VLSI LAB Digital Assignment 8

Submitted by:

Swarup Tripathy — 19BEE0167

School of Electrical Engineering

Faculty: **Professor Balamurugan S**

Course: **EEE-4028**

Course Name: **VLSI Lab**

Lab Slot: **L43 + L44**

This work is submitted in partial fulfilment of the requirement of the award of the degree of Bachelor of Technology in EEE

April 21, 2022

PIPELINED MAC DESIGN

Objectives

1. To provide students with the background needed to design, develop, and test digital arithmetic circuits using IEEE standard Verilog HDL.
2. To provide an understanding complex arithmetic circuit design principles and its architecture design.

Outcomes

1. After completion of this course the students will be familiar with design and implementation of Digital Arithmetic building blocks using Verilog HDL and Modelsim Software.

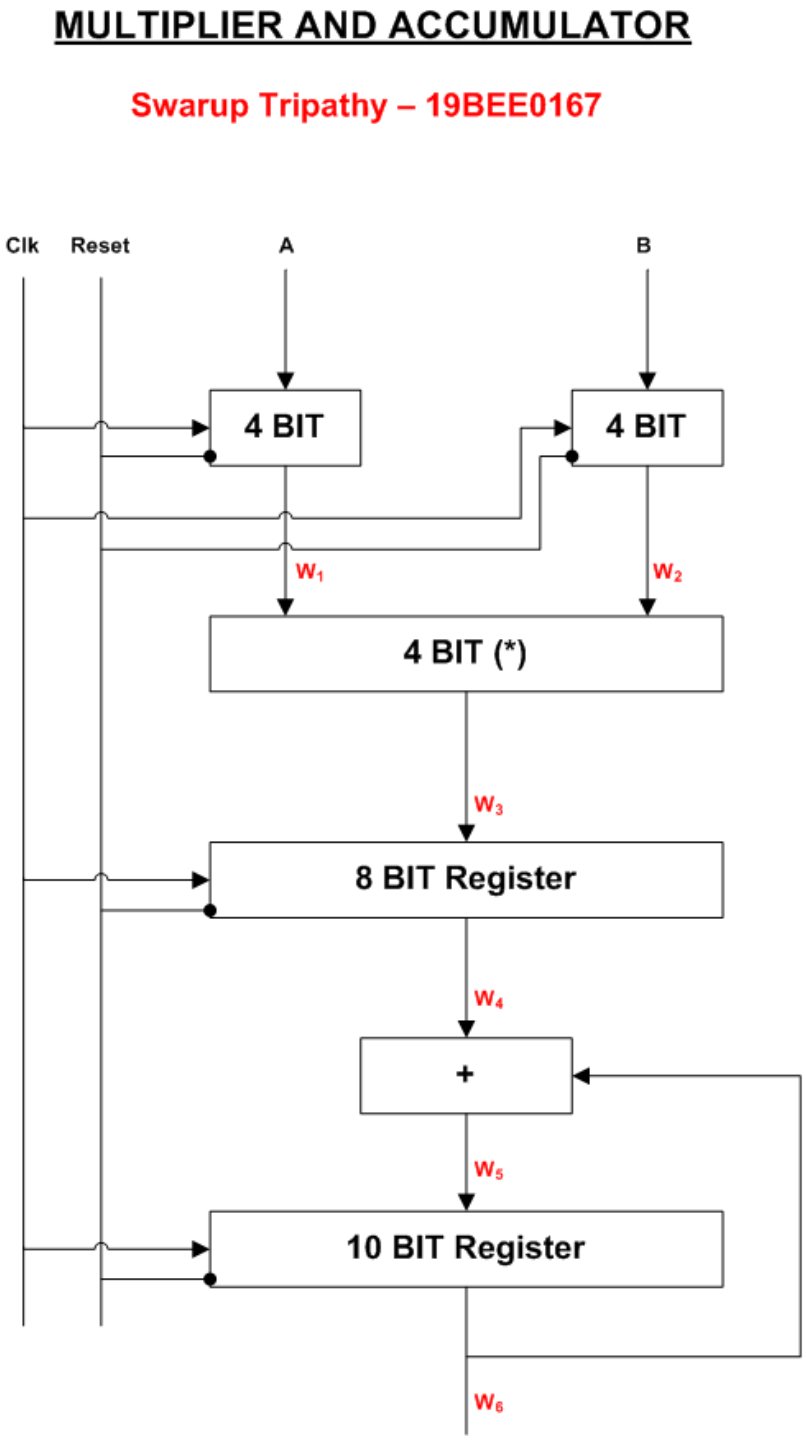
AIM

1. Design a pipelined mac design using 4bit, 8bit and 10 bit pipeline.

REQUIRED SOFTWARE

1. Model sim software for simulation
2. Microsoft Visio for making flowchart
3. Documentation to be done using LaTeX

CIRCUIT DIAGRAM



Design Code

1. **Verilog Module** — designcode.v

```
module pipeline_MAC(input clk, rst, input[3:0] a,b, output[9:0]y);
wire[3:0]w1,w2;
wire[7:0]w3,w4;
wire[9:0]w5,w6;
pipo_4b p1(clk, rst, a, w1);
pipo_4b p2(clk, rst, b, w2);
assign w3=w1*w2;
pipo_8b p3(clk, rst, w3, w4);
assign w5=w4+w6;
pipo_10b p4(clk, rst, w5, w6);
assign y=w6;
endmodule
```

2. **pipo8bit.v**

```
module pipo_8b(input clk, rst, input[7:0]d, output reg[7:0]q);
always@(posedge clk, negedge rst)
begin
if(!rst)
q<='b0;
else
q<=d;
end
endmodule
```

3. **pipo10bit.v**

```
module pipo_10b(input clk, rst, input[9:0]d, output reg[9:0]q);
always@(posedge clk, negedge rst)
begin
if(!rst)
q<='b0;
else
q<=d;
end
endmodule
```

4. **pipo4bit.v**

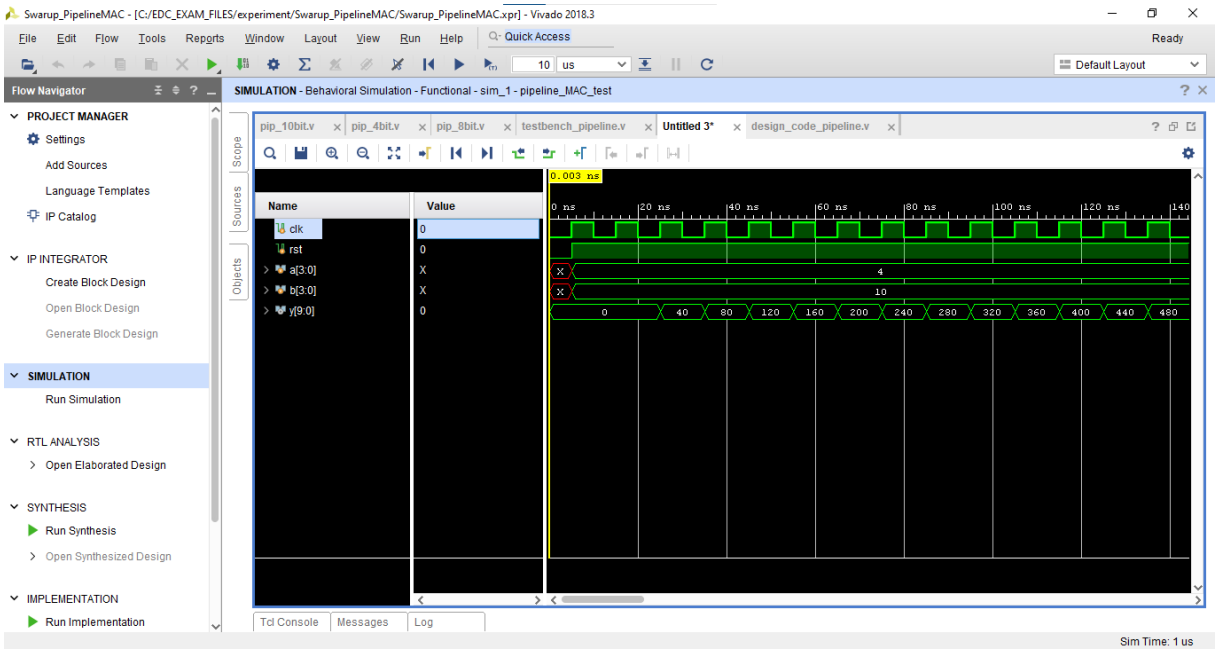
```
module pipo_4b(input clk, rst, input[3:0]d, output reg[3:0]q);
always@(posedge clk, negedge rst)
```

```
begin
if(!rst)
q<='b0;
else
q<=d;
end
endmodule
```

