COL 215 : Hardware Assignment 3

Tejas Anand, Kashish Goel

November 2022

1 Introduction

- In this assignment our objective was to implement Matrix Multiplication on the Basys3 FPGA Board.
- This assignment involves the design and integration of memories, registers and multiplier-accumulator components (MAC) in our design.

2 Problem Statement

Given two input matrices of size 128x128 and 8-bit unsigned integer, design a core in VHDL which perform matrix multiplication given input matrices.

3 Initial Thoughts...

- Entire Matrix Multiplication can be divided into separate vector by vector multiplications.
- We will design a Matrix Accumulator block, which will be driven by a clock that will read from the registers, compute the product and accumulate it to the existing sum.
- We will make 2 registers that will read from the ROM
- The timings of reading, computing and writing will be controlled by making states and changing those states via clocks. This will be our FSM.
- Our FSM will have 5 states namely, *read*, *compute*,*write*,*countreset* and *reset*. These states will be explained later.

4 Designing Individual Components

4.1 Register

A register has a clock input, bus-width, input data, write enable and output signal. output signal stores the value of the input when clock has a *rising_edge* and write enable is 1. We can read from the register when write enable is 0.

```
entity reg is
generic (bus_width : integer );
 Port (clk : in std_logic;
        din : in std_logic_vector(bus_width-1 downto 0);
        we : in std_logic;
        dout : out std_logic_vector(bus_width-1 downto 0) );
end reg;
architecture Behavioral of reg is
begin
process(clk)
begin
if rising_edge(clk) and we = '1' then
    dout<=din;
end if;
```

```
end process;
```

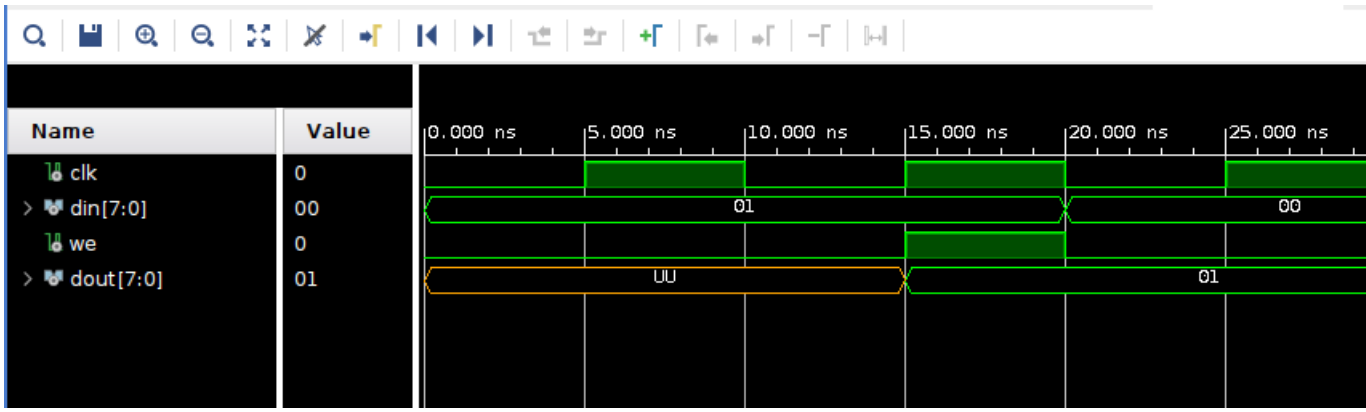


Figure 1: Register Test Simulation. We see that dout latches din when rising_edge occurs and we = 1. After we = 0 , dout is not affected by din

4.2 Matrix Accumulate Block (MAC)

- MAC has a clock input, 2 numbers of type *std_logic_vector* as input, a control signal, an enable signal and a sum output.
- Control signal acts as a reset, when it is 1 we reset the sum to 0.
- We Accumulate the product of the numbers only when we have a rising edge of the clock and enable signal is 1.

```
process(clk,en,cntrl)
begin
if cntrl = '1' then
    sum_sig<="0000000000000000";
elsif rising_edge(clk) and en = '1' then
    sum_sig<= std_logic_vector(UNSIGNED(sum_sig)+UNSIGNED(num1)*UNSIGNED(num2));
end if;
end process;
sum<=sum_sig;
end Behavioral;
```

