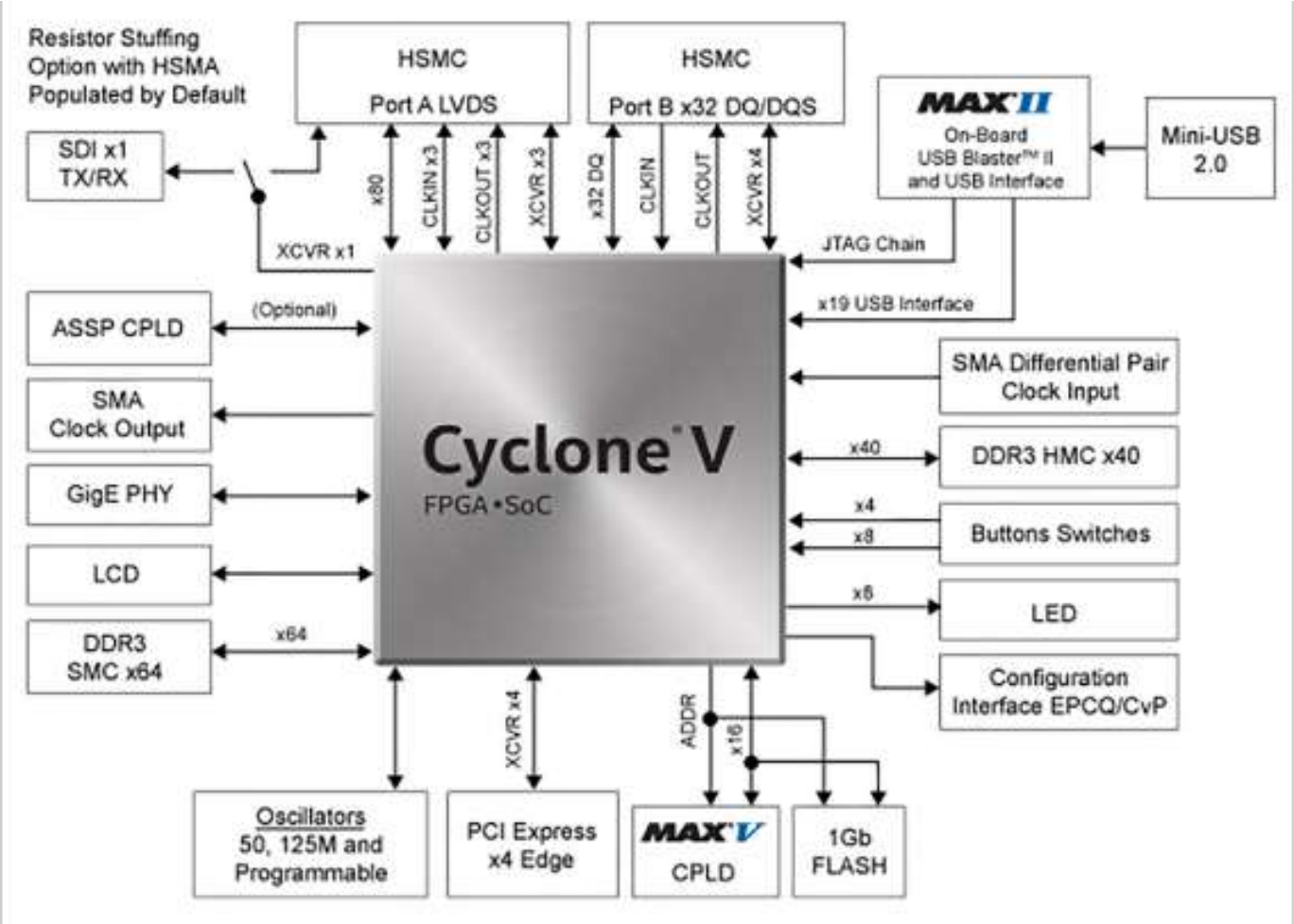
EC-340

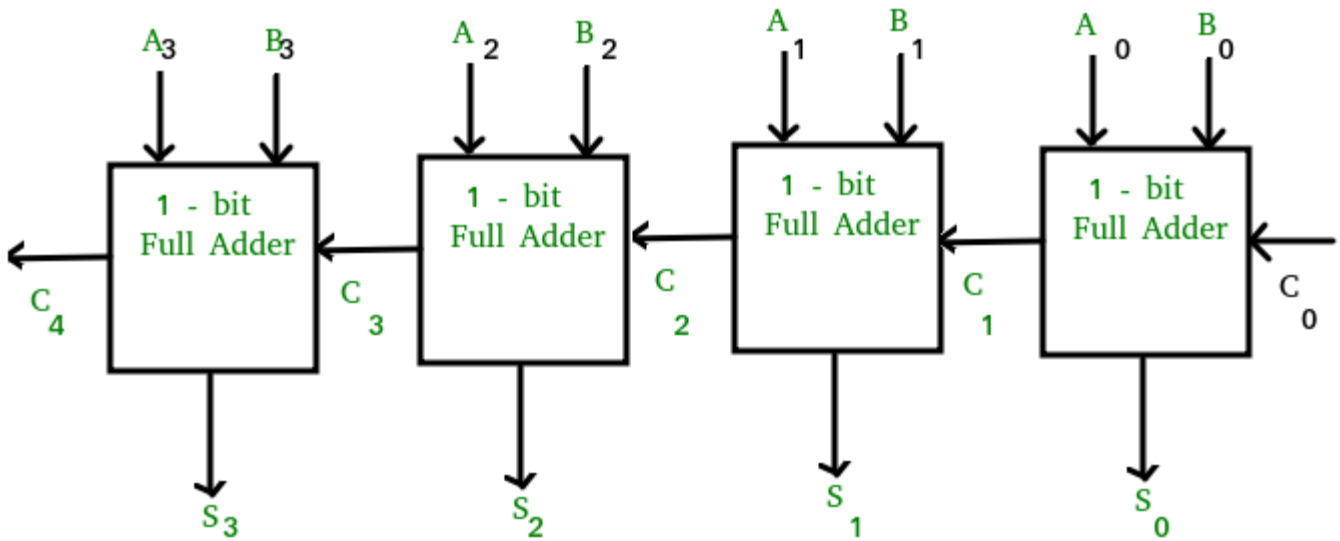
COMPUTER ORGANIZATION AND ARCHITECTURE



UTKARSH MAHAJAN 201EC164

1. (a) 8 bit ripple carry adder

It is a combinational circuit for adding 2 binary numbers requiring N full adders for N bit adder. Here for 8 bits, we will take 8 full adders, connect the cin to the first cin of first full adder and then cout of the first adder to the cin of the next and so on. While inputs a[i], b[i] to the i'th full adder. And take output of each full adder as s[i]. considering the cout of the last full adder as the output cout.



(Image from GFG)

Verilog Code for the module:

```
module ripple_carry_adder (input [7:0] a , b, input cin, output [7:0] s,
output cout);
wire [8:0] c;
assign c[0] = cin;
assign cout = c[8];
genvar i;
    generate
    for (i=0; i<8; i= i+1) begin: full_adders
        full_adder fa(.a(a[i]), .b(b[i]), .cin(c[i]), .s(s[i]),
.cout(c[i+1]));
    end
endgenerate
endmodule
```

Verilog code for the full adder:

```
module full_adder(input a, b, cin, output s, cout);
assign {cout, s} = a + b + cin;
endmodule
```

Verilog code for the test bench:

```
module ripple_carry_adder (input [7:0] a , b, input cin, output [7:0] s,
output cout);
wire [8:0] c;
assign c[0] = cin;
assign cout = c[8];
genvar i;
    generate
    for (i=0; i<8; i= i+1) begin: full_adders
        full_adder fa(.a(a[i]), .b(b[i]), .cin(c[i]), .s(s[i]),
.cout(c[i+1]));
    end
endgenerate
endmodule
```

Device Utilization Statistics:

Flow summary:

r.v ✕ | testbench.v ✕ | Compilation Report - ripple_carry_adder ✕

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Feb 17 04:56:54 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	ripple_carry_adder
Top-level Entity Name	ripple_carry_adder
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	21 / 32,070 (< 1 %)
Total registers	0
Total pins	26 / 457 (6 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

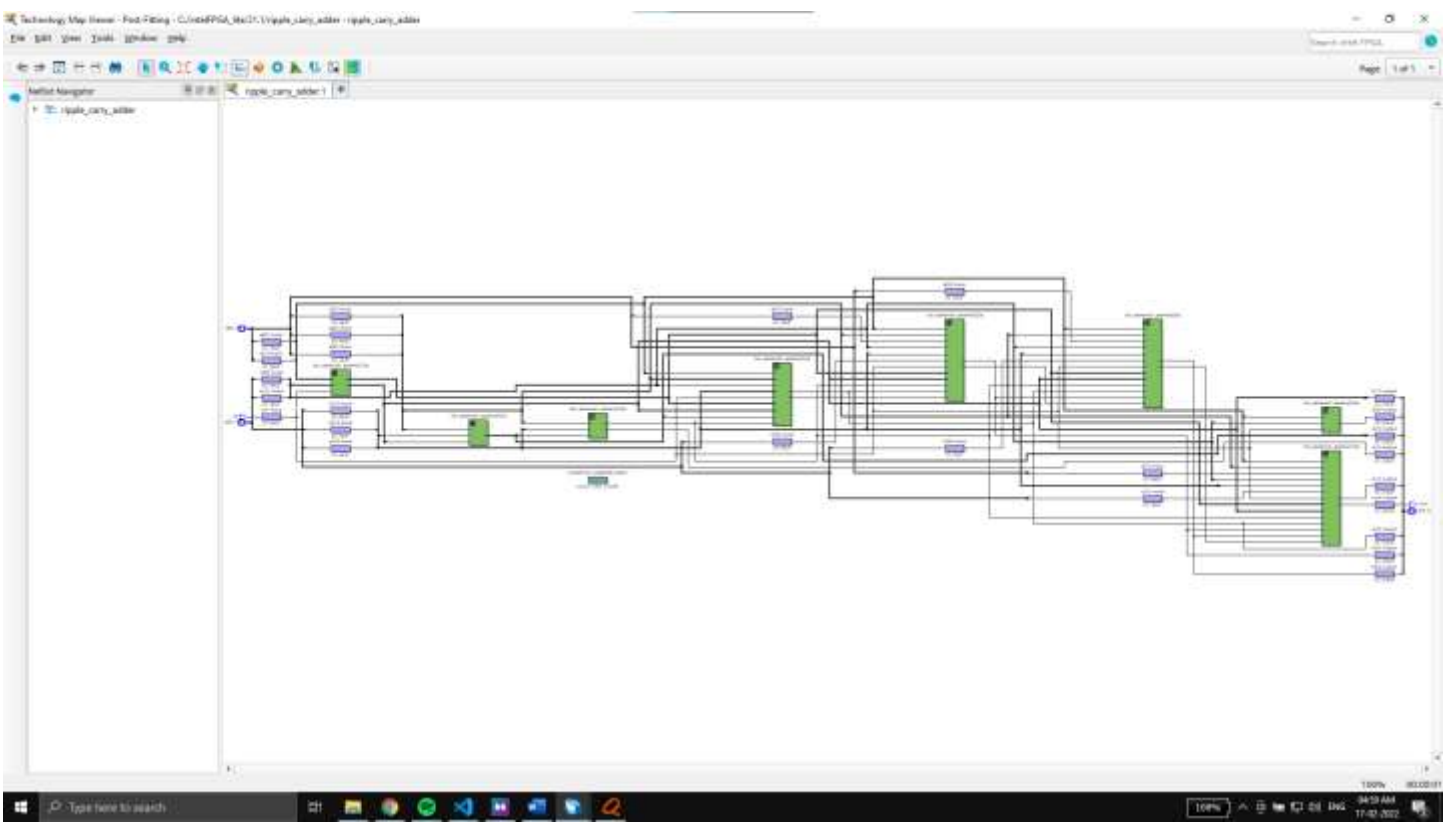
Resource usage summary:

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	20
2		
3	Combinational ALUT usage for logic	40
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	3
4	-- 4 input functions	13
5	-- <=3 input functions	24
4		
5	Dedicated logic registers	0
6		
7	I/O pins	26
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	a[2]~input
12	Maximum fan-out	10
13	Total fan-out	168
14	Average fan-out	1.83

Block Diagram (Technology map viewer):



Timing reports:

der.v ✕ | testbench.v ✕ | Compilation Report - ripple_carry_adder ✕

Clocks

No clocks to report.

dder.v ✕ | testbench.v ✕ | Compilation Report - ripple_carry_adder ✕

Slow 1100mV 85C Model Fmax Summary

No paths to report.

Unconstrained Paths Summary

<<Filter>>

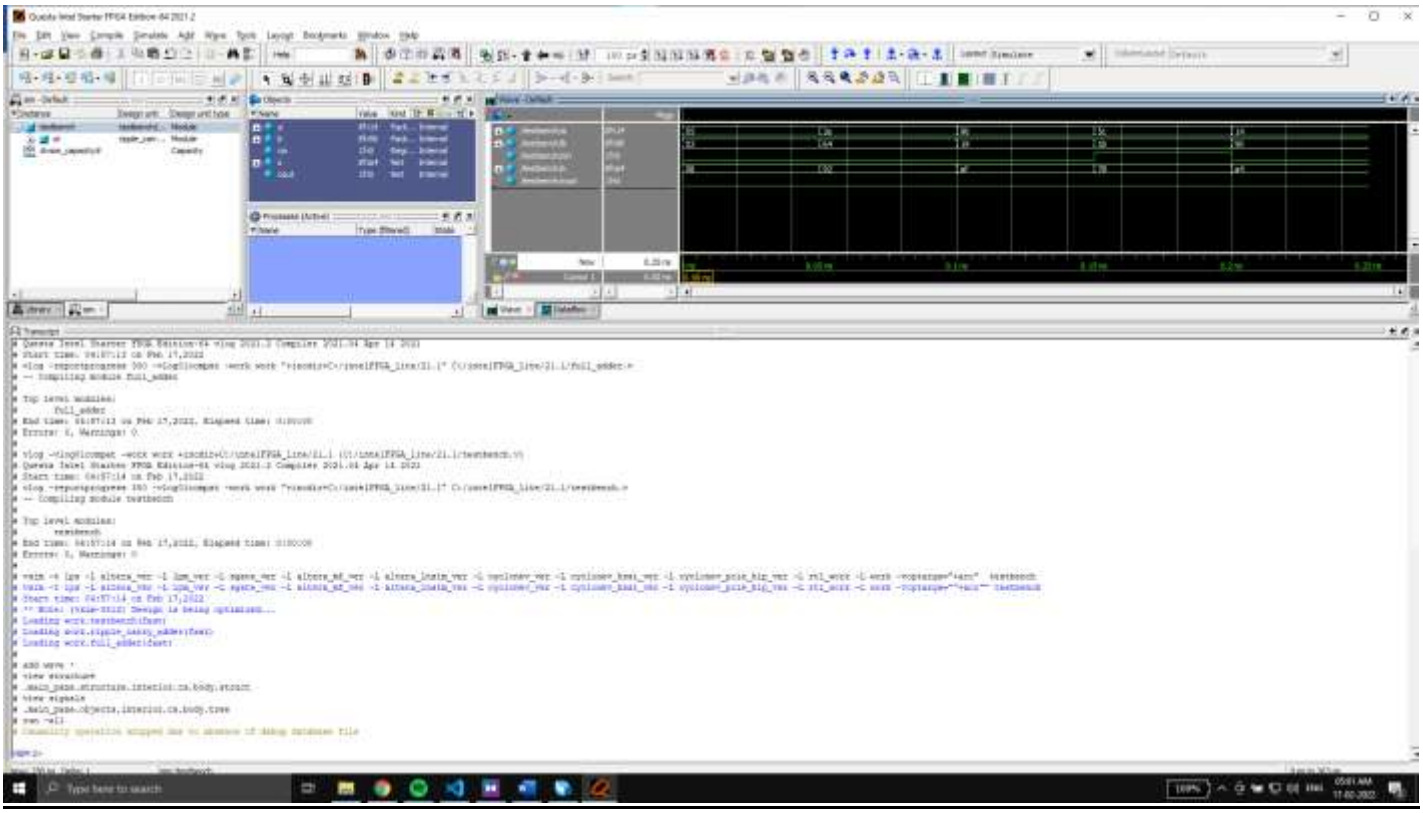
	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	17	17
4	Unconstrained Input Port Paths	97	97
5	Unconstrained Output Ports	9	9
6	Unconstrained Output Port Paths	97	97

Multicorner Timing Analysis Summary

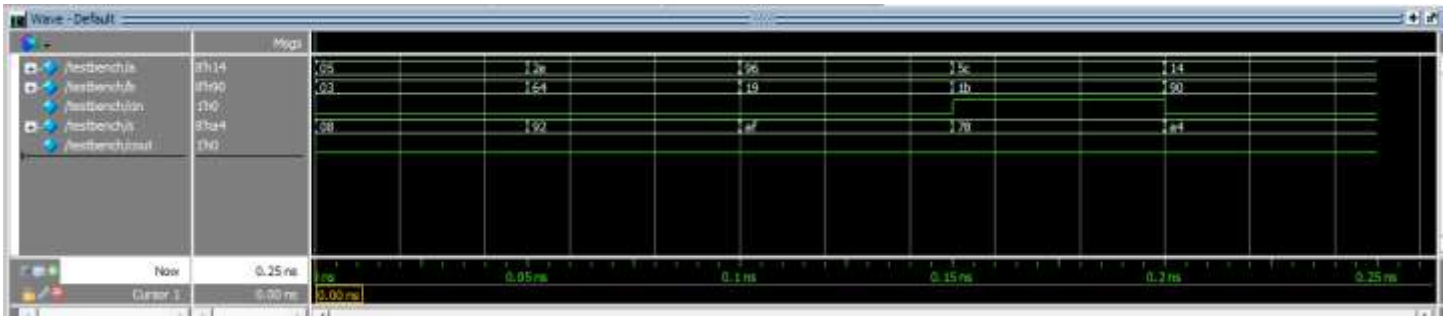
<<Filter>>

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	N/A	N/A	N/A	N/A	N/A
2	Design-wide TNS	0.0	0.0	0.0	0.0	0.0

Simulation in Questa through Quartus:



Wave:

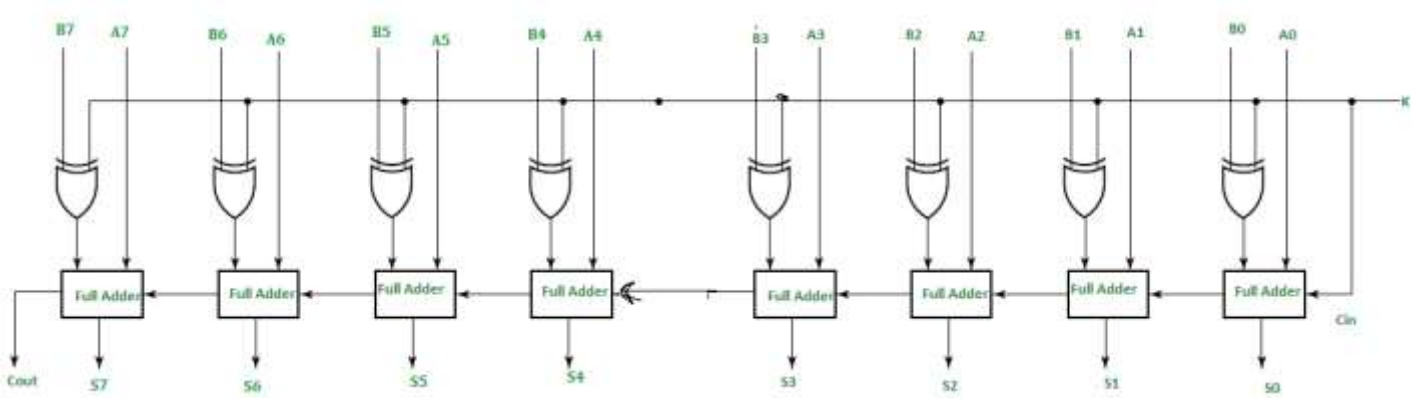


1. (b) 8-bit controllable adder/subtractor

An 8-bit controllable adder/subtractor is capable of both addition and subtraction of binary numbers. We will make use of 8 full adders and XOR gates for designing the circuit with Verilog.

We will have 3 inputs, A and B of width 8 bits. The addition or subtraction operation will be performed on them depending upon the third input K of size 1 bit.

If k=1, We will perform subtraction and if k=0, then we will perform addition.



(Image from GFG(edited))

Verilog Code for the module:

```
module add_sub(input [7:0]a, b, input k, output [7:0]s, output cout);
wire [8:0] c;
genvar i;
assign c[0] = k;
assign cout = c[8];
generate
    for (i=0; i<8; i= i+1) begin: add_sub
        full_adder fa(.a(a[i]), .b(b[i]^k), .cin(c[i]), .s(s[i]),
.cout(c[i+1]));
    end
endgenerate
endmodule
```

Verilog code for the full adder:

```
module full_adder(input a, b, cin, output s, cout);
    assign {cout, s} = a + b + cin;
endmodule
```

Verilog code for the test bench:

```
module testbench();
reg [7:0] a, b;
reg k;
wire [7:0]  s;
wire cout;
add_sub asc(a, b, k, s, cout);
initial begin
 a=5; b=3; k=0;
#50 a=46; b=100; k=0;
#50 a=150; b=25; k=0;
#50 a=8'b01011100; b=8'b00011011; k=1;
#50 a=20; b=8'b10010000; k=0; #50;
end
endmodule
```

<u>Device Utilization Statistics:</u>

Flow summary:

Flow Summary

<<Filter>>

Flow Status	In progress - Thu Feb 17 03:25:49 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	add_sub
Top-level Entity Name	add_sub
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	26
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Resource usage summary:

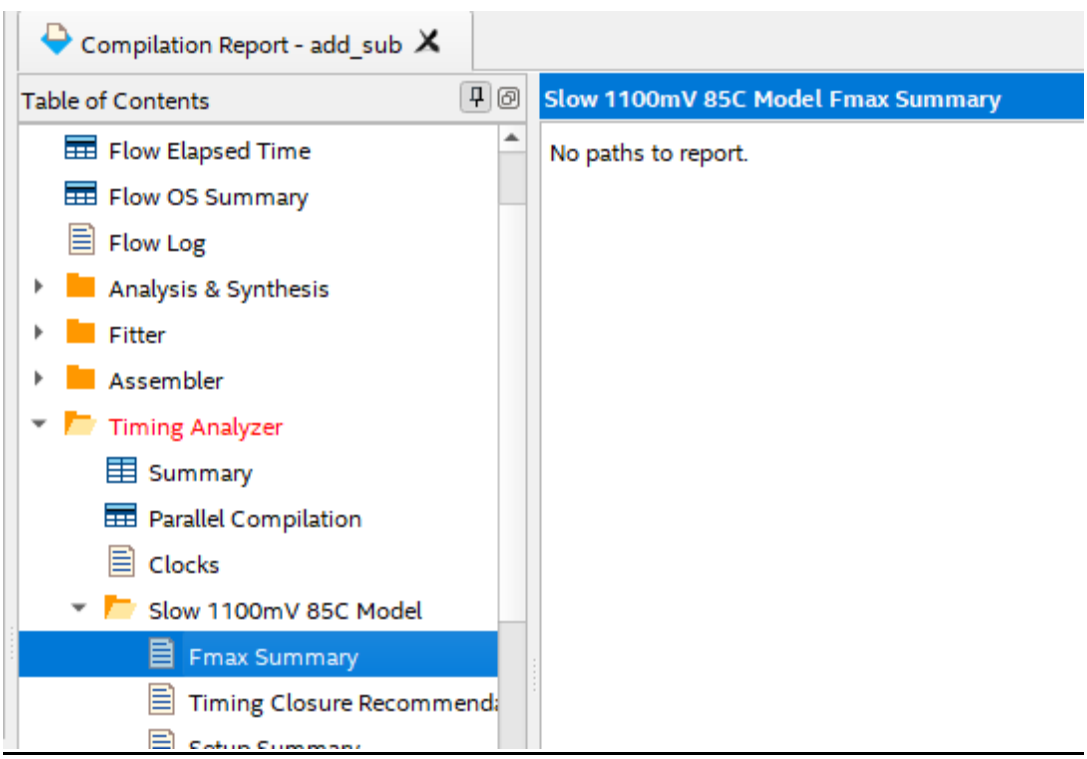
Analysis & Synthesis Resource Usage Summary

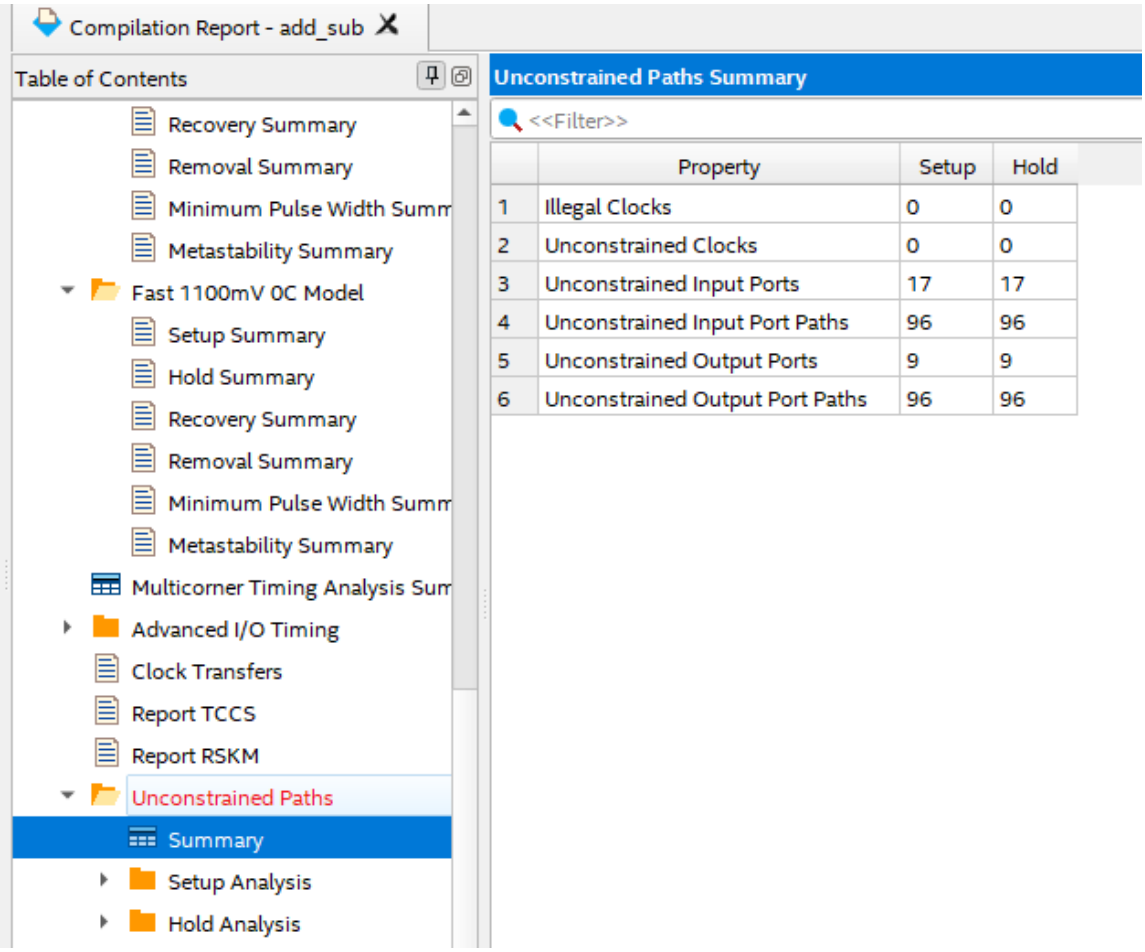
<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	21
2		
3	Combinational ALUT usage for logic	42
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	3
4	-- 4 input functions	16
5	-- <=3 input functions	23
4		
5	Dedicated logic registers	0
6		
7	I/O pins	26
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	k~input
12	Maximum fan-out	20
13	Total fan-out	182
14	Average fan-out	1.94

Block Diagram (Technology map viewer):

Timing reports:

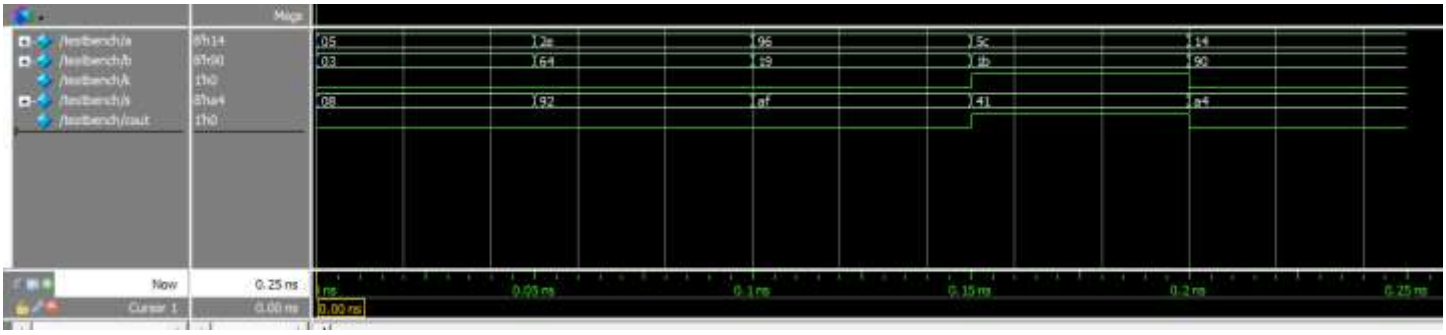




	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	17	17
4	Unconstrained Input Port Paths	96	96
5	Unconstrained Output Ports	9	9
6	Unconstrained Output Port Paths	96	96

Simulation in Questa through Quartus:

Wave:



2. Decade Counter - Model a synchronous up/down decade counter with asynchronous reset. The counter is rising edge triggered.

If the **load** = 1, the data **data_in** is loaded into the counter.

If the **counter_on**=**counter_up**=1, the counter is incremented, the Terminal carry output TC=1 when the counter is in state 9

If the **counter_on**=1 and **counter_up**=0, the counter is decremented, the Terminal carry output TC=1 when the counter is in state 0.

->

The decade counter counts up and down from 1to 9 and 9 to 1 in binary depending upon the input.

We will consider 10 states from s0 to s9, a clock, asynchronous reset, option to load values.

We have set different output values for different respective states.

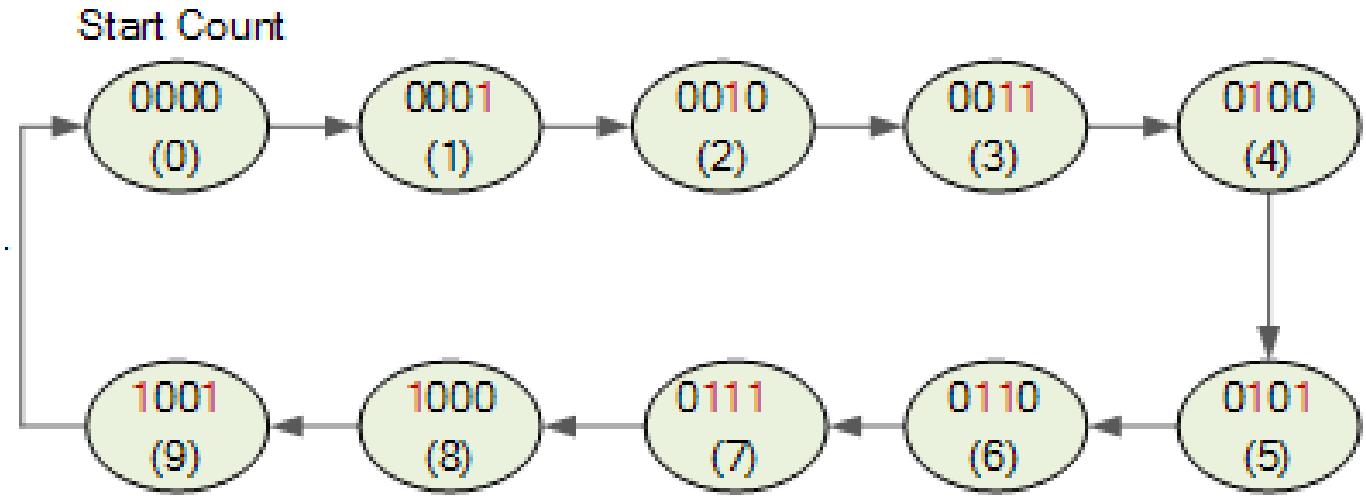
Depending upon the input value of counter_on. We will set the next_state for our FSM.

For example: if the system is in state s0 and count_up is 1, then the next state will be s1 while if the count_up was 0 then the next state will be s9 as it has to count downwards and switch to maximum after 0.

Our FSM will be positive edge activated hence we use it in our sensitive list for changing states.

For ex: s0 state will give output 0 for count, while s1 will give s1.

We will set TC to 1 for the conditions mentioned above in the question.



Verilog Code for the Decade counter:

```
module decade_counter (count_up, reset, load, counter_on, clk, data_in, count, TC);
input count_up, reset, load, counter_on, clk;
input [3:0] data_in;
output reg [3:0]count;
output reg TC;

localparam [3:0] s0=4'b0000, s1=4'b0001, s2=4'b0010, s3=4'b0011, s4=4'b0100,s5=4'b0101,
s6=4'b0110 , s7=4'b0111, s8=4'b1000, s9=4'b1001;
reg [3:0] state, next_state;
always @ (posedge clk or negedge reset) begin
    if(!reset) begin state<= s0; end
    else if(load) begin
        state <= data_in;
    end else if(counter_on) state<= next_state;
end

always @(state) begin
    case(state)
    s0: begin if(count_up) begin next_state <= s1; TC<=0; end
                else begin next_state <= s9;  TC<=1; end
                count<= 0;
        end
    s1: begin if(count_up)begin next_state <= s2; end else begin next_state <= s0; end
count <= 1; TC<=1; end
    s2: begin if(count_up)begin next_state <= s3; end else begin next_state <= s1; end
count <= 2; TC<=0; end
    s3: begin if(count_up)begin next_state <= s4; end else begin next_state <= s2; end
count <= 3; TC<=0; end
    s4: begin if(count_up)begin next_state <= s5; end else begin next_state <= s3; end
count <= 4; TC<=0; end
```

```
    s5: begin if(count_up)begin next_state <= s6; end else begin next_state <= s4; end
count <= 5; TC<=0; end
    s6: begin if(count_up)begin next_state <= s7; end else begin next_state <= s5; end
count <= 6; TC<=0; end
    s7: begin if(count_up)begin next_state <= s8; end else begin next_state <= s6; end
count <= 7; TC<=0; end
    s8: begin if(count_up)begin next_state <= s9; end else begin next_state <= s7; end
count <= 8; TC<=0; end
    s9: begin if(count_up)begin next_state <= s0; TC<=1; end else begin next_state <= s8;
TC<=0; end count <= 9;  end
    default: begin next_state <=s0; count <=0; TC=0; end
    endcase
end
endmodule
```

Verilog code for the test bench:

```
module testbench_1();
integer i;
reg clk, counter_on, reset, load, count_up;
reg [3:0] data_in;
wire [3:0] count;
wire TC;
decade_counter dc1(count_up, reset, load, counter_on, clk, data_in, count,
TC);
initial
begin
counter_on=1; count_up =1;
reset=1; clk=1; #1; reset=0; #1 reset=1;
load=0; data_in =4'b1000;
for( i=0; i<10; i=i+1)begin
clk =1; #20; clk =0; #20;
end
load =1;
clk =1; #20; clk =0; #20; load =0;
count_up =0;
for( i=0; i<5; i=i+1)begin
clk =1; #20; clk =0; #20;
end
end
endmodule
```

Device Utilization Statistics:

Flow summary:

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Feb 16 15:39:03 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	decade_counter
Top-level Entity Name	decade_counter
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	9 / 32,070 (< 1 %)
Total registers	4
Total pins	14 / 457 (3 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Resource usage summary:

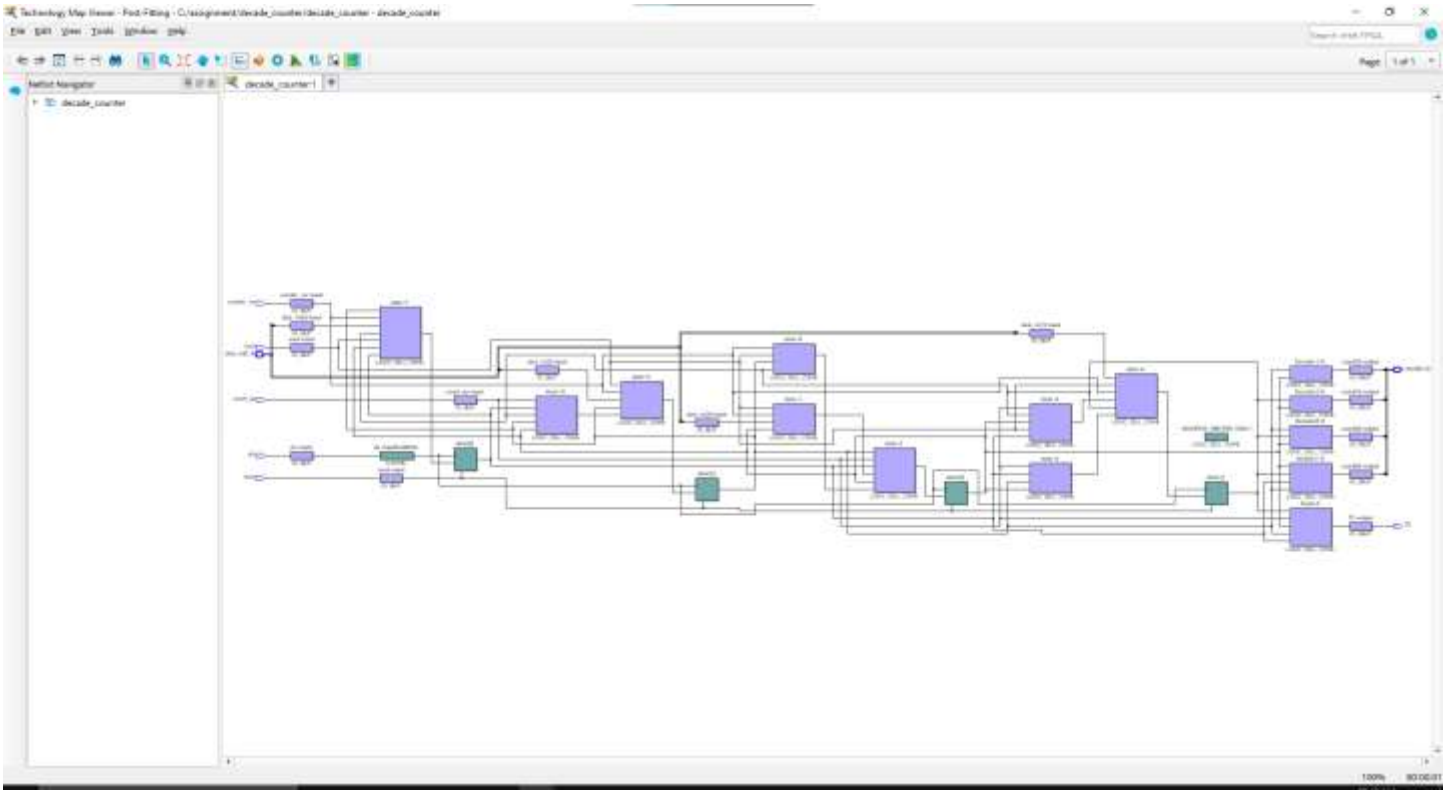
decade_counter.v testbench_1.v

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	9
2		
3	Combinational ALUT usage for logic	14
1	-- 7 input functions	1
2	-- 6 input functions	2
3	-- 5 input functions	4
4	-- 4 input functions	4
5	-- <=3 input functions	3
4		
5	Dedicated logic registers	4
6		
7	I/O pins	14
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	state[3]
12	Maximum fan-out	11
13	Total fan-out	93
14	Average fan-out	2.02

Block Diagram (Technology map viewer):



<u>Timing reports:</u>

Multicorner Timing Analysis Summary

<<Filter>>

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	-0.859	0.174	N/A	N/A	-0.394
1	clk	-0.859	0.174	N/A	N/A	-0.394
2	Design-wide TNS	-2.039	0.0	0.0	0.0	-2.264
1	clk	-2.039	0.000	N/A	N/A	-2.264

For slow 85C model:

Compilation Report - decade_counter

Table of Contents

- Parallel Compilation
- Clocks
- Slow 1100mV 85C Model
  - Fmax Summary

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	551.57 MHz	551.57 MHz	clk	

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.813	-1.991

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.376	0.000

Slow 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-2.096

For slow 0C model:

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	537.92 MHz	537.92 MHz	clk	

Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.859	-2.039

Slow 1100mV 0C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.391	0.000

Slow 1100mV 0C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-2.264

For fast 85C model:

Fast 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.035	-0.035

Fast 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.181	0.000

Fast 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.080	-0.320

For fast 0C model:

Fast 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.018	0.000

Fast 1100mV 0C Model Hold Summary

<<Filter>>

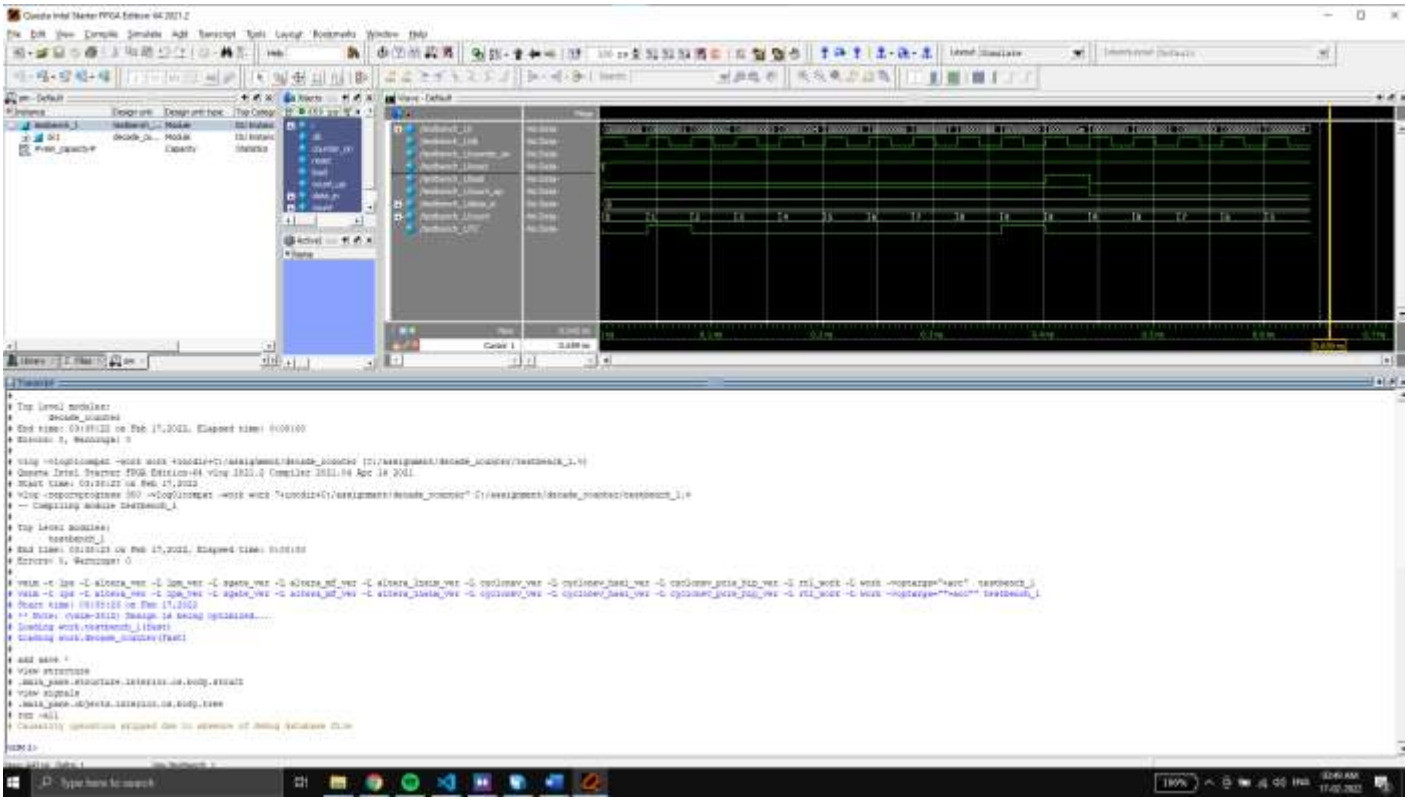
	Clock	Slack	End Point TNS
1	clk	0.174	0.000

Fast 1100mV 0C Model Minimum Pulse Width Summary

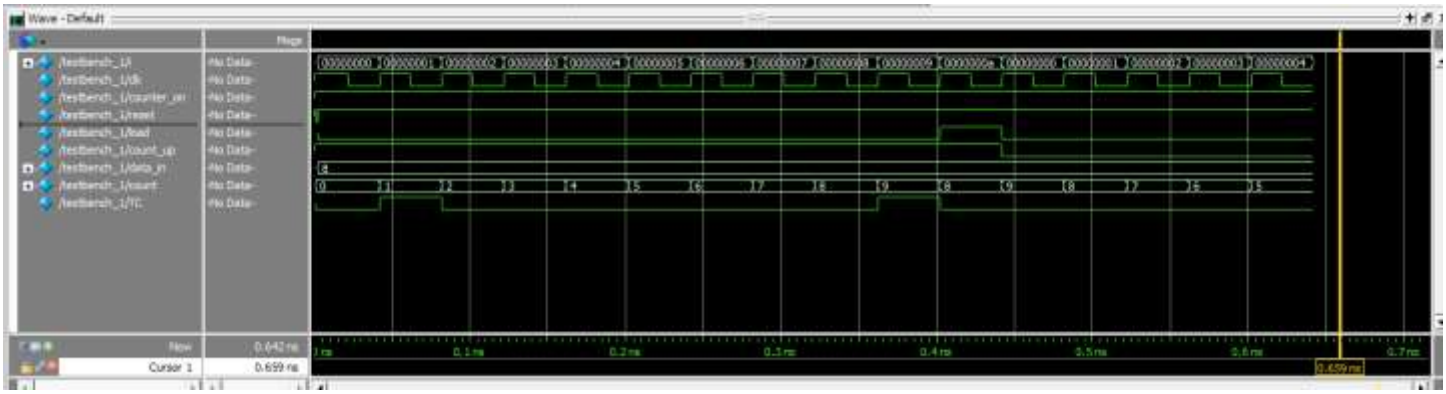
<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.081	-0.324

Simulation in Questa through Quartus:



Wave:



3. Serial adder to add two 8 bit numbers using a single full adder and registers

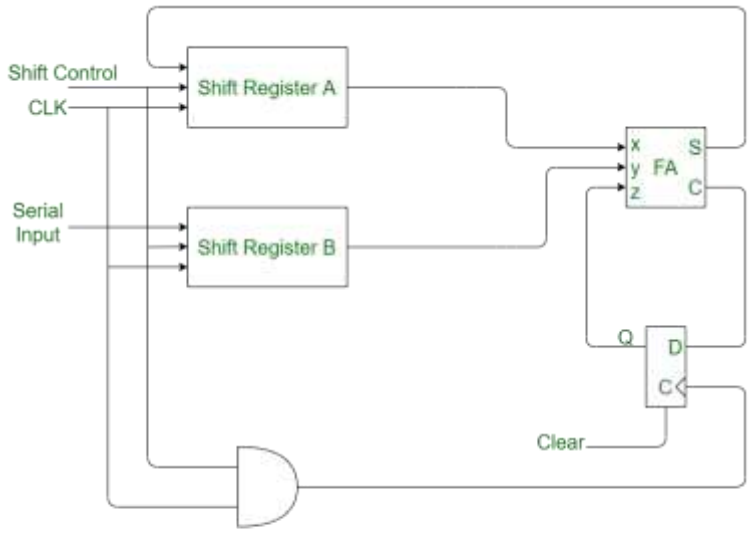
->Serial adder performs the addition of 2 binary numbers in serial form.

For designing the module, we will use separate submodules:

- 8 Right Shift registers: for inputs a, b and output s (sum)
- D flip flop: for storing carry
- Full adder: for carrying out addition of LSB of both inputs a and b and storing it in the msb of output sum.

We will use output register **done** for indicating the completion of addition. Keeping our FSM positive edge activated with a asynchronous reset.

We will take inputs at the MSB of inputs A and B and then shift them throughout the shift register, while at the same time we will right shift the sum. after 15 clock cycles we will have completely done taking in the input and adding them and storing it in output shift register sum. At which we will enable done. (image from gfg)



Block Diagram of Serial Binary Adder

Verilog Code for the Serial Adder:

```
// sum output is at output reg sum and the carry of the sum is at output reg
C.
module serial_adder(
input ain, bin, clk, start, resetn,
output [7:0] sum, output reg done, output  c);
//wires for connecting different modules
wire [7:0] rxQ, ryQ, rsQ;wire fas, fac, dffc;
reg [3:0] count =4'b1111;
reg [7:0] D =0;
reg load = 0;
//connecting wires from modules to output registers.
assign sum = rsQ;
assign c = fac;
shiftreg regX(.D(D), .load(load), .shiftR(!done),.lin(ain), .clk(clk),
.Q(rxQ));
shiftreg regY(.D(D), .load(load), .shiftR(!done),.lin(bin), .clk(clk),
.Q(ryQ));
shiftreg regSum(.D(D), .load(load), .shiftR(!done),.lin(fas), .clk(clk),
.Q(rsQ));
fulladder fa(.a(rxQ[0]), .b(ryQ[0]), .cin(dffc), .sum(fas), .cout(fac));
dflipflop da(.d(fac), .reset(load), .clk(clk), .q(dffc));
//for decrementing counter and making a reset button for entire module.
always @(posedge clk, negedge resetn)
begin
if(!resetn) begin
load<= 1;
count<=15;
done<=0;
end else if(start) begin
load<= 0;
if(count >0) begin done <= 0;
count<= count -1;
    end
else done <=1;
end
end
endmodule
//single bit full adder module required.
module fulladder(input a, b, cin, output sum, cout);
assign sum = cin^(a^b);
assign cout = (a&b) | cin&(a^b);
endmodule
// D flip flop
module dflipflop(input d, reset, clk, output reg q);
always@(posedge clk, posedge reset)
begin
if(reset) q=0;
```

```
else begin
q=d;
end
end
endmodule
//8 bit shift register with load support.
module shiftreg (input [7:0]D, input load, shiftR, lin, clk, output reg
[7:0]Q);

always @(posedge clk)
begin
if(load) Q<=D;
else if(shiftR) Q <= {lin, Q[7:1]};
end
endmodule
```

Verilog code for the test bench:

```
module tb2();
reg a, b, clk, start, resetn;
wire [7:0]sum;
wire done, c;
reg [7:0] testinputx, testinputy;
integer i;
serial_adder ut(.ain(a), .bin(b), .clk(clk), .start(start),
.resetn(resetn),.sum(sum),
.done(done), .c(c));
initial
begin
//giving starting default values
a=0; b=0; start=1; clk=0;
//setting teh module at reset.
resetn=1; #1; resetn=0; #1; resetn=1;
//input values for x and y, from left to right.
testinputx = 8'b01010110;
testinputy = 8'b10110010;
//cyclic inputs x and y transversing the above values.
for( i=7; i>=0; i=i-1)begin
a = testinputx[i];
b = testinputy[i];
clk =1; #20;
clk =0; #20;
end
//additional cycles for computing the adder result.
for( i=7; i>=0; i=i-1)begin
clk =1; #20;
clk =0; #20;
end
end
endmodule
```

Device Utilization Statistics:

Flow summary:

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Feb 17 04:03:01 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	serial_adder
Top-level Entity Name	serial_adder
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	10 / 32,070 (< 1 %)
Total registers	33
Total pins	15 / 457 (3 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Resource usage summary:

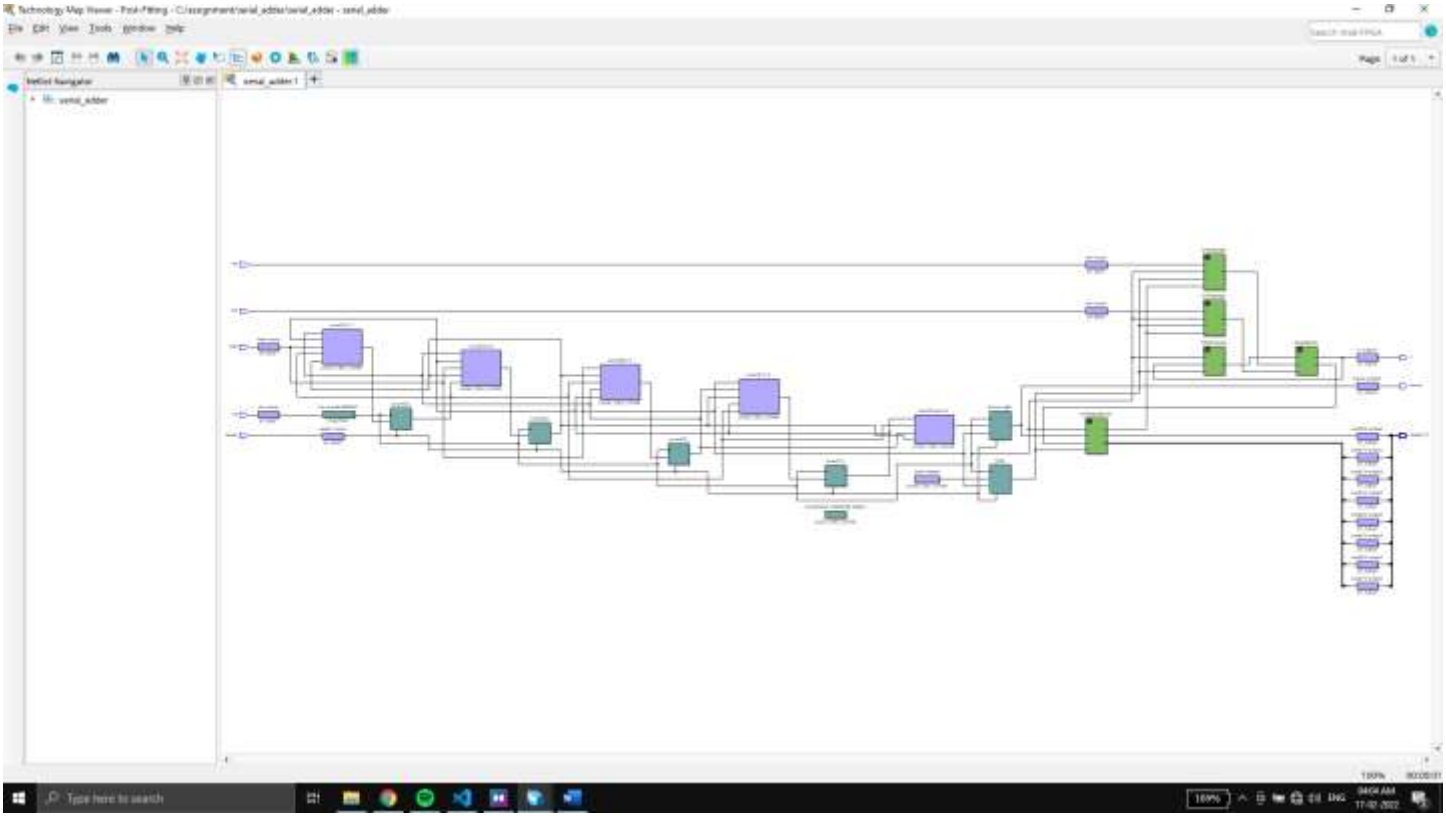
Compilation Report - serial_adder

Analysis & Synthesis Resource Usage Summary

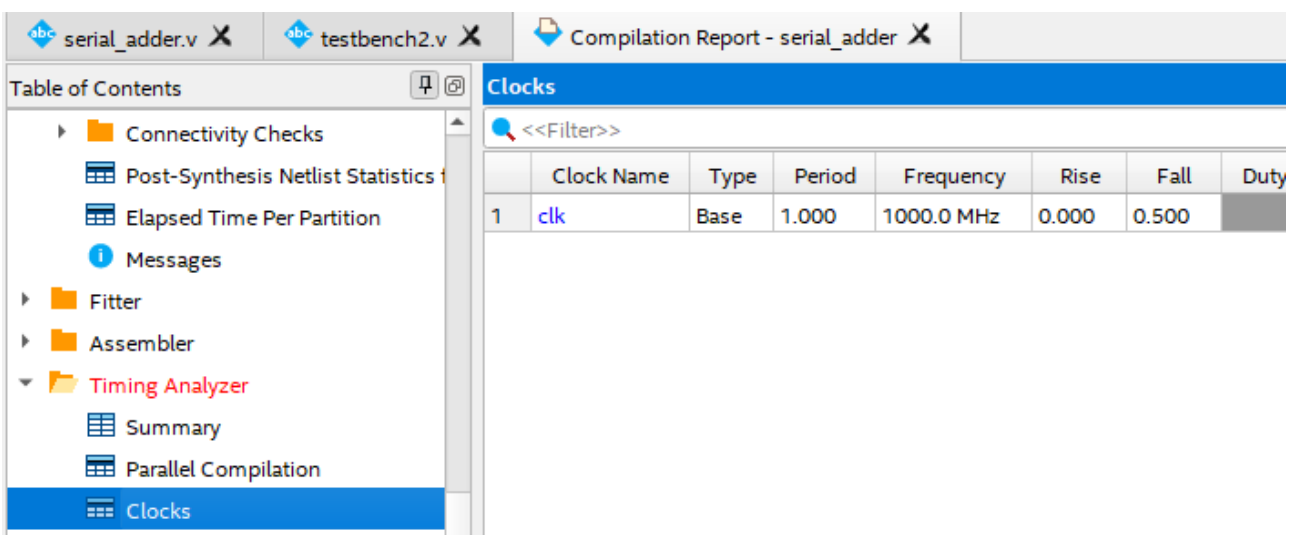
<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	16
2		
3	Combinational ALUT usage for logic	8
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	4
4	-- 4 input functions	1
5	-- <=3 input functions	3
4		
5	Dedicated logic registers	31
6		
7	I/O pins	15
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	clk~input
12	Maximum fan-out	31
13	Total fan-out	175
14	Average fan-out	2.54

Block Diagram (Technology map viewer):



Timing reports:



	Clock Name	Type	Period	Frequency	Rise	Fall	Duty
1	clk	Base	1.000	1000.0 MHz	0.000	0.500	

Multicorner Timing Analysis Summary

<<Filter>>

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	-0.761	0.112	-0.320	0.402	-0.394
1	clk	-0.761	0.112	-0.320	0.402	-0.394
2	Design-wide TNS	-19.735	0.0	-0.32	0.0	-19.044
1	clk	-19.735	0.000	-0.320	0.000	-19.044

For slow 85C model:

serial_adder.v | testbench2.v | Compilation Report - serial_adder

Table of Contents

- Parallel Compilation
- Clocks
- Slow 1100mV 85C Model
  - Fmax Summary

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Nan
1	567.86 MHz	567.86 MHz	clk

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.761	-19.735

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.204	0.000

Slow 1100mV 85C Model Recovery Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.320	-0.320

Slow 1100mV 85C Model Removal Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.779	0.000

Slow 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-17.578

For slow 0C model:

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	589.97 MHz	589.97 MHz	clk	

Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.695	-18.194

Slow 1100mV 0C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.191	0.000

Slow 1100mV 0C Model Recovery Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.261	-0.261

Slow 1100mV 0C Model Removal Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.723	0.000

Slow 1100mV 0C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-19.044

For fast 85C model:

Fast 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.043	0.000

Fast 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.124	0.000

Fast 1100mV 85C Model Recovery Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.239	0.000

Fast 1100mV 85C Model Removal Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.447	0.000

Fast 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.084	-2.718

For slow 0C model:

Fast 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.108	0.000

Fast 1100mV 0C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.112	0.000

Fast 1100mV 0C Model Recovery Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.296	0.000

Fast 1100mV 0C Model Removal Summary

<<Filter>>

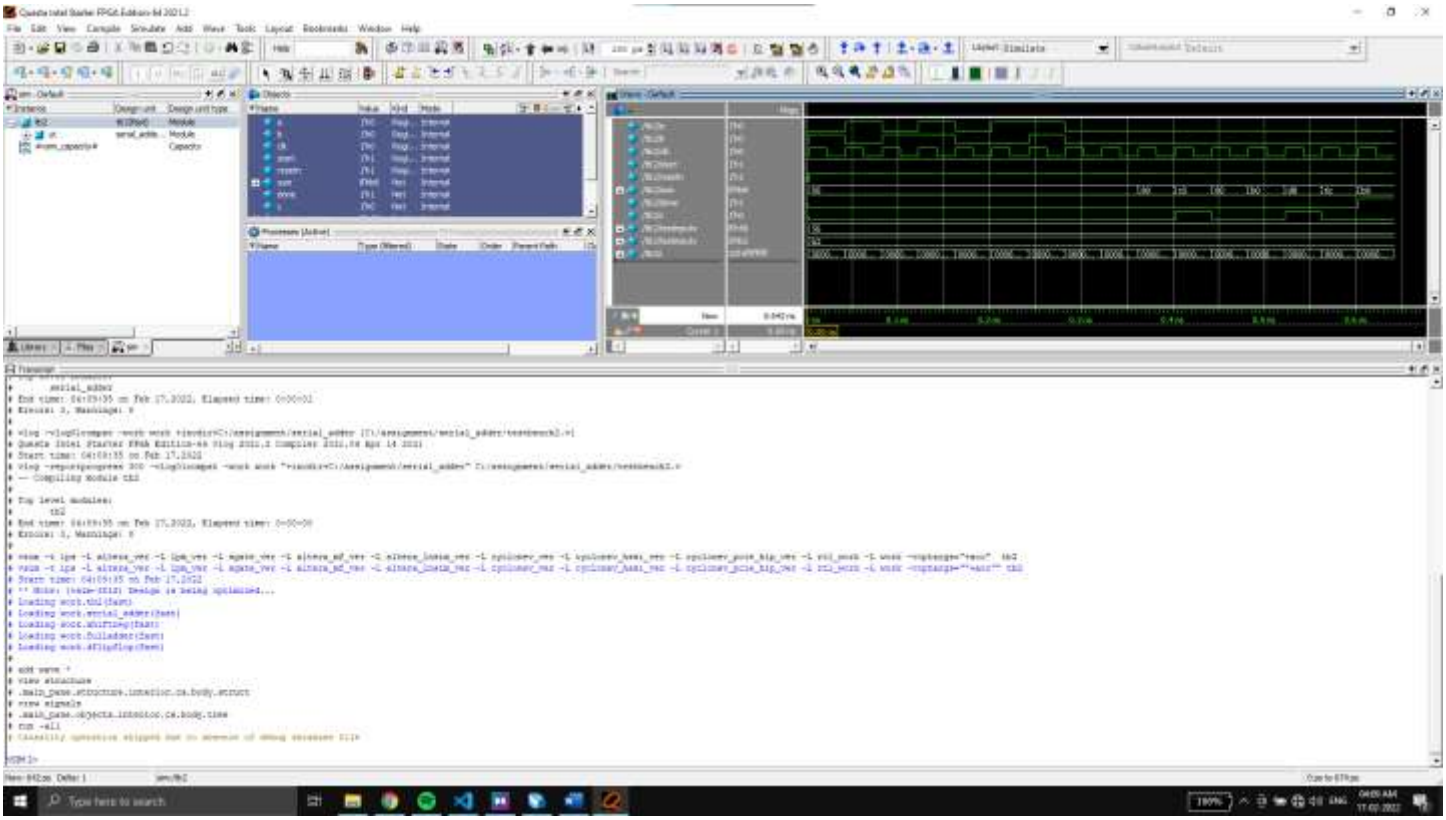
	Clock	Slack	End Point TNS
1	clk	0.402	0.000

Fast 1100mV 0C Model Minimum Pulse Width Summary

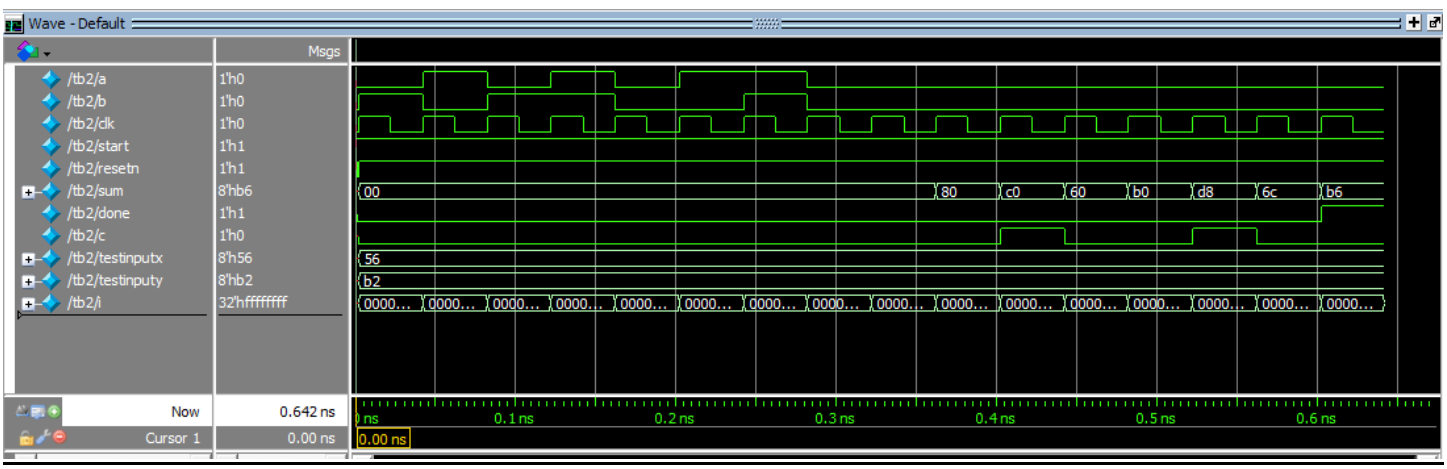
<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.083	-2.669

Simulation in Questa through Quartus:



Wave:



4. Shift and add multiplier in Verilog to multiply two 8 bit numbers using RTL approach. It should be capable of multiplying both positive and negative numbers. Negative numbers are represented in 2'sC representation.

Storing the input values in 8 bit registers A and B we will display the output using a 16bit Register P.

we know that for signed multiplication when multiplicand is negative we need to ignore carry and use 1 as the msb's while adding to the register P when the multiplier bit is 1 while when multiplier is negative we just need to add two's complement in the last step.

We will use 10 states from s0 to s9 indicating value 0 to 9 respectively.

We will make our system positively edge activated with asynchronous reset.

On each state we will be storing the left shifted value of A_q in A_d storing the right shifted value of B_q in B_d.

On each shift to positive edge of the clk(clock), we will update the output pdt_q with the calculated next_ouput from the 2nd always block according to the method discussed above. We will also updating the shifted values in the A_q and B_q (from A_d and B_d respectively) at that time.

Verilog Code for the Shift and add multiplier:

```
module shift_multiply(input clk, resetn, start, input [7:0]A, B, output
[15:0] P, output reg done);
localparam S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7, S8=8, S9=9;
reg [4:0] state_d, state_q;
reg [7:0] B_d, B_q;
reg [15:0] A_d, A_q, pdt_d, pdt_q;
assign P = pdt_q;
always @ (posedge clk , negedge resetn)
begin
    if(!resetn) state_q <=S0;
    else begin
    state_q <= state_d;
    pdt_q <=pdt_d;
    A_q <= A_d;
    B_q <= B_d;
    end
end

always @(state_q) begin
    state_d = state_q;
    done = 1'b0;
```

```
    case(state_q)
        S0:if(start) state_d=S1;
        S1: state_d=S2;
        S2: state_d=S3;
        S3: state_d=S4;
        S4: state_d=S5;
        S5: state_d=S6;
        S6: state_d=S7;
        S7: state_d=S8;
        S8: state_d=S9;
        S9: begin
        done= 1'b1;
        if(start)state_d =S1;
        end
        default: state_d=S0;
    endcase
end

always @(state_q or pdt_q or A_q or B_q)
begin
pdt_d = pdt_q; A_d = A_q; B_d = B_q;
case (state_q)
S0: begin
pdt_d = {16{1'b0}};
A_d = A;
B_d = B;
end
S1: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {8'b11111111,A_q[7:0]} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S2: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {7'b1111111,A_q[8:1], 1'b0} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S3: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {6'b111111,A_q[9:2], 2'b00} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
```

```
end
S4: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {5'b11111,A_q[10:3], 3'b000} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S5: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {4'b1111,A_q[11:4], 4'b0000} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S6: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {3'b111,A_q[12:5], 5'b00000} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end
S7: begin
A_d = A_q << 1;
B_d = B_q >> 1;
if (B_q[0] == 1'b1) begin
    if(A[7] == 1'b1) pdt_d = {2'b11,A_q[13:6], 6'b000000} +pdt_q;
    else pdt_d = A_q + pdt_q;
    end
end

S8: begin
    A_d = A_q << 1;
    B_d = B_q >> 1;
    if (B_q[0] == 1'b1) begin
        if(B[3] == 1'b1) pdt_d = {1'b0, (~A+1'b1),7'b0000000} +pdt_q;
        else if(A[3] == 1'b1) pdt_d = {1'b1,A_q[15:8], 7'b0000000} +pdt_q;
        else pdt_d = A_q + pdt_q;
    end
end
default: pdt_d = pdt_q;
endcase
end
endmodule
```

Verilog code for the test bench:

```
module test_bench();
reg clk, resetn, start;
reg [7:0] A, B;
wire [15:0] P;
wire done;
integer i;
shift_multiply ut(.clk(clk), .resetn(resetn), .start(start), .A(A), .B(B),
.P(P), .done(done));
initial
begin
    clk=0;resetn=0;
    #2; clk=1;
    A=8'b00101011; B= 8'b01001101;
    #2 resetn=1; clk=0; start=1'b1;
    for(i=0; i<=9;i=i+1) begin
    #20 clk =1; #20 clk=0;
    end
end
endmodule
```

Device Utilization Statistics:

Flow summary:

Compilation Report - shift_multiply

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Feb 17 04:16:33 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	shift_multiply
Top-level Entity Name	shift_multiply
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	160 / 32,070 (< 1 %)
Total registers	51
Total pins	36 / 457 (8 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Resource usage summary:

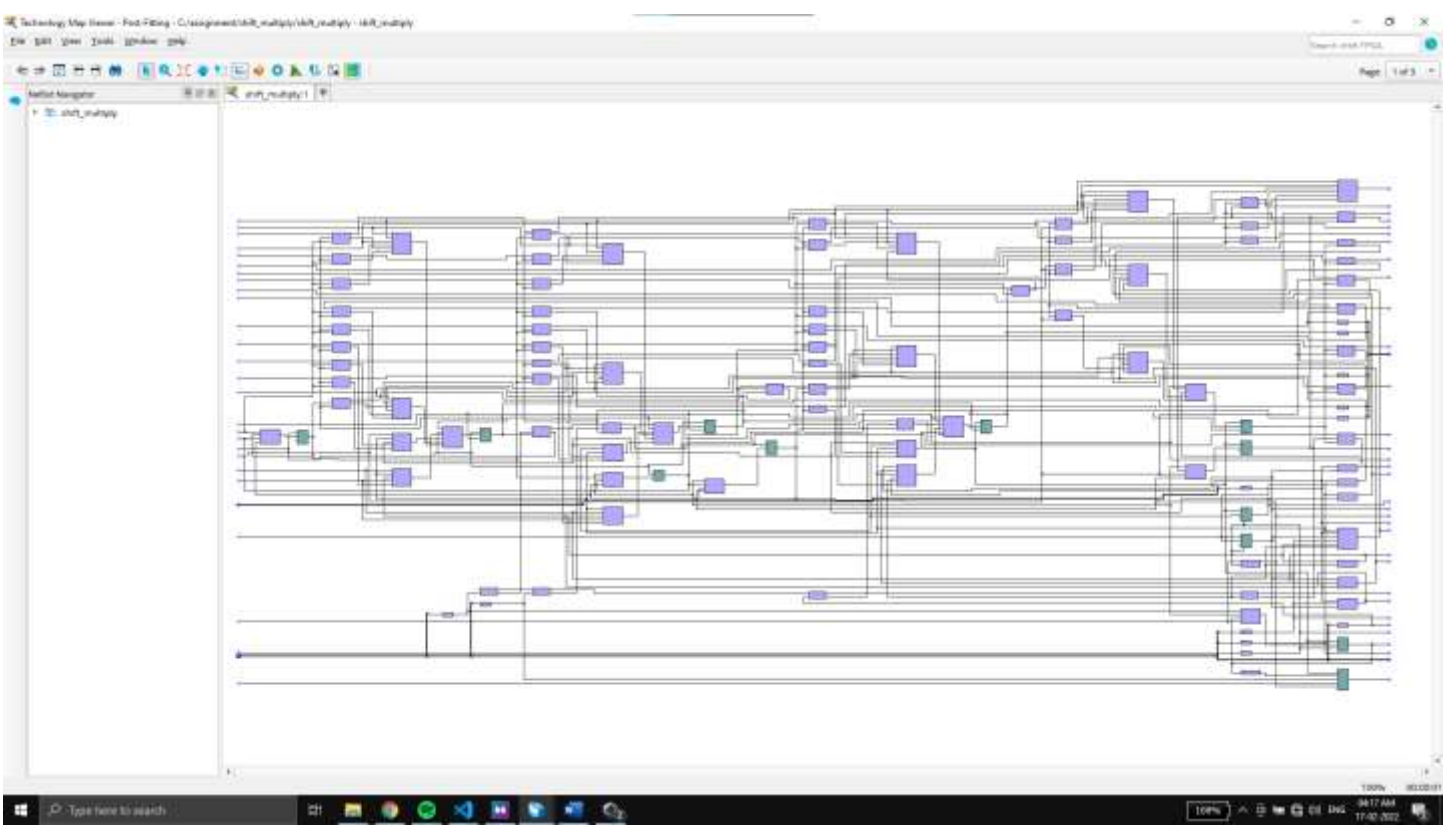
Compilation Report - shift_multiply

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	159
2		
3	Combinational ALUT usage for logic	251
1	-- 7 input functions	0
2	-- 6 input functions	52
3	-- 5 input functions	21
4	-- 4 input functions	19
5	-- <=3 input functions	159
4		
5	Dedicated logic registers	50
6		
7	I/O pins	36
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	clk~input
12	Maximum fan-out	50
13	Total fan-out	1124
14	Average fan-out	3.01

Block Diagram (Technology map viewer):



Timing reports:

Compilation Report - shift_multiply

Clocks

<<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Base	1.000	1000.0 MHz	0.000	0.500

For slow 85C model:

shift_multiply.v | test_bench.v | Compilation Report - shift_multiply

Table of Contents

- Parallel Compilation
- Clocks
- Slow 1100mV 85C Model
  - Fmax Summary

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	215.94 MHz	215.94 MHz	clk	

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-3.631	-99.885

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.349	0.000

Slow 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-26.960

For slow 0C model:

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	219.15 MHz	219.15 MHz	clk	

Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-3.563	-98.060

Slow 1100mV 0C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.331	0.000

Slow 1100mV 0C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.394	-28.695

For fast 85C model:

Fast 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-1.768	-45.016

Fast 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	0.178	0.000

Fast 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.084	-4.134

For fast 0C model:

Fast 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-1.509	-37.422

Fast 1100mV 0C Model Hold Summary

<<Filter>>

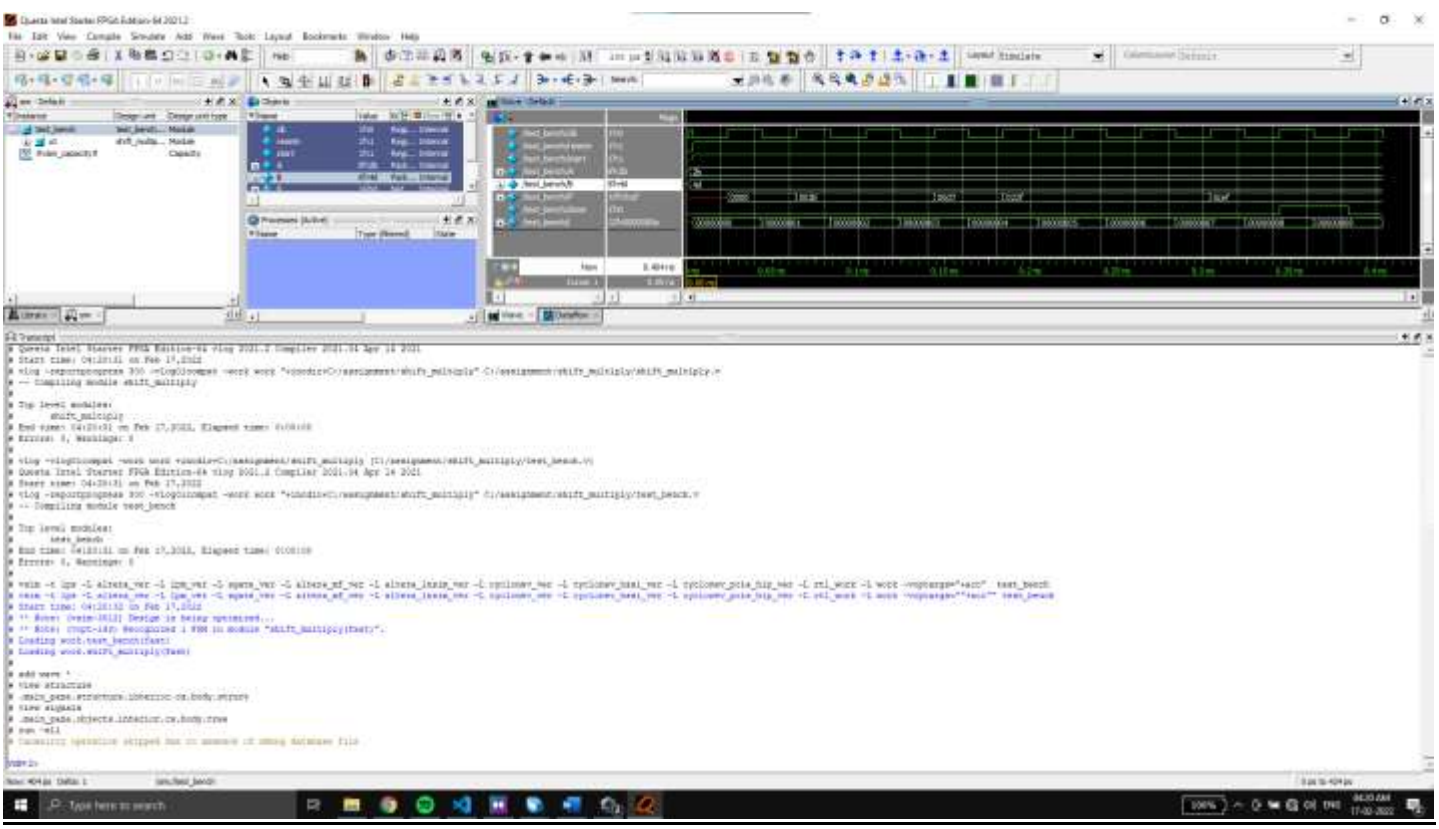
	Clock	Slack	End Point TNS
1	clk	0.169	0.000

Fast 1100mV 0C Model Minimum Pulse Width Summary

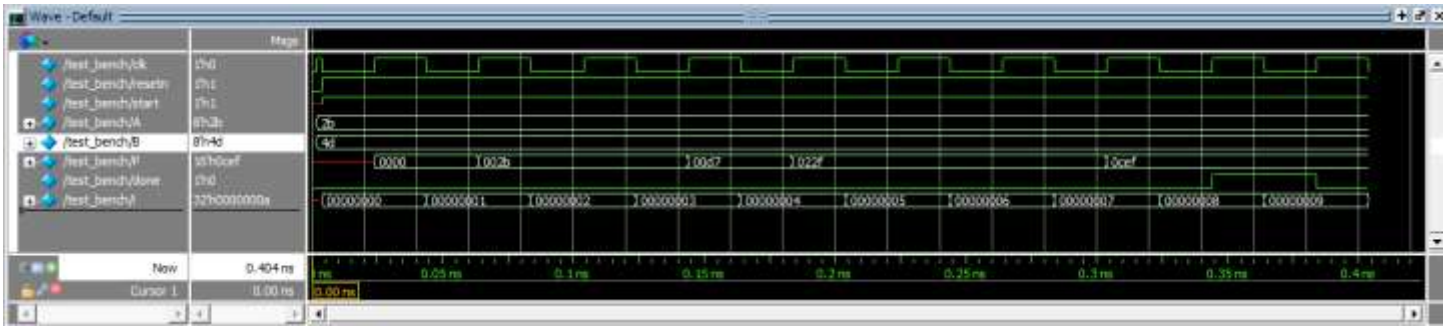
<<Filter>>

	Clock	Slack	End Point TNS
1	clk	-0.082	-4.059

Simulation in Questa through Quartus:



Wave:



5. To compute the factorial of a 3-bit number using RTL approach.

->

To design a system which calculates factorial of up to 3-bit numbers.

We will consider a 3bit input **a**, of which the factorial has to be computed and it will be displayed on the 13 bits output **out**.

We will consider an FSM with **8 states** with positive edge activated clock and asynchronous reset **resetn**.

In states 0 to 6, we will be multiplying the output with corresponding state value, and storing it in **out_next**.

On switch from negative edge to positive edge of the clock, **out_next** value will get stored in the output register **out**.

On reset, the output will revert to 0 and state to 0.

On competition of necessary factorial calculation, a condition will be met **(count<=state+1)** which will switch the state to **7** at which it will be stuck forever showing the required factorial and a **done** output bit will turn to 1 to indicate the completion of it.

The count register stores the value of input a.

We will assume that the input isn't 0.

Verilog Code for the Factorial calculator:

```
module factorial (input [2:0]a, input clock, resetn, output reg [12:0] out,
output reg done);
reg [3:0] state, next_state;
reg [2:0] count;
reg [12:0] out_next;

always @(posedge clock, negedge resetn)
begin
    if(!resetn) begin out <=1;state<=0; end
    else begin
    state <= next_state;
    out <= out_next;
    end
end

always @ (state) begin
    count <=a;
    case(state)
    0: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 1;
            done <=0;
            out_next <= 1*out;
            end
        end
    1: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 2;
            end
            done <=0;
            out_next <= 2*out;
        end
    2: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 3;
            end
            done <=0;
            out_next <= 3*out;
        end
    3: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 4;
            end
            done <=0;
            out_next <= 4*out;
        end
    4: begin
```

```
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 5;
            end
            done <=0;
            out_next <= 5*out;
        end
    5: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 6;
            end
            done <=0;
            out_next <= 6*out;
        end
    6: begin
            if(count<=state+1) begin next_state <= 7; end
            else begin next_state <= 7;
            end
            done <=0;
            out_next <= 7*out;
        end
    7: begin
            next_state <= 7;
            done <=1;
        end
    endcase
end
endmodule
```

Verilog code for the test bench:

```
module testbench();
reg [2:0] a;
reg clock, resetn;
wire [12:0] out;
wire done;
integer i;
factorial faccalc(a, clock, resetn, out, done);
initial begin
a=5;
resetn=1; clock =1;
#5 resetn=0; #5 resetn=1;
for(i=0; i<10; i=i+1)begin
    clock=1;#50; clock=0; #50;
end

end
endmodule
```

Device Utilization Statistics:

Flow summary:

Compilation Report - factorial

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Feb 17 04:36:27 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	factorial
Top-level Entity Name	factorial
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	55 / 32,070 (< 1 %)
Total registers	16
Total pins	19 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Resource usage summary:

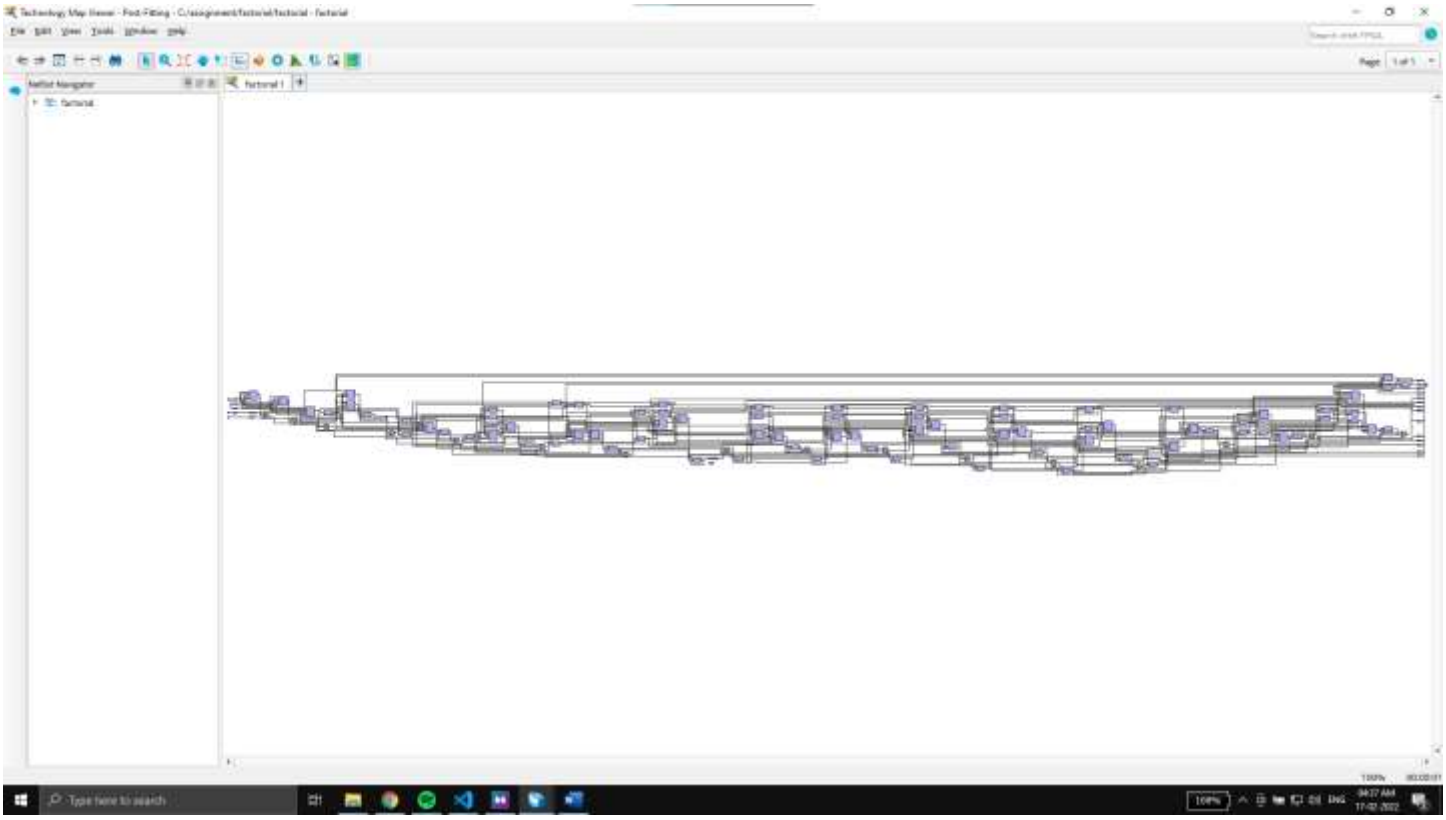
Compilation Report - factorial

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	55
2		
3	Combinational ALUT usage for logic	83
1	-- 7 input functions	1
2	-- 6 input functions	25
3	-- 5 input functions	3
4	-- 4 input functions	0
5	-- <=3 input functions	54
4		
5	Dedicated logic registers	16
6		
7	I/O pins	19
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	state[1]
12	Maximum fan-out	21
13	Total fan-out	406
14	Average fan-out	2.96

Block Diagram (Technology map viewer):



Timing reports:

Compilation Report - factorial

Clocks

<<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall
1	a[1]	Base	1.000	1000.0 MHz	0.000	0.500
2	clock	Base	1.000	1000.0 MHz	0.000	0.500

Compilation Report - factorial

Multicorner Timing Analysis Summary

<<Filter>>

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	-5.008	-0.421	N/A	N/A	-0.394
1	a[1]	-5.008	0.415	N/A	N/A	-0.077
2	clock	-2.391	-0.421	N/A	N/A	-0.394
2	Design-wide TNS	-82.814	-1.467	0.0	0.0	-9.1
1	a[1]	-63.294	0.000	N/A	N/A	-1.151
2	clock	-21.349	-1.467	N/A	N/A	-9.100

For slow 85C model:

Compilation Report - factorial

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	798.08 MHz	717.36 MHz	clock	lim...in)

Slow 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	a[1]	-4.898	-61.465
2	clock	-2.391	-21.349

Slow 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.299	-0.733
2	a[1]	1.068	0.000

Slow 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.394	-8.382
2	a[1]	0.161	0.000

For slow 0C model:

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	771.6 MHz	717.36 MHz	clock	lim...in)

Slow 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	a[1]	-5.008	-63.294
2	clock	-2.147	-19.478

Slow 1100mV 0C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.421	-1.467
2	a[1]	1.094	0.000

Slow 1100mV 0C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.394	-9.100
2	a[1]	0.135	0.000

For fast 85C model:

Fast 1100mV 85C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	a[1]	-2.208	-26.817
2	clock	-2.006	-16.764

Fast 1100mV 85C Model Hold Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	0.181	0.000
2	a[1]	0.437	0.000

Fast 1100mV 85C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.081	-1.252
2	a[1]	-0.077	-1.151

For fast 0C model:

Fast 1100mV 0C Model Setup Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	a[1]	-2.005	-24.527
2	clock	-1.714	-14.865

Fast 1100mV 0C Model Hold Summary

<<Filter>>

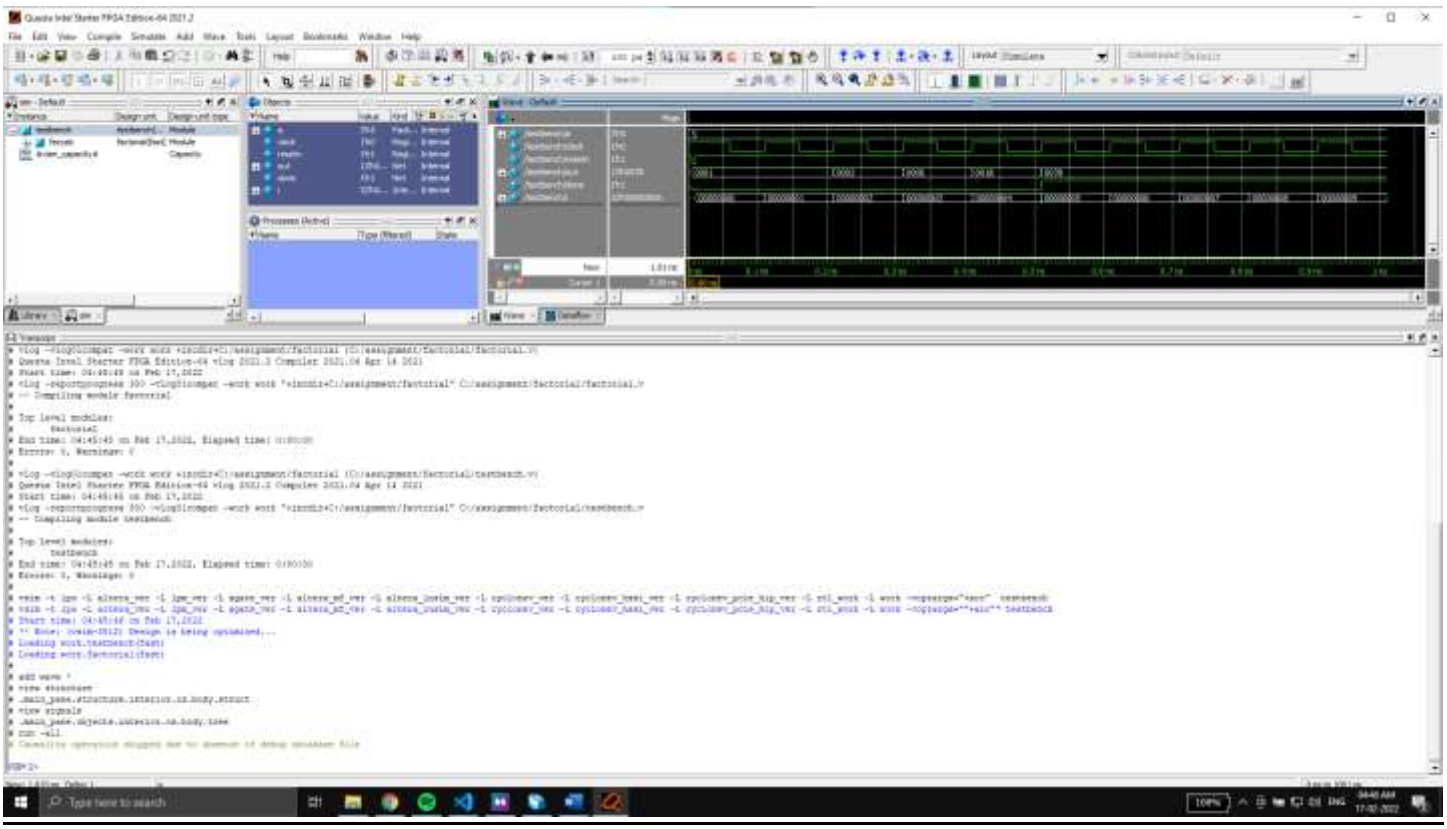
	Clock	Slack	End Point TNS
1	clock	0.173	0.000
2	a[1]	0.415	0.000

Fast 1100mV 0C Model Minimum Pulse Width Summary

<<Filter>>

	Clock	Slack	End Point TNS
1	clock	-0.080	-1.238
2	a[1]	-0.071	-1.055

Simulation in Questa through Quartus:



Wave: