ESE 507 | PROJECT 1

MULTIPLY AND ACCUMULATE HARDWARE

Aswin Natesh Venkatesh & SBU ID: 111582677
Ranjini Ramesh SBU ID: 111580271

PART 1

QUESTION 1:

The shortest possible clock period for the given system can be calculated from its critical path (the longest duration path) which gives us the shortest time to complete an operation.

Given Data for each FF:

[1] Propagation Delay ($T_{Propagation}$) : 2ns
[2] Setup Time (T_{Setup}) : 3ns
[3] Hold Time (T_{Hold}) : 1ns
[4] Comb. Logic 1($T_{Comb.1}$) : 4ns
[5] Comb. Logic 2 ($T_{Comb.2}$) : 3ns

$T_{clk} = T_{Setup} + T_{Propagation} + T_{Comb.1} + T_{Comb.2}$
$T_{clk} = 3ns + 2ns + 4ns + 3ns$
$T_{clk} = 12ns$ [Shortest Clock Period]
Freq. $= 1/ T_{clk}$
$= 1/ 12ns = 83.33$ Mhz [Fastest Clock Frequency]

QUESTION 2:

Given 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;
                2'b01: g1 = a;
                2'b10: g2 = a;
                2'b11: g3 = a;
        endcase end
endmodule
```

The combinational procedural block contains a set of case statements and just one variable in the block gets a value assigned. This implies that rest three variables take the previous values acting as memory. Doing this forms an inferred latch and would not synthesize. All variables in a combinational procedural block should get a value assigned at the end of an operation and therefore one solution would be to assign default values at the beginning of the block.

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)
                g0 = 0; g1 = 0; g2 = 0; g3 = 0;
                2'b00: g0 = a;
                2'b01: g1 = a;
                2'b10: g2 = a;
                2'b11: g3 = a;
        endcase end
endmodule
```

Now, all the variables used in the block (g0 – g3) gets a value assigned and therefore the code is now a combinational circuit and is synthesizable.

QUESTION 3:

Given 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
```

In the given code, the variable b gets multiple values assigned in a cycle since its declared as output logic pin, taking multiple values in fraction of seconds makes the voltage in the terminal to fluctuate and may induce noise in the fabricated circuit. The easiest way to avoid this would be to prevent variable b from taking up a value in the first combinational block.

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;
                        // b = c; // Statement Commented
                end
                else begin
                        e = 0;
                        // b = a; // Statement Commented
                end
        end
        always_comb begin
                d =a|c;
                b =e^a;
        end
endmodule
```

PART 2

QUESTION 4.A:

Two test-benches were developed to test the MAC Hardware (Multiplier and Accumulator).

[1] Random Values Test

This test-bench is designed to fetch inputs from a file and test the system for 800 sets of input data, and later writes the output to an output file. The input file consists of values for 3 variables (Valid_in, A & B) and is generated randomly by a C code, within the ranges mentioned below.

Valid_in: (0) or (1) - (1bit)
A : (+127) ~ (-127) - (8 Bit Signed)
B : (+127) ~ (-127) - (8 Bit Signed)

The C code also generates expected output values by performing the same MAC operation and stores them in a separate file. This file is then compared with the output file generated by the System Verilog test-bench for mismatch if any.

[1] Basic Test-bench with Overflow

This test-bench is a slightly modified version of the test-bench given. This test-bench includes overflow flag value displayed on the monitor along with the accumulated sum and valid_out signals. This test-bench tests the system for a given set of inputs pre-coded.

QUESTION 4.B:

The variables used in the code are declared as signed variables, and therefore an overflow can be detected by simply checking few conditions on the variable's sign bit. Below is the overflow logic implemented.

[1] Overflow - When the output has an opposite sign while the operands have the same sign
[2] Overflow - When there is a sign mismatch between previous output and current output.