Figure 2: A Test Simulation for MAC
We can see 4 (2*2) being accumulated at 5 ns , 24 (4*6) is accumulated at 15 ns when en = 1 and sum being reset when cntrl = 1

4.3 Read Only Memory (ROM)

- We synthesized ROM using the IP catalog.
- ROM has a clock signal as input, an address input and a data output.
- For Our use case , address is at max $128 * 128 = 16384$ which is equal to 2^{14}. So we require 14 bits in our address . We are storing 8-bit numbers so output would be 8 bits at max.

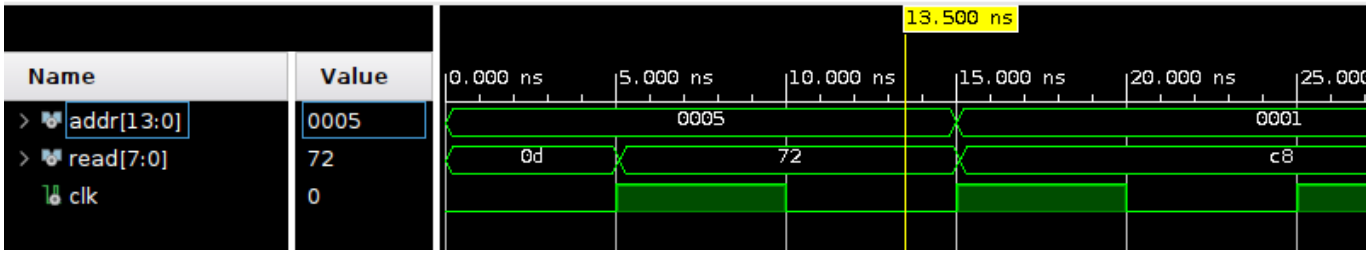


Figure 3: A Test Simulation for ROM
We see at 5 ns, read is showing data at 5th index, When address changes to 1, read changes to data at 1st index.

4.4 Random Access Memory

- We synthesized RAM using the IP catalog.
- It has along with clk, address as input also write enable and din signals.
- When write enable is 0 it acts as ROM, when write enable is 1 it writes into the memory.

5 Putting everything together and Designing the Finite State Machine

- In the FSM , we will store the state of our computation using three integers i, j, k
- Since the matrices are given in column major form, the index of a_{ij} is $128j + i$
- For fixed (i, j) , k is the index in the sum $\Sigma a_{ik} b_{kj}$
- We start from the read state , when $i = 0, k = 0$ and $j = 0$. We enable wreg which is the write register signal. Registers now have been written. In the next clock cycle we go to compute state for accumulation
- In compute state we increment k by 1 and set comp_enable to 1. If $k = 128$, this means all terms have been accumulated. We then go to write state. If $k < 127$, we again go to read state to read the next term.
- In write state we enable wram, ram write signal to 1 and disable the computation. From the write state we always go to the countereset state
- In the counteset state we reset k to 0 and increment i if $i < 127$ else we increment j. This basically means we are moving to the next pair of indices in the column major form of the resultant matrix.
- If in the countreset $i = 127$ and $j = 127$, this means the computation is complete and we move to the reset state. We stay in the countreset state for 2 clock cycles to let the RAM write by maintaining a wait count. If wait count is 0 we move to countreset state elsif waitcount is 1 we transition to Read State again for the next set of products. We set cntrl equal to 0 , to set the accumulated sum to 0.
- From, the reset state, we always stay in it. We now take the input from the board to display the entries of the output matrix by reading from the RAM.

