CSE/EE – 7387
DIGITAL SYSTEMS DESIGN
FINAL PROJECT REPORT

ON

MATRIX MULTIPLICATION

Done by,
Arjun Ajit Warty
STUDENT # 47319796
Department of Electrical Engineering
Southern Methodist University

ASSIGNMENT DATE: DEC 07, 2017
DUE ON: DEC 07, 2017
SUBMISSION DATE: DEC 07, 2017

Table of Contents

OBJECTIVE:

The objective of this project is to implement a circuit that will multiply a 3x4 matrix with its transpose. That is, the circuit will compute B where B = AT . A. Each matrix element, a_{ij} is in 8-bit, 2's complement form.

CONCEPT FOR DESIGN

Mathematically the function of this circuit is shown in the image below with its equation.

Equation: $B = A^T * A$

$$\begin{matrix} A^T \\ \begin{matrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \\ a_{14} & a_{24} & a_{34} \end{matrix} \end{matrix} \quad X \quad \begin{matrix} A \\ \begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{matrix} \end{matrix} \quad = \quad \begin{matrix} B \\ \begin{matrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{matrix} \end{matrix}$$

Matrix Multiplication Concept:

Equation for Matrix Multiplication is a very simple equation but yet it one of the hardest to implement in a circuit with restrictions. The Equation for Matrix Multiplication is given below

$$b_{11} = a_{11} * a_{11} + a_{21} * a_{21} + a_{31} * a_{31}$$

$$b_{12} = a_{11} * a_{12} + a_{21} * a_{22} + a_{31} * a_{32}$$

$$b_{13} = a_{11} * a_{13} + a_{21} * a_{23} + a_{31} * a_{33}$$

$$b_{14} = a_{11} * a_{14} + a_{21} * a_{24} + a_{31} * a_{34}$$

$$b_{22} = a_{12} * a_{12} + a_{22} * a_{22} + a_{32} * a_{32}$$

$$b_{23} = a_{12} * a_{13} + a_{22} * a_{23} + a_{32} * a_{33}$$

$$b_{24} = a_{12} * a_{14} + a_{22} * a_{24} + a_{32} * a_{34}$$

$$b_{33} = a_{13} * a_{13} + a_{23} * a_{23} + a_{33} * a_{33}$$

$$b_{34} = a_{13} * a_{14} + a_{23} * a_{24} + a_{33} * a_{34}$$

$$b_{44} = a_{14} * a_{14} + a_{24} * a_{24} + a_{34} * a_{34}$$

Thus, we have our 10 coefficients for the matrix B but the remaining six do not need to be calculated as they are equal to the other six coefficients namely, **$b_{21} = b_{12}$, $b_{13} = b_{31}$, $b_{14} = b_{41}$, $b_{23} = b_{32}$, $b_{24} = b_{42}$, $b_{34} = b_{43}$.**

THE DESIGN

We are able to implement the functionality of a Matrix Multiplier in Quartus II – Altera through many different techniques, but we always need to know the starting point of the project and that for me was the Scheduling Table.

SCHEDULING TABLE

My entire design is based on the table shown below. We only need to calculate 10 values, because not every of the 16 values are different. As the table shows, the initiation rate is 19 clock cycles and the latency is 28 cycles.

Clock Cycle	I/P	MAC	Output of MAC	Output
0	Initial State			
1	a11			
2	a21			
3	a31			
4	a12			
5	a22			
6	a32			
7	a13			
8	a23			
9	a33			
10	a14	$b_{11} = a_{11} * a_{11} + a_{21} * a_{21} + a_{31} * a_{31}$	b1	
11	a24	$b_{12} = a_{11} * a_{12} + a_{21} * a_{22} + a_{31} * a_{32}$	b2	
12	a34	$b_{13} = a_{11} * a_{13} + a_{21} * a_{23} + a_{31} * a_{33}$	b3	
13		$b_{14} = a_{11} * a_{14} + a_{21} * a_{24} + a_{31} * a_{34}$	b4	output_rdy=1
14		$b_{22} = a_{12} * a_{12} + a_{22} * a_{22} + a_{32} * a_{32}$	b5	b_{11}
15		$b_{23} = a_{12} * a_{13} + a_{22} * a_{23} + a_{32} * a_{33}$	b6	b_{12}
16		$b_{24} = a_{12} * a_{14} + a_{22} * a_{24} + a_{32} * a_{34}$	b7	b_{13}
17		$b_{33} = a_{13} * a_{13} + a_{23} * a_{23} + a_{33} * a_{33}$	b8	b_{14}
18		$b_{34} = a_{13} * a_{14} + a_{23} * a_{24} + a_{33} * a_{34}$	b9	b_{12}
19	clear	$b_{44} = a_{14} * a_{14} + a_{24} * a_{24} + a_{34} * a_{34}$	b10	b_{22}
20	a11			b_{23}
21	a21			b_{24}
22	a31			b_{13}
23	a12			b_{23}
24	a22			b_{33}
25	a32			b_{34}
26	a13			b_{14}
27	a23			b_{24}
28	a33			b_{34}
29	a14			b_{44}

DATA PATH

Once the Scheduling Table for the design was complete I had to start working on the designing a Data Path for my concept to work and shown below is the Data Path Diagram for the above concept.