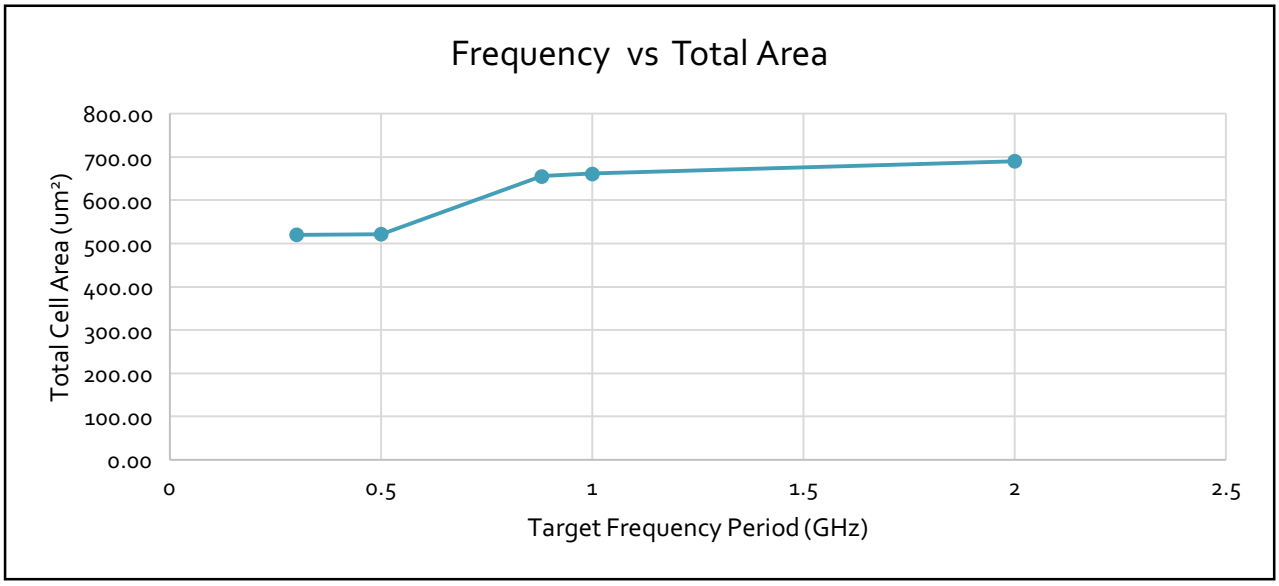
QUESTION 4.C:

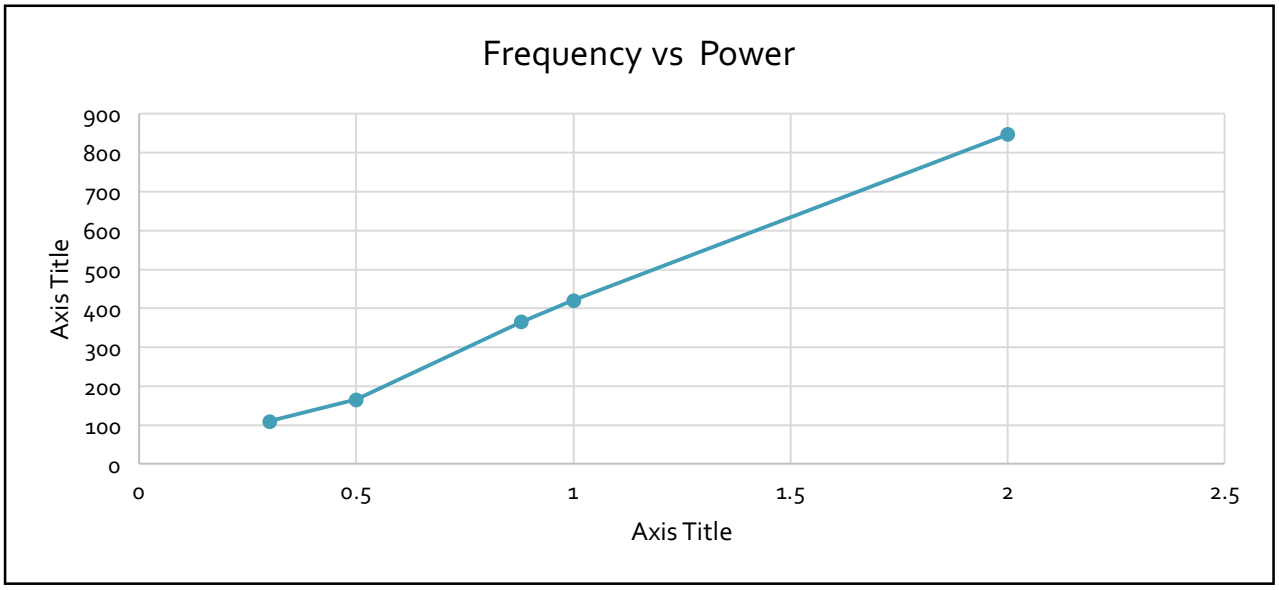
The module designed was tested for 5 different frequency period to observe the maximum operating speed of the MAC module. The maximum operating frequency for this part of the project is found to be 0.88Ghz. The critical part was observed to be same throughout the tested clock periods and it's from the output of Input Register A to the input of Output Register F. The results obtained are tabulated below.

