ABOUT THE AUTHOR

I am Saiteja and I am from Hyderabad, India. I completed my bachelors in Electronics and Communication Engineering from Gokaraju Rangaraju Institute of Engineering and Technology with a CGPA of 9.97/10 in the year 2019. I received a Gold Medal for the academic excellence from my university. Right after that, I got an opportunity to work for Tata Consultancy Services as a Systems Engineer and have worked for 3 years until July 2022. I speak English, Hindi, and Telugu where the last of these is my first language. My interests include but not limited to Cricket, Table-Tennis and learning new languages. Currently I am pursuing my master's in Computer Engineering at San Jose State University.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION:

This activity is used review the system-level design by designing, functionally verifying a digital system for accelerating the factorial computation.

MY GROUP:

Name: Student_Team 6
Members: Tirumala Saiteja Goruganthu (016707210), Harish Marepalli (016707314)

GIVEN TASK:

The task is to design a digital system for accelerating the factorial computation. The system should start execution upon receiving an external input "***Go***" and should output a "***Done***" signal when the execution is completed. In addition, an "***Error***" signal should be set when an input greater than 12 is entered. Apart from that, the factorial accelerator shall calculate, and display results up to 12!. Inputs for n should be unsigned. Use only one 4-bit down counter, one register, one 2x1 multiplexers, two greater than comparators, a 32-bit unsigned multiplier, and one tri-state buffer (although another 2x1 multiplexer is preferred). The tool used for this task is "***Vivado Xilinx***" in which we write the "***Verilog HDL*****"** to verify our design.

STEPS TAKEN TO COMPLETE THE TASK:

a. Spent time to understand the question carefully.
b. Designed the system's datapath using the given building blocks using Visio.
c. Drawn the two-piece CU-DP block diagram that shows all relevant signals using Visio.
d. Prepared an ASM chart which describes the cycle-by-cycle operations of the datapath.
e. Designed a bubble (state) diagram for the "*next-state logic*" (NS) part of the control unit.
f. Constructed an output table for the "*output logic*" part of the control unit.
g. Created a new project in the Vivado software and added new leaf-level modules for all the required components such as counter, multiplexer by specifying the correct input and output parameters. Note that we used behavioral modelling when writing the Verilog code.
h. Created a module for the Datapath which incorporates or instantiates all the leaf-level modules by connecting each-other through parameters.
i. Added a new module for the Control path which includes the code for current state combinational circuit, state assigning stage, and next state combinational circuit.
j. A module for the top-level module is created which instantiates data path and control path modules. Along with that, to verify our design, a top-level testbench module has been created with few relevant inputs.
k. Finally, ran the simulation and observed the output waveforms for their correctness.

DISCUSSION SECTION:

a. Datapath Explanation:

The first step taken to design the factorial accelerator digital system is to prepare a datapath. The requirement was to use only the following leaf-level components.

a. A Down Counter as shown in *Fig5.1*

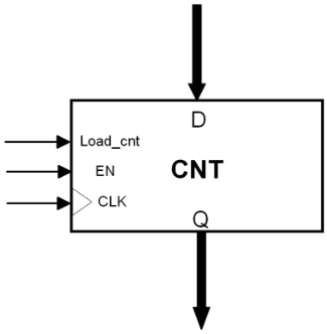


Fig5.1 – Down Counter

b. A Greater Than Comparator as shown in *Fig5.2*

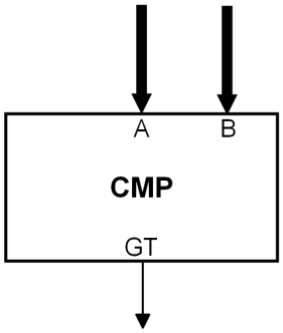


Fig5.2 – Comparator

c. The Multiplier as shown in *Fig5.3*

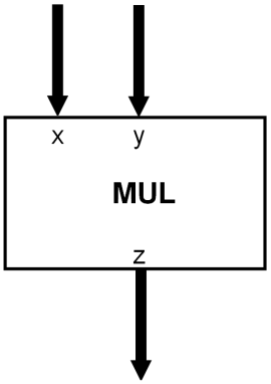


Fig5.3 – Multiplier

d. A Register as shown in *Fig5.4*

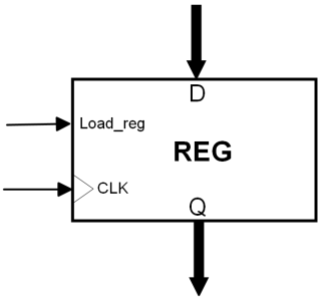


Fig5.4 – Register

e. A 2x1 Multiplexer as shown in *Fig5.5*

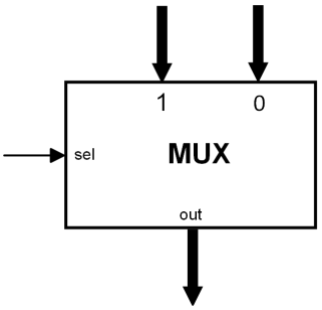


Fig5.5 – 2x1 Multiplier

The Datapath is designed by interconnecting the above digital components and is shown in *Fig 5.6* and is also shown under the appendix section.

a. First, after the user gives the external input '*Go*', the input value (*n*) is loaded into 4-bit down counter by setting '*Load_Cnt*' line to logic high. Note that since the counter is a sequential circuit, it needs the '*CLK*' input to be at positive edge or negative edge. In this exercise, we will be using positive edge triggered down counter.

b. At the same time, we send the input (*n*) to a 4-bit comparator to check if the n is greater than 12. If n value is greater than 12, then the line '*GT12*' will be high, and this signal will be sent to the control unit.

c. During this operation, a 32-bit 2x1 multiplexer is used to initialize the '*f*' value to 1 by setting the selection line '*Sel*' to 1.

d. This '*f*' value is loaded into the register by setting the '*Load_Reg*' line to logic high. This loading is part of the factorial loop where the corresponding '*f*' value that is getting stored in the register will change for each iteration.

e. After loading the input value (*n*), the counter will send it to another 4-bit comparator. This comparator is the starting point of a factorial loop where if the given input value is greater than 1 is checked. Depending on the output of the comparator, the '*GT1*' signal will be set to logic low or high.

f. The calculated '*f*' value and the current '*n*' value is multiplied using a 32-bit unsigned multiplier.

g. This multiplied value is transported back to the 2x1 multiplexer at 0 port where this value is iteratively used to load into the register and multiplied using the multiplier.

h. This process will be done until the comparator with output '*GT1*' outputs a logic low which indirectly means the n value is equal to or less than 1 and the factorial computation is done.

i. Finally, the result is then sent to another 2x1 multiplexer which uses '*OE*' as its selection line is used to determine the completion of factorial computation and simultaneously used to send the '*product*' as well as '*Done*' control signal to the control unit.

