INTEL UNNATI INDUSTRIAL TRAINING SUMMER 2023

PROBLEM STATEMENT	**DESIGN & IMPLEMENTATION OF SLOW AND FAST DIVISION ALGORITHM IN COMPUTER ARCHITECTURE**
NAME	**Ms.N.TURIKA**
INSTITUTE NAME	**B.S.ABDUR RAHMAN CRESCENT INSTITUTE OF SCIENCE AND TECHNOLOGY**
TEAM NAME	**FIXER**
TEAM SIZE	**1**
MENTOR NAME	**DR.S.ANUSOOYA**

DESIGN & INPLEMENTATION OF SLOW AND FAST DIVISION ALGORITHM IN COMPUTER ARCHITECTURE

ABOUT DIVISION ALGORITHM

A division algorithm is a method or procedure used to perform division operations on numerical values. Division is an arithmetic operation that involves dividing one value (the dividend) by another value (the divisor) to obtain a quotient and a remainder.

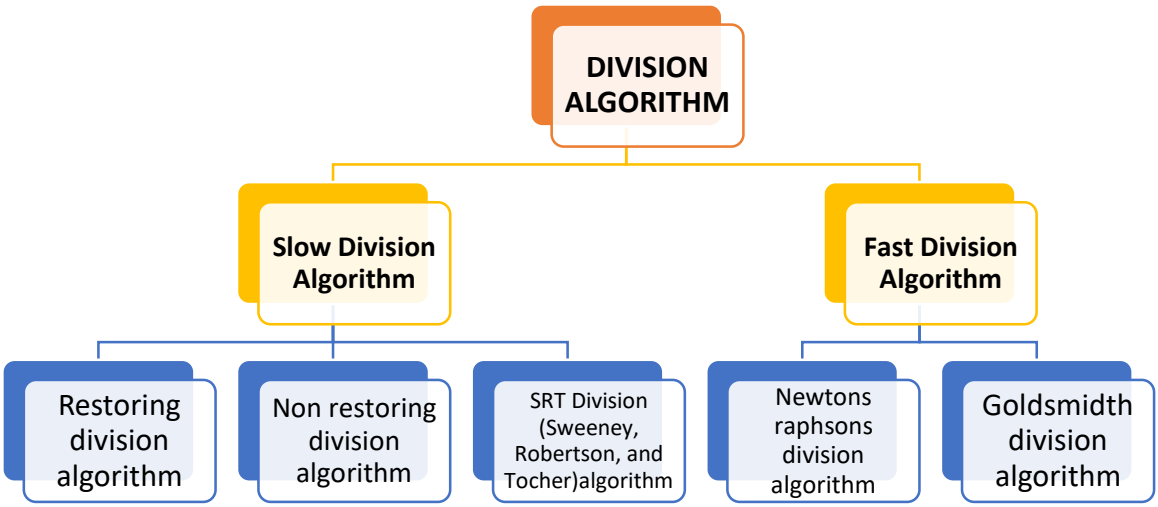
DIVISION ALGORITHM IN COMPUTER ARCHITECTURE

Division algorithms are essential in computer architecture because they provide a way to perform division operations using the available hardware resources and arithmetic operations supported by the computer system.

The design of division algorithms involves considerations such as accuracy, efficiency, speed, and resource utilization. Different division algorithms have varying trade-offs in terms of these factors.

Division algorithm in computer architecture uses registers for storing the numbers and calculations.

CLASSIFICATION OF DIVISION ALGORITHM



SLOW DIVISION ALGORITHM

Slow division algorithms produce one digit of the final quotient per iteration. Examples of slow division include restoring, non-restoring, and SRT division.

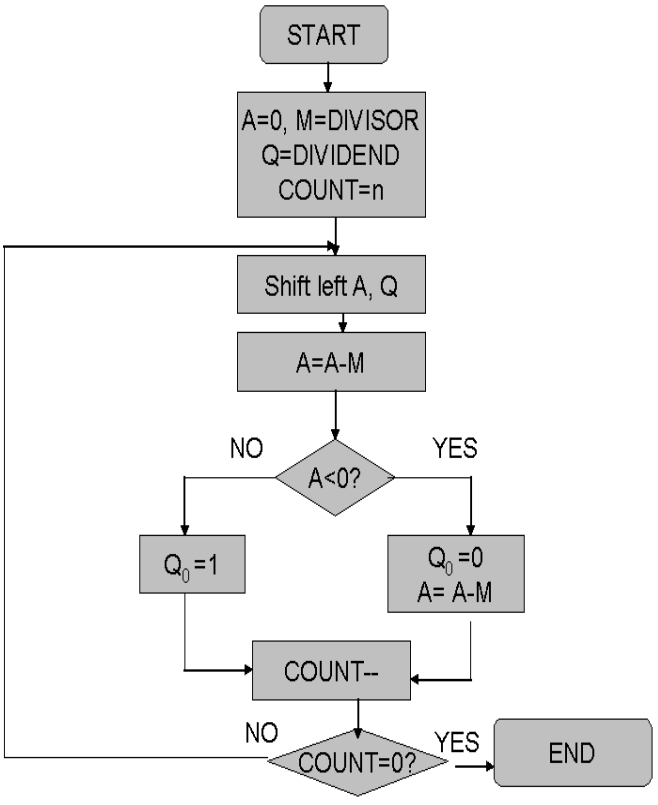
RESTORING DIVISION ALGORITHM

This algorithm is based on repeated subtraction and shifting to find the quotient and remainder. It involves restoring the dividend in each iteration and comparing it with the divisor. Due to this process, it has been named as Restoring Division Algorithm.

This algorithm is performed for an unsigned integer. Below are the letters and its operations applied in this algorithm.

- Register A contains remainder which will be restored after each iteration and initially it is kept as zero.
- Register Q contains quotient, where n-bit dividend is stored.
- M stores n-bit divisor.

FLOWCHART



MANUAL CALCULATION

TEST CASE (1)

Dividend (Nr)= 7 (place in Q) and Divisor(Dr) = 2 (place in M),count n=4

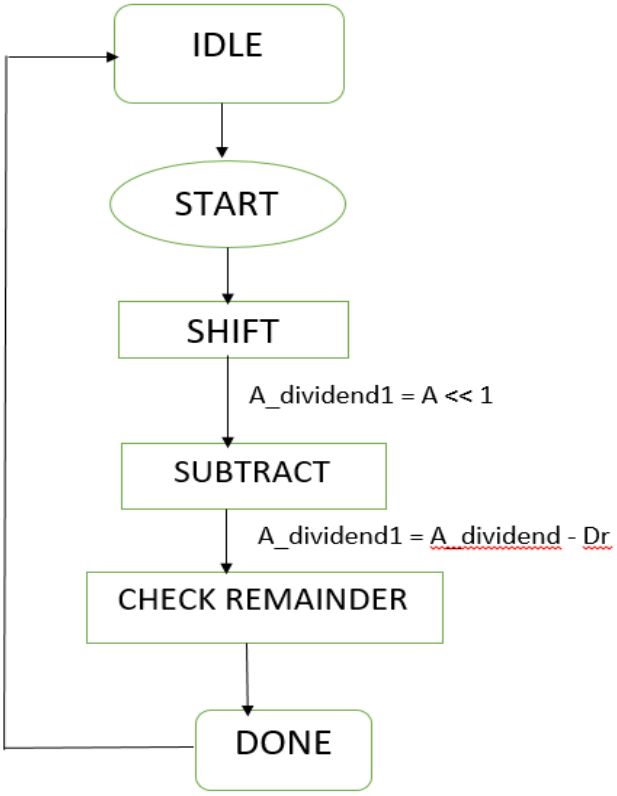
A	Q	OPERATION	COUNT
00000	0111	Initialize	4
00000	111_	Shift left A,Q	
11110	111_	A=A-M	
00000	1110	Q[0]<-0,restore A	
00001	110_	Shift left A,Q	3
11111	110_	A=A-M	
00001	1100	Q[0]<-0,restore A	
00011	100_	Shift left A,Q	2
00001	100_	A=A-M	
00001	1001	Q[0]<-1	
00011	001_	Shift left A,Q	1
00001	001_	A=A-M	
00001	0011	Q[0]<-1	
Final result : Quotient=0011 , Remainder=0001			0

TEST CASE (2)

Dividend (Nr)= 14 (place in Q) and Divisor(Dr) = 5 (place in M),count n=4

A	Q	OPERATION	COUNT
00000	1110	Initialize	4
00001	110_	Shift left A,Q	
11100	110_	A=A-M	
00001	1100	Q[O]<-0,restore A	
00011	100_	Shift left A,Q	3
11110	100_	A=A-M	
00011	1000	Q[O]<-0,restore A	
00111	000_	Shift left A,Q	2
00010	000_	A=A-M	
00010	0001	Q[O]<-1	
00100	001_	Shift left A,Q	1
11111	001_	A=A-M	
00100	0010	Q[O]<-0,restore A	
Final result : Quotient=0010 , Remainder=0100			0

MEALY STATE MACHINE



VERILOG CODE LOGIC

Input & Output Assumptions

Restoring Division algorithm for 4-bit operands in an FSM (Mealy State Machine) format with completely synthesizable Verilog Code.

Inputs

clk, reset, Nr, Dr (2 operands to be multiplied), start (This pulse indicates start of computation)

Outputs

Q (Quotient), R (Remainder) and done (This pulse indicates the availability of final answer)

1. In IDLE state, we wait for the start pulse. Upon its arrival, move to START state.

2. In START state, we follow the same algorithm and perform the computations using verilog logic as long as counter < 3. (Incrementing counter is used)

3. Upon counter completion, we send out done pulse and goto IDLE state.

VERILOG CODE

```
module slow_division1(clk,reset,start,Nr,Dr,done,Q,R);

input clk;
input reset;
input start;
input [3:0]Nr,Dr;
output [3:0]Q,R;
output done;

reg [7:0] A,next_A,A_dividend,A_dividend1;
reg next_state, pres_state;
reg [1:0] count,next_count;
reg done, next_done;

parameter IDLE = 1'b0;
parameter START = 1'b1;

assign R = A[7:4];
assign Q = A[3:0];

always @ (posedge clk or negedge reset)
begin
if(!reset)
begin
  A         <= 8'd0;
  done      <= 1'b0;
  pres_state <= 1'b0;
  count     <= 2'd0;
end
else
begin
  A         <= next_A;
  done      <= next_done;
  pres_state <= next_state;
  count     <= next_count;
end
end
```

```
always @ (*)
begin
case(pres_state)
IDLE:
begin
next_count = 2'b0;
next_done = 1'b0;
if(start)
begin
   next_state = START;
   next_A     = {4'd0,Nr};
end
else
begin
   next_state = pres_state;
   next_A     = 8'd0;
end
end

START:
begin
next_count = count + 1'b1;
A_dividend     = A << 1;
A_dividend1    = {A_dividend[7:4]-Dr,A_dividend[3:0]};
next_A     = A_dividend1[7] ? {A_dividend[7:4],A_dividend[3:1],1'b0} :
                    {A_dividend1[7:4],A_dividend[3:1],1'b1};
next_done = (&count) ? 1'b1 : 1'b0;
next_state = (&count) ? IDLE : pres_state;
end
endcase
end
endmodule
```

TEST BENCH CODE

```
module slow_division1_tb;

reg clk,reset,start;
reg [3:0]Nr,Dr;
wire [3:0]Q,R;
wire done;

always #5 clk = ~clk;

slow_division1 inst (clk,reset,start,Nr,Dr,done,Q,R
);
```

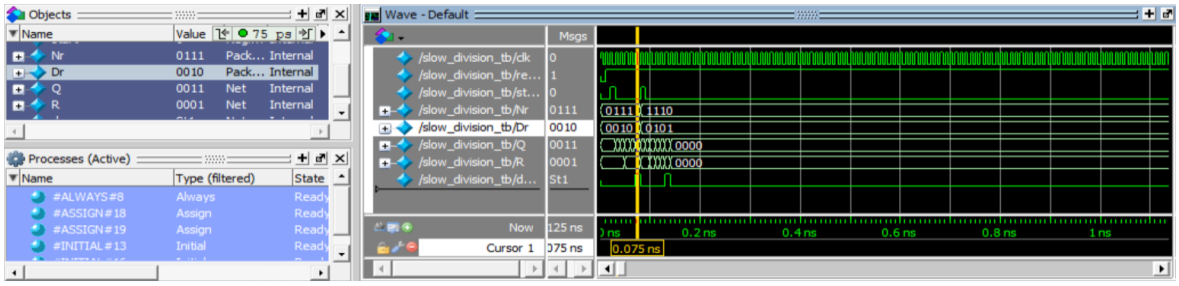
```
initial
$monitor($time,"Nr=%d, Dr=%d, done=%d, Q=%d, R=%d ",Nr,Dr,done,Q,R);

initial
begin
Nr=7;Dr=2;clk=1'b1;reset=1'b0;start=1'b0;
#10 reset = 1'b1;
#10 start = 1'b1;
#10 start = 1'b0;
@done
#10 Nr=14;Dr=5;start = 1'b1;
#10 start = 1'b0;
end
endmodule
```

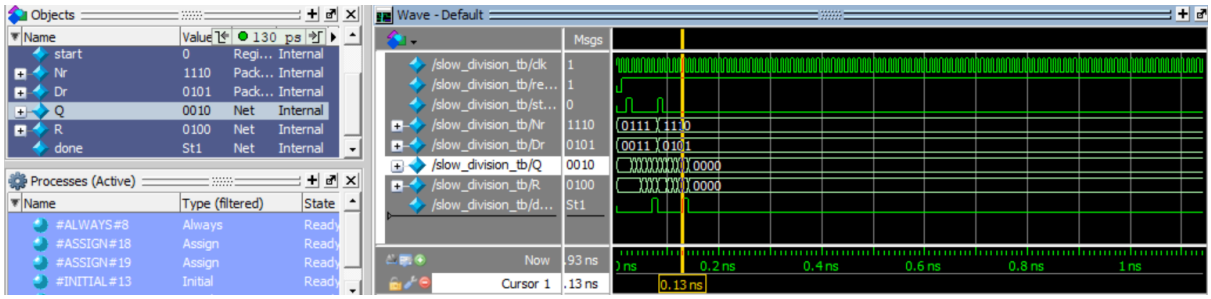
SIMULATION RESULT

```
  0Nr= 7, Dr= 2, done=0, Q= 0, R= 0
 30Nr= 7, Dr= 2, done=0, Q= 7, R= 0
 40Nr= 7, Dr= 2, done=0, Q=14, R= 0
 50Nr= 7, Dr= 2, done=0, Q=12, R= 1
 60Nr= 7, Dr= 2, done=0, Q= 9, R= 1
 70Nr= 7, Dr= 2, done=1, Q= 3, R= 1
 80Nr=14, Dr= 5, done=0, Q= 0, R= 0
 90Nr=14, Dr= 5, done=0, Q=14, R= 0
100Nr=14, Dr= 5, done=0, Q=12, R= 1
110Nr=14, Dr= 5, done=0, Q= 8, R= 3
120Nr=14, Dr= 5, done=0, Q= 1, R= 2
130Nr=14, Dr= 5, done=1, Q= 2, R= 4
140Nr=14, Dr= 5, done=0, Q= 0, R= 0
```

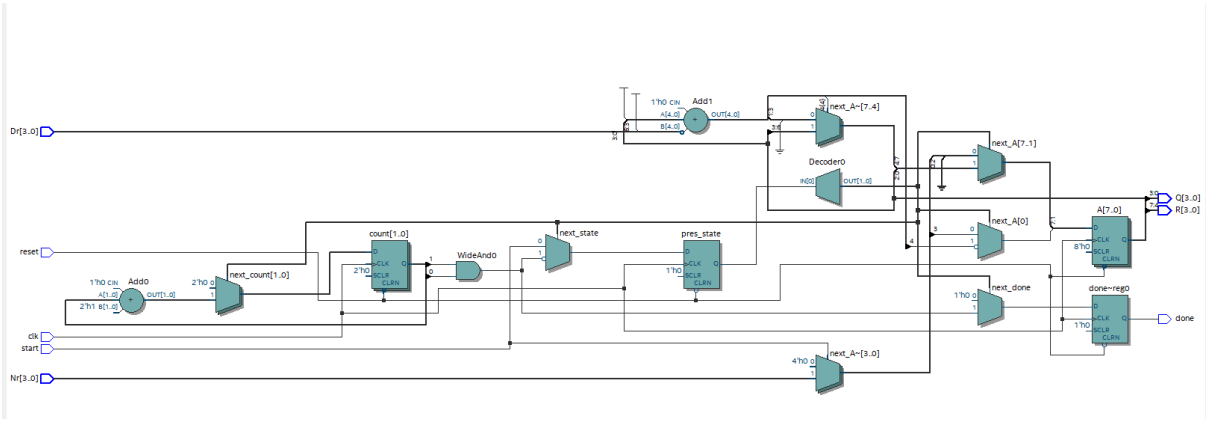
RESULT OF TEST CASE (1)



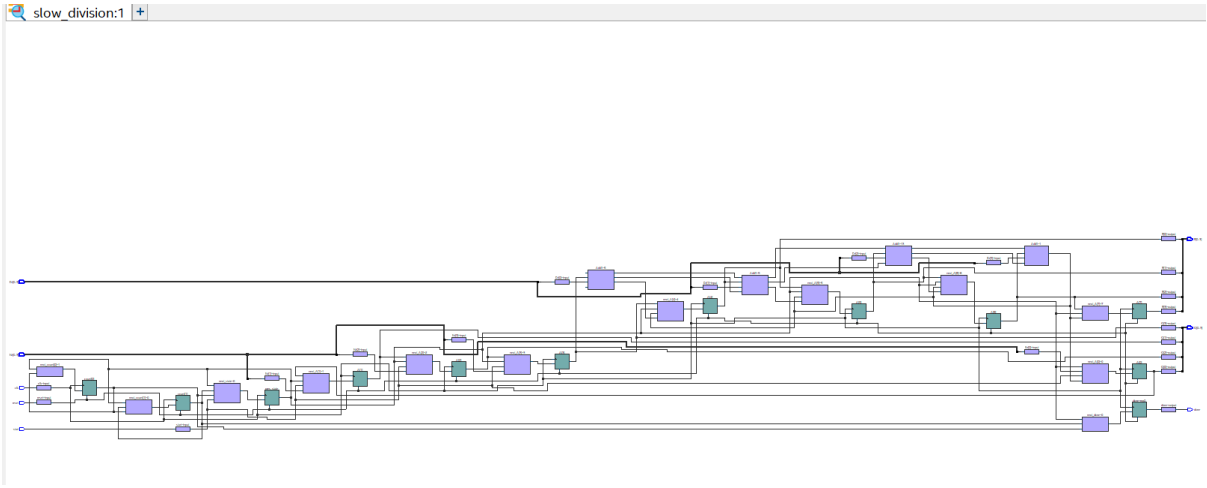
RESULT OF TEST CASE (2)



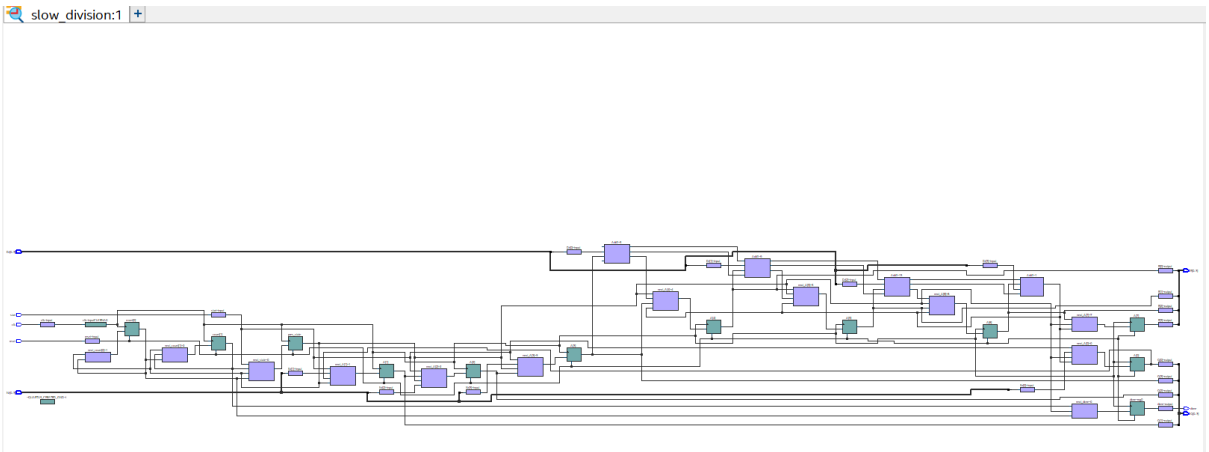
RTL VIEWER



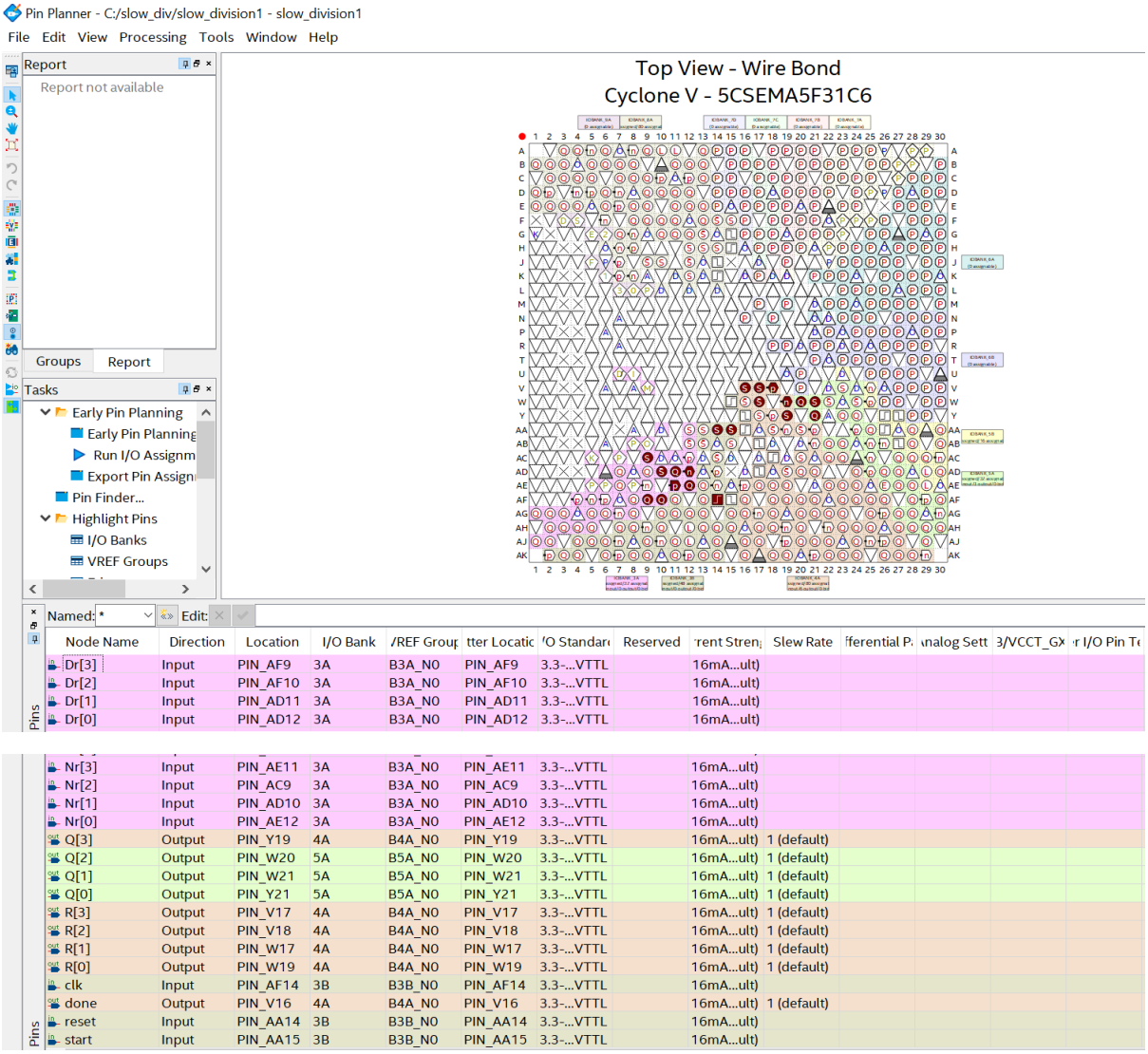
TECHNOLOGY MAP VIEWER (POST MAPPING)



TECHNOLOGY MAP VIEWER (POST FITTING)

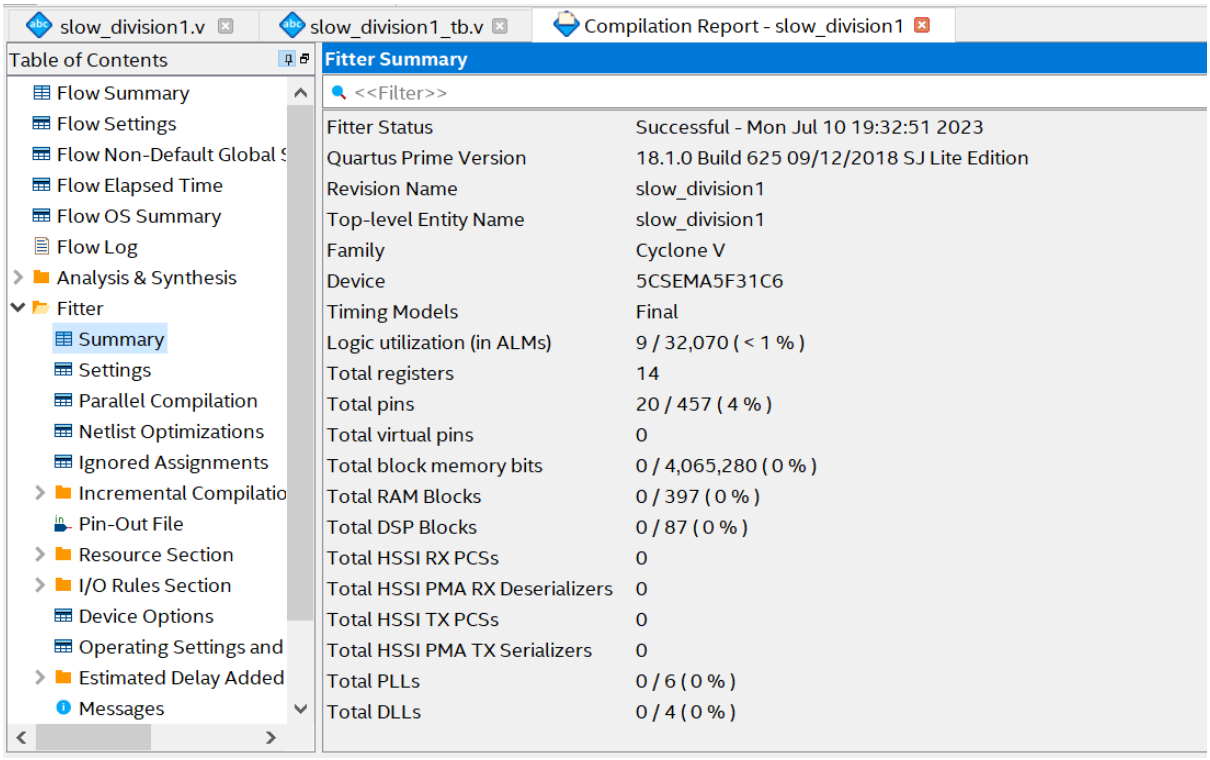


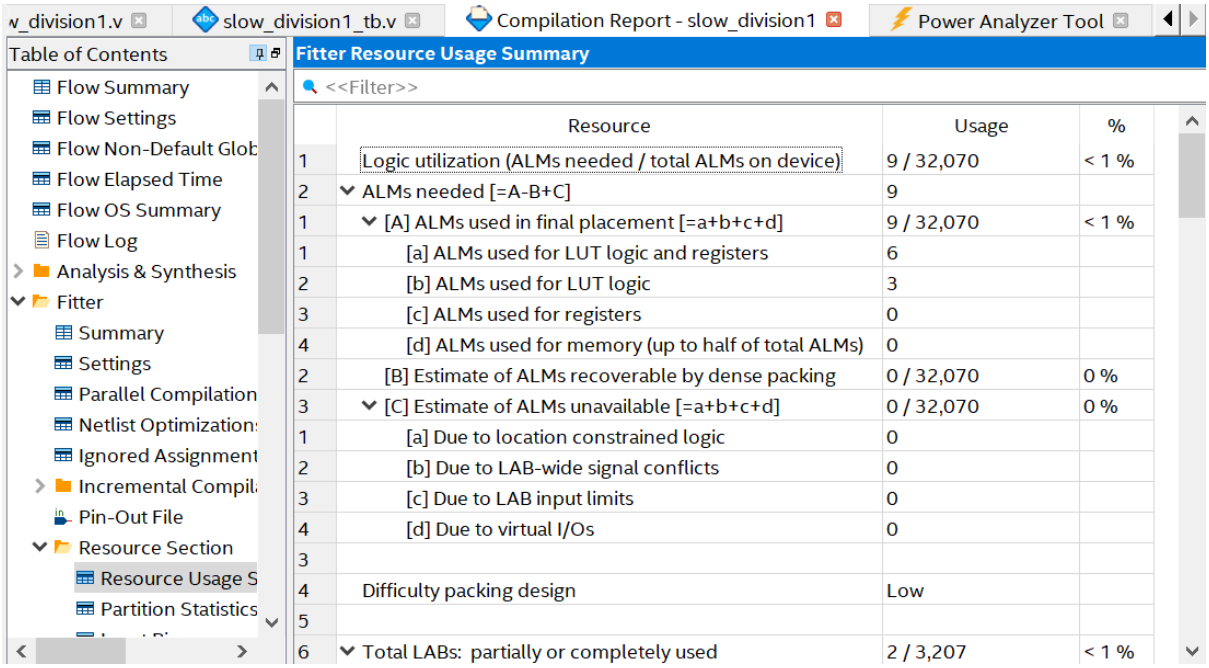
PIN PLANNER

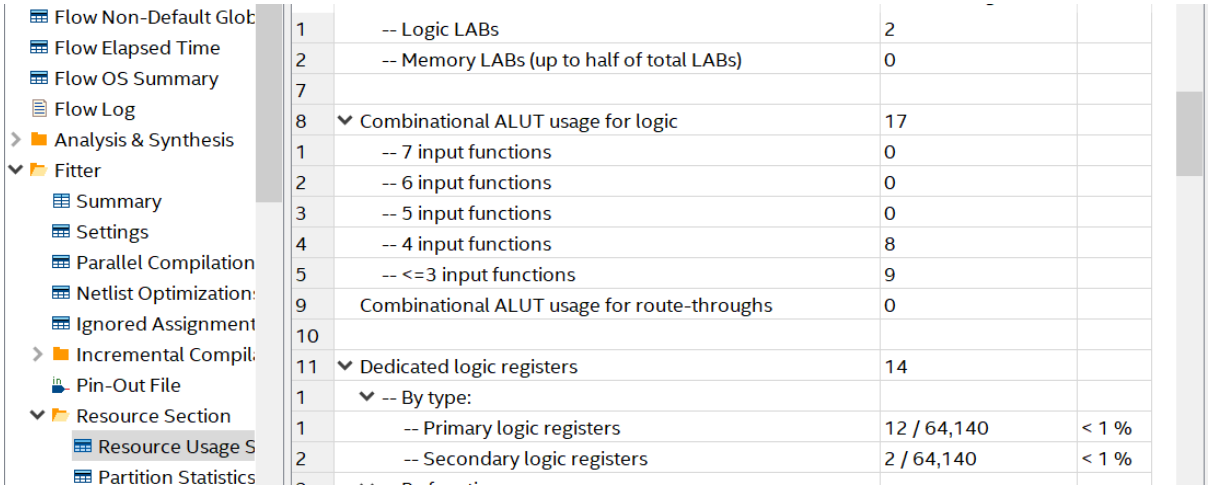


CHIP PLANNER

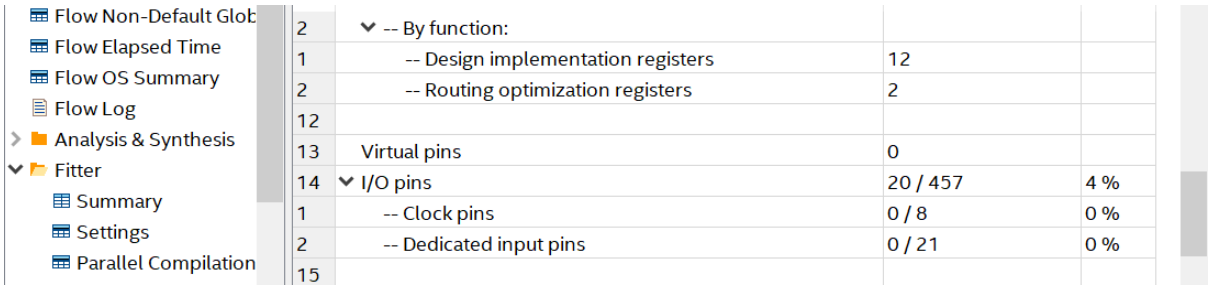
AREA REPORT


slow_division1.v
slow_division1_tb.v
Compilation Report - slow_division1
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global S
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Summary
Settings
Parallel Compilation
Netlist Optimizations
Ignored Assignments
Incremental Compilatio
Pin-Out File
Resource Section
I/O Rules Section
Device Options
Operating Settings and
Estimated Delay Added
Messages
Fitter Summary
<<Filter>>
Fitter Status Successful - Mon Jul 10 19:32:51 2023
Quartus Prime Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name slow_division1
Top-level Entity Name slow_division1
Family Cyclone V
Device 5CSEMA5F31C6
Timing Models Final
Logic utilization (in ALMs) 9 / 32,070 (< 1 %)
Total registers 14
Total pins 20 / 457 (4 %)
Total virtual pins 0
Total block memory bits 0 / 4,065,280 (0 %)
Total RAM Blocks 0 / 397 (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 %)

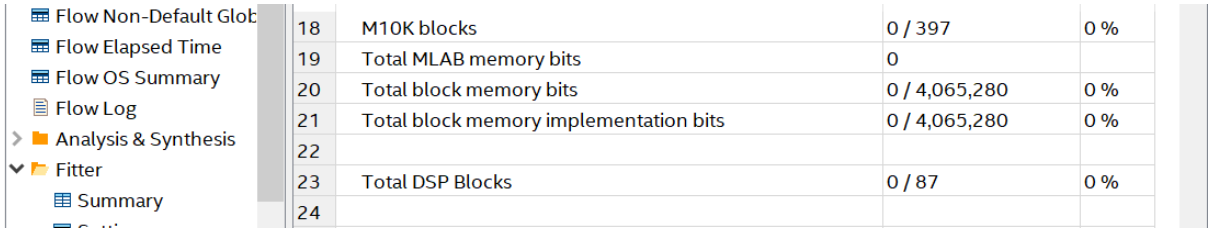

v_division1.v
slow_division1_tb.v
Compilation Report - slow_division1
Power Analyzer Tool
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Glob
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Summary
Settings
Parallel Compilation
Netlist Optimization
Ignored Assignment
Incremental Compil
Pin-Out File
Resource Section
Resource Usage S
Partition Statistics
Fitter Resource Usage Summary
<<Filter>>
Resource Usage %
1 Logic utilization (ALMs needed / total ALMs on device) 9 / 32,070 < 1 %
2 ALMs needed [=A-B+C] 9
1 [A] ALMs used in final placement [=a+b+c+d] 9 / 32,070 < 1 %
1 [a] ALMs used for LUT logic and registers 6
2 [b] ALMs used for LUT logic 3
3 [c] ALMs used for registers 0
4 [d] ALMs used for memory (up to half of total ALMs) 0
2 [B] Estimate of ALMs recoverable by dense packing 0 / 32,070 0 %
3 [C] Estimate of ALMs unavailable [=a+b+c+d] 0 / 32,070 0 %
1 [a] Due to location constrained logic 0
2 [b] Due to LAB-wide signal conflicts 0
3 [c] Due to LAB input limits 0
4 [d] Due to virtual I/Os 0
3
4 Difficulty packing design Low
5
6 Total LABs: partially or completely used 2 / 3,207 < 1 %



1	-- Logic LABs	2	
2	-- Memory LABs (up to half of total LABs)	0	
7			
8	Combinational ALUT usage for logic	17	
1	-- 7 input functions	0	
2	-- 6 input functions	0	
3	-- 5 input functions	0	
4	-- 4 input functions	8	
5	-- <=3 input functions	9	
9	Combinational ALUT usage for route-throughs	0	
10			
11	Dedicated logic registers	14	
1	-- By type:		
1	-- Primary logic registers	12 / 64,140	< 1 %
2	-- Secondary logic registers	2 / 64,140	< 1 %

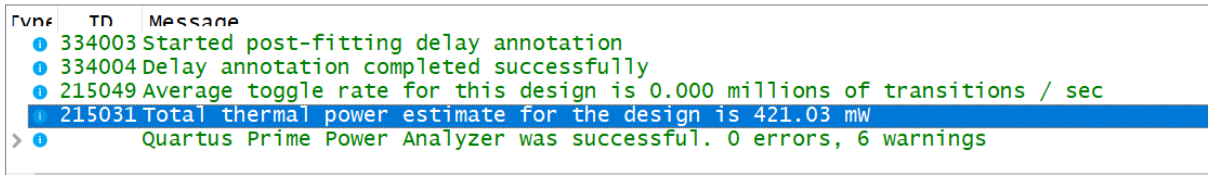


2	-- By function:		
1	-- Design implementation registers	12	
2	-- Routing optimization registers	2	
12			
13	Virtual pins	0	
14	I/O pins	20 / 457	4 %
1	-- Clock pins	0 / 8	0 %
2	-- Dedicated input pins	0 / 21	0 %
15			

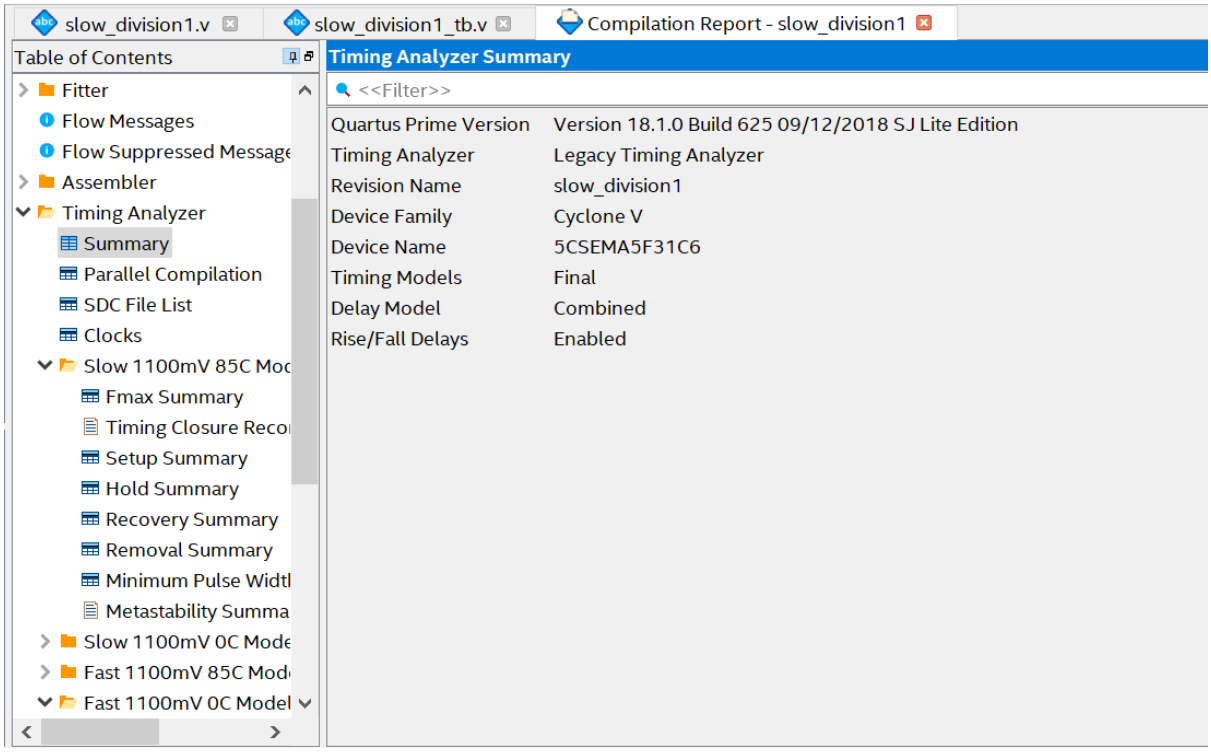


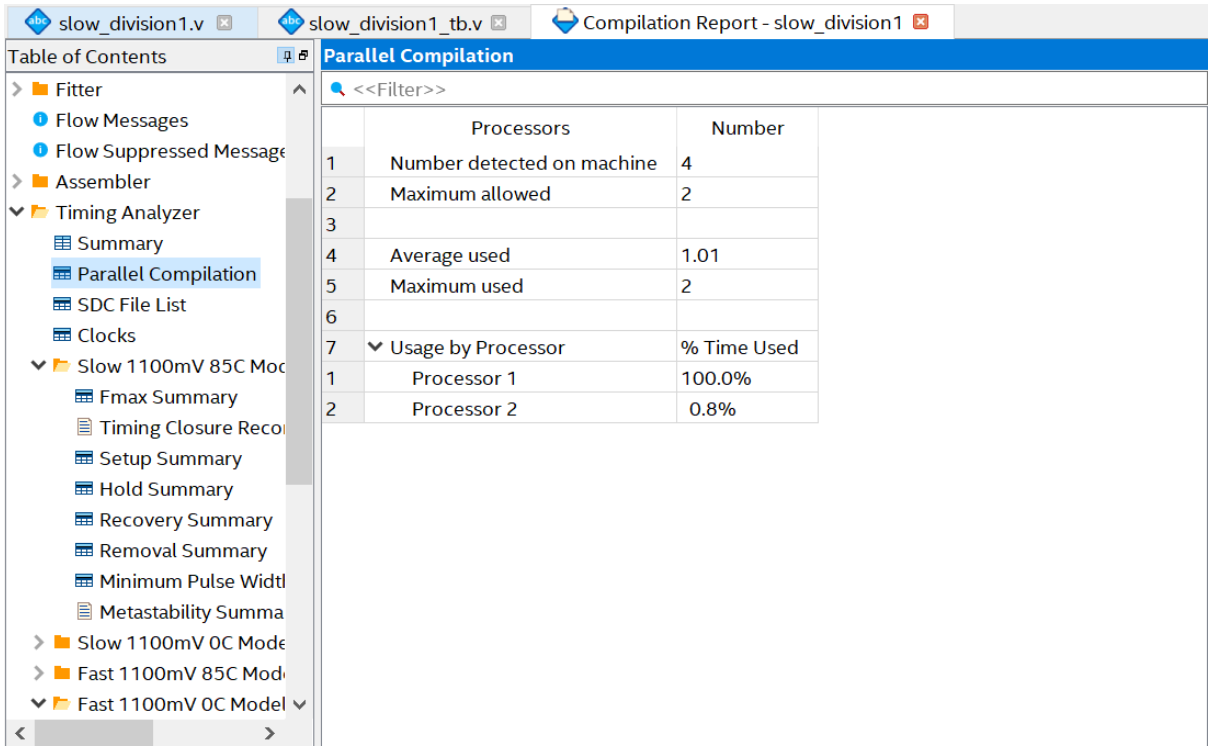
18	M10K blocks	0 / 397	0 %
19	Total MLAB memory bits	0	
20	Total block memory bits	0 / 4,065,280	0 %
21	Total block memory implementation bits	0 / 4,065,280	0 %
22			
23	Total DSP Blocks	0 / 87	0 %
24			

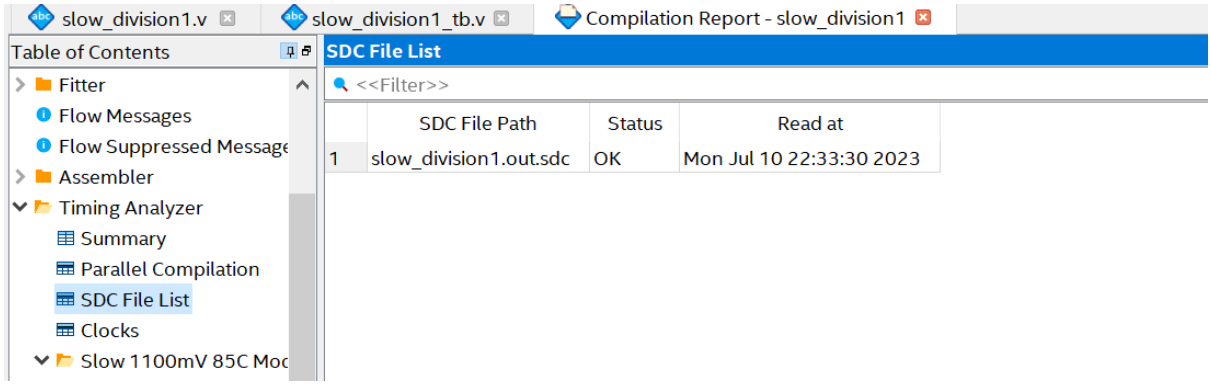
POWER REPORT

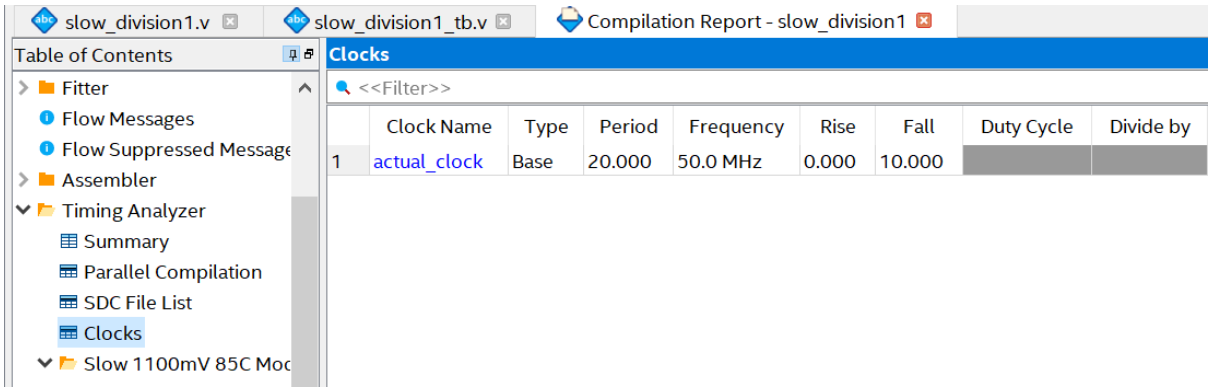


TIMING REPORT

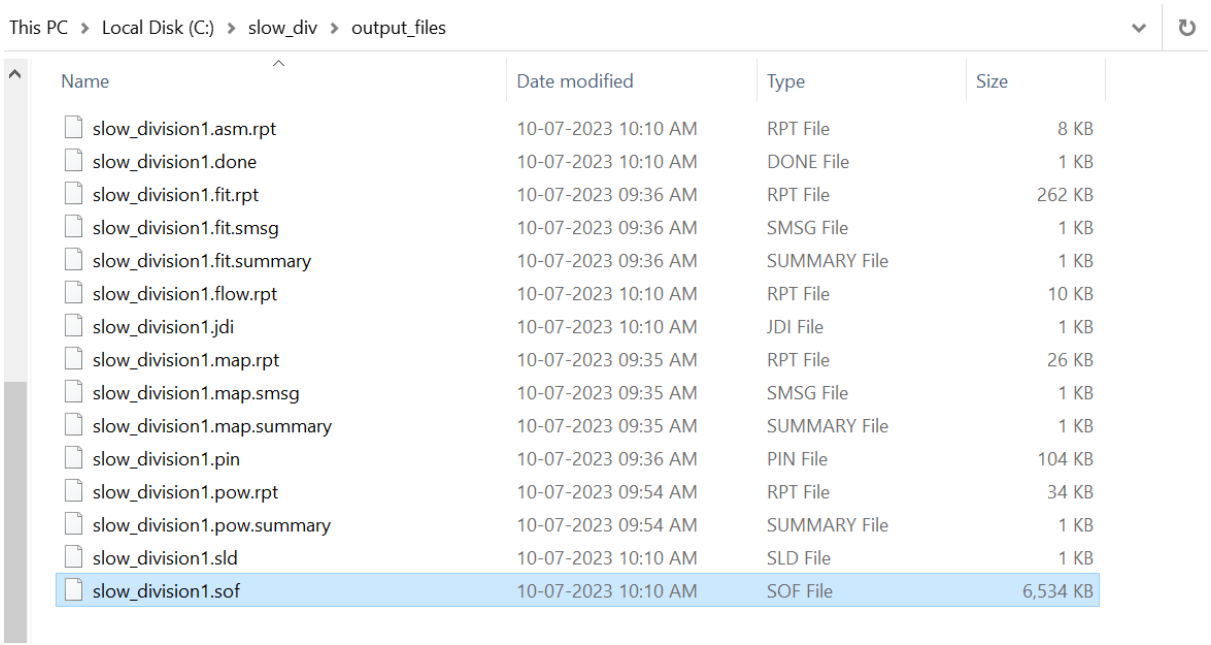








CREATION OF .sof FILE



IMPLEMENTATION ON INTEL FPGA LABORATORY

TEST CASE (1)

TEST CASE (2)