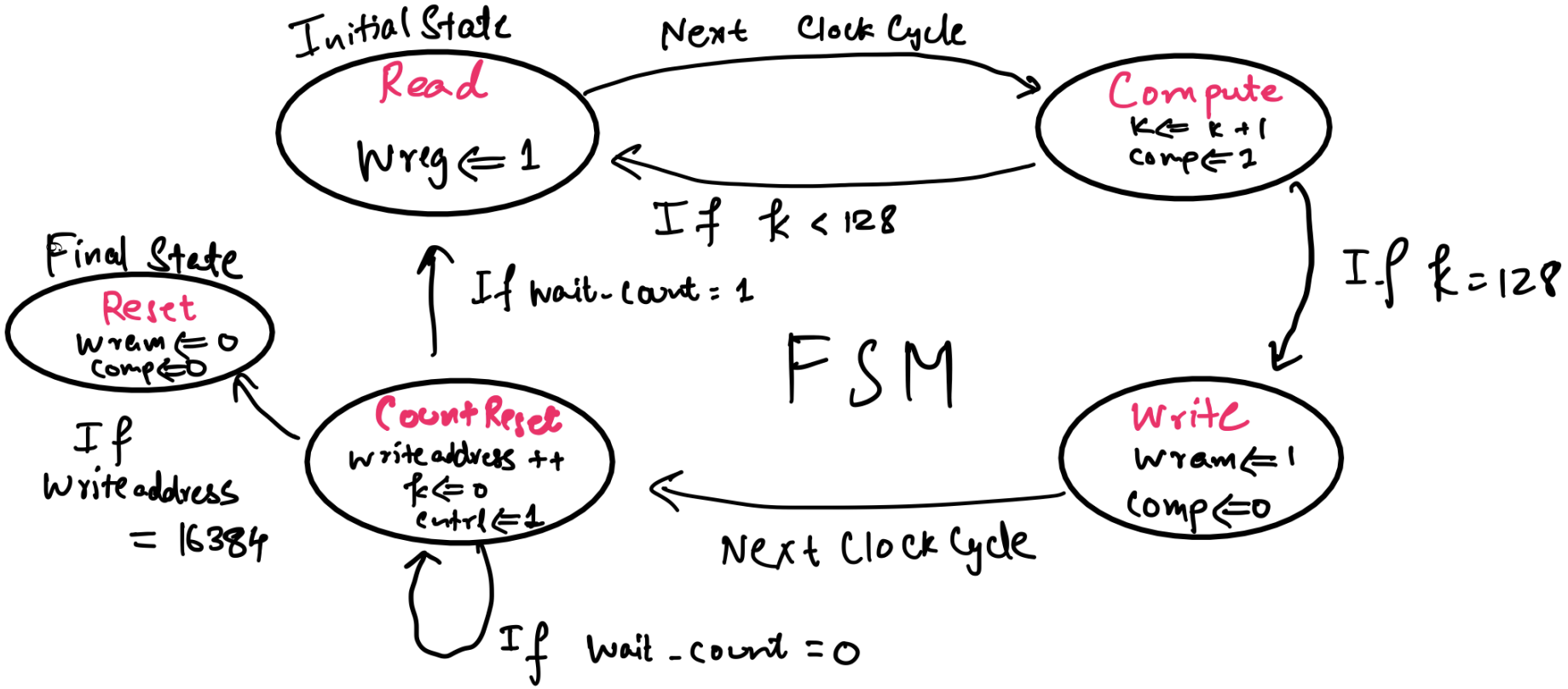
- We make a state transition each clock cycle, except in the countreset state, in which we wait for 2 clock cycles.

```
process(curstate)
begin
case curstate is
    when reset =>
        wram<='0'; -- write disabled
        comp_en<='0'; --compute disabled
        nextstate<=reset;
        write_address_int<=(to_integer(unsigned(add)));
        dig3(3)<=ram_out(15);
        dig3(2)<=ram_out(14);
        dig3(1)<=ram_out(13);
        dig3(0)<=ram_out(12);
        dig2(3)<=ram_out(11);
        dig2(3)<=ram_out(10);
        dig2(3)<=ram_out(9);
        dig2(3)<=ram_out(8);
        dig1(3)<=ram_out(7);
        dig1(3)<=ram_out(6);
        dig1(3)<=ram_out(5);
        dig1(3)<=ram_out(4);
        dig0(3)<=ram_out(3);
        dig0(3)<=ram_out(2);
        dig0(3)<=ram_out(1);
        dig0(3)<=ram_out(0);

    when read=>
        wreg<='1'; -- write register enabled . Now registers have been written, read from rom
        nextstate<=compute;


    when write=>
        comp_en<='0'; -- computation disabled
        wram<='1'; -- write ram enabled
        nextstate<=countreset;


    when compute =>
        wreg<='0'; -- register content wont be modified while computing
        k<=k+1;
        comp_en<='1';
        if (k<128) then
            nextstate<=read;
        elsif k = 128 then
            nextstate<= write;
        end if;

    when countreset=>
        k<=0;
        if write_address_int =16383 then
            nextstate<=reset;
        else
        write_address_int<=write_address_int+1;
        nextstate<=read;
--        if wait_count = 0 then
--            nextstate<=countreset;
--            wait_count<=1;
--        elsif wait_count =1 then
--            nextstate<=read;
```

```
--            wait_count<=0;
--            wram<='0';
--            write_address_int<=write_address_int+1;
--          end if;
          end if;
          cntrl<='1';
      end case;



end process;


end machine;
```

