Name:
Section:
x500:

Lab 6: 8-bit Multiplier

1. Algorithm/Block Diagram

Following are executed at the rising edge of the clock signal .
*Note every shift adds $(001)_2$ to the counter, K=1 when the counter is $(111)_2$ and M is LSB of the multiplier.

State 0:

When my set=0 there is an output of zero transitioning back to the same state (0) in the process initializing done<=0.

However, when set=1 the multiplier is loaded into the accumulator register going to execute the following statements in state 1. Transition from state 0 to state 1.

State 1:

If the least significant bit of the multiplier [M] has a value of 1, then we must add the multiplicand from the adder register to the accumulator transitioning to state 2.

If the counter has a binary value of 7, that means the value of k=1 at when [LSB Mplier] M=0. So then the accumulator shifts to the right and adds $(001)_2$ to the counter that transitions from state 1 to state 3.

In the case where [LSB Mplier] M=0 and k=0 when the counter has not reached $(111)_2$ there is a right shift in the accumulator which adds $(001)_2$ to the counter. State does not change in this if statement.

State 2:

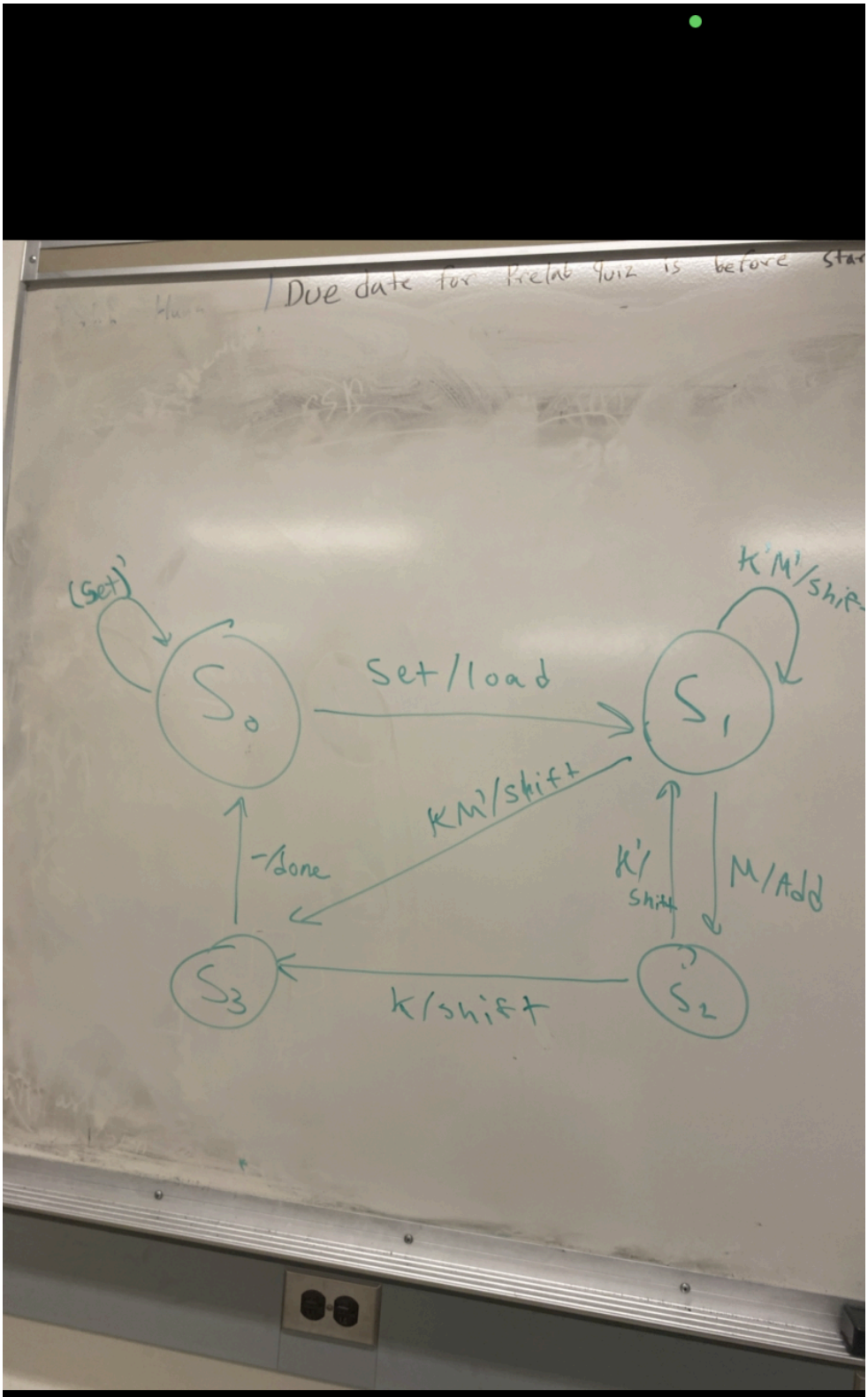
When the value of done is 0, the following will execute: if counter is $(111)_2$ meaning k=1 there will be a right shift in the accumulator giving the new counter's value of $(000)_2$ then

transitioning to state 3. However, if the [LSB Mplier] M=1 then we shift the accumulator to the right and add $(001)_2$ to the counter.

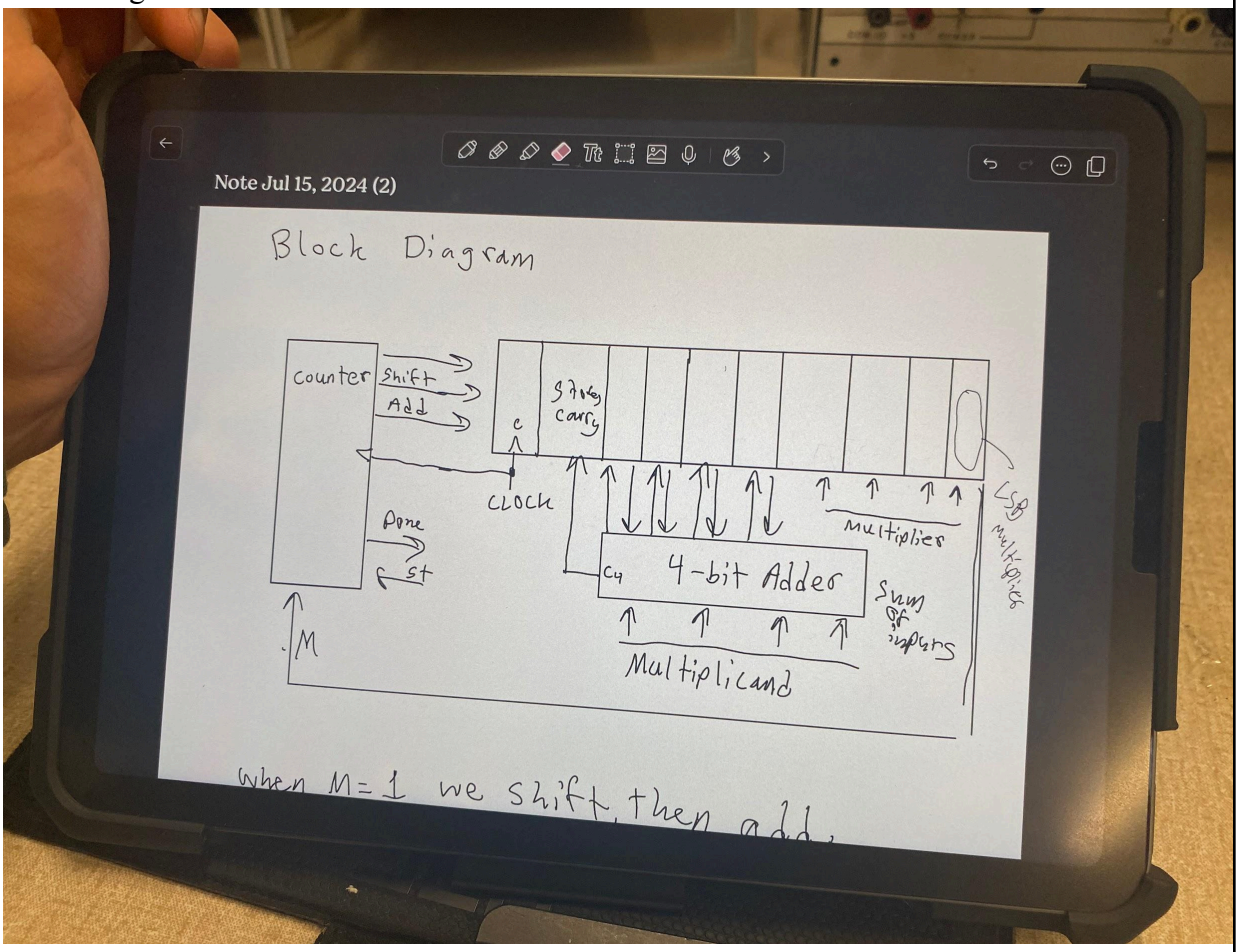
State 3:

Once in state 3 assuming the value of done initially is 0 when entering state 3, then from state 3 the value of the done register is given a value of 1. Now the accumulator's value is assigned to the final wire product. State transition goes from state 3 to state 0.

State Diagram:


Due date for Prelab Quiz is before
(Set)'
K'M'/shift
Set/load
S0
S1
KM'/shift
done
K'/shift
M/Add
S3
S2
K/shift

Block Diagram:



Describe the multiplication algorithm that you used in your design. If you are doing the Finite State Machine, you need to explain the operation inside every state. Include relevant details such as State Diagram, Block Diagram (sequential & combinational), etc.
(Simply writing "We followed the algorithm from the textbook and implemented it in Verilog" in the Procedure section just won't cut it.)

2. Verilog Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 07/18/2024 10:03:06 AM
// Design Name:
// Module Name: BinaryMultiplier
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module BinaryMultiplier(
input [7:0] Multiplier,
input [7:0] Multiplicand,
input CLK,
input set,
output reg [15:0] Product = 0
    );

    reg [1:0] currstate = 0;
    reg done = 0;
    reg [16:0] accumulator;
    reg [2:0] counter = 0;




    always @(posedge CLK) begin
        case(currstate)
            0 : begin
                if (~done && set) begin
```

```
            Product <= 0;
            accumulator <= Multiplier;
            currstate <=1;
        end
        if (done && ~set) begin
            done <= 0;
        end
    end
    1 : begin
        if (~done) begin
            if (accumulator[0] == 1)begin
                accumulator <= accumulator + {Multiplicand, 8'b00000000};
                currstate <= 2;
            end
            else begin
                if (counter == 7) begin
                    accumulator <= accumulator >> 1;
                    counter = counter + 1;
                    currstate <= 3;
                end
                else begin
                    accumulator <= accumulator >> 1;
                    counter = counter + 1;
                end
            end
        end
    end
    2 : begin
        if (~done) begin
            if (counter == 7) begin
                accumulator <= accumulator >> 1;
                counter <= 0;
                currstate <= 3;
            end
            else begin
                accumulator <= accumulator >> 1;
                counter = counter + 1;
                currstate <= 1;
            end
        end
    end
    3 : begin
```

```
            if (~done) begin
               done <= 1;
               currstate <= 0;
               Product <= accumulator[15:0];
            end
         end
      endcase
   end

endmodule
```

A Copy of Verilog Code used for the design of 8-bit Multiplier

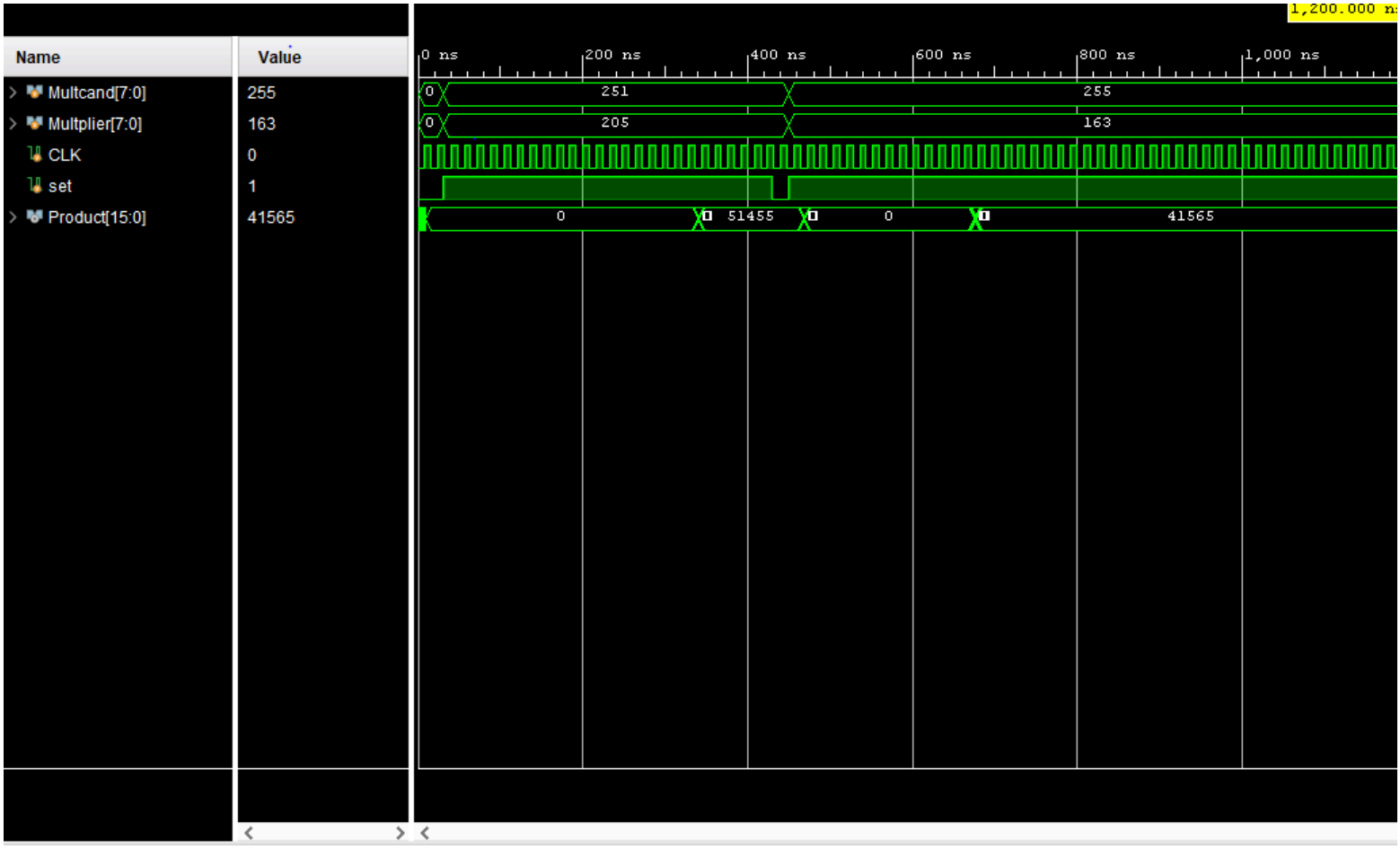
3. Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 07/18/2024 10:27:16 AM
// Design Name:
// Module Name: BinaryMultiplier_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module BinaryMultiplier_tb();
    reg [7:0] Multcand;
    reg [7:0] Multplier;
    reg CLK = 0;
    reg set;
    wire [15:0] Product;

    BinaryMultiplier instanc(
    .Multiplicand(Multcand),
    .Multiplier(Multplier),
    .CLK(CLK),
    .set(set),
    .Product(Product));

    always #8 begin
    CLK = ~CLK;
    end

    initial
```

```
    begin
    set = 0; Multcand = 0; Multplier = 0;
    #30
    set = 1; Multcand = 8'b11111011; Multplier = 8'b11001101;
    #400
    set = 0;
    #20
    set = 1; Multcand = 8'b11111111; Multplier = 8'b10100011;
    end
endmodule
```

Copy of Testbench used for Simulation

4. Simulated Waveform



Screenshot of Waveforms after Behavioral Simulation

5. Constraint File

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
#create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]



## Clock signal
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]

## Switches
set_property PACKAGE_PIN V17 [get_ports {Multiplier[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[0]}]
set_property PACKAGE_PIN V16 [get_ports {Multiplier[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[1]}]
set_property PACKAGE_PIN W16 [get_ports {Multiplier[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[2]}]
set_property PACKAGE_PIN W17 [get_ports {Multiplier[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[3]}]
set_property PACKAGE_PIN W15 [get_ports {Multiplier[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[4]}]
set_property PACKAGE_PIN V15 [get_ports {Multiplier[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[5]}]
set_property PACKAGE_PIN W14 [get_ports {Multiplier[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[6]}]
set_property PACKAGE_PIN W13 [get_ports {Multiplier[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplier[7]}]


set_property PACKAGE_PIN V2 [get_ports {Multiplicand[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[0]}]
```

```
set_property PACKAGE_PIN T3 [get_ports {Multiplicand[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[1]}]
set_property PACKAGE_PIN T2 [get_ports {Multiplicand[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[2]}]
set_property PACKAGE_PIN R3 [get_ports {Multiplicand[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[3]}]
set_property PACKAGE_PIN W2 [get_ports {Multiplicand[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[4]}]
set_property PACKAGE_PIN U1 [get_ports {Multiplicand[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[5]}]
set_property PACKAGE_PIN T1 [get_ports {Multiplicand[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[6]}]
set_property PACKAGE_PIN R2 [get_ports {Multiplicand[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Multiplicand[7]}]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {Product[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[0]}]
set_property PACKAGE_PIN E19 [get_ports {Product[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[1]}]
set_property PACKAGE_PIN U19 [get_ports {Product[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[2]}]
set_property PACKAGE_PIN V19 [get_ports {Product[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[3]}]
set_property PACKAGE_PIN W18 [get_ports {Product[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[4]}]
set_property PACKAGE_PIN U15 [get_ports {Product[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[5]}]
set_property PACKAGE_PIN U14 [get_ports {Product[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[6]}]
set_property PACKAGE_PIN V14 [get_ports {Product[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[7]}]

set_property PACKAGE_PIN V13 [get_ports {Product[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[8]}]
set_property PACKAGE_PIN V3 [get_ports {Product[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[9]}]
set_property PACKAGE_PIN W3 [get_ports {Product[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[10]}]
set_property PACKAGE_PIN U3 [get_ports {Product[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[11]}]
```

```
set_property PACKAGE_PIN P3 [get_ports {Product[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[12]}]
set_property PACKAGE_PIN N3 [get_ports {Product[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[13]}]
set_property PACKAGE_PIN P1 [get_ports {Product[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[14]}]
set_property PACKAGE_PIN L1 [get_ports {Product[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Product[15]}]



set_property PACKAGE_PIN T18 [get_ports set]
set_property IOSTANDARD LVCMOS33 [get_ports set]
```

Copy of Constraint File used for Implementation

6. Answer the Following Questions

The latency is the number of clock cycles it takes from the moment the set(start) signal is at 1 till when the output product has been outputted. The more 1's in the inputted multiplier and multiplicand binary numbers would result in having more clock cycles maximizing the latency for the multiplication operation.

a) What is the latency (in clock cycles) for a multiplication operation using your design? Explain how you obtain the latency for the multiplication

Yes, the number of 1's for the Mcand or Mplier changes the latency time in clock cycles.
In my test bench I had 8'b00000000 for both the multiplier and multiplicand to get the minimum latency, but I had values 8'b11111111for both the multiplicand and multiplier to achieve the maximum latency.

Maximum latency in clock cycles in this binary multiplication is 18 clock cycles between when the Set(start) is turned on until the output has been reached.

Minimum latency in clock cycles in this binary multiplication is 9 clock cycles between when the Set(start) is turned on until the output is reached.

b) Does it vary based on the value in Mcand and Mplier? If so, what are the min and maximum latency?

7. Discussion

Something we recognized was that based on our .v files there were different flip flop estimations and LUT values when we implemented our design on Vivado. Furthermore, the power the chip used based on our implementations differed based on the conditional statements and operations we set for the files to execute between specific inputs and outputs assigned in the constraints file.

My LUT utilization is 0.02% greater than my partner.

My Flip Flop utilization is 0.02% greater than my partner.

My max latency was 2 clock cycles greater than my partner.

My code was much longer with more conditions in comparison to my partner's code.

As a new finding I have recognized that there is a larger maximum and minimum latency time (in clock cycles) when there are more conditions and operations to be followed based on the creator's .v file increasing the utilization of LUT(Look-Up-Table) and Flip Flops.

Furthermore, another finding that I have established was

Find at least 1 additional group in your lab. Compare the algorithm, implementation (coding style), max latency, and the resulting utilization (LUTs and FFs). Describe your findings and any ideas about why one design may be advantageous. Feel free to discuss this with your TA, discussion section, or the professor.