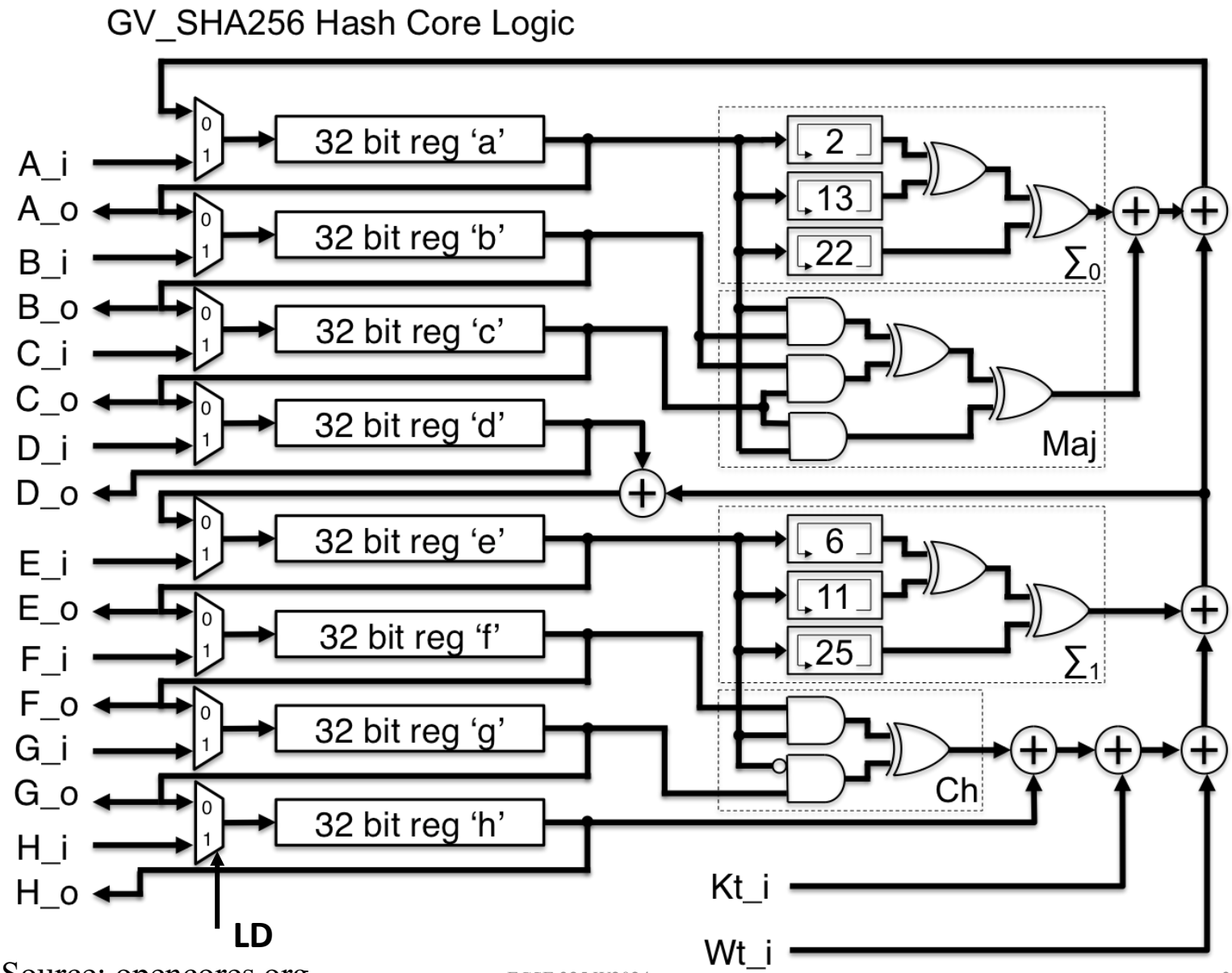
ECSE-325
Digital Systems

Lab #3 – *Static Timing Analysis and Pipelining* **Winter 2024**

Part 1 : Timing Analysis.

There are two parts to this lab. Each should take you one two-hour lab session, hence this will be a two-week lab.

In this lab you will describe and analyze the remaining parts of the Hash Core Logic.



Source: opencores.org

VHDL Description of the Hash_Core circuit.

Use the following form for the VHDL entity declaration of the Hash_Core circuit (replace the values in the header with your own information).

```
--
-- entity name: gNN_Hash_Core(replace "NN" by your group's number)
--
-- Version 1.0
-- Authors: (list the group member names here)
-- Date: March ??, 2024 (enter the date of the latest edit to the file)

library ieee; -- allows use of the std_logic_vector type
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- needed if you are using unsigned rotate operations

entity gNN_Hash_Core is
 port ( A_o, B_o, C_o, D_o, E_o, F_o, G_o, H_o  : inout std_logic_vector(31 downto 0);
          A_i, B_i, C_i, D_i, E_i, F_i, G_i, H_i : in std_logic_vector(31 downto 0);
          Kt_i, Wt_i : in std_logic_vector(31 downto 0);
          LD, CLK: in std_logic
);
end gNN_Hash_Core;
```

VHDL Description of the Hash_Core circuit.

You need to complete the VHDL description by adding in the architecture section.

Some notes:

- The addition operations are all 32-bit and do not have any carry-in or carry-out (hence the sum values will wrap-around back to 0) – just use the "+" operation.
- You will need to convert the std_logic_vectors to unsigned before adding, and then back to std_logic_vectors after adding.
- Implement the registers with a single clocked process block. All other operations (e.g., the SIG, CH, MAJ and additions) should be implemented outside of the process block.
- Use a component instantiation statement to include the gNN_SIG_CH_MAJ circuit you designed in lab #2.
- The "*inout*" port type for A_o, B_o,…H_o" allows these signals to be used as outputs as well as able to be assigned to internally.
- When the LD input is high the registers are loaded with the initial input values. When LD is low the registers will act like a large shift register (actually a special type of shift register called a *Linear Feedback Shift Register* (LFSR) which we will cover in a later lecture).
- The normal operation of the Hash_Core during the SHA-256 process is that LD will be high for one clock cycle, then low for 64 clocks cycles during which time the initial inputs will be scrambled in a practically irreversible manner. After these 64 cycles the register outputs will be concatenated to form one 256-bit vector which represents the Hash code.

VHDL Description of the SHA256 circuit.

You will need a top-level file for the entire SHA256 circuit, which will ultimately connect to pins on the FPGA and communicate with the outside world.

In this lab you will just use a *skeleton version* of this, for the purposes of analyzing the timing of the Hash_Core circuit. You will add to and modify this file in later labs.

Use the following form for the VHDL entity declaration of the SHA256 circuit (replace the values in the header with your own information).

```
--
-- entity name: gNN_SHA256 (replace "NN" by your group's number)
--
-- Version 1.0
-- Authors: (list the group member names here)
-- Date: March ??, 2021 (enter the date of the latest edit to the file)

library ieee; -- allows use of the std_logic_vector type
use ieee.std_logic_1164.all;

entity gNN_SHA256 is  -- to be changed in future labs
 port (
   hash_out  : out std_logic_vector(255 downto 0);
          RESET, CLK : in std_logic
);
end gNN_SHA256;
```

Use the VHDL code shown below for the SHA256 skeleton circuit (you don't need to edit anything now, but you will in future labs). Name this file gNN_SHA256.vhd

```
architecture lab3 of gNN_SHA256 is
signal Kt : std_logic_vector(31 downto 0) := X"428a2f98";
signal Wt : std_logic_vector(31 downto 0) := x"00000000";

signal h0 : std_logic_vector(31 downto 0) := x"6a09e667";  -- initial hash values
signal h1 : std_logic_vector(31 downto 0) := x"bb67ae85";
signal h2 : std_logic_vector(31 downto 0) := x"3c6ef372";
signal h3 : std_logic_vector(31 downto 0) := x"a54ff53a";
signal h4 : std_logic_vector(31 downto 0) := x"510e527f";
signal h5 : std_logic_vector(31 downto 0) := x"9b05688c";
signal h6 : std_logic_vector(31 downto 0) := x"1f83d9ab";
signal h7 : std_logic_vector(31 downto 0) := x"5be0cd19";

type SHA_256_STATE is ( INIT, RUN, IDLE);
signal state : SHA_256_STATE;
signal A_o, B_o, C_o, D_o, E_o, F_o, G_o, H_o  : std_logic_vector(31 downto 0);
signal LD : std_logic;
signal round_count : integer range 0 to 63;

COMPONENT g00_Hash_Core
          PORT (
   A_o, B_o, C_o, D_o, E_o, F_o, G_o, H_o  : inout std_logic_vector(31 downto 0);
          A_i, B_i, C_i, D_i, E_i, F_i, G_i, H_i : in std_logic_vector(31 downto 0);
          Kt_i, Wt_i : in std_logic_vector(31 downto 0);
          LD, CLK : in std_logic
);
END COMPONENT;

-- continued on next slide…
```

```
begin
i1 : g00_Hash_Core
          PORT MAP (A_o => A_o,B_o => B_o,C_o => C_o,D_o => D_o,E_o => E_o,
          F_o => F_o,G_o => G_o,H_o > H_o,A_i => h0,B_i => h1,C_i => h2,D_i => h3,
          E_i => h4,F_i => h5,G_i => h6,H_i => h7,Kt_i => Kt,Wt_i => Wt,
          LD => LD, CLK => CLK );

                    hash_out <= A_o & B_o & C_o & D_o & E_o & F_o & G_o & H_o;
-- concatenate to form the 256 bit hash output

                    process(RESET, CLK)
                    begin
                              if RESET = '1' then
                                        state <= INIT;
                              elsif rising_edge(CLK) then
                                        case state is
                                        when INIT =>
                                                  LD <= '1';
                                                  state <= RUN;
                                                  round_count <= 0;
                                        when RUN =>
                                                  LD <= '0';
                                                  round_count <= round_count + 1;
                                                  if round_count = 63 then
                                                            state <= IDLE;
                                                  end if;
                                        when IDLE =>
                                                  LD <= '1';
                                        end case;
                              end if;
                    end process;

end lab3;
```

Static Timing Analysis of the `gNN_Hash_Core` circuit

Run the Quartus program and create a new project called gNN_SHA256 (where "NN" is your group number). This will be the top-level project you will use for the remainder of the course. It will be modified and added to in the following labs.

As in lab #1 set the device to Cyclone V 5CSEMA5F31C6

Add the VHDL files you just wrote (gNN_SHA256.vhd and gNN_Hash_Core.vhd) to the Project. Also add the gNN_SIG_CH_MAJ.vhd file from lab #2.

Set gNN_SHA256.vhd as the top level of the project (to do this, select the gNN_SHA256.vhd tab in the Quartus editor, then select "*Set as Top-Level Entity*" from the *Project* menu)

Using the TimeQuest Timing Analyzer .

To ensure a properly working circuit, the designer must take into consideration various timing constraints. In class we saw that for a register to correctly store an input value, the input must be held stable for a period (called the ***setup time***) before the clock edge, and also for a period (called the ***hold time***) after the clock edge.

Whether a circuit meets these timing constraints can only be known after the circuit is *synthesized*. After synthesis is done one can analyze the circuit to see if the timing constraints (setup and hold times) are satisfied.

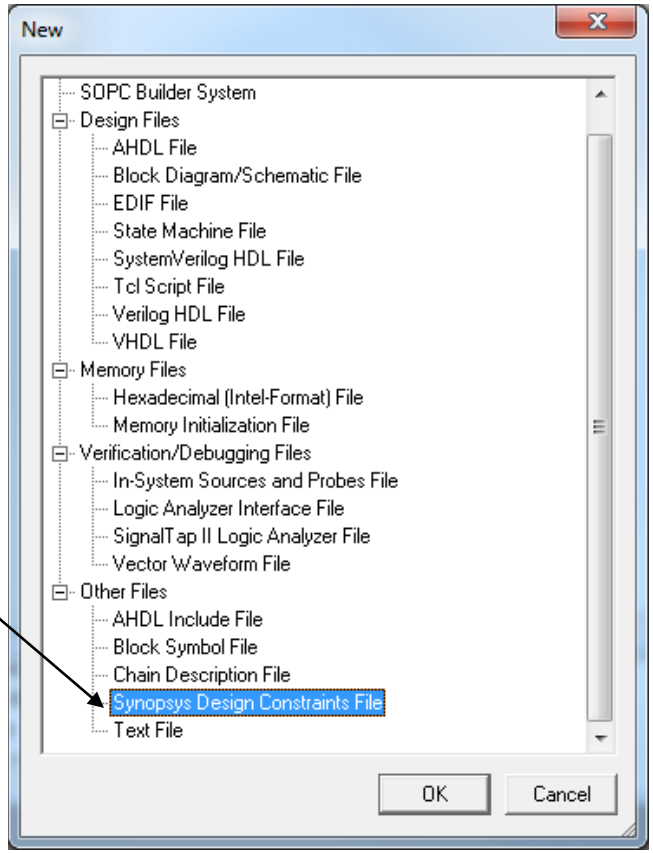
In Quartus, timing analysis can be done using the ***TimeQuest Timing Analyzer***.

From the “File/New” menu item, select “Synopsys Design Constraints File”.

This “.sdc” file is where we can specify various timing constraints for our design.

We will add a single constraint specifying the clock period.

The Quartus Fitter (the process that maps the design to the FPGA logic array) will attempt to do the mapping so as to meet the constraints.



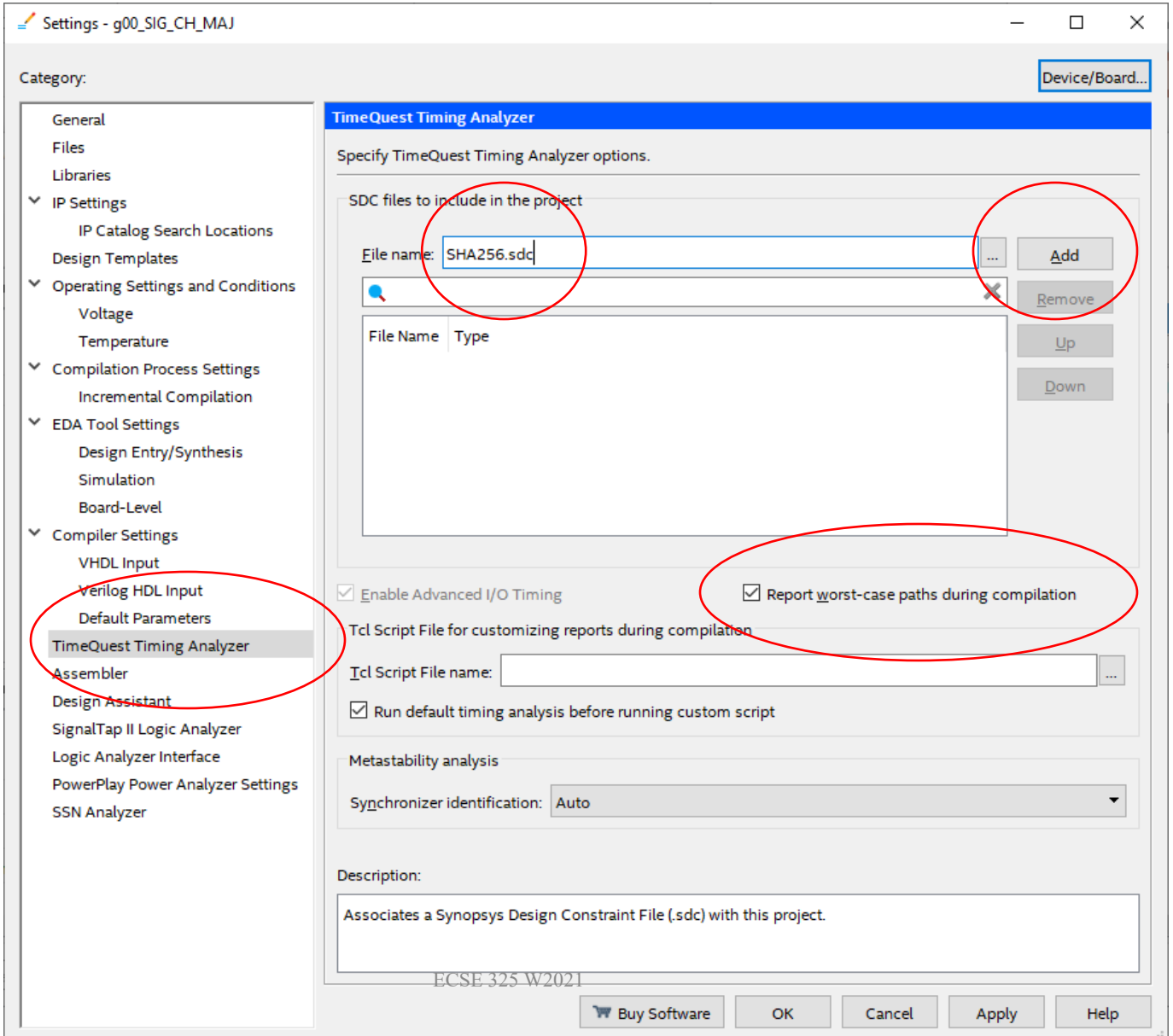
The gNN_SHA256.sdc file will be used to tell the timing analyzer that your clock period is ***6ns*** (corresponding to a clock frequency of 166MHz).

Enter the following single line into the file:

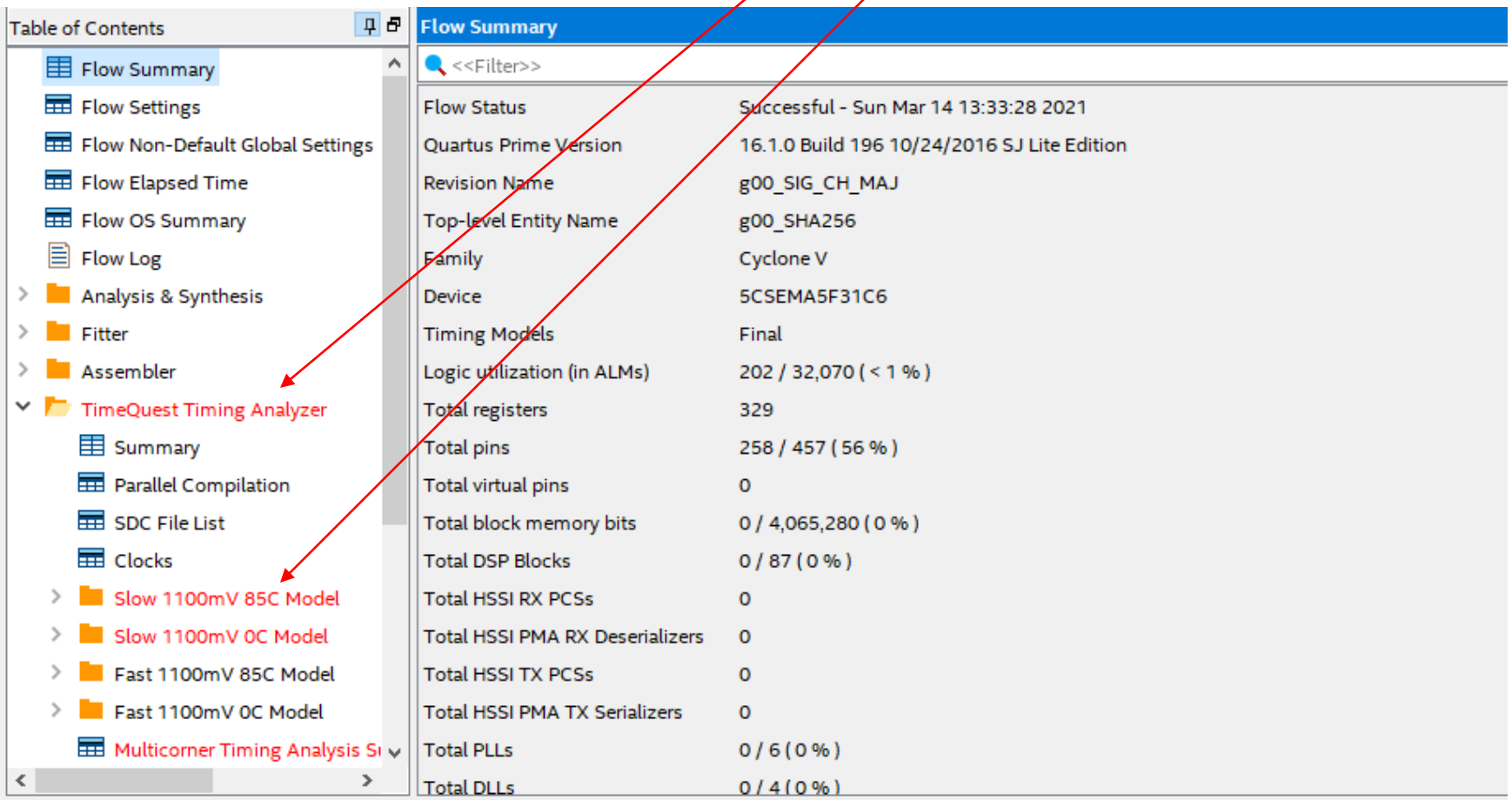
```
create_clock -period 6 [get_ports {clk}]
```

The Timing Analyzer will read the .sdc file and use the constraint information when doing its analysis.

To tell the TimeQuest Timing Analyzer to use your constraints file, go to the "Settings" item under the "Assignments" menu item. Add the file "SHA256.sdc". Also, select "Report worst-case paths during compilation".



Compile your SHA256 project.
After compiling your design, there will be a "*TimeQuest Timing Analyzer*" section in the Compilation Report. Click on "*Slow 1100mV 85C Model*".



Note that some of the headings will be in RED indicating a ***failed*** timing. The Fmax reported in the Slow 1100mV 85C Model is less than that specified in the constraints file.

File Edit View Project Assignments Processing Tools Window Help

g00_SIG_CH_MAJ

Project Navigator | Hierarchy

Entity:Instance

Cyclone V: 5CSEMA5F31C6

g00_SHA256

Tasks | Compilation

Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm
TimeQuest Timing Analysis

g00_Hash_Core.vhd | g00_SHA256.vhd | g00_SIG_CH_MAJ.vhd | SHA256.sdc | Compilation Report - g00_SIG_CH_MAJ

Table of Contents

- Analysis & Synthesis
- Fitter
- Assembler
- TimeQuest Timing Analyzer
  - Summary
  - Parallel Compilation
  - SDC File List
  - Clocks
  - Slow 1100mV 85C Model
    - Fmax Summary
    - Timing Closure Recommer
    - Setup Summary
    - Hold Summary
    - Recovery Summary
    - Removal Summary
    - Minimum Pulse Width Sun
    - Metastability Summary
  - Slow 1100mV 0C Model
  - Fast 1100mV 85C Model

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	140.0 MHz	140.0 MHz	CLK	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera

All | <<Filter>> | Find... | Find Next

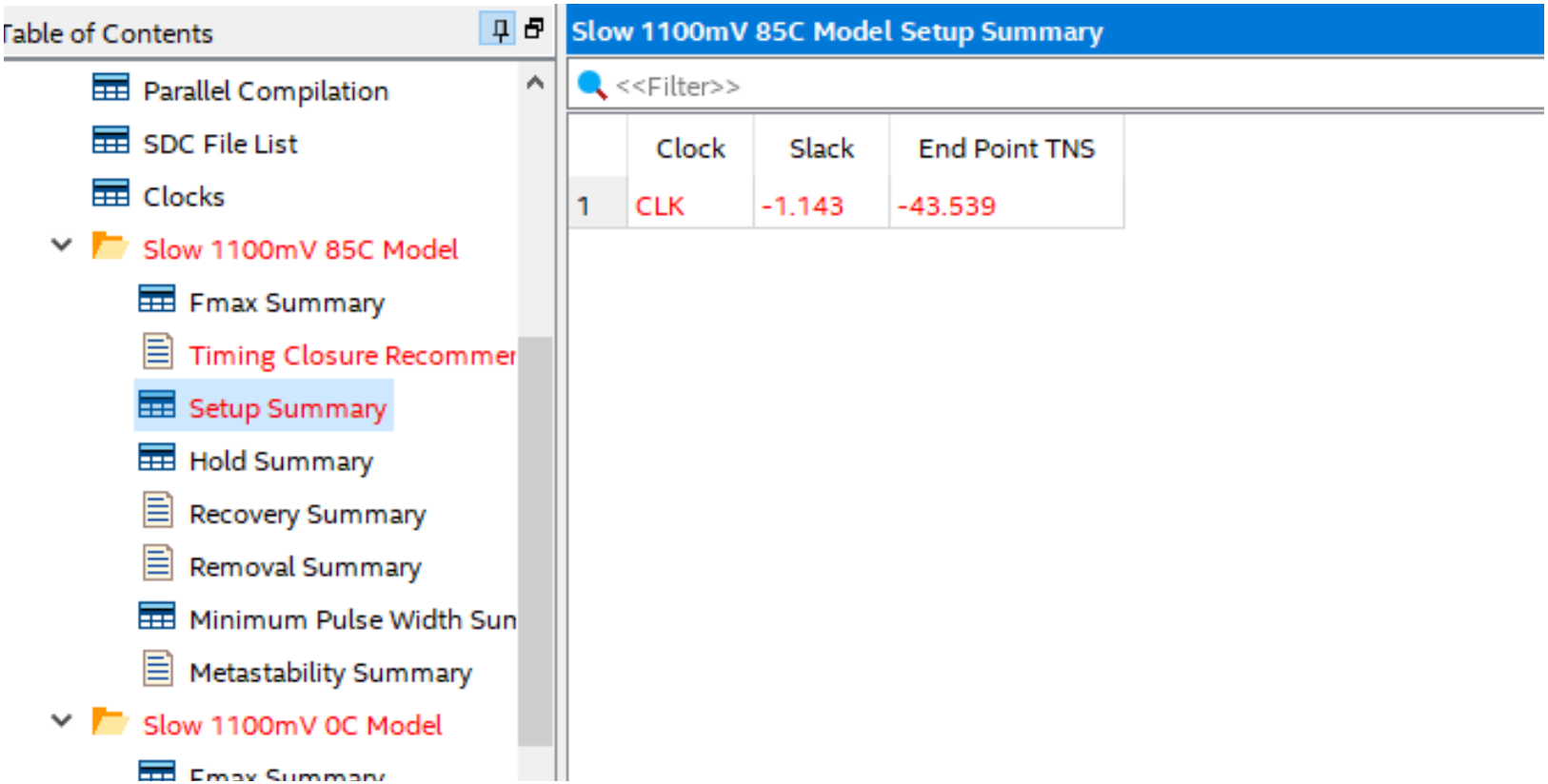
```
Type    ID     Message
        Running Quartus Prime EDA Netlist Writer
        Command: quartus_eda --read_settings_files=off --write_settings_files=off g00_SHA256 -c g00_SIG_CH_MAJ
 18236  Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an
 10905  Generated the EDA functional simulation netlist because it is the only supported netlist type for this device.
204019  Generated file g00_SIG_CH_MAJ.vho in folder "E:/Users/owner/Documents/work/teaching/ECSE325/labs/W2021/lab2/simulation/modelsim/" for EDA simulation tool
        Quartus Prime EDA Netlist Writer was successful. 0 errors, 2 warnings
293000  Quartus Prime Full Compilation was successful. 0 errors, 56 warnings
```

Messages | System (1) | Processing (136)

100% 00:01:09

Click on the “Setup Summary” report. This will list the minimum setup time slack value over all paths in the circuit. The End Point TNS is the *total negative slack* summed over all paths and can give you an idea of how extensive the timing problems are.

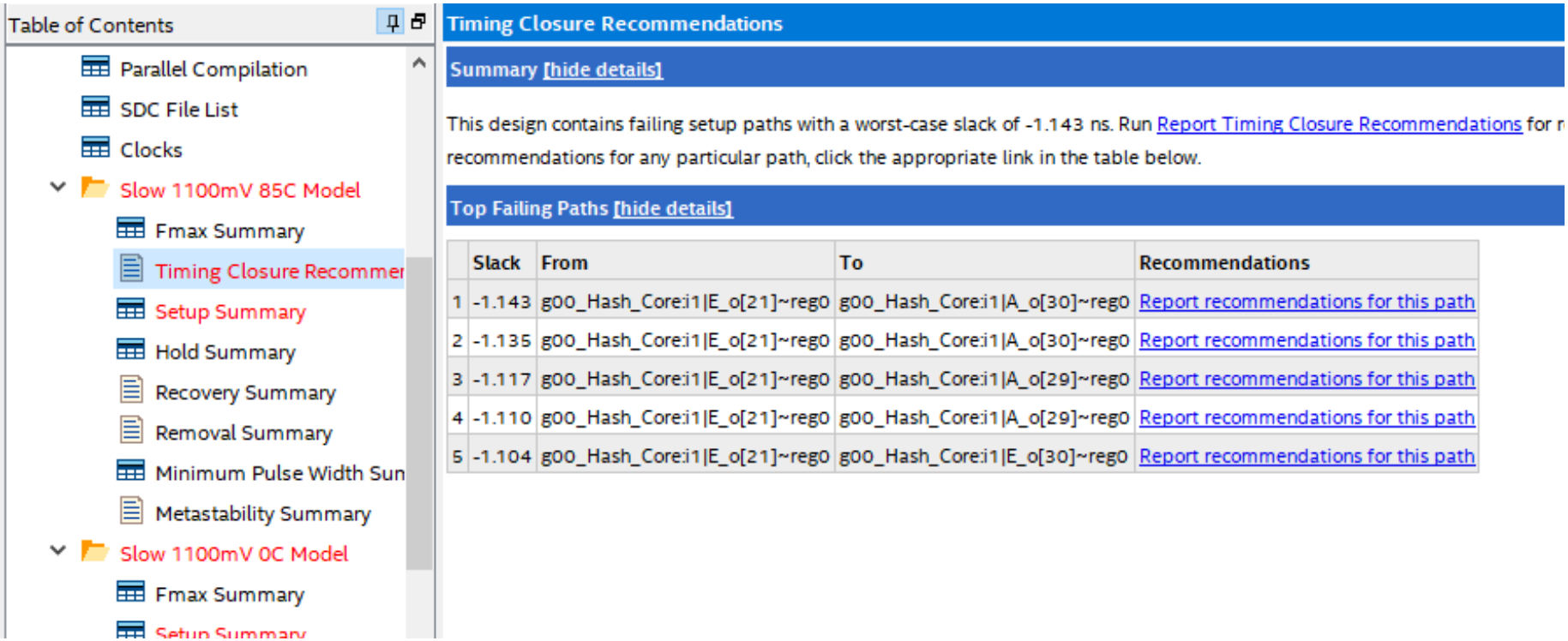
If this value is negative, then your circuit has timing problems somewhere.



Click on the “Timing Closure Recommendations” report. This will list the paths with the most negative failing setup time slack values.

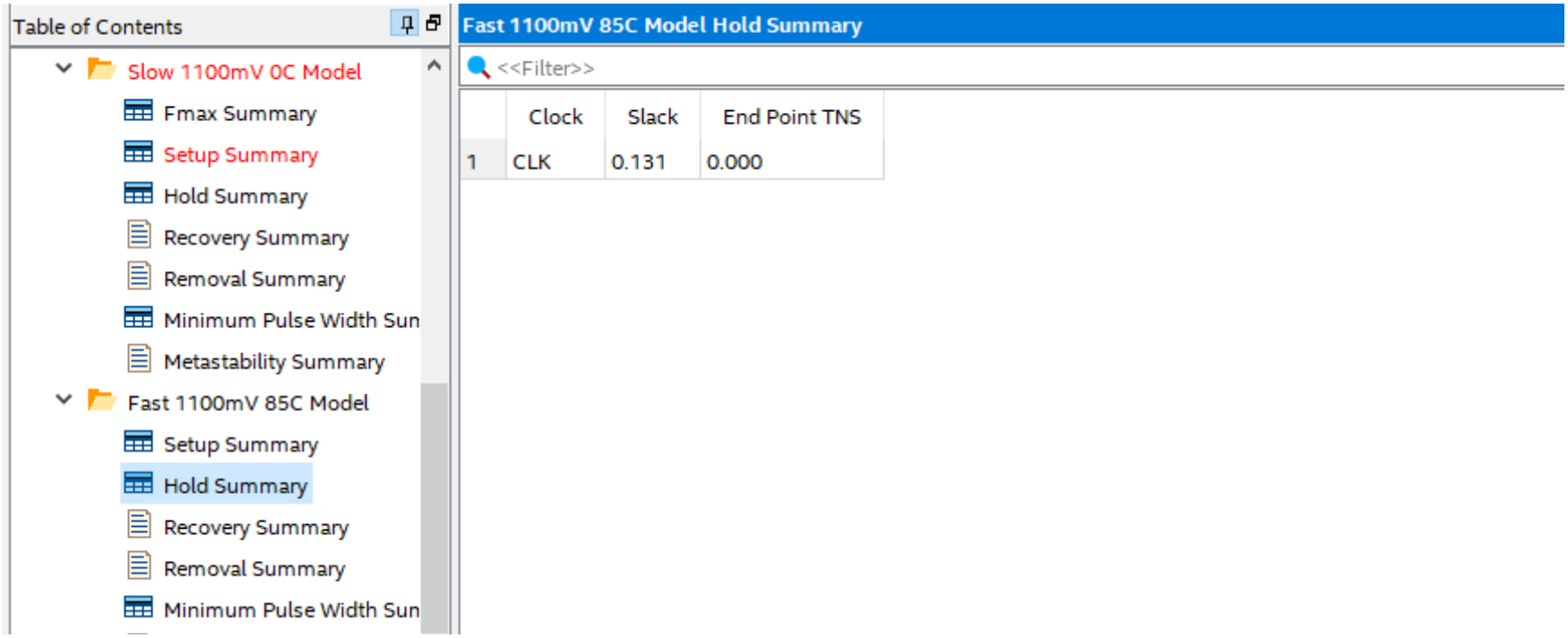
Make a note of which registers form the path. This will tell you where the maximum delays are, usually due to long chains of logic such as adders/multipliers etc.

Try to determine where in your circuit the long delays are.



Click on the “Hold Summary” report for the Fast 100mV 0C. This will list the minimum hold time slack value over all paths in the circuit. The End Point TNS is the *total negative slack* summed over all paths and can give you an idea of how extensive the timing problems are.

If this value is negative, then your circuit has timing problems somewhere. Usually, the Hold time slacks are positive, but you should check them anyway.



Mark down the following values, which you should include in your report:

Requested ***Fmax*** = ___166 MHz_____

Fast 1100mV 0C Model ***Hold*** Slack Value =_____

Slow 1100mV 85C Model ***Setup*** Slack Value = _____

Slow 1100mV 85C Model ***Fmax*** =_____

List the Worst-case Timing paths for the Setup times.

To try and get a passing timing analysis tell the timing analyzer that your clock period is ***8ns*** (corresponding to a clock frequency of 125MHz). Enter the following single line into the file:

```
create_clock -period 8 [get_ports {clk}]
```

The Timing Analyzer will read the .sdc file and use the constraint information when doing its analysis.

Recompile the project and look at the new timing analysis report. Does your circuit now pass the timing analysis?

Mark down the following values, which you should include in your report:

Requested ***Fmax*** = ___125 MHz_____

Fast 1100mV 0C Model ***Hold*** Slack Value =_____

Slow 1100mV 85C Model ***Setup*** Slack Value = _____

Slow 1100mV 85C Model ***Fmax*** =_____

You should also take a look at the “Flow Summary” after the compilation.
In particular, take note of the usage of ALMs (adaptive logic modules). So far, you should only be using a small fraction of the FPGA’s logic elements.

File Edit View Project Assignments Processing Tools Window Help

g00_SIG_CH_MAJ

Project Navigator | Hierarchy

Entity:Instance

Cyclone V: 5CSEMA5F31C6

g00_SHA256

Tasks | Compilation

Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm
TimeQuest Timing Analysis

g00_Hash_Core.vhd | g00_SHA256.vhd | g00_SIG_CH_MAJ.vhd | SHA256.sdc | Compilation Report - g00_SIG_CH_MAJ

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Assembler
- TimeQuest Timing Analyzer
  - Summary
  - Parallel Compilation
  - SDC File List
  - Clocks
  - Slow 1100mV 85C Model
    - Fmax Summary
    - Timing Closure Recommer
    - Setup Summary
    - Hold Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Sun Mar 14 14:09:21 2021
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g00_SIG_CH_MAJ
Top-level Entity Name	g00_SHA256
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	202 / 32,070 (< 1 %)
Total registers	316
Total pins	258 / 457 (56 %)
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 %)

All | <<Filter>> | Find... | Find Next

```
Type   ID     Message
       Running Quartus Prime EDA Netlist Writer
       Command: quartus_eda --read_settings_files=off --write_settings_files=off g00_SHA256 -c g00_SIG_CH_MAJ
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an
 10905 Generated the EDA functional simulation netlist because it is the only supported netlist type for this device.
204019 Generated file g00_SIG_CH_MAJ.vho in folder "E:/Users/owner/Documents/work/teaching/ECSE325/labs/W2021/lab2/simulation/modelsim/" for EDA simulation tool
       Quartus Prime EDA Netlist Writer was successful. 0 errors, 2 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 54 warnings
```

Messages | System (1) | Processing (150)

100% 00:01:08

Also take a look at the Chip Planner, found in the Fitter report. It gives an overview of how the design was mapped to the FPGA logic array.

Quartus Prime Lite Edition - E:/Users/owner/Documents/work/teaching/ECSE325/labs/W2021/lab2/g00_SHA256 - g00_SIG_CH_MAJ

File Edit View Project Assignments Processing Tools Window Help

Project Navigator | Hierarchy

Entity:Instance

Cyclone V: 5CSEMA5F31C6

g00_SHA256

Tasks | Compilation

Task

Fitter (Place & Route)

Edit Settings

View Report

Chip Planner

Technology Map Viewer (Pc

g00_Hash_Core.vhd | g00_SHA256.vhd | g00_SIG_CH_MAJ.vhd | SHA256.sdc | Compilation Report - g00_SIG_CH_MAJ

Table of Contents

Flow Summary

Flow Settings

Flow Non-Default Global Settings

Flow Elapsed Time

Flow OS Summary

Flow Log

Analysis & Synthesis

Fitter

Assembler

TimeQuest Timing Analyzer

EDA Netlist Writer

Flow Messages

Flow Suppressed Messages

Flow Summary

Flow Status	Successful - Sun Mar 14 14:09:21 2021
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g00_SIG_CH_MAJ
Top-level Entity Name	g00_SHA256
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	202 / 32,070 (< 1 %)
Total registers	316
Total pins	258 / 457 (56 %)
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 %)

Messages

```
Type   ID     Message
       Quartus Prime EDA Netlist Writer was successful. 0 errors, 2 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 54 warnings
       ************************************************************
       Running Quartus Prime Netlist Viewers Preprocess
       Command: quartus_npp g00_SHA256 -c g00_SIG_CH_MAJ --netlist_type=atom_fit
18236  Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an
       Quartus Prime Netlist Viewers Preprocess was successful. 0 errors, 1 warning
```

System (1) | Processing (155)

100% 00:00:01

Chip Planner - E:/Users/owner/Documents/work/teaching/ECSE325/labs/W2021/lab2/g00_SHA256 - g00_SIG_CH_MAJ

File Edit View Tools Window Help

Search altera.com

Report

Report not available

Coordinate: Editing Mode: ECO - 5CSEMA5F31C6

ARM A9 Subsystem

Layers Settings

Basic

Background

None

Block Utilization

Layers Settings Color Legend

Node Properties

Selected elements: CLK

Properties/Modes	Values
Full Name	|g00_SHA256|CLK
Coordinate	(32 , 0)

Pad Input Buffer

Properties Fan-in Fan-out

Tasks

Generate Clock Data

Toggle Background C

Report Resources...

Report Compilation M

tcl>

Console History

Timing Located Objects

NOTE: Why don't we also do a timing simulation?

- Recall the following from the lecture on simulation:

Gate-level timing	Simulation using a post-fit timing netlist, testing design's functional and timing correctness. Not supported for Arria V, Cyclone V, or Stratix V devices.	■ Testbench ■ Altera simulation libraries ■ Post-fit timing netlist ■ Post-fit Standard Delay Output File (.sdo)

!!!

☞ Gate-level timing simulation of an entire design can be slow and should be avoided. Gate-level timing simulation is not supported for Arria V, Cyclone V, or Stratix V devices. Rely on TimeQuest static timing analysis rather than on gate-level timing simulation.

- You could do another functional simulation, but you already simulated most of the functionality in lab 2.

Write up part 1 of the Lab Report .

Write up a short report describing the ***gNN_Hash_Core*** circuit that you designed in this lab. This report should be submitted in pdf format.

The report must include the following items:

- A header listing the group number, the names and student numbers of each group member.
- A title, giving the name (e.g. ***gNN_ Hash_Core***) of the circuit.
- A description of the circuit's function, listing the inputs and outputs.
- A screenshot of the Flow Summary and of the Chip Planner layout.
- A summary of the timing analyses done at different clock frequency constraint values. This should give the worst-case setup and hold time slack values, as well as the identity (i.e. which registers the path connects) of the worst (if any) failing paths.
- An estimate of how many hashes (the 65-clock cycle process) you could perform per second at the maximum clock rate. From the Flow Summary, how many copies of the Hash Core could you put onto the FPGA? Assuming that these copies could all run at the same maximum clock rate, what would be the total number of hashes per second you could achieve with this FPGA? Why would it be unlikely that you would be able to reach this number?

Part 2 : Pipelining.

In Part 2 of this lab you will implement a ***pipelined*** version of a sinusoidal function computation.

The circuit takes as input a 13-bit unsigned number (0-8191, representing angles from 0 to pi/2) as the angle and generates a 13-bit unsigned value (0-8191, representing fractional values from 0 to 1) for the sine of that angle.

The angle input values are assumed to be non-negative and cover the range from 0 degrees to 90 degrees. Since both the angle and the sine of the angle are non-negative over this range, we can use unsigned numbers. Values for other angles can be obtained by various symmetry transformations, which we won't do in this lab.

You are to use the fixed-point sine approximation given at: https://www.nullhardware.com/blog/fixed-point-sine-and-cosine-for-embedded-systems/

The c-code for this method is shown on the next slide.

Your job is to translate this to VHDL!

```c
/*
Implements the 5-order polynomial approximation to sin(x).
@param i   angle (with 2^15 units/circle)
@return    16 bit fixed point Sine value (4.12) (ie: +4096 = +1 & -4096 = -1)

The result is accurate to within +- 1 count. ie: +/-2.44e-4.
*/
int16_t fpsin(int16_t i)
{
    /* Convert (signed) input to a value between 0 and 8192. (8192 is pi/2, which is the region of the curve fit). */
    /* ---------------------------------------------------------------------- */
    i <<= 1;
    uint8_t c = i<0; //set carry for output pos/neg

    if(i == (i|0x4000)) // flip input value to corresponding value in range [0..8192)
        i = (1<<15) - i;
    i = (i & 0x7FFF) >> 1;
    /* ---------------------------------------------------------------------- */

    /* The following section implements the formula:
     = y * 2^-n * ( A1 - 2^(q-p)* y * 2^-n * y * 2^-n * [B1 - 2^-r * y * 2^-n * C1 * y]) * 2^(a-q)
    Where the constants are defined as follows:
    */
    enum {A1=3370945099UL, B1=2746362156UL, C1=292421UL};
    enum {n=13, p=32, q=31, r=3, a=12};

    uint32_t y = (C1*((uint32_t)i))>>n;
    y = B1 - (((uint32_t)i*y)>>r);
    y = (uint32_t)i * (y>>n);
    y = (uint32_t)i * (y>>n);
    y = A1 - (y>>(p-q));
    y = (uint32_t)i * (y>>n);
    y = (y+(1UL<<(q-a-1)))>>(q-a); // Rounding

    return c ? -y : y;
}
```

Since we are assuming the input angles to be non-negative, we can omit the first part of the algorithm.

The c-code expression **(uint32_t)i** is merely placing the 16-bit input into a 32-bit word, where the 16 MSBs are all set to zero.

The expression ***nn*UL** means that the variable ***nn*** is an *unsigned long*, which in the C-programming language means a 32-bit integer.

All of the multiplications in this algorithm compute the product of two 32-bit integer (unsigned) operands.

The notation **y >> n** means shift right by n places. This is equivalent to dividing by 2^n.

These shifts/divisions are there to scale the output of the multiplications and additions back into a 32-bit range. You can implement these merely by throwing away the 16 LSBs of the computation or by doing a shift right by 16 operation and throwing away the 16 MSBs.

You are to implement two versions (in VHDL) of your sine function.

In the first version, store the input and the output in registers (i.e. by using an assignment statement in a clocked process block) but all other computations should be done in a combinational block (i.e. no clock) that sits between the input and output registers.

In the second version, fully pipeline the algorithm by inserting pipeline registers after every multiplication operation. Make sure to ***balance all pathways*** from the input register to the output register so that they all pass through the same number of registers.

Draw a flow diagram showing all of your operations to make it easier to visualize that the paths are balanced.

For each version, do the following:

- Functional simulation using at least 4 different input values as a check to see whether the computation of the sine value is correct. Do this first to make sure your generated sine values are correct, then move to the timing analysis.

- Static timing analysis to see what is the maximum clock frequency.

- Take note of the FPGA resources used by the design (e.g. ALM logic elements and DSP blocks)

Write up part 2 of the Lab Report

Write up a short report describing the ***gNN_Sine*** circuit that you designed in this lab. This report should be submitted in pdf format.

The report must include the following items:

- A header listing the group number, the names and student numbers of each group member.
- A title, giving the name (e.g. ***gNN_ Sine***) of the circuit.
- A description of the circuit's function, listing the inputs and outputs.
- For each version of your design provide the following:
    - A screenshot of the Flow Summary showing the usage of ALM and DSP blocks.
    - Static timing analysis report, showing maximum clock frequency.
    - Functional simulation results for 4 different input values.
- Discussion of the relative merits of your two designs.

Submit the Lab Report to myCourses .

The lab report, and all associated design files (.vhd and .sdc files) must be submitted, as an assignment to the myCourses site.

Only one submission need be made per group (all group members will receive the same grade).

Combine all of the files that you are submitting into one *zip* file and name the zip file gNN_LAB_3.zip (where NN is your group number).

The report is due on Tuesday April 2 (later due to the Easter holiday), at 11:59 PM.