Write address = 128 j + i

i = write address % 128 j = write address // 128



6 Block Diagram

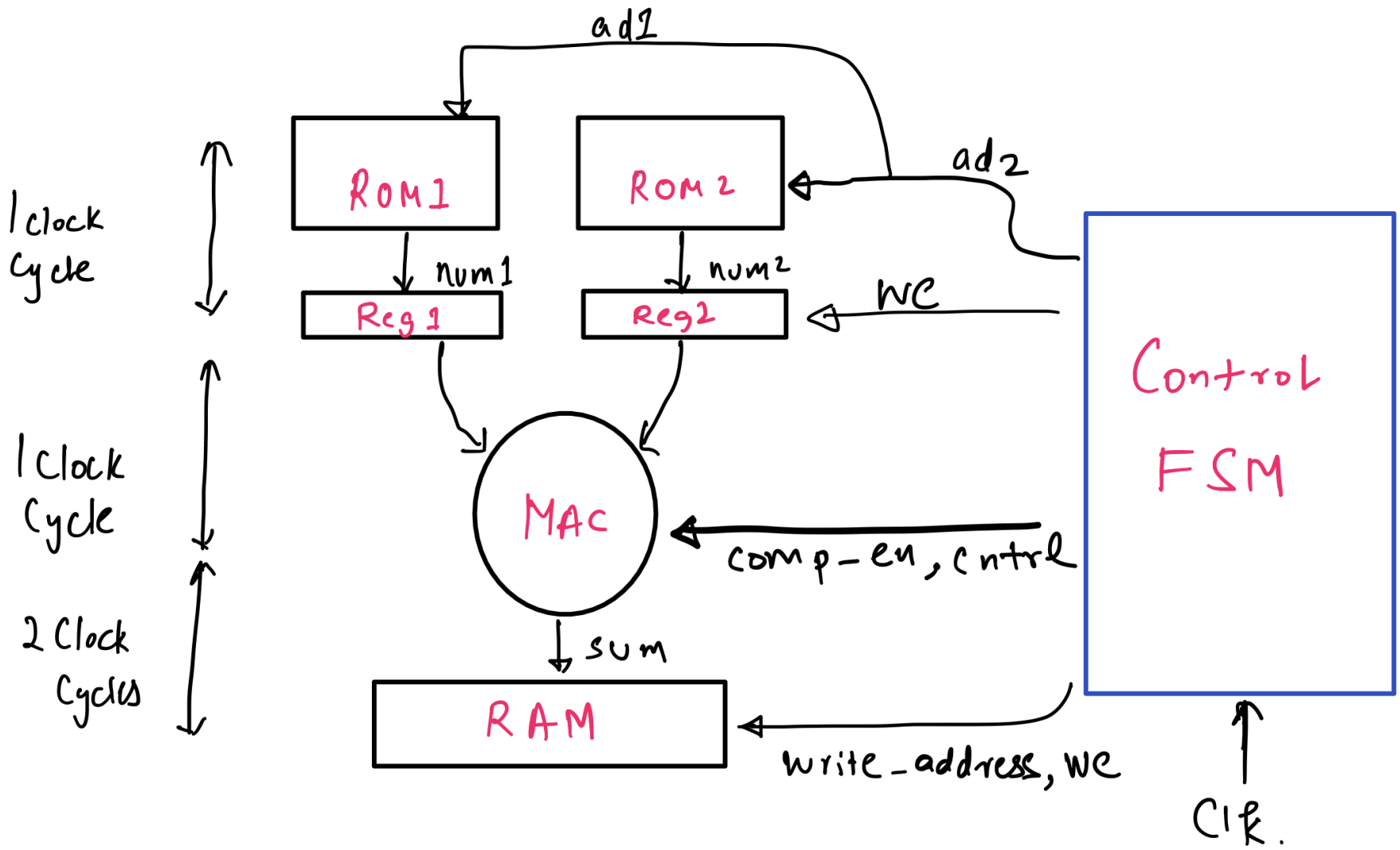

ad1
ad2
ROM1
ROM 2
num1
num2
Reg 1
Reg2
WE
Control
FSM
MAC
comp_en, cntrl
sum
RAM
