Report for ESE 507 Project 1

By

Aishwarya Gandhi (111424452)

&

Kewal Raul (111688087)

Part 1

Question 1:

Time period for each flip flop=Propagation delay+ Setup time

= 2ns +3ns

=5ns

Considering the longest path,

Shortest possible clock period (T) =5ns + 4ns + 5ns + 3ns

=17ns

Fastest possible clock frequency = 1/T

=1/17ns

=58.82Mhz

Question 2:

Corrected code:

```
module mod1(sel, g0, g1, g2, g3, a);
input [1:0] sel;
input a;
output logic g0, g1, g2, g3;

always_comb begin
case(sel)
2'b00: g0 = a; g1=0; g2=0; g3=0;
2'b01: g1 = a; g0=0; g2=0; g3=0;
2'b10: g2 = a; g0=0; g1=0; g3=0;
2'b11: g3 = a; g0=0; g1=0; g2=0;
endcase
end

endmodule
```

Explanation:

In every case statement, only one of the output was clearly defined. Instead, every output should be assigned a specific value.

Question 3:

Code:

```
module mod2(a, b, c, d, e);
input a, c;
output logic b, d, e;

always_comb begin
if (c == 1) begin
e = a;
b = c;
end
else begin
e = 0;
b = a;
end
end

always_comb begin
d = a | c;
b = e ^ a;
end

endmodule
```

Explanation:

'b' is assigned in both the comb blocks simultaneously which cannot be simulated or synthesized correctly.

Part 2

Q.4 a)

We have implemented two types of testbenches. One is the regular System Verilog testbench & another testbench which compares the outputs with the outputs of a C program in order to check the correctness of the generated outputs.

We have tried to incorporate most of the possible cases in both our testbenches.

Reset=0/1, Valid_in=0/1, Positive & Negative Overflow are the critical test case scenarios for the system. By testing all these cases, we prove the correctness of our system. So instead of doing an exhaustive testing we prioritized testing these cases thoroughly as much as possible.

We can design a more robust testbench by combining exhaustive/random testing with the above critical test case scenarios to test behaviour of output and overflow at multiple instances.

Q.4 b):

Overflow occurs when the result of an arithmetic operation cannot be represented using the allotted number of bits. So, in case of an adder overflow can be detected in following scenarios:
- addition of two positive values, to get a negative answer, or
- addition of two negative values, to get a positive answer.

By comparing the sign bits of the two operands (multiplier output and feedback) of the adder with the sign bit of the output, we detected the overflow state. That is if the sign bit of the operands is not equal to the sign bit of the output there is change of sign, thereby an overflow.

While testing, the inputs a and b can take values from -127 to 127 thus, approximately three iterations of accumulation are required to test overflow.

The following exceptions were also considered while detecting overflow;

- If the output is negative but not overflowed after the system is reset.

Q.4 c)

Time period(ns)	Frequency(Ghz)	Area (μm^2)	Power(µW)	Critical Path Location
1.19(slack violated)	0.840	728.84	409.09	q1_reg to overflow_reg
1.21	0.826	714.74	392.34	q1_reg to overflow_reg
1.23	0.813	699.58	388.86	q1_reg to overflow_reg
1.25	0.8	691.33	376.74	q1_reg to overflow_reg
1.28	0.781	689.47	373.18	q1_reg to overflow_reg
1.3	0.77	676.70	356.89	q1_reg to overflow_reg
1.33(slack violated)	0.752	742.41	371.68	q1_reg to overflow_reg

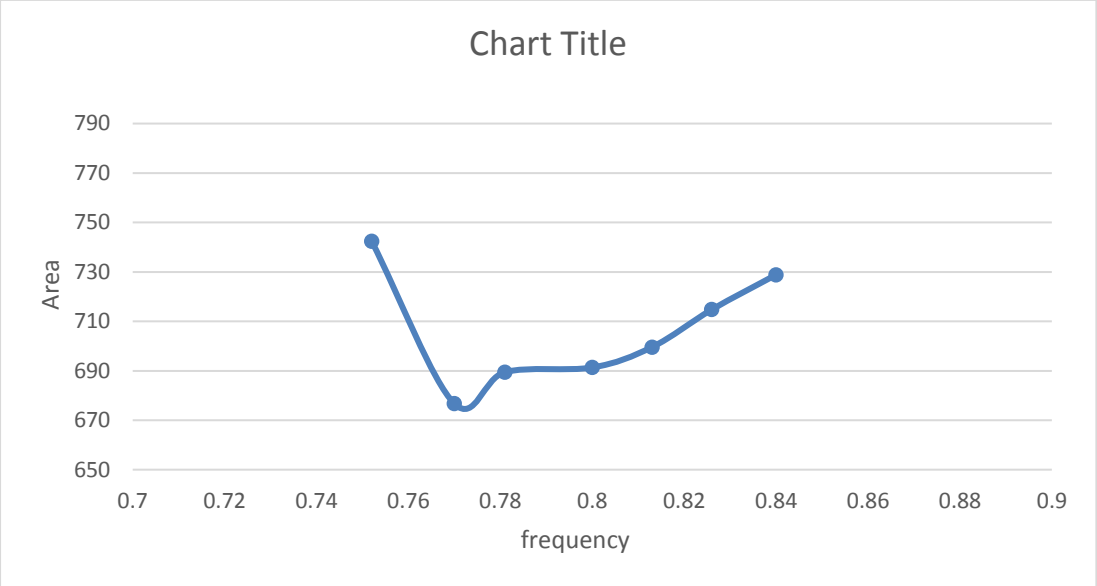
The minimum and maximum time periods for which the slack conditions are met are 1.21ns and 1.3ns. Thus, we reported the area and power at these frequencies. We also checked for time periods less than 1.21ns and greater than 1.3ns which resulted in slack violation.

Maximum reachable frequency: 0.826Mhz

In the code, q1 is the registered value for input a and overflow is the registered value of overflow signal. As it is the critical path, it is the longest path the signal follows in the design.

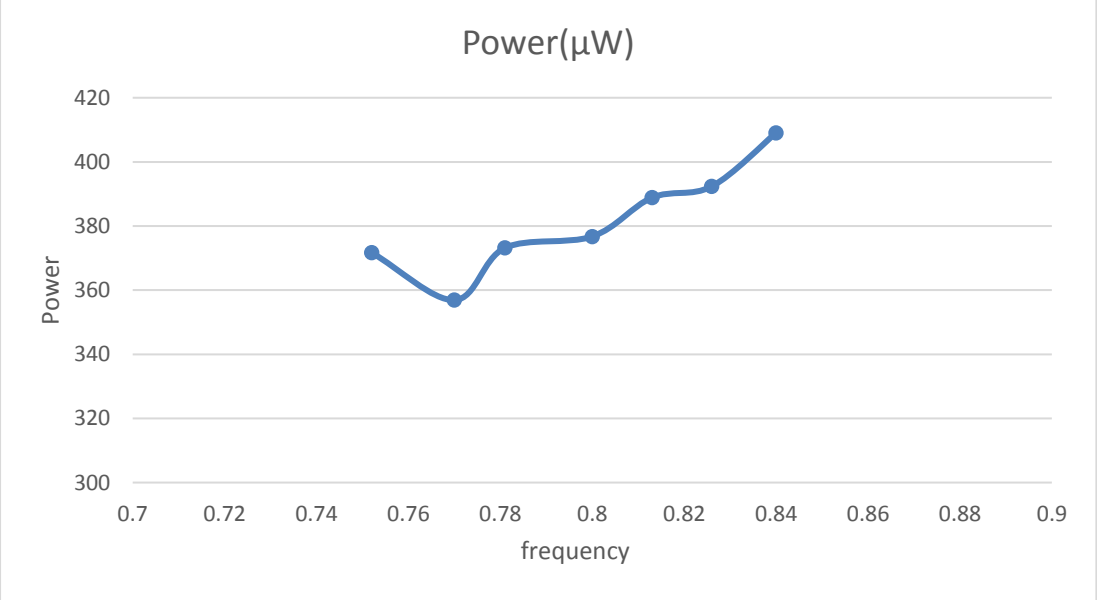
Q4 d)

Frequency vs Area:



For slack met conditions, that is between the frequency range of 0.77Ghz to 0.826Ghz area increases as frequency increases. But at frequencies below this range, as frequency decreases area increases.

Frequency vs Power:



For slack met conditions, that is between the frequency range of 0.77Ghz to 0.826Ghz power increases as frequency increases. But at frequencies below this range, as frequency decreases, power increases.

Q4 e)

Maximum clock frequency: 0.826Ghz

Total power :392.34µW

Minimum time period: 1.21ns

Number of input cycles: 50

Therefore, number of output cycles required= 50 + 2(as output takes two cycles to arrive) +1(as system is reset at the beginning) = 53

Energy consumed = 392.34uW * 53 * 1.21ns = 25.16 pJ

Q4 f)

Maximum clock frequency: 0.826Ghz

Power at this frequency: 392.34µW

Best-case energy per operation= 392.34 * 0.826 = 474.46 pJ

Q4 g)

Energy per operation increases with increase in clock frequency. As observed in the frequency vs power data retrieved previously, it can be stated that Power increases with increase in frequency. Energy is defined as Power/Frequency. Thereby an increase in Power will lead to an in increase in Energy.

Q4 h)

Yes, it is necessary to reset all registers. In this part of project, we have 4 registered values – inputs a & b, output f & overflow. So, considering an overflow condition occurs, and the overflow register is reset to 0. But the value in f now is overflowed & opposite in sign to that of the operands. Thereby any further calculations on it will result in an incorrect value. Thus, it gets imperative to reset f as well. Inputs a & b must be reset as and when needed.

Part 3

Q1)

Adding an extra register between the multiplier and the adder will lead the system to require an extra clock cycle to process the outputs. Thereby delaying the output and valid_out signals. Also affecting overflow signal.

We delayed the enable signal of f register by a clock cycle by adding always_ff block to it thereby delaying the outputs.

Yes, we need to do slight changes in the testbench by adding additional @(posedge clk) cycles before we reset the system in order to receive the output synchronously of the latest inputs.

Period (ns)	Frequency(Ghz)	Area(μm²)	Power(μW)	Critical Path location	Energy per operation(pJ)
0.91	1.09	804.9	657.16	f_reg to overflow_reg	602.89

Q2)

We implemented pipelining using instantiation for multiplier for stages s=2 to s=5. There was a considerable increase in frequency upto stage s=4, after which there was stagnation in maximum reachable frequency for s=5 and beyond.

Highest frequency reached in stage s=4: 1.64Ghz

Q3)

Every stage of pipelining, induced an additional delay to process the outputs. So, to counter the delay, we propagated the enable signal with delay from the input registers to the f register by adding always_ff blocks.

Stage(s)	Period (ns)	Max Frequency(Ghz)	Area(μm²)	Power(μW)	Critical Path location	Energy per operation(pJ)
2	0.81	1.23	880.9	930.5	S1_reg to qf_reg	756.09
3	0.64	1.56	1054	1525	f_reg to overflow_reg	977.56
4	0.61	1.64	1100.4	1756	f_reg to overflow_reg	1071.16
5	0.605	1.65	1161.35	1901	f_reg to overflow_reg	1150.1

Energy per operation for Part 2: 474.46 pJ

After pipelining, there is an increase in energy per operation for every stage.

Q4)

The most optimal design according to us is the 2-stage pipelined design(s=2).

Comparing with the rest of the stages, stage 2 gives the best trade-off between power and area for the acquired frequency. While in stage 3, there is a significant increase in frequency but it also increased area and power considerably which is not an optimal trade-off.

Q5)

Adding a pipelining to the adder will result in difficulties to handle feedback & also detection of overflow. The additional flipflop block will create an additional delay to the output f. So, computing overflow by comparison of the recent output value & its previous operands would not be possible simultaneously. Thus, it might as well create a delay in computing overflow.