San Jose State University
Department of Computer Engineering

CMPE 200 Report

Assignment 1 Report

Title System-Level Design Review

Semester Fall 2022 **Date** 10/05/2022

by

Name HARISH MAREPALLI *(typed)* **SID** 016707314 *(typed)*

ABOUT THE AUTHOR

Harish Marepalli had done bachelor's in Electronics and Communication Engineering at Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, India. He has 3 years of experience as a Systems Engineer in Tata Consultancy Services and his role was software developer. Currently, he is pursuing his master's in Computer Engineering at San Jose State University. His native place is Hyderabad and mother tongue is Telugu. Additionally, he does speak English and Hindi and would love to connect with the people of any region and build network.

LIST OF CONTENTS

1. INTRODUCTION

This activity is used to review system-level design by designing, functionally verifying a digital system for computing the factorial of n. The designed system should start execution upon receiving an external input "Go" and should given an output as a "Done" signal upon completion of the execution. In addition, an "Error" signal should be set when given an input greater than 12.

2. MY GROUP

Name: Student_Team 6
Harish Marepalli – 016707314
Tirumala Saiteja Goruganthu – 016707210

3. GIVEN TASK

a. The task is to understand the given algorithm for computing the factorial of a number.
b. To design the system's datapath using the given building blocks.
c. To draw the two-piece CU-DP system block diagram that shows all the relevant signals.
d. To draw the ASM chart which describes the cycle-by-cycle operations of the datapath.
e. To draw bubble diagram for the "next state logic" (NS) part of the control unit.
f. To construct an output table for the "output logic" part of the control unit.
g. To write Verilog design code for the factorial accelerator, as well as testbench code to functionally verify the design.
h. The factorial accelerator shall calculate and display results upto 12 factorial.

4. GIVEN INFORMATION TO ACHIEVE THE TASK

a. Functional building blocks are given that are to be used in building the datapath of the system.
b. CNT is a down counter with parallel load control and an enable signal.
c. REG is a data register with a load control signal.
d. CMP is a comparator with a GT (Greater-Than) output.
e. MUL is a combinational multiplier for unsigned integers.
f. MUX is a 2 to 1 multiplexer.
g. Only one of each building block given is to be used to build the accelerator's datapath.
h. To capture the inputs that are greater than 12, an additional comparator should be used.

5. STEPS TAKEN TO COMPLETE THE TASK

a. Understood the given information that is already provided and noted the important points.
b. Used online resources to understand how to draw a datapath for any system.
c. Drawn Datapath in Visio tool by using all the blocks that are already provided.
d. Used extra multiplexer instead of buffer to get the output.
e. Drawn Control Unit by carefully giving the input and output signals.
f. Connected the control unit with the datapath to form a CU-DP system.
g. Drawn ASM chart and bubble diagram to represent the algorithm and state transition.
h. Constructed an output table to include output of the states and all the relevant signals.
i. Written the Verilog code for factorial accelerator and testbench using the given information.
j. Verified the output by running the code and checking simulation waveforms.

6. DATAPATH

a. The datapath module contains one down counter (*Figure 5 in appendix*) with 4-bit input to decrement the value of the input. There is a parallel load control and enable signal for it.
b. It contains a comparator (*Figure 6 in appendix*) which takes 4-bit inputs and is used to check if the input is greater than 12 or not.
c. It has a 2-to-1 multiplexer (*Figure 7 in appendix*) with a select signal, 32-bit input, and an input that comes as an output from multiplier (*Figure 8 in appendix*).
d. The multiplexer is connected to a data register (*Figure 9 in appendix*) by giving the output and the register has a load register signal.
e. It contains another comparator which is used to check if the number that is coming from the down counter is greater than 1 or not.
f. The output of the counter is also connected to a multiplier which is used for multiplication operation. It is for unsigned integers.
g. This multiplier takes the input from register also and gives the output to the multiplexer.
h. It contains another multiplexer which is used for output control. It takes input from the register and another 32-bit signal.
i. The signals that are sent as output to the control unit are GT_Inp, GT_Fact, and Factorial output.
j. *Figure 1* shown in appendix is the system's datapath drawn using the above modules.

7. CU-DP SYSTEM

a. The CU-DP system is the factorial generator that contains Control Unit and Data Path connected together to produce the required output.
b. The control unit module contains a Go signal that starts the state machine.
c. It also takes the inputs GT_Inp and GT_Fact from the datapath.
d. At each state, the control unit outputs are changed to control in the modules within the datapath.
e. The system executes when it receives the "Go" input and it outputs a "Done" signal when the execution is completed.
f. An "Error" signal is set when the system detects an input greater than 12.
g. *Figure 2* shown in appendix is the two-piece CU-DP system block diagram that shows all the relevant signals.

8. ASM CHART

a. The Algorithmic State Machine (ASM) chart describes the cycle-by-cycle operations of the datapath.
b. Different block shapes are used to indicate the input, conditional operation etc which are used to indicate the flow of the operation.
c. The system is in idle state at the beginning. This state is denoted as 'S0.'
d. The next step is to check if the input is greater than 12 or not. It executes normally if the input is less than 12.
e. If the input value is greater than it gives and error and goes back to the idle state.
f. Next state is the load control and register state. Here, the down counter and register are passed with the signals Load_cnt and Load_reg respectively. This state is denoted as 'S1.'
g. The state that comes after this is the wait state. This state is denoted as 'S2.'
h. After that the next operation is performed by the comparator.
i. The next is called as Output Enable (OE) and 'Done.' This state is also denoted as 'S3.'

j. After S3 state, the next state is Mult_AND_Dec (Multiplication and Decrement). Here, the value is multiplied for factorial operation and decremented afterwards.
k. *Figure 3* shown in appendix is the ASM chart that shows the steps diagrammatically described above.

9. BUBBLE DIAGRAM

a. State Transition Diagram (STD) or Bubble Diagram is drawn for the Next State (NS) part of the control unit.
b. The states are represented by bubbles.
c. There are 5 states involved in this factorial computation. They are given as below. The operations performed in those states are denoted using short forms.
    a. State S0 => Idle
    b. State S1 => Load_cnt_AND_reg
    c. State S2 => Wait
    d. State S3 => OE_AND_Done
    e. State S4 => Mult_AND_Dec
d. *Figure 4* shown in appendix is the State Transition Diagram.

10. OUTPUT TABLE

a. Output for the output logic of the control unit is recorded in a table.
b. At state S0:
    Sel = 1
    Load_cnt = 1
    EN = 1
    Load_reg = 0
    OE = 0
    Done = 0
    Error = NA
c. At state S1:
    Sel = 1
    Load_cnt = 1
    EN = 1
    Load_reg = 1
    OE = 0
    Done = 0
    Error = NA
d. At state S2:
    Sel = 0
    Load_cnt = 0
    EN = 0
    Load_reg = 0
    OE = 0
    Done = 0
    Error = NA
e. At state S3:
    Sel = 0
    Load_cnt = 0
    EN = 0
    Load_reg = 0
    OE = 1
    Done = 1
    Error = NA

f. At state S4:
    Sel = 0
    Load_cnt = 0
    EN = 1
    Load_reg = 1
    OE = 0
    Done = 0
    Error = NA
g. *Table 1* shown in appendix is the Output table for control unit shown in tabular format.

11. VERILOG CODE

a. A total of 9 files are used to write the Verilog code.
b. The structure of the Verilog code is given below:
    Design Sources:
    i. FactorialModule (FactorialModule.v)
        1. DP: FactDatapath (FactDatapath.v)
            - CMP_INP: Comparator (Comparator.v)
            - CMP_FACT: Comparator (Comparator.v)
            - COUNTER: Counter (Counter.v)
            - MUL: Multiplier (Multiplier.v)
            - MUX1: Multiplexer (Multiplexer.v)
            - MUX2: Multiplexer (Multiplexer.v)
            - REG: Register (Register.v)
        2. CU: FactControlpath (FactControlpath.v)
    ii. Register (Register.v)

    Simulation Sources:
        sim_1
            FactTestBench (FactTestBench.v)
                DUT: FactorialModule (FactorialModule.v)
                …
                …
c. A short description of the files is as follows:
    i. FactorialModule.v – Top-level module for the system
    ii. FactDatapath.v – Data Path for the Factorial Generator
    iii. Comparator.v – Performs comparing operation
    iv. Counter.v – Performs decrement operation.
    v. Multiplier.v – Performs multiplication operation.
    vi. Multiplexer.v – It involves MUX operation.
    vii. Register.v – This is for loading value in a register.
    viii. FactControlPath.v – Control Unit for the Factorial Generator.
    ix. FactTestBench.v – Test Bench for the system.
d. Source code can be found in the Appendix.

12. DISCUSSION SECTION

a. This part of the assignment can be seen as having two divisions.
b. First division is to design and test the control unit and data path for computing the factorial of n.
c. Second division is to design, test, and build the fully integrated factorial generator that consists of both the control unit and data path.
d. The system starts its execution when it receives the input signal 'Go.'
e. The system outputs a 'Done' signal when the execution is completed.
f. When the input is greater than 12, an 'Error' signal will be set.

The control unit is a combination of Moore and Mealy finite-state machine.

What is a Moore State Machine?
This is a synchronous sequential machine. It has an output that depends only on its present state.
It can be represented in the form of equation as below:
Next State = f (Present State, Inputs)
Output = g (Present State)

What is a Mealy State Machine?
Mealy State Machine is the one in which the output depends on both its present state and also its inputs.
It can be represented in the form of equation as below:
Next State = f (Present State, Inputs)
Output = g (Present State, Inputs)

Are conditional boxes in an ASM chart used only for mealy sequential circuits?
Yes, conditional output boxes are used only for mealy circuits whereas state boxes are used for Moore sequential circuits. Decision boxes are used for both types of circuits. As per the definition, a Moore circuit's output depends on the present state only whereas for a Mealy circuit, output depends on the present state as well as present input.

How to choose between Moore and Mealy State Machine?
Generally, Mealy machines have fewer states. However, fewer states don't always mean simpler to implement.

Moore machines may be safer to use because their status gets changed on the clock edge, whereas Mealy machines are faster, because the state is dependent on the input.

Although a Moore machine can sample inputs or change output on clock edges, I would think that a Mealy machine design could be made similarly.
While some might argue that such flops added to a Mealy machine would constitute part of the state machine's state, I would assume that it is more helpful to regard a state machine as only encompassing those aspects of state which are generated by the machine and feed back into it.

One advantage of the Moore machine is that it can be implemented in a Look up table or SRAM memory.
If the implementation is on an FPGA, using it might become easy.

To test the logic for the factorial generator, a simulation test bench is created to test the module. All combinations of the inputs are tested. The clock is then ticked through all states and the outputs are sent to the MUX output port. The Done signal goes high when the operation is complete. If an error occurs, the error flag goes high. The expected outputs are verified at each step of the process.

Figure 10 shows the simulation waveform factorial outputs for the inputs 1 and 2.
Figure 11 shows the simulation waveform factorial outputs for the inputs 4 and 5.
Figure 12 shows the simulation waveform error for the inputs 13 and 14.

13. COLLABORATION SECTION

1. Understood how to draw data path and control path by collaborating with each other.
2. Worked together on the data path and control path, and eventually prepared them in Microsoft Vision tool.
3. Collaborated in designing the FSM while preparing the corresponding ASM chart and the bubble diagram using the Microsoft Vision tool.
4. Worked together to construct the leaf level modules in the Vivado software using Verilog code.
5. By working together, we have written the test bench to verify a bunch of values and observed the output waveforms.

14. 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

Data Path:

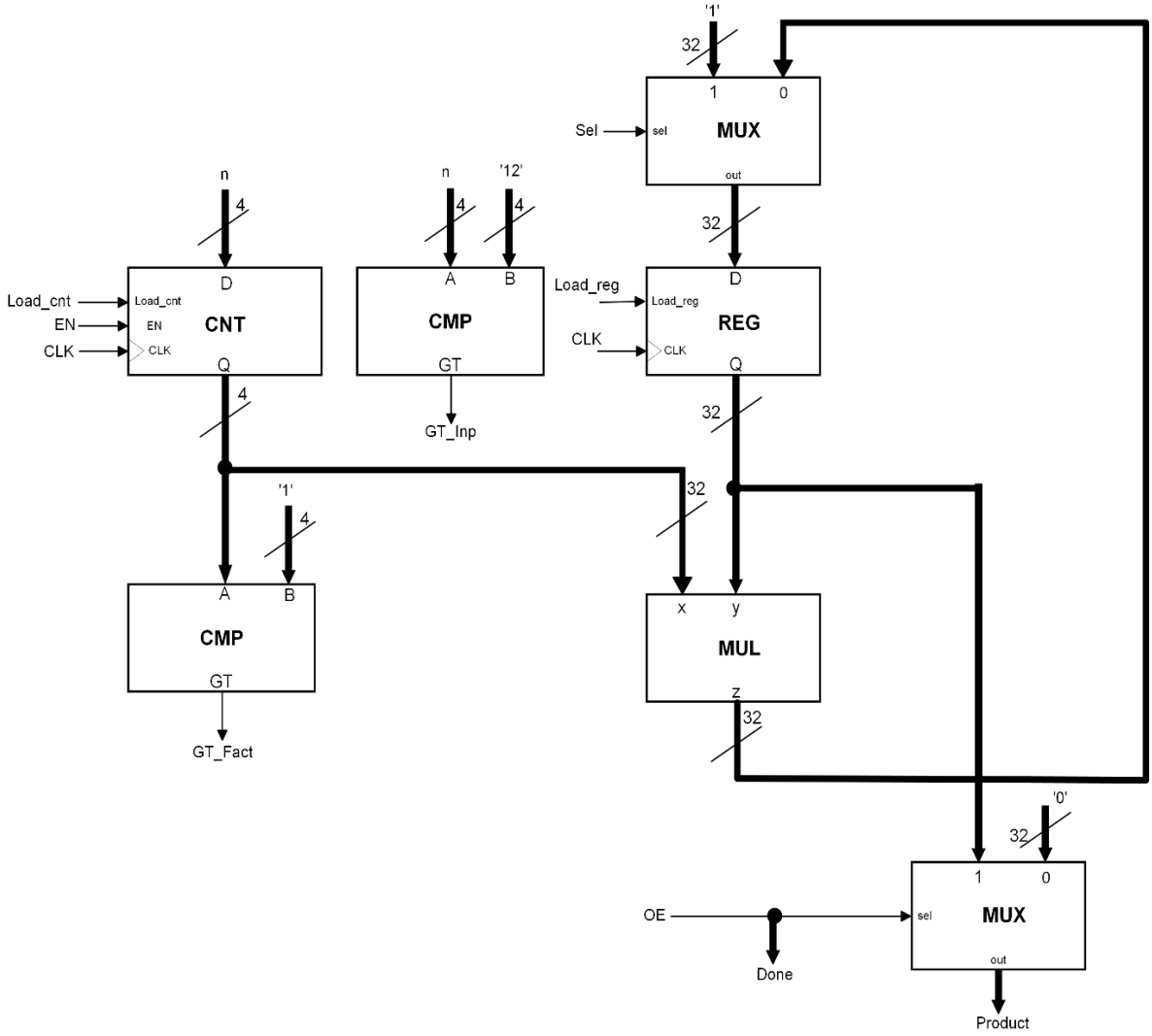

Fig. 1: Data path

CU-DP:

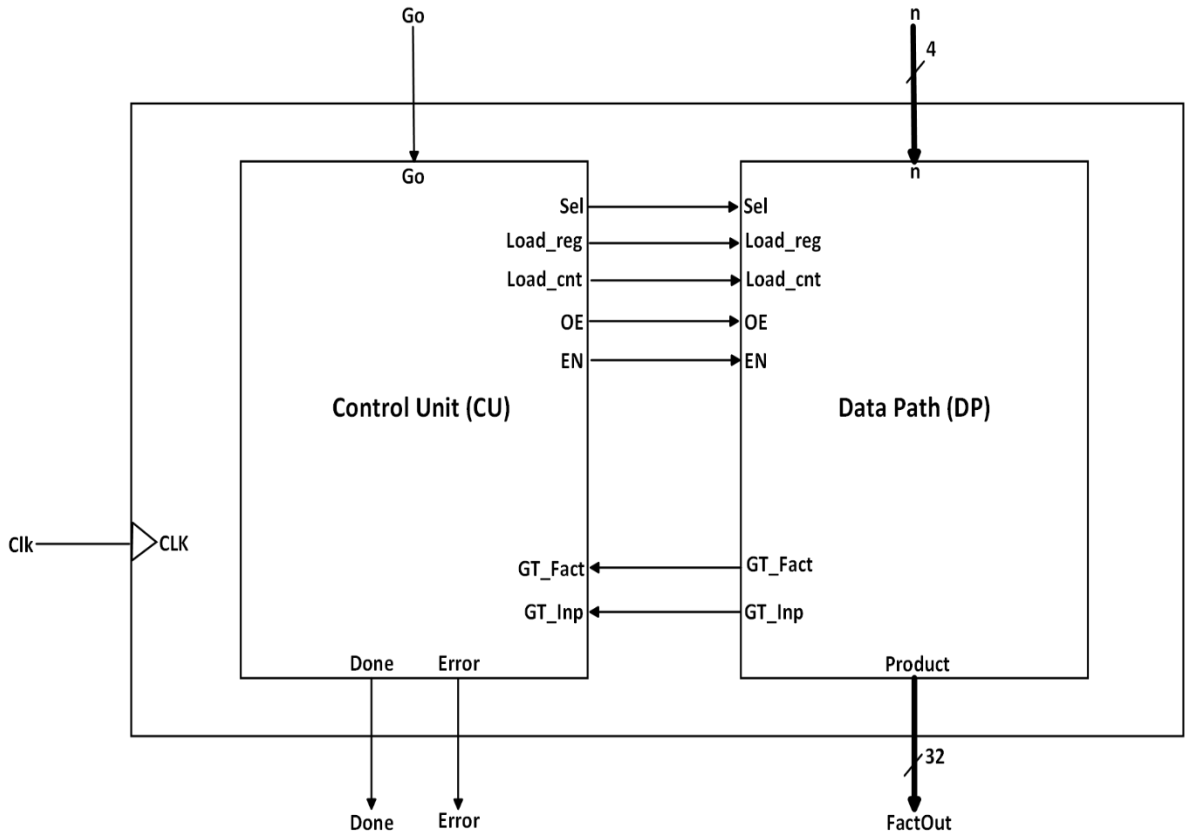

Fig.2: Two-piece CU-DP

ASM Chart:

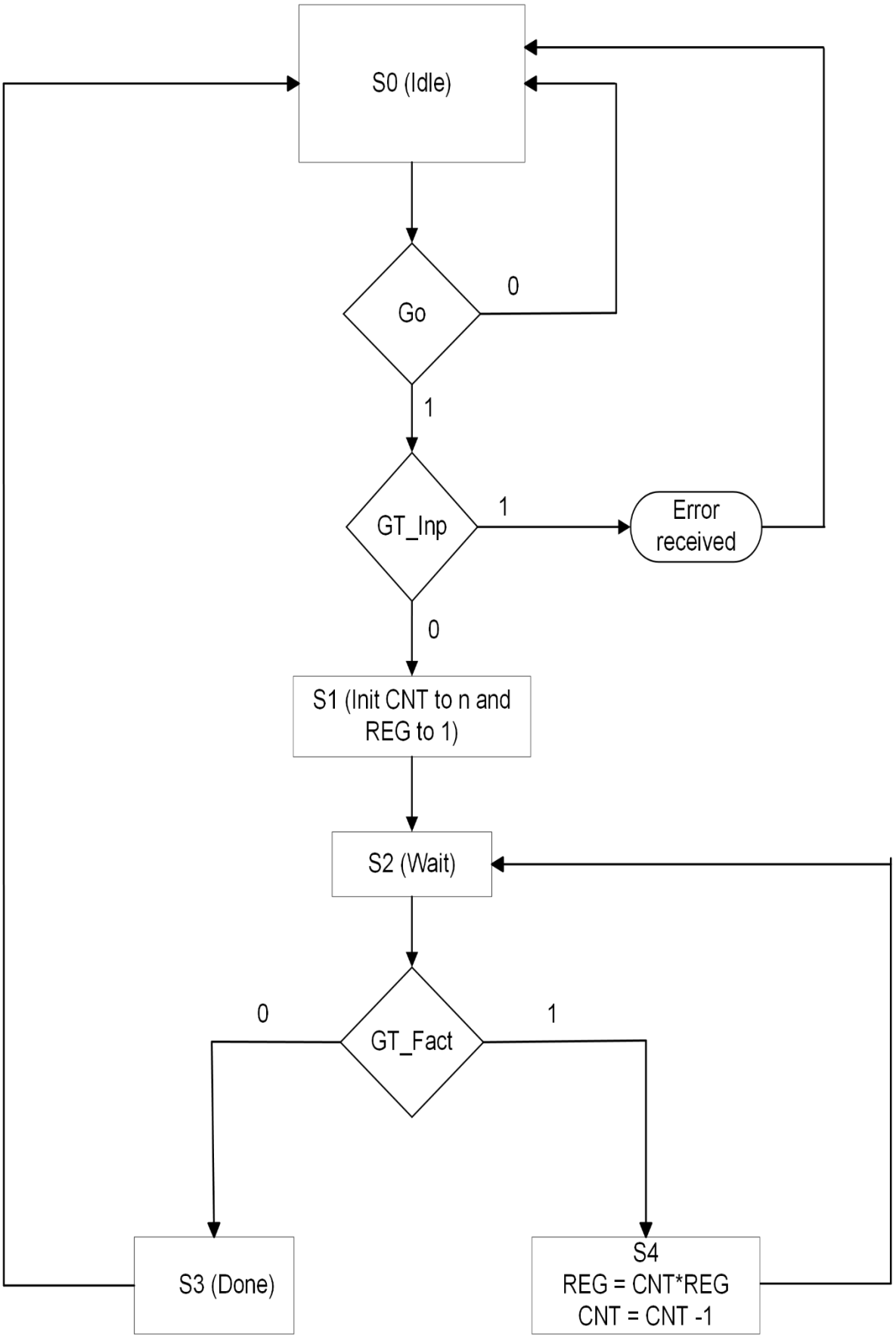

Fig. 3: ASM Chart

Bubble Diagram / State Transition Diagram:

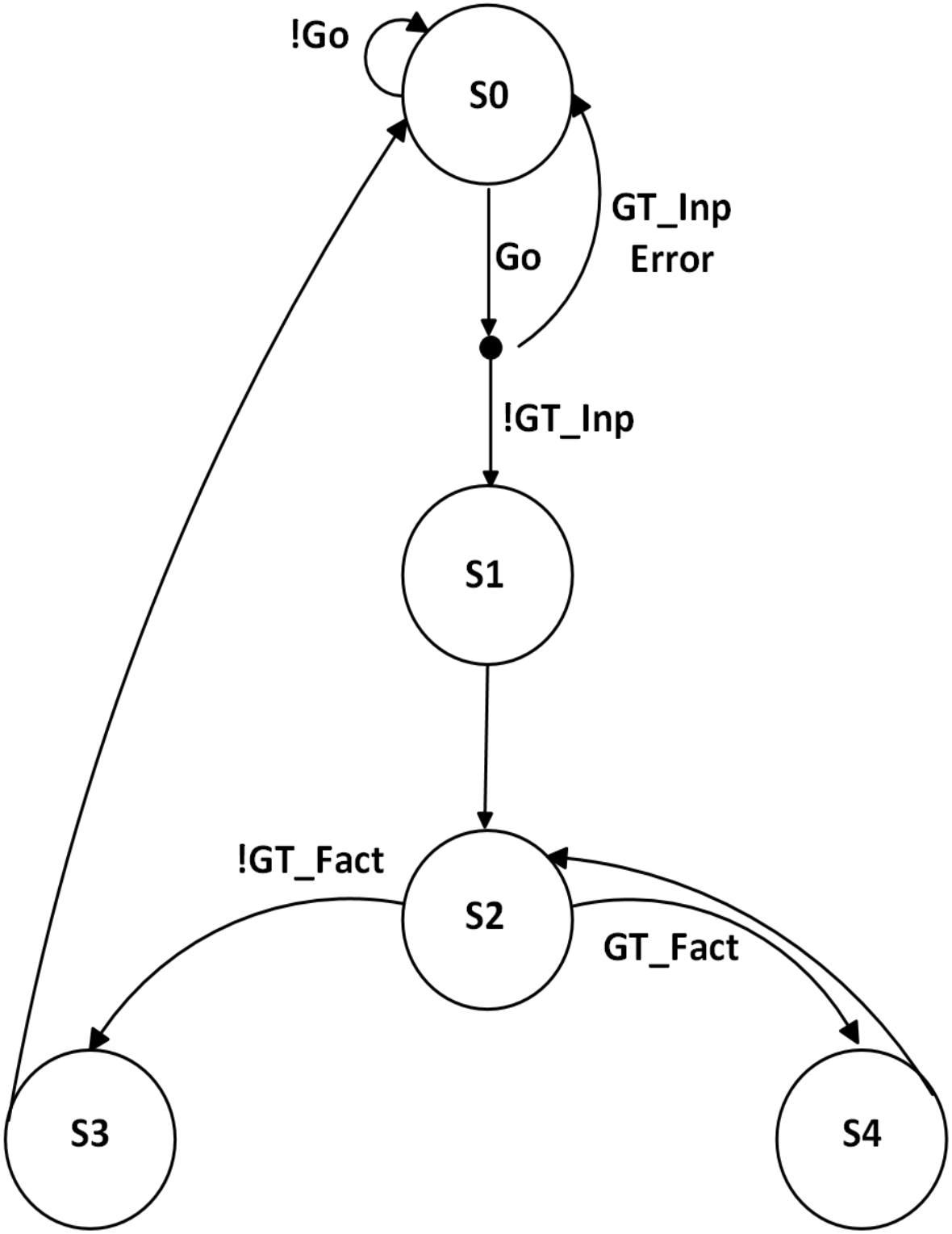

Fig. 4: Bubble Diagram / State Transition Diagram

Output Table:

State	Outputs						
	Sel	Load_cnt	EN	Load_reg	OE	Done	Error
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	-

Table 1: Output Table

Modules:

- **Counter**

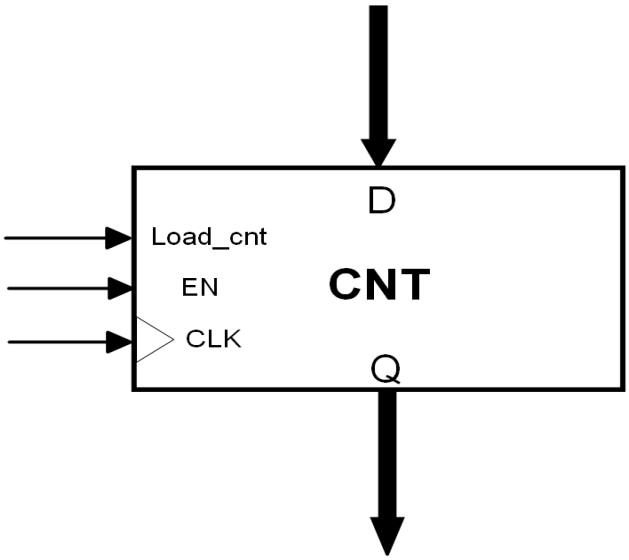

Fig. 5: Counter

- **Comparator**

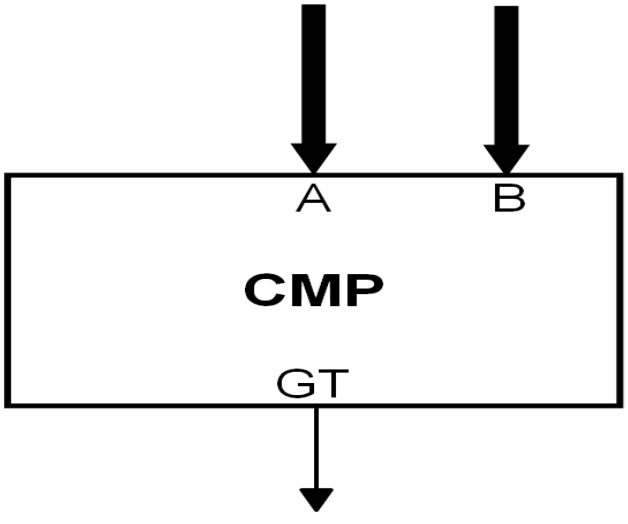

Fig. 6: Comparator

- **2-to-1 Multiplexer**

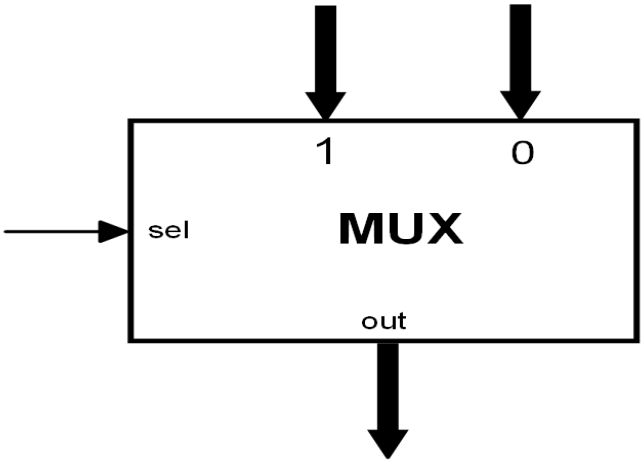

Fig. 7: Multiplexer

- **Multiplier**

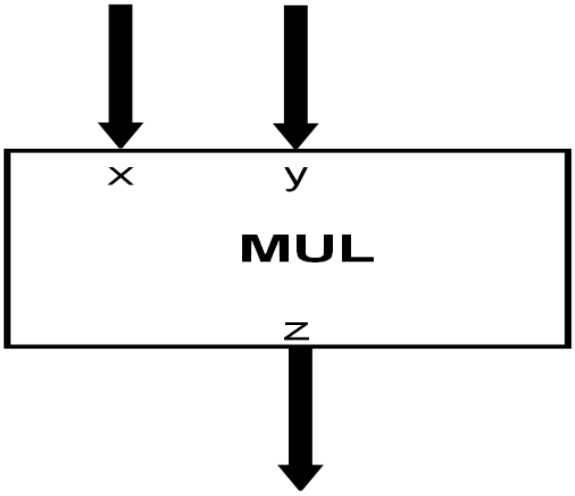

Fig. 8: Multiplier

- **Register**

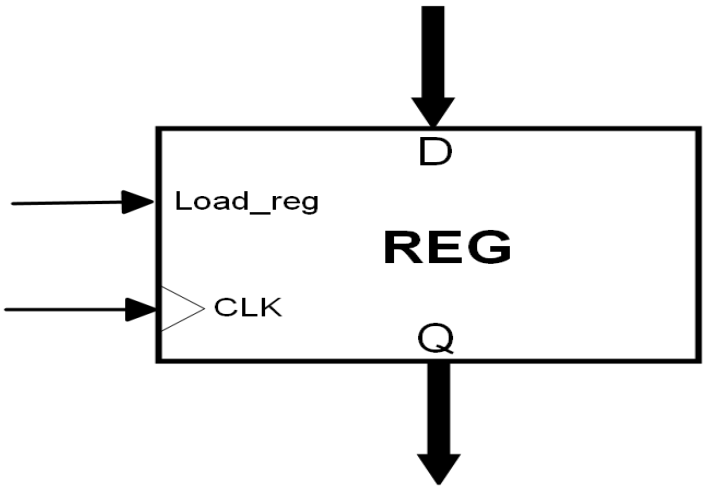

Fig. 9: Register

SOURCE CODE:
-> FactDatapath.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 08:47:36 PM
// Design Name:
// Module Name: FactDatapath
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FactDatapath #(parameter Data_Width = 32)(
    input [3:0] n,
    input Sel,
    input Load_reg,
    input Load_cnt,
    input OE,
    input EN,
    input CLK,
    output GT_Fact,
    output GT_Inp,
    output [Data_Width - 1:0] Product
    );
    wire [3:0] CNT_Out;
    wire [Data_Width - 1:0] MUX_Out;
    wire [Data_Width - 1:0] REG_Out;
    wire [Data_Width - 1:0] MUL_Out;

    Comparator #(4) CMP_INP (.A(n), .B(4'd12), .GT(GT_Inp));
    Comparator #(4) CMP_FACT (.A(CNT_Out), .B(4'b1), .GT(GT_Fact));
    Counter #(4) COUNTER
(.D(n), .Q(CNT_Out), .Load_cnt(Load_cnt), .EN(EN), .CLK(CLK));
```

```
    Multiplier #(32) MUL (.x({28'b0,CNT_Out}), .y(REG_Out), .z(MUL_Out));
    Multiplexer #(32) MUX1 (.inp2(32'b1), .inp1(MUL_Out), .Sel(Sel), .Out(MUX_Out));
    Multiplexer #(32) MUX2 (.inp1(32'b0), .inp2(REG_Out), .Sel(OE), .Out(Product));
    Register #(32) REG (.D(MUX_Out), .Q(REG_Out), .Load_reg(Load_reg), .CLK(CLK));
endmodule
```

-> FactControlpath.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 08:51:32 PM
// Design Name:
// Module Name: FactControlpath
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FactControlpath(
    input Go,
    input CLK,
    input GT_Fact,
    input GT_Inp,
    output reg Sel,
    output reg Load_reg,
    output reg Load_cnt,
```

```
output reg OE,
output reg EN,
output Done,
output Error
);
//Current and Next State Registers
reg [2:0] CS = 0, NS;
reg Error_Internal = 0, Done_Internal = 0;

assign Error = GT_Inp;
assign Done = Done_Internal;

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

//Next State Logic (combinational) based on State Transition Diagram
always @ (CS, Go)
begin
   case (CS)
      S0:
      case({Go, GT_Inp})
         2'b11: {NS, Error_Internal} <= {S0, 1'b1};
         2'b10: {NS, Error_Internal} <= {S1, 1'b0};
         2'b0?: {NS, Error_Internal} <= {S0, 1'b0};
         default: NS = S0;
      endcase
   S1: NS <= S2;
   S2: NS <= GT_Fact ? S4 : S3;
   S3: NS <= S0;
   S4: NS <= S2;
   endcase
end

//State Register (sequential)
always @ (posedge CLK)
```

```
      CS = NS;

   //Output Logic
   always @ (CS)
   begin
      case (CS)
         S0:
         begin
            {Sel, Load_cnt, EN, Load_reg, OE, Done_Internal} <= 6'b1_1_1_0_0_0;
         end
         S1:
         begin
            {Sel, Load_cnt, EN, Load_reg, OE, Done_Internal} <= 6'b1_1_1_1_0_0;
         end
         S2:
         begin
            {Sel, Load_cnt, EN, Load_reg, OE, Done_Internal} <= 6'b0_0_0_0_0_0;
         end
         S3:
         begin
            {Sel, Load_cnt, EN, Load_reg, OE, Done_Internal} <= 6'b0_0_0_0_1_1;
         end
         S4:
         begin
            {Sel, Load_cnt, EN, Load_reg, OE, Done_Internal} <= 6'b0_0_1_1_0_0;
         end
      endcase
   end
endmodule
```

-> FactorialModule.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 08:39:51 PM
// Design Name:
// Module Name: FactorialModule
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FactorialModule(
    input Go,
    input CLK,
    input [3:0] n,
    output Done,
    output Error,
    output [31:0] FactOut
    );
    wire Sel, Load_reg, Load_cnt, OE, EN;
    wire GT_Inp, GT_Fact;

    FactDatapath DP (
```

```
        .CLK(CLK),
        .EN(EN),
        .n(n),
        .Sel(Sel),
        .Load_reg(Load_reg),
        .Load_cnt(Load_cnt),
        .OE(OE),
        .GT_Inp(GT_Inp),
        .GT_Fact(GT_Fact),
        .Product(FactOut)
    );

    FactControlpath CU (
        .EN(EN),
        .Go(Go),
        .CLK(CLK),
        .Sel(Sel),
        .Load_reg(Load_reg),
        .Load_cnt(Load_cnt),
        .OE(OE),
        .GT_Fact(GT_Fact),
        .GT_Inp(GT_Inp),
        .Done(Done),
        .Error(Error)
    );
Endmodule
```

-> FactTestBench.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 11:27:01 PM
// Design Name:
```

```
// Module Name: FactTestBench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FactTestBench;
reg Go_tb, CLK_tb;
reg [3:0] n_tb;
wire Done_tb, Error_tb;
wire [31:0] FactOut_tb;

FactorialModule FM (
    .n(n_tb),
    .Go(Go_tb),
    .CLK(CLK_tb),
    .Done(Done_tb),
    .Error(Error_tb),
    .FactOut(FactOut_tb)
);

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

```
        #50;
    end
endtask

initial
begin
    $display("Factorial Test Starts Here");
    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, FactOut_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
```

-> Counter.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 09:00:01 PM
// Design Name:
// Module Name: Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Counter #(parameter Data_Width = 4)(
    input [Data_Width - 1:0]D,
    input Load_cnt,
    input EN,
    input CLK,
    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
```

-> Comparator.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 09:00:01 PM
// Design Name:
// Module Name: Comparator
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module Comparator #(parameter Data_Width = 4)(
  input[Data_Width - 1:0] A,
  input[Data_Width - 1:0] B,
```

```
    output reg GT
    );
    always@(A or B)
        begin
            GT <= 1'b0;
            if(A>B)
                GT <= 1'b1;
        end
endmodule
```

-> Multiplexer.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 09:00:01 PM
// Design Name:
// Module Name: Multiplexer
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module Multiplexer #(parameter Data_Width = 32)(
    input [Data_Width - 1:0] inp1,
```

```
    input [Data_Width - 1:0] inp2,
    input Sel,
    output reg [Data_Width - 1:0] Out
    );
    always @ (inp1, inp2, Sel)
        begin
            if(Sel)
                Out <= inp2;
            else
                Out <= inp1;
        end
endmodule
```

-> Multiplier.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 09:00:01 PM
// Design Name:
// Module Name: Multiplier
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
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
```

-> Register.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/27/2022 09:00:01 PM
// Design Name:
// Module Name: Register
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



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
```

SIMULATION WAVEFORMS:

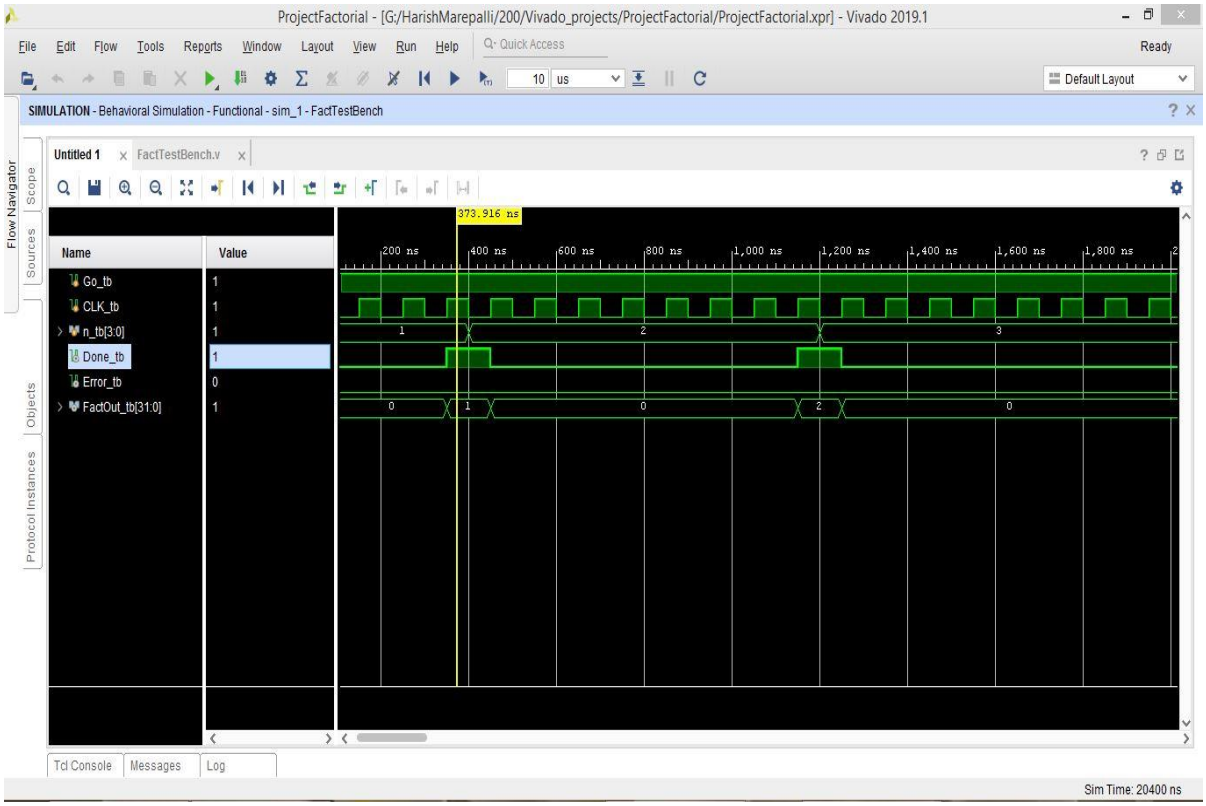

Fig. 10: Output waveform showing 1 and 2 factorial outputs

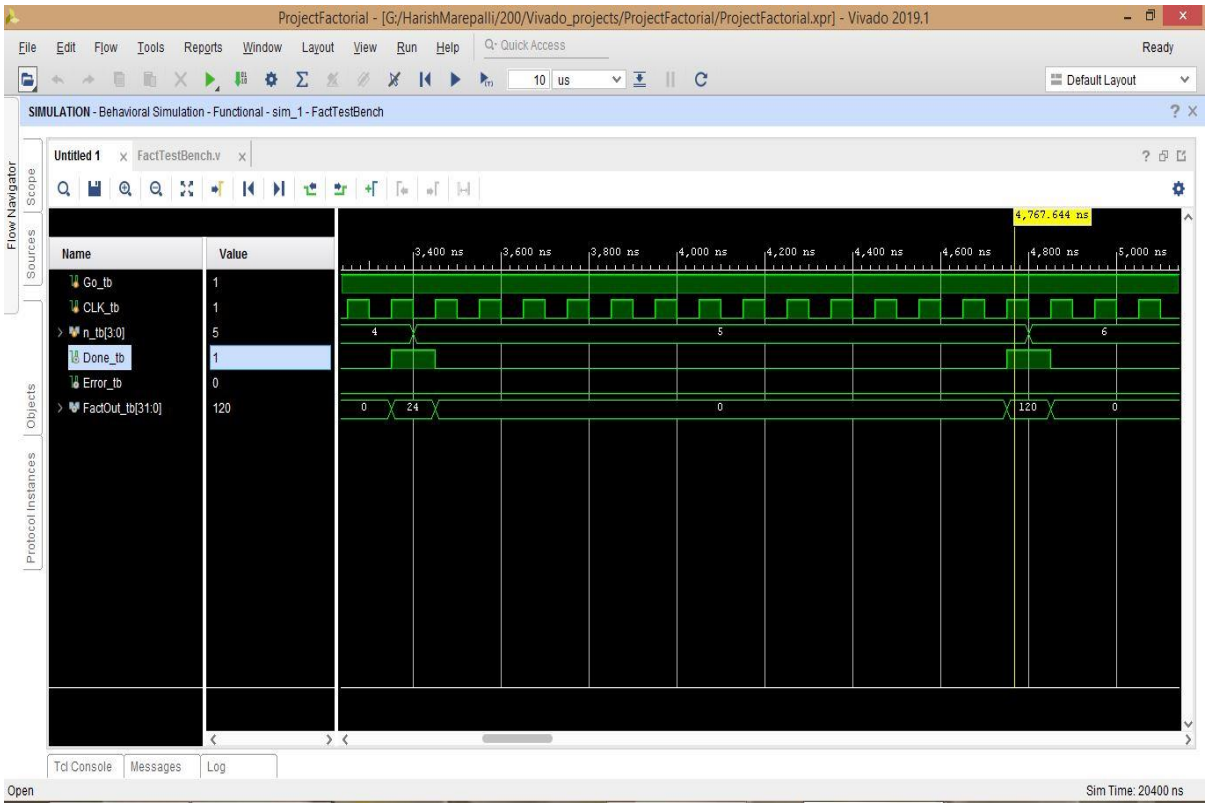

Fig. 11: Output waveform showing 4 and 5 factorial outputs

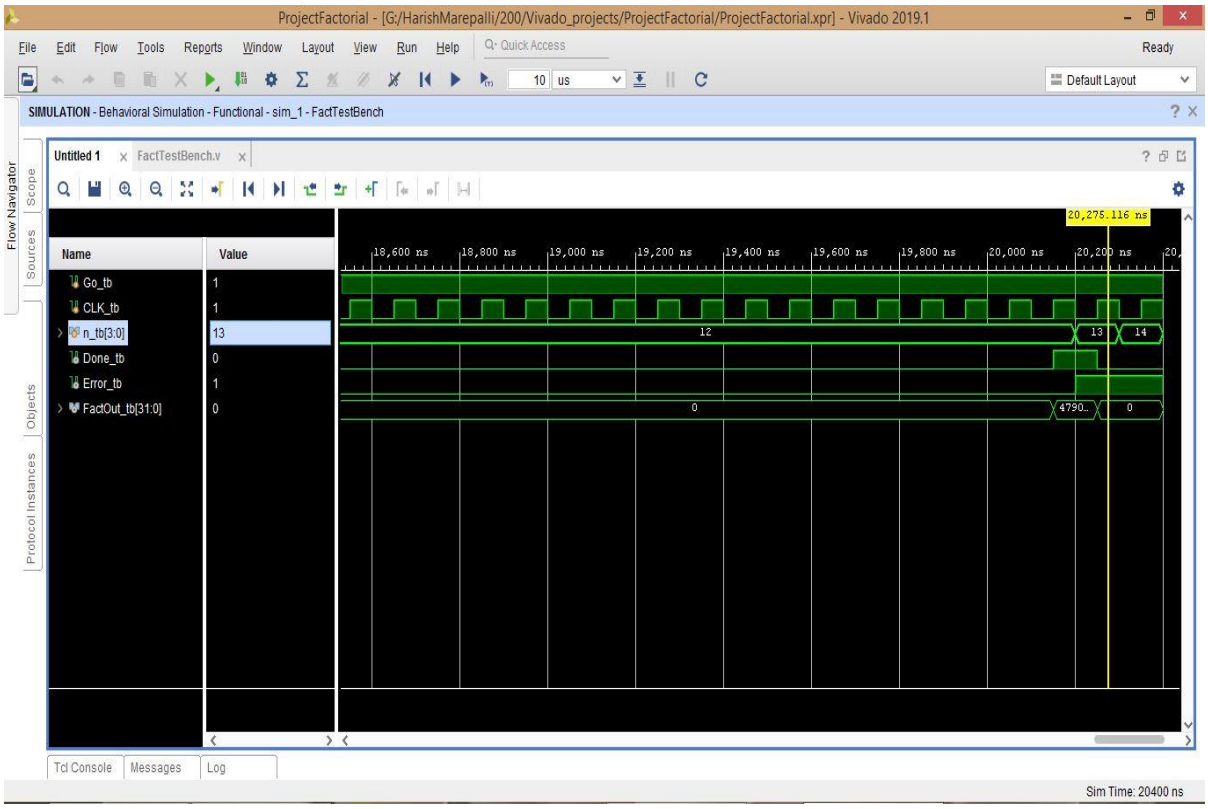

Fig 12. Error for input 13 and 14