write_address, WE
Clk.
1 clock
cycle
1 Clock
Cycle
2 Clock
Cycles

Figure 5:

7 Simulation Snapshots

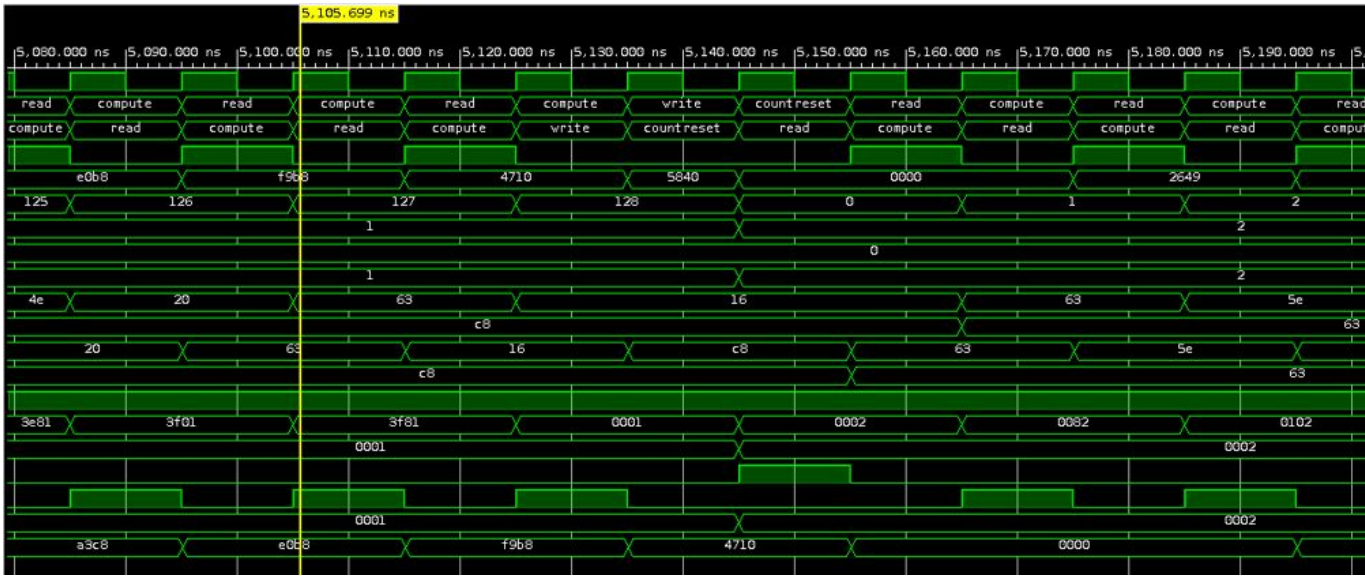


Figure 6: We can see that after $k = 128$ we go to countreset state, sum is reset to 0

