Part 1:

Question 1.

The minimum clock period will be calculated using the critical path. The critical path is the path which has the longest duration. This gives us the shortest possible time to complete the operation. Therefore,

The critical path has its starting point from the register with input "b" and the ending point is the output register.

$T_{clk} = T_{setup} + T_{prop.delay} + T_{comb}$

$= 2ns + 2ns + 4ns + 3ns = 11ns.$

So, the shortest clock period = 11 ns.

Fastest frequency = $1/ T_{clk} = 0.09$ Ghz

Question 2.

Incorrect Code:

```
module mod1(sel, f0, f1, f2, f3, a);
    input [1:0] sel;
    input a;
    output logic f0, f1, f2, f3;
    always_comb begin
        case (sel)
            2'b00: f0 = a;
            2'b01: f1 = a;
            2'b10: f2 = a;
            2'b11: f3 = a;
    endcase
end
endmodule
```

Answer:

The above code when synthesized will produce latches in its design and not as a combinatorial circuit. There are 4 different outputs and each of them are assigned a value individually for one particular case only. When one output is being

assigned a value, the other outputs are not set to null, but instead are latched on to their previous values. Hence they will be interpreted as "*Inferred Memory Devices*" or more precisely Latches.

The synthesizing tool displays a message clearly stating that the above code is not being implemented as a combinatorial circuit but as memory device or latches.

```
Inferred memory devices in process
    in routine mod1 line 4 in file
        './part1.sv'.
================================================================================
|    Register Name    | Type  | Width | Bus | MB | AR | AS | SR | SS | ST |
================================================================================
|       f2_reg        | Latch |   1   |  N  | N  | N  | N  | -  | -  | -  |
|       f3_reg        | Latch |   1   |  N  | N  | N  | N  | -  | -  | -  |
|       f1_reg        | Latch |   1   |  N  | N  | N  | N  | -  | -  | -  |
|       f0_reg        | Latch |   1   |  N  | N  | N  | N  | -  | -  | -  |
================================================================================
Warning:  ./part1.sv:4: Netlist for always_comb block contains a
latch. (ELAB-974)
```

Corrected Code:

```
module mod1(sel, f0, f1, f2, f3, a);
    input [1:0] sel;
    input a;
    output logic f0, f1, f2, f3;
    always_comb begin
        f0 = 1'bx; f1 = 1'bx; f2 = 1'bx; f3 = 1'bx;
        case (sel)
            2'b00: f0 = a;
            2'b01: f1 = a;
            2'b10: f2 = a;
            2'b11: f3 = a;
    endcase
end
endmodule
```

Question 3.

Incorrect 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
```

Answer:
In the above code, the output ‘b’ has been assigned multiple times to different values. This in the physical circuit translates to a short circuit, as by assigning ‘b’ to different values, we are eventually connecting both together. Hence this

Corrected 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;
        end
        else begin
            e = 0;
```

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

Part 2:

Question 4.

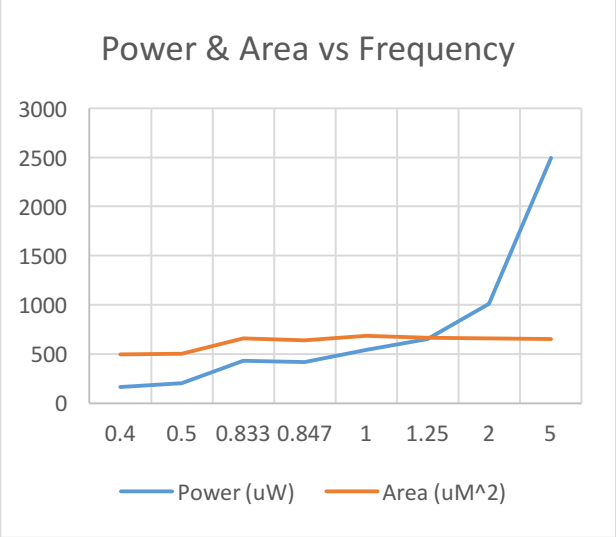
a) The test bench designed for the Multiplier and Accumulator (MAC) module, consists of 2 parts. A C program that generates test data for the module and a test bench code in System Verilog to feed the generated data to the module and run the MAC module designed against these values and log the output.
The C program generates 1000 random input data (8bit each, both +ve and –ve, range of -128 to 128) for each of the 2 input signals "*a ,b*". The generated input data is written to a file "*inputData*". It also generates another file "*expectedOutput*" which contains the list of expected outputs for each of the generated inputs, for verification.
The System Verilog test bench code reads the "*inputData*" file and stores the inputs into an array "*testdata*". The module then runs the MAC module with the inputs obtained and logs the output for each of the values on to a text file "*output*".
The generated file "*output*" is compared with "*expectedOutput*" to verify that the theoretical and simulated values are the same.
The testing is done over 1000 inputs in order to more thoroughly test the module.

b) The testbench developed is pretty straightforward. It tests it against randomly generated inputs of both positive and negative ranges.
A more robust method of testing would be to test it against extreme values or unique input cases alone, rather than a randomly generated input. Also, the test bench must be able to detect overflow or underflow conditions for the given test data.

c) The MAC module was synthesized for the following frequencies. The frequencies were chosen so as to cover equal distribution for values that satisfied and violated the slack requirements.
The maximum clock frequency attainable without violating slack requirements was 1.18 Ghz.

Clock (nS)	Frequency (Ghz)	Power (uW)	Area (uM^2)	Slack
2.5	0.4	163.1863	496.6219	MET
2	0.5	202.35	499.01599	MET
1.2	0.833	429.0667	657.5519	MET
1.18	0.847	414.0471	635.474	MET
1	1	539.5769	681.226	Violated
0.8	1.25	650.7803	663.1379	Violated
0.5	2	1011.6988	659.68	Violated
0.2	5	2491.9231	653.296	Violated

d) The frequency was increased and corresponding power consumption and area was noted down. It was observed that as the frequency increased the power consumption also increases. How ever the area initially increases but later on remains almost constant with a very slight decrease. This Is probably due to the optimization of design done by the synthesis tool.
The plot of power & area vs frequency is shown below.
The maximum frequency 1.18Ghz consumed 414.0471 nW .

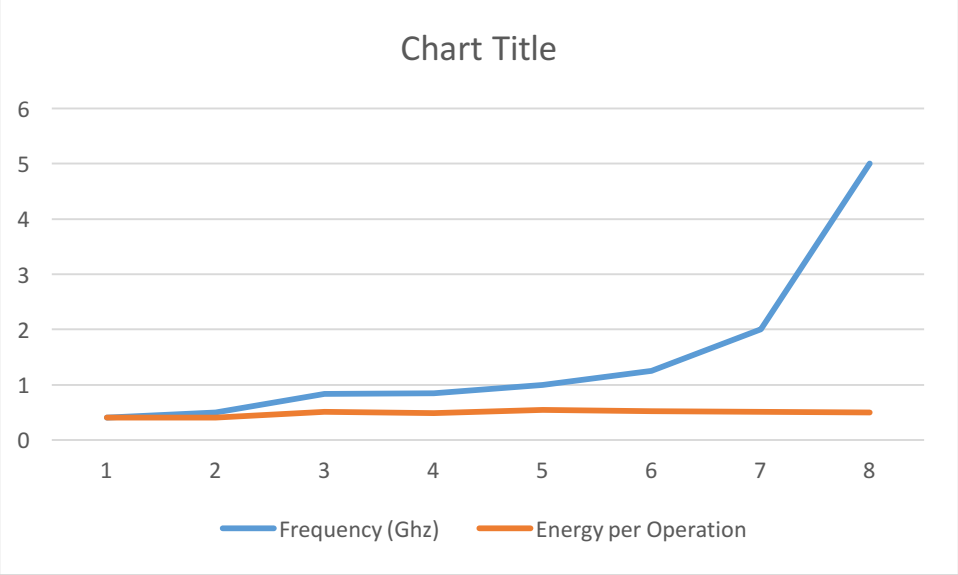


e) Energy per Operation

Power = Energy / Time
Energy per Operation = Power / Frequency
Energy per Operation = 414.0471 uW / 0.847Ghz
Energy per Operation = 0.488 J

f) The energy per operation varies with frequency. The energy per operation increases with frequency.

