San Jose State University
Department of Computer Engineering

CMPE 200 Report

Assignment 1 Report

Title System-Level Design Review

Semester: Fall 2023 **Date:** 09/16/2023

by

Name: Sai Kashyap Kurella **SID:** 016018925

FACTORIAL SYSTEM TOP LEVEL

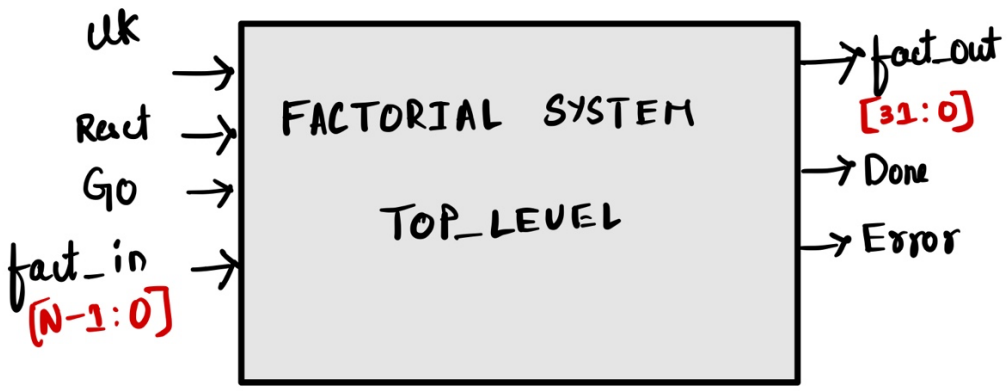


Fig.1 Top level view of Factorial System

SYSTEM DATA PATH

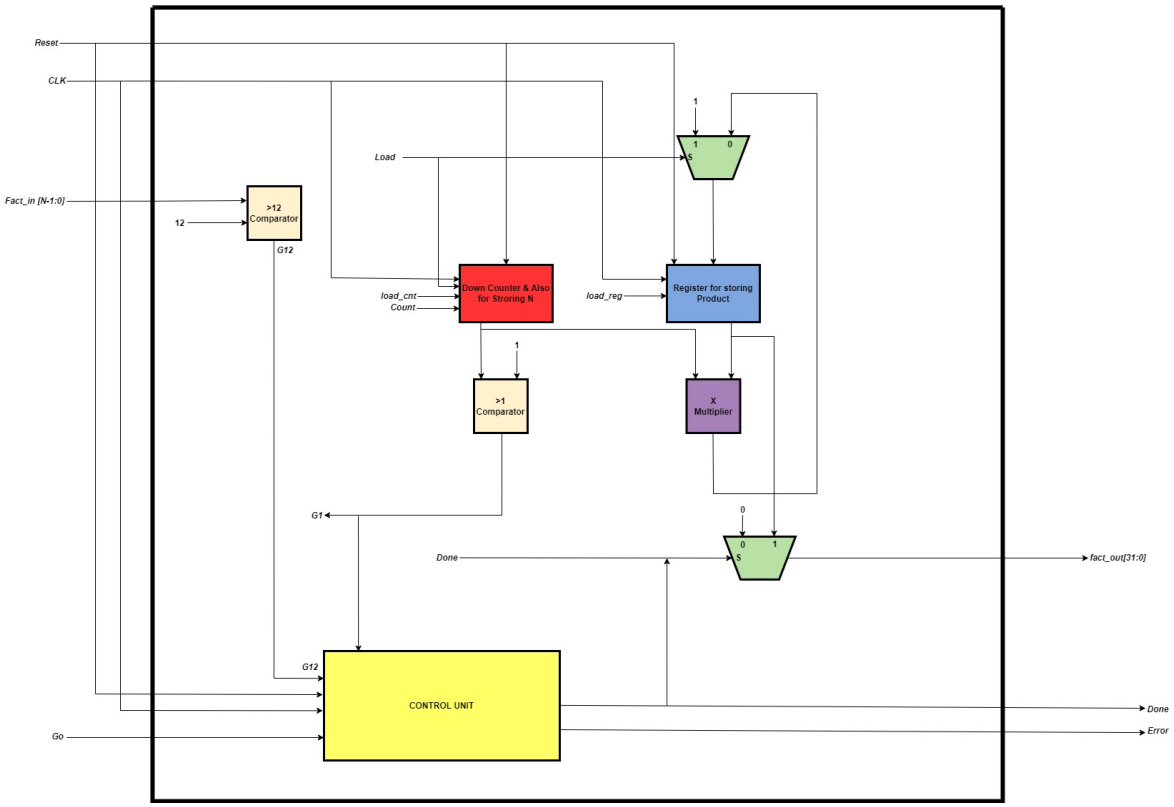


Fig.2 System Data Path with Control unit

Finite State Machine for Control Unit

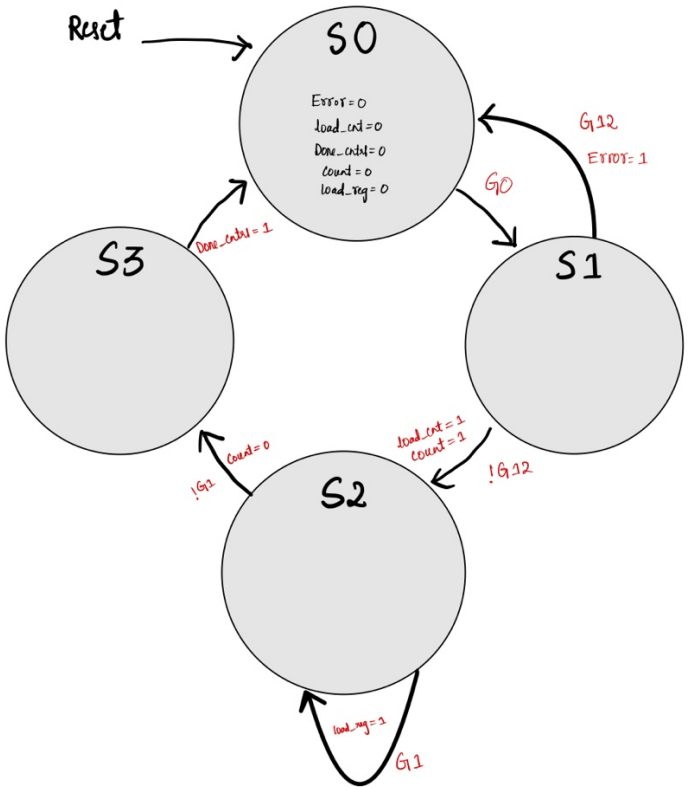


Fig.3 State Transitions of Control unit

ASM chart for Control Unit

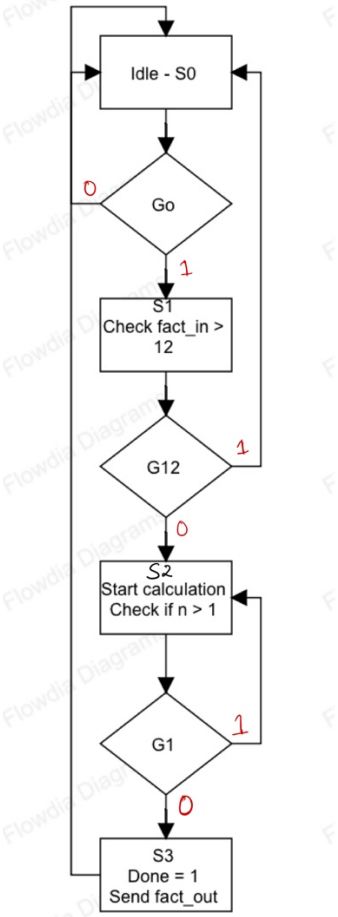


Fig.4 State transitions in ASM chart

Verilog Codes of Individual Modules

Counter

```
module CNT # (parameter N=4)
(
input clk,
input reset,
input en,
input load_cnt,
input [N-1:0] load_value,
output[N-1:0] count_q
);
reg[N-1:0] load_ff;
```

```
//Store the load Value whenever load_cnt is seen
always@(posedge clk or posedge reset)
	begin
		if(reset)
			begin
			load_ff <= 4'h0;
			end
		else if(load_cnt)
			begin
			load_ff <= load_value;
			end
		else
			begin
			load_ff <= load_ff;
			end
	end
reg[3:0] count_ff;
reg[3:0] nxt_count;

always@(posedge clk or posedge reset)
	begin
		if(reset)
			begin
			count_ff <= 4'hF;
			end
		else
			begin
				if(en)
					begin
					count_ff <= nxt_count;
					end
				else
					begin
					count_ff <= count_ff;
					end
			end
	end

assign nxt_count = load_cnt ? load_value : (count_ff == 4'h0) ? load_ff : count_ff - 4'h1;

//assign nxt_count = (count_ff == 4'h0) ? load_ff : count_ff - 1'h1;

assign count_q = count_ff;

endmodule
```

Comparator

```
module CMP #(parameter N = 4)
(
input [N-1:0] A,
input [N-1:0] B,
output G
);

assign G = (A>B) ? 1'b1: 1'b0;

endmodule
```

Multiplier

```
module MUL(input[3:0] A, input[31:0] B, output[31:0] O);

assign O = A*B;

endmodule
```

Multiplexer

```
module MUX #(parameter N =1 )
(
input [31:0] A,
input [31:0] B,
input S,
output [31:0] Y
);
assign Y = (S) ? B : A ;
endmodule
```

Data register with a load control signal

```
module REG #(parameter N = 4)
(
input clk,
input reset,
input load_reg,
input [31:0] reg_d,
output [31:0] reg_q
);
```

```
reg[31:0] store_reg;

//Sequential Block
always@(posedge clk or posedge reset)
	begin
		if(reset)
			begin
			store_reg <= 32'h1;
			end

		else
			begin
				if(load_reg)
					begin
					store_reg <= reg_d;
					end
				else
					begin
					store_reg <= 32'h1;
					end
			end

		end

//Combinational Block
assign reg_q = store_reg;

endmodule
```

Control Unit

```
module Control

(

input clk,

input reset,

input Go,

input G1,

input G12,
```

```
output reg Error,
output reg load_cnt,
output reg Done_cntrl,
output reg load_reg,
output reg count
);
parameter S0 = 2'b00, S1 = 2'b01,S2 =2'b10,S3=2'b11;
reg[1:0] present_state;
reg[1:0] next_state;
//Sequential Part
always@(posedge clk or posedge reset)
begin
        if(reset)
                begin
                present_state <= S0;
                end
        else
                begin
                present_state <= next_state;
                end
end
//Combinational Part
always@(*)
```

```
begin
 case(present_state)
        S0 :    begin                       // Initial Stage
                        Error = 1'b0;
                        load_cnt = 1'b0;
                        Done_cntrl = 1'b0;
                        count = 1'b0;
                        load_reg =1'b0;
                if(Go)
                        begin
                        next_state = S1;
                        end
                else
                        begin
                        next_state = S0;
                        end
                end

        S1 :    begin                       // Check if input is > 12
                if(G12)
                        begin
                        next_state = S0;
                        Error = 1'b1;
```

```
                end
            else
                begin
                next_state = S2;
                load_cnt = 1'b1;
                count = 1'b1;
                end
            end
        S2 :  begin                         //Calculation Stage
            load_cnt = 1'b0;
            if(G1)
                begin
                next_state = S2;
                load_reg = 1'b1;
                end
            else
                begin
                next_state = S3;
                count = 1'b0;
                end
end
        S3 :  begin                         //Done
                next_state = S0;
```

```
                    Done_cntrl = 1'b1;
                end
 endcase
end
endmodule
```

Top Level module

```
`include "CMP.v"
`include "CNT.v"
`include "MUL.v"
`include "REG.v"
`include "MUX.v"
`include "Control.v"

module Top #(parameter N = 4) (
input clk,
input reset,
input Go,
input [N-1:0] fact_in,
output [31:0] fact_out,
output Done,
output Error
);
wire load_cnt; //Control
```

```
wire load_reg; //Control

wire count; //Control

wire[N-1:0] count_q;

wire G1; //Control

wire[31:0] reg_d;

wire[31:0] reg_q;

wire[31:0] O;

wire G12; //Control

wire done; //Control

//Comparator-1 (for checking >1)

CMP c1(.A(count_q),.B(4'h1),.G(G1));




//Counter

CNT
cnt1(.clk(clk),.reset(reset),.en(count),.load_cnt(load_cnt),.load_value(fact_in),.count_q(count_q)
);

//Multiplier

MUL mul1(.A(count_q),.B(reg_q),.O(O));

//Data register with a load control signal

REG reg1(.clk(clk),.reset(reset),.load_reg(load_reg),.reg_d(reg_d),.reg_q(reg_q));

//Multiplexer-1

MUX mux1(.A(O),.B(32'h1),.S(load_cnt),.Y(reg_d));

//Multiplexer-2 (Buffer)
```

```
MUX mux2(.A(32'h0),.B(reg_q),.S(done),.Y(fact_out));

//Comparator-2 (for checking >12)

CMP c2(.A(fact_in),.B(4'hC),.G(G12));

//Controller

Control
ctrl(.clk(clk),.reset(reset),.Go(Go),.G1(G1),.G12(G12),.Error(Error),.load_cnt(load_cnt),.load_reg(load_reg),.Done_cntrl(done),.count(count));

assign Done = done;

endmodule
```

Top Level Test Bench

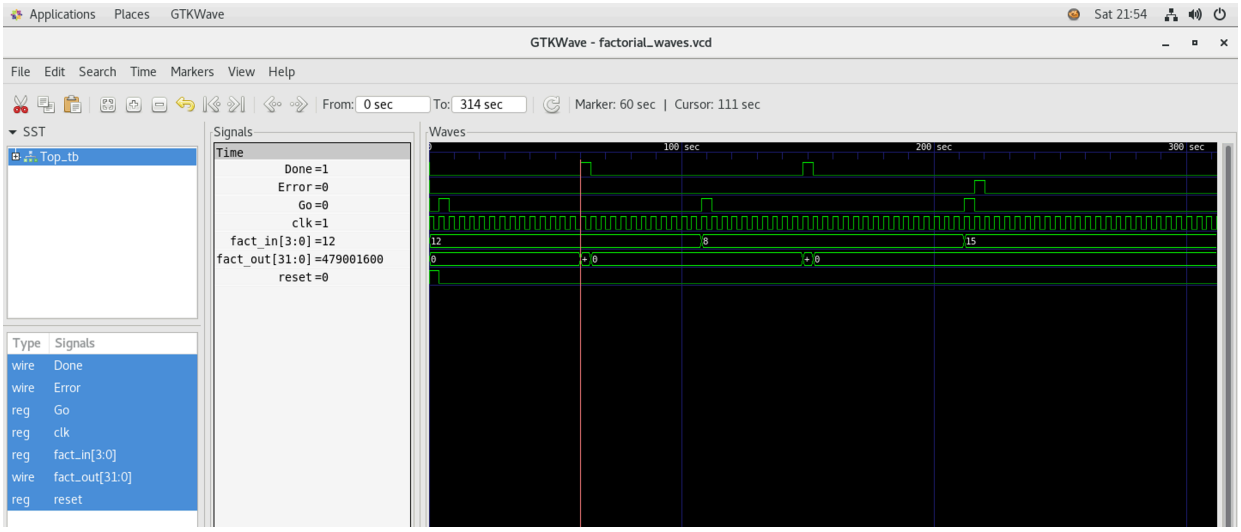
```
module Top_tb#(parameter N = 4)();

reg clk;

reg reset;

reg Go;

reg [N-1:0] fact_in;

wire[31:0] fact_out;

wire Done;

wire Error;
```

```
Top
t1(.clk(clk),.reset(reset),.Go(Go),.fact_in(fact_in),.fact_out(fact_out),.Done(Done),.Error(Error)
);

// Clock Generation

always

begin

        clk = 1'b1;

        #2;

        clk = 1'b0;

        #2;

end

// Stimulus

initial

begin

        reset = 1'b1;

        fact_in = 4'hC;

        Go = 1'b0;

        @(posedge clk);

        reset = 1'b0;

        Go = 1'b1;

        @(posedge clk);

        Go = 1'b0;

        #100;

        fact_in = 4'h8;
```

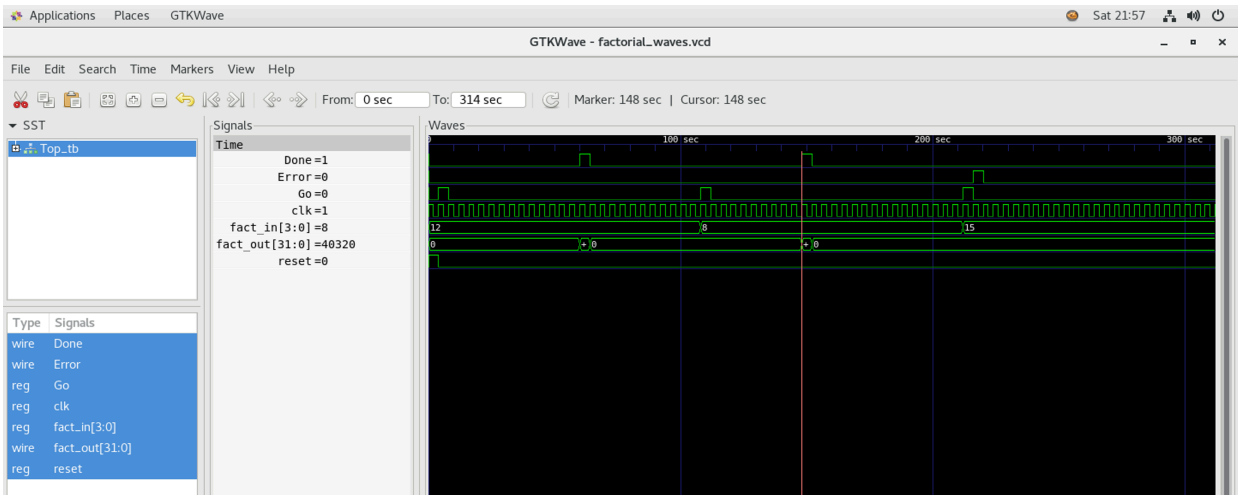
```
        @(posedge clk);
        Go =1'b1;
        @(posedge clk);
        Go = 1'b0;
        #100;
        fact_in = 4'hF;
        @(posedge clk);
        Go =1'b1;
        @(posedge clk);
        Go = 1'b0;
        #100;
        $finish;
end
initial
begin
        $monitor ("Result = %b", fact_out);
        $dumpfile ("factorial_waves.vcd");
        $dumpvars();
end
endmodule
```

Simulations:

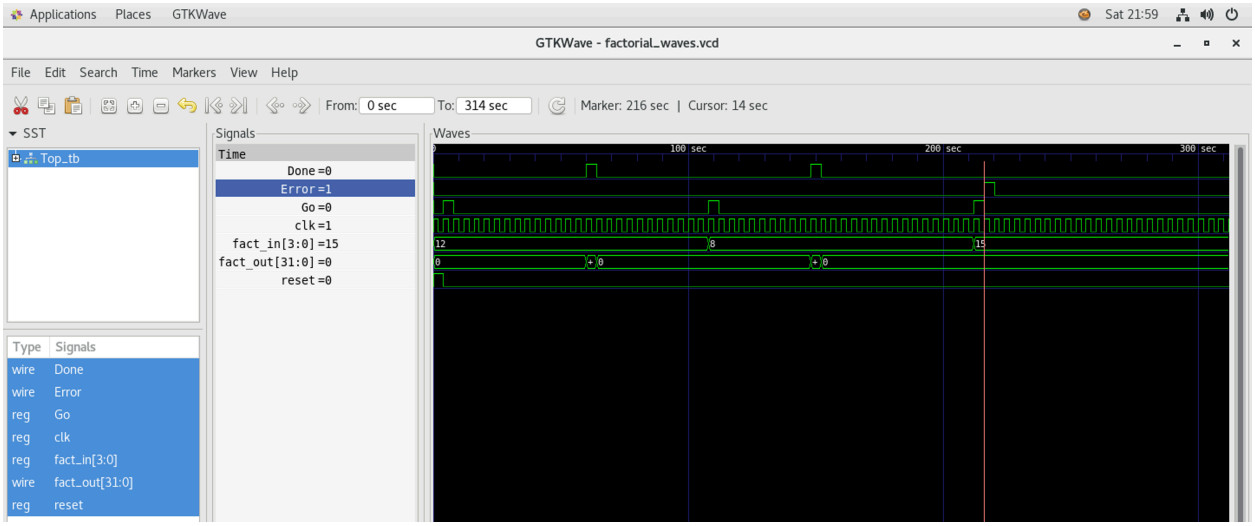
I. **For fact_in = 12**



II. For fact_in = 8



III. For fact_in = 16



Conclusion

The data-path and the control unit for the factorial system has been designed and verified for all the cases as shown above. All the verilog design files have been submitted along with the report.