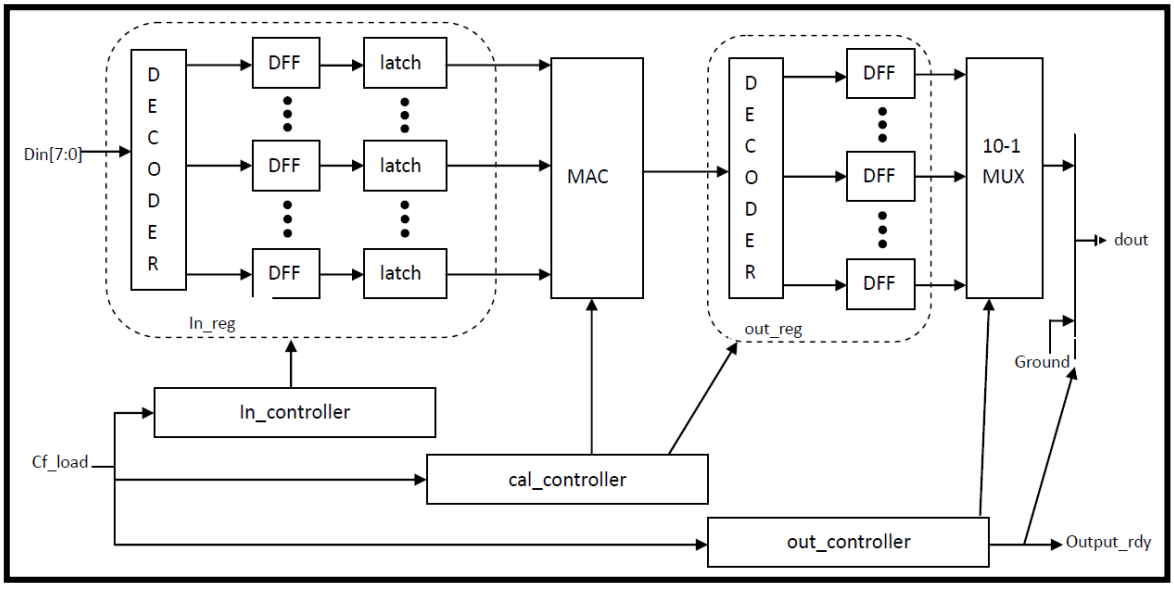

FIG. 1 : DATA PATH DIAGRAM

DESIGN OF MAC AND MAC CONTROLLER

MAC SHEMATIC FOR MATRIX MULTIPLICATION

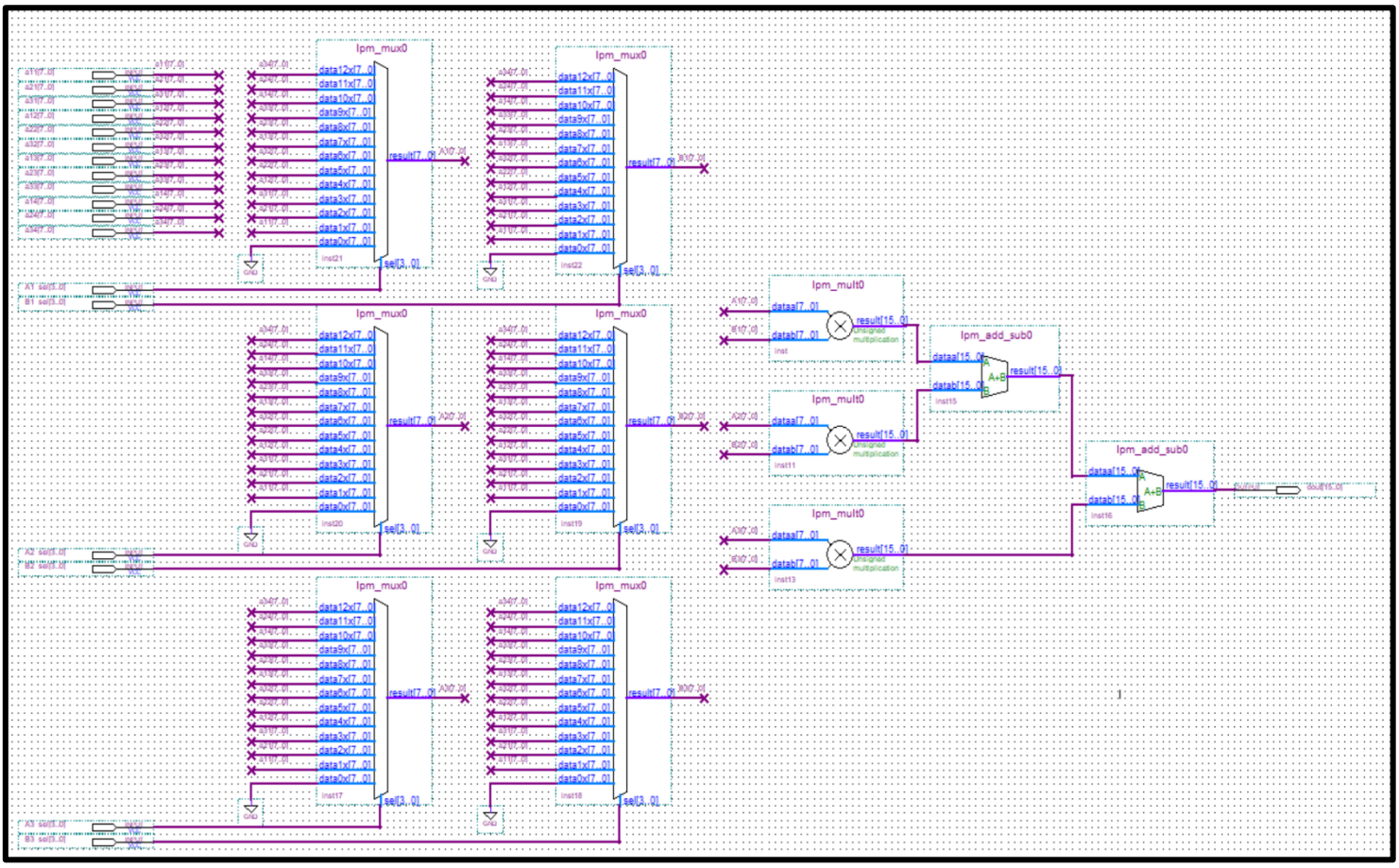

FIG. 2 : MAC UNIT SCHEMATIC

The MAC unit is designed to perform the arithmetic operations to the inputs provided by the input register and give the output to the output register.