Frequency (Ghz)	Energy per Operation
0.4	0.407
0.5	0.404
0.833	0.51
0.847	0.488
1	0.539
1.25	0.52
2	0.505
5	0.498



g) A reset signal is required for the proper functioning of the module. This is because without resetting the circuit, the initial values of the input and output are not specified which will lead to an unpredictable state. This causes errors in the output. Also in case of any noise or errors inducing the reset signal will completely reset the module to the initial state.

Part 3:

1. Synthesis results

Stages (S)	Least Clock Period (ns)	Maximum Frequency (Ghz)	Dynamic Power (in uW)	Leakage Power (in uW)	Total Power (uW)	Combinational area	Non-combinational area	Total cell Area
reg	0.92	1.08	619.4336	15.4946	634.922	476.671999	217.055992	693.727991

The introduction of an extra register between the multiplier and the adder changes the critical path and reduces it. This helps in increasing the maximum achievable frequency. How ever the latency of the output also increases with the output coming out every 3 clock cycles. It is observed that the maximum achievable frequency has increased from 0.847 to 1.08 Ghz. Due to the introduction of the extra adder, the power consumption has also now increased to 634.92 uW.

The new pipelined design does not require a new Verilog test bench as the value of the input and output are not affected by the introduction of pipelining and hence remain the same. How ever in the C code the expectedOutput file must be appropriately updated with the increased latency in the output produced by the Verilog module due to the introduction of pipelining. If this is not done then when the "*diff –w* "is executed, the outputs generated will not match and hence will produce a difference.

2. For each pipeline stage, we vary the clock period in the runsynth.tcl file and find out the maximum achievable frequency of the module. For each of the frequency tested we also log the total area of the module as well as the total power consumed. The maximum achievable frequency was found to be 1.63Ghz

Stages (S)	Least Clock Period (ns)	Maximum Frequency (Ghz)	Dynamic Power (in uW)	Leakage Power (in uW)	Total Power	Combinational area	Non-combinational area	Total cell Area
2	0.84	1.19	834.5416	16.4790	851.0206	447.943999	325.583988	773.527987
3	0.6	1.66	1534.3	19.6508	1553.9508	468.159998	481.459983	949.619981
4	0.61	1.63	1628.4	19.8312	1648.2312	432.515999	565.249979	997.765978
5	0.61	1.63	1748.8	20.5439	1769.3439	421.610000	624.035977	1045.645977
6	0.61	1.63	1888.3	21.7542	1910.0542	419.482000	696.387975	1115.869975
reg	0.92	1.08	619.4336	15.4946	634.922	476.671999	217.055992	693.727991

3. The energy per operation was found to be higher as compared to the one we observed in part 2. This is due to the increase in number of modules being added. As the number of registers increase the associated power consumption also increases there by increasing the power consumed. Also as the frequency increases it is found that the energy per operation increases.

Stages (S)	Energy per operation (in J)
2	.7012954622
3	.9242771084
4	.9990184049
5	1.072883436
6	1.158466258

4. The best design would be the MAC with the pipelined 2 stage multiplier and the register between the multiplier and the adder. This is because it gives a balance between frequency and power consumption. Beyond this point the increase in frequency isn't significant while costing a lot of power usage. Also the area of the circuit also increases. Moreover it stagnates for the higher stages of pipeline making increasing the registers redundant.

5. While pipelining the adder might increase the maximum achievable frequency, there might be certain problems associated with the same.

- The latency of the system will increase. Time taken for the output will be higher.
- Area of system and components will increase as more registers are introduced, hence power consumption will also increase.