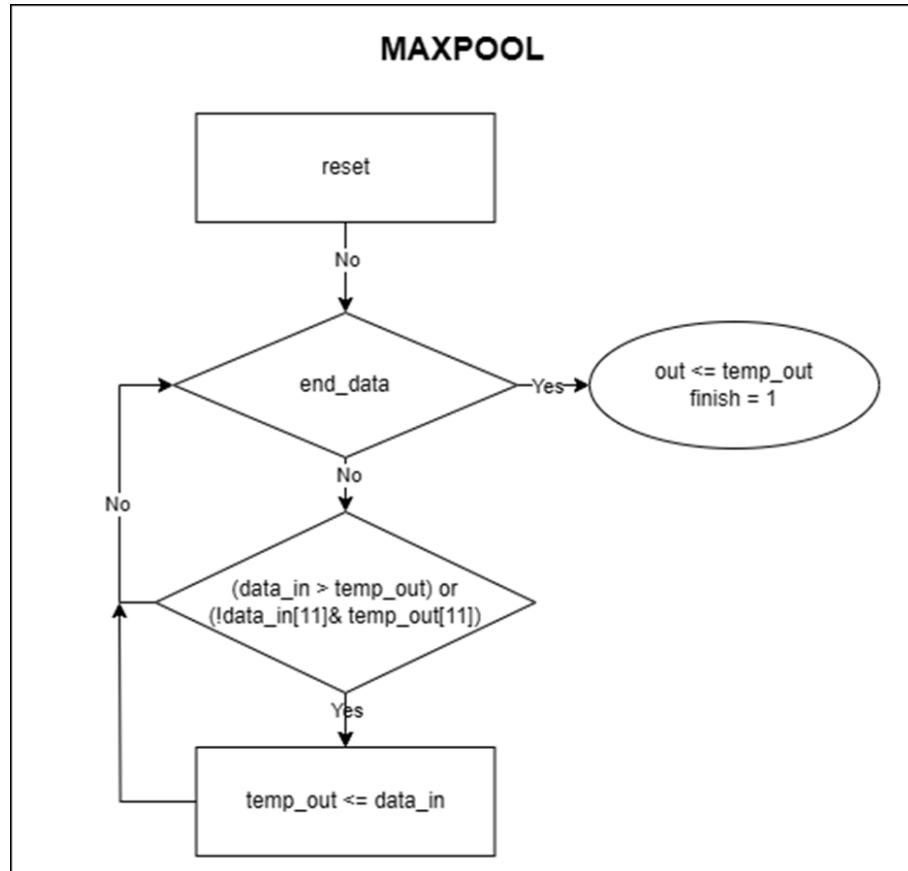
trình so sánh và bật cờ *finish*. Có thể sử dụng khối này dùng cho quá trình Max pooling hoặc Global pooling.

3.4.2 Thiết kế khối Maxpool

Bảng 7: Module MaxPool Signal Definitions

Signal Name	Direction	Width	Description
clk	Input	1	System clock
rst	Input	1	Active-high reset
data_in	Input	12	Input data (Q4.8 format)
end_data	Input	1	End of input data flag
finish	Output	1	Operation completion flag
out	Output	12	Max pooling result (Q4.8 format)

Thiết kế khối Maxpool theo flow sau bằng ngôn ngữ verilog.

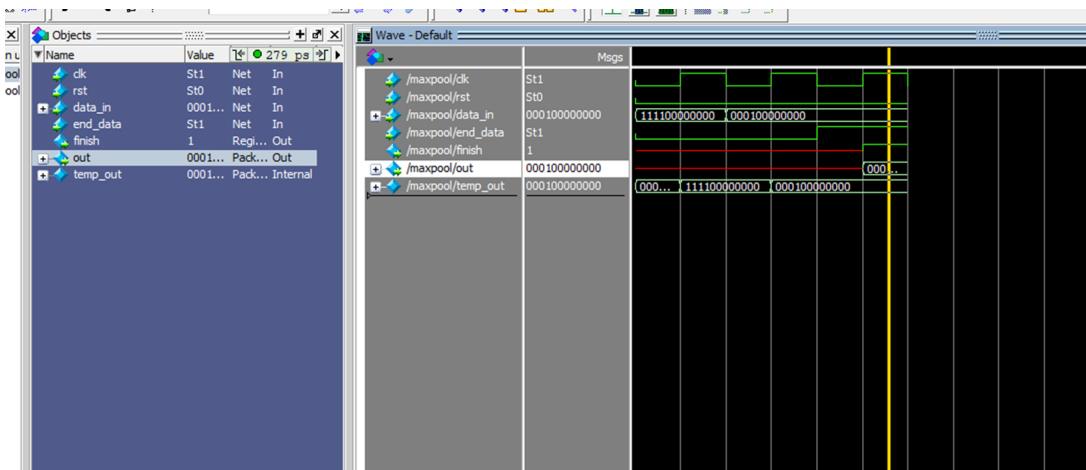


Hình 3.11: Maxpool flow chart

3.4.3 Kiến thử & Xác minh

Bảng 8: MaxPool Module Verification Plan

Test Category	Verification Method	Expected Result
Reset Verification		
Power-on reset	Assert reset, then release	All registers cleared, outputs zero
Mid-operation reset	Assert reset during window processing	Immediate output reset to zero
Basic Functionality		
Single 2x2 window	Input [1.0, 0.5, 1.5, 0.0] (Q4.8)	Output = 1.5 (max value)
Identical values	Input [0.8, 0.8, 0.8, 0.8]	Output = 0.8 (any duplicate accepted)
Boundary Cases		
Max Q4.8 value	Input [7.996, 0.0, 0.0, 0.0] (16'h7FF)	Output = 7.996
Timing Control		
end_data timing	Toggle end_data during streaming	finish asserts 1 cycle after last window
Data starvation	Delay between input windows	Correct output timing with gaps
Error Cases		
Incomplete window	Send 3 values then end_data	Output remains zero (no false trigger)

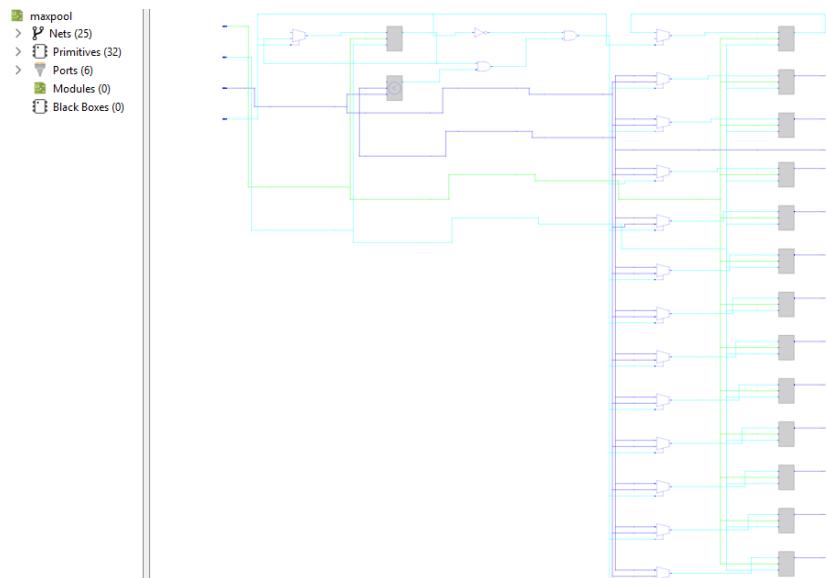


Hình 3.12: Dạng sóng ngoã ra cho testcase chức năng chính khối maxpool

Quan sát khi tín hiệu *finish*=1 kết quả ngõ ra bằng 1 (Q4.8) do ngõ vào lần lượt bằng -1 và 1 sau khi cờ *end_data* được bật sẽ bật cờ *finish* và trả kết quả ngõ ra bằng 1.
Viết code testbench và kiểm tra dạng sóng các testcase còn lại.

3.4.4 Synthesize

Sử dụng phần mềm Gowin FPGA Designer để chạy synthesis ta có kết quả Netlist như hình 3.13.



Hình 3.13: Kết quả netlist synthesis khối Maxpool

Ta có được báo cáo tài nguyên sử dụng và timing như sau.



Bảng 9: Báo cáo sử dụng tài nguyên FPGA cho khối Maxpool

Tài nguyên	Sử dụng/Tổng (đơn vị)	Utilization
Logic	14 (2 LUT, 12 ALU) / 8640	<1.0%
Register	14 / 6693	<1.0%
• Register as Latch	0 / 6693	0.0%
• Register as FF	14 / 6693	<1.0%
BSRAM	0 / 26	0.0%

Max Frequency Summary:

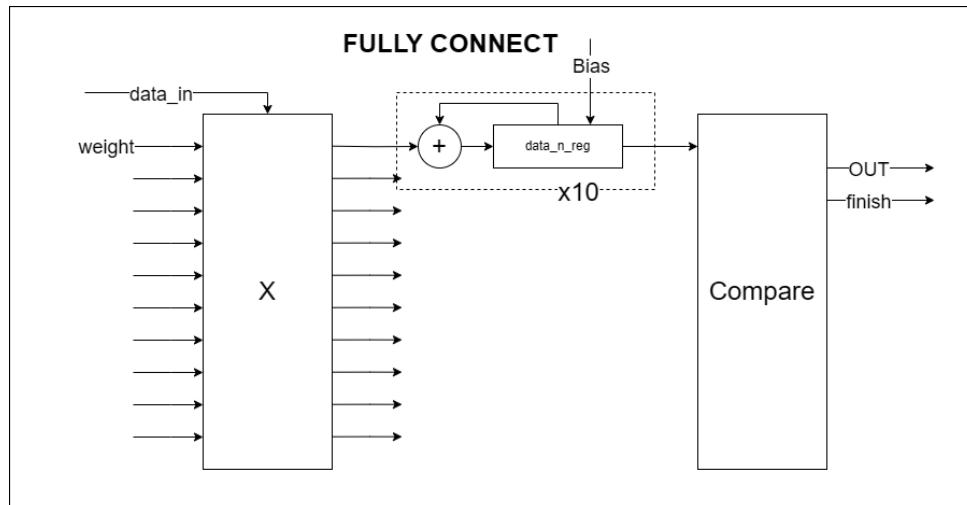
NO.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	clk	50.000(MHz)	177.883(MHz)	5	TOP

Hình 3.14: Báo cáo timing khối Maxpool

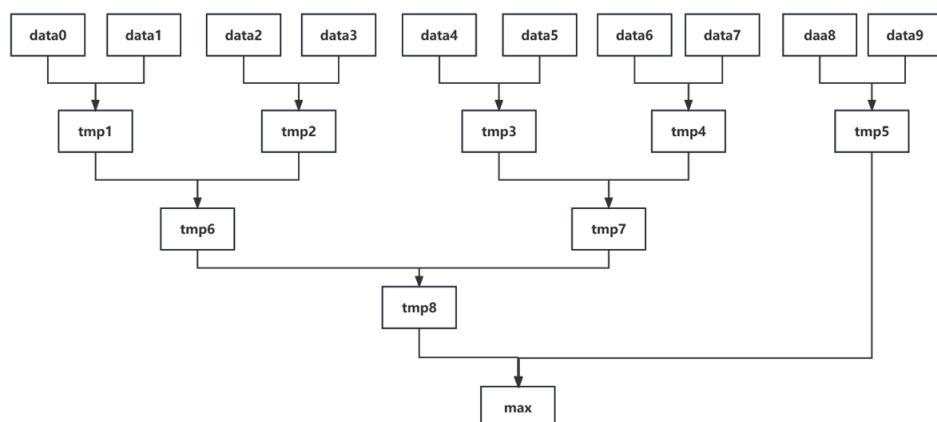
3.5 Khối Fully Connect

3.5.1 Ý Tưởng Thiết Kế

- Thiết kế bộ fully connected layer tối ưu cho bài toán phân lớp 10 lớp:
 - Dầu vào dạng fixed-point Q4.8 (12-bit)
 - Trọng số dạng Q4.8 (12-bit)
 - Dầu ra dạng one-hot 4-bit (10 lớp)
 - Hỗ trợ batch size lên đến 16 mẫu (counter 4-bit)
- Kết hợp 2 module chính:
 - Multiply-Accumulate (MAC)*: 10 bộ MAC song song cho 10 lớp đầu ra
 - Max Tournament*: Cây so sánh 5 tầng để tìm lớp có giá trị lớn nhất



Hình 3.15: Sơ đồ khối FullyConnect với pipeline 3 giai đoạn



Hình 3.16: Sơ đồ khối cây so sánh 5 tầng

- Cơ chế pipeline:
 - Stage 1 (Compute): Nhân-ma-trận và cộng bias từ ROM (1 chu kỳ)
 - Stage 2 (Vote): So sánh 10 giá trị dense qua cây tournament (3 chu kỳ)
- Triển khai phép toán signed MAC:

```

// Signed multiplication with sign extension
mul_res[i] = {12'b0, data_in} * {{12{w[i][11]}}, w[i]};
// Accumulation with 16-bit bias
dense_temp[i] = dense[i] + mul_res[i][23:8];
  
```

- Tối ưu hóa phần cứng:

- Sử dụng DSP blocks cho phép nhân 12-bit thay vì LUT
- Shared weight loading qua SIPO110 tiết kiệm 78% bus điều khiển

3.5.2 Thiết kế khối FullyConnect

Bảng 10: Module FullyConnect Signal Definitions

Signal Name	Direction	Width	Description
clk	Input	1	System clock
rst	Input	1	Active-high reset
data_in	Input	12	Input feature map (Q4.8)
w	Input	120	Weight value (Q4.8)
pos_data	Output	4	Position counter
load_weight	Output	1	Weight loading flag
cnn_out	Output	4	Classification result
finish	Output	1	Operation completion flag

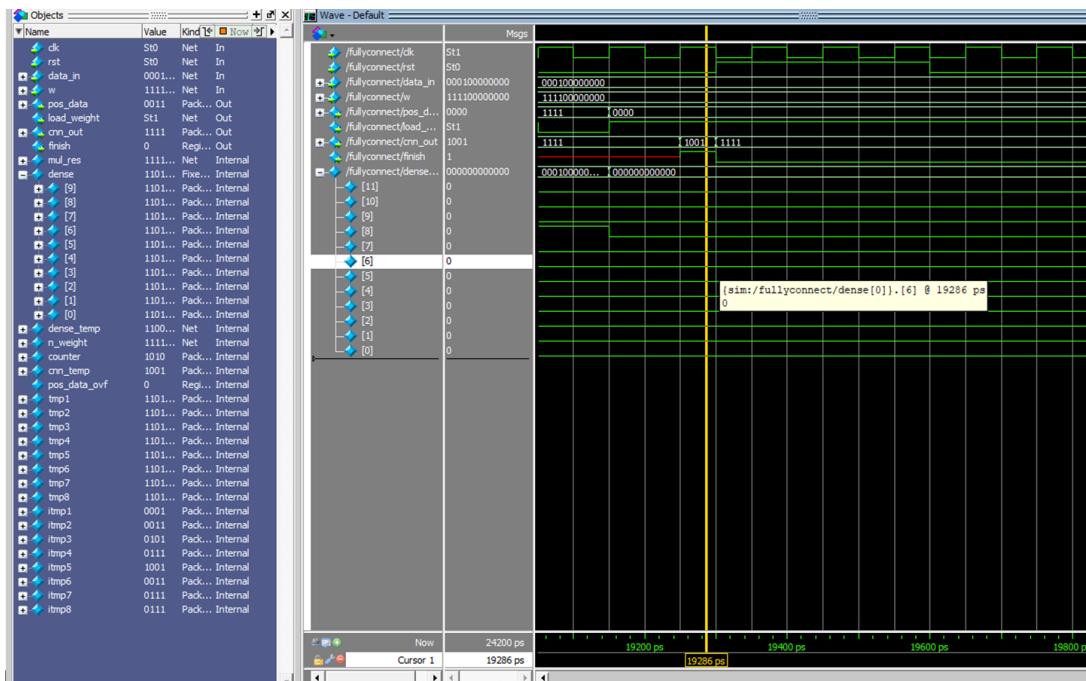
- Cơ chế hoạt động:
 - Giai đoạn tính toán:
 - * Thực hiện phép nhân có dấu Q4.8 × Q4.8
 - * Cộng dồn với bias pre-load từ bộ nhớ ROM
 - * Pipeline 3 giai đoạn: Nhân → Cộng → So sánh
 - Giai đoạn voting:
 - * So sánh 10 giá trị dense theo cơ chế tournament sort
 - * Xuất kết quả lớp có giá trị lớn nhất (4-bit one-hot)
- Kiến trúc Pipeline:
 - Stage 1: 10 bộ nhân song song (12-bit × 12-bit)
 - Stage 2: 10 bộ cộng có dấu 16-bit với bias
 - Stage 3: Cây so sánh 5 tầng để tìm max



3.5.3 Kiểm Thử & Xác Minh

Bảng 11: FullyConnect Verification Plan

Test Category	Test Case	Verification Method	Expected Result
Reset Verification	Power-on reset	Assert reset, then release	All registers cleared, cnn_out=15
Weight Loading	Serial weight shifting	Input known weight pattern	Correct output after 12 cycles
Basic Inference	All weights=1.0	Input fixed value 1.0	Correct dense[i] = bias[i] + 1.0
Boundary Cases	Max Q4.8 input (7.996)	Input 12'h7FF with random weights	Proper saturated accumulation
Negative Values	Mixed sign weights	Alternate positive/negative weights	Correct signed arithmetic
Timing Control	Finish signal timing	Monitor pos_data counter	finish=1 when pos_data overflows
Comparison Logic	All dense equal	Set identical dense values	Any class selected (one-hot)
Bias Loading	ROM initialization	Check initial dense values	Match predefined biasrom.mem



Hình 3.17: Dạng sóng testbench với tất cả giá trị dense giống nhau

Với tất cả giá trị dense giống nhau khi đưa vào cây so sánh theo thuật toán cây so sánh nếu tất cả các giá trị so sánh bằng nhau kết quả trả về sẽ là 9.

Viết code testbench và sử dụng chức năng monitor của Modelsim để kiểm tra các testcase còn lại.

```

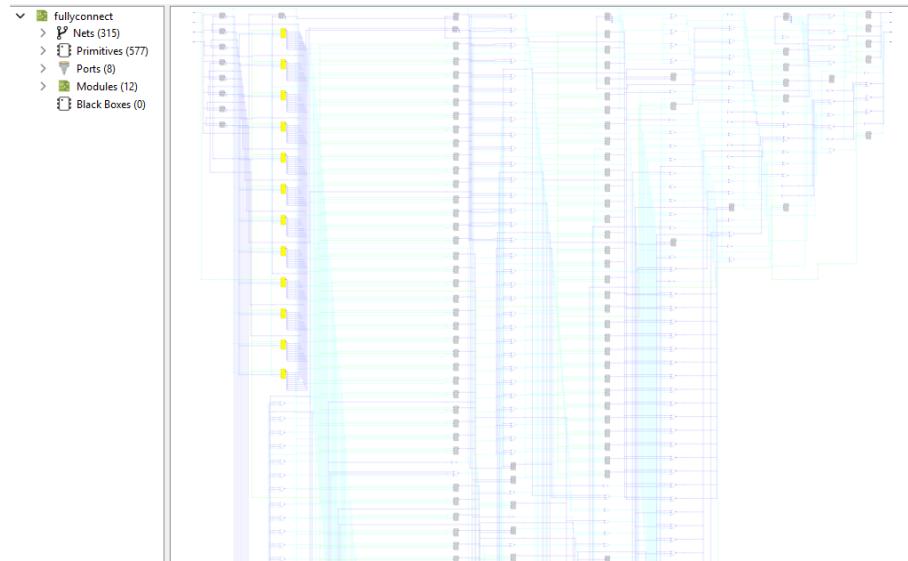
# [TEST] 1. Reset Verification
#
# [TEST] 2. Weight Loading
#
# [TEST] 3. Basic Inference
#
# [TEST] 4. Boundary Cases
#
# [TEST] 5. Negative Values
#
# [TEST] 6. Timing Control
# t=2085000: Classification result = 1001
# t=2095000: Classification result = 1001
# t=2105000: Classification result = 1001
# t=2115000: Classification result = 1001
# t=2125000: Classification result = 1001
#
# [TEST] 7. Comparison Logic
#
# [TEST SUMMARY]
# PASSED: All testcases completed successfully
# ** Note: $finish : C:/Users/ADMIN/Desktop/HK242/DA1/cnnverilog/tb/tb_fc.sv(118)
#   Time: 2155 ns Iteration: 0 Instance: /tb_FullyConnect

```

Hình 3.18: Testbench Fully Connect

3.5.4 Synthesize

Sử dụng phần mềm Gowin FPGA Designer để chạy synthesis ta có kết quả Netlist như hình 3.19.



Hình 3.19: Kết quả netlist synthesis khối FullyConnect

Ta có được báo cáo tài nguyên sử dụng và timing như sau.

Bảng 12: Báo cáo tài nguyên FPGA cho FullyConnect

Tài nguyên	Sử dụng/Tổng (đơn vị)	Utilization
Logic	550 (310 LUT, 240 ALU) / 8640	7.0%
Register	182 / 6693	3.0%
• FF	0 / 6693	0%
• Latch	182 / 6693	3.0%
DSP		
• MULT18X18	10	

Max Frequency Summary:

NO.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	clk	50.000(MHz)	142.090(MHz)	4	TOP

Hình 3.20: Báo cáo timing khối Fully Connect

3.6 Khối BSRam

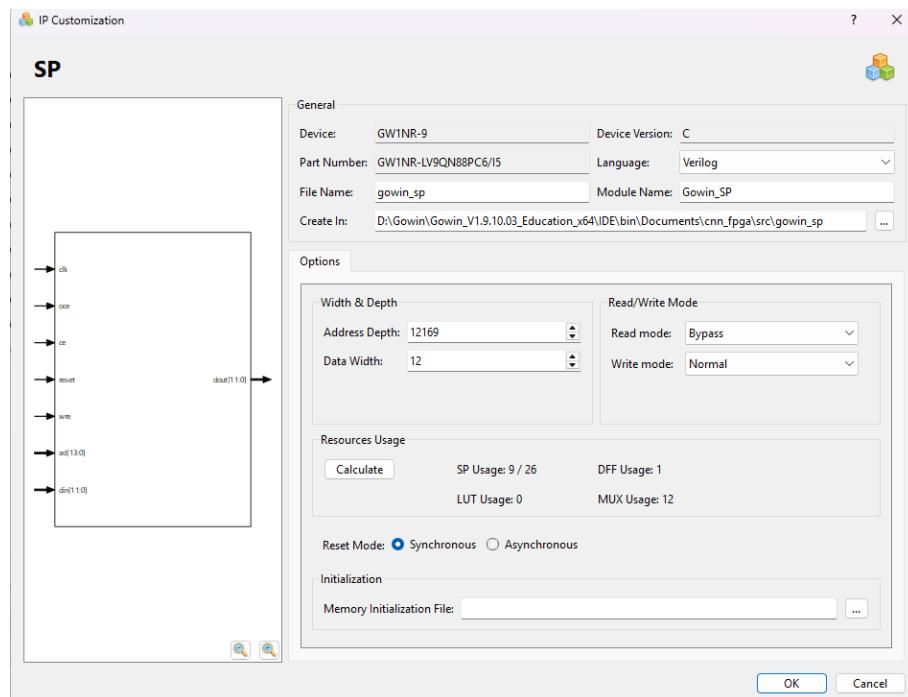
Ta sẽ phân vùng Ram như sau để lưu dữ liệu từng lớp.



Bảng 13: Bảng memory map hệ thống CNN (14-bit địa chỉ, 12-bit dữ liệu)

Vùng	Địa chỉ bắt đầu	Kích thước	Mô tả
PADDING	14'd0 (0x0000)	1 words (0x0001)	Chứa zero dùng để làm padding cho CONV
CONV1	14'd1 (0x0001)	784 words (0x0310)	Chứa dữ liệu 1 ảnh 28x28 đầu vào CONV1
CONV2	14'd785 (0x0311)	3136 words (0x0C40)	Chứa dữ liệu 4 ảnh 28x28 đầu vào CONV2
MPOOL1	14'd3921 (0x0F51)	3136 words (0x0C40)	Chứa dữ liệu 4 ảnh 28x28 đầu vào MPOOL1
CONV3	14'd7057 (0x1B91)	784 words (0x0310)	Chứa dữ liệu 4 ảnh 14x14 đầu vào CONV3
CONV4	14'd7841 (0x1EA1)	1568 words (0x0620)	Chứa dữ liệu 8 ảnh 14x14 đầu vào CONV4
MPOOL2	14'd9409 (0x24C1)	1568 words (0x0620)	Chứa dữ liệu 8 ảnh 14x14 đầu vào MPOOL2
CONV5	14'd10977 (0x2AE1)	392 words (0x0188)	Chứa dữ liệu 8 ảnh 7x7 đầu vào CONV5
GMPOOL	14'd11369 (0x2C69)	784 words (0x0310)	Chứa dữ liệu 16 ảnh 7x7 đầu vào GMPOOL
DENSE	14'd12153 (0x2F79)	16 words (0x0010)	Chứa dữ liệu 16 ảnh 1x1 đầu vào Fully Connect
TỔNG		12169 words	

Như vậy để có thể lưu trữ được tất cả dữ liệu cần thiết cho nhận diện số viết tay ta cần dùng bộ nhớ ram có kích thước khoảng 18kByte.



Hình 3.21: BSRam ip core Gowin

Để đơn giản việc thiết kế ta có thể re-use một số tính năng IP Core của tool Gowin để tạo khối BSRAM. Kit Tang Nano 9K hỗ trợ đến 26 khối BSRAM với kích thước mỗi khối là 16kBit. Theo hình ta thấy cần sử dụng 9 khối BSRAM tương ứng với 18kByte như tính toán bên trên.

3.7 Các khối ROM chứa dữ liệu trọng số

Triển khai mô hình AI trên phần cứng đòi hỏi chuyển đổi trọng số từ định dạng dấu phẩy động (float32) sang định dạng cố định (fixed-point) Q4.8 và nhị phân của lớp convolution. File .mem trong Verilog chứa dữ liệu nhị phân/hex, được sử dụng để khởi tạo bộ nhớ ROM/RAM.



Layer (type)	Output Shape	Param #
conv1 (BinaryConv2D)	(None, 28, 28, 4)	36
batchnormalize1 (BatchNormalization)	(None, 28, 28, 4)	16
activation1 (ReLU)	(None, 28, 28, 4)	0
conv2 (BinaryConv2D)	(None, 28, 28, 4)	144
batchnormalize2 (BatchNormalization)	(None, 28, 28, 4)	16
activation2 (ReLU)	(None, 28, 28, 4)	0
pool1 (MaxPooling2D)	(None, 14, 14, 4)	0
conv3 (BinaryConv2D)	(None, 14, 14, 8)	288
batchnormalize3 (BatchNormalization)	(None, 14, 14, 8)	32
activation3 (ReLU)	(None, 14, 14, 8)	0
conv4 (BinaryConv2D)	(None, 14, 14, 8)	576
batchnormalize4 (BatchNormalization)	(None, 14, 14, 8)	32
activation4 (ReLU)	(None, 14, 14, 8)	0
pool2 (MaxPooling2D)	(None, 7, 7, 8)	0
conv5 (BinaryConv2D)	(None, 7, 7, 16)	1,152
batchnormalize5 (BatchNormalization)	(None, 7, 7, 16)	64
activation5 (ReLU)	(None, 7, 7, 16)	0
pool3 (GlobalMaxPooling2D)	(None, 16)	0
dense1 (Dense)	(None, 10)	170

Total params: 2,526 (9.87 KB)

Trainable params: 2,446 (9.55 KB)

Non-trainable params: 80 (320.00 B)

Hình 3.22: Báo cáo trọng số từng lớp CNN

Các bước thực hiện

- B1: Trích xuất Trọng số từ CNN bằng cách sử dụng API của TensorFlow (`model.layers[i].get_weights()`).
- B2: Chuyển đổi sang Nhị phân
 - Làm phẳng (flatten) filter 3x3 và chuyển đổi trọng số của lớp conv về dạng nhị phân ta được một chuỗi 9-bit tương ứng với 9 trọng số của filter 3x3.
 - Tính toán trước hệ số θ và ϕ mà ta thiết kế theo 4 hệ số trọng số $\alpha, \beta, \gamma, \mu$ của lớp Batch Normalize và lượng tử về định dạng Q4.8.
 - Lượng tử trọng số và bias của lớp fully connect về dạng Q4.8.

- B3: Tạo File .mem cho Verilog
- B4: Khởi tạo bộ nhớ trong Verilog với từng khối ROM.

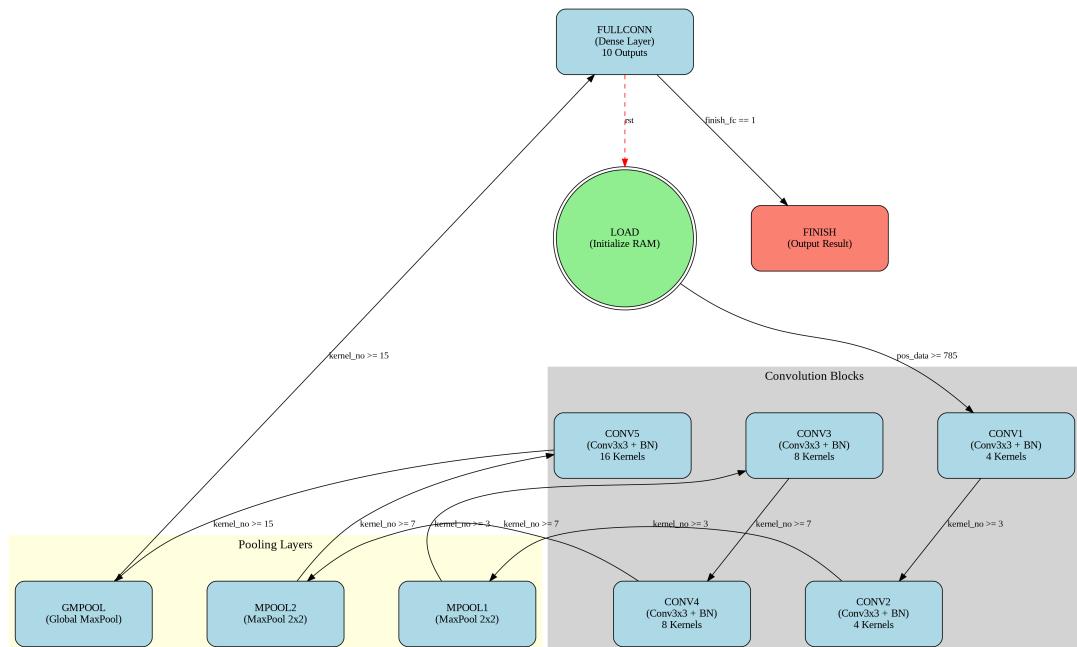
Kích thước Rom

- Tổng các param của lớp conv là: $36 + 144 + 288 + 576 + 1152 = 2196$. Vậy ta cần 1 ROM chứa 244×9 -bit để lưu trọng số lớp conv.
- Tổng các param của lớp batch normalize là: $16 + 16 + 32 + 32 + 64 = 160$ do ta đã giảm một nửa số param của lớp này nên còn 80 param. Vậy ta cần 1 ROM chứa 80×12 -bit để lưu trọng số lớp Batch normalize.
- Tổng số param của lớp fully connect là 170 bỏ qua 10 trọng số bias được load sẵn ta cần 1 ROM chứa 160×12 -bit để lưu trọng số lớp này.

3.8 Khối điều khiển (FSM)

3.8.1 Tổng quan

Dựa theo flow tính toán và sơ đồ của mạng CNN ta lập được một sơ đồ máy trạng thái như sau (Gộp 3 lớp BinaryConv + BatchNormalize + ReLU thành 1 trạng thái CONV):

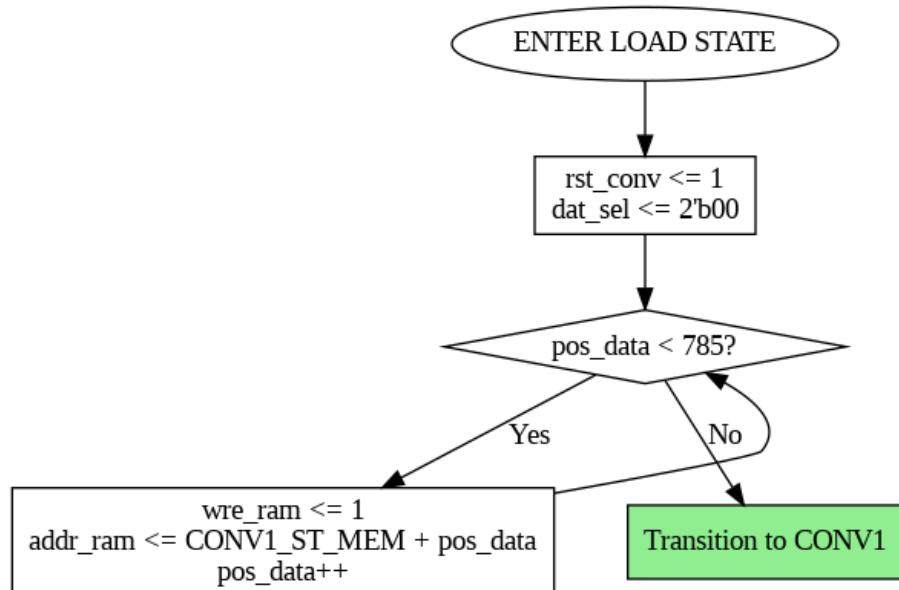


Hình 3.23: Sơ đồ trạng thái của khối điều khiển

Ở các trạng thái khác nhau sẽ có tín hiệu điều khiển cho datapath khác nhau.

3.8.2 Giai đoạn LOAD

Ở giai đoạn này sẽ xuất các tín hiệu điều khiển để lưu ảnh mới 8-bit qua khối chuẩn hóa chuyển thành kiểu dữ liệu Q4.8 và lưu vào Bsram.

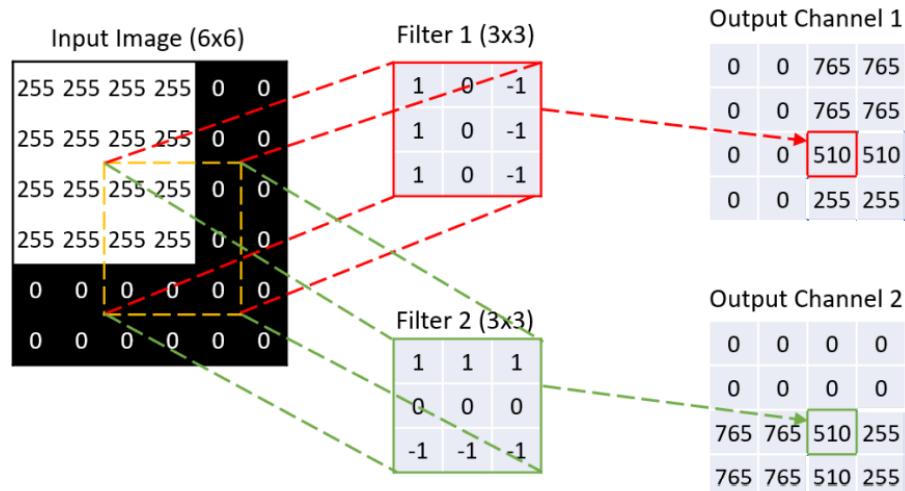


Hình 3.24: Flow chart ở giai đoạn LOAD

Kết thúc giai đoạn này ta sẽ có được 784 pixel dữ liệu hình ảnh đã được chuẩn hóa lưu ở trong Ram.

3.8.3 Giai đoạn CONV1

Ở giai đoạn ta cần điều khiển địa chỉ của ROM chứa trọng số của lớp Conv và BatchNormalize theo từng Kernel. Ngoài ra còn phải tính toán địa chỉ điểm ảnh của ảnh vừa được load vào trong Ram. Ảnh được load vào trong Ram là ảnh đã được làm phẳng do đó ta cần 2 thành ghi là x_temp_pos và y_temp_pos để thực hiện quá trình quét ảnh 2D của lớp Conv



Hình 3.25: Quá trình quét ảnh 2D

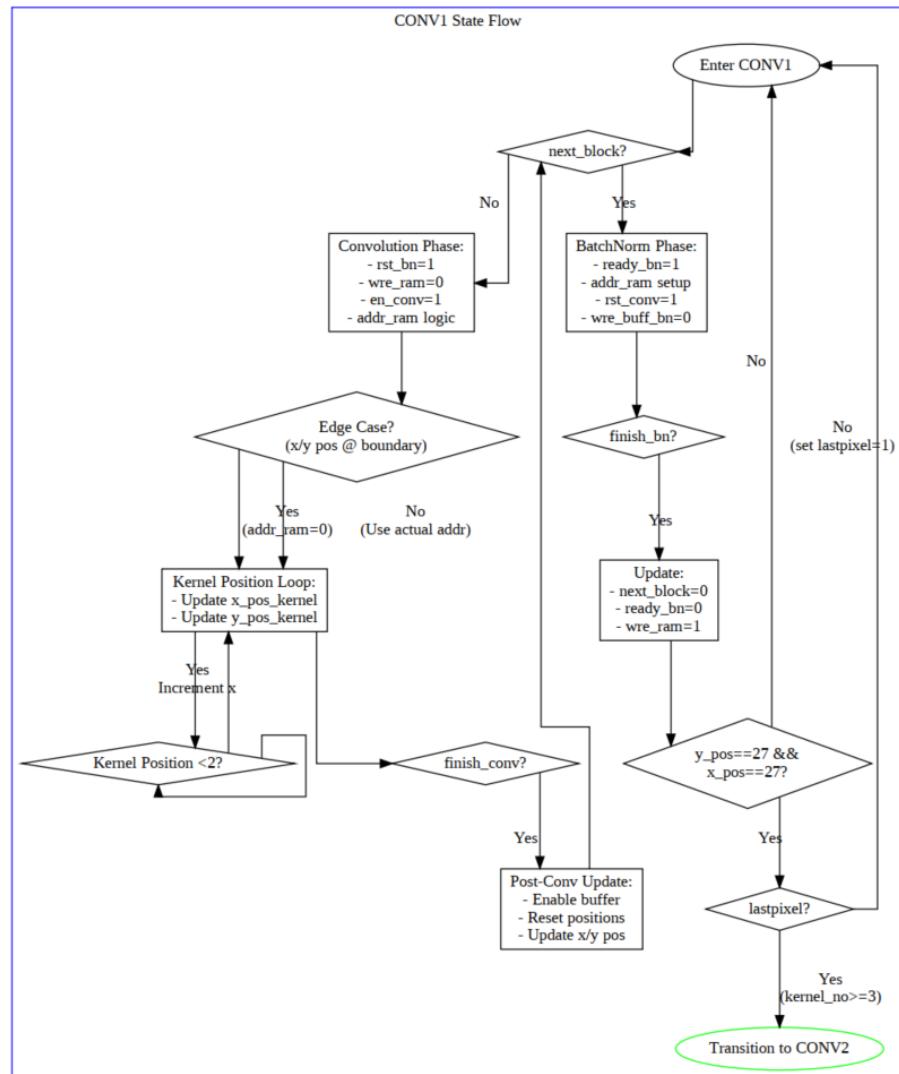
Ngoài 2 thanh ghi lưu tọa độ của điểm ảnh ta cần thêm 2 thanh ghi để lưu tọa độ 8 điểm ảnh lân cận. Vì như hình 3.25 tất cả các Kernel (Filter) ta sử dụng trong mô hình này đều là 3×3 mỗi lần trượt của Filter sẽ lấy 9 giá trị điểm ảnh nhân tích chập với trọng số để tạo ra được điểm ảnh mới ở ngõ ra.

Và theo như mô hình ta chọn lúc đầu kích thước ảnh sau khi qua lớp Conv sẽ giữ nguyên (VD: 28x28) nhưng vì Kernel có size là 3x3 theo thực tế kích thước ảnh sẽ giảm 2 pixel (VD: 26x26). Vì thế ta sử dụng kỹ thuật Padding là các viền ngoài của ảnh sẽ vào sẽ cho bằng 0. Và theo như giải thuật trên ta sẽ thiết kế sao cho dựa vào 4 giá trị tọa độ điểm ảnh để phát hiện ra trường hợp điểm ảnh Padding.

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 0 & 3 & 3 & 4 & 4 & 7 & 0 & 0 \\ \hline
 0 & 9 & 7 & 6 & 5 & 8 & 2 & 0 \\ \hline
 0 & 6 & 5 & 5 & 6 & 9 & 2 & 0 \\ \hline
 0 & 7 & 1 & 3 & 2 & 7 & 8 & 0 \\ \hline
 0 & 0 & 3 & 7 & 1 & 8 & 3 & 0 \\ \hline
 0 & 4 & 0 & 4 & 3 & 2 & 2 & 0 \\ \hline
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array}
 \quad *
 \quad
 \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array}
 \quad =
 \quad
 \begin{array}{|c|c|c|c|c|c|} \hline
 -10 & -13 & 1 & & & \\ \hline
 -9 & 3 & 0 & & & \\ \hline
 & & & & & \\ \hline
 \end{array}
 \quad
 6 \times 6$$

Hình 3.26: Padding cho lớp convolution

Từ các ý trên ta có được các tín hiệu điều khiển như sau:

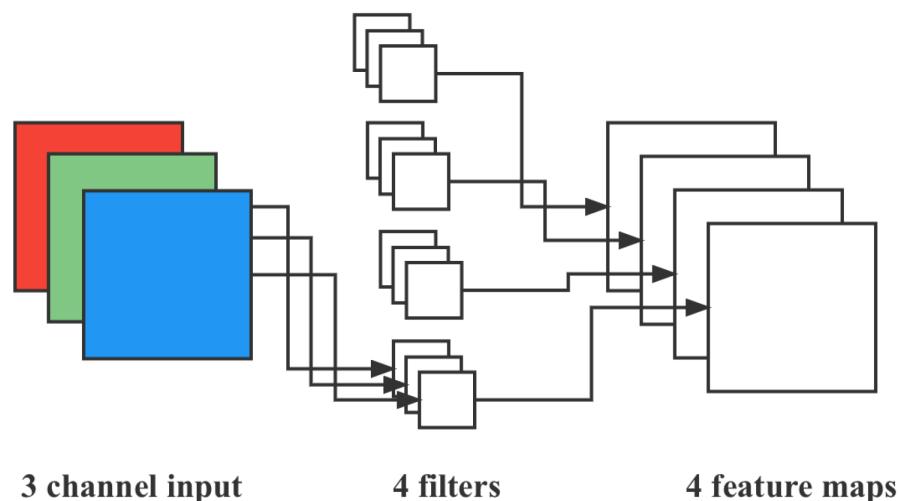


Hình 3.27: Flowchart CONV1

Kết thúc giai đoạn CONV1 ta sẽ có được 4 kênh ảnh 28x28 mới lưu ở trong Ram và tiếp tục đến giai đoạn CONV2

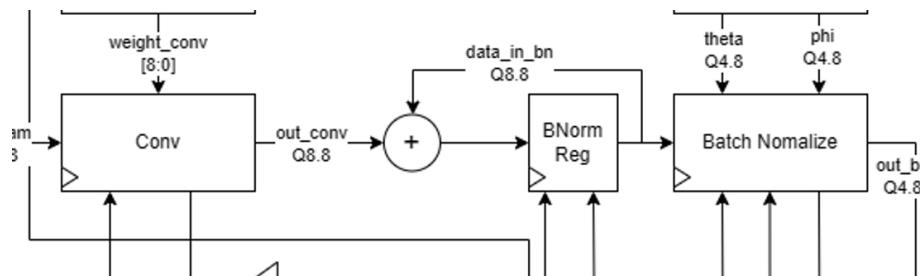
3.8.4 Giai đoạn CONV2

Ở giai đoạn CONV2 sẽ tương tự giai đoạn đầu tiên, tuy nhiên số lượng ảnh kenh ngõ vào đã tăng lên thành 4 kênh. Như vậy ở mỗi Kernel ứng với mỗi kênh ảnh sẽ có mỗi Filter khác nhau (Window).



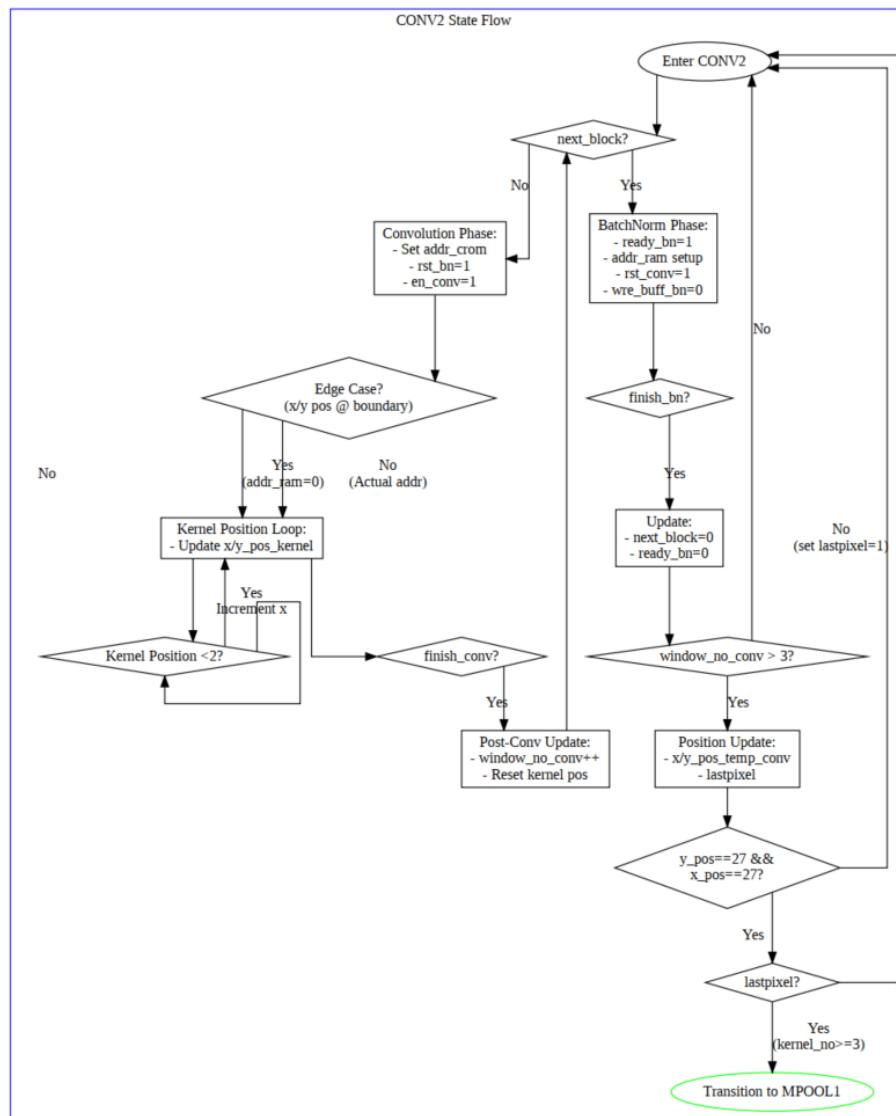
Hình 3.28: *Tích chập với ngõ vào nhiều kênh*

Vì thế tại mỗi điểm ảnh của mỗi kênh ngõ ra CONV2 sẽ là tổng tích chập của 4 kênh ảnh ngõ vào và 4 Window nên trong sơ đồ thiết kế phần cứng ta mới có thêm thanh ghi đếm Bnorm Reg trước khói BatchNormalize và bộ cộng để thực hiện điều này. Lúc này ở giai đoạn CONV2 ta cần phối hợp điều khiển thêm 2 tín hiệu là *rst_buf_bn* và *wre_buf_bn* để reset và cho phép ghi thanh ghi đếm này.



Hình 3.29: *Thanh ghi đếm Bnorm*

Ta lập được sơ đồ điều khiển các tín hiệu như sau:

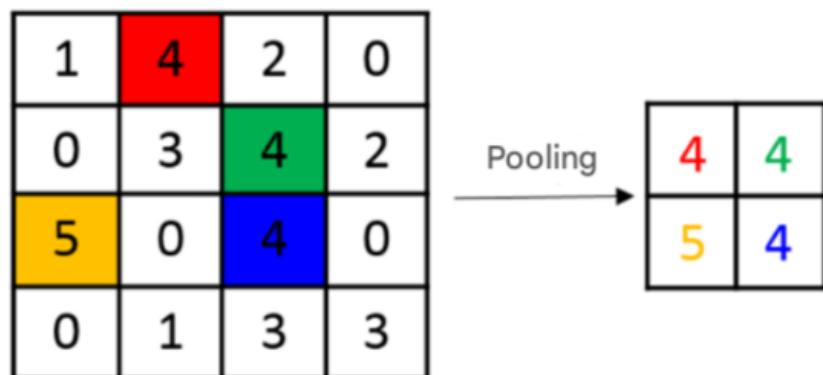


Hình 3.30: Flowchart CONV2

Kết thúc giai đoạn CONV2 ta sẽ có được 4 kênh ảnh 28x28 mới và tiếp tục đến giai đoạn MAXPOOL1

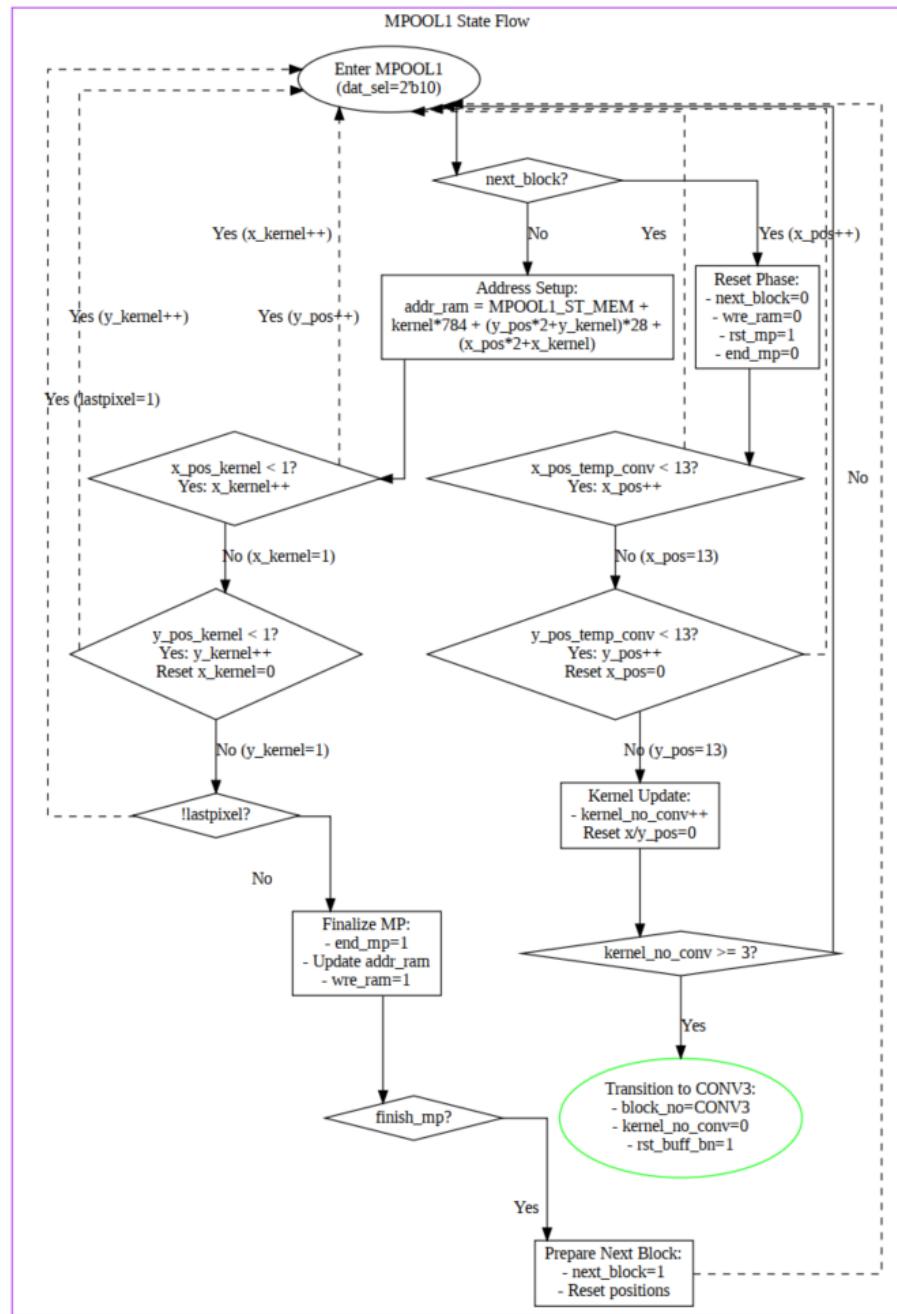
3.8.5 Giai đoạn MAXPOOL1

Với giai đoạn max pooling 2D này ta sẽ so sánh 4 giá trị điểm ảnh trong vùng không gian 2x2 trên ảnh 2 chiều và chọn ra điểm ảnh có giá trị lớn nhất. Với kỹ thuật này sẽ giảm nửa chiều dữ liệu ngõ vào cho lớp sau.



Hình 3.31: Maxpooling 2D

Ta lập được sơ đồ điều khiển các tín hiệu cho giai đoạn max pooling như sau:



Hình 3.32: Flowchart MAXPOOL1

Kết thúc giai đoạn MAXPOOL1 ta sẽ có được 4 kênh ảnh 14x14 mới lưu ở trong Ram và tiếp tục đến giai đoạn CONV3

3.8.6 Giai đoạn CONV3

Giai đoạn này gần giống với CONV2 với số Kernel là 8 và số Window là 4. Và dữ liệu hình ảnh lúc này là 14x14 nên ta cần giảm số vòng lặp và thông số vị trí vùng Padding. Kết thúc giai đoạn CONV3 ta sẽ có được 8 kênh ảnh 14x14 mới lưu ở trong Ram và tiếp

tục đến giai đoạn CONV4.

3.8.7 Giai đoạn CONV4

Giai đoạn này giống với giai đoạn CONV3 với số lượng Window lúc này là 8.

Kết thúc giai đoạn CONV4 ta sẽ có được 8 kênh ảnh 14×14 mới lưu ở trong Ram và tiếp tục đến giai đoạn MAXPOOL2.

3.8.8 Giai đoạn MAXPOOL2

Tương tự giai đoạn MAXPOOL1 lúc này ta sẽ giảm một nửa dữ liệu. Lúc này dữ liệu ảnh cho giai CONV5 sau là 8 kênh ảnh 7×7 .

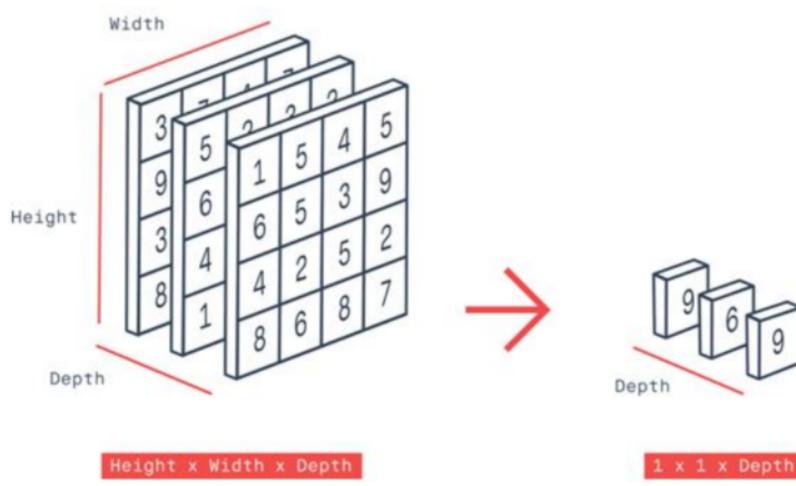
3.8.9 Giai đoạn CONV5

Giai đoạn này giống với các giai đoạn CONV trên với số lượng Kernel là 16, Window là 8 tương ứng với 8 kênh ảnh 7×7 ở ngõ vào.

Kết thúc giai đoạn CONV5 ta sẽ có được 16 kênh ảnh 7×7 mới lưu ở trong Ram và tiếp tục đến giai đoạn GMPOOL.

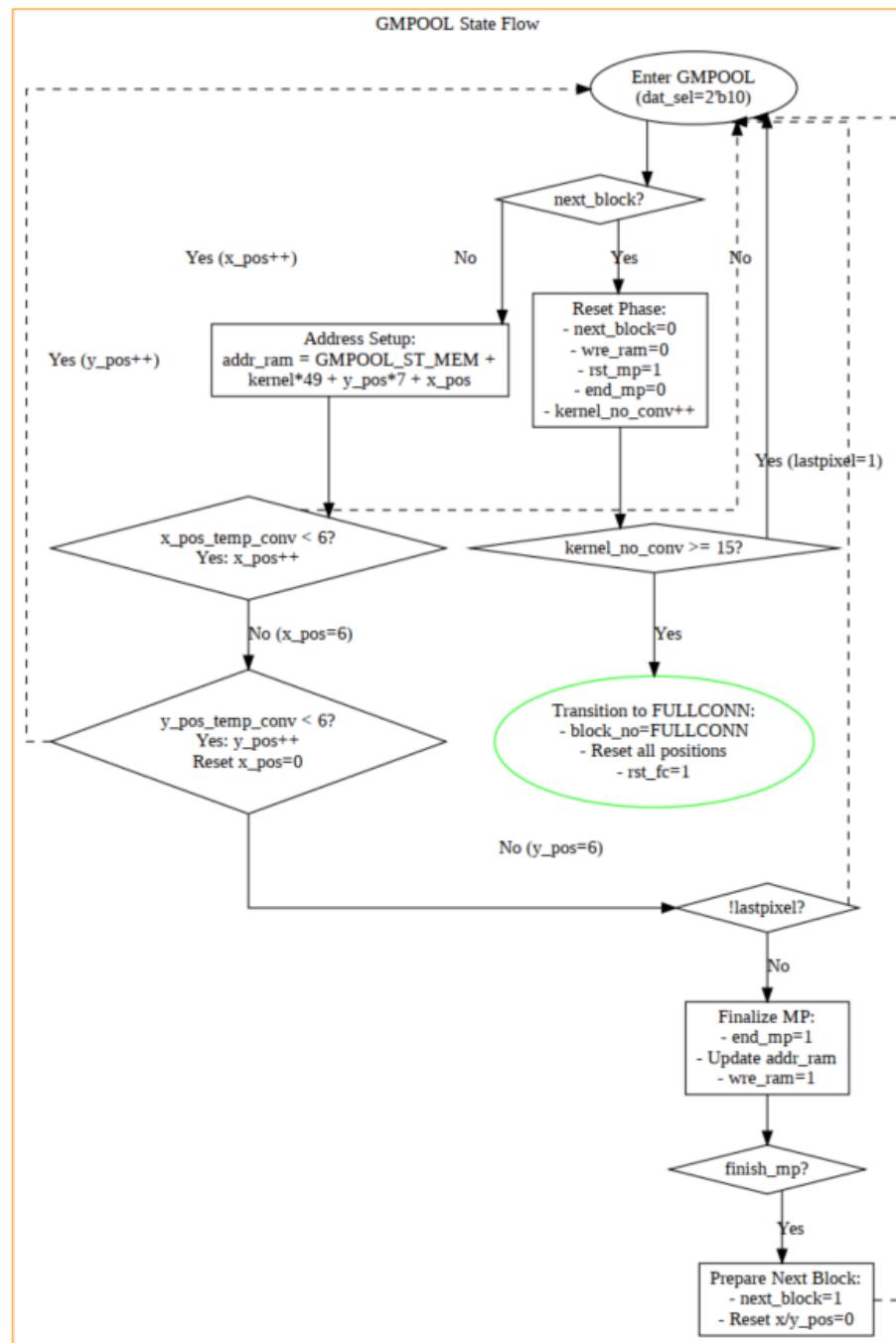
3.8.10 Giai đoạn GMPOOL

Như đã nói ở các phần trước nhằm để giảm số lượng trọng số và tính toán ở lớp Fully Connect ta sử dụng kỹ thuật giảm chiều dữ liệu này với 16 kênh ảnh 7×7 từ giai đoạn trước đó.



Hình 3.33: Kỹ thuật global max pool

Diều khiển các tín hiệu tương tự Max Pooling ta có được flow chart sau:

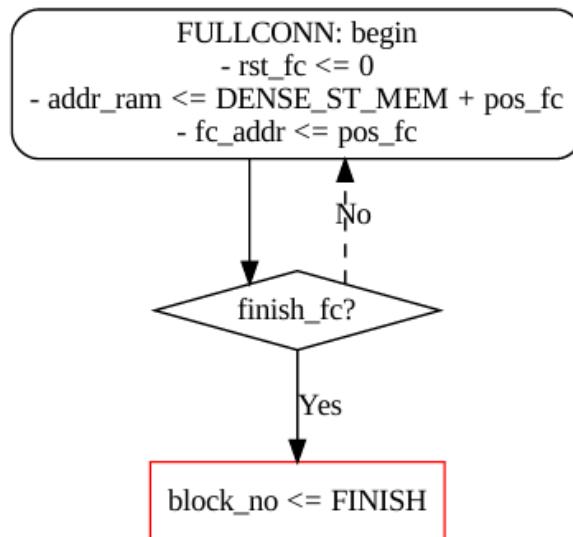


Hình 3.34: Flowchart GlobalMaxPool

Kết thúc giai đoạn GMPOOL ta sẽ có được 16 kênh ảnh 1x1 mới lưu ở trong Ram và tiếp tục đến giai đoạn FULLYCONN.

3.8.11 Giai đoạn FULLYCONN

Với ngõ vào là 16 kênh tương ứng ta có 16 Window, ngõ ra 10 kênh ứng với 10 Kernel. Sử dụng các thanh ghi để làm bộ đếm để lấy đúng dữ liệu trọng số ứng với Kernel và Window từ ROM và đưa vào khối FullyConnect.



Hình 3.35: Flowchart FULLYCONN

Kết thúc quá trình load dữ liệu khối fully connect sẽ tự tính toán và so sánh để đưa ra kết quả dự đoán số viết tay. Máy trạng thái sẽ chuyển qua trạng thái FINISH, kích hoạt cờ finish, để thực hiện chu trình dự đoán tiếp theo chỉ cần đưa tín hiệu reset cho máy trạng thái.

3.9 CNN Top

3.9.1 Thiết kế khối CNN Top

Dựa vào sơ đồ thiết kế phần cứng (Hình 3.1) ta kết nối khối điều khiển và datapath với nhau để hoàn chỉnh module nhận diện số viết tay.

Bảng 14: CNN Module Signal Definitions

Signal Name	Direction	Width	Description
clk	Input	1	System clock
rst	Input	1	Active-high reset
en	Input	1	Module enable control
data_in	Input	8	Input pixel data (unsigned byte)
pos_data	Output	10	Position counter/address bus
finish	Output	1	Operation completion flag
cnn_out	Output	4	Classification result (0-9)



3.9.2 Synthesize

Sử dụng tool Gowin để synthesis ta có báo cáo sử dụng tài nguyên như bảng 15.

Bảng 15: Resource Usage Summary

Resource	Usage
I/O Port	26
I/O Buf	25
• IBUF	10
• OBUF	15
Register	410
• DFF	16
• DFFE	34
• DFFRE	1
• DFFPE	103
• DFFC	6
• DFFCE	250
LUT	1353
• LUT2	289
• LUT3	352
• LUT4	712
ALU	1039
• ALU	1039
SSRAM	3
• RAM16S4	3
INV	17
• INV	17
DSP	-
• MULT18X18	10
• MULTADDALU18X18	5
BSRAM	15
• SP	12
• pROM	2
• pROMX9	1

Ta thấy thiết kế sử dụng rất ít tài nguyên với 2427 LUT, 410 thanh ghi flipflop và 15x16kbit bộ nhớ Ram.



Bảng 16: Resource Utilization Summary

Resource	Used/Total (units)	Utilization
Logic	2427 (1370 LUT, 1039 ALU, 3 RAM16) / 8640	29.0%
Register	410 / 6693	7.0%
• Register as Latch	0 / 6693	0.0%
• Register as FF	410 / 6693	7.0%
BSRAM	15 / 26	58.0%

3.9.3 Verification Plan

Bảng 17: CNN Module Verification Plan

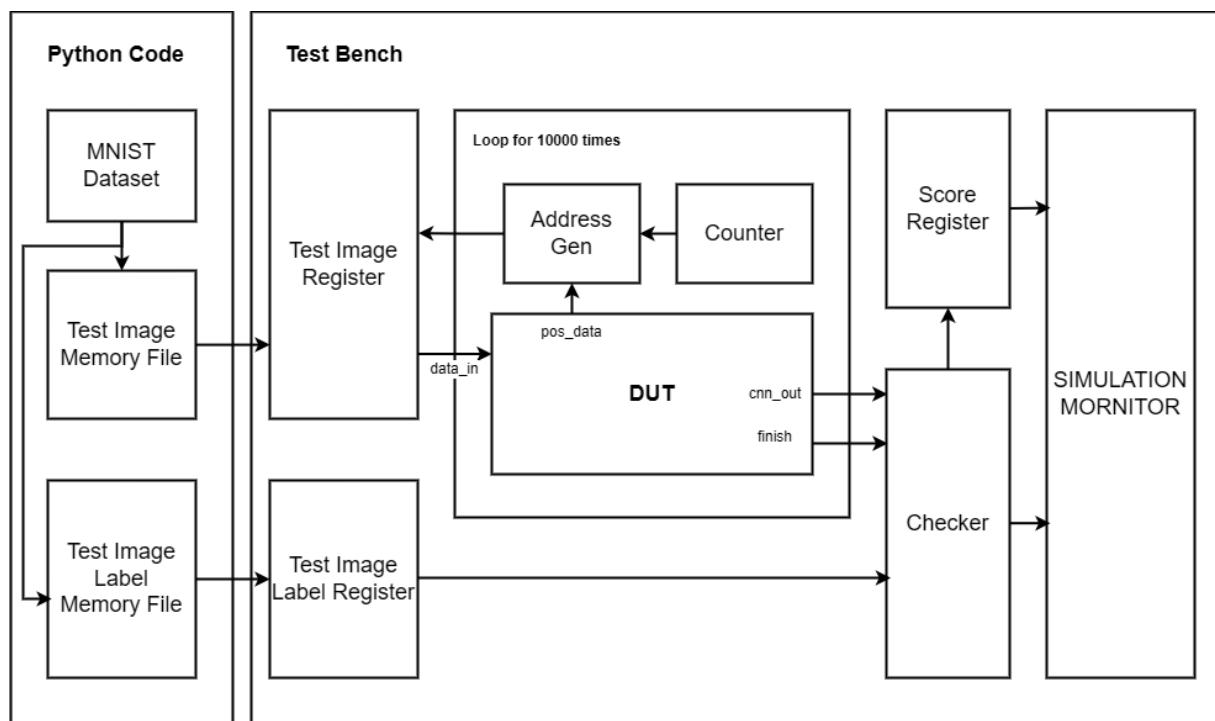
Test Category	Test Case	Verification Method	Expected Result
Reset Verification	Power-on reset	Assert reset for 10 cycles	All states reset to LOAD, pos_data=0, finish=0
Convolution Layers	CONV1 edge pixels	Input boundary coordinates (x=0, y=0)	Zero-padded address (addr_ram=0) selected
BatchNorm Integration	Post-CONV1 processing	Monitor bn_addr and wre_buff_bn	Correct BN parameters applied to conv output
Max Pooling	MPOOL1 downsampling	Input 4x4 pattern with max=255	Output 2x2 max values at correct addresses
State Transitions	CONV1→CONV2 transition	Trigger finish_bn signal	block_no increments to CONV2 within 1 cycle
Memory Addressing	CONV3 weight fetching	Check addr_crom during computation	Correct CROM offset (20 + kernel_no*4 + window_no)
Global Pooling	GMPOOL operation	Feed 7x7 input with single peak	Maximum value stored at DENSE_ST_MEM
Full Connection	Final classification	Load precomputed GMPOOL outputs	cnn_out matches MNIST label (one-hot)
Error Recovery	Mid-process reset	Assert rst during CONV2 operation	Immediate return to LOAD state
MNIST Accuracy	10k test images	Compare cnn_out with labels	98% classification accuracy

Từ kế hoạch kiểm thử trên ta sẽ kiểm tra thiết kế ở phần sau.

4 KIỂM TRA THIẾT KẾ

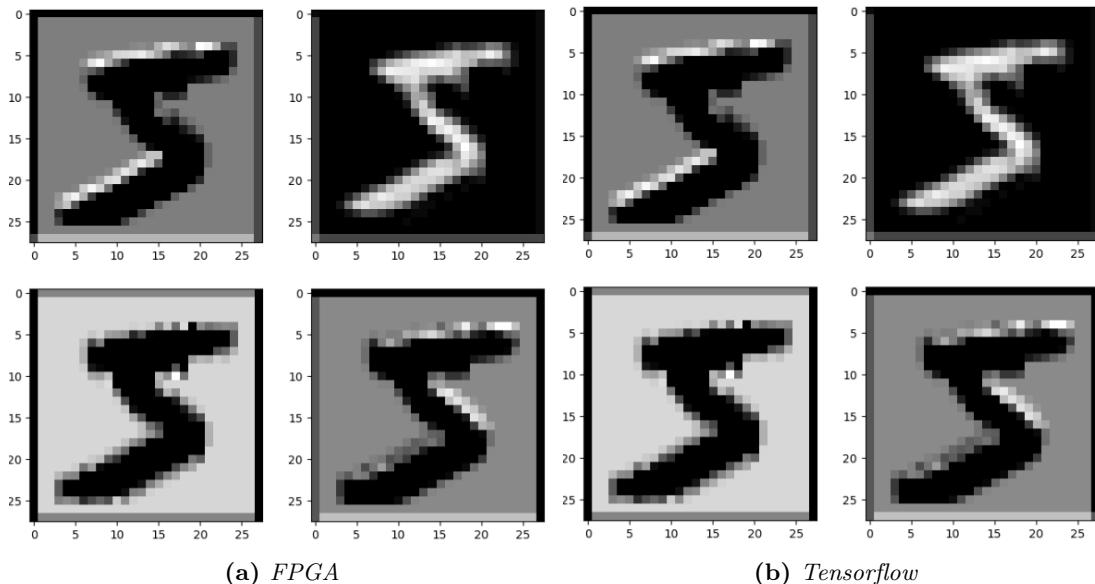
4.1 Simulation

Dựa vào kế hoạch kiểm thử ta tạo một testbench gồm thanh ghi lưu 10000 dữ liệu ảnh từ Test-set MNIST đã được làm phẳng và chuyển thành dạng nhị phân 8-bit và thanh ghi chứa label của từng ảnh. Tạo vòng lặp để lần lượt đưa từng pixel ảnh vào khối DUT để nhận diện đưa qua bộ Checker để so sánh kết quả với label của ảnh và đưa ra console.

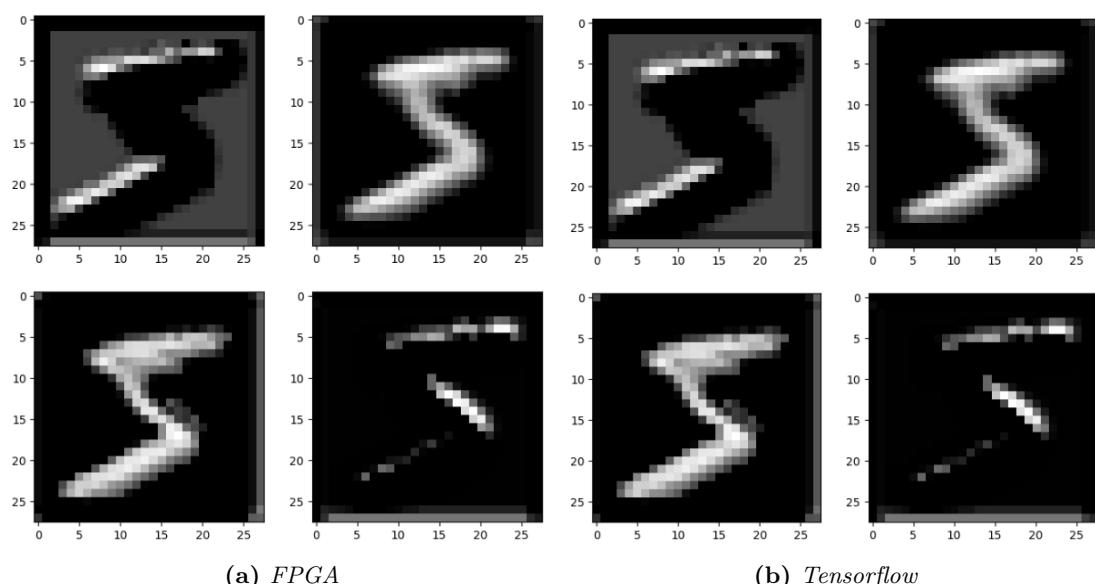


Hình 4.1: Sơ đồ khối test simulation

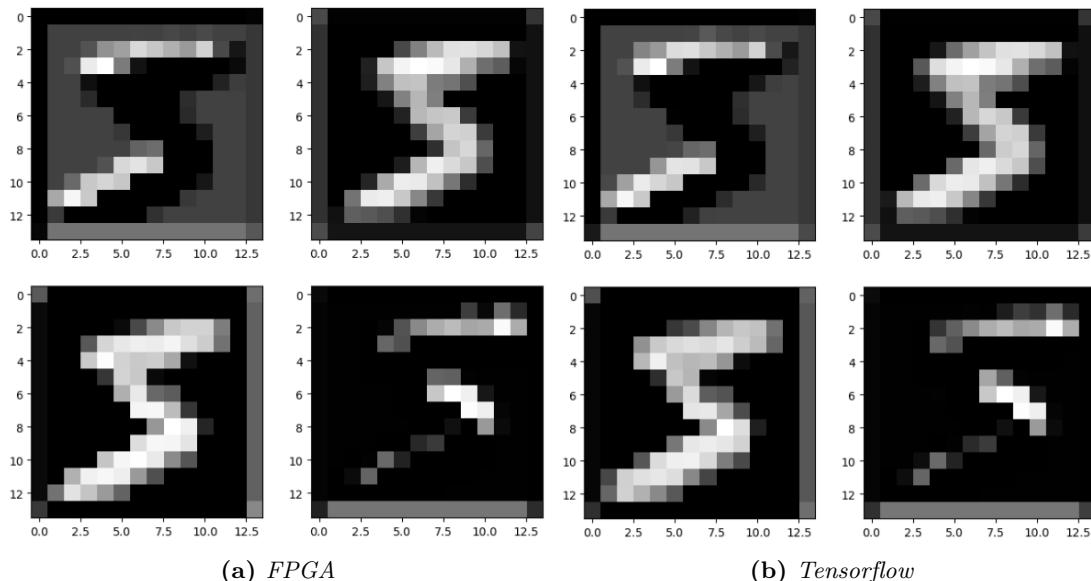
Dưới đây là các hình so sánh ngõ ra từng lớp khi chạy tính toán CNN trên FPGA ta thiết kế và model CNN chạy trên Tensorflow. Dữ liệu ảnh từng lớp ngõ ra ta lấy bằng cách dùng tính năng *Export data patterns* để lấy dữ liệu lưu trong khối BSRam và dựa vào bảng memory map 13 để lấy ra ảnh của từng lớp.



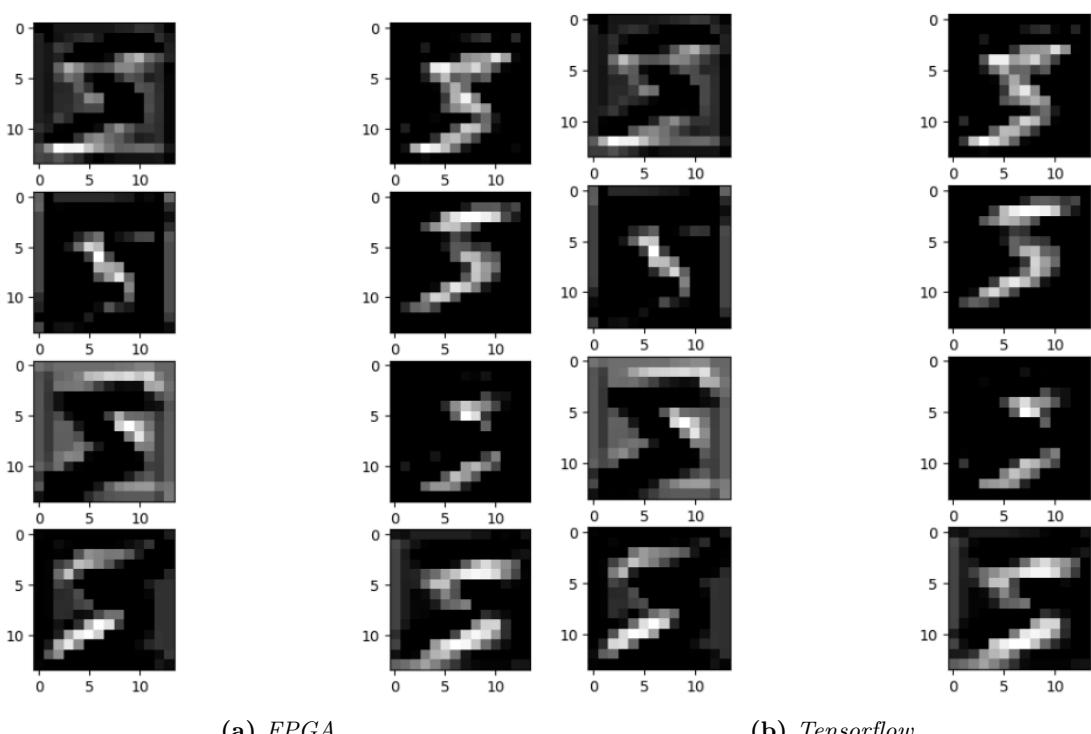
Hình 4.2: *Ảnh ngõ ra lớp Conv1*



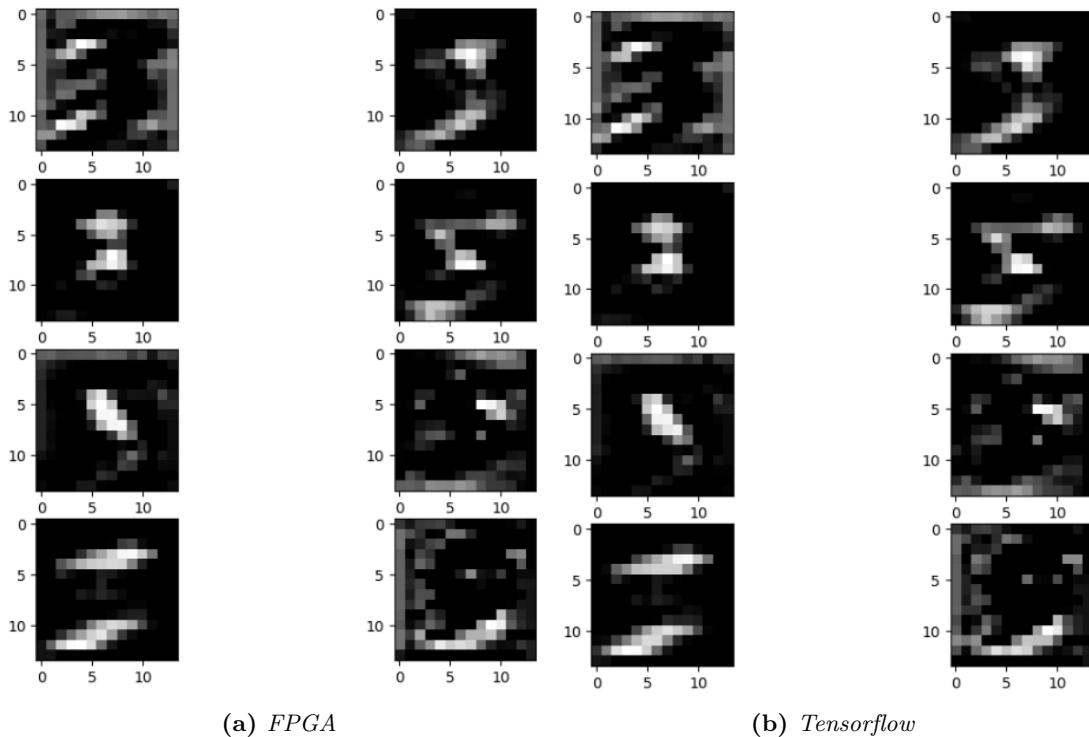
Hình 4.3: *Ảnh ngõ ra lớp Conv2*



Hình 4.4: Ảnh ngõ ra lớp $Mpool1$



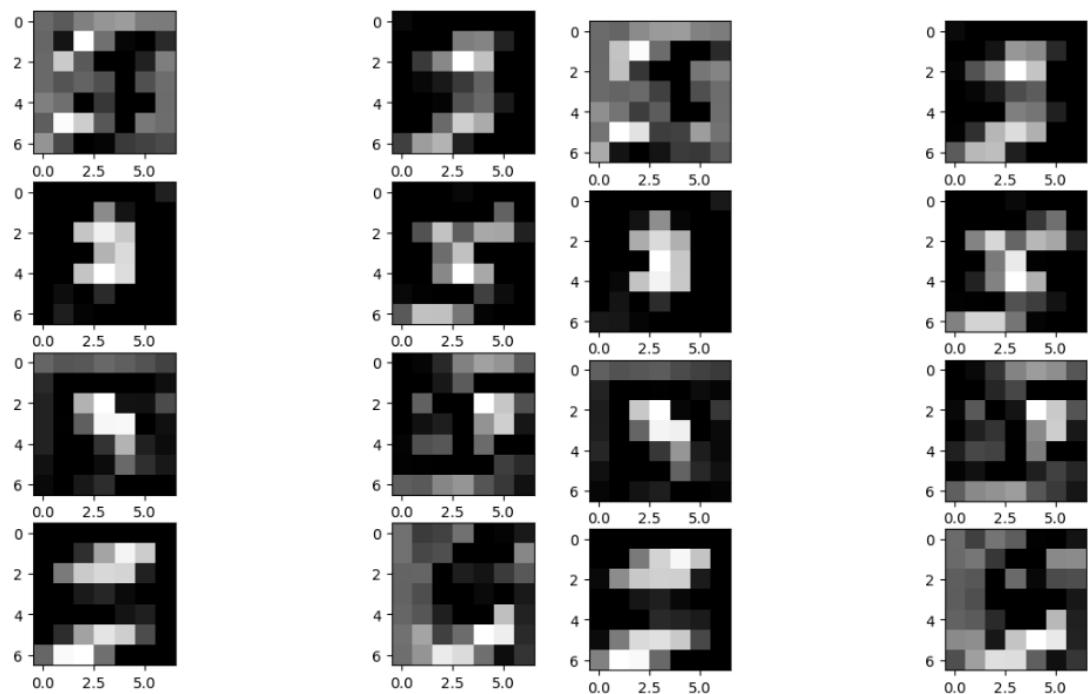
Hình 4.5: Ảnh ngõ ra lớp $Conv3$



(a) *FPGA*

(b) *Tensorflow*

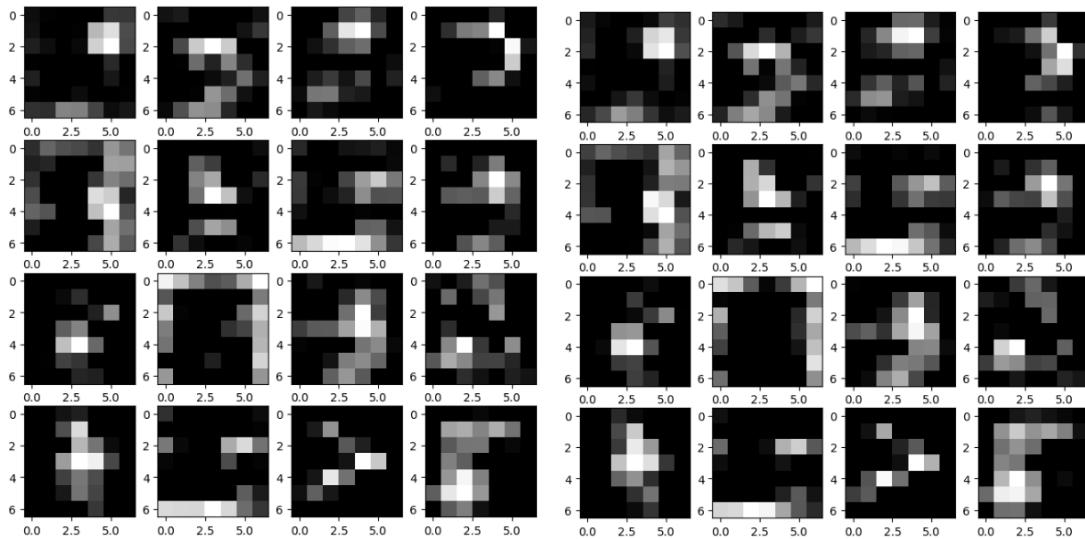
Hình 4.6: *Ảnh ngõ ra lớp Conv4*



(a) *FPGA*

(b) *Tensorflow*

Hình 4.7: *Ảnh ngõ ra lớp Mpool2*



(a) *FPGA*

(b) *Tensorflow*

Hình 4.8: *Ảnh ngõ ra lớp Conv5*

```
[5.2890625 3.92578125 3.0390625 5.07421875 4.171875 1.67578125
 3.1015625 3.1953125 5.9765625 1.45703125 4.11328125 4.57421875
 4.3671875 2.53125 2.23046875 4.1953125 ]
```

(a) *FPGA*

```
[5.36134192 3.90686803 3.19106441 4.37567185 4.36000084 1.56711426
 2.94919049 3.64318054 5.88770314 1.82071261 5.06273605 5.48654882
 4.55651431 2.60375804 2.17327976 4.80474815]
```

(b) *Tensorflow*

Hình 4.9: *Kết quả ngõ ra Global max pooling*

Ở kết quả ngõ ra Global max pooling có thể thấy kết quả 2 bên có phần chênh lệch với nhau do các trọng số trên FPGA đã được lượng tử hóa nên sẽ có sai số lượng tử.



```
P Terminal - ca104@blue:~/cnnverilog/xm
File Edit View Terminal Tabs Help
Test 9979: [ CORRECT ] Predict 1|Label 1( 9130/ 9979)
Test 9980: [ CORRECT ] Predict 7|Label 7( 9131/ 9980)
Test 9981: [ CORRECT ] Predict 2|Label 2( 9132/ 9981)
Test 9982: [ CORRECT ] Predict 6|Label 6( 9133/ 9982)
Test 9983: [ CORRECT ] Predict 5|Label 5( 9134/ 9983)
Test 9984: [ CORRECT ] Predict 0|Label 0( 9135/ 9984)
Test 9985: [ CORRECT ] Predict 1|Label 1( 9136/ 9985)
Test 9986: [ CORRECT ] Predict 2|Label 2( 9137/ 9986)
Test 9987: [ INCORRECT] Predict 8|Label 3( 9137/ 9987)
Test 9988: [ CORRECT ] Predict 4|Label 4( 9138/ 9988)
Test 9989: [ CORRECT ] Predict 5|Label 5( 9139/ 9989)
Test 9990: [ CORRECT ] Predict 6|Label 6( 9140/ 9990)
Test 9991: [ CORRECT ] Predict 7|Label 7( 9141/ 9991)
Test 9992: [ CORRECT ] Predict 8|Label 8( 9142/ 9992)
Test 9993: [ CORRECT ] Predict 9|Label 9( 9143/ 9993)
Test 9994: [ CORRECT ] Predict 0|Label 0( 9144/ 9994)
Test 9995: [ CORRECT ] Predict 1|Label 1( 9145/ 9995)
Test 9996: [ CORRECT ] Predict 2|Label 2( 9146/ 9996)
Test 9997: [ CORRECT ] Predict 3|Label 3( 9147/ 9997)
Test 9998: [ CORRECT ] Predict 4|Label 4( 9148/ 9998)
Test 9999: [ CORRECT ] Predict 5|Label 5( 9149/ 9999)
Test 10000: [ CORRECT ] Predict 6|Label 6( 9150/10000)

-----
Result: 9150/      10000
Simulation complete via $finish(1) at time 70837499995 NS + 2
./tb/tb_cnn_data.sv:71           $finish;
xcelium> exit
TOOL:  xrun(64)      24.09-s005: Exiting on Apr 16, 2025 at 23:42:37 +07  (to
tal: 01:30:32)
[ca104@black xm]$
```

Hình 4.10: Giả lập dự đoán tập dữ liệu test MNIST trên Xcelium

Tiến hành test trên bộ dữ liệu Test-set MNIST ta thấy tỉ lệ dự đoán chính xác là 91.5%, giảm 0.86% so với mô hình Binary CNN ta tính toán trên máy tính là 92.36% (Hình 2.8) do ta lượng tử hóa các trọng số về kiểu dữ liệu số fixed-point Q4.8. Thời gian nhận diện 10000 ảnh là 70837499995ns với xung clock với chu kì 10ns vì chi ta tốn 708375 chu kì clock để nhận diện 1 ảnh. Nếu Module nhận diện số viết tay này sử dụng xung clock 10MHz ta sẽ tốn $\approx 71ms$ để xử lí 1 ảnh.

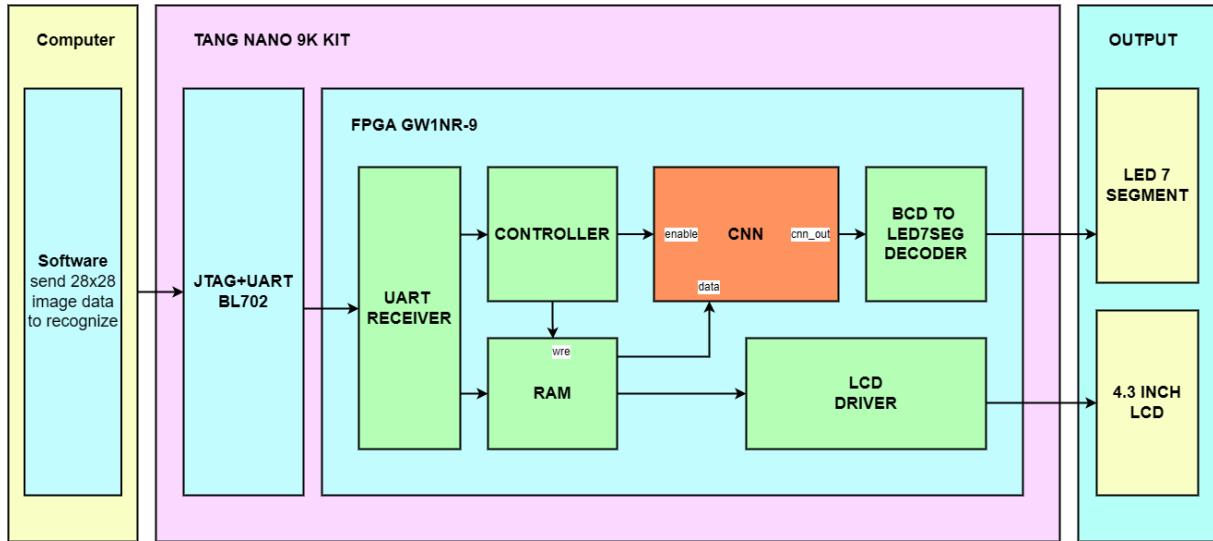
Bảng 18: *FPGA-based MNIST Recognition Models Comparison*

Model Architecture	Binarized CNN (BNN)	8-bit Quantized CNN	3-Layer MLP	Pruned CNN (Wino-grad)	Spiking Neural Network	Hybrid CNN-SVM
FPGA Device	Xilinx Zynq-7020	Xilinx Virtex-7	Intel Cyclone V	Xilinx Kintex-7	Xilinx Spartan-6	Xilinx Zynq UltraScale
LUTs	4,500	120,000	2,300	85,000	6,200	65,000
Clock Cycles	30	50	100	20	500	70
Accuracy (%)	96.4	99.1	92.0	98.5	89.7	98.8
Freq. (MHz)	200	150	100	250	50	180
Power (W)	1.5	4.2	0.8	3.0	0.5	2.5
Toolchain	FINN	Vivado HLS	Quartus	Vitis AI	Verilog	PYNQ
Year	2018	2020	2017	2021	2019	2022
Remarks	Binary weights/activations; low power	High DSP usage (256 blocks); pipelined	Minimal resources; no DSP	Winograd transforms; 80% pruned	Event-driven; ultra-low power	CNN + SVM classifier

So sánh model của đồ án này với các model MNIST FPGA khác thì còn nhiều mặt hạn chế như tốc độ xử lý và độ chính xác tuy nhiên cấu trúc của thiết kế đơn giản phù hợp cho việc nghiên cứu cách hoạt động của mạng CNN cũng như FPGA so với các kiến trúc được tạo tự động từ các tool HLS (High-Level Synthesis).

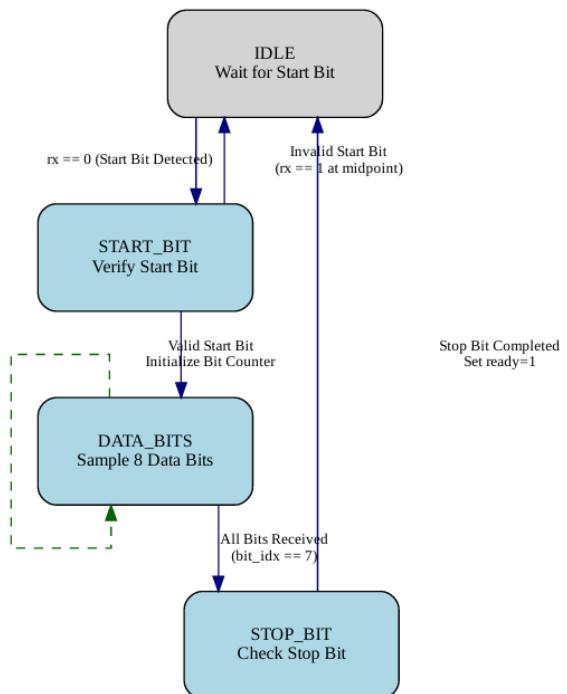
4.2 Dưa lên kit thực tế

Để kiểm tra hoạt động của thiết kế ở thực tế ta xây dựng hệ thống đơn giản bằng cách truyền dữ liệu ảnh xám 28x28 cần nhận diện qua UART module nhận diện chữ viết tay như hình 4.11.



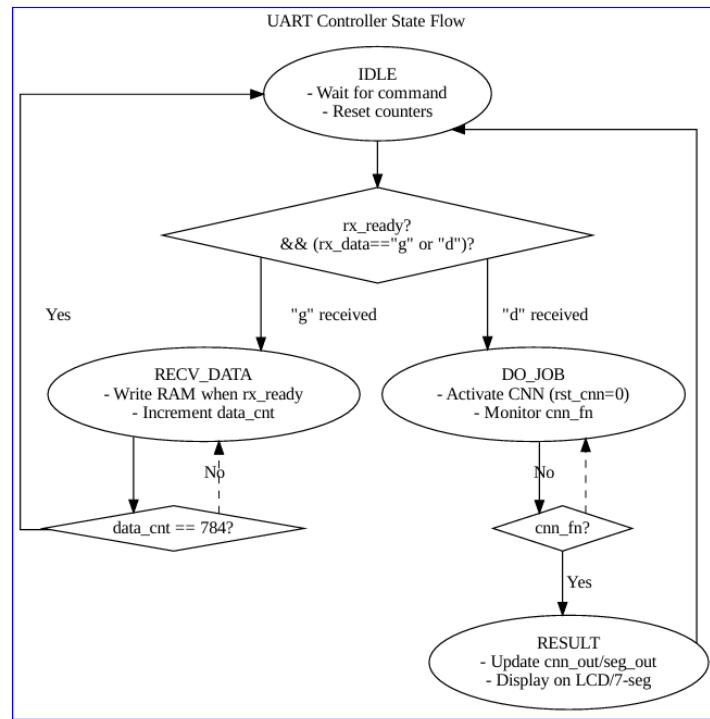
Hình 4.11: Sơ đồ khái niệm hệ thống thực tế

Với khối UART receiver ta thiết kế như lưu đồ sau.



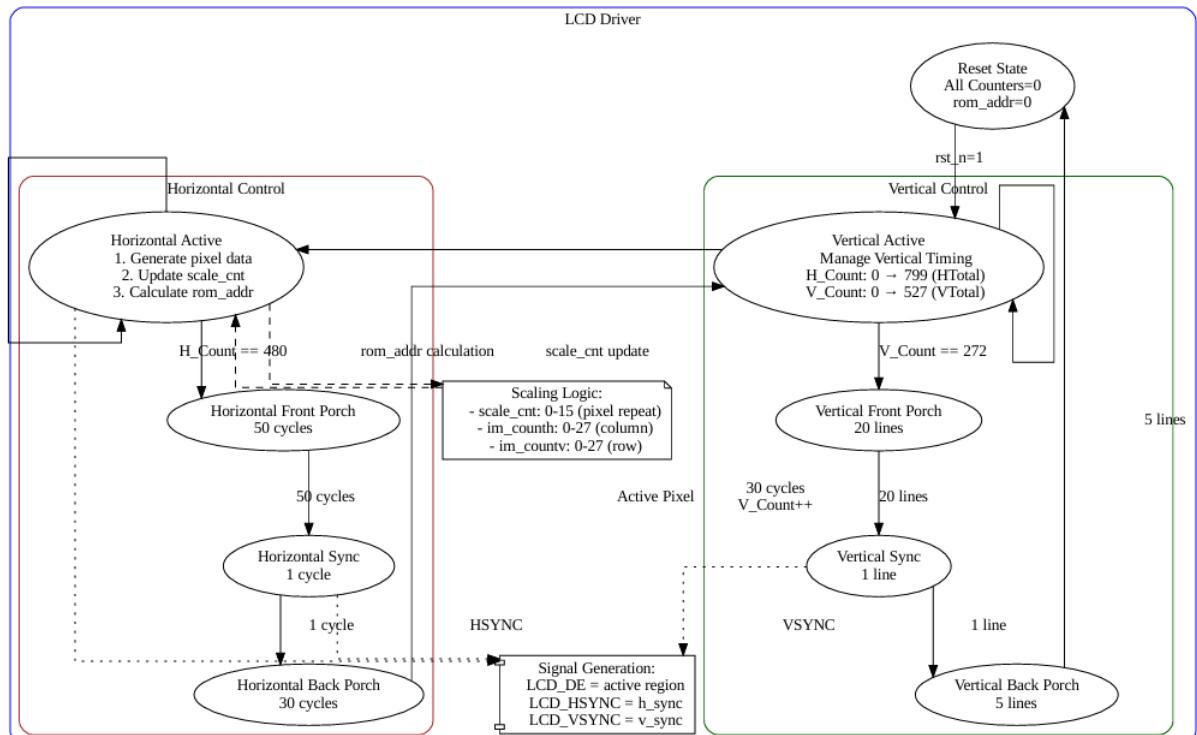
Hình 4.12: UART receiver fsm

Khối controller sẽ nhận các lệnh từ UART bao gồm lệnh load ảnh vào RAM (*nhận kí tự "g"*) và lệnh bắt đầu nhận diện số viết tay (*nhận kí tự "d"*).



Hình 4.13: Controller flow chart

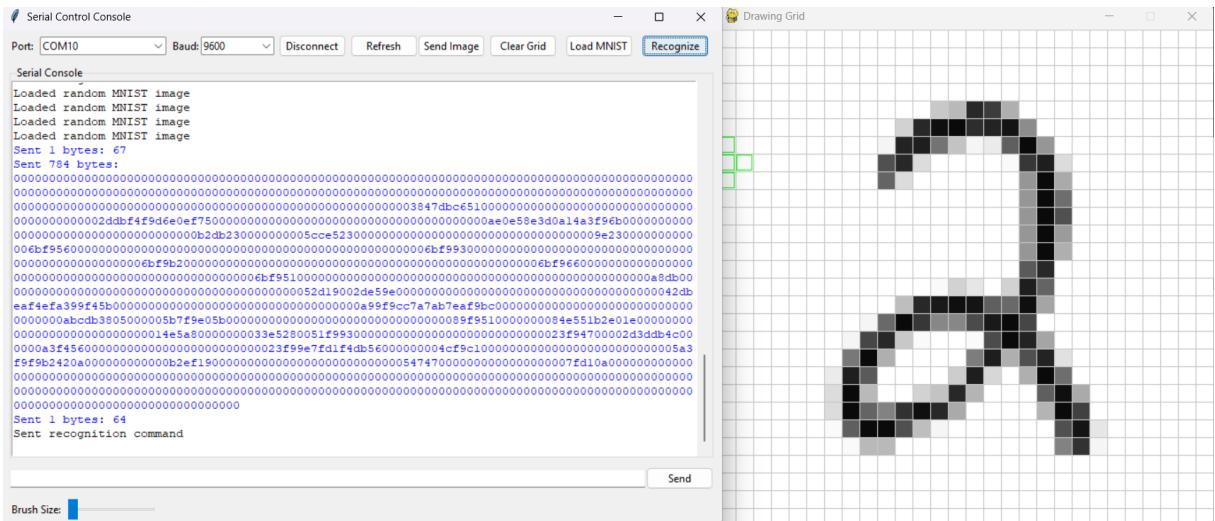
Khối LCD Driver có nhiệm vụ đọc dữ liệu ảnh từ RAM và điều khiển các tín hiệu quét ngang và dọc để đưa ảnh lên màn hình.



Hình 4.14: LCD driver flow chart

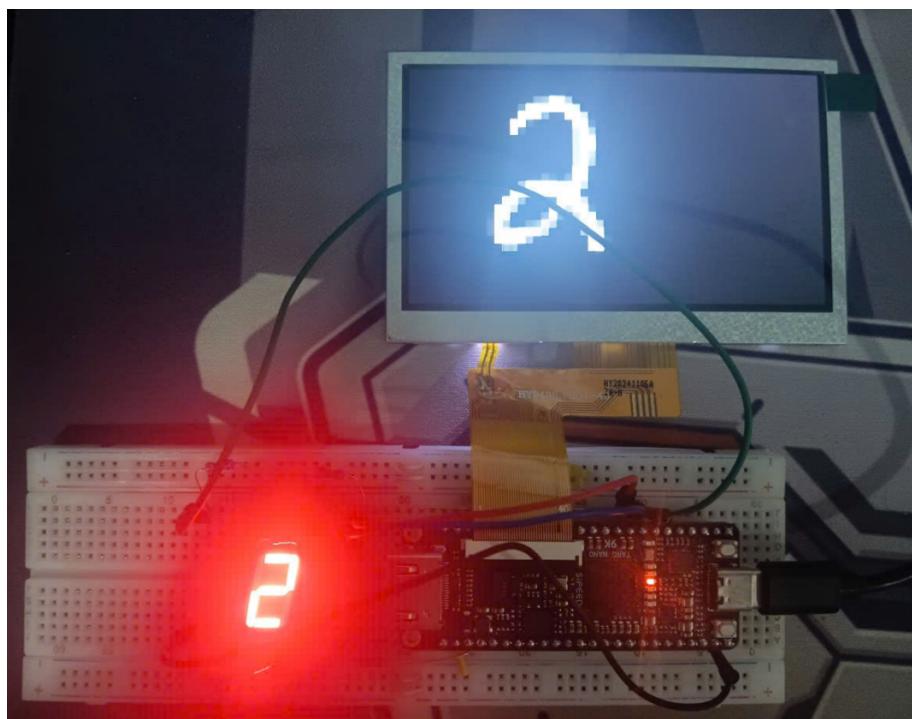
Ngoài ra ta cần viết thêm phần mềm đơn giản bằng python sử dụng thư viện Tkinter để

làm UI và Pyserial để giao tiếp UART để gửi lệnh và dữ liệu ảnh số viết tay cần nhận diện và có thêm tính năng cho người dùng dùng vẽ số viết tay để kiểm tra hoạt động của thiết kế.



Hình 4.15: Giao diện phần mềm

Gửi hình ảnh số 2 trên phần mềm như hình 4.15 và gửi lệnh nhận diện ta có kết quả ngõ ra hiển thị ngõ ra của khối CNN trên led của kit là 0100 và kết quả hiển thị led 7 đoạn là 2.



Hình 4.16: Kết quả nhận diện trên FPGA

Bảng dưới đây là báo cáo tài nguyên mà hệ thống này sử dụng trên kit Tang Nano 9K.



Bảng 19: Resource Utilization Summary

Resource	Used/Total (units)	Utilization
Logic	3305 (1841 LUT, 1446 ALU, 3 RAM16) / 8640	39.0%
Register	519 / 6693	8.0%
• Register as Latch	0 / 6693	0.0%
• Register as FF	519 / 6693	8.0%
BSRAM	20 / 26	77.0%

Với báo cáo timming với $F_{max} = 15.733MHz$ trên kit Tang Nano 9K để cho dễ dàng canh thời gian cho khung truyền UART ta chọn được tần số 10.8Mhz với baudrate 9600 vì 10.8Mhz chia hết cho 9600 thì UART sẽ truyền dữ liệu tốt nhất.

Timing

Clock Summary:

NO.	Clock Name	Type	Period	Frequency(MHz)	Rise	Fall	Source	Master	Object
1	xtal_clk	Base	37.037	27.0	0.000	18.519			xtal_clk_ibuf/I
2	rpll/rpll_inst/CLKOUT.default_gen_clk	Generated	92.593	10.8	0.000	46.296	xtal_clk_ibuf/I	xtal_clk	rpll/rpll_inst/CLKOUT
3	rpll/rpll_inst/CLKOUTP.default_gen_clk	Generated	92.593	10.8	0.000	46.296	xtal_clk_ibuf/I	xtal_clk	rpll/rpll_inst/CLKOUTP
4	rpll/rpll_inst/CLKOUTD.default_gen_clk	Generated	185.185	5.4	0.000	92.593	xtal_clk_ibuf/I	xtal_clk	rpll/rpll_inst/CLKOUTD
5	rpll/rpll_inst/CLKOUTD3.default_gen_clk	Generated	277.778	3.6	0.000	138.889	xtal_clk_ibuf/I	xtal_clk	rpll/rpll_inst/CLKOUTD3

Max Frequency Summary:

NO.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	rpll/rpll_inst/CLKOUT.default_gen_clk	10.800(MHz)	15.733(MHz)	47	TOP

Hình 4.17: Báo cáo timming hệ thống

5 ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN

Sau thời gian thực hiện đồ án, em có những đánh giá và kết luận chung sau khi hoàn thành như sau:

- Về những kết quả thu được:

- **Thiết kế hệ thống phần cứng:** Đã thành thạo quy trình thiết kế hệ thống trên FPGA, bao gồm phân tích yêu cầu, chia module chức năng (LCD driver, UART controller, CNN) và triển khai bằng Verilog/SystemVerilog.
- **Mô phỏng và kiểm tra:** Làm chủ công cụ mô phỏng (ModelSim/Xcelium) để verify chức năng các module, phát hiện lỗi timing thông qua testbench tự động và phân tích waveform.
- **Nâng cao kỹ năng tích hợp:** Rèn luyện khả năng tích hợp đa module (kết hợp phần cứng-phần mềm), xử lý giao tiếp UART, điều khiển VGA và tối ưu hóa pipeline cho bộ suy luận CNN.

- Về những hạn chế còn tồn đọng:

- **Thời gian xử lý:** Thời gian xử lý còn chậm so với các model cùng chức năng khác.
- **Tối ưu tài nguyên:** Chưa khai thác hiệu quả BRAM/DSP slices trên FPGA, dẫn đến lãng phí tài nguyên và hạn chế tốc độ xử lý.
- **Chưa xử lý lỗi thời gian thực:** Thiếu cơ chế phục hồi khi gặp lỗi giao tiếp UART hoặc mất đồng bộ tín hiệu VGA, gây treo hệ thống.

Qua những đánh giá trên, bản thân đã:

- + Tiếp thu sâu về kiến trúc FPGA và phương pháp thiết kế RTL
- + Phát triển tư duy hệ thống thông qua tích hợp đa thành phần
- + Nhận thức rõ tầm quan trọng của timing closure và power analysis

Tuy nhiên, cần khắc phục ngay:

- Bổ sung kỹ thuật timing constraints (SDC) để đảm bảo Fmax
- Nâng cao kỹ năng debug bằng công cụ của tool và giả lập.
- Xây dựng testbench coverage-driven cho các trường hợp biên



6 TÀI LIỆU THAM KHẢO

Tài liệu

- [1] Hubara, I., Courbariaux, M., Soudry, D., & El-Yaniv, R. (2016). "Binarized Neural Networks". *Advances in Neural Information Processing Systems* (NIPS).
- [2] Umuroglu, Y., et al. (2017). "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference". *ACM/SIGDA International Symposium on FPGAs*.
- [3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE*.
- [4] Intel FPGA (2024). "Optimized CNN Acceleration for Cyclone V FPGAs". *Intel White Paper WP-01234*.
- [5] Rastegari, M., et al. (2016). "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". *European Conference on Computer Vision (ECCV)*.
- [6] Yann LeCun & Corinna Cortes. (1998). "The MNIST Database of Handwritten Digits". [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [7] IEEE Standard 1364-2005 (2005). *IEEE Standard for Verilog Hardware Description Language*.