MAC CONTROLLER VERILOG

The design for the MAC is stored in the MAC_CONTROLLER. V file in the folder along with its block diagram.

```
module MAC_CONTROLLER(reset, CLK, cf_load, a1, b1, a2, b2, a3, b3, sclr, output_select);
 input   cf_load, reset, CLK;
 output  reg [3:0] a1,b1;
 output  reg [3:0] a2,b2;
 output  reg [3:0] a3,b3;
 output  sclr;
 output  reg [3:0] output_select;         //10 output value
 reg         [4:0]pstate;
 reg         [4:0]nstate;
 parameter s0  = 5'b00000,          s1  = 5'b00001,          s2  = 5'b00010,          s3  = 5'b00011,
           s4  = 5'b00100,          s5  = 5'b00101,          s6  = 5'b00110,          s7  = 5'b00111,
           s8  = 5'b01000,          s9  = 5'b01001,          s10 = 5'b01010,          s11 = 5'b01011,
           s12 = 5'b01100,          s13 = 5'b01101,          s14 = 5'b01110,          s15 = 5'b01111,
           s16 = 5'b10000,          s17 = 5'b10001,          s18 = 5'b10010,          s19 = 5'b10011,
           s20 = 5'b10100,          s21 = 5'b10101,          s22 = 5'b10110;

 always@(posedge CLK or posedge reset )
 if (reset == 1'b1)
 pstate <= s0;
 else
 pstate <= nstate;

 always@(pstate or cf_load)
 begin
 a1 <= 4'b0;
 a2 <= 4'b0;
 a3 <= 4'b0;
 b1 <= 4'b0;
 b2 <= 4'b0;
 b3 <= 4'b0;
 output_select [3:0]    <= 4'b0;
```

FIG. 3 : MAC CONTROLLER CODE

DESIGN OF INPUT REGISTERS AND INPUT CONTROLLER

INPUT REGISTER SCHEMATIC

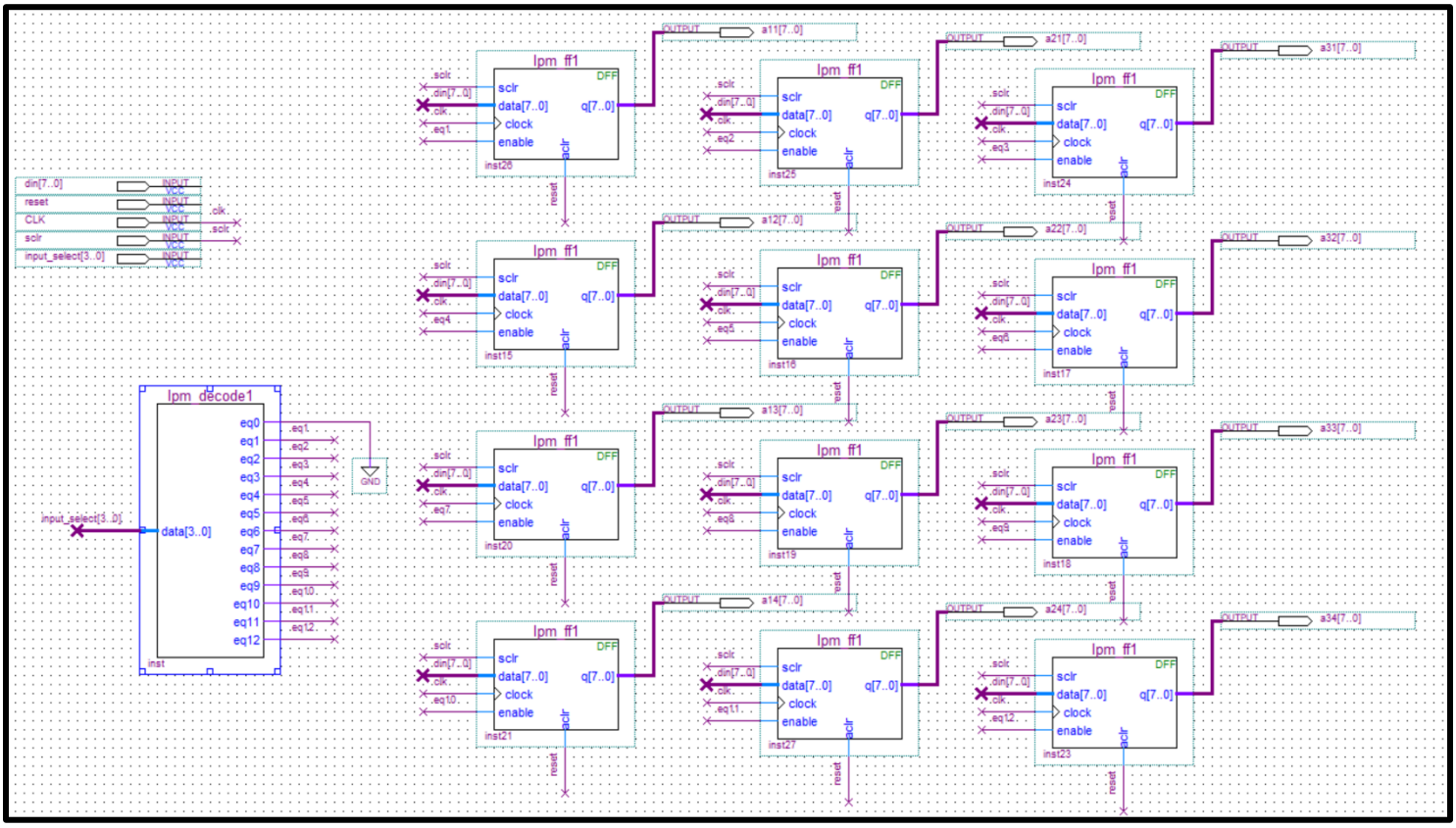

FIG. 4 : INPUT REGISTER SCHEMATIC

All the data coming from the source goes into goes to the Input Registers which are made up of twelve 8-bit D-FF. The decoder is used to select the D-FF in which we store coefficient of the input matrix.