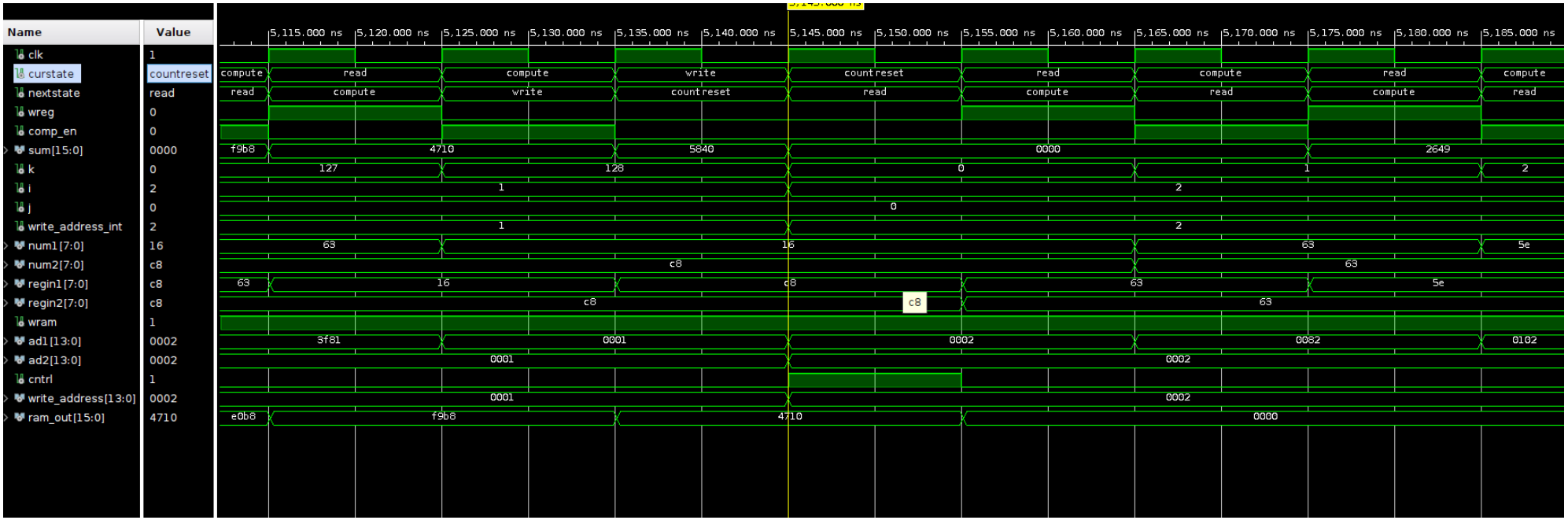

Name
Value
clk
1
curstate
countreset
nextstate
read
wreg
0
comp_en
0
sum[15:0]
0000
k
0
i
2
j
0
write_address_int
2
num1[7:0]
16
num2[7:0]
c8
regin1[7:0]
c8
regin2[7:0]
c8
wram
1
ad1[13:0]
0002
ad2[13:0]
0002
cntrl
1
write_address[13:0]
0002
ram_out[15:0]
4710
5,115.000 ns
5,120.000 ns
5,125.000 ns
5,130.000 ns
5,135.000 ns
5,140.000 ns
5,145.000 ns
5,150.000 ns
5,155.000 ns
5,160.000 ns
5,165.000 ns
5,170.000 ns
5,175.000 ns
5,180.000 ns
5,185.000 ns
compute
read
write
countreset
f9b8
4710
5840
0000
2649
127
128
0
1
2
63
16
5e
3f81
0001
0002
0082
0102
e0b8
0
c8