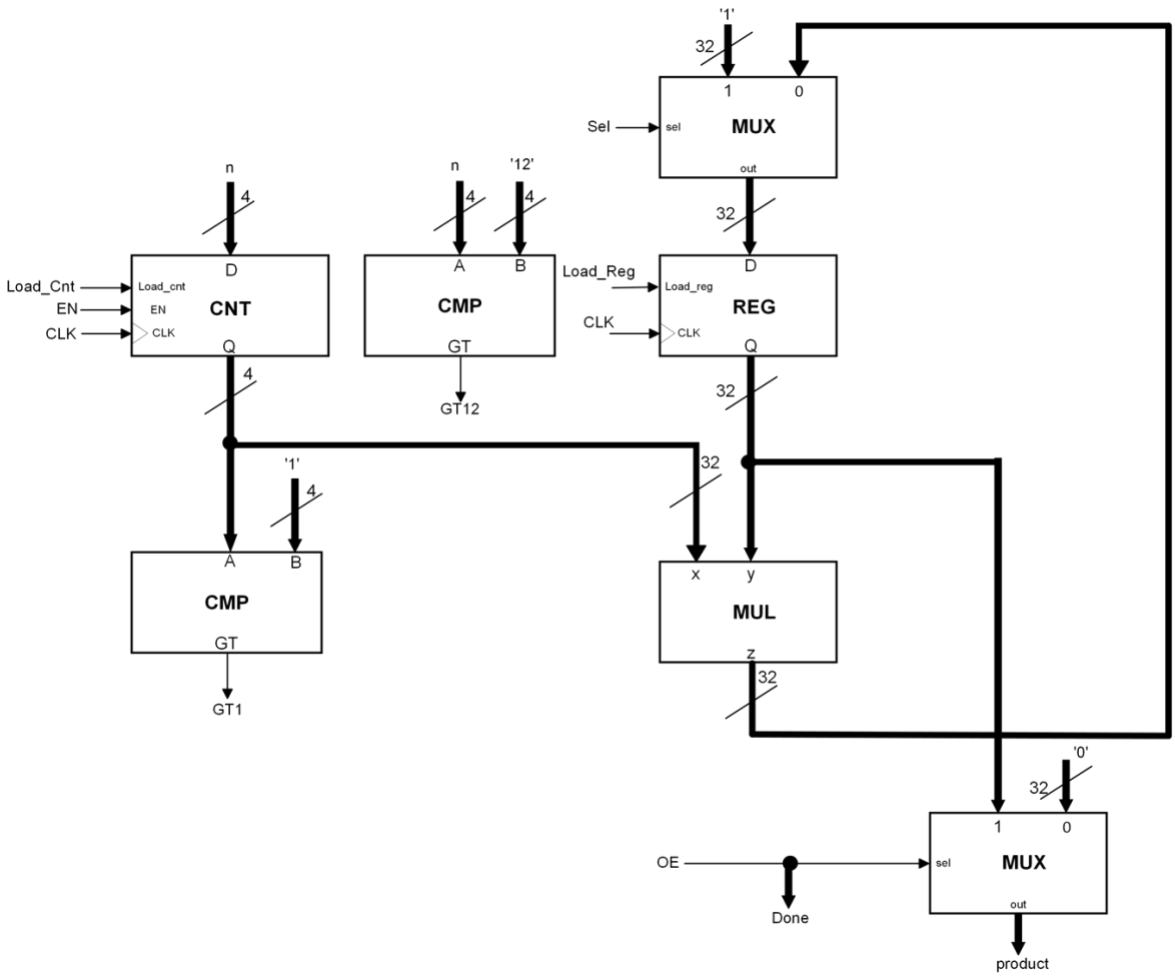


Fig5.6 – Factorial Accelerator Datapath (Using Visio)

b. CU-DP Diagram Explanation:

After the design of the Datapath, the next step is to proceed by drawing the CU-DP block diagram. In general, the CU-DP diagram consists of two blocks, the Control Unit, and the Datapath Unit. This diagram will help us to see the design at the top level with few user inputs and outputs and few communication lines between the data path and the control path. The CU-DP diagram for the current task is shown in *Fig5.8* and it is also shown in the appendix section.

a. Let us look at the control unit signal description as shown in the *Table5.7*

Signal Name	Signal Function	Description
CLK	External Input	Clock Signal
Go	External Input	System starts if Go = 1
GT1	Input from Datapath	n>1 condition check
GT12	Input from Datapath	n>12 condition check
Sel	Output from Controlpath	f=1 / reg selection line
Load_Reg	Output from Controlpath	Load Enable for register
Load_Cnt	Output from Controlpath	Load Enable for counter
OE	Output from Controlpath	Selection for multiplexer#2
EN	Output from Controlpath	Enable signal for counter
Done	External Output	System output if Done = 1
Error	External Output	System output if Error = 1

Table5.7 – Control Unit Signals Description

b. Let us look at the data unit signals in the CU-DP block diagram. The most important signal is the input (n) which is a 4-bit value. All the components in the data path that are discussed in the previous section such as Counter, Comparator, multiplexer need control signals from the control unit. These control signals' description has already been covered in the *Table5.7*. Finally, the 32-bit output of the factorial is made available by a signal called '*product*.'

c. The CU-DP diagram can be looked at as the top-level block diagram for any digital system. It is drawn in the user's point of view and thus has far less control signals than the actual inner circuit. In the current task, there are 3 external inputs and 3 external outputs that are to be given by the user to start the system. Since, the requirement was to start the system when Go signal is given, this signal is in the hands of the user. CLK signal and the 4-bit input (*n*) are also the external inputs from the user and the signals Done, Error, product are the external outputs from the system after the execution.

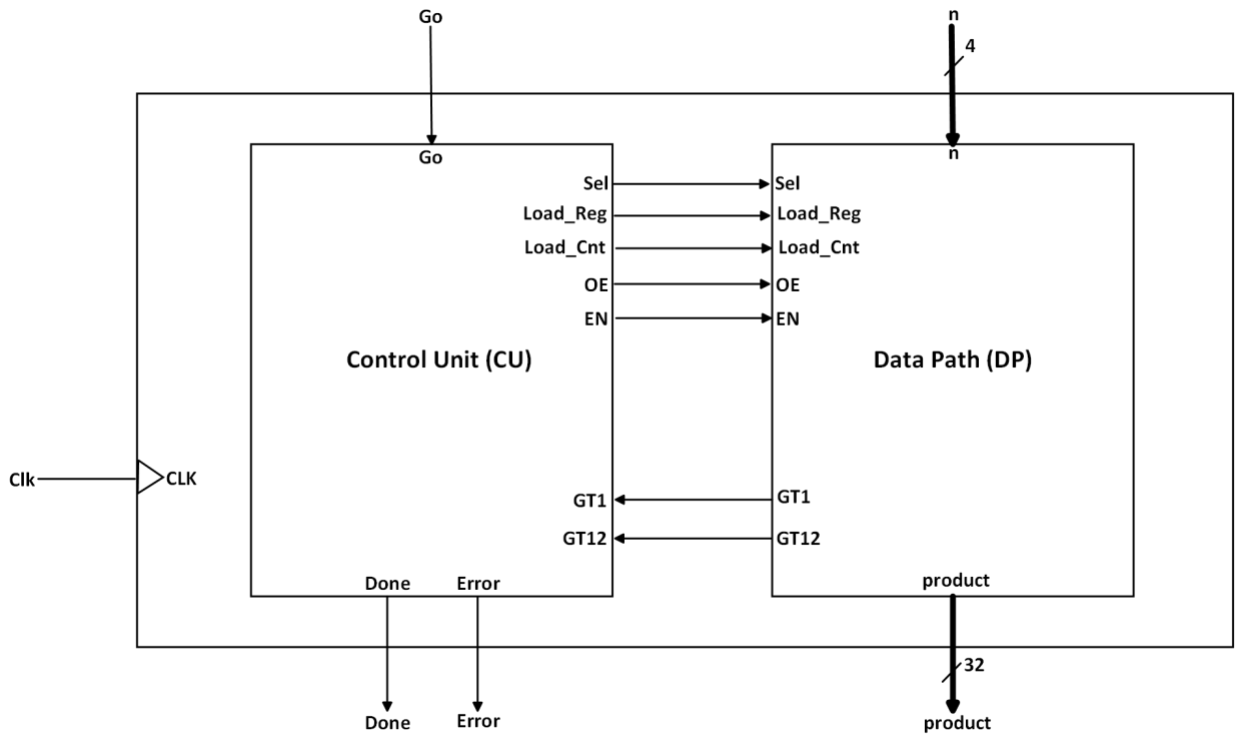


Fig5.8 – CU-DP Block Diagram

c. ASM Chart Explanation:

The Algorithmic State Machine (ASM) method is a method for designing Finite State Machines (FSMs) and is used to represent diagrams of digital circuits. The ASM diagram is like a state diagram but more structured and, thus, easier to understand. An ASM chart is a method of describing the sequential operations of a digital system. In the current task, the ASM chart is shown in the *Fig5.9*.

a. First state S0 is an idle state where the system will not perform any function during this state.

b. Next there is a condition box which checks the Go input signal. If this signal is 0, the system will go back to the idle state else it will go to another condition box which is used to check if the input value (*n*) is greater than 12 by setting GT12 signal. If the GT12 is 1, then we can say that the error has been received and the system will go back to being idle i.e., S0 else if GT12 is 0, then the state is changed from S0 to S1.

c. In the state S1, the counter is loaded with the value n using the Load_Cnt control signal and the register is loaded with 1 by enabling the Sel signal and Load_Reg control signal. After this, the control goes to the next state S2.

d. In the state S2, the control can go to either state S3 or S4 depending upon the GT1 comparator value. If the GT1 is 0, then the factorial is done, and state changes to S3 where the user can collect the output. If the GT1 value is 1, then the factorial loop is

executed by calculating the f=f*n and decrementing the counter value by 1 and the state is changed to S2.

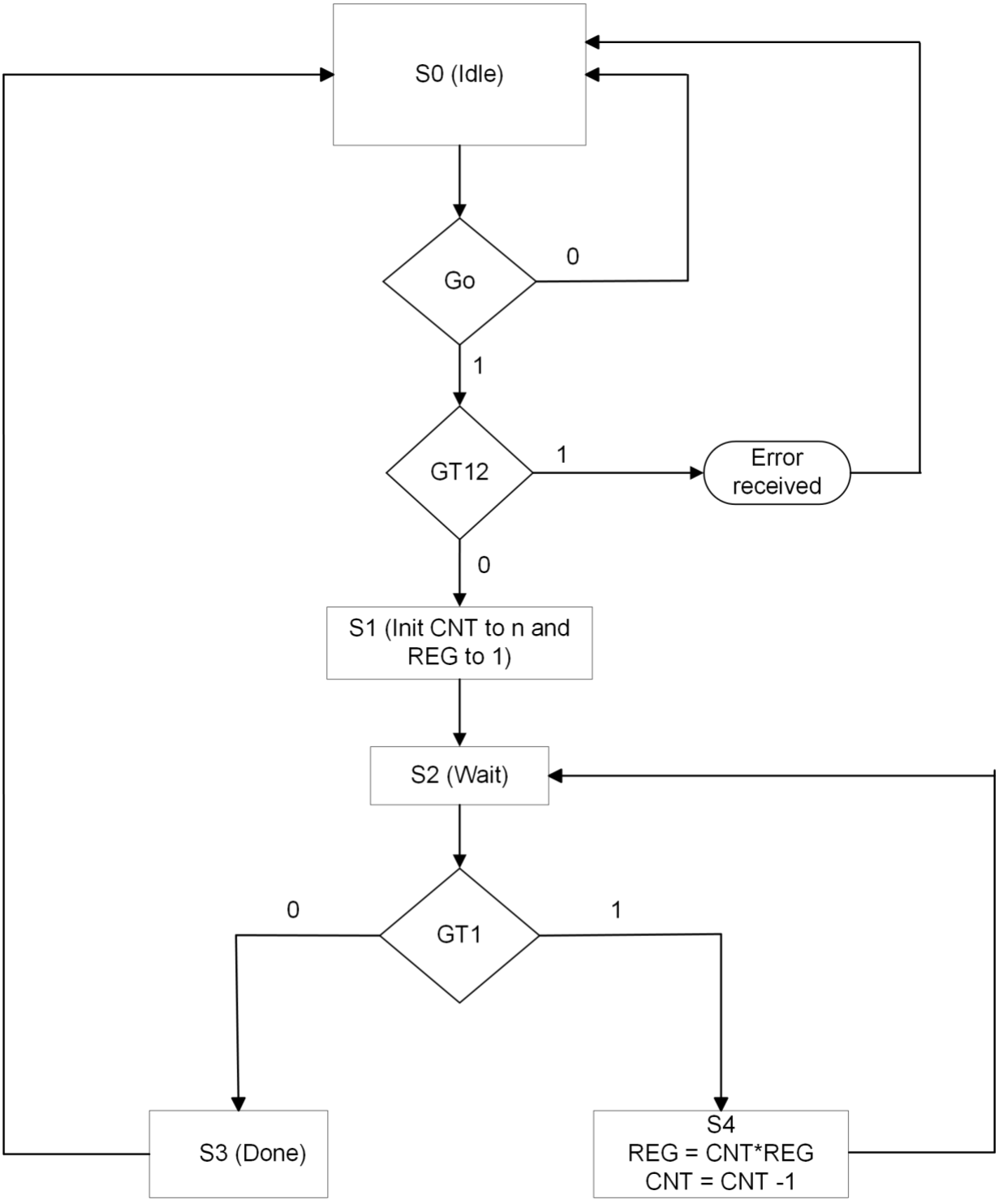


Fig5.9 – ASM Chart for the Factorial Accelerator

d. Bubble Diagram Explanation:

The next step in the design is preparing a bubble/state diagram for factorial accelerator. The state diagram is a visual representation of how sequential circuits behave. It clearly depicts the transition of states from one state to the next, as well as the output for a given input. Each current state is represented by a circle in a state diagram. The bubble diagram for the current task has been shown in the *Fig5.10*.

a. The explanation is similar for the ASM chart and for the bubble diagram since both talk about the sequential part of the current task and involves states into the picture. The state S0 will go to S1 only if Go=1 and GT12=0 else will stay back in state S0. Once it is in S1, it will go to S2 regardless of the condition. Changing the states from S2 to S3 or S4 heavily depends on the GT1 comparator output. If GT1 is 0, then state is changed to S3, and the process is done else it will go into factorial loop (S4) until finally it changes to state S3.

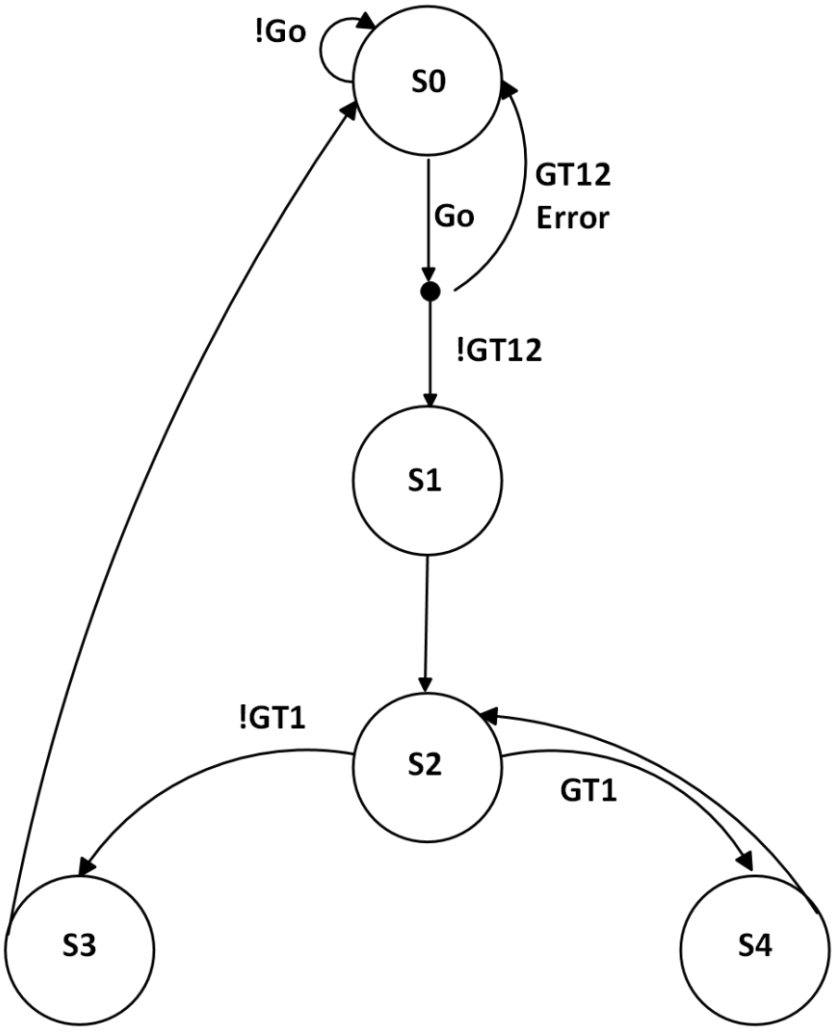


Fig5.10 – Bubble Diagram for the Factorial Accelerator

e. Output Logic Table Explanation:

The final step in the design phase is to draw an output logic table which shows the values of all control signals for each and every state defined. In the current task, the output table is shown below in *Table5.11*. Just to revisit, the control signals that we used are Sel, Load_Cnt, EN, Load_Reg, OE, Done and Error.

a. In S0, the state is idle which is why all the control signals are at logic zero.

b. In state S1, we need to initialize and load the register with 1 and load the counter with n. This is done by enabling Sel, Load_Reg, EN, and Load_Cnt.

c. In state S2, this is a wait state so all the control signals will be at logic zero.

d. In state S3, the factorial computation is done which is why OE and Done signals are enabled.

e. In state S4, the factorial loop is calculated which means the Sel will be set to logic zero and Load_Reg, EN signals are set to logic high to the current f value.

State	Outputs						
	Sel	Load_Cnt	En	Load_Reg	OE	Done	Err
S0	0	0	0	0	0	0	-
S1	1	1	1	1	0	0	-
S2	0	0	0	0	0	0	-
S3	0	0	0	0	1	1	-
S4	0	0	1	1	0	0	-

Table5.11 – Output Logic Table for the Factorial Accelerator

This concludes the Design phase of our digital system. In the next step, we use Hardware Description Language (HDL) such as Verilog to write high-level code and verify the design by simulating the written modules.

We use Vivado Xilinx software to write our Verilog code. First step is to create a project and, in that project, create and add modules. At the beginning, leaf-level component modules such as counter, multiplexer are written and then these modules are instantiated in more top-level modules such as data path and control path.

To verify the code that we have written, we must run the simulation by giving some inputs. These inputs are given using a test bench in the vivado. A test bench is another module which instantiates a given module by providing few test cases or inputs.

After writing the testbench for all test cases, we can just click on "*Run Simulation*" button to start the simulation which in-turn will produce output waveforms for all the variables mentioned in the testbench. These waveforms are covered in the appendix section.

COLLABORATION SECTION:

1. Worked together on the data path and control path design, and eventually prepared them using Microsoft Visio tool.

2. Collaborated in designing the FSM while preparing the corresponding ASM chart and the bubble diagram using the Microsoft Visio tool.

3. Worked together to construct the leaf-level modules in the vivado using Verilog code.

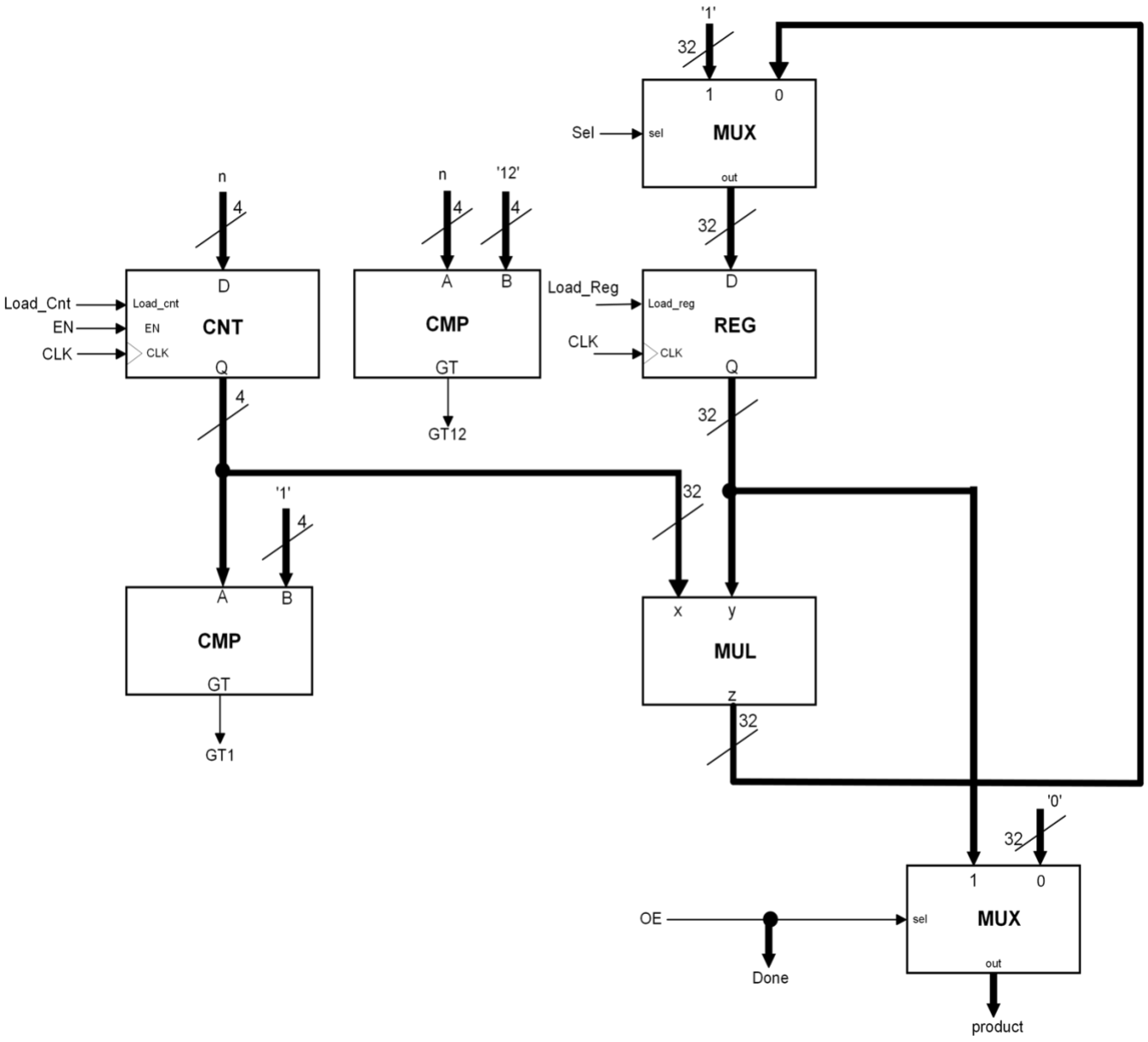
4. By working together, we have written the test bench to verify a bunch of values and observed the output waveforms.

CONCLUSION:

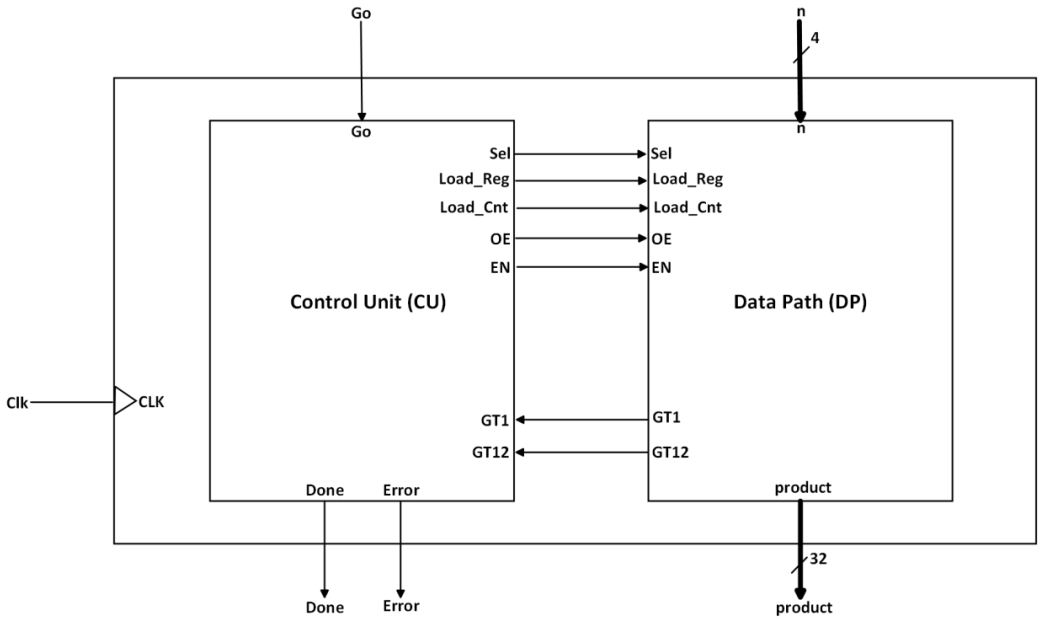
In conclusion, we end the report by understanding the system-level design by designing and verifying a digital system for factorial acceleration. We also gained insight into Verilog coding and the usage of vivado Xilinx software for verifying the digital systems. We also got some hands-on the Microsoft Visio tool for creating block diagrams and other functional visualizations.

APPENDIX

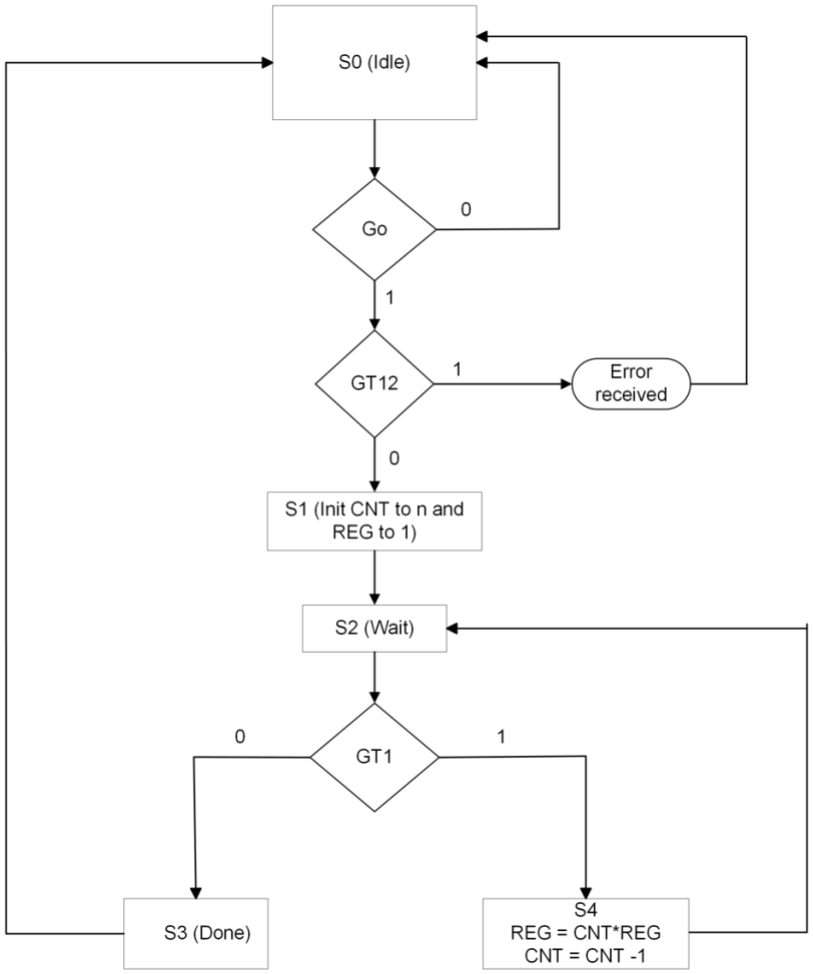
a. Datapath Diagram



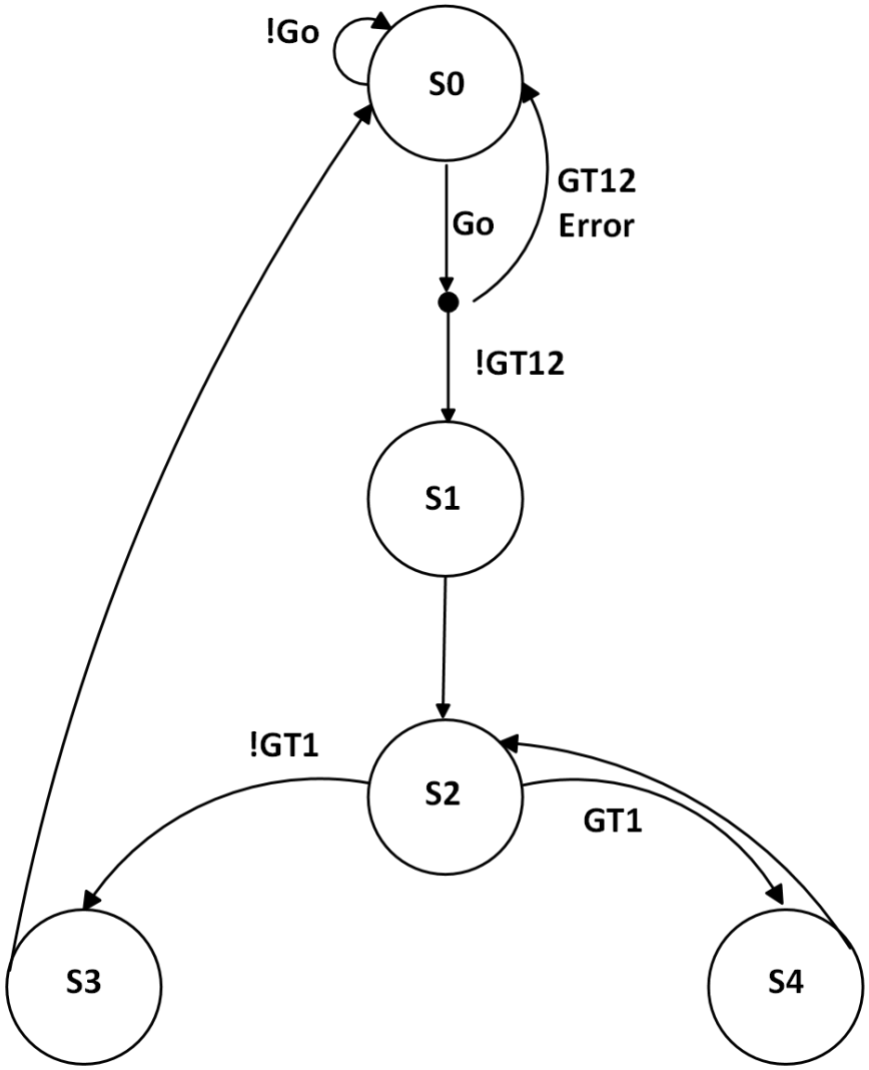
b. CU-DP Block Diagram



c. ASM Chart



d. Bubble Diagram



e. Output Logic Table

State	Outputs						
	Sel	Load_Cnt	En	Load_Reg	OE	Done	Err
S0	0	0	0	0	0	0	-
S1	1	1	1	1	0	0	-
S2	0	0	0	0	0	0	-
S3	0	0	0	0	1	1	-
S4	0	0	1	1	0	0	-

f. Source Code – FATopModule_tb.v (TestBench Module)

```
`timescale 1ns / 1ps
module FATopModule_tb;
reg Go_tb, CLK_tb;
reg [3:0] n_tb;
wire Done_tb, Error_tb;
wire[31:0] product_tb;

FATopModule FATM(.n(n_tb),
                 .Go(Go_tb),
                 .CLK(CLK_tb),
                 .Done(Done_tb),
                 .Error(Error_tb),
                 .product(product_tb));

task automatic tick;
   begin
      CLK_tb <= 1'b0;
      #50;
      CLK_tb <= 1'b1;
      #50;
   end
endtask

initial
begin
   $display("Factorial Testing Begins");
   CLK_tb = 0;
   n_tb = 4'd1;
   tick;
   while(n_tb < 15)
   begin
      Go_tb = 1;
      CLK_tb = 0;
      tick;
      while(!(Done_tb) || (Error_tb))
      begin
         tick;
      end
      if(Done_tb)
      begin
         $display("%0d! = %0d", n_tb, product_tb);
      end
      else if(Error_tb)
      begin
         $display("Error received, input = %0d", n_tb);
      end
      n_tb = n_tb + 4'd1;
```

```
    end
    $display("Test Complete");
    $finish;
  end
  endmodule
```

g. Source Code - FATopModule.v

```
`timescale 1ns / 1ps
module FATopModule(
    input Go,
    input CLK,
    input[3:0] n,
    output Done,
    output Error,
    output[31:0] product
    );

    wire Sel, Load_Reg, Load_Cnt, OE, EN;
    wire GT1, GT12;

    FA_Datapath FADP(.CLK(CLK),
                     .EN(EN),
                     .n(n),
                     .Sel(Sel),
                     .Load_Reg(Load_Reg),
                     .Load_Cnt(Load_Cnt),
                     .OE(OE),
                     .GT1(GT1),
                     .GT12(GT12),
                     .product(product));

    FA_Controlpath FACP(.EN(EN),
                        .Go(Go),
                        .CLK(CLK),
                        .Sel(Sel),
                        .Load_Reg(Load_Reg),
                        .Load_Cnt(Load_Cnt),
                        .OE(OE),
                        .GT1(GT1),
                        .GT12(GT12),
                        .Done(Done),
                        .Error(Error));
endmodule
```

h. Source Code – FA_Datapath.v

```
`timescale 1ns / 1ps
module FA_Datapath #(parameter Data_width = 32)(
        input CLK,
        input [3:0]n,
        input Sel,
        input Load_Reg,
        input Load_Cnt,
        input OE,
        input EN,
        output GT1,
        output GT12,
        output [Data_width - 1:0] product
        );

        wire [3:0] Counter_Output;
        wire [Data_width - 1:0] Mux_Output;
        wire [Data_width - 1:0] Reg_Output;
        wire [Data_width - 1:0] Mul_Output;

        Comparator #(4) Cmp_GT12Checker (.A(n), .B(4'd12), .GT(GT12));
        Comparator #(4) Cmp_GT1Checker (.A(Counter_Output), .B(4'd1), .GT(GT1));
        Counter #(4) CNT (.D(n), .Q(Counter_Output), .Load_Cnt(Load_Cnt), .EN(EN),
        .CLK(CLK));
        Multiplier #(32) MUL (.X({28'b0,Counter_Output}), .Y(Reg_Output),
        .Z(Mul_Output));
        Multiplexer2x1 #(32) MUX (.Inp1(32'b1), .Inp0(Mul_Output), .Sel(Sel),
        .Out(Mux_Output));
        Multiplexer2x1 #(32) Out_Buffer (.Inp0(32'b0), .Inp1(Reg_Output), .Sel(OE),
        .Out(product));
        Register #(32) REG (.D(Mux_Output), .Q(Reg_Output), .Load_Reg(Load_Reg),
        .CLK(CLK));
endmodule
```

i. Source Code – FA_Controlpath.v

```
`timescale 1ns / 1ps
module FA_Controlpath(
   input CLK,
   input Go,
   input GT1,
   input GT12,
   output reg Sel,
   output reg Load_Reg,
   output reg Load_Cnt,
   output reg OE,
   output reg EN,
```

```
    output Done,
    output Error
    );
    reg[2:0] CurrState=0, NextState;
    reg Err_Flag = 0, Done_Flag = 0;

assign Error = GT12;
assign Done = Done_Flag;

//Initialize the states
parameter S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4;

always @ (CurrState, Go)
begin
        case (CurrState)
        S0: //Idle State
        case({Go, GT12})
        //Idle State and Go='1', GT12='1'
        2'b11: {NextState, Err_Flag} <= {S0, 1'b1};

        //Idle State and Go='1', GT12='0'
         2'b10: {NextState, Err_Flag} <= {S1, 1'b0};

        //Idle State and Go='0', GT12='0/1'
        2'b0?: {NextState, Err_Flag} <= {S0, 1'b0};

        default: NextState = S0;
        endcase

        //Next State will directly be S2
        S1: NextState <= S2;

        //Next State will depend on GT1 value
        S2: NextState <= GT1 ? S4 : S3;

        //Next State will directly be S0
        S3: NextState <= S0;

        //Next State will directly be S2
        S4: NextState <= S2;
        endcase
end

always @(posedge CLK)
CurrState = NextState;

always @(CurrState)
begin
        case(CurrState)
```

```
                S0: //Idle State
                begin
                        {Sel, Load_Cnt, EN, Load_Reg, OE, Done_Flag} <= 6'b1_1_1_0_0_0;
                end
                S1: //Load Register with '1' and Load Counter with n
                begin
                        {Sel, Load_Cnt, EN, Load_Reg, OE, Done_Flag} <= 6'b1_1_1_1_0_0;
                end
                S2: //Conditional state
                begin
                        {Sel, Load_Cnt, EN, Load_Reg, OE, Done_Flag} <= 6'b0_0_0_0_0_0;
                end
                S3: //Output is ready and Done is set to '1'
                begin
                        {Sel, Load_Cnt, EN, Load_Reg, OE, Done_Flag} <= 6'b0_0_0_0_1_1;
                end
                S4: //Decrement the counter and Load the register after multiplication
                begin
                        {Sel, Load_Cnt, EN, Load_Reg, OE, Done_Flag} <= 6'b0_0_1_1_0_0;
                end
        endcase
    end
endmodule
```

j. Source Code – Comparator.v

```
`timescale 1ns / 1ps
module Comparator #(parameter DataWidth =4)(
   input [DataWidth - 1:0] A,
   input [DataWidth - 1:0] B,
   output reg GT
   );
   always @(A or B)
      begin
         //Initialize GT value with '0'
         GT <= 1'b0;

         if(A > B)
            GT <= 1'b1;
      end
endmodule
```

k. Source Code – Multiplexer.v

```
`timescale 1ns / 1ps
module Multiplexer2x1 #(parameter Data_width = 32)(
    input [Data_width - 1:0] Inp0,
    input [Data_width - 1:0]Inp1,
    input Sel,
    output reg [Data_width-1 : 0] Out
    );
    always @(Inp0, Inp1, Sel)
        begin
            if(Sel)
                Out <= Inp1;
            else
                Out <= Inp0;
        end
endmodule
```

l. Source Code – Multiplier.v

```
`timescale 1ns / 1ps
module Multiplier #(parameter Data_width = 32)(
    input [Data_width - 1:0] X,
    input [Data_width - 1:0] Y,
    output reg [Data_width - 1:0] Z
    );
    always @ (X or Y)
        begin
            Z <= X * Y;
        end
endmodule
```

m. Source Code – Register.v

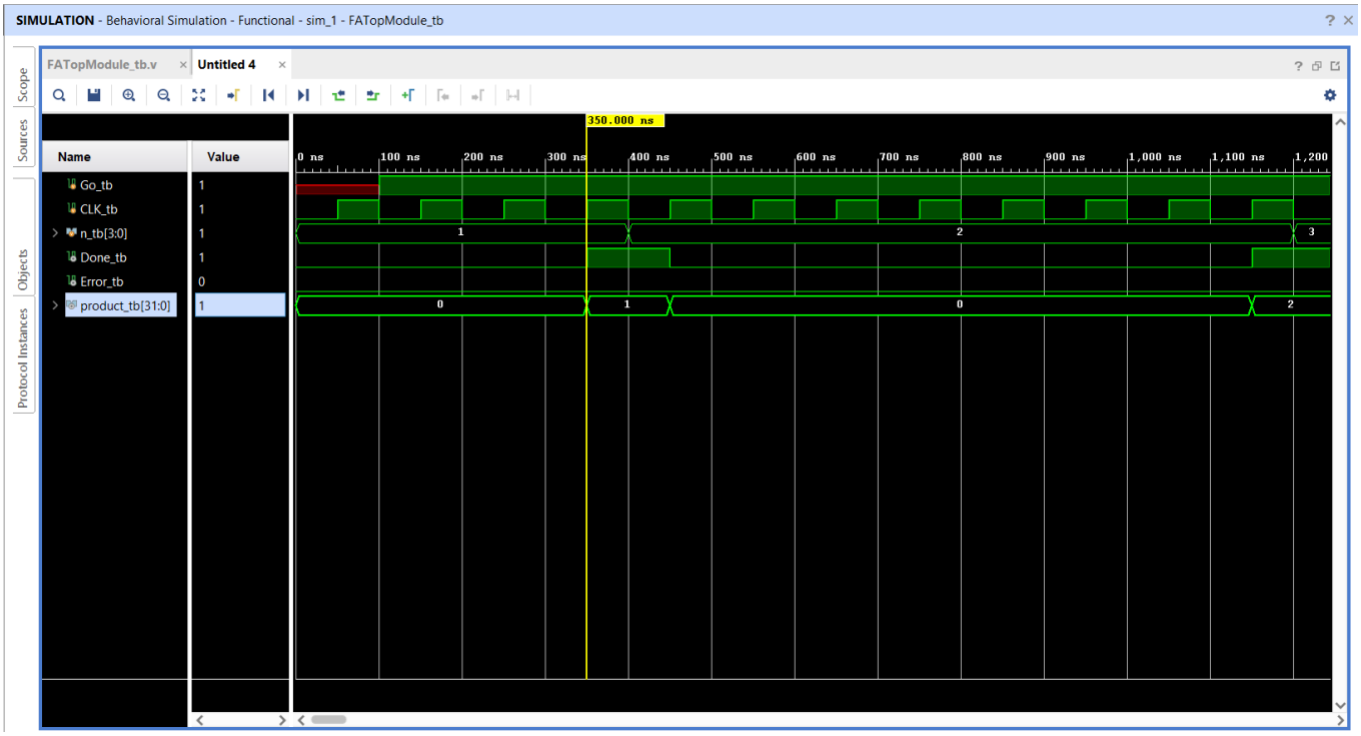
```
`timescale 1ns / 1ps
module Register #(parameter Data_width = 32)(
    input [Data_width - 1:0] D,
    input Load_Reg,
    input CLK,
    output reg [Data_width - 1:0] Q
    );
    always @(posedge CLK)
        begin
            if(Load_Reg)
                Q <= D;
        end
endmodule
```

n. Source Code – Counter.v

```
`timescale 1ns / 1ps
module Counter #(parameter Data_width = 4)(
   input CLK,
   input [Data_width - 1: 0] D,
   input Load_Cnt,
   input EN,
   output reg [Data_width - 1: 0]Q
   );
   always @ (posedge CLK)
   begin
      if(EN)
      begin
         if(Load_Cnt)
            Q <= D;
         else
         begin
            Q <= Q - 1'b1;
         end
      end
   end
endmodule
```

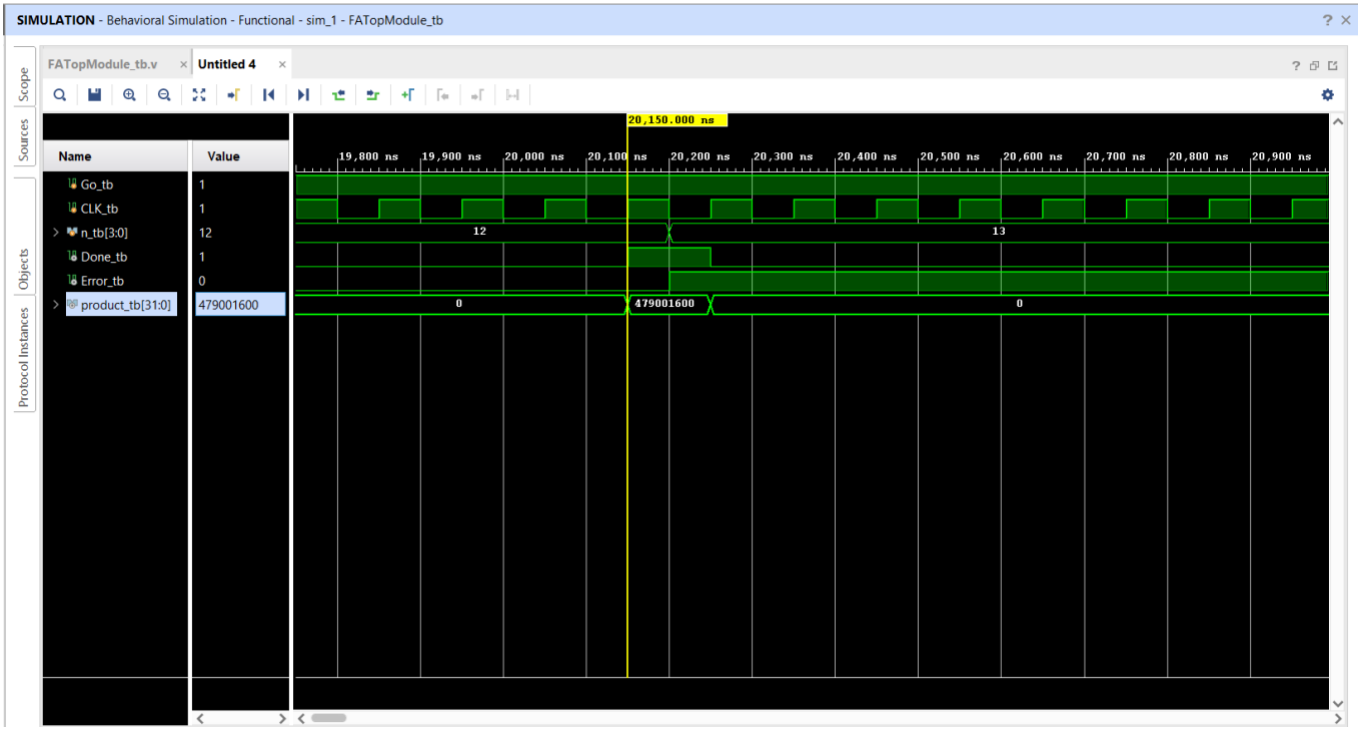
o. Output Waveforms

i. Output for n=1 case with yellow vertical cursor pointed case.



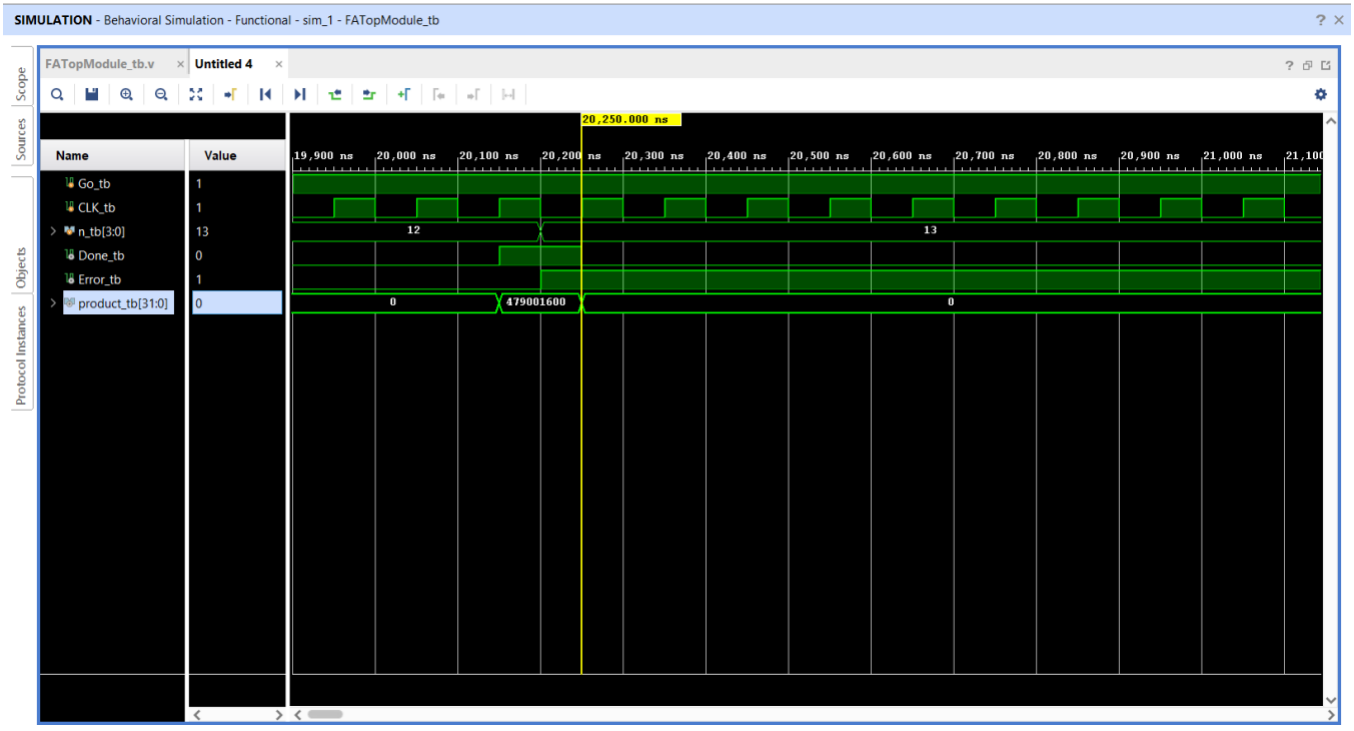
Waveform of the Factorial Accelerator (n = 1 case)

ii. Output for n=12 with yellow vertical cursor pointed case.



Waveform of the Factorial Accelerator (n = 12 case)

iii. Output for n=13 with yellow vertical cursor pointed case (Error case).



Waveform of the Factorial Accelerator (n = 13 case -> Error Case)