INPUT CONTROLLER VERILOG

```
module INPUT_CONTROLLER(reset, clk, cf_load, clr, gate, input_sel);
 input  cf_load, reset, clk;                    //input
 output clr;
 output reg [3:0] gate;                        //12 input reg
 output reg input_sel;
 reg    [4:0] pstate, nstate;

 parameter s0  = 5'b00000,      s1  = 5'b00001,      s2  = 5'b00010,      s3  = 5'b00011,
           s4  = 5'b00100,      s5  = 5'b00101,      s6  = 5'b00110,      s7  = 5'b00111,
           s8  = 5'b01000,      s9  = 5'b01001,      s10 = 5'b01010,      s11 = 5'b01011,
           s12 = 5'b01100,      s13 = 5'b01101,      s14 = 5'b01110,      s15 = 5'b01111,
           s16 = 5'b10000,      s17 = 5'b10001,      s18 = 5'b10010,      s19 = 5'b10011;
                                                        //20 state, including 1 initiation state
 always@(posedge clk or posedge reset)
begin
 if (reset == 1'b1)
 pstate <= s0;
 else
 pstate <= nstate;
 end

 always@(pstate or cf_load)                          //state transform
begin
 gate [3:0] <= 4'b0;
 input_sel <= 0;

case(pstate)

s0: begin
 if (cf_load == 1'b1)
```

FIG. 5 : INPUT CONTROLLER CODE

DESIGN OF OUTPUT CONTROLLER AND OUTPUT REGISTER

OUTPUT REGISTER SCHEMATIC

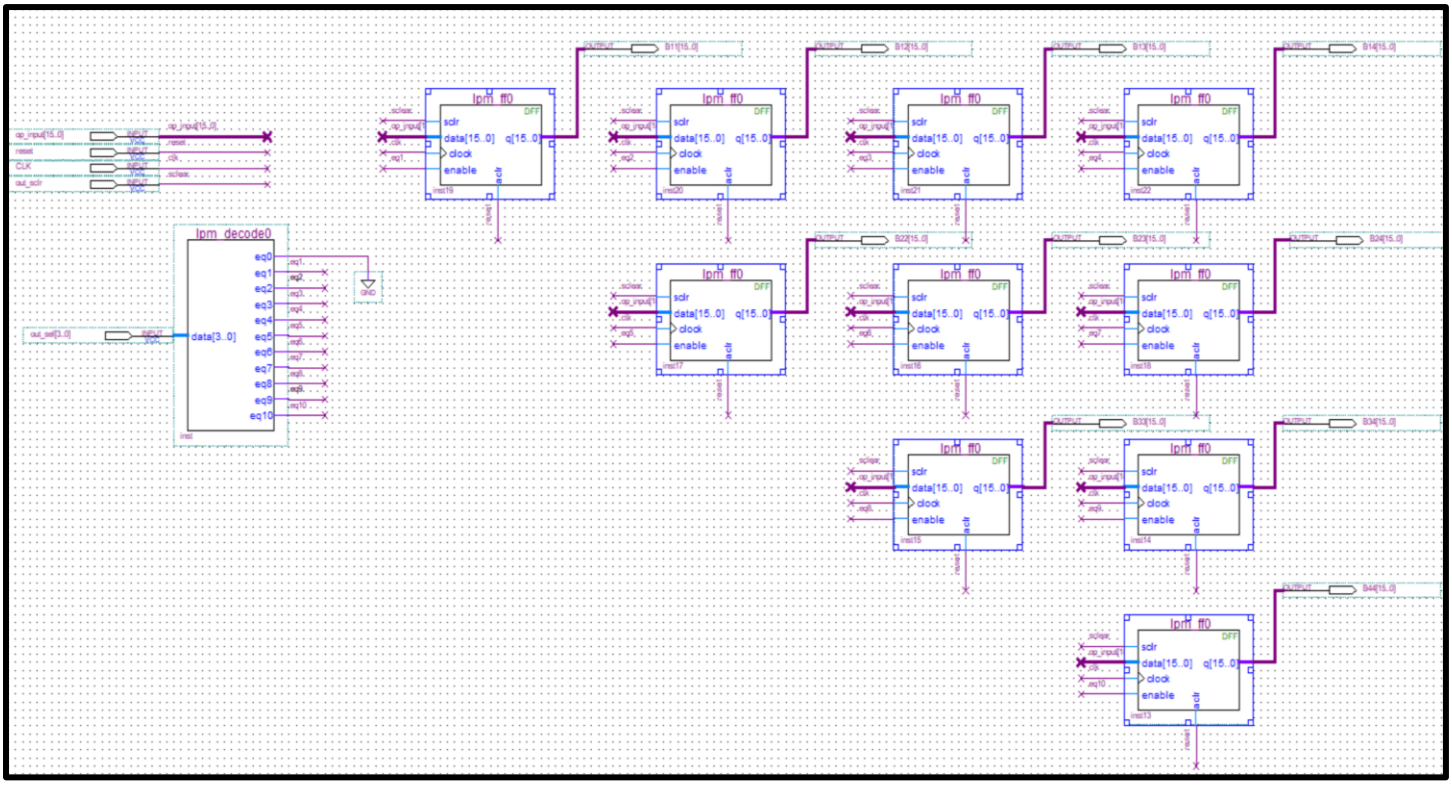

FIG. 6 : OUTPUT REGISTER SCHEMATIC

All the data coming from the MAC goes into goes to the Output Registers which are made up of ten 16-bit D-FF. The decoder is used to select the D-FF in which we store coefficient of the Output matrix.

