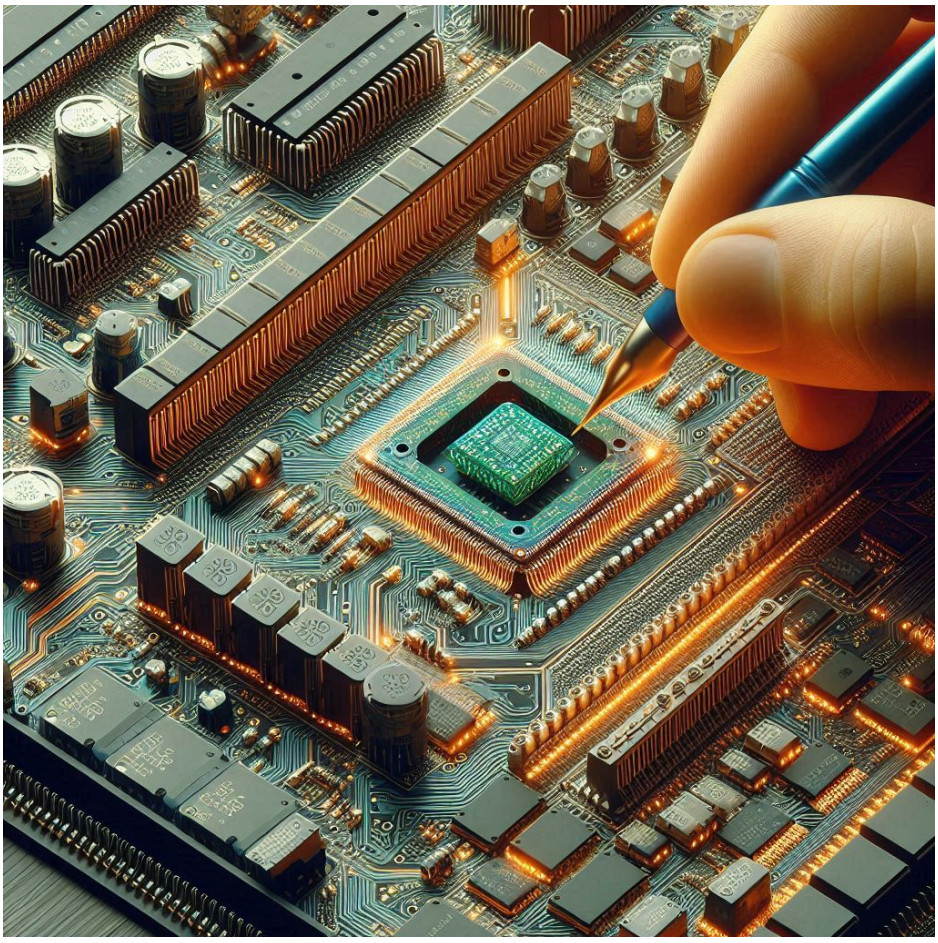
A Comprehensive Guide to Creating and Compiling FPGA Projects in Quartus

Mohamed Niazy Hassaneen Ali Mohamed

September 2024

A Project Submitted as Part of Professional Development
Prepared by a Graduate of Electronics and Electrical Communications Engineering
Cairo University



Contact Information
Gmail: mohamedniazy972@gmail.com
LinkedIn: Mohamed Niazy

Contents

1 Create a project **1**

2 Compilation of project **3**
2.1 Configure compiler settings . . . 3
2.1.1 Using Quartus GUI . . . 3
2.1.2 Using TCL command . . . 3
2.2 Compilation processes . . . 4
2.2.1 Analysis and Synthesis . . . 4
2.2.2 Fitter (Place & route) . . . 7
2.2.3 Assembler (Generate programming files) . . . 7
2.2.4 Timing Analysis . . . 7
2.2.5 EDA Netlist Writer . . . 12
2.3 Compile Design & Flow Summary . . . 12

3 RTL Simulation **13**
3.1 Add the path of the simulation tool . . . 13
3.2 Add test bench files and set test settings . . . 13
3.3 Run the Simulation . . . 14

4 Program Device (Load Design to FPGA) **14**
4.1 Programming vs Configuration . . . 14
4.2 Memory Types in FPGAs . . . 14
4.3 Compilation and Bitstream Files . . . 15
4.4 Loading the Design . . . 15

5 Feedback **16**

Abstract

This document provides a comprehensive guide to the FPGA design flow using Quartus Prime software. The design flow covers essential steps from creating the design in RTL to loading the final bitstream file onto the FPGA. It includes detailed explanations of the key processes such as project creation, design entry, synthesis, RTL simulation, and device programming. In addition, it outlines how to configure simulation tools, manage compilation results, and understand programming vs configuration in FPGA systems. Through a series of practical illustrations and step-by-step instructions, this guide aims to support digital designers in successfully implementing FPGA designs using Quartus.

Introduction

FPGA (Field Programmable Gate Array) technology has revolutionized the field of digital design, allowing engineers to create complex logic circuits that can be programmed and reprogrammed based on application needs. Quartus Prime, developed by Intel, is a robust tool that enables designers to implement, simulate, and program FPGAs efficiently.

This document serves as a practical guide to the design flow within Quartus, detailing each critical step from project setup to programming the FPGA device. Designers new to FPGA development often face challenges understanding the compilation process, simulation setup, and loading designs onto the hardware. This guide simplifies these concepts, walking the user through creating a design, running RTL simulations, and programming the final bitstream file into the FPGA.

By following this document, engineers will gain a solid understanding of the end-to-end design flow within Quartus and acquire the practical knowledge needed to implement and test FPGA designs with confidence.

1 Create a project

Contents

1. open Quartus then press on "**New Project Wizard**" then next

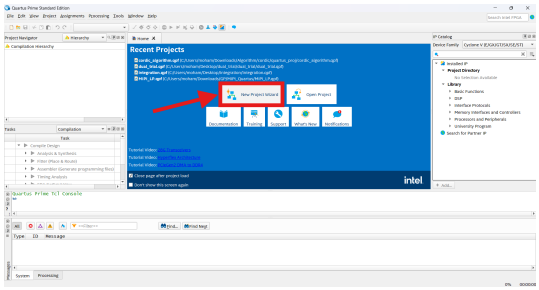


Figure 1: Create Project

2. write the name of project and the name of top_module then next , note that the name of any RTL file must be the same name of entire module name .

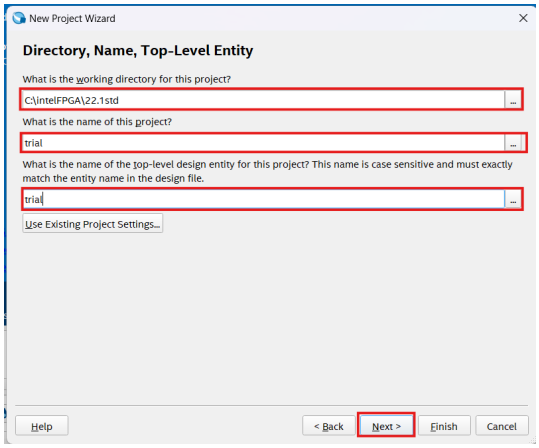
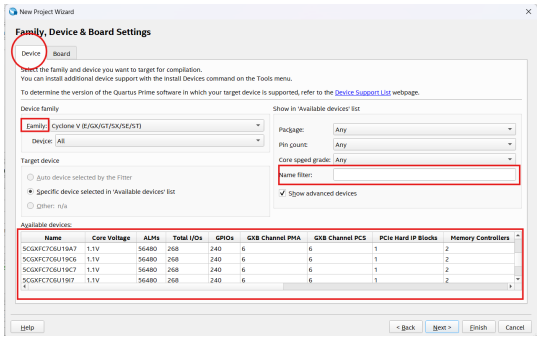


Figure 2: Set project name

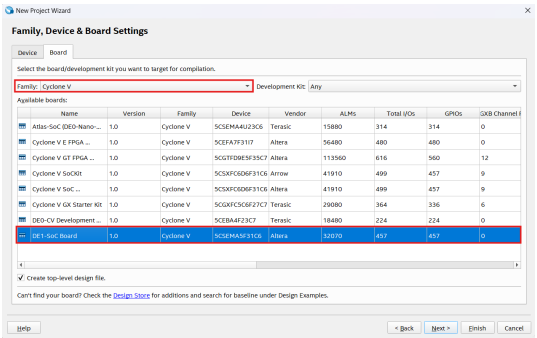
3. select "**Empty Project**" then next

4. add files then next (if u didn't have files yet, skip this step)

5. select the family of used FPGA and its name , if u have board select its name (the difference between two option is more information will be included in ur project about the connection between the FPGA pins and the modules in the kit(like 7_segment / push_buttoms / LEDs) this step is useful when we want to access the modules in our board)



(a) Select devices



(b) Select boards

Figure 3: Select the FPGA

6. set all none (we will configure them later) then press finish

2 Compilation of project

In this chapter, we will cover several key steps in the compilation processes. First, we will discuss how to configure compiler settings to achieve the desired results. Second, we will explain how to perform a full compilation. Third, we will demonstrate how to examine the compilation results.

2.1 Configure compiler settings

now we will assign the compilation parameters of our project

2.1.1 Using Quartus GUI

press **Assignments** then

- **Devices**(to assign the board or FPGA family , we did it before)

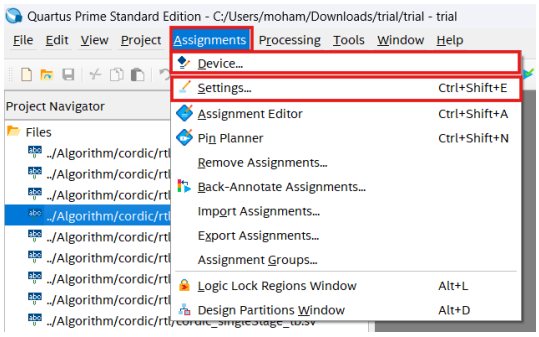
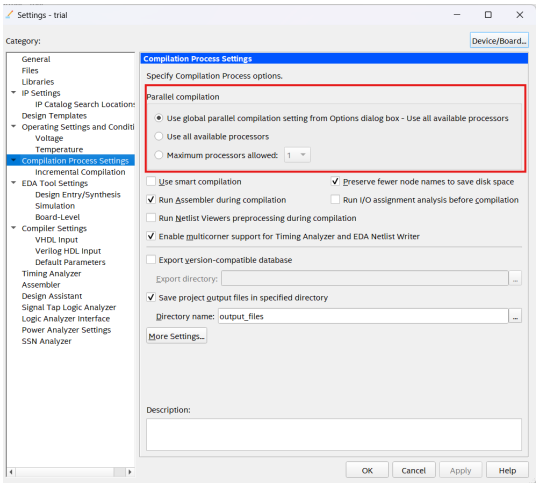
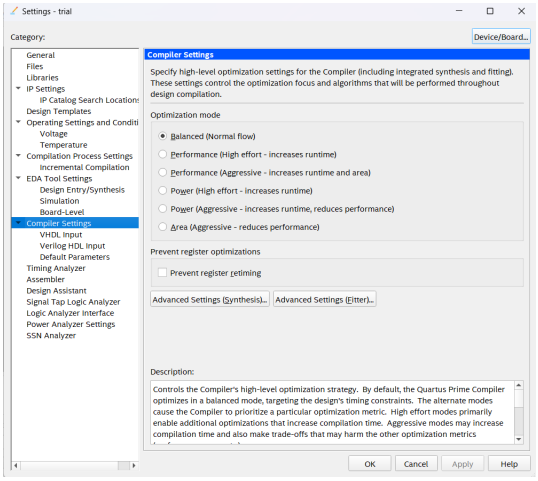


Figure 4: set compilation Assignments by GUI

- **Settings**: it contain all the Assignments we can config for our project like
  - **simulation** -i will discuss later-
  - **compilation process setting** -assign No. of processor used in compilation-
  - **compiler settings** -assign the optimization mode of the compilation process-



(a) no.of processors



(b) optimization modes

Figure 5: set Assignments

2.1.2 Using TCL command

we can set Assignments of compilation by tcl command written in

1. **file.QSf** (not recommended): All of the compiler's settings we configured are saved in the `file.qsf` as TCL commands, which Quartus runs each time we open our project. To view the contents of `file.qsf`, navigate to your project directory, search for the `.qsf` file, and open it in Visual Studio Code.

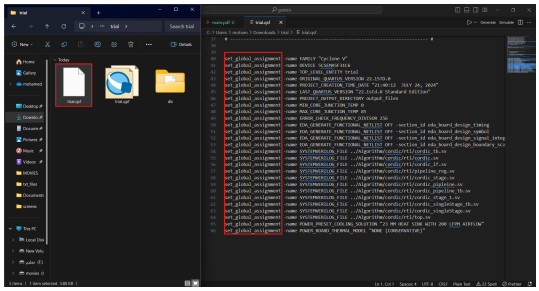


Figure 6: contain of QSF file

2. **TCL window**

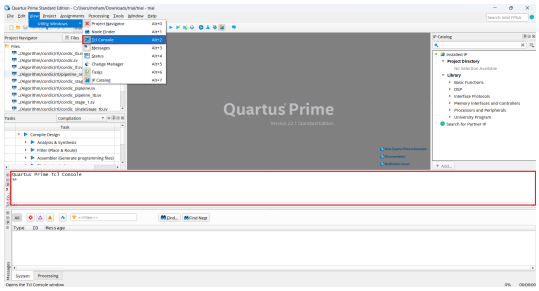


Figure 7: TCL console

2.2 Compilation processes

2.2.1 Analysis and Synthesis

The Analysis and Synthesis module in Quartus performs several critical functions during the compilation process. First, it checks the design source files for errors. Next, it builds a database that organizes all the design files into a hierarchical structure. Finally, it synthesizes and optimizes the logic design, mapping the design logic to the device resources through **Pin Planner** and **Chip Planner**.

In most cases, you are not dealing with just the FPGA device but with a board that contains the FPGA and many peripherals (such as LEDs, 7-segment displays, switches, keys, and communication protocols), and it may also include a processor (in the case of an SoC board). Therefore, FPGA pins need to be assigned to these peripherals to enable the design to interact with them. In the following, I will explain **Pin Assignments** in more detail.

- **Get the Pin Names:**

  An important question arises: where can I get the names of the pins connected to the peripherals? The answer is always in the user guide manual. Any information about the board can be found in the manual.

  1. Search on Google for **name-of-board user manual** and download the PDF from any site.

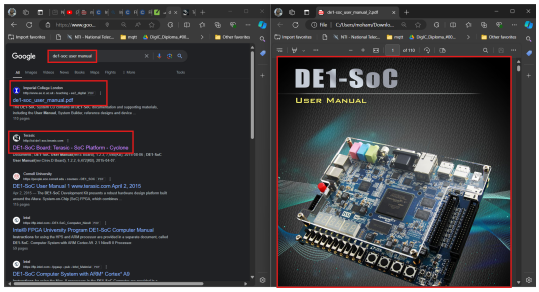


Figure 8: Download user manual

  2. Search for what you need. In most cases, you need to insert clocks, so open the manual and search for the term "clock." Press enter until you find the table that lists all clocks on the board. Note that not

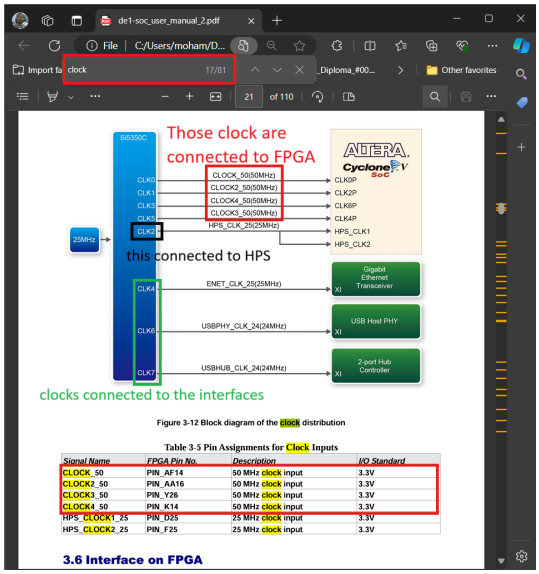


Figure 9: Find clocks in user manual

all clocks in this table are connected to the FPGA; some may be connected to communication interfaces or the HPS. Make sure that the chosen clock is connected to the FPGA, and take note of its period.

If we want to access any peripheral, go to the introduction chapter in manual and look up the layout of the board. Get the name of the desired peripheral, then search for it until you find the table containing the pin names. An important note: when dealing with 7-segment displays or LEDs, check whether they are connected to a common anode or cathode. Similarly, when working with keys, check if they are connected with pull-up or pull-down resistors.

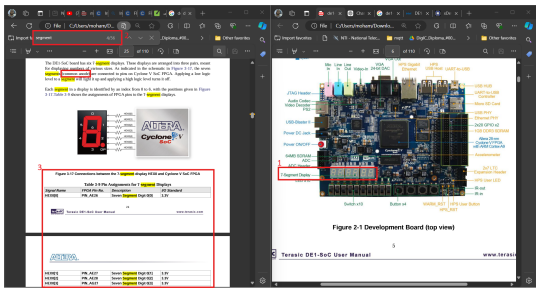


Figure 10: Get 7-Segment Pins

- **Pin Assignments using Pin Planner:**

1. From the **Analysis & Synthesis** section, run "Analysis & Elaboration." Now all the ports in your design will be recognized by Quartus. Then, double-click on **Pin Planner**.

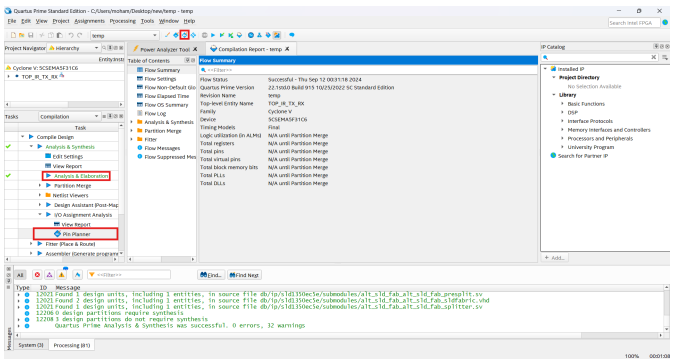


Figure 11: Open Pin Planner

2. Enter the pin name in the **Location** field (change the I/O standard if necessary), then double-click on **Run I/O Assignments** to check for any issues.

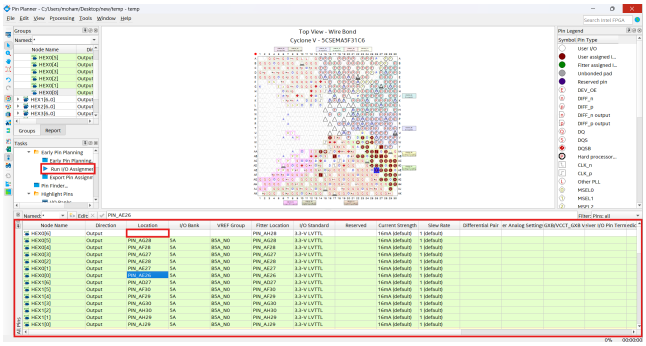


Figure 12: Assign Ports to Pins

- **Pin Assignments using System Builder:**

  This method is a bit tricky but useful for becoming familiar with project files like the **QSF** file. Earlier, we learned how to assign pins to peripherals like the 7-segment display, but if we want to use all 7-segment displays on the board, we would need to assign 35 pins manually (which can be tedious). Instead, we will use the System Builder to complete this task more efficiently.

  1. Download the System Builder for your board, then run **BoardName_SystemBuilder.exe**. Select the name of your project, then select the peripherals for which you want to assign pins, and click **Generate**. Open the directory of generated project , then open the **file.qsf**.

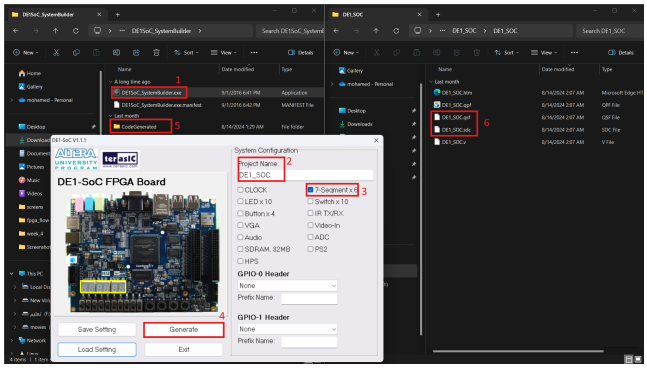


Figure 13: Generate project with pin assignments

  2. Open the **file.qsf** for your project and copy the peripheral assignments from the **file_sb.qsf** generated by the System Builder into the **file.qsf** for your project (make sure Quartus is closed while doing this).

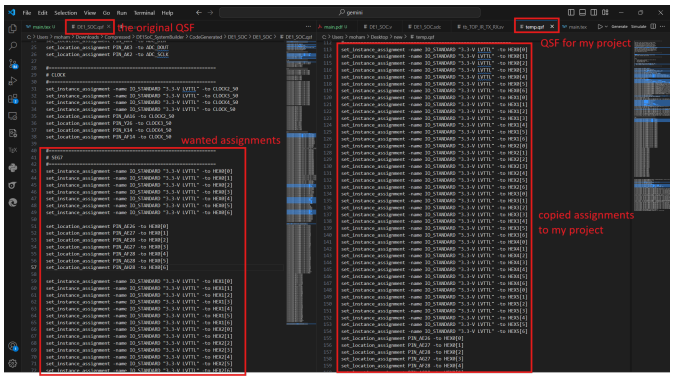


Figure 14: Edit the QSF files

  3. Finally, make sure the port names in your design match the names used in the pin assignments.

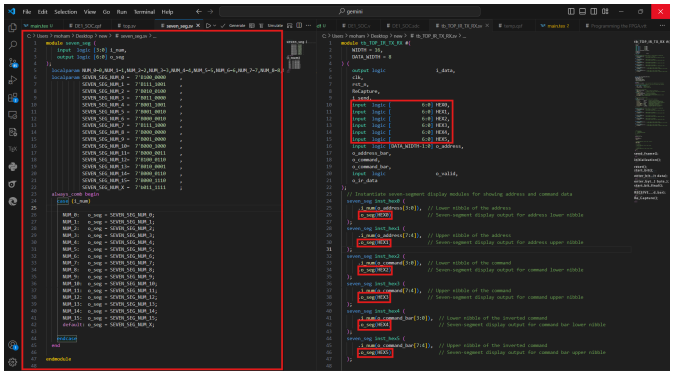


Figure 15: Use assigned pins in design

2.2.2 Fitter (Place & route)

The Fitter module in Quartus takes the synthesized logic and maps it onto the physical resources of the target device. It places the logic into the appropriate logic cells and establishes the necessary routing between these cells. Additionally, the Fitter ensures that timing requirements are met and optimizes the placement for performance and resource utilization.

2.2.3 Assembler (Generate programming files)

The Assembler completes the project processing by finalizing the placement of logic cells, pin assignments, and routes. It integrates these elements into a device programming image, which is then converted into one or more programming files. This process ensures that the design is properly configured and optimized for the target device, preparing it for programming and deployment.

2.2.4 Timing Analysis

The Static Timing Analyzer in Quartus evaluates and verifies the timing performance of the design to ensure it meets the specified requirements. This process involves analyzing the timing paths and validating that the design operates within the required timing constraints. To perform a full timing analysis, you must first complete the synthesis and fitting stages. However, you can run early timing estimates before fitting to assess preliminary timing and help refine your design constraints. This early analysis can provide insights into potential timing issues and guide adjustments before the final fitting process.
Next, we will outline some steps to help you become familiar with the **TimeQuest timing analyzer**.

1. **Open the TimeQuest Timing Analyzer on an FPGA design:**
   from **Tools** select **Timing Analyzer**

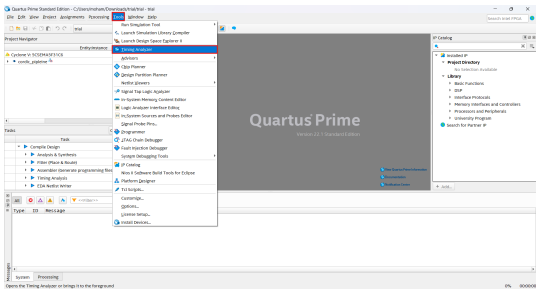


Figure 16: Open the TimeQuest Timing Analyzer

2. **Create a Timing Netlist:**
   From **task window** press double click on **create Timming Netlist** or from **Netlist** select **create Timming Netlist**

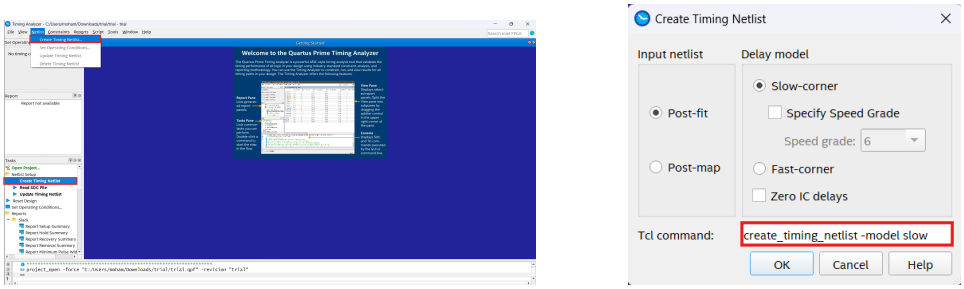


Figure 17: Create Time Netlist

3. **Write constraints in TimeQuest:**

   (a) **Create Clock:** From the **Constraints** menu, select **Create Clock**, then:

   - Set the name of the clock you will create.
   - Specify the period for your system.
   - Connect your created clock to the RTL clock by selecting the target.
   - Enter the name of the clock port in the filter to help you find your clock in the system. (If you wrote your RTL in SystemVerilog and used an interface, write the name of the instantiated interface before the name of the clock).
   - select port then press **ok** then run
   - After pressing **Run**, look in the TCL command window. You will see the TCL command representing the actions you previously performed using the GUI.

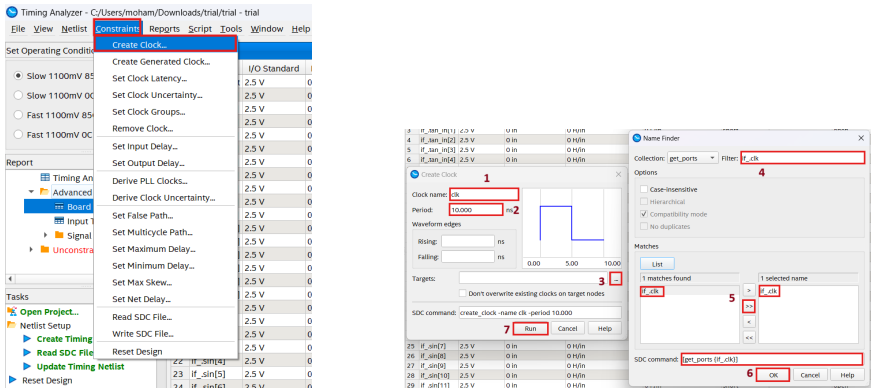


Figure 18: Create Clock

   (b) **Update the timing netlist:** Double-click **Update Timing Netlist** in the Task Window. In general, after making any changes to the constraints, update the timing netlist to reflect these changes.

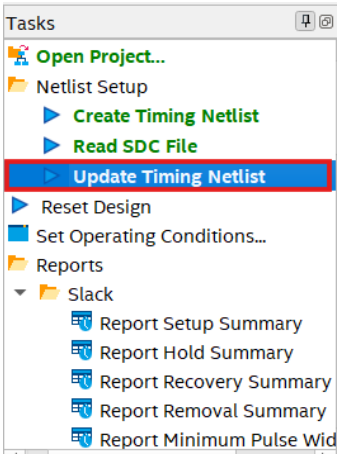


Figure 19: Update the timing netlist

(c) Review the clock constraints from **Report Clock** in the **Task Pane**.

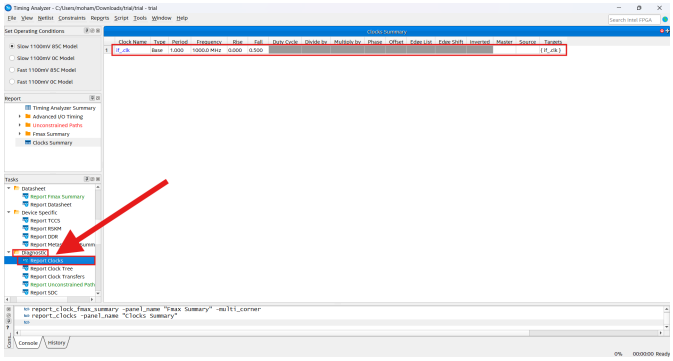
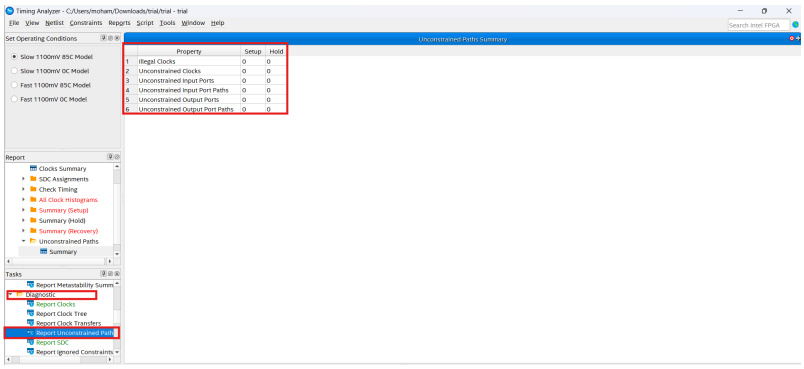


Figure 20: Review the clock constraints

(d) View unconstrained paths: From **task Pin**, select **Report unconstrained paths**. (Since we only set constraints for the clock, all I/O ports are unconstrained.)



(e) **Set I/O delay constraints:** The timing analyzer knows all the internal timing delays, but it does not have any information about timing delays external to the device. Therefore, we need to define the input delay and the output delay. The input delay is the net sum of all external delays, typically consisting of the delay from the common clock source to the external device flip-flop, plus the flip-flop clock-to-out delay, plus the delay in the PCB traces to the FPGA input, minus the delay from the common clock source to the FPGA clock input. To be accurate, you'll need measurements or models of the PCB delay and data sheet information for the external device that drives the inputs. Typically, the input delay is a few nanoseconds, although it can be negative. The maximum delay is usually slightly more than the minimum delay.

Input delay: From the **Constraints** menu, select **Create Clock**, then:

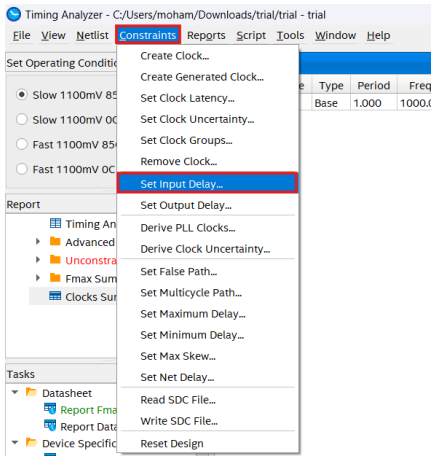
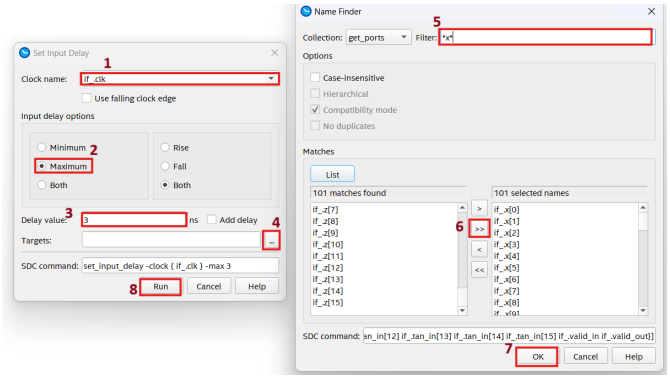
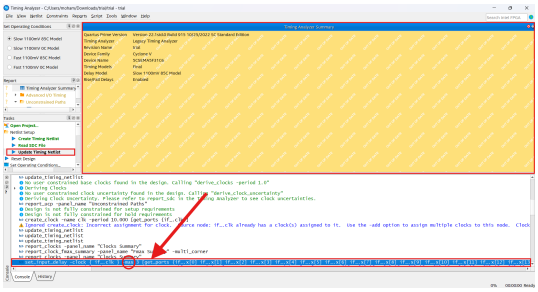


Figure 21: set input delay

i. Select the clock name.
ii. Choose **Maximum** in the **Input Delay Options**.
iii. Enter the delay value.
iv. Select the target port.
v. Choose the port name, then add it. (Repeat this step until all input ports are added.)
vi. After pressing "OK" and "Run," you will see the background turn yellow, indicating that the netlist needs to be updated. Press **Update Timing Netlist**.
vii. Now, repeat the previous steps for the minimum input delay options, or take the last command in the TCL window, change **max** to **min**, and then update the delay value.



(a) set maximum input delay



(b) Tcl command / update netlist

We can repeat the previous steps for the **output delay**. After setting the constraints, you can review them from **SDC Assignments** in the **Report Pin** section.

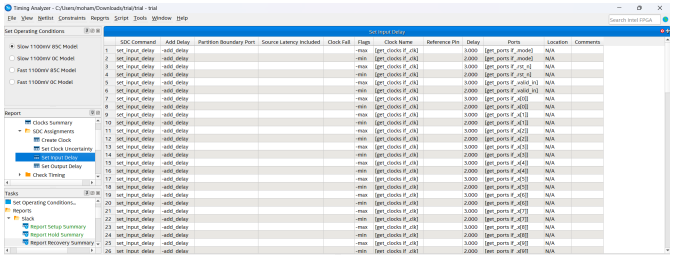


Figure 23: SDC Assignments

4. **Use TimeQuest reports to resolve timing problems:**

(a) **View the report:** All paths should be displayed, indicating whether they are constrained. You should notice that some input paths are unconstrained. In the Report pane, click on **Summary Setup**. The summary setup report shows the negative slack, which indicates a setup violation.

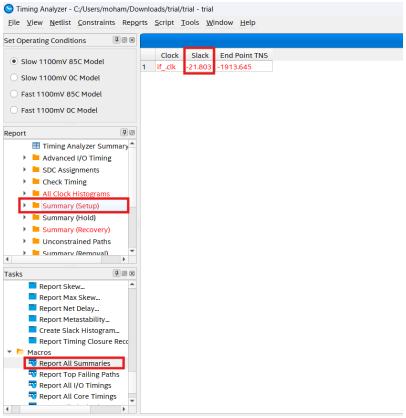


Figure 24: Report all Summaries

(b) **Get the maximum frequency:** From the **Task Pane**, go to **Data Sheet** and double-click on **Report Fmax Summary**. This will show you the maximum frequency according to your design.

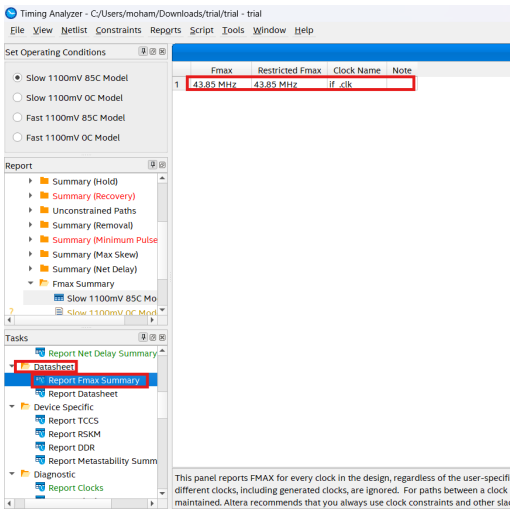
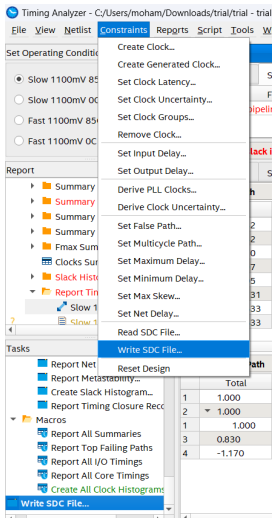


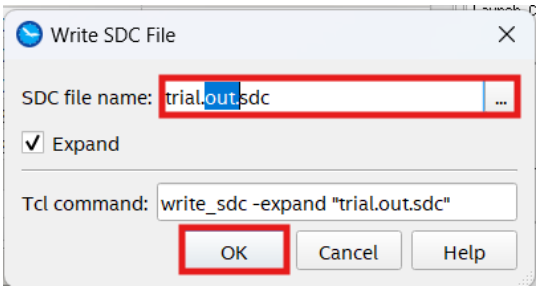
Figure 25: Fmax

5. **Save an SDC constraints file:**
Now that we've finished writing all the constraints, we need to ensure they are reflected during the compilation process by generating an SDC file, which serves as input to the compiler.
Select **Write SDC File** from the **Constraints** menu or the **Task Pane**. When saving the file, remove the **".out"** extension and then press **OK**. After saving the SDC file, close the **Timing Analyzer** and feel free to recompile the project with the new constraints.

   - Note that the name of the constraints file must be the same as the project name because when you run **Timing Analysis**, Quartus executes the commands in the file named "PROJECT_NAME.sdc".



(a) Write SDC File



(b) Change name of SDC file

6. **Read / Re-open SDC file:**
If you close **TimeQuest** and want to open it again to add more constraints, you need to:

   (a) Open **TimeQuest**.

   (b) Select **Read SDC File** in the task pane. When you press it, a command will be executed in the **TCL Console**.

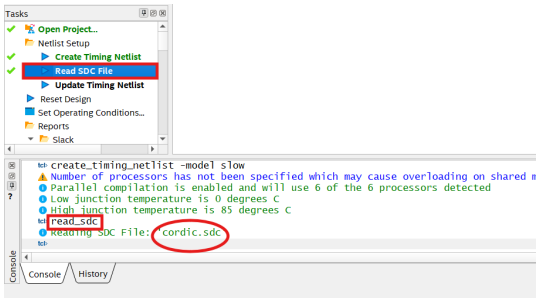


Figure 27: Open constraints file

7. **Adjust the SDC file:**
   If you want to change values such as delay, clock period, delete constraints, or manually add constraints:

   (a) Close **TimeQuest**.

   (b) Browse the project folder for **file.sdc** and open it.

   (c) After finishing adjustments, save the changes and close **file.sdc**.

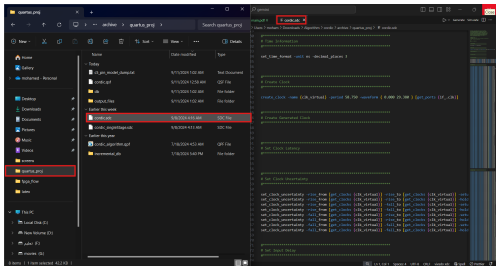


Figure 28: Change in constraints file

2.2.5 EDA Netlist Writer

The EDA Netlist Writer in Quartus generates netlists that are essential for further analysis and verification of designs. It creates netlists outlining the connections and components of the design, supporting formats such as SDF, Verilog, VHDL, and EDIF. These netlists are used by EDA tools for simulation, verification, and optimization. The netlist includes timing and constraint information, aiding in timing analysis and ensuring performance requirements are met. Additionally, it serves as a valuable tool for documentation, debugging, and iterative design improvements.

2.3 Compile Design & Flow Summary

We can run all compilation processes simultaneously by pressing **Ctrl + L** or by double-clicking on **Compile Design**. During the compilation, you can:

- Monitor the progress of each compilation process in the Task window or in the bottom right corner.
- View the results of the running processes in the Message window. If any errors occur, read the messages and try to resolve them.

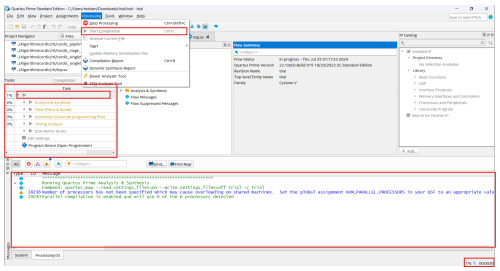


Figure 29: Compile Design

3 RTL Simulation

We need to test our RTL to ensure the correctness of its functionality. Quartus provides the ability to test your RTL. In the following, we will explain the steps to run a simulation in Quartus.

3.1 Add the path of the simulation tool

- From **Tools**, select **Options**.
- Select **EDA Tool Options**, then add the paths of the tool as shown in the figures below:

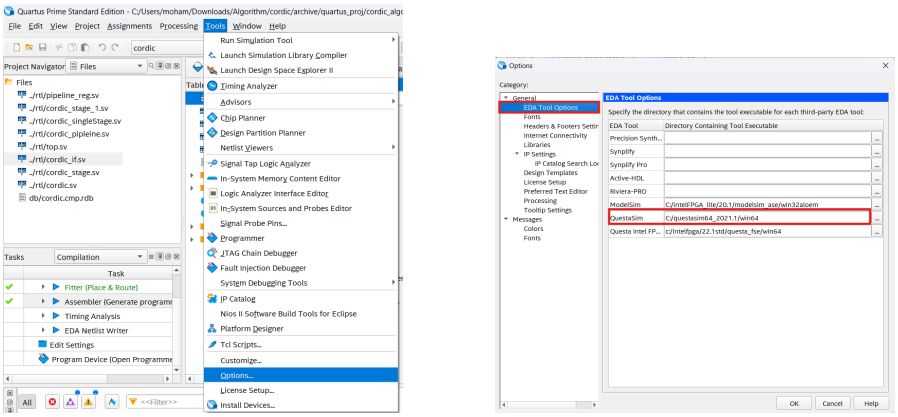


Figure 30: Add tool paths

3.2 Add test bench files and set test settings

- Open **Settings** from **Assignments**.
- For **Category**, select **Simulation**.
- From the **Tool Name** dropdown, choose the simulation tool you want to use.
- Optionally, choose the output netlist formats.
- Select **Compile Test Bench**, then click on **Test Benches**.
- Press **New**.
- Enter the name of the test module (it must match the name in the test file).
- Browse the test files, then press **Add**.
- Press **OK** on all open windows.

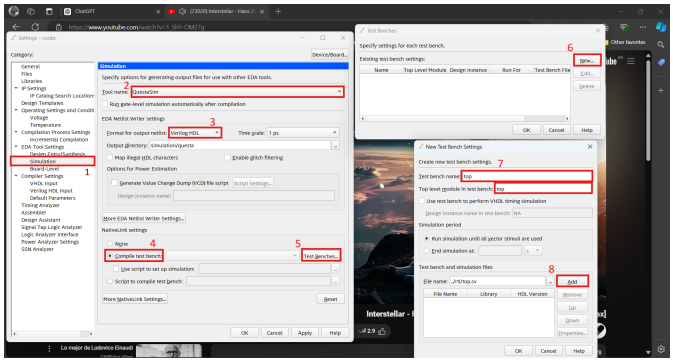


Figure 31: Add test bench files and set test settings

3.3 Run the Simulation

- From **Tools**, select **Run Simulation Tool**, then press **RTL Simulation**.

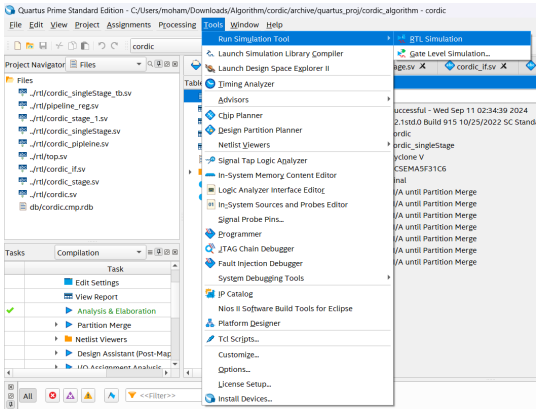


Figure 32: Run Test Bench

Make sure all simulation tools like ModelSim and QuestaSim are closed before running the simulation.If they are not closed, you will encounter an error.

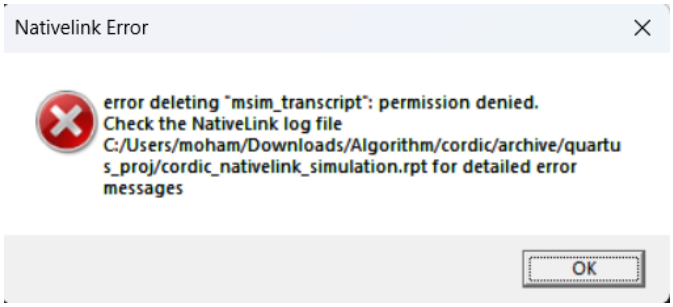


Figure 33: Possible error you may encounter

4 Program Device (Load Design to FPGA)

Programming and configuration are two important concepts that must be understood.

4.1 Programming vs Configuration

1. **Programming**: This is the action of placing a bitstream file into the FPGA's memory. It is not limited to programmable logic devices; it is also used in microcontrollers.

2. **Configuration**: This is the process of loading the memory contents into the device to establish its characteristic behavior.

4.2 Memory Types in FPGAs

FPGAs need to initialize their connections, LUTs, block memory, and registers. These configurations need to be saved in non-volatile memory to retain their contents when powered off. In most cases, flash memory is used for this purpose.

In modern devices, there are two types of memory:

- **Flash memory** (non-volatile): This is used to store configurations permanently, and it retains data even when the power is off.

- **SRAM memory** (volatile): Directly connected to the programmable logic, any change in the SRAM immediately affects the logic. However, changes in flash memory only take effect when the device powers up, as the configuration is loaded into SRAM from flash memory at that time.

4.3 Compilation and Bitstream Files

We can load our design into either type of memory. In safety-critical scenarios, we typically load the design into non-volatile memory, not into flash, to preserve the default configurations stored in the flash memory. This allows us to recover them when the system powers up again.
After writing the design, we test it and then compile it. After compilation, Quartus generates two types of bitstream files:

- **.pof** (Programming Object File): Used to load the design into flash memory.
- **.sof** (SRAM Object File): Used to load the design into SRAM memory (recommended for use).

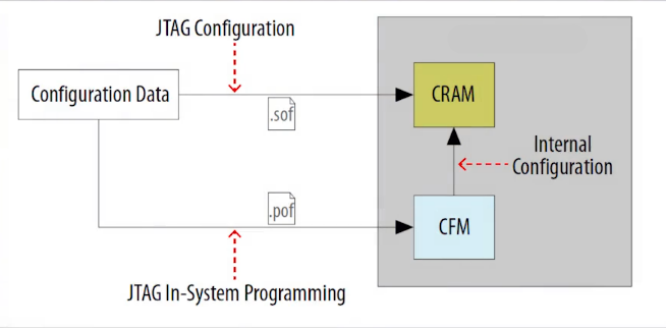


Figure 34: Altera programming system

4.4 Loading the Design

1. Connect the FPGA board to power and the PC.
2. Double-click on **Program Device**, then press **Hardware Setup**. In the "Currently selected hardware" section, choose the interface between the PC and the FPGA, then close the window.

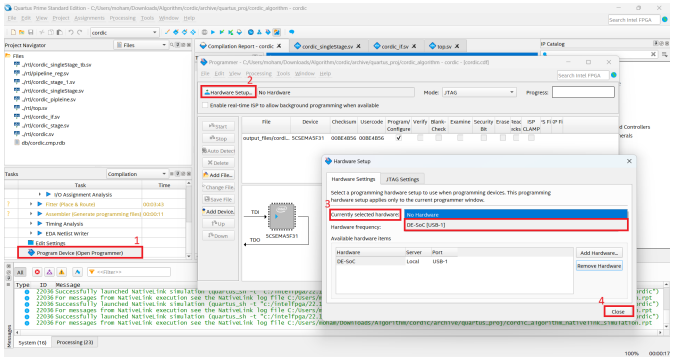


Figure 35: Select Hardware

3. Press **Auto Detect**, then select the first device and press "OK".

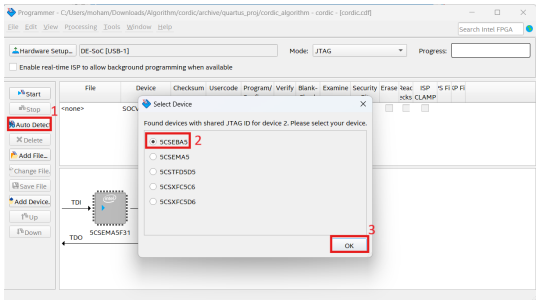


Figure 36: Select Device

4. Right-click on the added device, then from the "Edit" menu, select **Change File**.

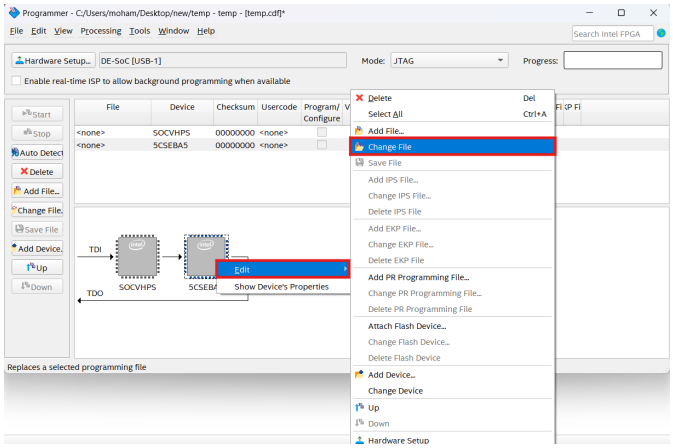


Figure 37: Add file.sof

5. Browse the project directory, navigate to the `./output_files` folder, and choose `project_name.sof`.

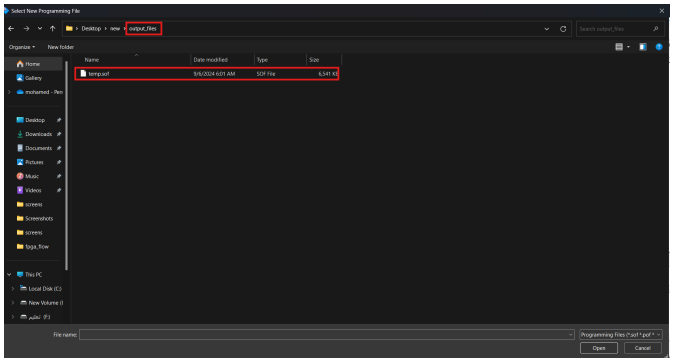


Figure 38: Select .sof file

6. Check the box next to the selected file, then press **Start**.

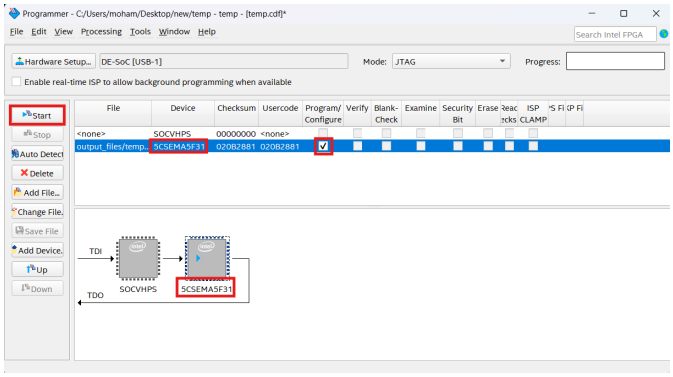


Figure 39: Start Programming

5 Feedback

If you find any errors or have any questions regarding this document, please feel free to get in touch with me. My contact information is provided on the cover page. I would appreciate any feedback or corrections to ensure the accuracy and clarity of the content.