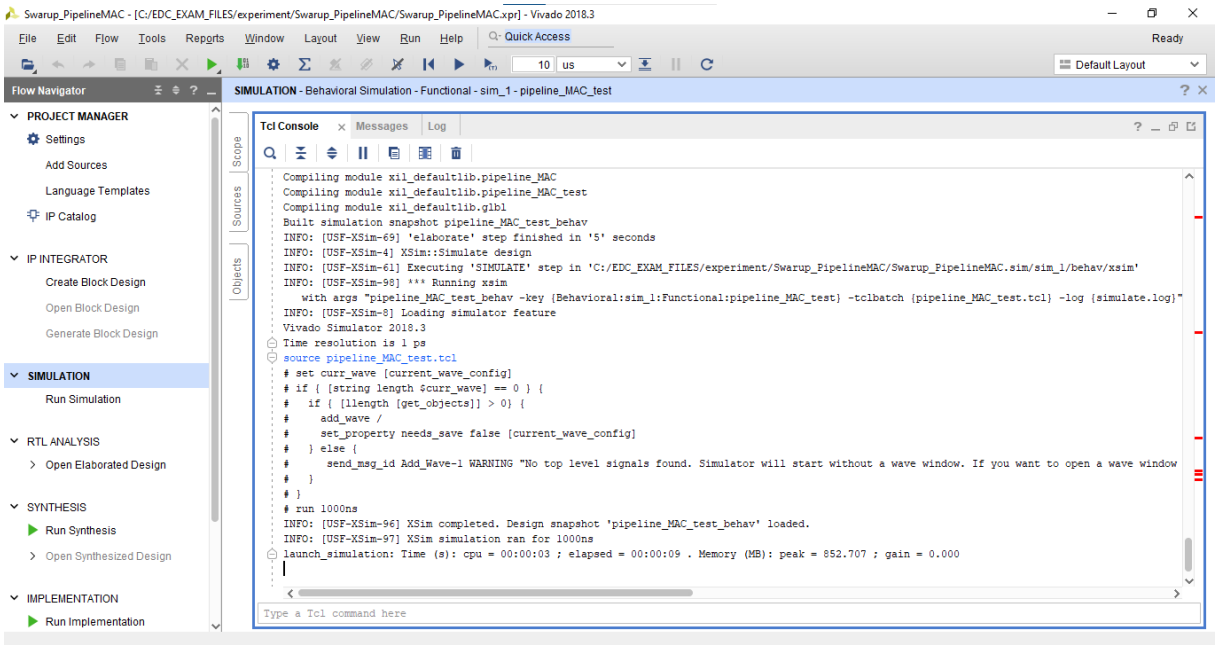
5. **Test Fixture** — Test.v

```
module pipeline_MAC_test();
reg clk,rst;
reg[3:0]a,b;
wire[9:0]y;
pipeline_MAC UUT(clk, rst, a, b, y);
initial begin
rst=1'b0;
clk=1'b0;
#5;
rst=1'b1;
a=4'b0100;
b=4'b1010;
#10;
end
always #5 clk=~clk;
```

6. **Output**



7. Console Output



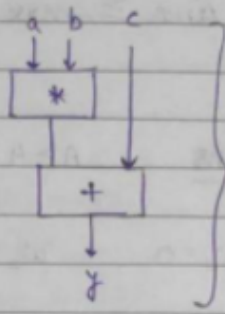
Result

1. Successfully the pipeline mac multiplier has been designed and the output was verified.

(8)