OUTPUT CONTROLLER VERILOG

```
module OUTPUT_CONTROLLER(reset, CLK, cf_load, output_sel, output_rdy);

input   cf_load, reset, CLK;

/*control output*/
output  reg [3:0] output_sel;        //output_sel
output  reg output_rdy;

reg         [4:0]pstate;               //current state
reg         [4:0]nstate;               //next state

parameter s0  = 5'b00000,    s1  = 5'b00001,    s2  = 5'b00010,    s3  = 5'b00011,
          s4  = 5'b00100,    s5  = 5'b00101,    s6  = 5'b00110,    s7  = 5'b00111,
          s8  = 5'b01000,    s9  = 5'b01001,    s10 = 5'b01010,    s11 = 5'b01011,
          s12 = 5'b01100,    s13 = 5'b01101,    s14 = 5'b01110,    s15 = 5'b01111,
          s16 = 5'b10000,    s17 = 5'b10001,    s18 = 5'b10010,    s19 = 5'b10011,
          s20 = 5'b10100,    s21 = 5'b10101,    s22 = 5'b10110,    s23 = 5'b10111,
          s24 = 5'b11000,    s25 = 5'b11001,    s26 = 5'b11010,    s27 = 5'b11011,
          s28 = 5'b11100,    s29 = 5'b11101;

always@(negedge CLK or posedge reset )
if (reset == 1'b1)
pstate <= s0;
else
pstate <= nstate;

always@(pstate or cf_load)
begin
output_sel  [3:0]  <= 4'b0;
output_rdy         <= 0;
```

FIG. 7 : OUTPUT REGISTER SCHEMATIC

MATRIX MULTIPLIER DESIGN

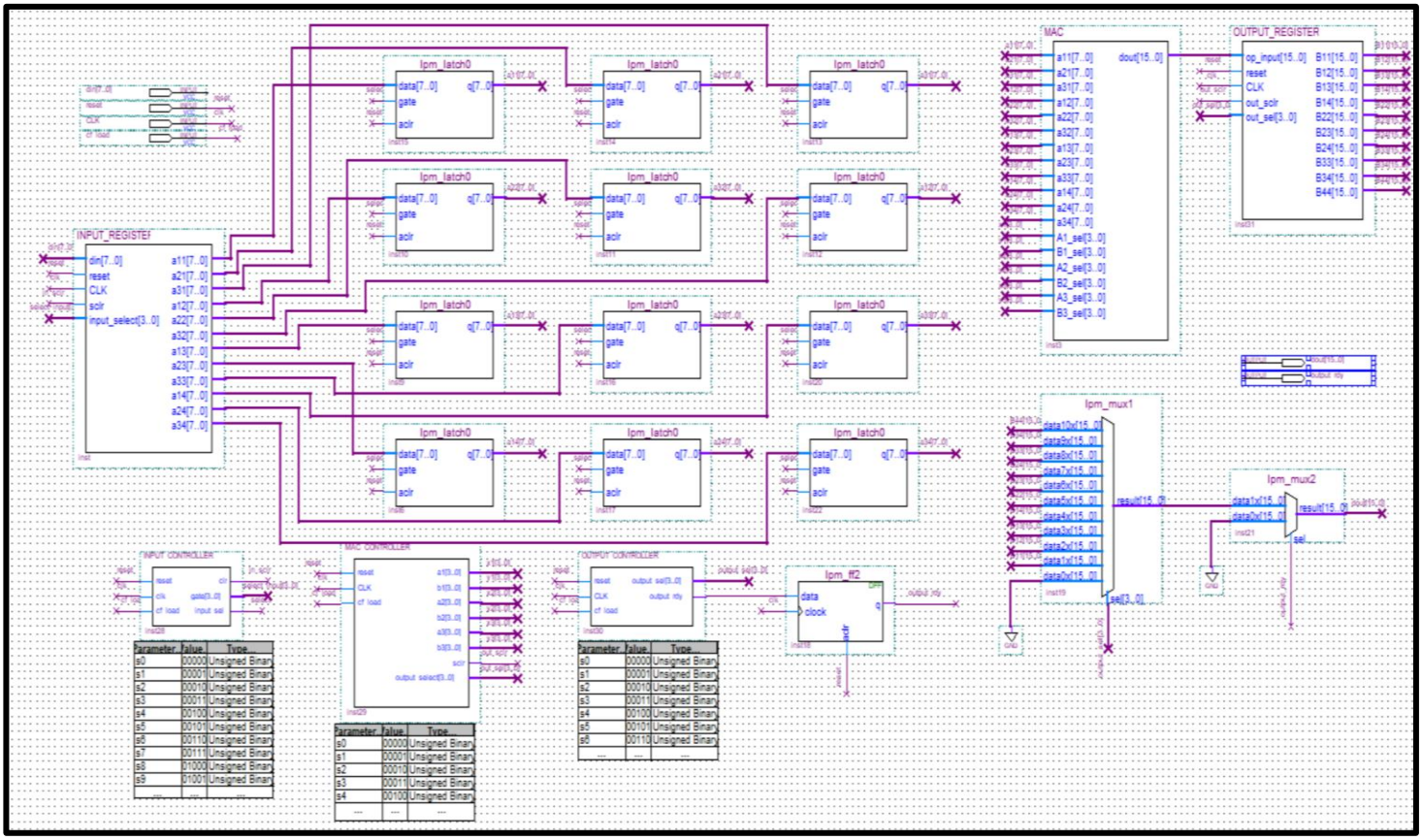

FIG. 8 : MATRIX MULTIPLICATION SCHEMATIC USING CUSTOM DESIGNED SYMBOLS

Functionality Claims

To perform functional test we have to compile and simulate a test bench waveform and match it to the functional requirements and the golden waveform respectively.

Functional requirements for the project are

1. **Design must fit within the Altera Cyclone EP1C3T100A8 and the timing analyzer must report at least 45 MHz for the register-to-register delay.**
2. **Design must fit within the Altera MAXII EPM1270F256A5 and the timing analyzer must report at least 25 MHz for the register-to-register delay.**

For Altera Cyclone EP1C3T100A8

SIMULATION REPORT

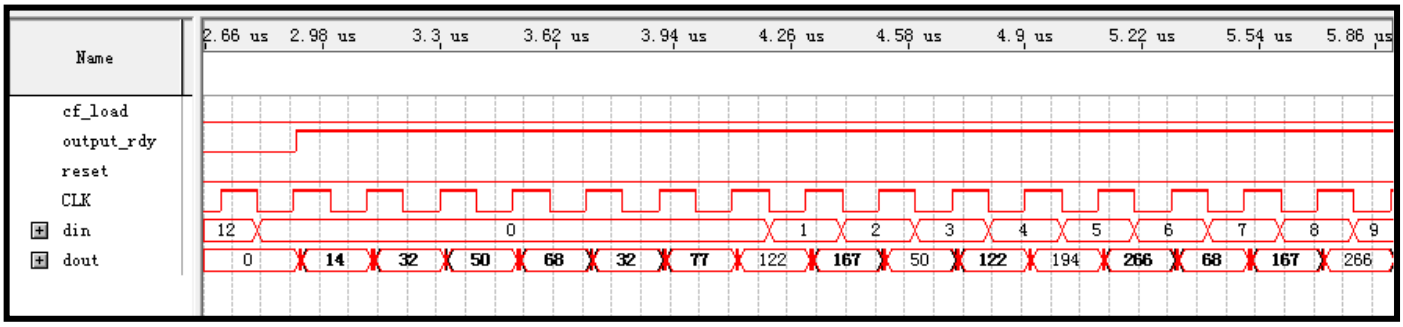

FIG. 9 : SIMULATION REPORT

REGISTER TO REGISTER DELAY REPORT

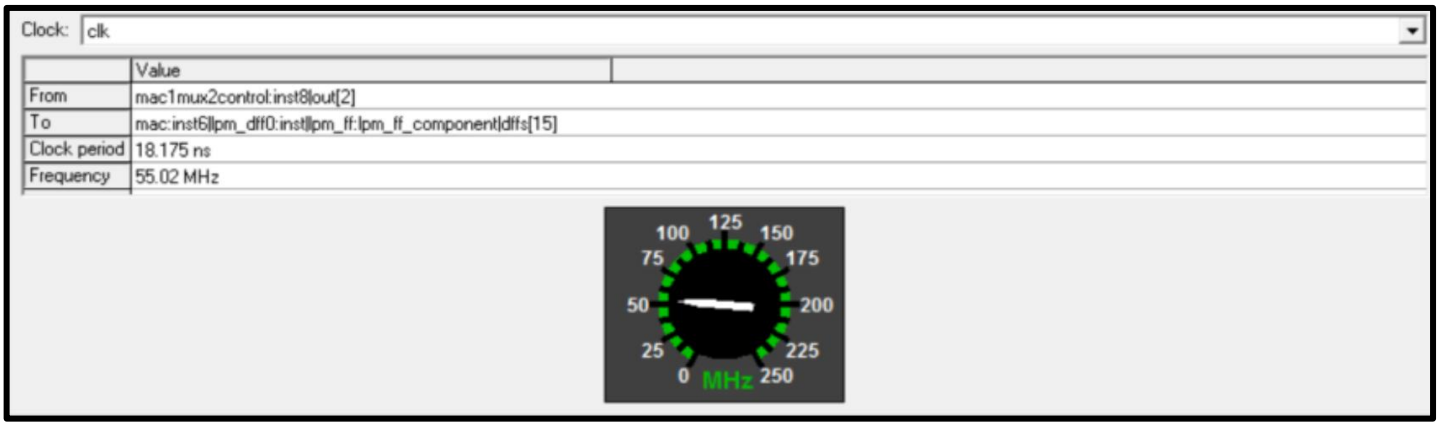

FIG. 10 : REGISTER TO REGISTER DELAY REPORT

For Altera MAXII EPM1270F256A5

SIMULATION REPORT

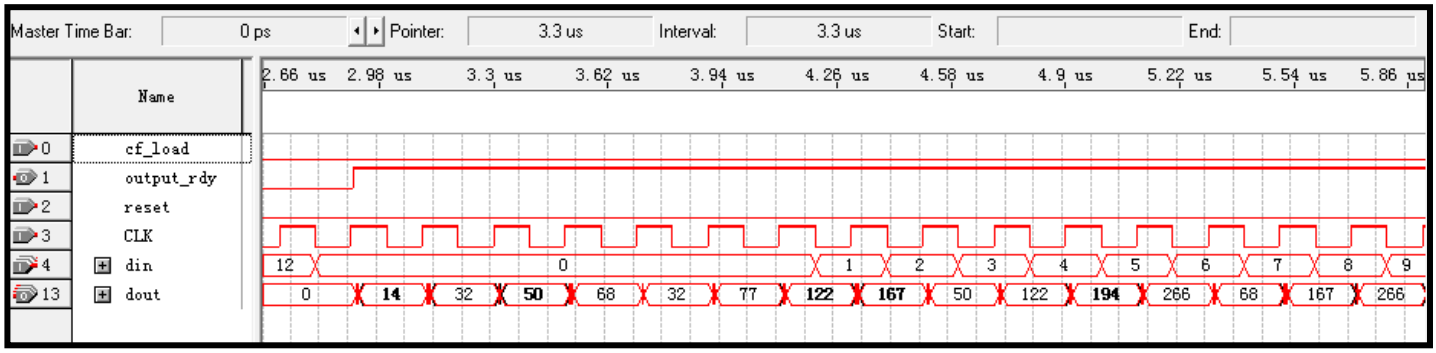

FIG. 11 : SIMULATION REPORT

REGISTER TO REGISTER DELAY REPORT

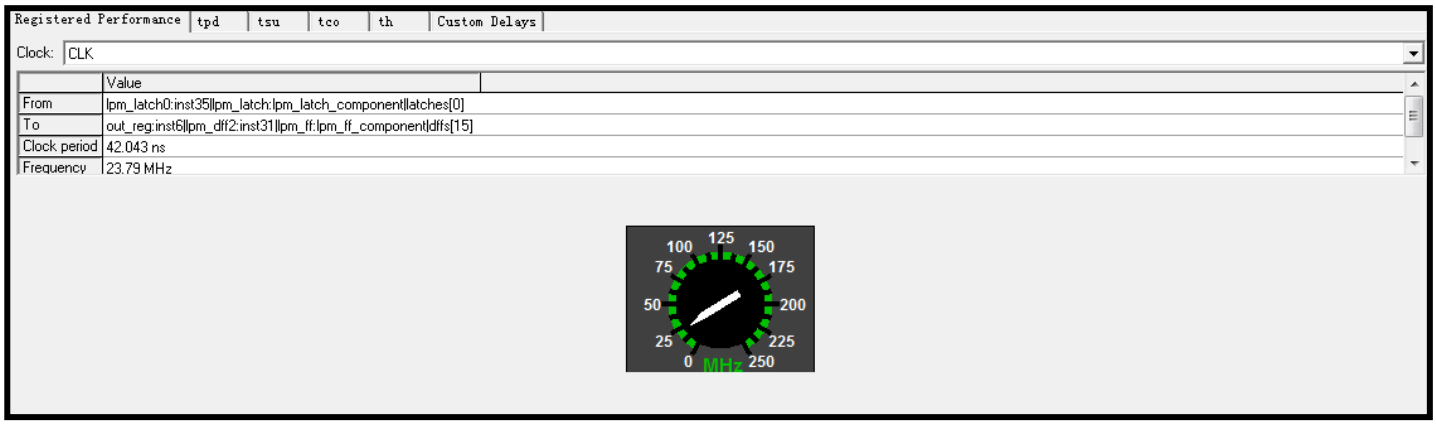

FIG. 12 : REGISTER TO REGISTER DELAY REPORT

ASM CHART

INPUT CONTROLLER

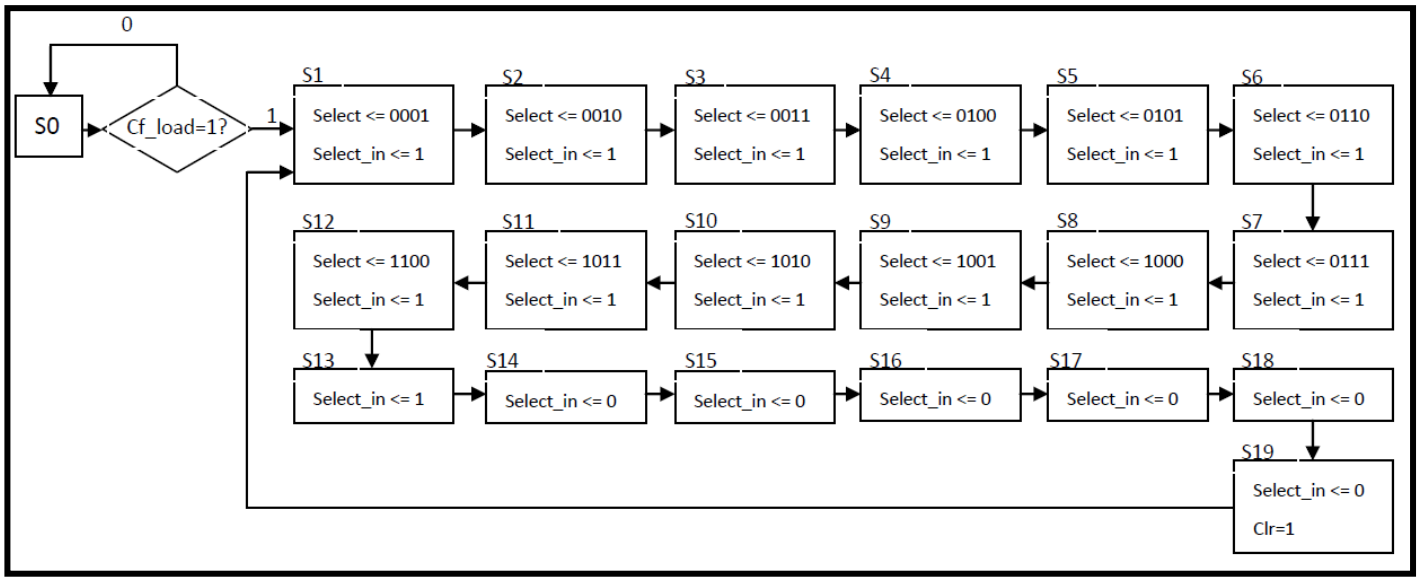

FIG. 13 : ASM CHART FOR INPUT CONTROLLER

MAC CONTROLLER

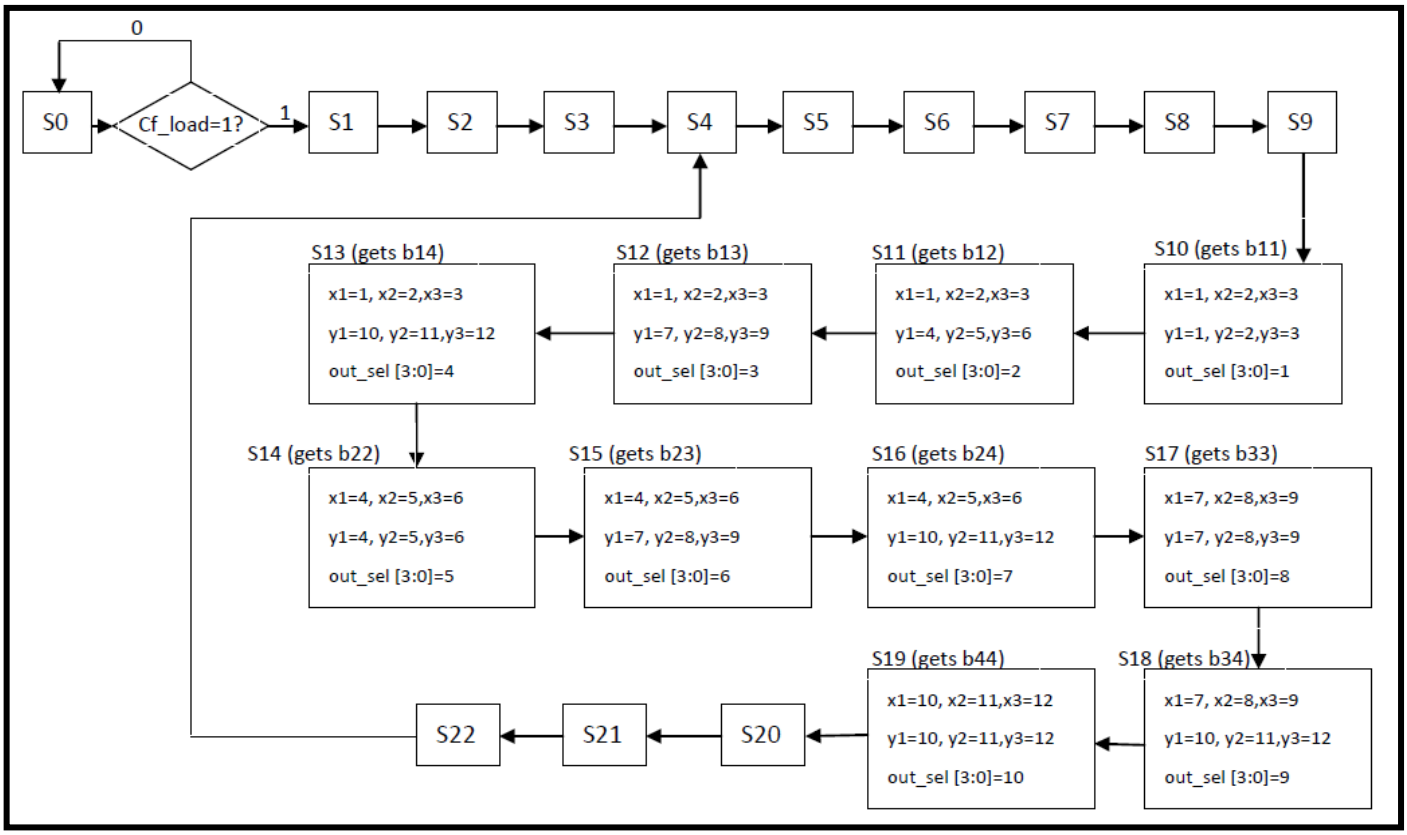

FIG. 13 : ASM CHART FOR MAC CONTROLLER

OUTPUT CONTROLLER

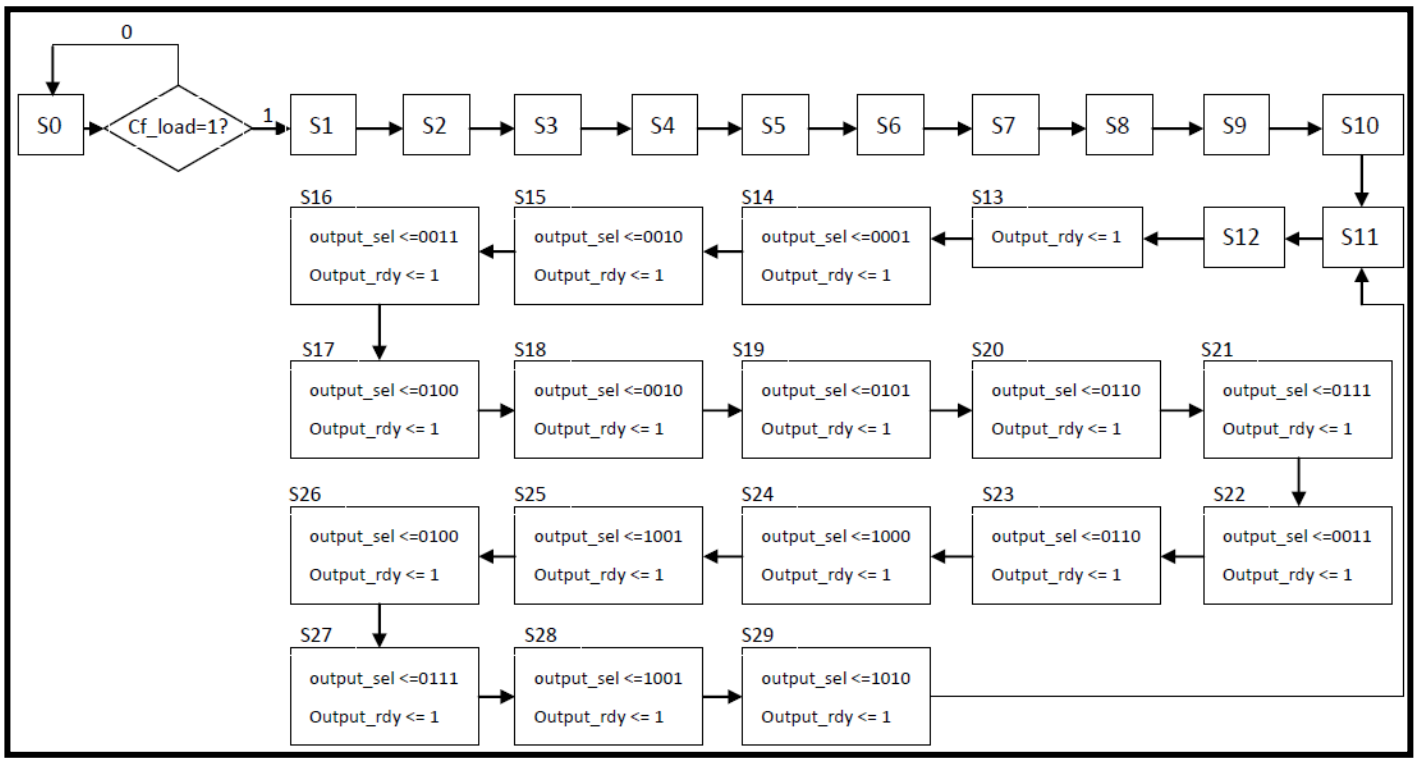

FIG. 13 : ASM CHART FOR OUTPUT CONTROLLER

PROBLEMS FACED

I have tried to solve this project through various ways and have been unsuccessful to get the project match the timing analysis provided in the questionnaire. I found this method of implementation of MAC much easier since it would be loading all data for a single output coefficient at the same time making it easier to make a state machine and clear the flipflops for the new set of inputs at the same time.

CONCLUSION

After detailed analysis and implementation of the project. I am successfully completed the project of performing Matrix Multiplication for a given set of test bench and met the functional requirements provided in the questionnaire.