Figure 7: 10

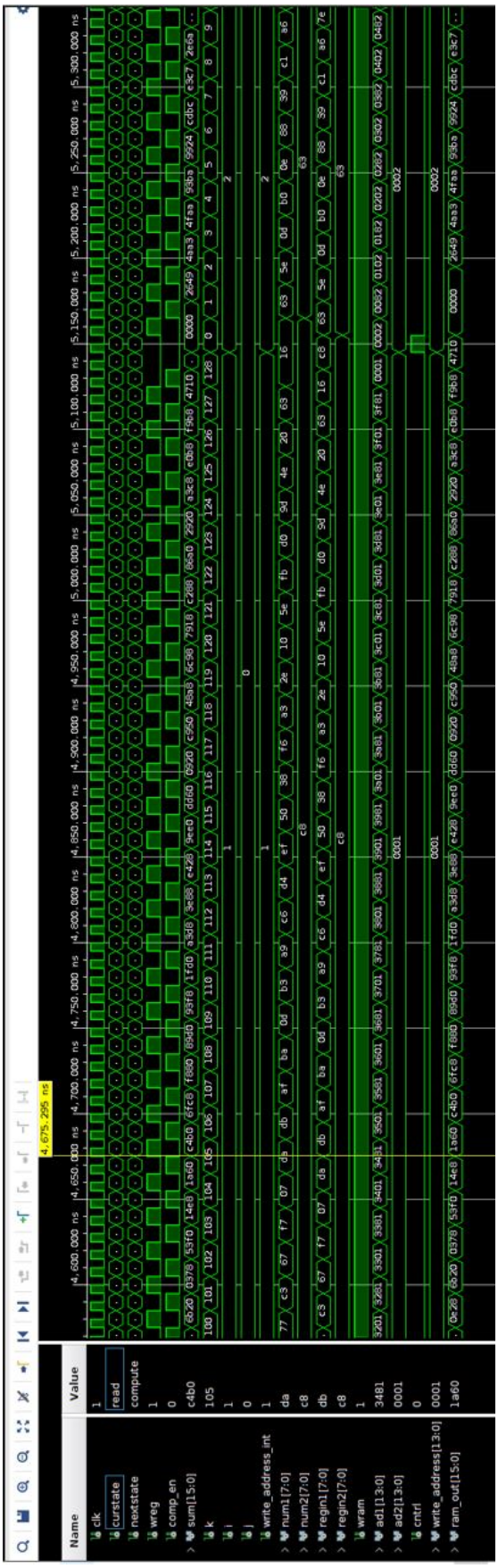

Name
Value
clk
curstate
nextstate
wreg
comp_en
sum[15:0]
k
i
j
write_address_int
num1[7:0]
num2[7:0]
regin1[7:0]
regin2[7:0]
wram
ad1[13:0]
ad2[13:0]
cntrl
write_address[13:0]
ram_out[15:0]
4,675,295 ns

Figure 8:

Resource	Utilization	Available	Utilization %
LUT	6813	20800	32.75
LUTRAM	1792	9600	18.67
FF	500	41600	1.20
IO	26	106	24.53

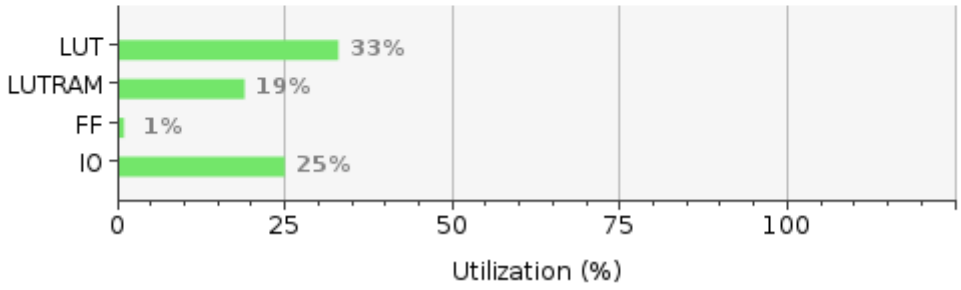


Figure 9: Resource Utilization

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.106 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (11.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

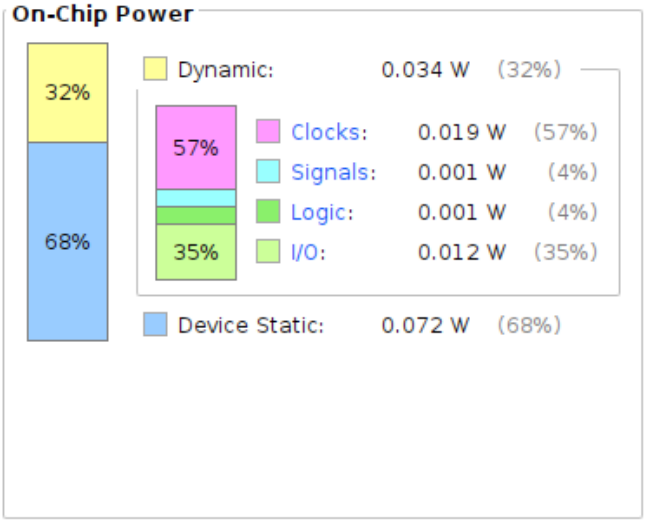


Figure 10: Power Utilization

8 Test Cases

We show the output on the board for some indices.

Figure 11: 0th index for M^2, where M is the sample matrix given

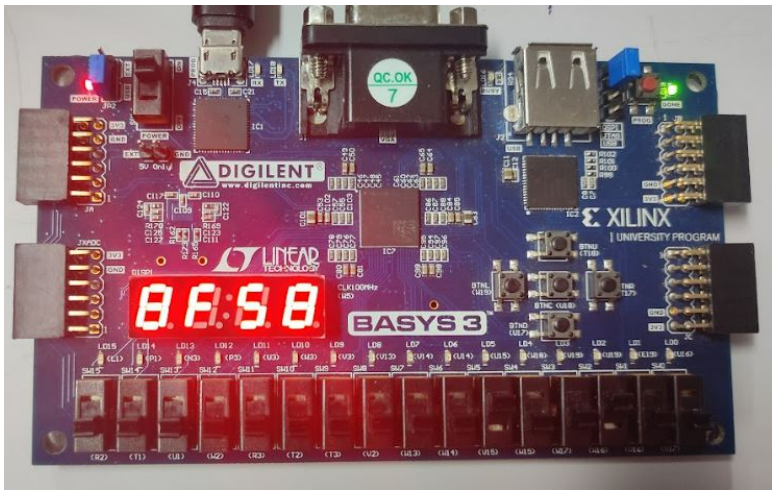


Figure 12:

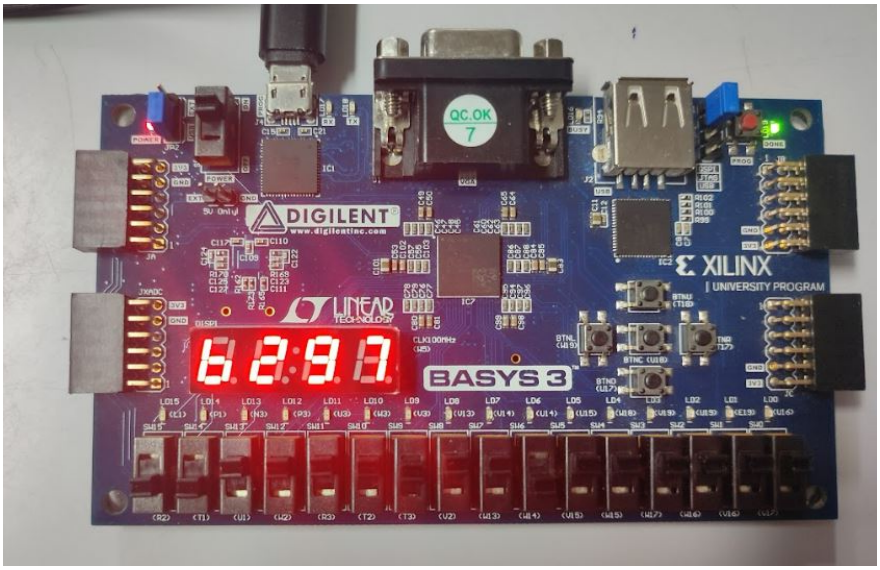

QC.OK
7
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3

Figure 13: