Simulation, debugging and implementing the design with ModelSim and Quartus Prime Lite

Contents

Introduction

Traditional debugging techniques often involve using an external pattern generator to exercise the logic and a logic analyzer to study the output waveforms during runtime. The In-System Sources and Probes Editor (ISSP) in the Quartus software extends the portfolio of verification tools. It allows you to monitor various signals in the design.

ISSP Editor consists of a probe function and interface to control the instances during run time. It operates over JTAG. Each instance of ISSP can drive and toggle values up to 512 signals. It can create up to 128 instances of ISSP using IP Catalog.

The main difference between ISSP and Signal Tap (discussed in the next section) is that ISSP does not have a clock as reference to the output signals we will probe, it will only sample signals in the current time.

Prerequisites

You should:

- be familiar with basics of logic and digital design,
- be familiar with Verilog constructs,
- have Quartus Prime Lite installed.

Objectives

You will familiarize yourself with the following:

1. How to create design in Quartus Prime
2. How to simulate design using ModelSim
3. How to debug design on a board using ISSP and SignalTapII.
4. How to implement design on a board.

Before going forward

Before going forward with the Lab be sure that you have all files in the working folder.

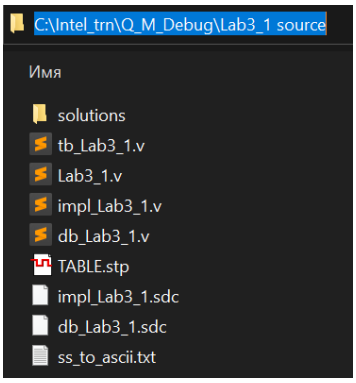


Figure 1

Creating the design in Quartus Prime

Design overview

This lab utilizes a simple digital logic design. There is only one source file with a description of the design.

The source code is in the file **C:\Intel_trn\Q_M_Debug\Lab3_1 \Lab3_1.v**.

```
`timescale 1ns/1ns
module Lab3_1
#( parameter div_par = 25'd4) (
    input CLK,
    input RST,
    input DIR,
    output [3:0] DIG,
    output reg [6:0] HEX
);
    reg     [24:0]  div_cnt     = 25'd1;  //Clock divider
    wire            cout;                 //Carry out
    reg     [3:0]   cnt_val;              //value to count
    reg     [1:0]   rst_int     = 2'd0;   //Synchronized reset
    reg     [3:0]   Counter     = 4'd0;   //Counter
//==========================================================
// Reset
//
always @(posedge CLK)
    rst_int <= {rst_int[0], RST};
//==========================================================
// Clock Divider
//
always @(posedge CLK, negedge rst_int[1])
    if(!rst_int[1] )
        div_cnt <= 25'd1;
    else



            div_cnt <= div_cnt + 25'd1;

assign cout = (div_cnt == div_par);
// Counter
//
always @(posedge CLK)
    cnt_val = (DIR)? (-4'd1):(4'd1); //value for counting

always @(posedge CLK, negedge rst_int[1])
    if(!rst_int[1])
        Counter <= 4'd0;
    else
        if (cout)
        begin
            Counter <= Counter + cnt_val;
            case ({DIR, Counter})
                5'b10000: Counter <= 4'd9;
                5'b01001: Counter <= 4'd0;
            endcase
        end
//==========================================================
// Coder
//
always @(posedge CLK, negedge rst_int[1])
if(!rst_int[1])  HEX <= 7'b0111111;
else
    case(Counter)
        4'b0000: HEX <= 7'b0111111; // "0"
        4'b0001: HEX <= 7'b0000110; // "1"
        4'b0010: HEX <= 7'b1011011; // "2"
        4'b0011: HEX <= 7'b1001111; // "3"
        4'b0100: HEX <= 7'b1100110; // "4"
        4'b0101: HEX <= 7'b1101101; // "5"
        4'b0110: HEX <= 7'b1111101; // "6"
        4'b0111: HEX <= 7'b0000111; // "7"
        4'b1000: HEX <= 7'b1111111; // "8"
        4'b1001: HEX <= 7'b1101111; // "9"
        default: HEX <= 7'b0111111; // "0"
    endcase
//==========================================================
// Constant value
//
assign DIG  = 4'b1000;
//==========================================================
endmodule
```

Figure 2

Inputs for the top-level module Lab3_1 are:

- CLK – clock signal. It is FPGA pin connected to a Clock Generator on the board. Frequency is 25MHz.
- RST – asynchronous reset signal. It is FPGA pin connected to Push Button on the board.
- DIR – direction for counting (1 – counting DOWN ; 0 – Counting UP). It is FPGA pin connected to a switch on the board.

Outputs for the top-level module Lab3_1 are:

- [6:0] HEX – 7-segment code. These are FPGA pins connected to 7-segment display on a development board.
- [3:0] DIG – constant value. These are FPGA pins connected to inputs switching on\off digits of 7-segment display.

Algorithm of the project is:

- Clock Divider (**div_cnt)** divides input clock and provides carry out (**cout**) signal.
    - The value of division (**div_par**) is a parameter.
- Counter (**cnt_val**) is

- binary-decimal counter
- synchronized by clock (**CLK**),
- enabled by carry out (**cout**) signal, coming from Clock Divider.
- the value for counting (**cnt_val**) is based on input DIR

- Coder
  - Codes 4-bits binary-decimal code, coming from Counter, into 7-segment code.
  - Sets constant value "1000" for DIG outputs (the leftmost digit of 7-segment indicator is on, the rest are off).

Pay attention that the code provided has some issues, which you will fix during simulation procedure.

Creating the project in Quartus Prime

1 Start Quartus Prime **Lite** development tool

2 Create a project

a. Working directory: **C:\Intel_trn\Q_M_Debug\Lab3_1**
b. Project name: Lab3_1
c. Top-Level design entity: Lab3_1
d. Project Type: Empty project
e. Add files from the folder C:\Intel_trn\Q_M_Debug\Lab3_1: **Lab3_1.v**
f. Device: EP4CE6E22C8
g. EDA tools: NONE

3 In Quartus Prime window, select **Assignment => Settings...**

4 In the window appeared:

a. Select **Compilation Process Settings**
   i Set **Use all available processors**
   ii Set **Use Smart Compilation**
   iii Click **OK**

5 To check if the project is correct:

a. Select **Project navigator** window => **Hierarchy** in the drop down menu.
b. Select **Lab3_1** top level unit
c. Select **Processing** menu => **Start => Start Analysis and Synthesis** 
d. Be sure that you have neither Errors nor Critical Warnings.
   i You can have one warning dealing with constant values on DIG outputs

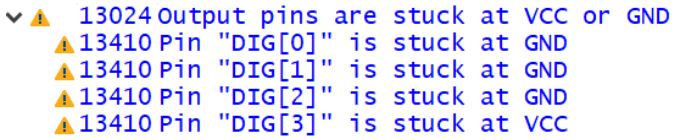


Figure 3

Simulating the design in ModelSim

Top-level file for simulation

The source code for the testbench is in the file **C:\Intel_trn\Q_M_Debug\Lab3_1 \tb_Lab3_1.v**.

```
`timescale 1ns/1ns
module tb_Lab3_1 ();
    reg             tb_clk;
    reg     [5:0]   tb_mem [0:127];
    reg             tb_dir;
    wire    [3:0]   tb_dig;
    wire    [6:0]   tb_hex;
    reg             tb_reset;
    wire    [6:0]   tb_ss;
initial
begin :  clock_gen
    tb_clk = 1'b0;
    while (1) #10 tb_clk = ~tb_clk;
end

Lab3_1 #(25'd4) Lab3_1_inst (
    .CLK    (tb_clk     ),
    .RST    (tb_reset   ),
    .DIR    (tb_dir     ),
    .DIG    (tb_dig     ),
    .HEX    (tb_hex     )
);

initial
begin: reset_gen
    tb_reset = 1'b0;
    #100 tb_reset = 1'b1;
end

initial
begin : control_gen
    tb_dir = 1'b0;
    #1100 tb_dir = 1'b1;
    #880;
end

initial
begin : mem_reading
    $readmemb ("ss_to_ascii.txt", tb_mem);
    #2220 $stop;
end

assign tb_ss = tb_mem [tb_hex];

endmodule
```

Figure 4

This is a simple testbench in which there are:

- Clock_gen – a procedural block for clock signal (**tb_clk**) generation.
- Reset_gen – a procedural block for reset signal (**tb_reset**) generation.
- Control_gen – a procedural block for counting direction signal (**tb_dir**) generation.
- Mem_reading – a procedural block for reading data from file **ss_to_ascii.txt** into array **tb_mem**.
- Lab3_1_inst – an instantiation of entity Lab3_1.

7-segment code, coming from HEX bus outputs of Lab3_1_inst, are decoded into ASCII symbols by using Look-up table implemented as the array **tb_mem**.

File **ss_to_ascii.txt** contains address (in hexadecimal code), which are 7-segment codes, and data values (in binary code), which are ASCII codes for digital values coded in 7-segment code.

ss_to_ascii.txt

```
@3F 110000
@06 110001
@5B 110010
@4F 110011
@66 110100
@6D 110101
@7D 110110
@07 110111
@7F 111000
@6F 111001
```

Figure 5

Launching ModelSim

1 Start ModelSim Intel FPGA Starter Edition.

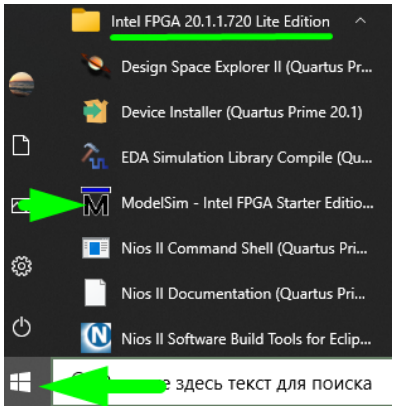


Figure 6

2 Change Directory.
   a. Select **File => Change Directory** and change to the directory **C:\Intel_trn\Q_M_Debug\Lab3_1**.
   b. *OR type in the Transcript window*: **cd C:/Intel_trn/Q_M_Debug/Lab3_1**

3 Create the working library.
   a. Select **File => New => Library =>work**.
   b. *OR type in the Transcript window*: **vlib work**

4 Compile source files:
   a. Select **Compile => Compile =>** "Lab3_1.v" "tb_Lab3_1.v".
   b. *OR type in the Transcript window*: **vlog -work work "Lab3_1.v" "tb_Lab3_1.v"**
   c. Be sure that there are no any Errors. If there are ones you will need to correct source files.

5 Now you're ready to load the design into the simulator:
   a. In the Library window, click the '+' sign next to the work library to show the files contained there and double-click **tb_Lab3_1** to load the design.
   b. *OR type in the Transcript window*: **vsim work.tb_Lab3_1**
   c. Be sure that there are no errors reported in the Transcript window.

Simulation and debugging

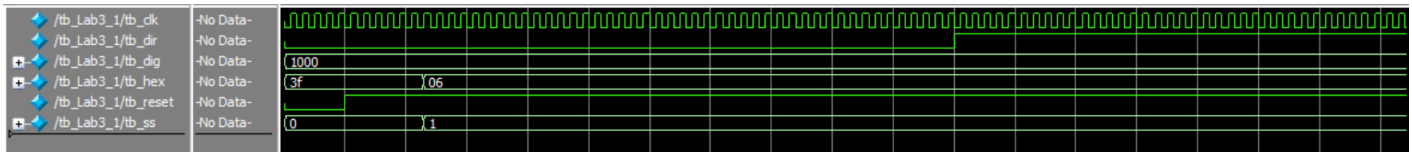
1 In the Objects window
   a. Right-click in the window and select **Add to => Wave => Signals in region**

2 In the Wave window appeared
   a. Right-click **tb_ss** signal and select **Radix => ASCII**
   b. Right-click **tb_hex** signal and select **Radix => Hexadecimal**

3 Start simulation by clicking  icon.
   a. *Or by typing in the Transcript window*: **run -all**

4 Zoom full Wave window.

5 Pay attention on Wave window – you will see that the project does not work properly.



a. You can see a value 0 (ASCII), then a value 1 (ASCII) and then it seems that project stalls. It keeps value 1 (ASCII) by the end of simulation.

6 To find a reason of such behavior add to the Wave window some signals from Lab3_1 unit:

a. In the **Sim** window select **Lab3_1_inst**

b. In the **Object** window select: **div_cnt, cout, Counter**

c. Right-click and select **Add to** => **Wave** => **Selected Signals**

d. In the Wave window

i select **div_cnt** and **Counter**,

ii right-click and select **Radix** => **Unsigned**

7 Restart simulation:

a. Click  icon

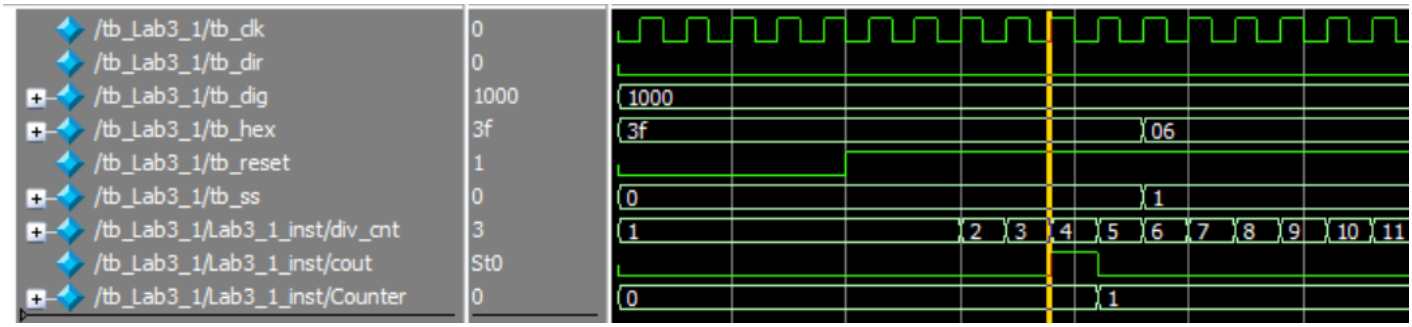
i *OR type in the transcript window:* **restart**.

b. Click OK in the window appeared (if it is appeared.).

8 Start simulation by clicking  icon.

a. *Or by typing in the Transcript window*: **run -all**

9 Zoom in Wave window.



10 Pay attention on Wave window:

a. You can see that at the beginning the **cout** signal is correct (**div_cnt** has value 4, in accordance with the parameter value in tb_Lab3_1.v file).

b. Then the **div_cnt** continues to count instead of re-starting to count from 1.

It seems that Clock Divider (**div_cnt**) in Lab3_1.v does not work properly.

You need to check and correct **div_cnt** description in Lab3_1.v file (lines 25-33). You can use any text editor for it.

The hint:

```
21  // Clock Divider
22  //
23  always @(posedge CLK, negedge rst_int[1])
24      if(!rst_int[1] )
25          div_cnt <= 25'd1;
26      else
27          if (cout)
28              div_cnt <= 25'd1;
29          else
30              div_cnt <= div_cnt + 25'd1;
31
32  assign cout = (div_cnt == div_par);
33  //==========================================================
```

Figure 7

11 In **ModelSim**: recompile Lab3_1.v

a. Select **Library** window.
b. Click '+' sign next to the **work** library to show the files contained there.
c. Right-click **Lab3_1** and select **Recompile**.
d. Be sure that there are no any Errors. If there are ones you will need to correct the source file.

12 Restart simulation:

a. Click  icon
   i *OR type in the transcript window*: **restart**.
b. Click OK in the window appeared.

13 Start simulation by clicking  icon.
   a. *Or by typing in the Transcript window*: **run -all**

14 Zoom Full Wave window.

15 Pay attention on Wave window
   a. The project works properly for now
      i On **tb_ss** bus you can see ASCII values form 0 up to 9
      ii The Counter counts Up and Down in accordance with DIR values.
      iii The Counter counts as Binary-Decimal counter.

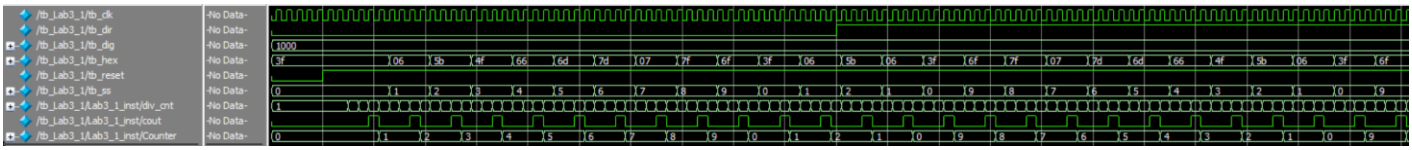


Figure 8

16 In **Memory List** window:
   a. Double-click tb_mem
   b. In the window appeared:
      i Right-click in the address area and select **Properties**
      ii In the window appeared select **Words per Line = 8** and click **OK**.
      iii In **Memory Data** window you can see **tb_mem** memory array initialized by the values from **ss_ascii.txt** file (Address is in hexadecimal format. Data is in binary format).

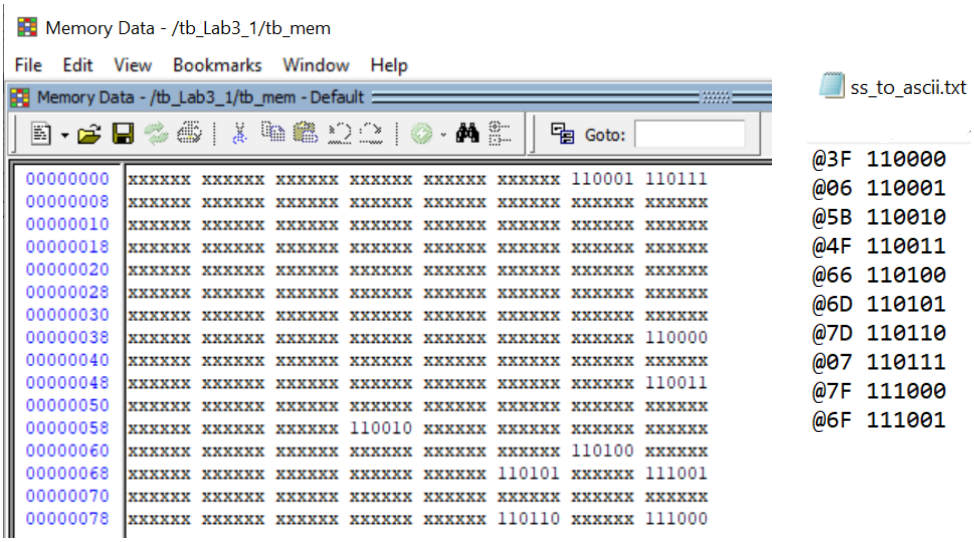


Figure 9

17 Quit **ModelSim**: **File=>Quit**.

Debugging the design

Add IP components for debugging

Some IP components should be added for debugging purposes. These are:

- **Intel FPGA In-System Source & Probes** – the unit will be used for setting (sourcing) and watching (probing) internal project signals (instead of using real FPGA pins).
- **ALTPLL** – the unit will be used for multiplying input clock. Multiplied clock will be used in Signal Tap II unit as a clock source. Multiplied clock allows us to display source clock in the Signal Tap II as a waveform.

1 Open Quartus Prime Lite (if it was closed) and load project **Lab3_1**.

2 Create SP_unit using IP module **Intel FPGA In-System Source & Probes**

a. Find **Intel FPGA In-System Source & Probes** IP function by typing **In-System** in the Find field of IP catalog. If you do not have the IP Catalog already open, go to View menu => Utility Windows => IP Catalog to view the window.

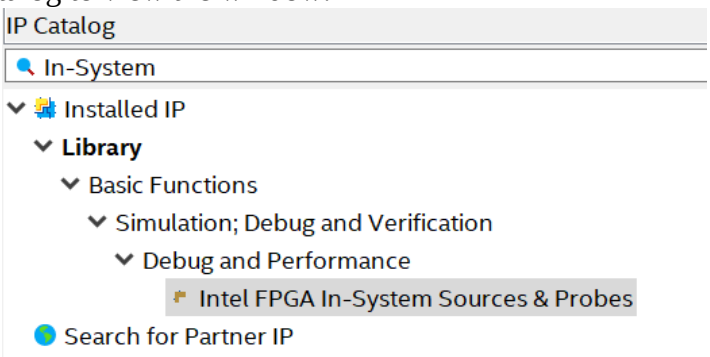


Figure 10

b. Double click **Intel FPGA In-System Source & Probes.**

c. Name the file as **SP_unit** (*Please make sure to name it as **SP_unit)*** in the **New IP Variation** window that came up.

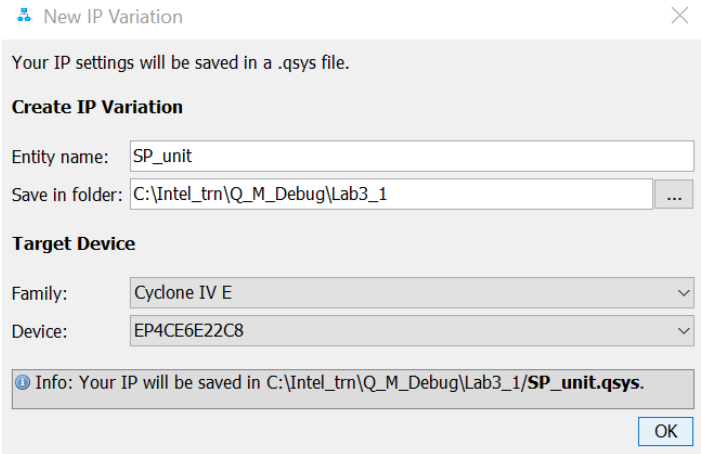


Figure 11

d. Click **OK**.

e. In the window appeared

i Set Probe Port Width = **11** (7 bits for [6:0] HEX outputs, 4 bits for [3:0] DIG outputs).

ii Set Source Port Width = **2** (1 bit for RST input, 1 bit for DIR input).

iii Check in the check box **Use Source Clock** (Source ports will be synchronous).

iv Set Instance ID as **SP_**

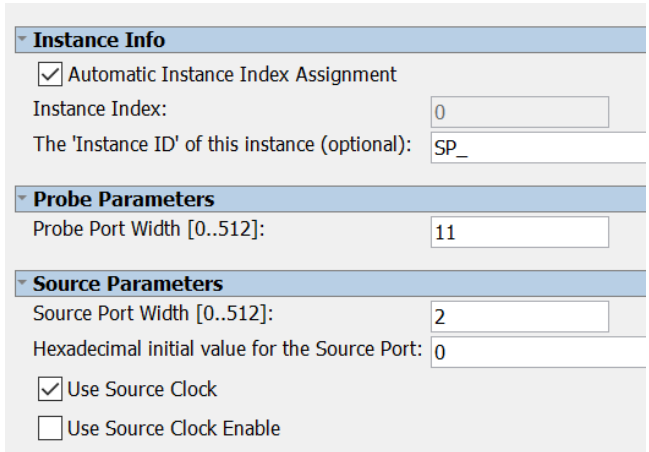


Figure 12

v Click **Finish.**

vi In the window appeared click **Close.**

vii In the window appeared click **Yes.**

viii In the window appeared check that you set options in lined with Figure 13

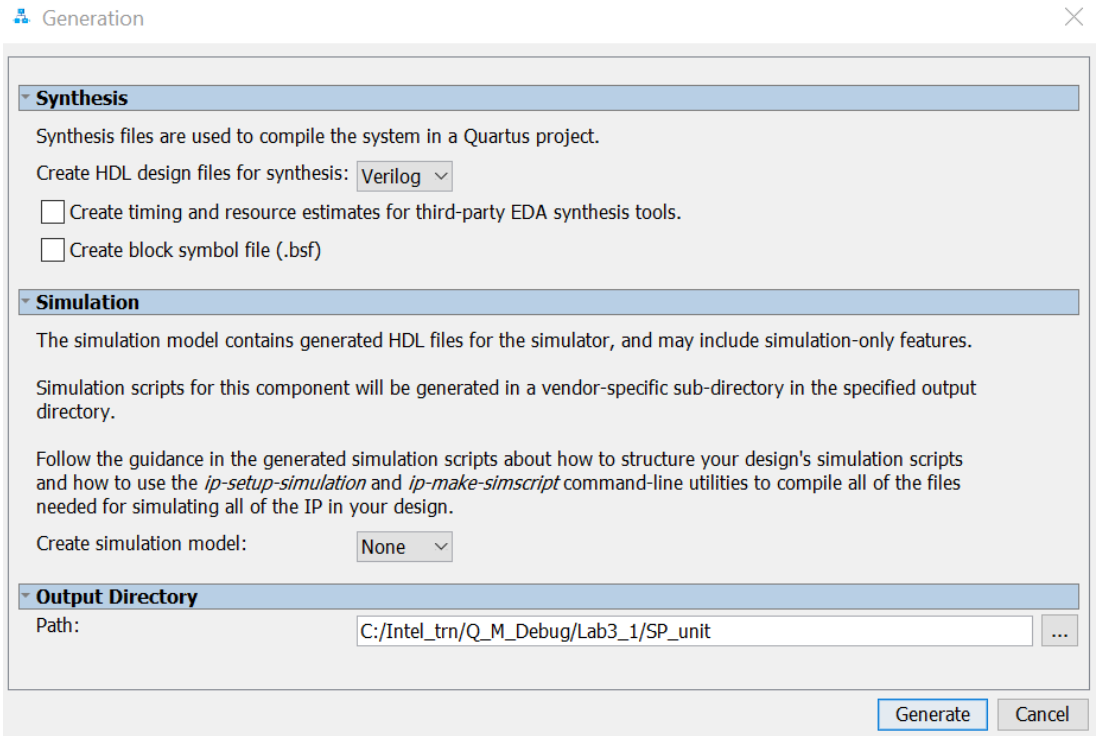


Figure 13

ix Click **Generate.**

x In the window appeared click **Finish.**

f. In the window appeared (if it is appeared) there is a reminder that you need to add SP_unit.qip to the project manually. Click OK.

g. In Quartus Prime select **Project => Add\Remove Files in Project…**

h. In the window appeared **Find** ... the **SP_unit.qip** file in the folder **C:\Intel_trn\Q_M_Debug\Lab3_1\SP_unit\synthesis .**

i. Click **Open.**

j. Be sure that the file is added to the project.

File Name	Type
SP_unit/synthesis/SP_unit.qip	IP Variation File (.qip)
Lab3_1.v	Verilog HDL File

Figure 14

k. Click **OK**.

3 Create PLL_unit by using IP module **ALTPLL**

a. Find **ALTPLL** IP function by typing **ALTPLL** in the **Find** field of IP catalog. If you do not have the IP Catalog already open, go to View menu => Utility Windows => IP Catalog to view the window.

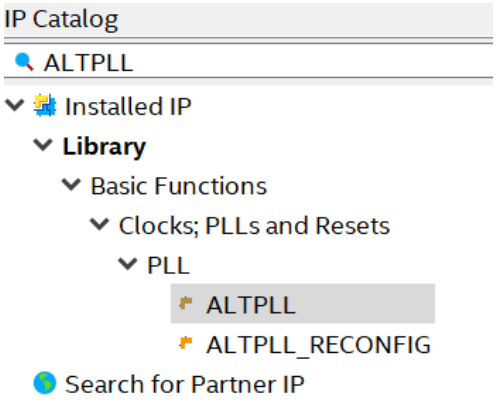


Figure 15

b. Double-click **ALTPLL.**

c. Name the file as (*Please make sure to name it as **PLL_unit***) in the **Save IP Variation** window that came up.

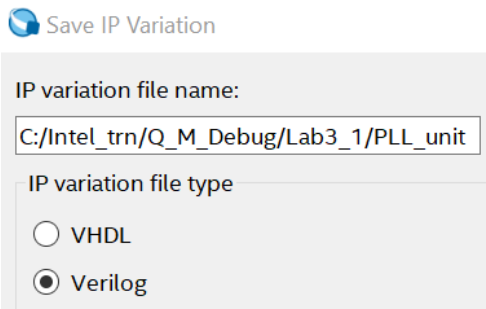


Figure 16

d. Click **OK.**

e. In the window appeared set frequency for **inclk0** input as 25 MHz

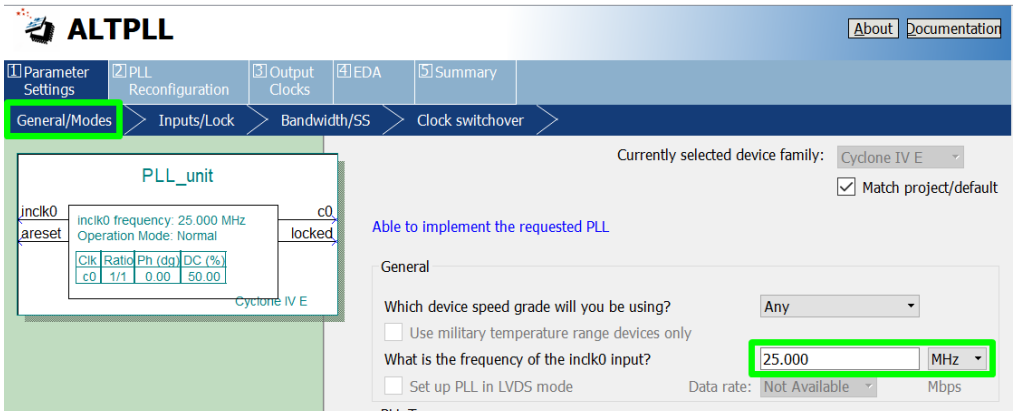


Figure 17

f. Go to folder Inputs/Lock and deselect all check boxes.

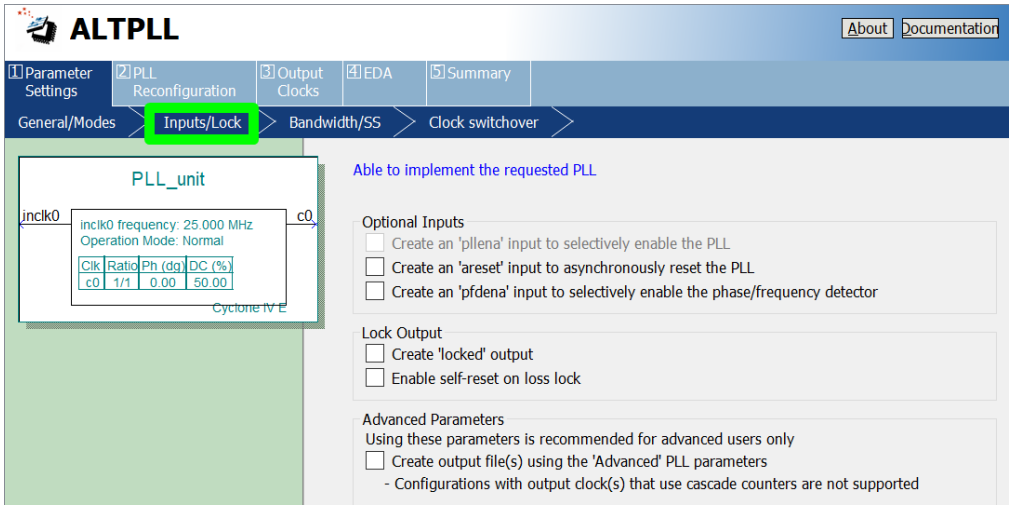


Figure 18

g. Go to folder **Output Clocks => clk c0** and set **Clock multiplication factor** as **2** (to have output frequency 50MHz).

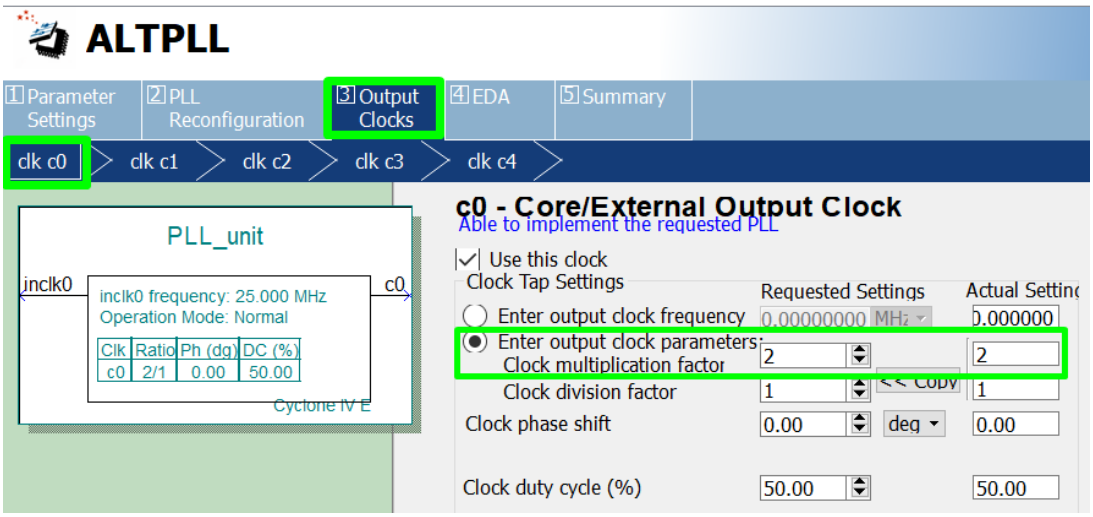


Figure 19

h. Go to Summary folder and select PLL_unit_inst.v file (it could be used as a prompt for instantiating the unit in a project file).

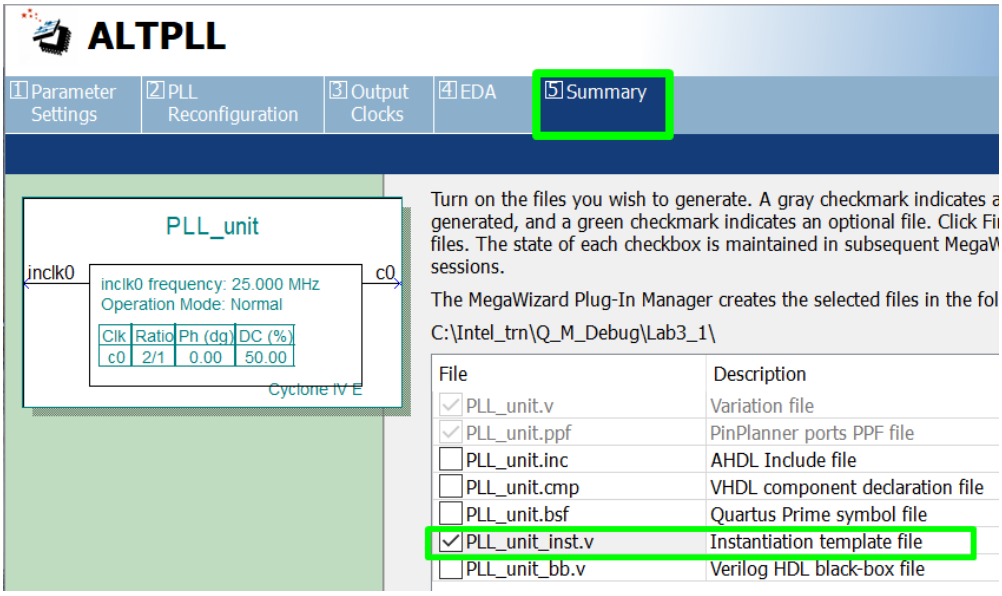


Figure 20

i. Click **Finish.**

j. In the window appeared click **Yes** – PLL_unit.qip file will be added to the project automatically.

4 To check if the IPs are added to the design: select **Project navigator** => **IP Components** in the drop down menu.

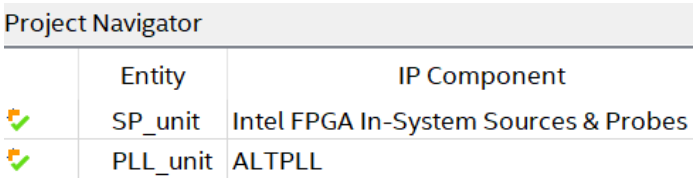


Figure 21

Top-level file for debugging

It is necessary to create top-level file for debugging. The file **db_Lab3_1.v** is provided for the lab in the folder **C:\Intel_trn\Q_M_Debug\Lab3_1**.

```
1   module db_Lab3_1 (
2       (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23"  *)
3           input CLK
4   );
5       wire [6:0] HEX;
6       wire [3:0] DIG;
7       wire    db_reset;
8       wire    db_dir;
9       wire    db_clk_high;
10
11  Lab3_1              Lab3_1_inst
12  (
13      .CLK        (CLK        ),
14      .RST        (db_reset   ),
15      .DIR        (db_dir     ),
16      .DIG       (DIG         ),
17      .HEX        (HEX        )
18  );
19  SP_unit SP_unit_inst (
20      .source     ({db_reset, db_dir} ),
21      .probe      ({HEX, DIG}         ),
22      .source_clk (CLK                )
23  );
24  PLL_unit PLL_unit_inst(
25      .inclk0     (CLK        ),
26      .c0         (db_clk_high)
27  );
28  endmodule
```

Figure 22

The top-level file includes:

- Input CLK – it is FPGA input pin, connected with clock generator on the board.
    - chip_pin attribute sets the number of the pin.
    - altera_attribute sets I/O standardfor the pin.
- Lab3_1_inst – instantiation of Lab3_1 entity.
- SP_unit_inst – instantiation of SP_unit entity.
- PLL_unit_inst – instantiation of PLL_unit entity.
    - Output clock (**c0**) of the **PLL_unit_inst** is connected with signal **db_clk_high**, which is not connected to any units in the top level description. The clock **db_clk_high** will be used as a clock source for Signal TapII logic analyzer, which will be added to the design later.

Pay attention that the source code has some issues; you will fix the issues during debugging procedure.

Analysis and Elaboration

1 Add the file **db_Lab3_1.v** to the project:

a. In Quartus Prime: select **Project => Add\Remove Files in Project…**

b. In the window appeared:

i click ... find icon,
ii select the **db_Lab3_1.v** file (it is in the folder **C:\Intel_trn\Q_M_Debug\Lab3_1),**
iii Click **Open**
iv Be sure that the you add the file to the project.

File Name	Type
db_Lab3_1.v	Verilog HDL File
SP_unit/synthesis/SP_unit.qip	IP Variation File (.qip)
Lab3_1.v	Verilog HDL File
PLL_unit.qip	IP Variation File (.qip)

Figure 23

v Click OK.

2 Set the file **db_Lab3_1.v** as a top level file for the project
a. In the **Project Navigator** are select **Files.**
b. Right-click **db_Lab3_1.v** file and select **Set as a top level Entity.**
c. In the **Project Navigator** are select **Hierarchy** and check that **db_Lab3_1** is a top level entity for now.

3 Select: **Processing => Start => Start Analysis & Elaboration**

4 Be sure that you have neither Errors nor Critical Warnings. If you got ones you need to check and correct IPs created.

5 Select: **Tools => Netlist Viewer => RTL Viewer**

6 Be sure that you have something like Figure 24.

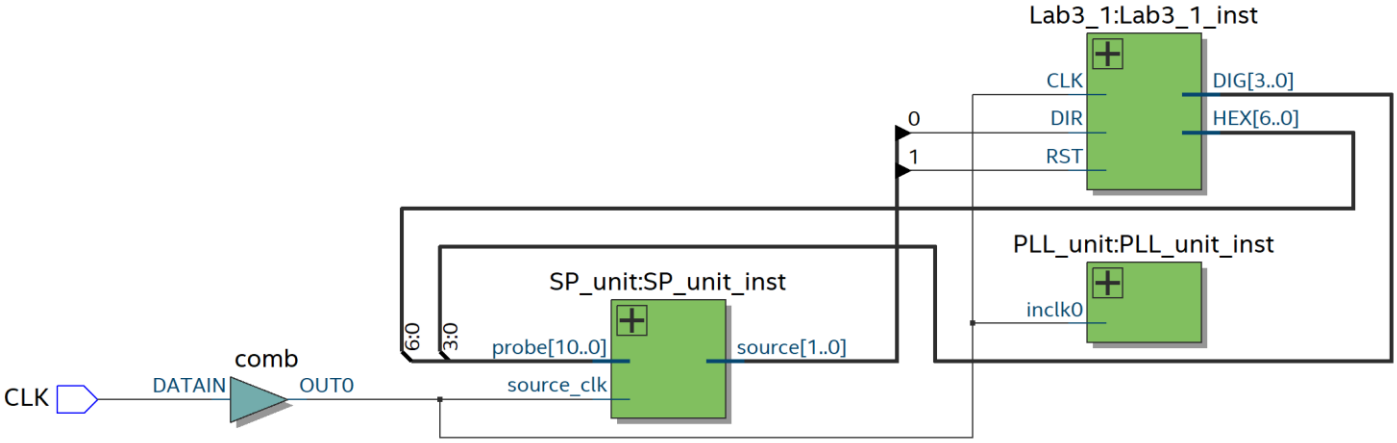


Figure 24

7 Close **RTL_Viewer**.

Adding and setting up Signal TapII

We are going to use Signal Tap II logic analyzer for debugging. It is necessary to set up the logic analyzer.

1 Launch Signal Tap II: select **Tools => Signal Tap Logic Analyzer**
2 In the window appeared select **Setup tab**.

3 **Setup tab**: double click in the empty area to add some nodes for analysis.

4 In the window appeared:

a. In Filter field: select **Signal Tap: pre-synthesis**

b. Click List

c. In the **Matching Nodes** field select: CLK, DIR, RST, DIG, HEX, Counter, div_cnt

d. Click > sign

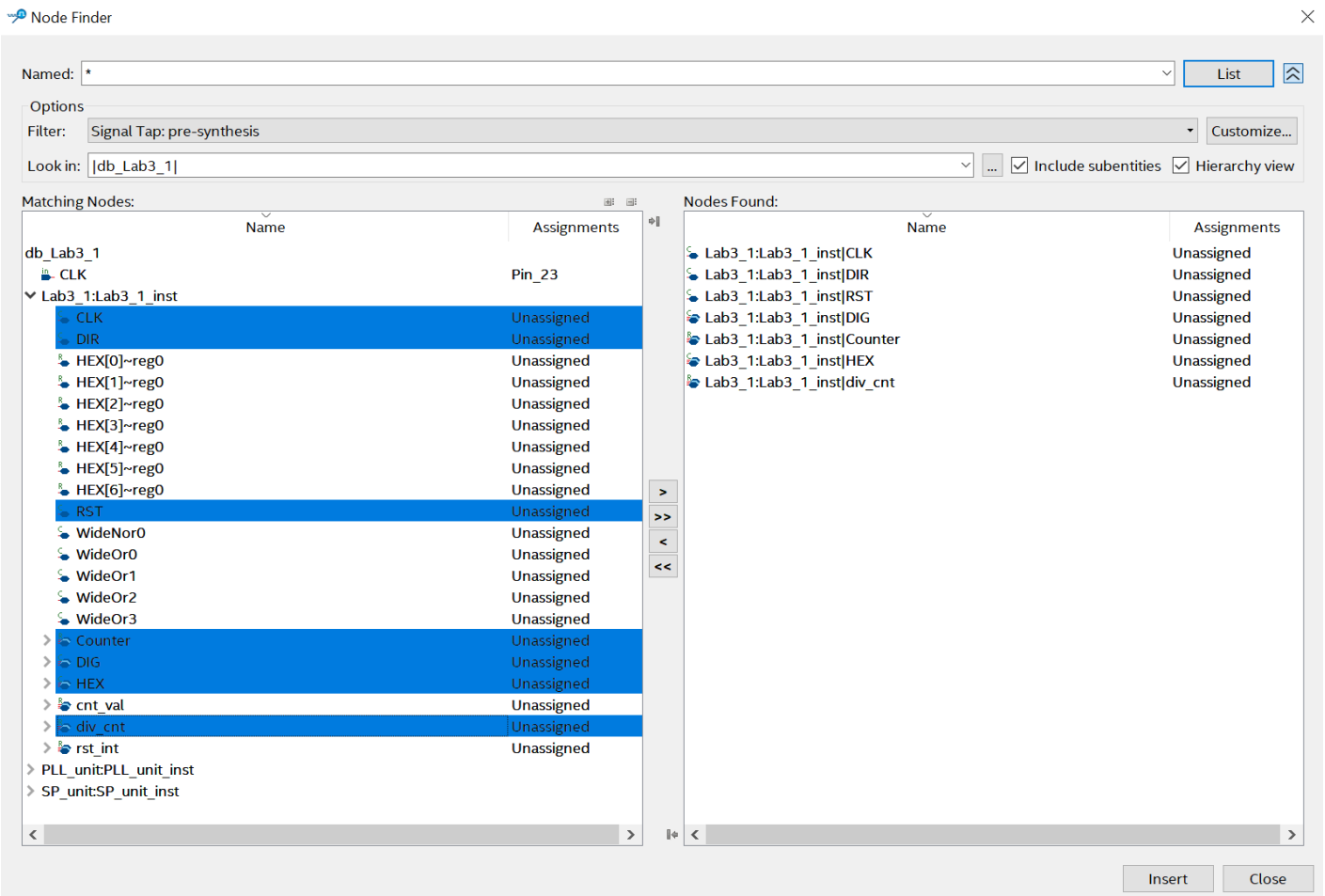


Figure 25

e. Click Insert.

f. Click Close

5 **Setup tab**: right click in the empty area and select **Mnemonic Table Setup**

6 In the window appeared:

a. Select **Import Table**

b. Select file TABLE.stp in the folder C:\Intel_trn\Q_M_Debug\Lab3_1

c. Click Open

d. In the window appeared

i Select ss_udec: width = 7

ii Click > sign

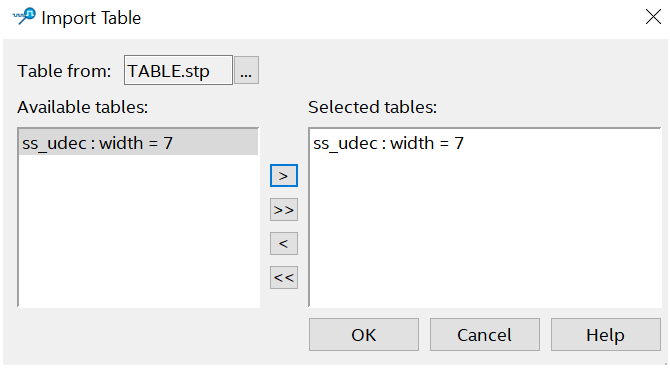
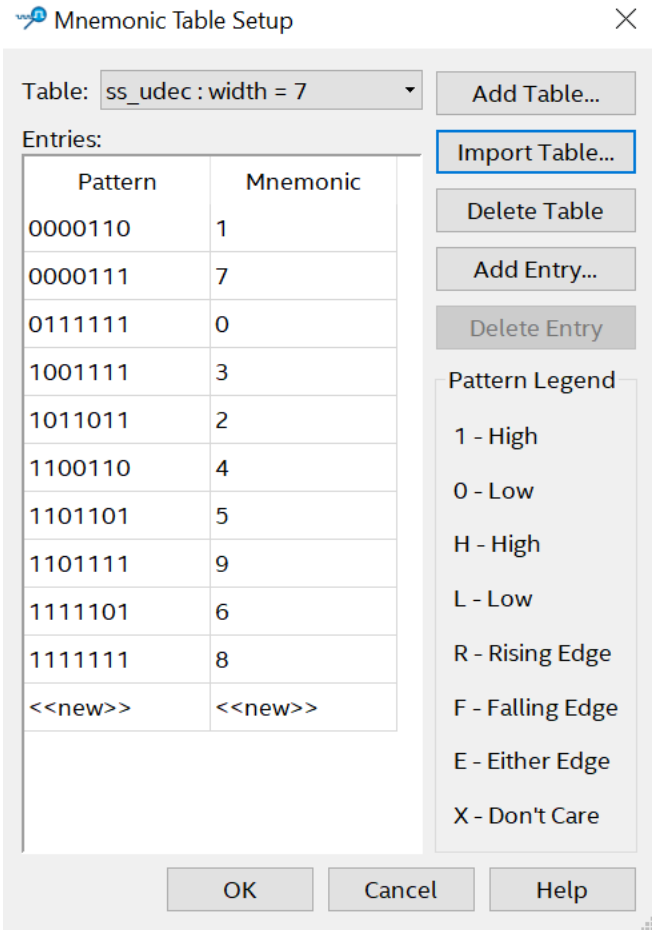


Figure 26

iii Click OK.

e. In the window appeared:



Pattern	Mnemonic
0000110	1
0000111	7
0111111	0
1001111	3
1011011	2
1100110	4
1101101	5
1101111	9
1111101	6
1111111	8
<<new>>	<<new>>

Figure 27

i You could see 7-segment code (left part of the table) and corresponding *mnemonic*, i.e. unsigned decimal values (right side of the table). The table will be used to simplify the debugging procedure.

ii Click OK.

7 **Setup tab**: right click **HEX[6..0]** bus and select **Bus Display Format => ss_udec: width = 7**

a. The mnemonic table will be used for displaying values on the bus.

8 **Setup tab**:

a. selct **div_cnt** bus and **Counter** bus,

b. right-click and select **Bus Display Format => Unsigned decimal**

9 **Setup tab**: right click **Trigger Conditions** field (for signal **RST**) and select Rising Edge.

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	43	43	1 Basic AND
		Lab3_1:Lab3_1_inst|CLK	✓	✓	
		Lab3_1:Lab3_1_inst|DIR	✓	✓	
		Lab3_1:Lab3_1_inst|RST	✓	✓	↑
		Lab3_1:Lab3_1_inst|DIG[3..0]	✓	✓	Xh
		Lab3_1:Lab3_1_inst|HEX[6..0]	✓	✓	XXh
		Lab3_1:Lab3_1_inst|div_cnt[24..0]	✓	✓	XXXXXXXXXXXXXXXXXXXXXXXXXb
		Lab3_1:Lab3_1_inst|Counter[3..0]	✓	✓	XXXXb

Figure 28

10 In the field **Signal Configuration** (main Signal Tap II window), in the area **Clock:**

a. Click ... button

b. In the window appeared:

i Click **List**.

ii Select **c0** output of the **PLL_unit.**

iii Click > sign.

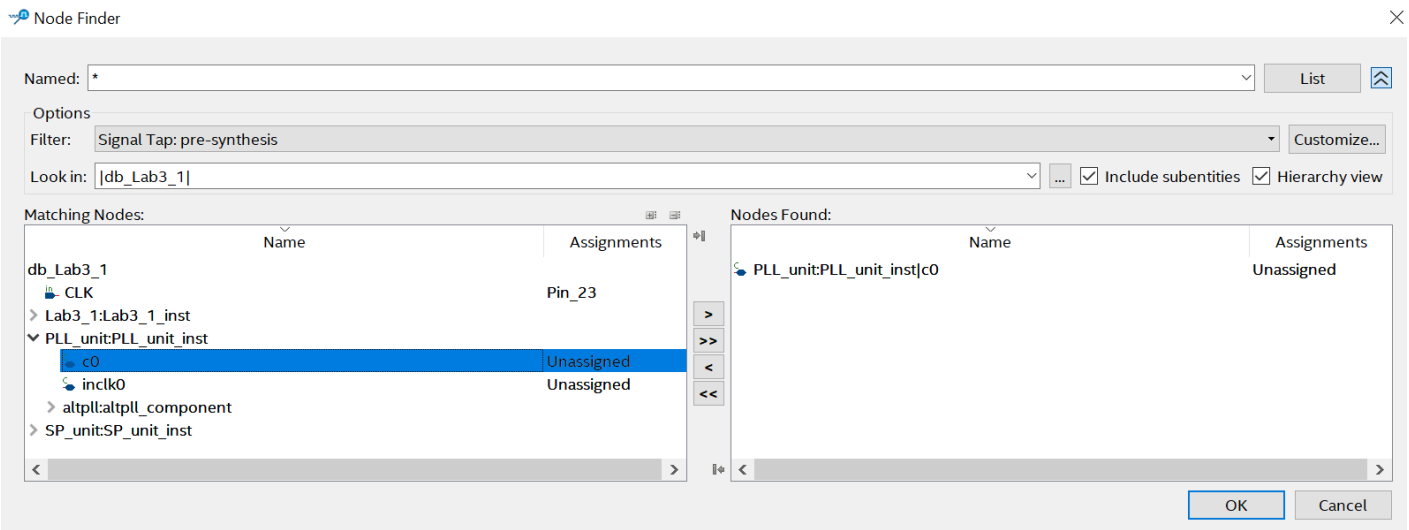


Figure 29

c. Click **OK.**

d. The clock signal for **Signal Tap II** synchronization is assigned.

11 In the field **Signal Configuration** (main Signal Tap II window), in the area **Sample depth,** set 128.

12 Check Signal Configuration field and compare with the Figure 30.

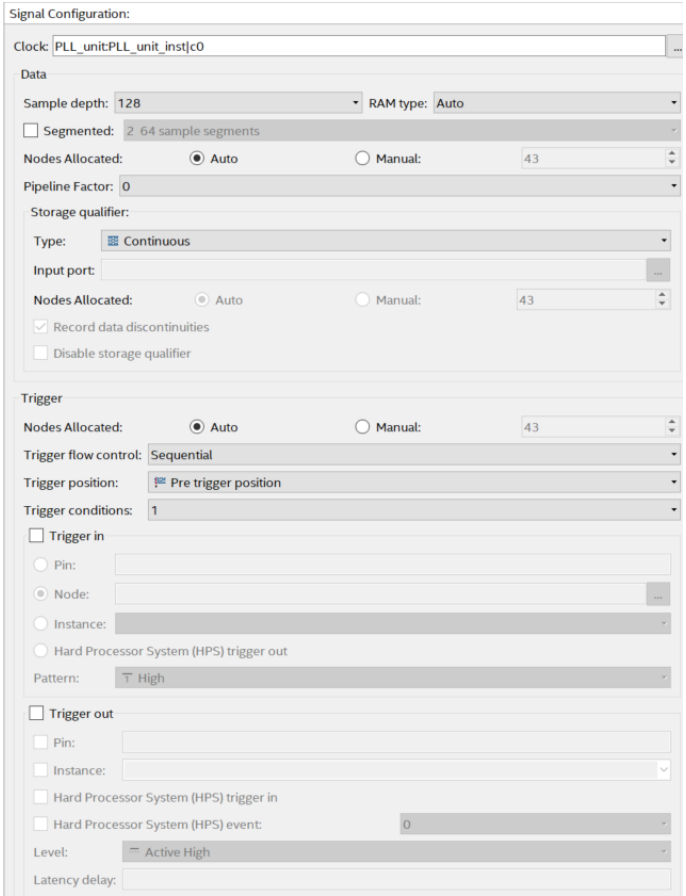


Figure 30

13 In the field **Instance Manager** (main Signal Tap II window), in the area **Instance,** set name of the instance as **ST_unit**.

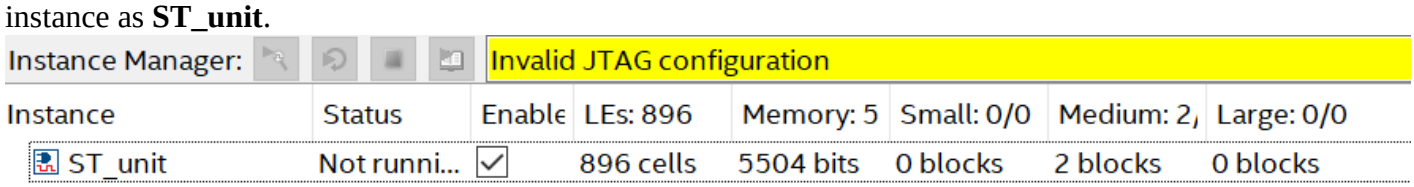


Figure 31

14 Save the current settings: select **File => Save As**

a. Type the name as: **Lab3_1.stp**

b. In the window appeared click **Yes** (it will add the Lab3_1.stp file to the project)

15 Close **Signal Tap II** window.

16 In Quartus Prime window: select **Assignment => Settings => Signal Tap Logic Analyzer**

a. Be sure that the check box is selected and the file Lab3_1.stp is pointed.

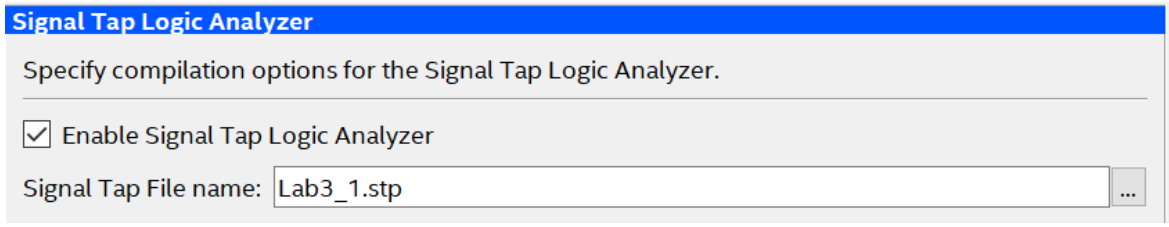


Figure 32

b. Click **OK**.

For the Full Compilation (for correct timing analysis) we need to have .sdc file with timing constrains. For this lab the file **db_Lab3_1.sdc** is provided. It is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1.

1 In Quartus Prime window: select **Assignment => Settings...=> Timing Analyzer.**

2 In the window appeared:

a. Find and Add the **db_Lab3_1.sdc** (it is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1).

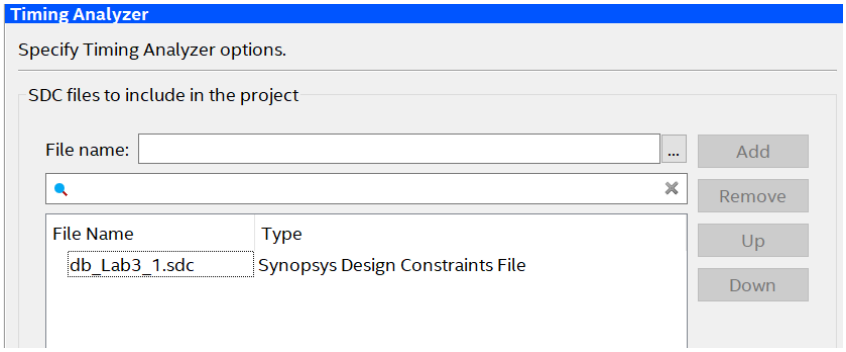


Figure 33

b. Click **OK.**

3 Start full compilation: **Processing => Start Compilation** 

4 Be sure that you have neither Errors nor Critical Warnings.

a. It is acceptable to have some warnings like ones displayed on Figure 34

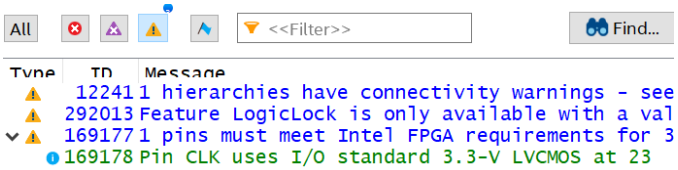


Figure 34

5 Check Compilation Report => Timing Analyzer:

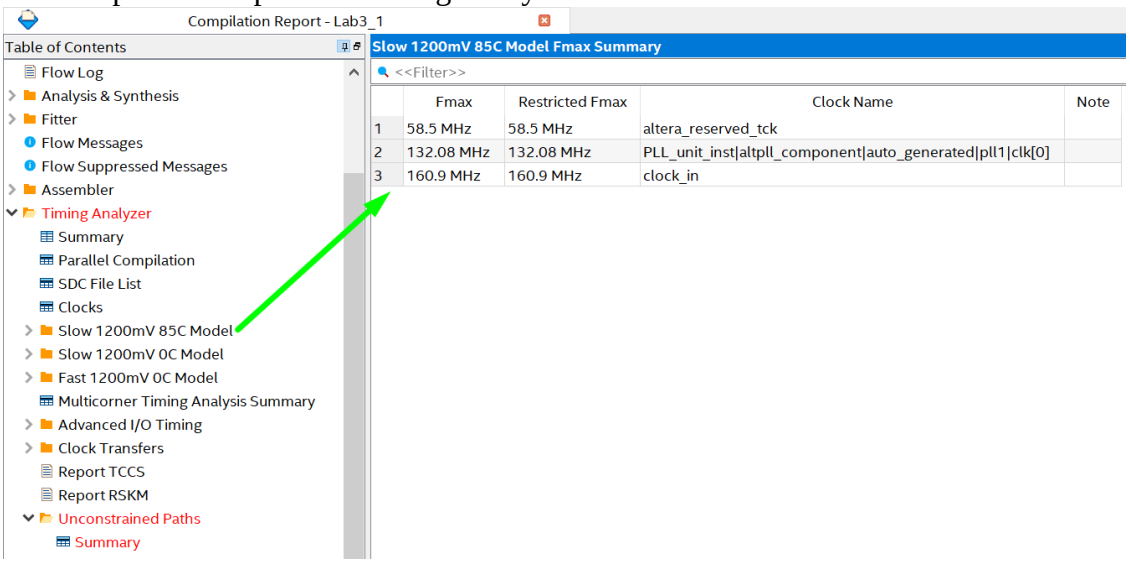


Figure 35

a. There are no any timing violations.

i The allowed Fmax for CLK input (clock_in) in the **Slowest Corner** is much higher than required 25 Mhz.

b. Unconstrained Paths are JTAG inputs for SignalTap II.

Debugging the design

1 Connect the board to USB port of your PC.

2 Power Up your board.

3 In the Quartus Prime window select **Tools => Programmer**  **.**

4 In the window appeared:

a. Click **Hardware Setup.**

b. In the window appeared:

i In the field **Available Hardware** double-click **USB-Blaster.**

ii Click **Close**.

c. Select (using **Add File..** button) file for configuring FPGA: **Lab3_1.sof** (It is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1\output_files)

i If the file already pointed in the field **File** you need to check that it is the file with the right name - **output_files/Lab3_1.sof .**

d. Select check box **Program/Configure.**

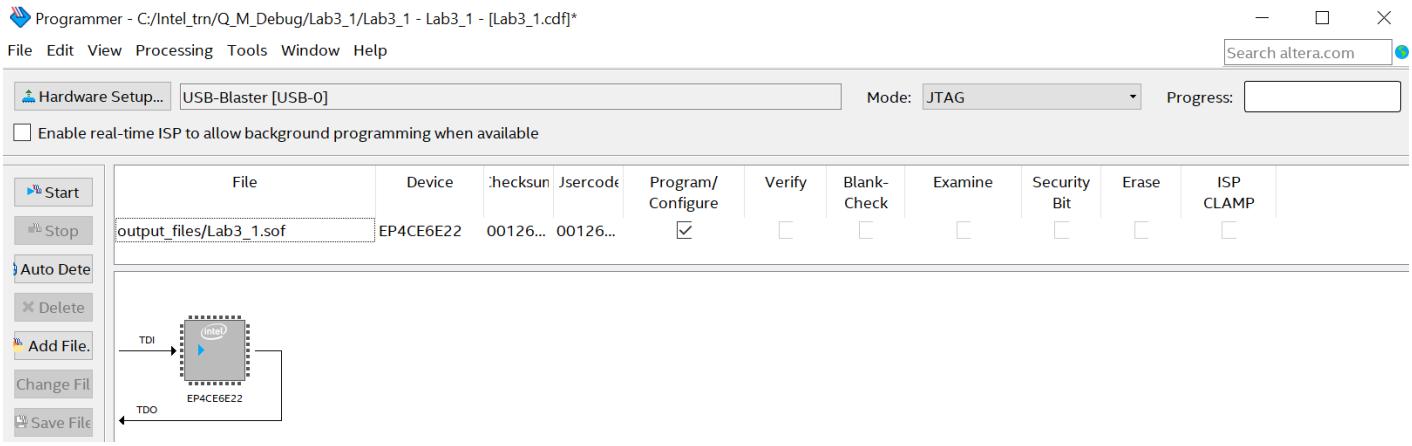


Figure 36

e. Click **Start**.

f. Close Programmer: **File => Close**.

g. In the window appeared (if it is appeared) click **Yes**.

5 In the Quartus Prime window: select **Tools => In-System Source and Probes Editor.**

6 In the window appeared (if it is appeared) click **OK**.

7 In the **In-System Source and Probes Editor** window:

a. In the field **Hardware** select USB-Blaster. You will see that **SP_** instance (created on the previous steps) is recognized and the **Instance Manager** is ***Ready to acquire***.

b. Adjust all settings, signal and busses as pointed on Figure 37.

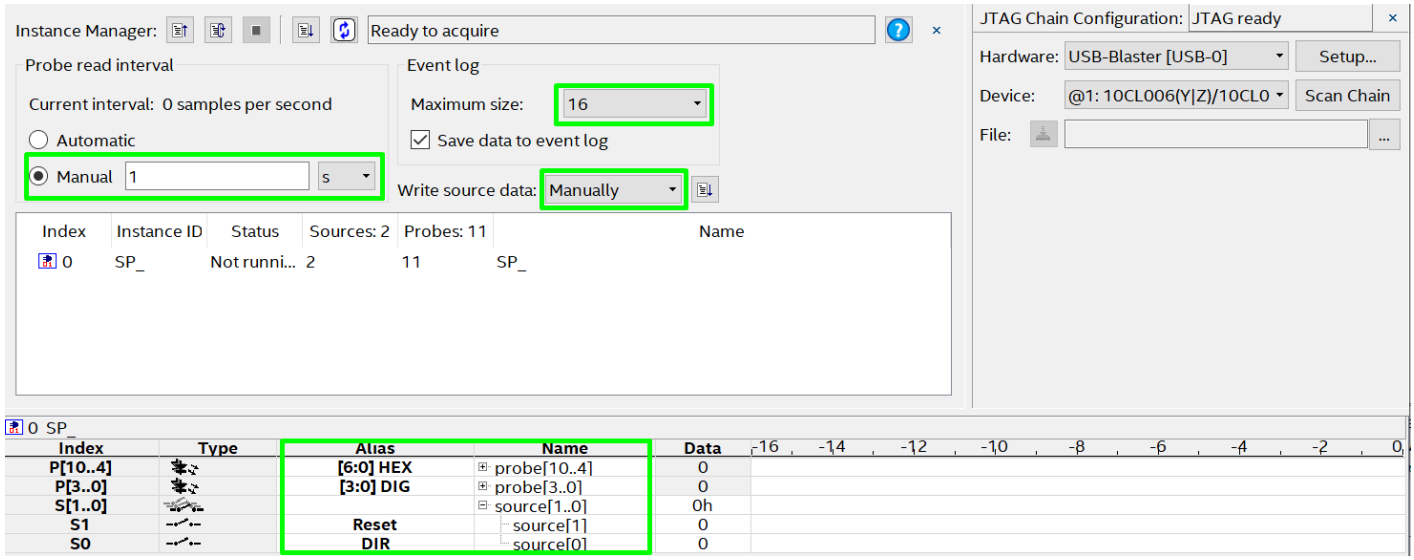


Figure 37

c. Right click **probe[10..4]** and select **Bus Display Format => Hexadecimal.**

d. Right click **probe[3..0]** and select **Bus Display Format => Hexadecimal.**

e. Right click **source[1..0]** and select **Bus Display Format => Hexadecimal.**

f. Save current settings: **File=>Save as..**

i The name is **Lab3_1.spf.**

8 In the **In-System Source and Probes Editor** window:

a. Click **Processing => Continuously Read Probe Data** 

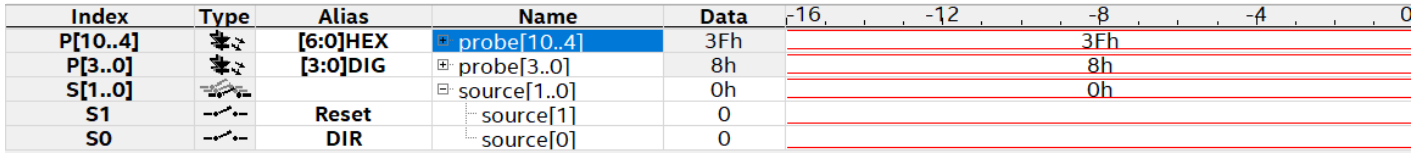


Figure 38

You will see that:

- HEX bus outputs have a constant value 3F (it is 7-segment code for 0).
  - It is right while Reset signal is 0 (Active).
- DIG bus outputs have a constant value 8 .
  - It is right: the left most digit of 7-segment indicator is selected.

b. Click in **Data** field of **Reset** signal (value from 0 will be changer to 1 and it will be displayed in red).

c. Click **Processing => Write Source Data**  .

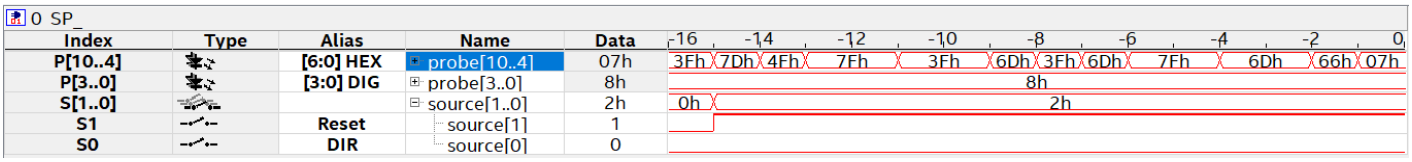


Figure 39

You will see that HEX bus outputs values are not in the expected sequence for the 7-segments codes. The expected sequence: when DIR = 0 (counting UP): 3F; 06; 5B; 4F; 66; 6D; 7D; 07; 7F; 6F …

d. Click **Processing => Stop Continuously Reade Probe data**  .

e. Click in **Data** field of **Reset** signal (value from 1 will be changer to 0 and it will be displayed in red).

f. Click **Processing => Write Source Data**  .

g. ***Keep the In-System Source and Probes Editor window opened!***

The hint: to understand a reason of the wrong behavior you need to check the design with Signal Tap II.

9 In the Quartus Prime window: select **Tools => Signal Tap Logic Analyzer**

10 In the **SignalTap II** window:

a. In the field **Hardware** select USB-Blaster. You will see that SignalTapII instance ST_unit is recognized and the **Instance Manager** is ***Ready to acquire***.

11 In the **SignalTap II** window: select **Processing => Run Analysis** 

a. Signal Tap II will wait a **rising** edge on RST signal.

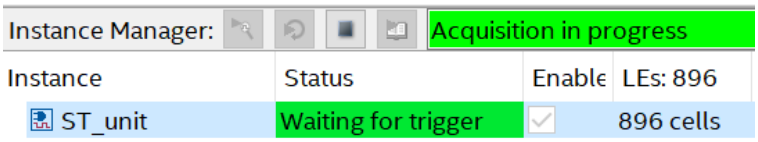


Figure 40

12 In the **In-System Source and Probes Editor** window:

a. Click in **Data** field of **Reset** signal (value from 0 will be changer to 1 and it will be displayed in red).

b. Click **Processing => Write Source Data**  .

13 In the **SignalTap II** window: you will see in the **Data** tab the captured data.

Figure 41

You can see that:

- Code Sequence (7-segment codes) on the HEX bus outputs (*The values are displayed in digital format in accordance with Mnemonic table*) is correct (*DIR =0, i.e. Counting UP*) and is aligned with the outputs of the Counter bus.
- It seems that there are some issues with Clock divider:
    - It is expected that Counter bus outputs and Code Sequence (7-segment codes) on the HEX bus outputs change after 25 000 000 periods of input clock (CLK).
    - In the waveform, we can see that those change after 4 periods of input clock (CLK).

You need to check and correct the source files of your project.

The hint:

We already debugged **Lab3_1.v** source code during the simulation stage. Therefore, we can assume that a problem is in **db_Lab3_1.v** code. Pay attention on a division factor for the clock divider, which is a parameter in Lab3_1.v and correct db_Lab3_1.v code.

```
Lab3_1 #(25'd25000000) Lab3_1_inst
(
    .CLK        (CLK        ),
    .RST        (db_reset   ),
    .DIR        (db_dir     ),
    .DIG       (DIG         ),
    .HEX        (HEX        )
);
```

Figure 42

14 Correct and save source code db_Lab3_1.v.

15 In Quartus Prime window: start full compilation: **Processing => Start Compilation.**

a. Click **Yes** in all windows appeared.

16 Be sure that you have neither Errors nor Critical Warnings.

17 Check **Compilation Report => Timing Analyzer.**

a. There are no any timing violations.

18 Open **Programmer** and reconfigure the device (if you wish you can do reconfiguration just from In-System Source and Probes Editor window or from Signal Tap II window).

19 Close Programmer.

20 Open **In-System Source and Probes Editor** window (re-initialize it if it is necessary by clicking **Scan Chain**)

a. Set Reset = 0 and DIR =0.

b. Click **Processing => Write Source Data**  **.**

c. Click **Processing => Continuously Read Probe Data**  **.**

You will see 3F on HEX output. It is right result.

d. Set Reset = 1 .

e. Click **Processing => Write Source Data**  **.**

You will see the right code sequence on HEX bus outputs. The expected sequence:

- For DIR = 0 (Counting UP): 3F; 06; 5B; 4F; 66; 6D; 7D; 07; 7F; 6F …

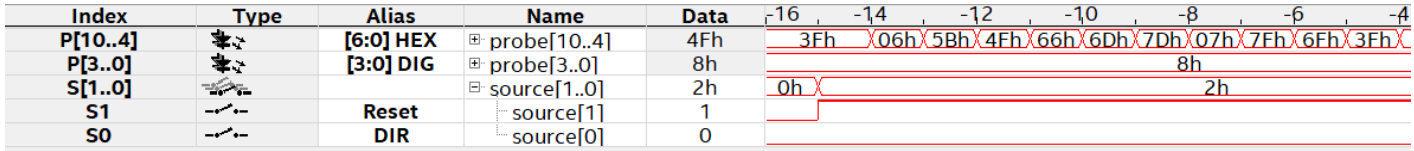


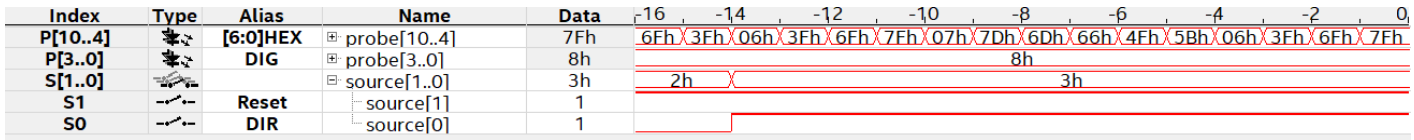
Figure 43

f. Set DIR = 1.

g. Click **Processing => Write Source Data**  .

You will see the right code sequence on HEX output. The expected sequence:

- For DIR = 1 (Counting Down): 3F; 6F; 7F; 07; 7D; 7D; 6D; 66; 4F; 5B; 06; …



h. Click **Processing => Stop Continuously Reade Probe data**  .

i. Set DIR = 0.

j. Click **Processing => Write Source Data**  .

21 Open **Signal Tap II** window (re-initialize it if it is necessary by clicking **Scan Chain**).

22 In the **Signal Tap II** window:

a. **Tab Setup**: Change Trigger Conditions filed of RST signal to Don't Care.

i Right click the field and select .

b. **Tab Setup**: Change Trigger Conditions for **div_cnt** bus to 25000000.

i Right click in the field **Trigger Condition** for **div_cnt** bus => Insert Value.

ii Type 25000000.

iii Click Ok.

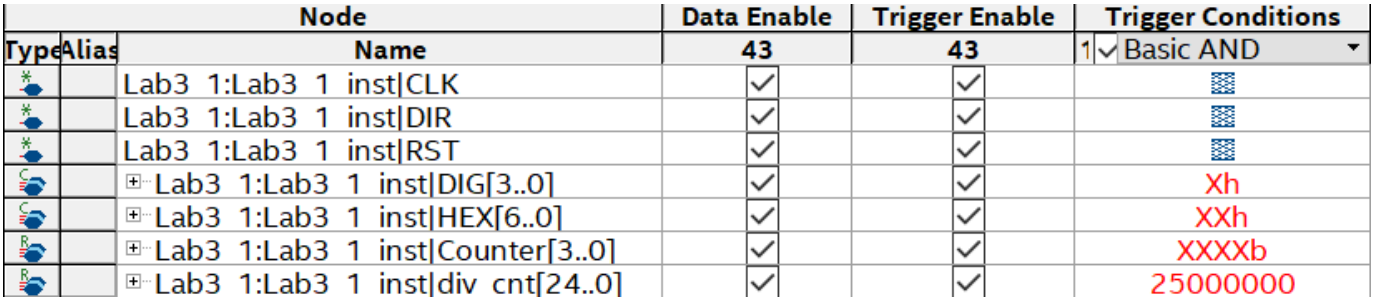


Figure 44

23 In the **Signal Tap II** window:

a. Click **Processing => Run Analysis**  .

b. Zoom in to see waveform like on Figure 45.

You could see that the signal **div_cnt** equal the value 25000000 and then start to count from 1.

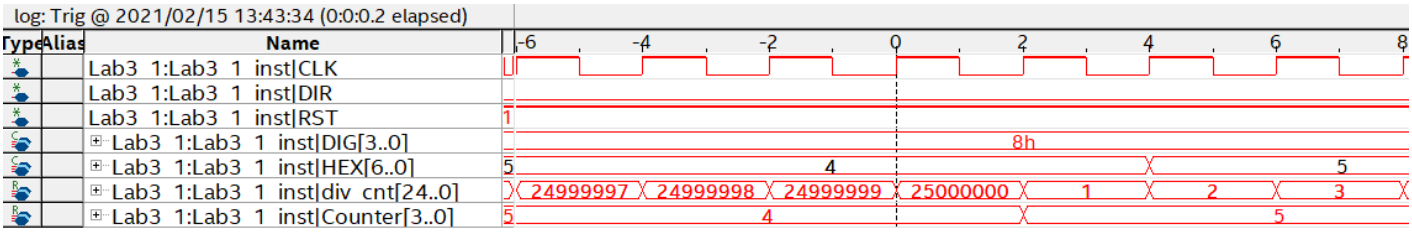


Figure 45

24 In the **Signal Tap II** window:

a. **Tab Setup**: Change Trigger Conditions field of HEX bus to Either Edge (capital E).

i Right click in the field **Trigger Condition** for **HEX** bus => Either Edge.

b. **Tab Setup**: Change Trigger Conditions field of div_cnt bus to Don't care (letter X).

i Right click in the field **Trigger Condition** for **div_cnt** bus => Don't care.

c. **Tab Setup**: Set **Segmented Acquisition Trigger** field (field which is just under the Trigger Conditions name) as **Basic OR.**

d. **Signal Configuration** field: Check **Segmented** check box.

e. **Signal Configuration** field: Select value for Segmented as **16 8 sample segments.**

f. **Signal Configuration** field: in **Trigger Position** area set **Center Trigger Position.**

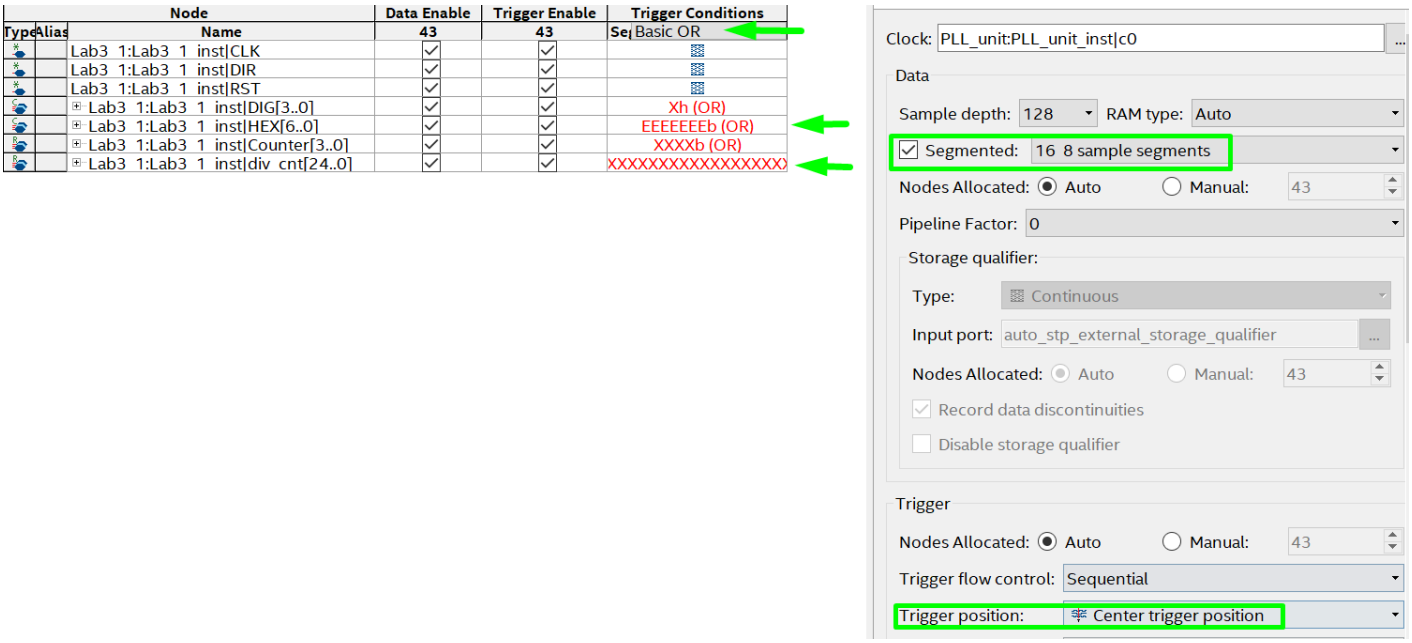


Figure 46

g. Save file with current settings.

25 In the **Signal Tap II** window: start full compilation selecting **Processing => Start Compilation**

26 Be sure that you have neither Errors nor Critical Warnings.

27 Check **Compilation Report => Timing Analyzer.**

b. There are no any timing violations.

28 In Quartus Prime window: open Programmer and reconfigure the device (you can do reconfiguration just from In-System Source and Probes Editor window and Signal Tap II windows too.).

29 Close Programmer.

30 Open **In-System Source and Probes Editor** window (re-initialize it if it is necessary by clicking **Scan Chain**).

a. Set Reset = 1 and DIR =0.

b. Click **Processing => Write Source Data**  .

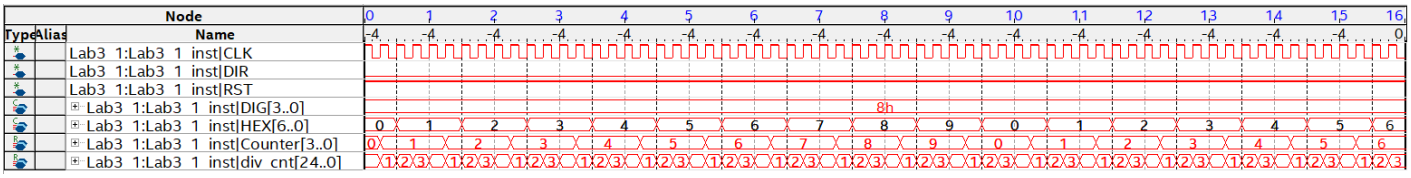
31 Open **Signal Tap II** window (re-initialize it if it is necessary by clicking **Scan Chain**).

32 In the **Signal Tap II** window:

a. Click **Processing => Run Analysis**  .

b. You will need to wait near 16s for getting data: Signal Tap II need to wait 16 values on the HEX bus. The values are updated one time per second.

You will see the right sequence on the HEX bus output for DIR = 0 – i.e. counting UP (values on the bus are from the Mnemonic Table).



33 In In-System Source and Probes Editor window

a. Set DIR = 1.

b. Click **Processing => Write Source Data**  .

34 In Open Signal Tap II window.

a. Click **Processing => Run Analysis**  .

b. You will need to wait near 16s for getting data: Signal Tap II need to wait 16 values on the HEX bus. The values are updated one time per second.

You will see the right sequence on the HEX bus output for DIR = 1 – i.e. counting DOWN (values on the bus are from the Mnemonic Table).

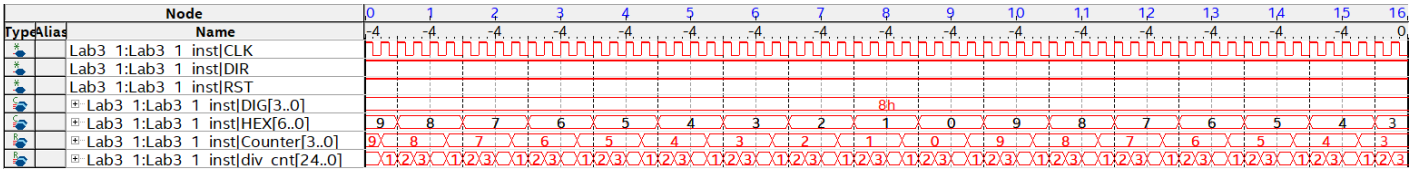


Figure 47

35 Save and close **Signal Tap II** widow.

36 Save and close **In-System Source and Probes Editor** widow.

Implementing the design

Top-level file for implementation

Source file for implementation stage should be based on db_Lab3_1.v file with some modifications implemented:

- All inputs and outputs of the project should be added and associated with FPGA inputs
- All additional components, used for debugging purposes, should be removed.

The source file impl_Lab3_1.v is provided for the lab. It is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1.

```
module impl_Lab3_1 (
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "23"  *)
        input CLK,
    (* altera_attribute = "-name IO_STANDARD \"2.5 V\"", chip_pin = "64" *)
        input RST,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "88" *)
        input DIR,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "84, 76, 85, 77, 86, 133, 87" *)
        output [6:0] HEX,
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"", chip_pin = "73, 74, 80, 83" *)
        output [3:0] DIG
    );

Lab3_1 #(25'd25000000) Lab3_1_inst
(
    .CLK        (CLK    ),
    .RST        (RST    ),
    .DIR        (DIR    ),
    .DIG        (DIG    ),
    .HEX        (HEX    )
);

endmodule
```

Figure 48

1. Add the file **impl_Lab3_1.v** to the project:
   a. In Quartus Prime select **Project => Add\Remove Files in Project…**
   b. In the window appeared **Find**  the **impl_Lab3_1.v** file in the folder **C:\Intel_trn\Q_M_Debug\Lab3_1**
   c. Click **Open**
   d. Click OK.
2. Set the file **impl_Lab3_1.v** as a top level file for the project
   a. Select: **Project Navigator => Files**
   b. Right-click **impl_Lab3_1.v** file and select **Set as a top level Entity**
   c. In the **Project Navigator** select **Hierarchy** and check that **impl_Lab3_1** is a top level entity.
3. Select: **Processing => Start => Start Analysis & Elaboration**
4. Select: Tools => Netlist Viewer => RTL Viewer
5. Be sure that you got something like Figure 49.

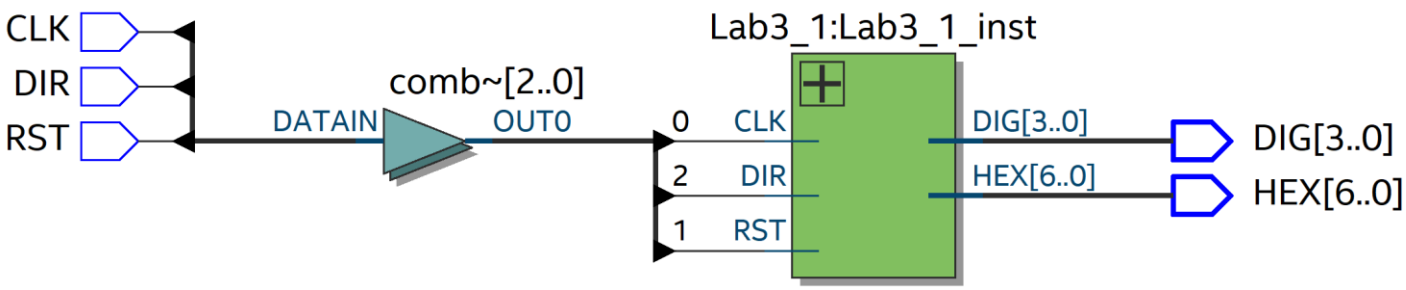


Figure 49

Full compilation

For the Full Compilation (for correct timing analysis) we need to have .sdc file with timing constrains. For this lab the file **impl_Lab3_1.sdc** is provided. It is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1.

1 In Quartus Prime window: select **Assignment => Settings...**

2 In the window appeared:

a. Select Timing Analyzer

i Select **db_Lab3_1.sdc** file and click Remove

ii Find and Add the **imple_Lab3_1.sdc** (it is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1)

File Name	Type
impl_Lab3_1.sdc	Synopsys Design Constraints File

Figure 50

b. Click **Apply.**

c. Select **Signal Tap Logic Analyzer.**

i Clear check box **Enable Signal Tap Logic Analyzer.**

d. Click **OK**.

3 In Quartus Prime window: start full compilation selecting **Processing => Start Compilation**

4 Be sure that you have neither Errors nor Critical Warnings after Full compilation.

a. It is acceptable to have some warnings like ones displayed on Figure 51.

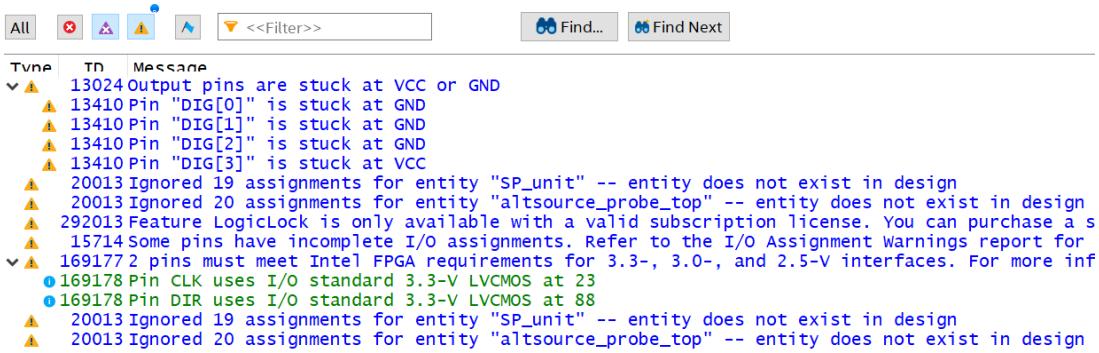


Figure 51

5 Check Compilation Report => Timing Analyzer:

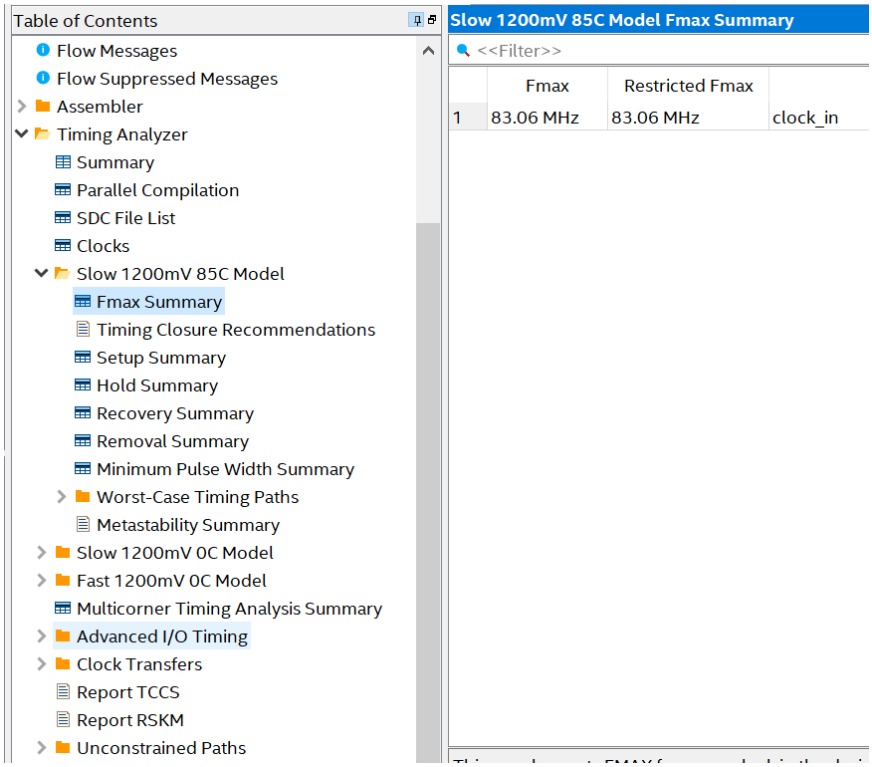


Figure 52

a. There are no any timing violations.

i The allowed Fmax for CLK input (clock_in) in the **Slowest Corner** is much higher than required 25 Mhz.

Programming and verifying the board

1 Connect the board to USB port of your PC.

2 Power Up your board.

3 In the Quartus Prime window select **Tools => Programmer**  .

4 In the window appeared:

a. Click on **Hardware Setup**

b. In the window appeared:

i in **Available Hardware** field double-click **USB-Blaster**

ii Click Close

c. Select (using **Add File..** button) file for configuring FPGA: **Lab3_1.sof** (It is in the folder C:\Intel_trn\Q_M_Debug\Lab3_1\output_files)

i If the file already pointed in the field **File** you need to check that it is the file with the right name - **output_files/Lab3_1.sof**

d. Select check box **Program/Configure**

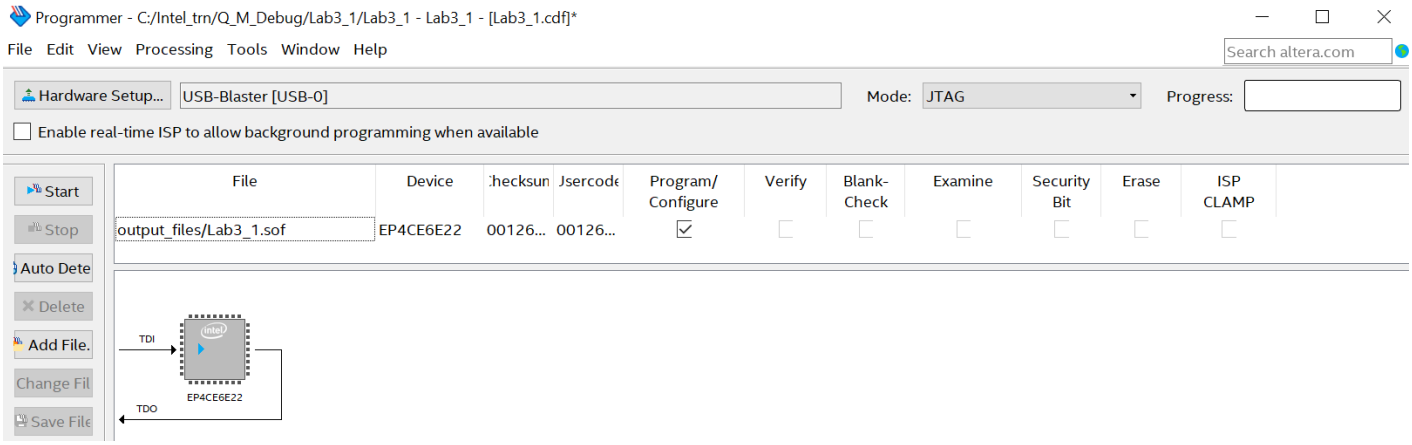


Figure 53

e. Click **Start**.

f. Close Programmer: File => Close.

5 On the Board:

a. Set sw [7] = 0

i Check that 7-segment indicator displays 0, 1, …9, 0…

b. Set sw [7] = 1

i Check that 7-segment indicator displays 9, 8, … 0, 9…

c. Push button PBA

i Check that 7-segment indicator displays 0

Conclusions

With this lab you learnt how to:

- create design in Quartus Prime
- simulate and debug source code using ModelSim
- debug design on board using ISSP and SignalTapII.
- Implement and check the design on the board.