Multiplier and Accumulator

$y = ab + c$



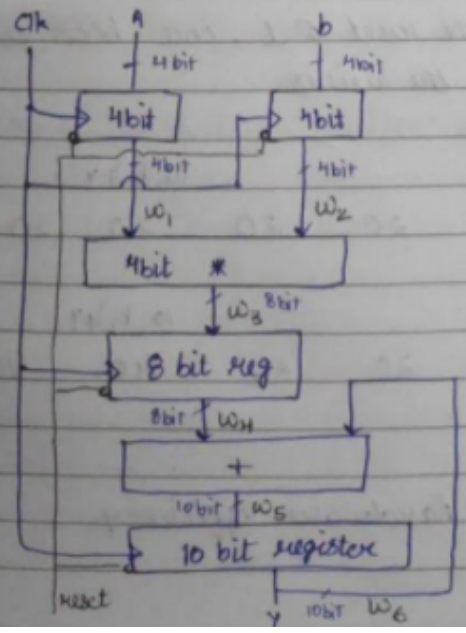
- To introduce a pipeline
- whenever performing addition we don't accept the input of a & b
- multiple execution are implemented in an overlapped manner

A = 4 b = 5

@50ns 4 × 5 = 20

when pipelined

- 3 bit register min required for i/p a and b



- growing size register depends on the resistor
- 4bit multiplier results in 8 bit.
- Whenever using a resistor, we require a reset signal for each resistor

clock → To trigger a signal

clock time ≥ max of multiplier, Adder

@ 0ns → A = 4 ; B = 5

reset=0	$w_1=0$	$w_2=0$	$w_3=0$	$w_4=0$	$w_5=0$	$w_6=0$

0 → represents a reset signal

↓

needs to be 0

reset = 1, ↑	$w_1=4$	$w_2=5$	$w_3=20$	$w_4=0$
	$w_5=0$	$w_6=0$		

+ve pulse	a_0	b_0		
↑	$w_1=4$	$w_2=5$	$w_3=20$	$w_4=20$
	$w_5=20$	$w_6=0$		
	$(a_0 \cdot b_0)+Y_0$			

we are keeping the reset @ 1, once becomes 0 will clear all the wires.

				$(a_1 b_1)+Y$	
↑	4	5	20	20	20
				$(a_2 b_2)+Y$	
↑	4	5	20	40	40

- maximising the hardware efficiency

Parallel in Parallel out

accumulation → previous value

CODE:

d → i/p
q → o/p

```
module pp_4bit (input clk, reset,
                input [3:0] d,
                output reg [3:0] q);

always @ (posedge clk, negedge reset)

if (!reset)   // reset = 1'b0
   q <= 4'b0000;

else
   q <= d;

end
endmodule
```

= Blocking
<= non blocking

Transistor Sizing

- sizing of transistors to balance performance

```
module pipelined_MAC (input clk, reset, input [3:0] a, b,
                      output [9:0] y)

wire [3:0] w1, w2;
wire [7:0] w3, w4;
wire [9:0] w5, w6;

pip_4bit p1 (clk, reset, a, w1);
pip_4bit p2 (clk, reset, b, w2);

assign w3 = w1 * w2;

pip_8bit p3 (clk, reset, w3, w4);

assign w5 = w4 + w6;

pip_10bit (clk, reset, w5, w6);
assign y = w[6];
endmodule
```