TABLE 1

Target Frequency Period (GHz)	**2**	**1**	**0.88**	**0.5**	**0.3**
Clock Period (ns)	0.50	1.00	1.14	2.00	3.33
Combinational Area (um^2)	529.07	500.61	497.69	363.88	362.03
Non - Combinational Area (um^2)	161.46	160.93	158.27	158.27	158.27
Total Cell Area (um^2)	690.54	661.54	655.96	522.16	520.30
Total Dynamic Power (uW)	832.05	405.03	350.16	155.73	99.37
Cell Leakage Power (uW)	15.83	15.27	15.19	10.32	10.22
Total Power (uW)	847.88	420.30	365.34	166.05	109.59
Energy per Cycle Operation (mJ)	423.94	420.30	415.16	332.10	365.31
Timing Report (Slack)	-0.62	-0.12	0.00	0.01	1.29
	VIOLATED	VIOLATED	MET	MET	MET
Critical Part - Start Point	Output of a_reg	Output of a_reg	Output of a_reg	Output of a_reg	Output of a_reg
Critical Part - End Point	Input of f_reg	Input of f_reg	Input of f_reg	Input of f_reg	Input of f_reg
Output File Name	output4.txt	output.txt	output1.txt	output2.txt	output3.txt

QUESTION 4.D:





It can be inferred from the graphs above that the total area increases slightly (170um²) while increasing the operating frequency and the power shoots up from 109uW to 365.34uW (Slack MET) while increasing the frequency from 0.3GHz to 0.88Ghz. Therefore, for fastest processing, power required is enormous.

QUESTION 4.E:

Energy is a product of Power and Time. Energy per cycle is calculated by the relation given below.

$$Energy\ per\ Cycle\ Operation\ (mJ) = \frac{Power\ (\mu W)}{Frequency\ (GHz)} = \frac{365.34}{0.88} = 415.16\ mJ = 0.415\ J$$

$$Energy\ for\ 50\ Cycles\ (mJ) = \frac{Power\ (\mu W)}{Frequency\ (GHz)} \times 50 = \frac{365.34}{0.88} \times 50 = 20750\ mJ = 20.75\ J$$

QUESTION 4.F:

Best-Case Energy per operation is at the lowest operating frequency the module is tested. The module was tested for 0.3GHz for which the energy per operation is calculated below is a product of Power and Time. Energy per cycle is calculated by the relation given below.

$$Energy\ per\ Operation\ (mJ) = \frac{Power\ (\mu W)}{Frequency\ (GHz)} = \frac{109.59}{0.3} = 365.30\ mJ = 0.365\ J$$

QUESTION 4.G:

The Energy per operation increases with increase in clock frequency. This can be inferred from the last table which has energy per operation for various frequencies. It can be inferred that higher the frequency, more the energy is required for an operation.

QUESTION 4.H:

Reset signals are important because they clear the memory (Previous State Values) and reset variables to their initialized state. This is important for proper functioning of any system and also to get back the system to initial state when overclocked. Reset signals in advanced systems are handled by watchdog timers which resets the system automatically when a single operation exceeds a certain amount of time.\

PART 3

SUMMARY OF DIFFERENT PIPELINE DESIGNS

TABLE 2

Operations	Part 2 Design from Previous Part (For Comparison)	Part 3 With Additional Register	Part 3 With Additional Register and Pipelined Multiplier (Stage 2)
Target Frequency Period (GHz)	**0.88**	1.09	1.22
Clock Period (ns)	1.14	0.92	0.82
Combinational Area (um^2)	497.69	506.20	458.58
Non - Combinational Area (um^2)	158.27	239.67	348.19
Total Cell Area (um^2)	655.96	745.86	806.78
Total Dynamic Power (uW)	350.16	551.34	788.67
Cell Leakage Power (uW)	15.19	16.63	16.98
Total Power (uW)	365.34	567.98	805.65
Energy per Cycle Operation (mJ)	415.16	522.54	660.64
Timing Report (Slack)	0.00	0.00	0.00
	MET	MET	MET
Critical Part - Start Point	Output of a_reg	Product Register	Mult_Instance S1
Critical Part - End Point	Input of f_reg	Sum Register	Product Register
Output File Name	output1.txt	Output_Part3_01.txt	Output_Part3_02.txt
System Verilog Code	Part2_Tb1.sv	Part3_01.sv	Part3_03.sv

TABLE 3

Operations	Pipelined Multiplier				
	Stage 2	**Stage 3**	**Stage 4**	**Stage 5**	**Stage 6**
Target Frequency Period (GHz)	1.41	1.59	1.56	1.59	1.82
Clock Period (ns)	0.71	0.63	0.64	0.63	0.55
Combinational Area (um^2)	493.43	452.73	451.93	427.20	425.87
Non - Combinational Area (um^2)	289.94	393.41	434.64	547.16	656.22
Total Cell Area (um^2)	783.37	846.15	886.58	974.36	1082.09
Total Dynamic Power (uW)	833.40	1144.20	1201.30	1464.90	1964.40
Cell Leakage Power (uW)	17.54	18.09	18.48	19.56	21.38
Total Power (uW)	850.95	1162.29	1219.78	1484.46	1985.78
Energy per Cycle Operation(mJ)	604.17	732.24	780.66	935.21	1092.18
Timing Report (Slack)	0.00	0.00	0.00	0.00	0.00
	MET	MET	MET	MET	MET
Critical Part - Start Point	A Register	Mult_Instance S2	Mult_Instance S2	Mult_Instance S4	Mult_Instance S5
Critical Part - End Point	Mult_Instance S1	Sum Register	Mult_Instance S3	Sum Register	Sum Register
Output File Name	Output_Part3_04.txt	Output_Part3_05.txt	Output_Part3_06.txt	Output_Part3_07.txt	Output_Part3_08.txt
System Verilog Code	Part3_02.sv	Part3_02.sv	Part3_02.sv	Part3_02.sv	Part3_02.sv

QUESTION 3:

In this part of the project, the MAC module was tested with three pipelining designs.

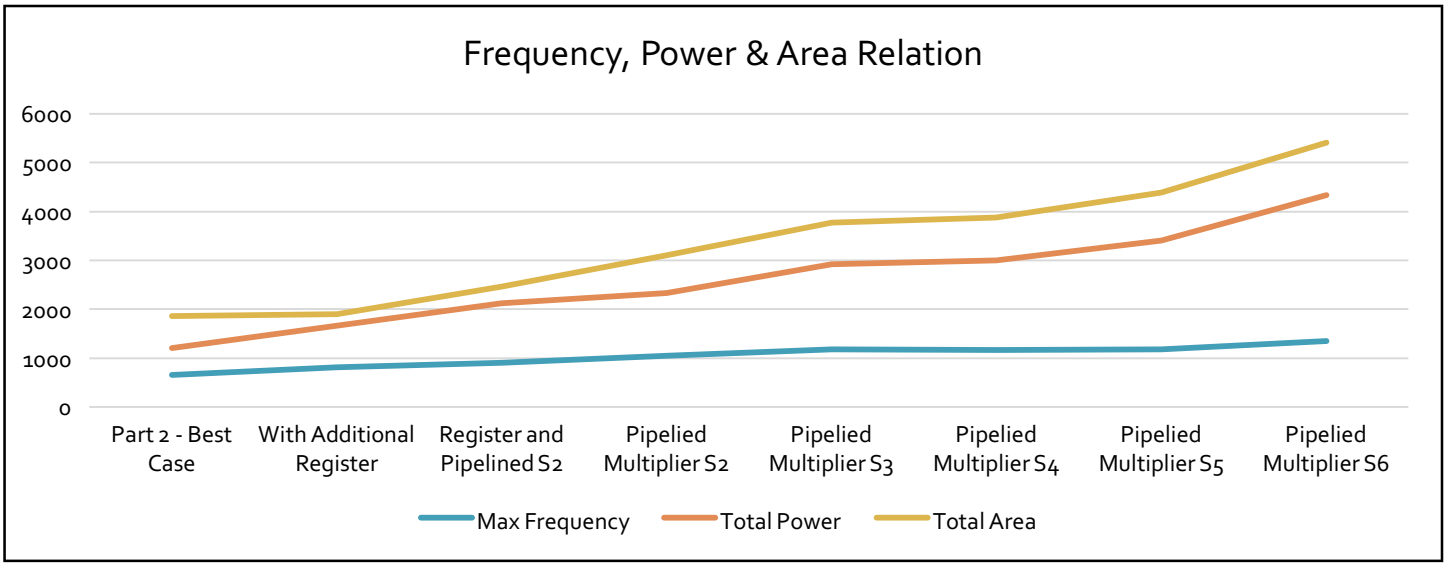
- The first design consists of a register between the multiplier and adder. Doing this improves the maximum clock speed from 0.88GHz (Part 2 Design) to 1.09Ghz (Part 3 Design). This is tabulated in Table 2.
- The second design consists of a multistage pipelined multiplier provided by Synopsys. The MAC module has been tested for 6 Stages of pipelined multiplier and the values are tabulated in Table 3. It can be inferred that more the amount of pipelining stages is added, the higher power and area is required and faster the frequency is. The maximum frequency observed is 1.82 GHz when implementing a 6 Stage Pipelined Multiplier.
- The third design is a combination of the above two designs, which includes a register between the multiplier and adder and a 2 Stage pipelined multiplier. This design achieves a maximum frequency of 1.22 GHz. This is given in Table 2.

QUESTION 4:

The best design for this MAC module would be implementing a 3 Stage Pipelined Multiplier. This can be concluded by taking into consideration Frequency, Area and Power for all the designs. The Table 4 below shows the values and it can be seen from the Graph below that after 3 Stage Pipelined Multiplier, the frequency remains almost constant while the area and power keeps increasing. Therefore 3 Stage Pipelined Multiplier is the best design for this Multiply and Accumulate Hardware Module operating at 1.58 GHz.

TABLE 4

Parameters	Max Frequency (GHz)	Total Power (uW)	Total Area (uM^2)
Part 2 - Best Case	0.88	365.34	655.96
With Additional Register	1.09	567.98	239.67
With Additional Register and Pipelined Multiplier (Stage 2)	1.22	805.65	348.19
Pipelied Multiplier Stage 2	1.40	850.94	783.36
Pipelied Multiplier Stage 3	1.58	1162.29	846.14
Pipelied Multiplier Stage 4	1.56	1219.78	886.57
Pipelied Multiplier Stage 5	1.58	1484.46	974.35
Pipelied Multiplier Stage 6	1.81	1985.78	1082.08



QUESTION 5:

The Adder module cannot be pipelined because it involves feedback signals as a operand. For Accumulation Operation, the input values have dependency on the previous stage output (Multiplier) and therefore cannot be parallel computed.

**** End of Report ****

APPENDIX: PART 2 SYSTEM VERILOG CODE

APPENDIX: PART 2 SYSTEM VERILOG CODE

```
module part2_mac(clk, reset, a, b, valid_in, f, overflow, valid_out);

   input clk, reset, valid_in;                // Inputs (1 Bits)
   input signed [7:0] a, b;                   // Inputs A & B (8 Bits)

   output logic signed [15:0] f;              // Output F (16 Bits)
   output logic signed valid_out=0, overflow=0;

   logic signed [7:0] areg, breg;
   logic signed [15:0] product, sum;
   logic voflag=0, offlag=0;
   logic  signed [15:0] sumprev;              // Added on Saturday

   always_ff @(posedge clk) begin             // Control Unit 1 | Reg A & Reg B

      if(reset == 1) begin                    // Check Reset Signal
         areg <= 0;                           // If Yes -> Pass 0 for areg & breg
         breg <= 0;
         voflag <= 0;
      end

      else if (valid_in == 1) begin
       areg <= a;                             // Else -> Pass Inputs for areg & breg
         breg <= b;
         voflag <= 1;
     end

      else begin
      areg <= 0;                              // If Yes -> Pass 0 for areg & breg
      breg <= 0;
      voflag <= 0;
     end
   end

   always_ff @(posedge clk) begin             // Control Unit 2 | Reg C

      if(reset == 1)                          // Check Reset Signal
         f <= 0;
      else  begin
          f <= sum;
          valid_out <= voflag;
          offlag <= overflow;
          sumprev <= sum;
    end
   end

   always_comb begin
      product = areg * breg;
      sum = product + f;
      if(areg[7]==breg[7] && f[15]!=breg[7] || f[15]!=sumprev[15]) overflow <= 1;
      else if(reset != 1 && offlag == 1) overflow <= 1;
      else overflow <= 0;
   end

endmodule
```