The Design Document for CSEE 4840 Embedded System Design

Haichun Zhao (hz2833)
Haomiao Li (hl3619)
Qixiao Zhang (qz2487)
Tianchen Yu (ty2485)
Yue Niu (yn2433)

Contents:

1. Introduction

Considering the feasibility and workload of this project, we decided to alter our project to a simpler CNN inference accelerator. We are going to use the Fashion MNIST to train the model and then use FPGA to implement the inference. We will use the Linux system built in SD card to store the test data and then use ioctl to transfer data into the board.

Thanks to the hardware design from Tianchen that he finished in course EE6350, we can move forward our project to the FPGA part. Besides, we have confirmed that the non-parallel version algorithm in the hardware only costs less than 1% resource utilization on the board. Therefore, we can move on to make our algorithm parallel so that the CNN can truly be accelerated. The communication system and other details will be described as follows.

2. Overall Block Diagram

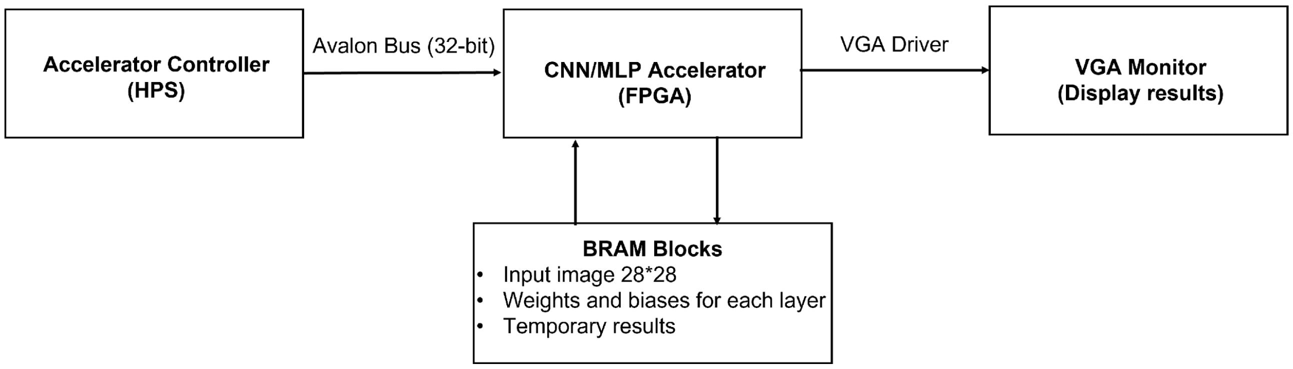

Accelerator Controller (HPS)
Avalon Bus (32-bit)
CNN/MLP Accelerator (FPGA)
VGA Driver
VGA Monitor (Display results)
BRAM Blocks
• Input image 28*28
• Weights and biases for each layer
• Temporary results

Figure 1. Overall block diagram of the system

The overall system block diagram of our project is shown in Figure 1. Generally our project focuses on constructing a CNN/MLP hardware accelerator to accelerate a certain neural network algorithm. The input to the accelerator will be pre-stored image pixel value, network weight, network bias or the computation results from last layer. The hardware will take the control signals (instructions) from software (HPS) and fetch the input values from corresponding memory addresses until the all the computation is completed. The output results of the neural network (results of an image recognition task) will be displayed directly by the hardware (VGA driver) on the VGA display in text format.

3. Neural Network Structure

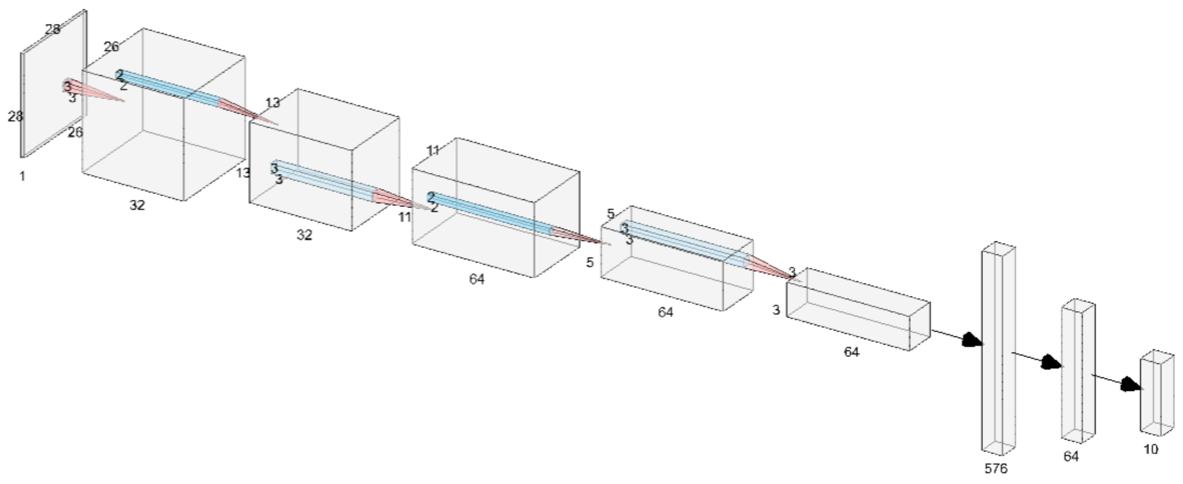
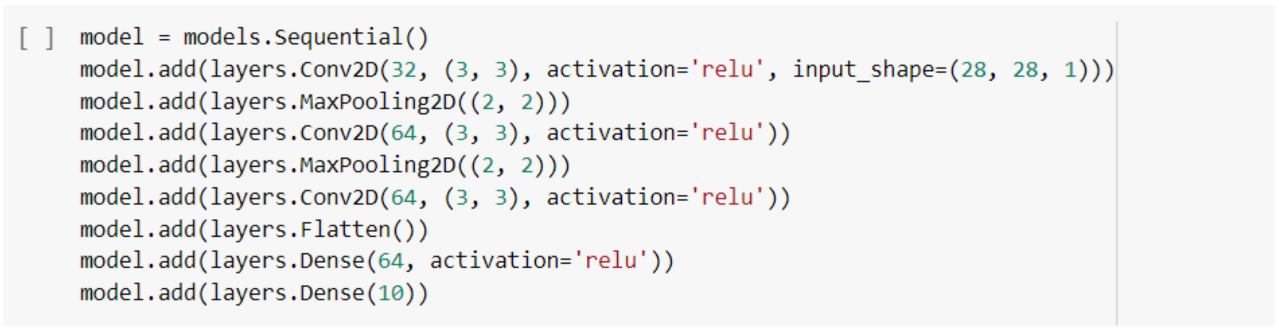


Figure 2. CNN architecture of our algorithm



```
[ ] model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10))
```

Figure 3. Network specification in TensorFlow

A typical CNN architecture is chosen for our project as shown in Figure 2. Totally there are four possible operations in the algorithm, 2D convolution, maxpooling, image flatten, fully connected layer. All of them are standard CNN operations. MNIST or fashion MNIST dataset will be used to benchmark the performance of our system, thus the input matrix size is (28,28,1). Both MNIST and fashion MNIST has 10 output possibilities thus the final output layer has 10 neurons. The network structure in the middle should be clear as shown in Figure 2 and 3. This architecture achieves ~89% of accuracy when tested in TensorFlow for the fashion MNIST dataset, which is high enough to be implemented as a course project.

(*Note: originally we are trying to accelerate an architecture called MobileNet, but we simplified our network structure to this typical CNN architecture for the feasibility concern of the project.)

4. Memory Estimation

The memory usage of our CNN architecture could be calculated as follows:

$$CNN\ Coefficients = (3\times3 + 32 + 3\times3\times32 + 64 + 3\times3\times64 + 64)\times8\ bit = 1033\ B$$

$$FC\ Coefficients = (576\times64 + 64 + 64\times10 + 10)\times8\ bit = 37578\ B$$

$$Max\ data\ memory\ of\ two\ adjacent\ layers = (26\times26\times32 + 13\times13\times32)\times8\ bit = 27040\ B$$

$$Input\ picture = 28\times28\times8\ bit = 784\ B$$

Which means the memory requirement for our project is:

$$Total\ memory = 1033\ B + 37578\ B + 27040\ B + 784\ B = 66435\ B\approx67\ kB$$

This value is much smaller than the total amount of block memory (BRAM) in the FPGA (500 kB), indicating that no external SDRAM is needed.

5. Architecture specification of accelerator (Hardware)

The top-level architecture is consisting of three blocks, a computing core, a FSM and a SSFR as shown in Figure 4. The computing core is responsible for taking inputs from memory blocks and generate MAC results to the output. Relevant control signals for driving the computing core will be generated automatically from FSM. The SSFR stores some configuration options of the core and is specified in Table 1. The detailed implementation of computing core (NPU core) is shown in Figure 5. The architecture and state transition table of FSM are shown in Figure 6 and Table 2 and 3. The operation timing diagram is shown in Figure 7.

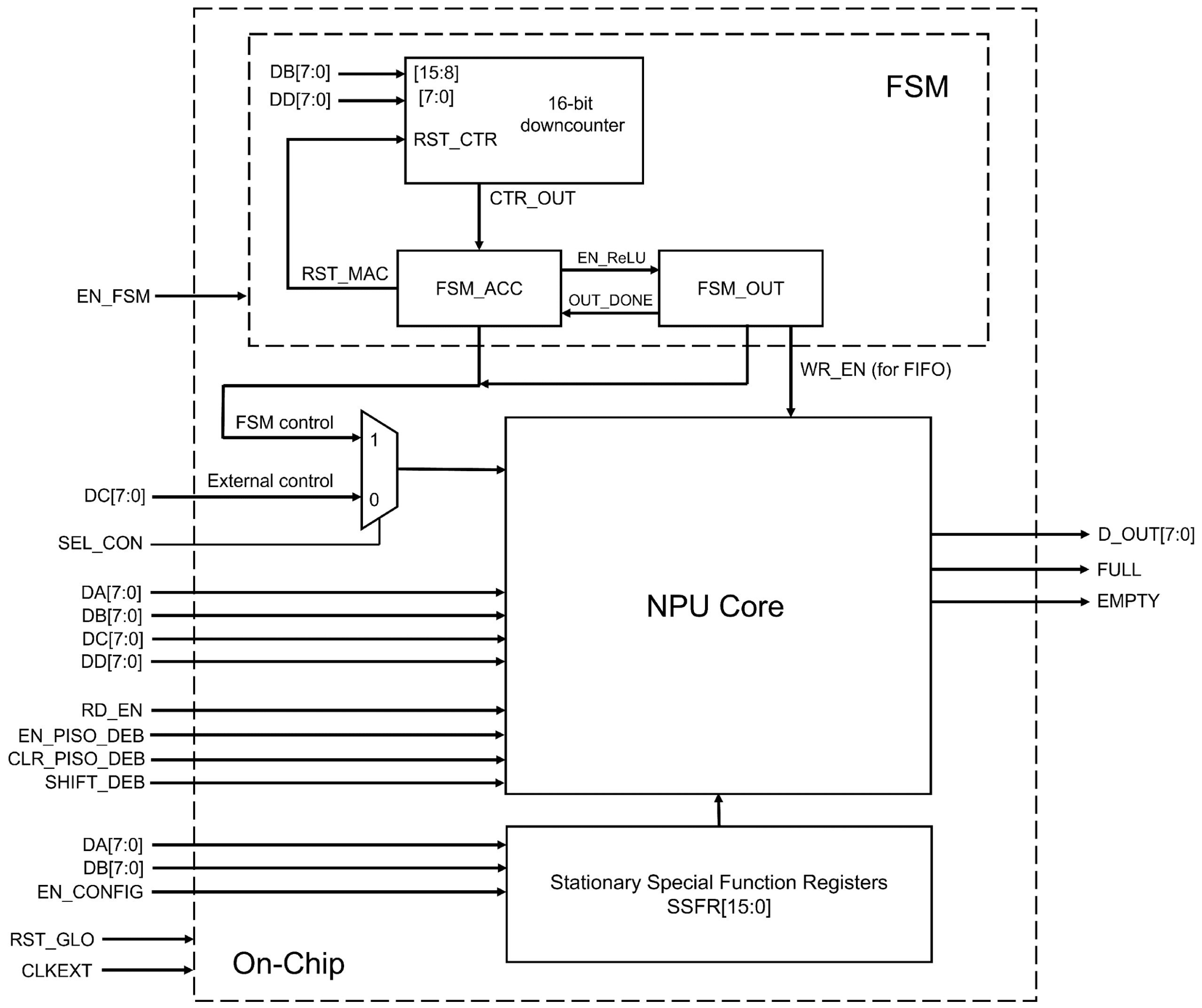


Figure 4. Top level architecture of accelerator

Table 1. Specification of each bit of SSFR

Stationary Special Function Registers (SSFR[15:0])							
SSFR[15]	SSFR[14]	SSFR[13]	SSFR[12]	SSFR[11]	SSFR[10]	SSFR[9]	SSFR[8]
SEL_OUT[2]	SEL_OUT[1]	SEL_OUT[0]	BYPASS_ReLU1	BYPASS_ReLU2	EN_COMP	RST_COMP	EN_FIFO
SSFR[7]	SSFR[6]	SSFR[5]	SSFR[4]	SSFR[3]	SSFR[2]	SSFR[1]	SSFR[0]
RST_FIFO	Unused						
Default Values (if reset)							
SSFR[15:8]	SSFR[7:0]						
00100010	10000000						

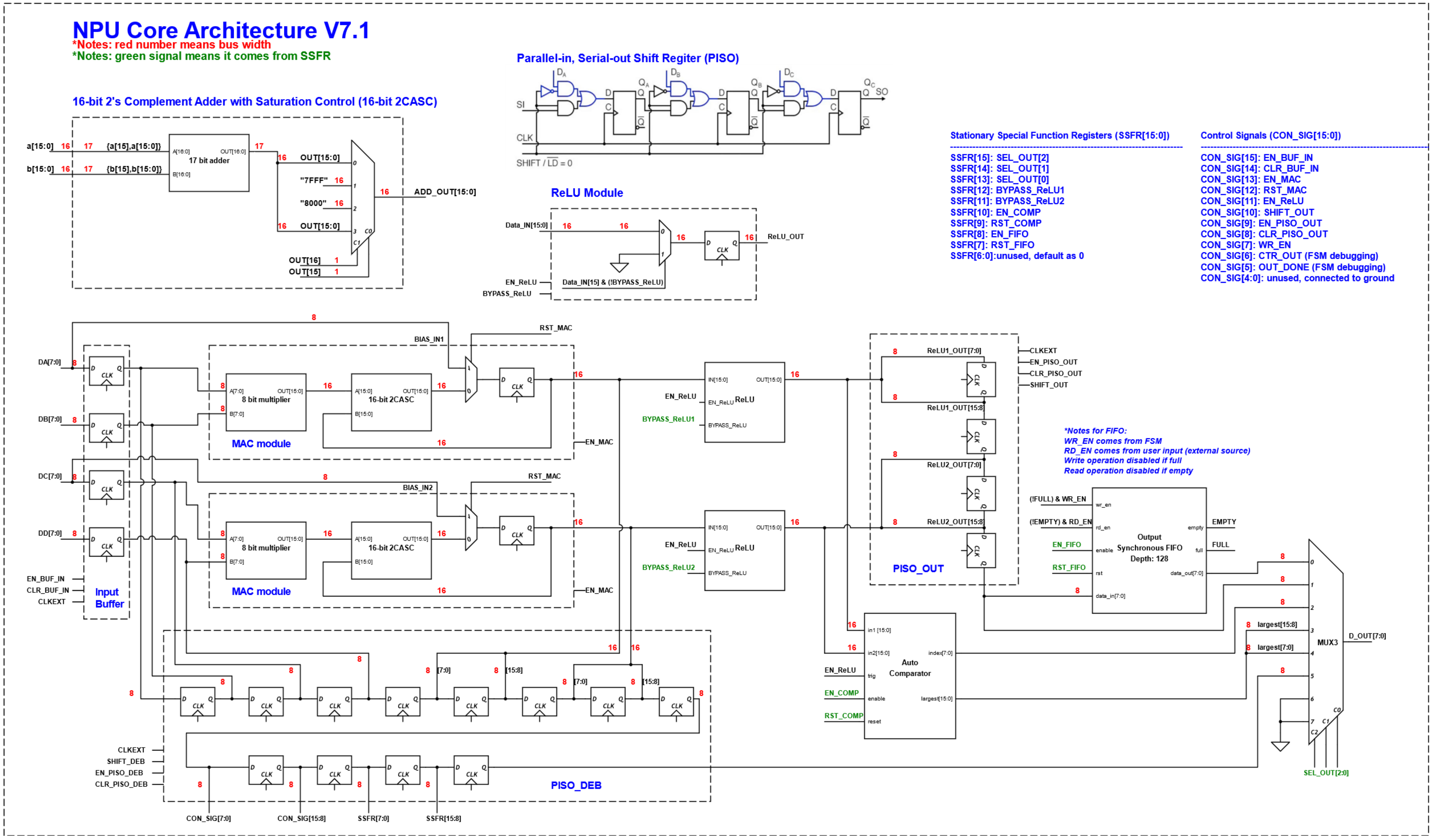


Figure 5. Detailed block diagram of the NPU core

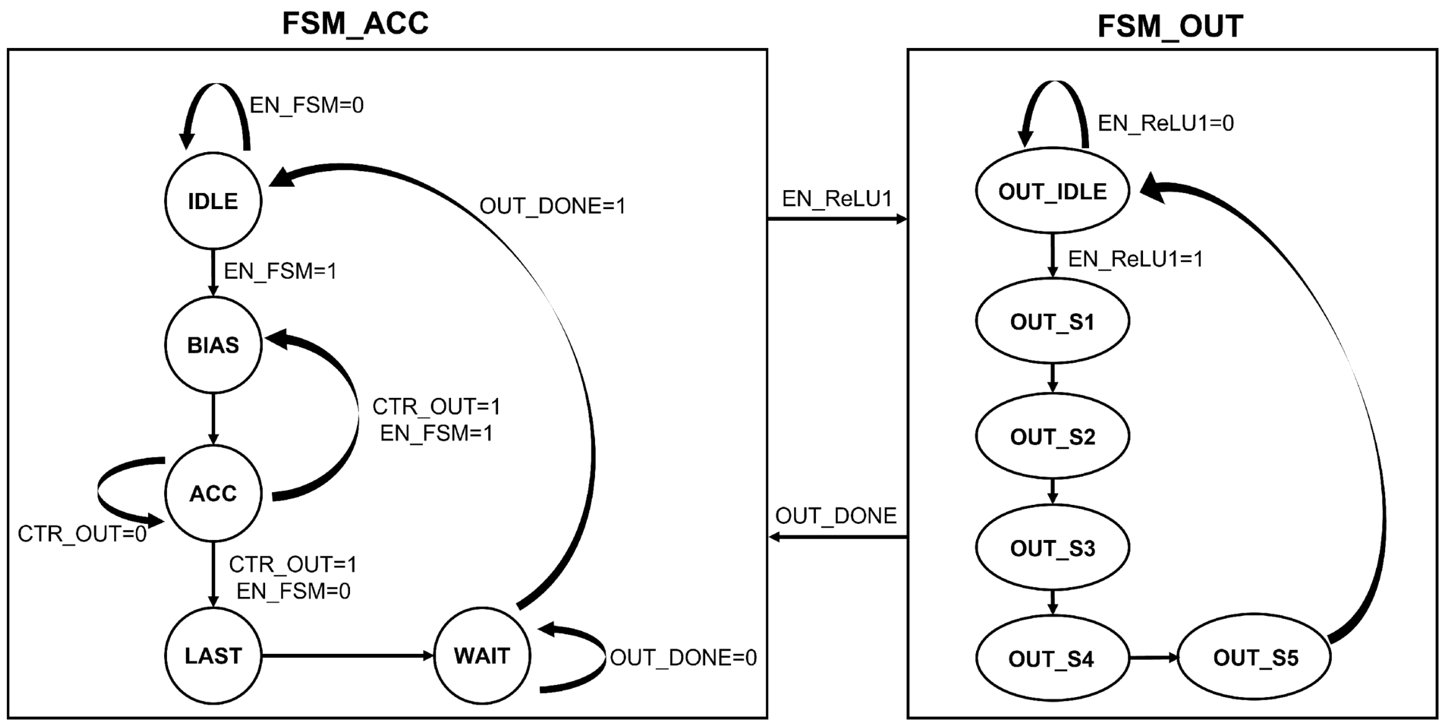


Figure 6. FSM architecture

Table 2. FSM ACC Transition Table

FSM_ACC State Transition Table (Architecture V7)												
Current State	Next State				Output							
	EN_FSM=0 CTR_OUT=0	EN_FSM=0 CTR_OUT=1	EN_FSM=1 CTR_OUT=0	EN_FSM=1 CTR_OUT=1	EN_BUF_IN	CLR_BUF_IN	EN_MAC	RST_MAC	CLR_PISO_OUT	EN_ReLU		ACC_FLAG
										ACC_FLAG =0	ACC_FLAG =1	
IDLE	IDLE	IDLE	BIAS	BIAS	0	1	0	0	1	0	0	0
BIAS	ACC	ACC	ACC	ACC	0	1	1	1	0	0	1	No change
ACC	ACC	LAST	ACC	BIAS	1	0	1	0	0	0	0	1
LAST	WAIT	WAIT	WAIT	WAIT	0	0	1	0	0	1	1	No change
	OUT_DONE=0		OUT_DONE=1									
WAIT	WAIT		IDLE		0	0	0	0	0	0	0	No change

Table 3. FSM OUT Transition Table

FSM_OUT State Transition Table (Architecture V7)						
Current State	Next State		Output			
	EN_ReLU=0	EN_ReLU=1	Shift/~LD_OUT	EN_PISO_OUT	OUT_DONE	WR_EN
OUT_IDLE	OUT_IDLE	OUT_S1	1	0	0	0
OUT_S1	OUT_S2	OUT_S2	0	1	0	0
OUT_S2	OUT_S3	OUT_S3	1	1	0	1
OUT_S3	OUT_S4	OUT_S4	1	1	0	1
OUT_S4	OUT_S4	OUT_S4	1	1	0	1
OUT_S3	OUT_IDLE	OUT_IDLE	1	0	1	1

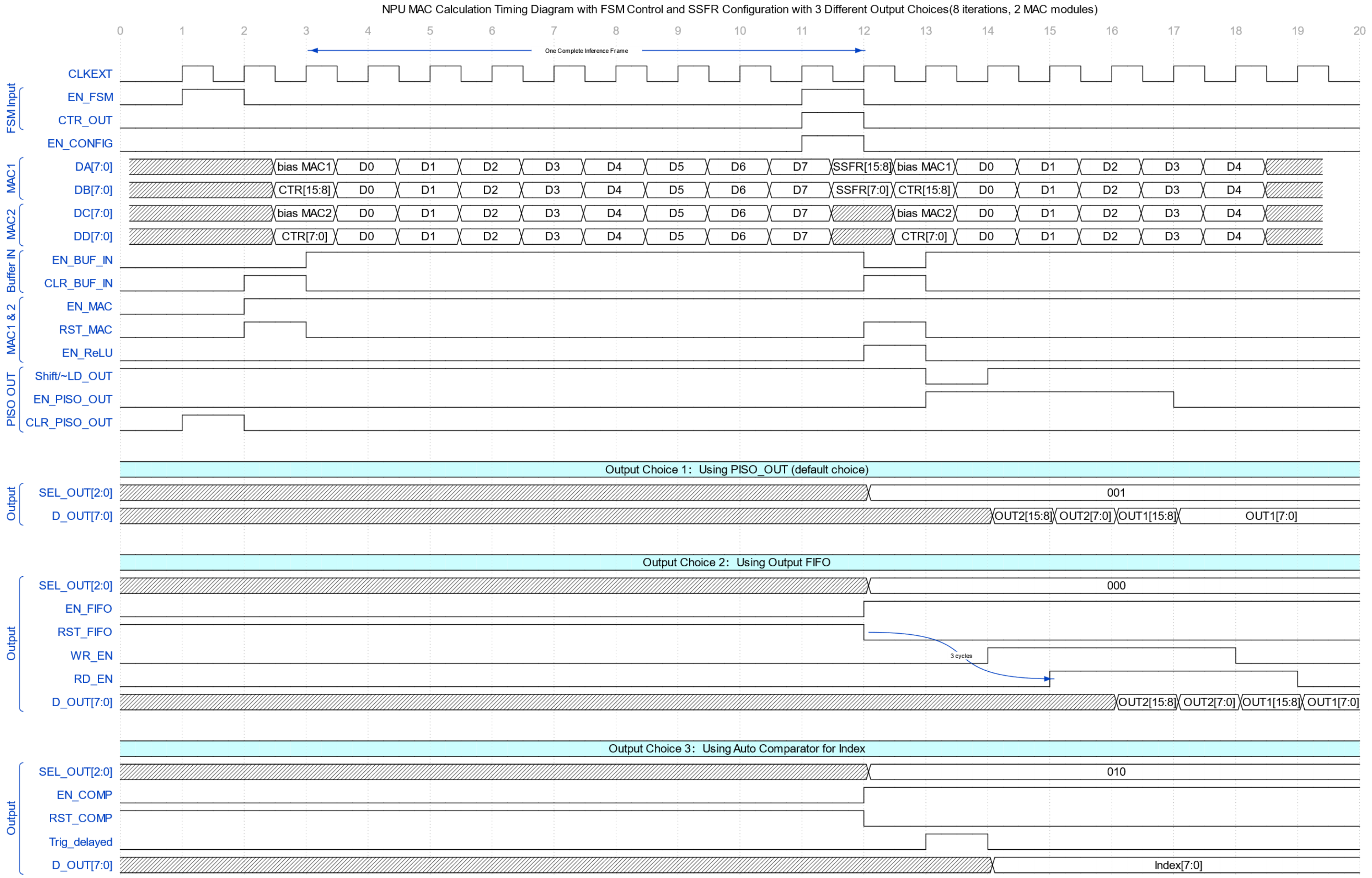


Figure 7. Computing core operation timing diagram

6. Hardware and Software Interface

The software and hardware interface via Avalon bus is not fully decided yet. A total of 32 bits transaction between HPS and FPGA is allowed. Generally ~10 bits will be used for controlling signals and the rest ~20 bits will be used as instruction code. An instruction set will be developed to tell the computing core about the memory address for the current computation.

The accelerator will be registered as a Linux device in the device tree. It will communicate with the FPGA portion through a device driver written in C code. Systemverilog file will specify incoming and outgoing signals which will eventually be passed to the userspace C program.

For embedded memories, we will be using 10 Kb M10k blocks to hold calculation results from each layer using dual-port RAM. The total size of M10k blocks in Cyclone V SE A4 is 3970 Kb, which is sufficient according to calculations in the previous section.

M10k can be invoked in a style like this:

```
//============================================================
// M10K module for testing
//============================================================

module M10K_256_32(
    output reg [31:0] q,
    input [31:0] d,
    input [7:0] write_address, read_address,
    input we, clk
);
         // force M10K ram style
    reg [31:0] mem [255:0]  /* synthesis ramstyle = "no_rw_check, M10K" */;

    always @ (posedge clk) begin
        if (we) begin
            mem[write_address] <= d;
        end
        q <= mem[read_address]; // q doesn't get d in this clock cycle
    end
endmodule
```

Source: DE1-SoC FPGA memory examples ECE 5760 Cornell University
https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/Memory/index.html

7. Milestones

1. CNN architecture prototype in C and implementation research
2. FPGA hardware upgrade to include memory blocks and optimized interface
3. Software controller development and overall system integration with benchmarking