TESTBENCH

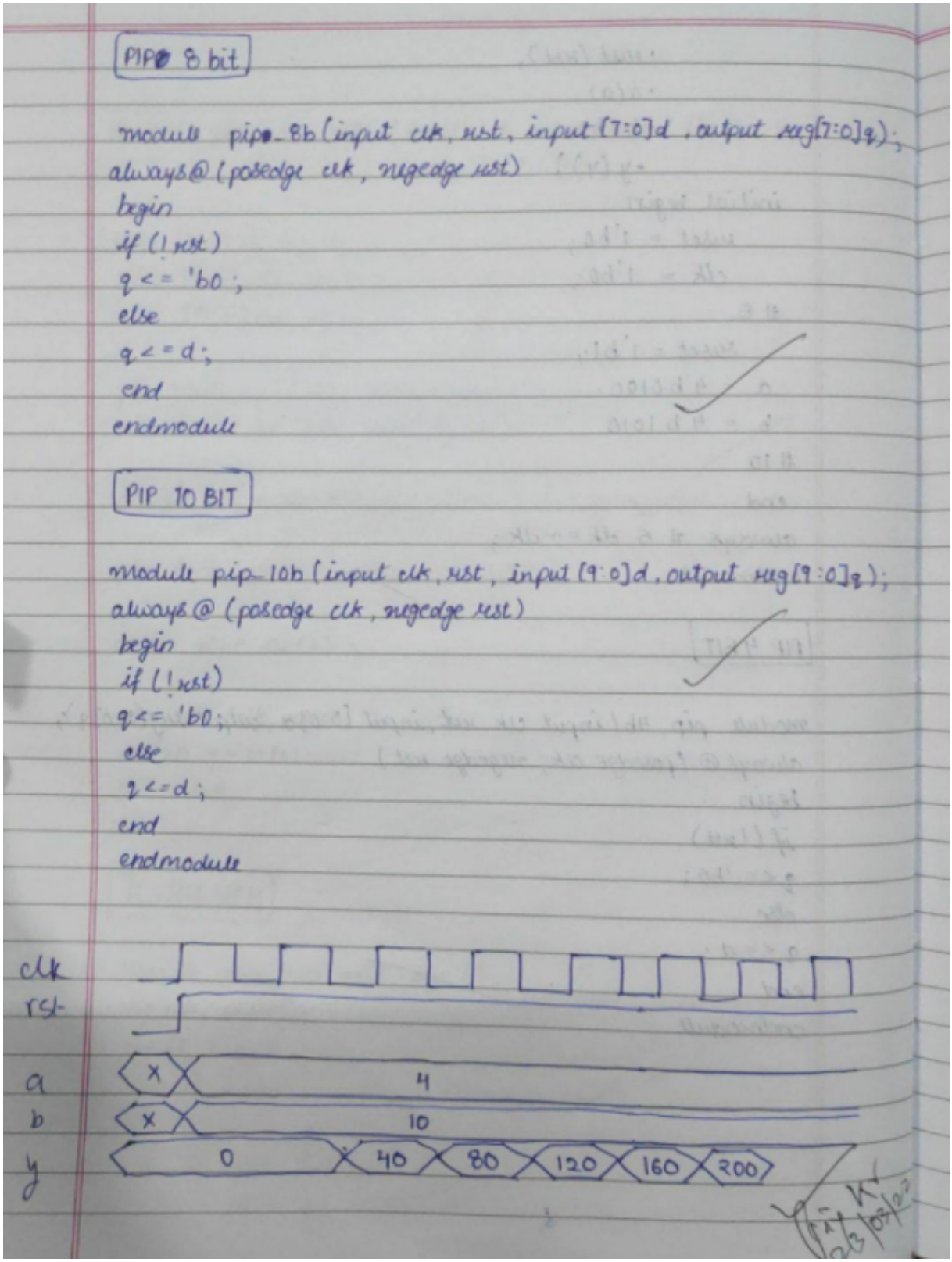
```
module pipelined_mac_test ();
reg clk, reset;
reg [3:0] a, b;
wire [9:0] y;

pipelined_MAC UUT (
                .clk (clk),
```

```
        .rust (rst),
        .a(a),
        .b(b),
        .y(y))
initial begin
    ruset = 1'b0;
    clk = 1'b0;
#5
    ruset = 1'b1.,
 a = 4'b0100.
 b = 4'b1010
#10
 end
always #5 clk = ~clk;
endmodule
```

PIP 4 BIT

```
module pip_4b(input clk, rst, input [3:0] d, output reg [3:0] q);
always @ (posedge clk, negedge rst)
begin
 if (!rst)
 q <= 'b0;
 else
 q <= d;
 end
endmodule
```



Inference

1. In this experiment learnt about how to construct pipelined MAC design.