

ECE 4156/6156 Hardware-Oriented Security and Trust

Spring 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Pre Lab 1, 10 pts.

Due **Wednesday**, January 22 prior to 11:55pm

Lab 1, 90 pts.

Due Friday, January 24 prior to 11:55pm

(Please turn in homework electronically on Canvas)

Prelab

The prelab sets up Quartus and ModelSim on your computer. While this document is long, the vast majority is simply installing Quartus and troubleshooting problems that may pop up in the installation. Then you would have to simulate a small VHDL program which is given by us to check that Quartus and ModelSim are working as expected.

Installation and Use of the Quartus Prime software

This first section prepares you to begin actual labs using the DE-10 Standard board, but it still requires a little bit of time. It is a tutorial to familiarize you with the installation and basic functionality of the Quartus software. The installation will be done in the Prelab 1 step, and this will require a computer running Windows or Linux. Then a tutorial of creating a VHDL file and implementing it on the development board will be done in the prelab steps.

This tutorial has been written using Quartus Prime version 19.1, but it is applicable to earlier versions, as far back as version 15, or more recent versions of Quartus Prime. This document is derived from ECE 2031 Lab 0, so you may be able to skip some of the install steps if your computer is already set-up for use with the DE-10 Lite board.

NOTE: ECE 2031 has used a different FPGA (Field Programmable Gate Array) device than the DE-10 Lite board in the past, so you may need to install Cyclone V device support for the DE-10 specifically.

Documentation and resources for the DE-10 Standard board can be found at:

<https://rocketboards.org/foswiki/Documentation/DE10Standard>

Lab 1 Prelab Steps

This course will make extensive use of a computer-aided design (CAD) application called Quartus Prime, originally developed by the Altera Corporation, which is now a division of Intel. Quartus Prime will allow you to "capture" your designs with schematics and other means and fit the designs to be implemented on the FPGA board used as the primary development platform. Even when we build circuits with discrete integrated circuits, Quartus is a convenient way to draw schematics.

Requirements to begin

You must have a computer meeting one of the following general descriptions:

- A Windows 10* PC
- A PC running one of the following Linux distributions (although it is likely others will work)
 - Red Hat Enterprise Linux 6
 - Red Hat Enterprise Linux 7
 - SUSE SLE 12
 - Ubuntu LTS (at least 14.04)
- A Mac with one of the following additional operating systems installed
 - Windows 10* in a Boot Camp hard disk partition
 - Windows 10* in a Parallels VM (Virtual Memory)
 - Windows 10* in some other VM (e.g., VMware)
 - One of the Linux variants above, in some VM

* Windows 7 and Windows 8 still seem to work fine but are not recommended. Windows 11 should work fine also.

Although all these configurations SHOULD be supported, the only ones being actively tested by the instructors are specifically:

- A Windows 10 PC
- Debian 11 PC

You will know by the end of Prelab if you have any issues with your computer.

In all configurations, the following also apply:

- You need a functional USB port, with a full-size [USB Type A receptacle](#), or some adapter, such as the USB-C to USB-A adapter needed for recent MacBooks ([one example, but there are others](#))

- For the limited number of users who may still be running a 32-bit version of their operating system, you may have to upgrade to the 64-bit version of the operating system.
- You will need at least 15 GB of disk space for the Quartus installation.
- You must have administrative rights, allowing you to install software on the system. To check in Windows, open Settings (Gear icon), select Accounts, and next to your account name, it should show *Administrator*

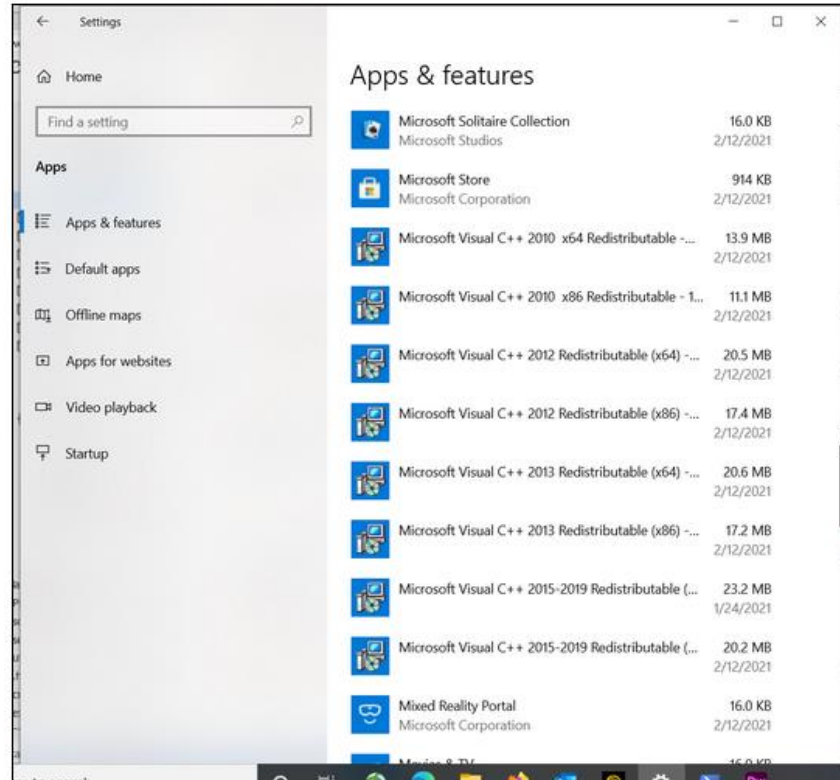
Once you have a computer at hand meeting these requirements, begin following the installation steps below. Most of them should be similar for all computer configurations if you are doing them within the Windows or Linux operating system on your computer. This first step is the main exception since it is not required for Linux users.

Step 1.

Linux users should skip this step.

On any Windows system (including Windows running on a Mac VM or Bootcamp), the Quartus installation will require the Visual C++ Runtime system, officially the *Microsoft Visual C++ Redistributable*. Your system *probably* has it already, but it will be worthwhile to check now. There are two ways to check:

Option A) Open your Windows Settings (Gear icon), go to Apps & Features, and look for any entries beginning with *Microsoft Visual C++ Redistributable....* If you have one or two ending with a date that is 2015 or later, you should be fine, and you can skip Step 2 and go to Step 3 below.



Step 2.

(Optional, only if required based on the result of Step 1.) Go to the URL below, then download and run `vc_redist.x64`

- <https://www.microsoft.com/en-us/download/details.aspx?id=48145>

Step 3.

We have found that the following website does not work properly with Chrome or Firefox, so if you have trouble downloading files in the steps that follow, try Edge or other browsers.

In a browser **under Windows or Linux**, go to the download site:

<https://fpgasoftware.intel.com/19.1/?edition=lite&platform=windows>. The top of the resulting page will resemble the image below. As shown here,

- select the Quartus Prime Lite edition at the top right,
- select release 19.1 or 20.1 (you may try newer versions but they have not been validated by us and the functionality may differ from what is presented in this document), and
- choose Windows (or Linux, if it applies to you).

It does not matter if you are on a Mac — you should not be in MacOS at this point, so select the Windows version of Quartus.

intel PRODUCTS SUPPORT SOLUTIONS DEVELOPERS PARTNERS FOUNDRY ENGLISH Search Intel.com

Documentation & Resources / FPGA Software Download Center

FPGA Software Download Center

[Quartus® Prime Pro](#)
[Quartus® Prime Standard](#)
[Quartus® Prime Lite](#)

Search this collection 29 Results

Filter by

EXPAND ALL | COLLAPSE ALL Featured

Intel Quartus Software ^
 CLEAR
 Intel® Quartus® Prime Design Software (22)
 Quartus® II Software (6)

Additional Software v
 Intel FPGA Device Family v
 Operating System v
 Content Type v
 Last Updated v

There may be additional results restricted from public access; sign in or register to ensure you are seeing all content available to you.

Title	ID	Date	Version		
Featured Intel® Quartus® Prime Standard Edition Design Software Version 23.1 for Linux	661483	12/10/23	23.1		
Featured Intel® Quartus® Prime Lite Edition Design Software Version 23.1 for Linux	661454	12/10/23	23.1		
Featured Intel® Quartus® Prime Lite Edition Design Software Version 23.1 for Windows	661455	12/10/23	23.1		

Step 4.

Click the Download button to download all required files.

Intel® Quartus® Prime Lite Edition Design Software Version 20.1 for Windows

ID	Date	Software Type	Software Package	Version	Operating Systems
661019	6/14/2020	FPGA Development To	Quartus® Prime Lite	20.1	Windows

A newer version of this software is available, which includes functional and security updates. Customers should [click here](#) to update to the latest version.

Users should upgrade to the latest version of the Intel® Quartus® Prime Design Software. The selected version does not include the latest functional and security updates. If you must use this version of software, follow the [technical recommendations](#) to help improve security. For critical support requests, please contact our [support team](#).

The Intel® Quartus® Prime Lite Edition Design Software, Version 20.1 is subject to removal from the web when support for all devices in this release are available in a newer version, or all devices supported by this version are obsolete. If you would like to receive customer notifications by e-mail, please subscribe to our [subscribe to our customer notification mailing list](#).

[Critical Issues and Patches for the Intel® Quartus® Prime Lite Edition Software, Version 20.1.](#)
[Knowledge Base: Search for Errata.](#) Also see [Critical Issues and Patches.](#)
[Problems and Answers on specific IP or Products.](#)

Downloads

[Multiple Download](#) | [Individual Files](#) | [Additional Software](#) | [Copyleft Licensed Source](#)

Multiple Download

Intel® Quartus® Prime Lite Edition Software (Device support included) ▼

<div style="background-color: #0070c0; color: white; padding: 5px; display: inline-block; border-radius: 3px;">Download</div> Quartus-lite-20.1.0.711-windows.tar	<p>Size: 5.9 GB</p> <p>SHA1: 101faf57b86e4737f40379339b9b7eed4dc8672d ▼</p>
--	--

What's Included?

** Nios® II EDS on Windows requires Ubuntu 18.04 LTS on Windows Subsystem for Linux (WSL), which requires a manual installation.

** Nios® II EDS requires you to install an Eclipse IDE manually.

Step 5.

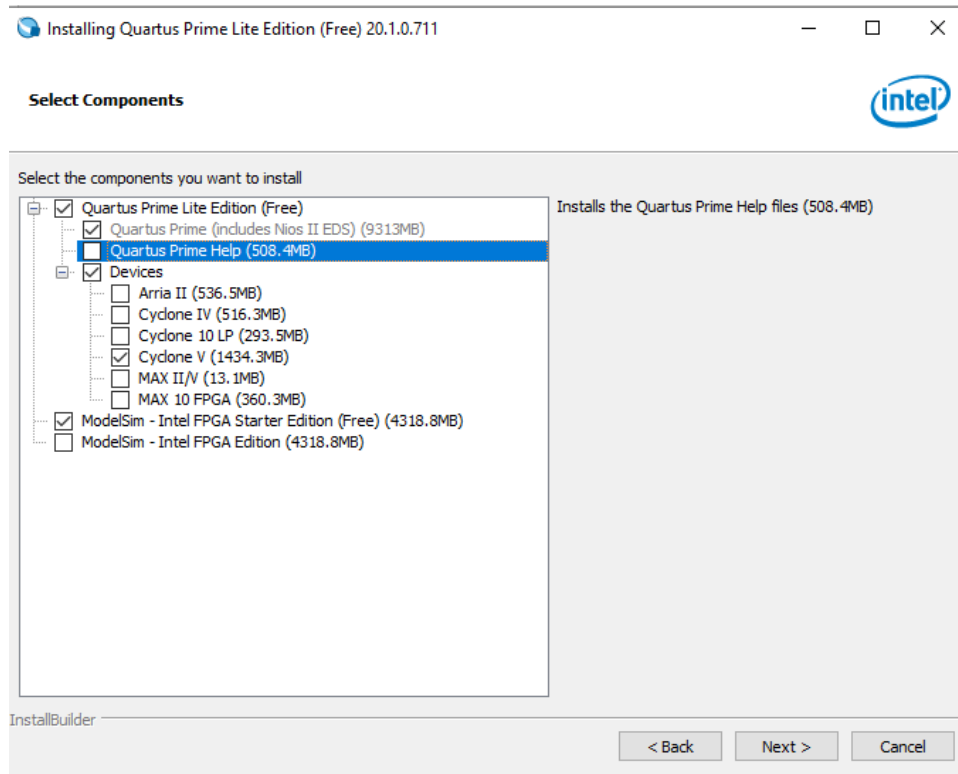
After downloading the file, extract the content of the .tar file. You should have the following files inside a “components” folder (plus some extra device supports we do not actually need).

- "QuartusLiteSetup-..."
- "ModelSimSetup-..."
- "cyclonev-....qdz"
- "QuartusHelpSetup-..." — you can delete this, if you prefer to save a little space.

The most common installation error is that one or more files is missing or wrong, and it can result in having to start over. Verify that you have the files shown here (for whichever version you downloaded):

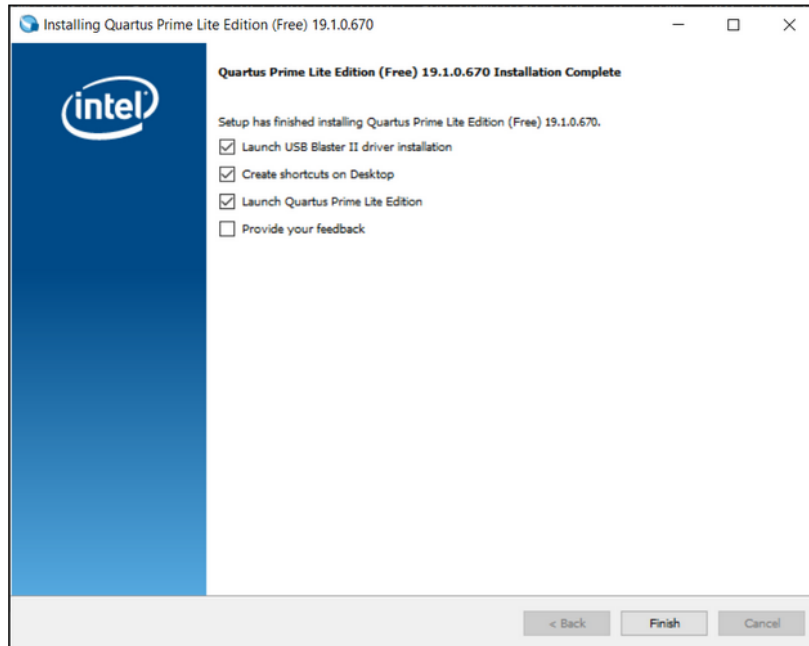
File Name	Date	Type	Size
arria_lite-20.1.0.711.qdz	6/6/2020 6:15 AM	QDZ File	511,095 KB
cyclone10lp-20.1.0.711.qdz	6/6/2020 6:13 AM	QDZ File	272,083 KB
cyclone-20.1.0.711.qdz	6/6/2020 6:15 AM	QDZ File	477,157 KB
cyclonev-20.1.0.711.qdz	6/6/2020 6:13 AM	QDZ File	1,412,205 KB
max10-20.1.0.711.qdz	6/6/2020 6:12 AM	QDZ File	292,209 KB
max-20.1.0.711.qdz	6/6/2020 6:13 AM	QDZ File	11,644 KB
ModelSimSetup-20.1.0.711-windows.exe	6/6/2020 9:33 PM	Application	1,216,188 KB
QuartusHelpSetup-20.1.0.711-windows.exe	6/6/2020 9:14 PM	Application	282,197 KB
QuartusLiteSetup-20.1.0.711-windows.exe	6/6/2020 10:08 PM	Application	1,710,821 KB

Still in that folder, run the *QuartusLiteSetup...* executable, **not** the *QuartusHelpSetup* or the *ModelSimSetup*. Follow all the instructions to complete the installation. It is recommended that you accept the default installation folder, usually *C:\intelFPGA_lite\19.1*. One step will be to confirm that you want to install the other items that you downloaded into the folder. The example below shows the items that you must install. It also shows that you should NOT install any edition of ModelSim other than the "Starter Edition," so if another one is there, leave it unchecked.

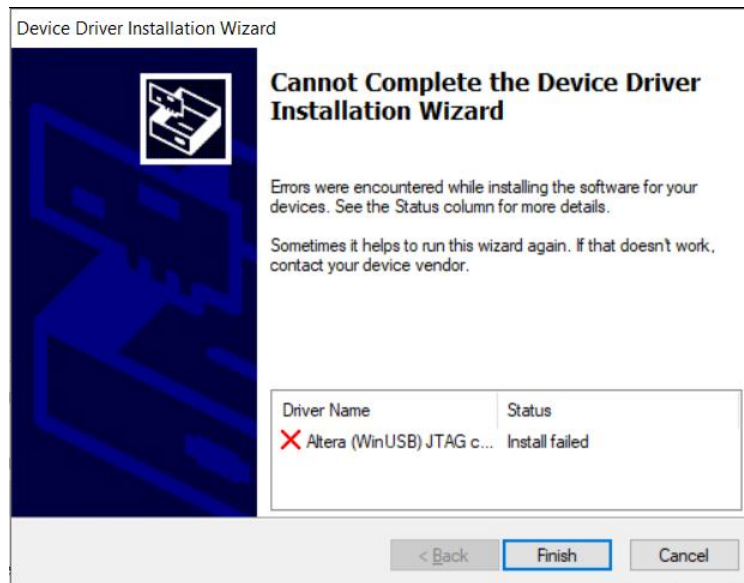


Step 6.

The installation can take a while. When it nears the end, the following window appears. Leave these three items checked, or uncheck the option for creating shortcuts, if desired. But make sure that the option to Launch USB Blaster II driver is checked.



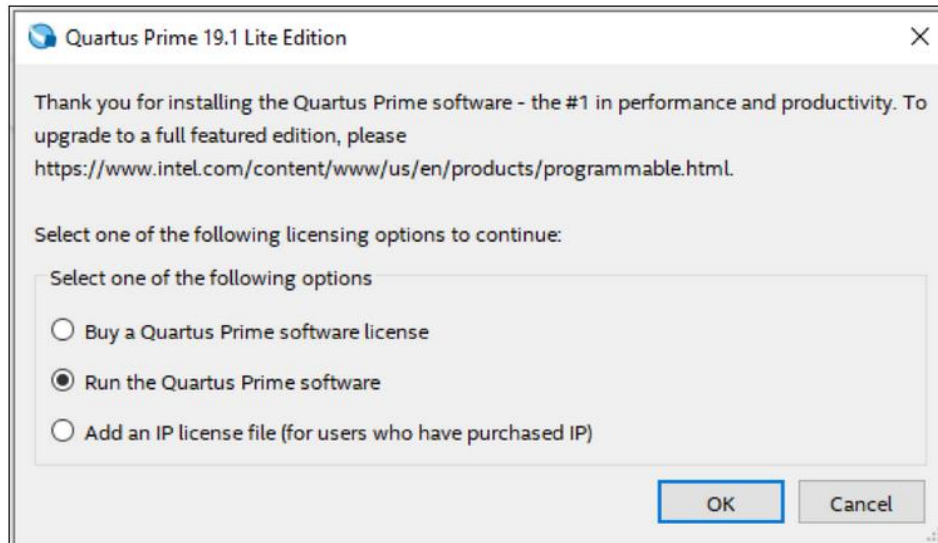
You will be presented with the Device Driver Installation Wizard. Make sure you click Next to continue. It is very possible that this installation will fail, giving the response below.



If you get this error, it will affect what you do in a later step. Consider yourself in the group of users who need to follow the steps to manually install the USB-Blaster, in a step coming up very soon.

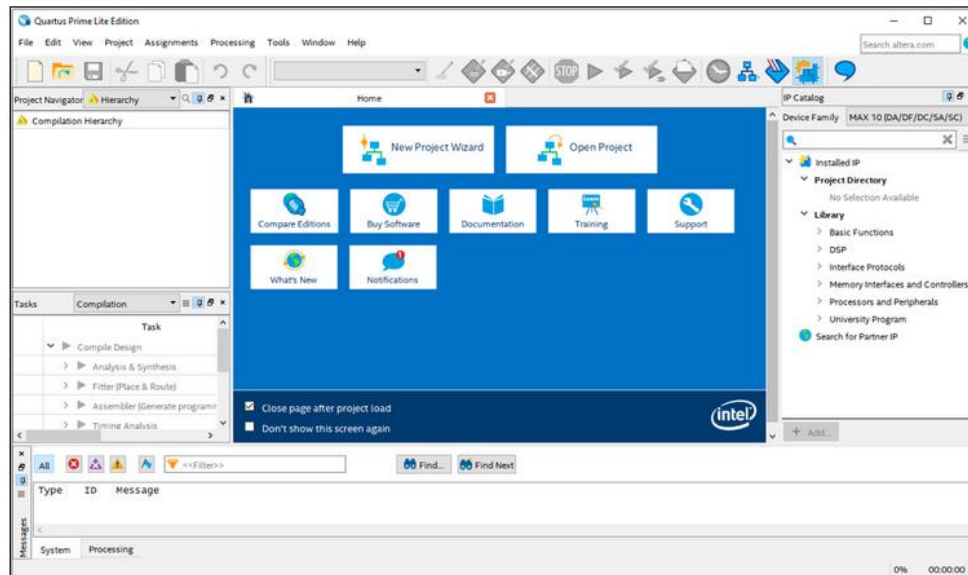
Step 7.

As it finishes the installation, you will be presented with the option below. At this time, and at any future time where it may ask, select "Run the Quartus Prime software." No purchase is necessary for this class.



Step 8.

When Quartus opens, it will appear similar to the image below.



You can delete the installation files from the temporary folder where you launched them.
<https://www.microsoft.com/en-us/download/details.aspx?id=48145>

Step 9.

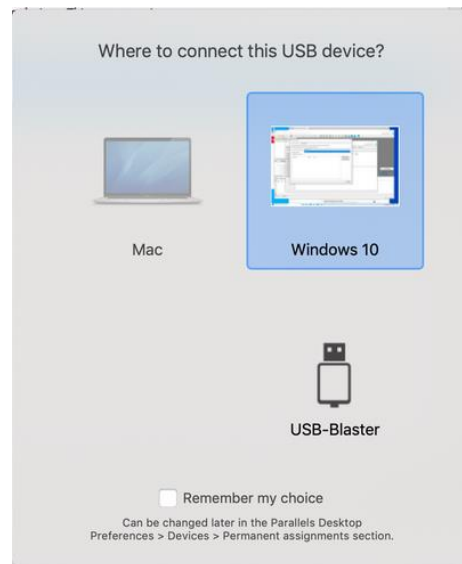
If, only a couple steps back, you received an error when installing the USB-Blaster driver, then do not proceed with any further steps until you first complete the steps in a separate section below, titled *Manual USB-Blaster Installation*.

Return to the next step, when finished.

Step 10.

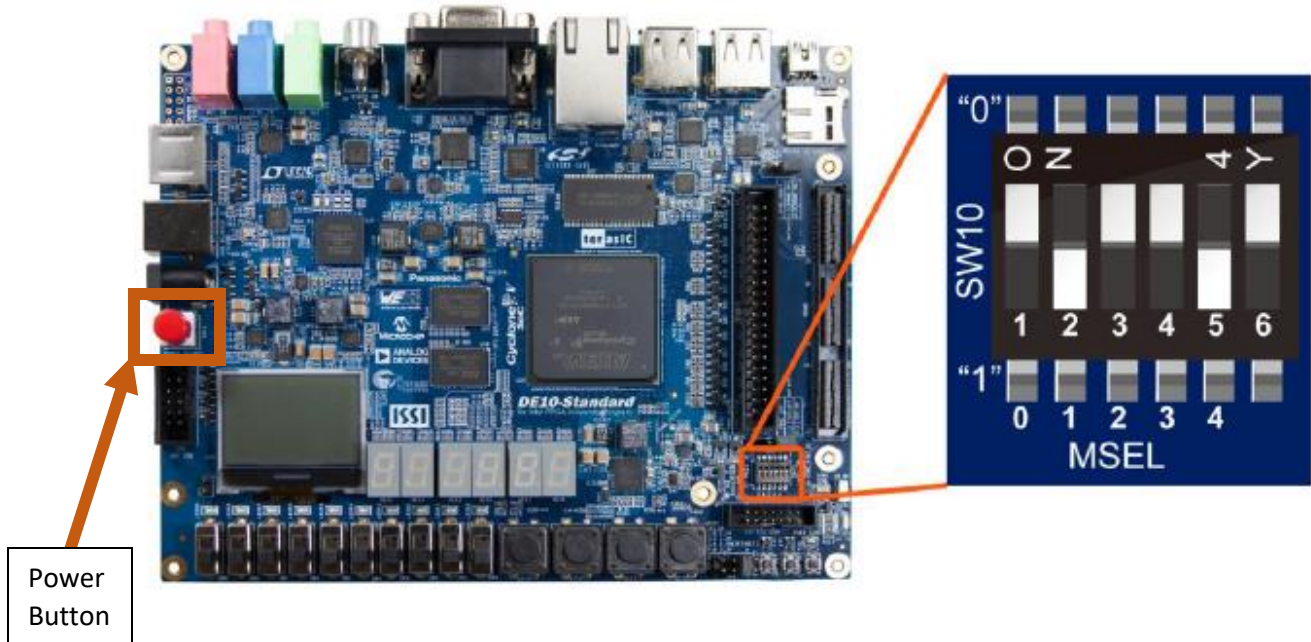
There is one additional thing to check at this time if you have already received your DE10-Standard FPGA development board. If not, this will happen near the end of the steps.

Connect your DE10-Standard board to your computer with the supplied white/gray USB cable. If you have Windows running in a VM on a Mac, you may get a prompt asking you if the newly detected device should be connected to the Mac or to Windows 10. Choose Windows, if you have this prompt.



The DE10-Standard should power up and immediately begin running the default programming file that is stored in non-volatile memory. We are going to verify that you can reprogram it through your cable.

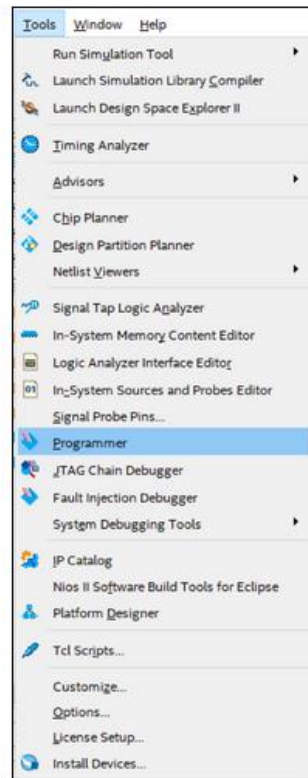
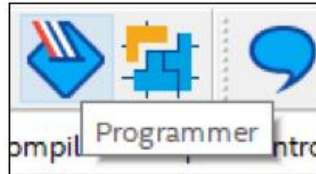
If it does not turn on immediately, check the following:



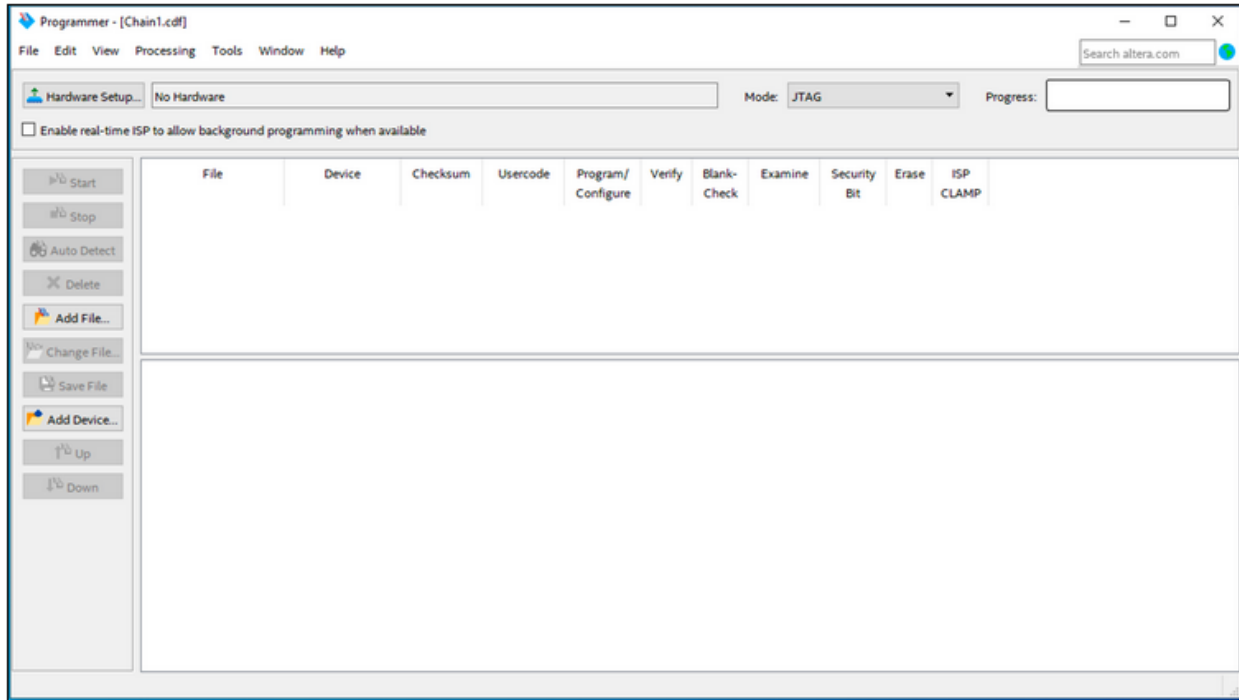
Ensure the MSEL switches are in the positions shown above and the red power button is pushed to the "down" position.

Step 11.

In Quartus, find the Programmer by either method below. Either look for the icon at the top (the one on the left of the three shown below) or the menu item under Tools, also shown below. Click on it to open



The Programmer will come up and appear similar to the image below. In particular, it will say "No Hardware" near the top, just to the right of "Hardware Setup," because it has never been configured to use the USB connection, called the *USB-Blaster*. Go ahead and click on "Hardware Setup."



Step 12.

The "Hardware Setup" window is shown below. In the "Available hardware items," if "USB-Blaster" appears, then you are done. But usually, you need to click on "Add Hardware," and in the window that comes up, select USB-Blaster (or EthernetBlaster if only that shows up). If you can do that successfully, then you will also be done. If not,

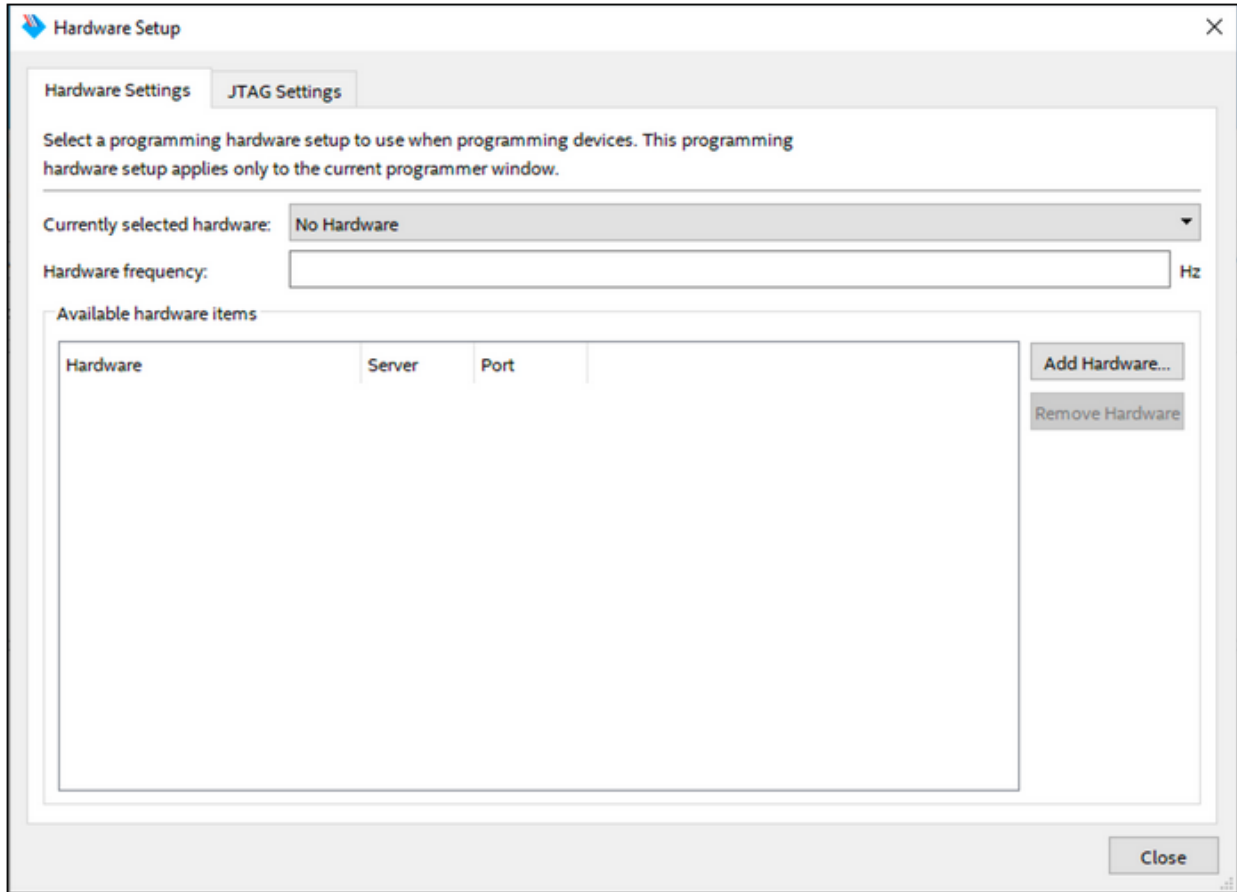
A) First make sure that the USB cable is plugged in **completely** to your computer and to the board. It is possible for it to supply power, yet not be connected properly. Then try again.

B) Should that fail, continue with the next step.

Linux users will need to take some additional steps, even if USB-Blaster is present, because of USB access permissions.

The best description appears to be found here: <https://blog.atomminer.com/fighting-altera-usb-blaster-on-ubuntu/>

Some students in summer reported that the script provided there did not work, and someone updated it to work for their setup, so you might have better luck with this script: usbblaster.sh.



Step 13.

If the previous step was unsuccessful, and you still are unable to make the USB-Blaster appear, contact the instructors for assistance. There may be a discussion thread in Canvas on this topic, if needed.

Step 14.

Continue with the Lab Steps for the prelab.

Manual USB-Blaster Installation

If you were able to use your USB-Blaster the first time you needed it to program your FPGA board, you should not need to follow any of the steps below. But if you saw errors near the end of the installation of Quartus Prime, indicating that NO device drivers were installed successfully, continue here. The problem was probably that Windows would not install the unsigned device driver provided by Intel.

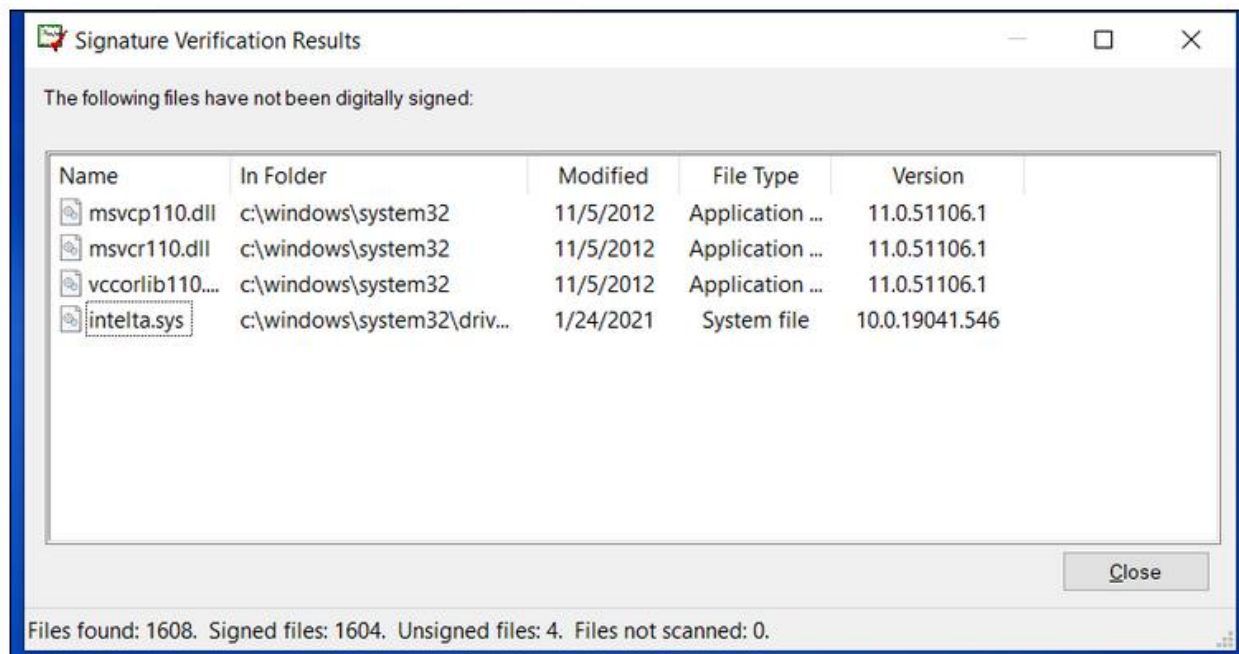
Requirements to begin

You must have successfully installed Quartus (except possibly the USB-Blaster device driver) on a Windows machine, either a PC, or Windows running on a Mac.

Step 1.

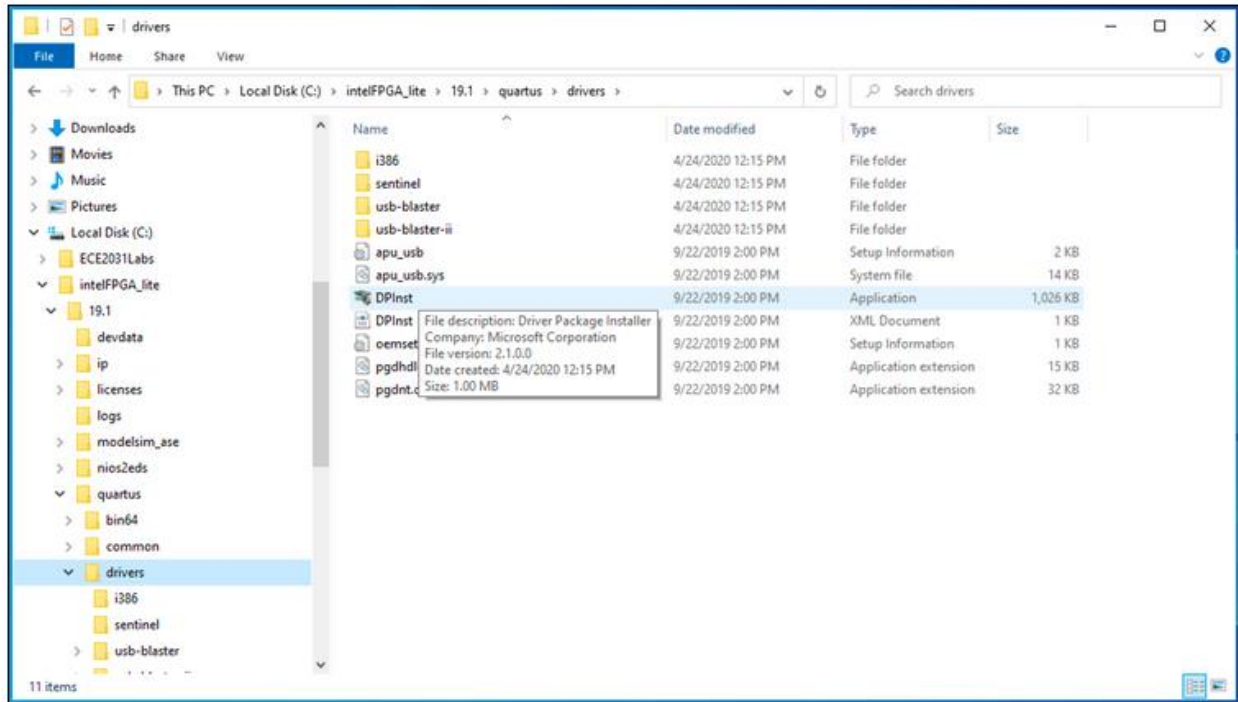
None of this is relevant to Linux users. If you have trouble programming the board on Linux, we will try to help, but we have very limited experience using Quartus on Linux.

First, check to see if the driver possibly DID install correctly. In the text entry area of the Windows Start Menu, type `sigverif` and press return. It will take a while to find all device drivers that are unsigned, but when it is done, see if your computer is missing the file name `intelta.sys`. The image below shows the result in a system that DOES have the driver installed correctly, along with three other unrelated drivers.

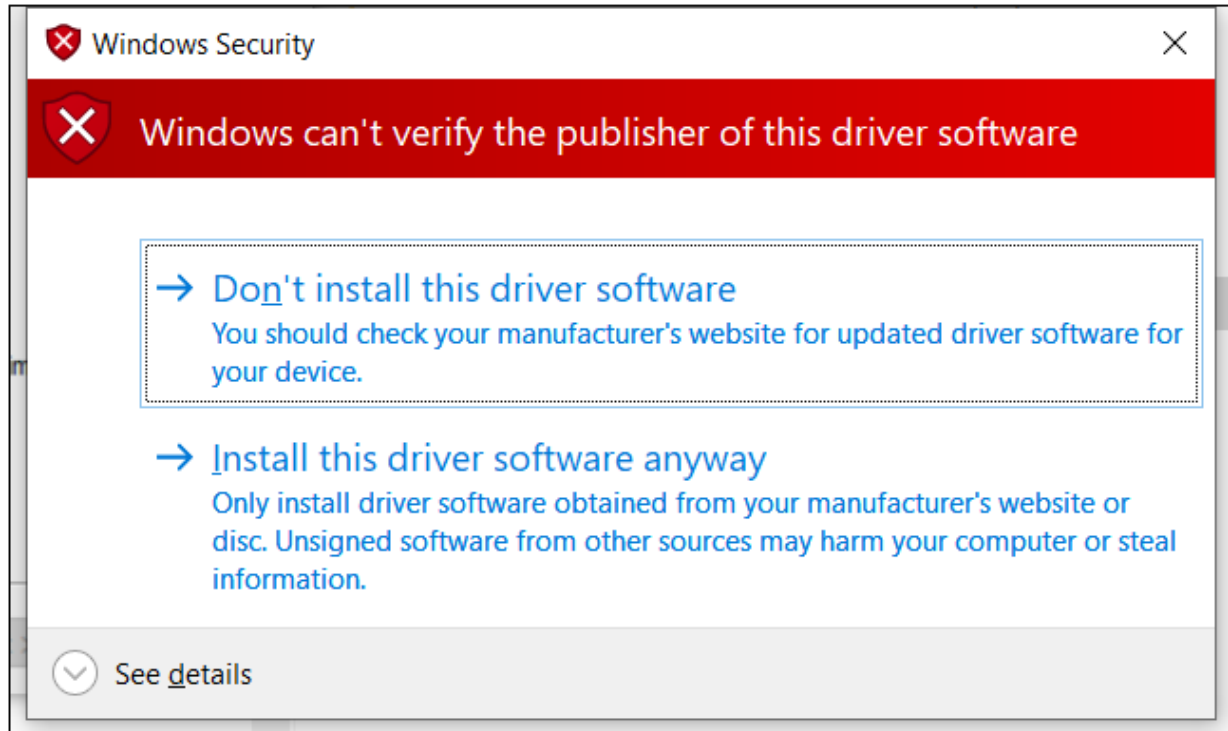


Step 2.

Assuming your system does **not** have the driver, open Windows File Explorer, and navigate to the folder *19.1/quartus/drivers*, within the folder that you used to install Quartus (which defaults to */intelFPGA_lite* on the system drive). Below, the file *DPinst* (the application, not the XML document) is highlighted.



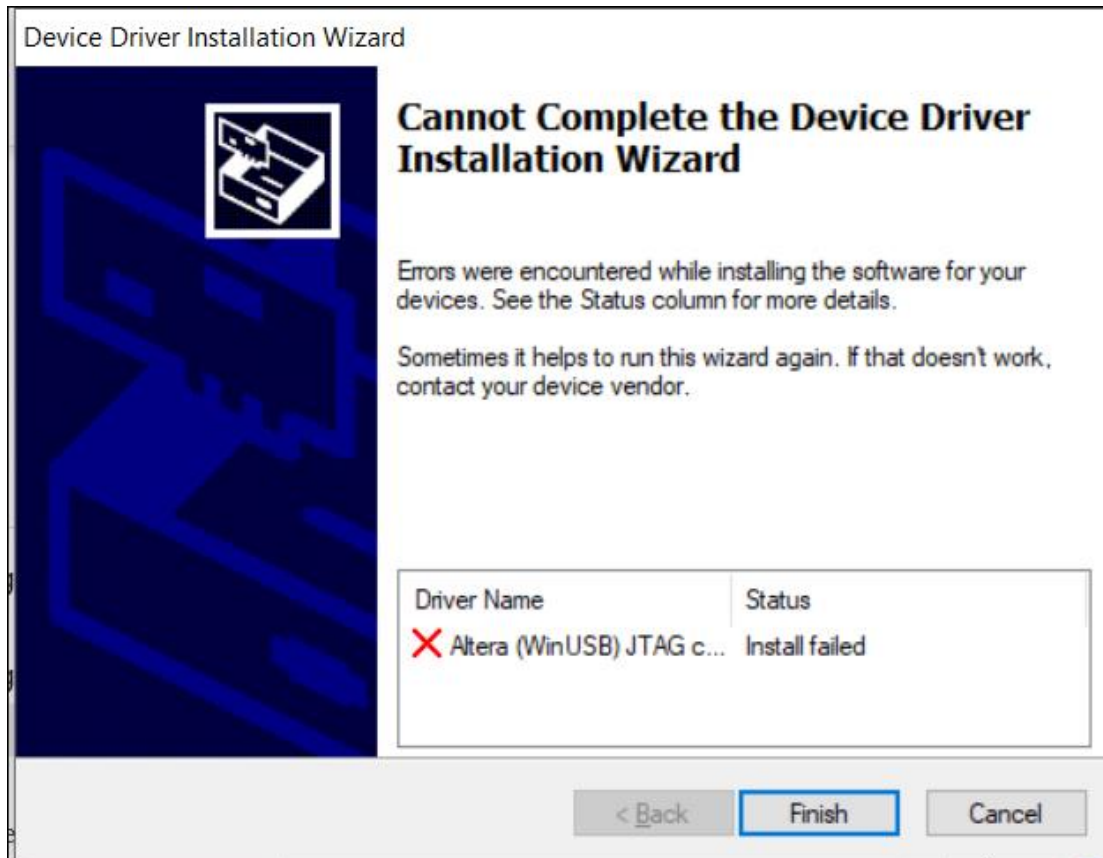
Double-click on that application and continue with the installation. Ignore warnings like the one below and select "Install this driver software anyway".



When it is finished, it may show that some items were not installed. But if the Altera USB-Blaster Device Driver was installed, as shown below, then this part was successful.

Step 3.

If the previous step was unsuccessful, you probably saw a message similar to the one below, or one that showed multiple devices, with none installed successfully.

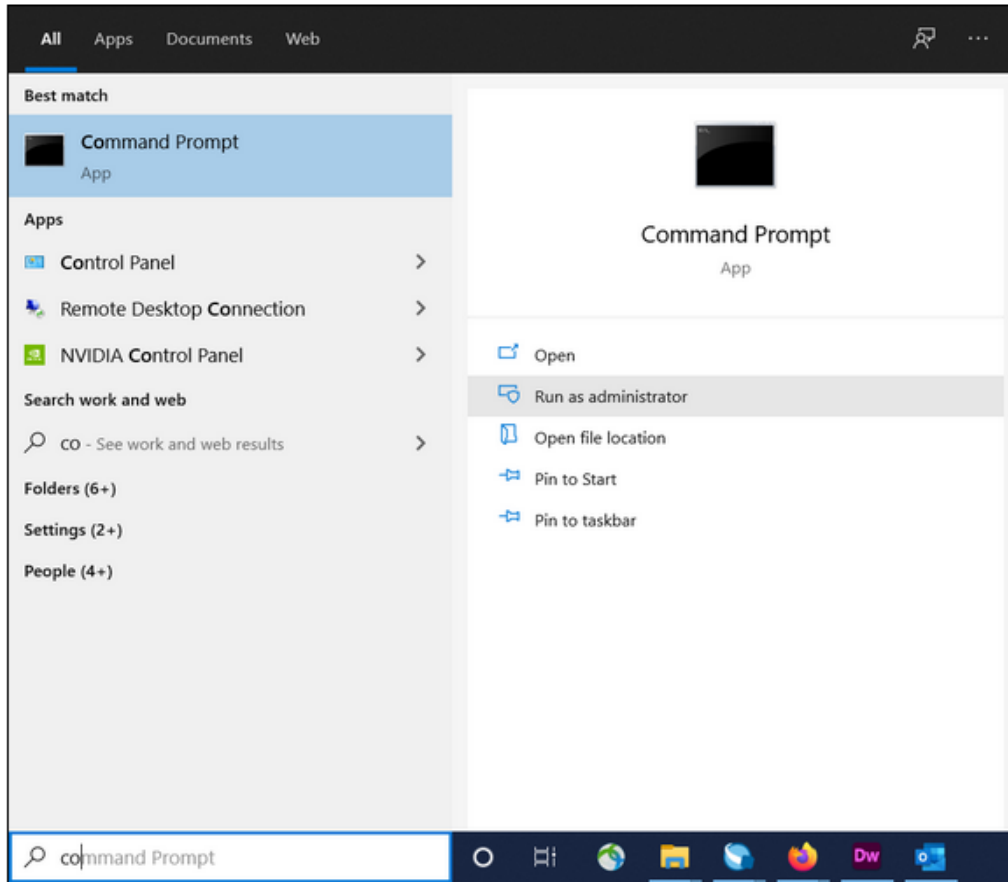


The only known cause for this is that your Windows system is set to not allow unsigned device drivers (those not tested by Microsoft) to be installed. It can be complicated to override this setting, but first we will try the easy way.

Step 4.

First, launch an Administrator Command Prompt by

- clicking the Start button, then
- typing at least the first few letters of "Command Prompt," and
- clicking on "Run as administrator" in the Command Prompt window that comes up, as shown below.

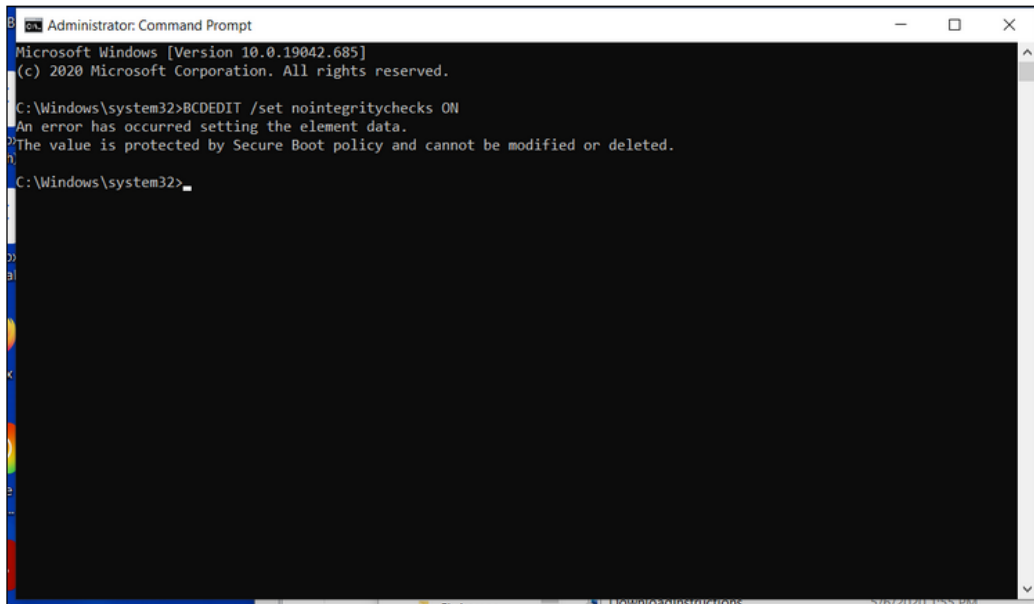


Step 5.

In the Command Prompt, type the following command:

```
BCDEDIT /set nointegritychecks ON
```

If you get an error related to the Secure Boot Policy, as shown below, continue with the next step. If you do **not** get this error, then you can go back to Step 2 above and see if it succeeds. But if it does not, continue to the next step.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>BCDEDIT /set nointegritychecks ON
An error has occurred setting the element data.
The value is protected by Secure Boot policy and cannot be modified or deleted.

C:\Windows\system32>
```

Step 6.

If you have not succeeded yet, then we have the more complicated situation where we must reboot the system in a mode that will allow installation of unsigned drivers. Begin by printing the instructions that follow, or making some notes, or opening them on a different device, because you will need to reboot your computer.

If your computer has Bitlocker enabled, **you will need to know your Bitlocker recovery key**. If you do not have it saved somewhere, do a search on "recover bitlocker" for tips (and consider saving it in whatever you use for password management in the future).

You need to get the advanced boot options menu, and the first step is to hold down the Shift key while you click the "Restart" option in Windows. Your computer will restart into the menu below. Choose "Troubleshoot."

Choose an option



Continue

Exit and continue to Windows 10



Turn off your PC



Use a device

Use a USB drive, network connection, or Windows recovery DVD



Troubleshoot

Reset your PC or see advanced options

In the Troubleshoot menu, as shown below, select "Advanced Options."

← Troubleshoot



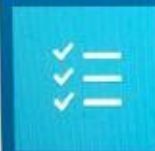
Reset this PC

Lets you choose to keep or remove your personal files, and then reinstalls Windows.



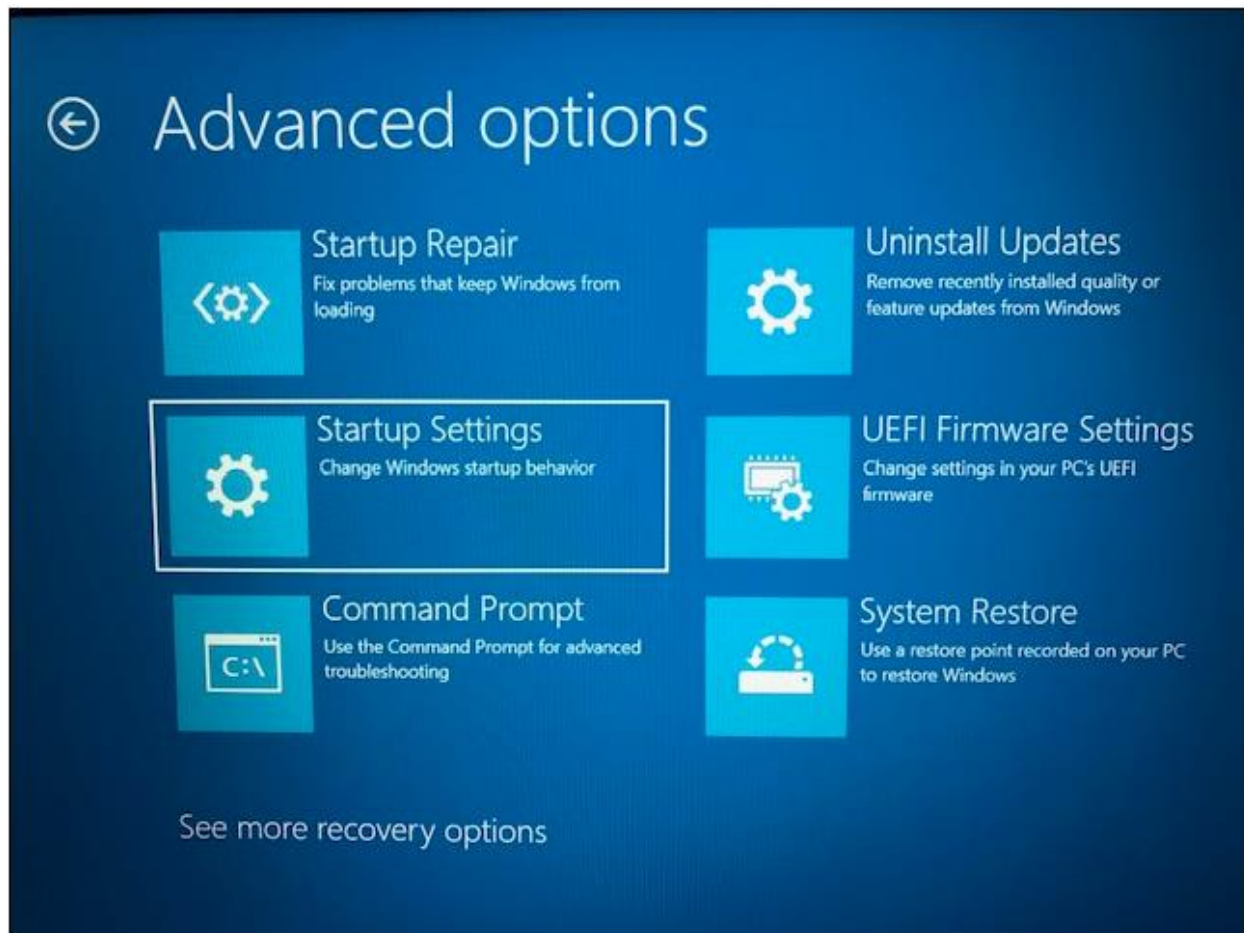
Factory Image Restore

Restore your system software to a saved system image.



Advanced options

In the Advanced Options menu, as shown below, select "Startup Settings."

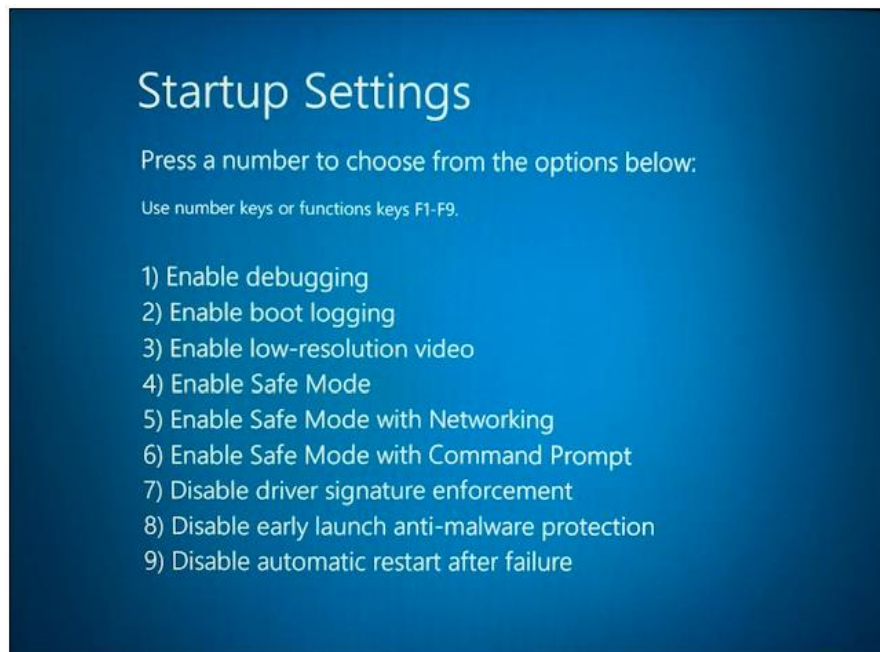


Clicking "Restart" on the following screen (shown below) will result in a reboot. But before clicking it, note two things:

- If your computer has Bitlocker enabled, this is where you will need to know your Bitlocker recovery key.
- You will need to do the driver installation on this next reboot. If you boot the system again, you will lose the ability to install unsigned drivers, and you will have to go back to restarting while holding the shift key, then going through the remaining menus.



Once the computer restarts, you will be presented with the following menu. Select option 7, "Disable driver signature enforcement".

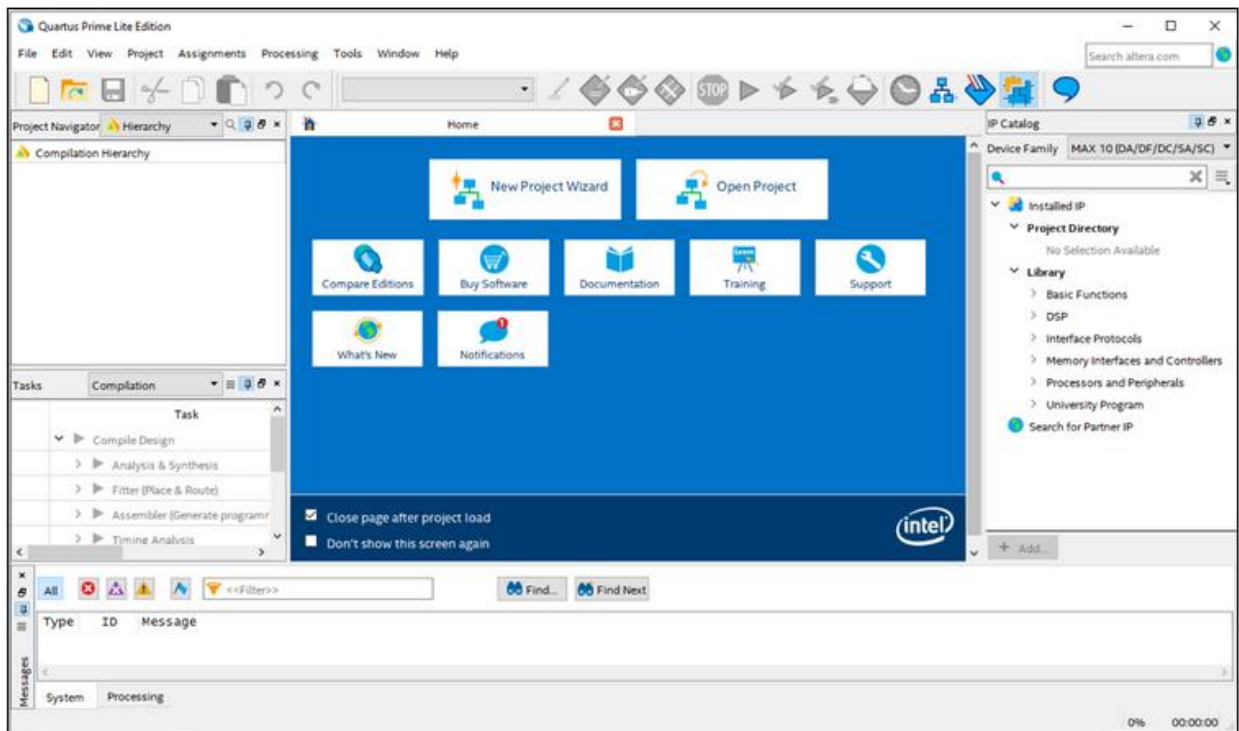


Once back in Windows, return to step 2 above and run DPlnst.exe, which should now succeed. If not, contact the instructors, because this is the last process, we have found necessary for anyone so far.

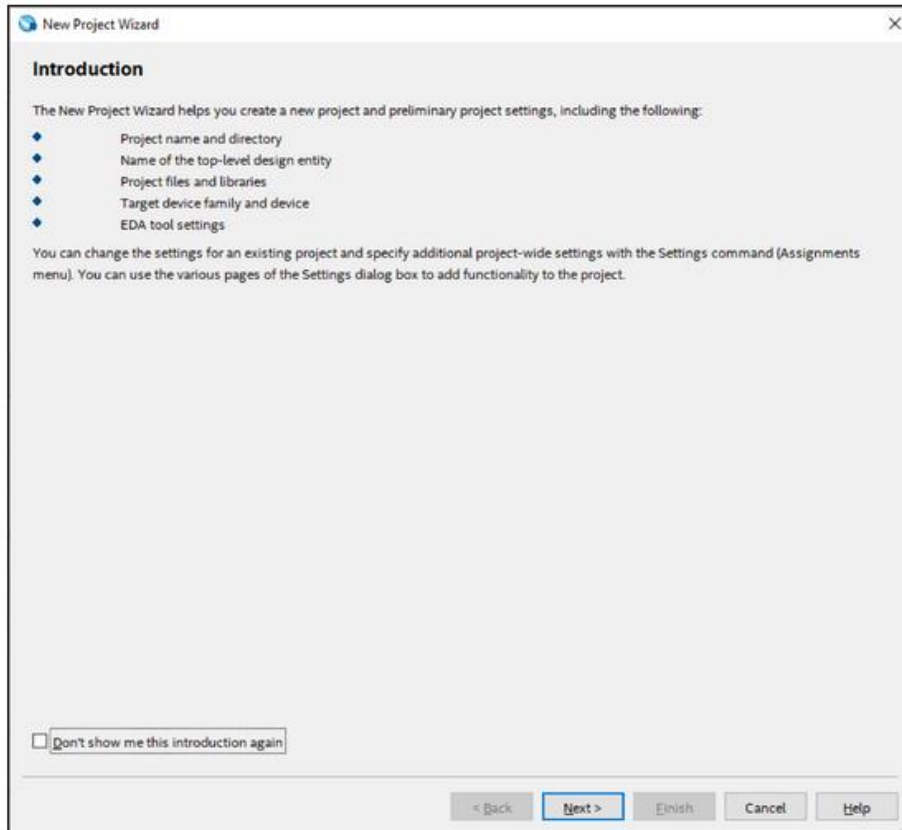
The concepts and procedures outlined here should enable you to begin working on laboratory projects. Additional features will also be explained throughout the laboratory exercises. However, you should refer back to this tutorial if at any time you forget any of these basic functions.

Simulating an Intro Circuit

Step 1.



Upon starting Quartus Prime (and perhaps dismissing a window that appears with the Lite edition), the main interface window will be presented as seen above. From the menu at the top, select **File => New Project Wizard...** to create a new project. (**NOT File ==> New**, because there is a fundamental difference between creating a project and creating a single file.) At this point, Quartus will display an introduction screen for the project wizard.



Get in the habit of reading information windows before moving on. On this screen, select **Next** to advance to the next window. You may also want to check the box in the lower left corner to avoid displaying this particular introduction screen again.

Step 2.

Directory, Name, Top-Level Entity

What is the working directory for this project?
D:/ECE3170/Lab1

What is the name of this project?
IntroCircuit

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
IntroCircuit

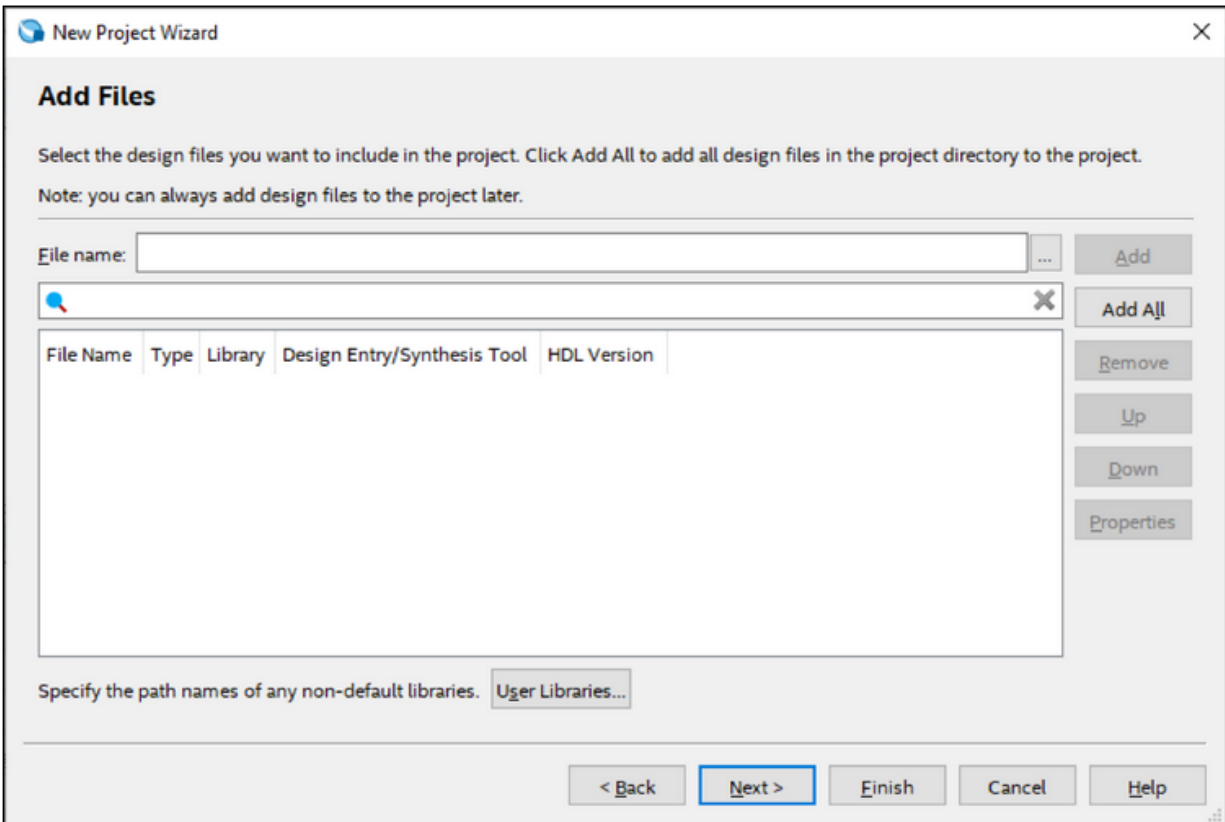
Use Existing Project Settings...

< Back Next > Finish Cancel Help

Enter a location to store your project files, as in the example above. **NOTE: You cannot store the project files in the default location as that folder is read-only. Choose a location outside of the C:/IntelFPGA_lite folder.** Include a dedicated directory (i.e., folder) for the particular project, which is *D:/ECE3170/Lab1* in the example here. Next, give the project a name. The example uses *IntroCircuit* for the name of the project. The last entry is the top-level design entity, which is an important concept. A project can consist of as little as a single design file, such as a single schematic. Or it can contain one file which describes how other design files are interrelated. We will create this project with only one file, which by definition must be the top-level design entity. Quartus will default to using the project name for the top-level design file, which works well here.

Then, select **Next** to advance to a window which asks you to choose between an "Empty project" or a "Project template." Choose the "Empty project" and select **Next** to advance.

Step 3.



Press **Next** to advance to the window above. Allow Quartus to create the project directory, if it does not yet exist. The user has the opportunity to add any files to the project that define logic, such as schematics created beforehand. Since there are no design files to add to this project, click **Next** to advance to the device selection window of the figure below.

Step 4.

New Project Wizard

Family, Device & Board Settings

Device | Board

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST) ▾
Device: All ▾

Target device

Auto device selected by the Fitter
 Specific device selected in 'Available devices' list
 Other: n/a

Show in 'Available devices' list

Package: Any ▾
Pin count: Any ▾
Core speed grade: Any ▾
Name filter:
 Show advanced devices

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel P
5CSXFC6D6F31A7	1.1V	41910	499	457	9	9
5CSXFC6D6F31C6	1.1V	41910	499	457	9	9
5CSXFC6D6F31C7	1.1V	41910	499	457	9	9
5CSXFC6D6F31C8	1.1V	41910	499	457	9	9
5CSXFC6D6F31C8ES	1.1V	41910	499	457	9	9
5CSXFC6D6F31I7	1.1V	41910	499	457	9	9
5CSXFC6D6F31I7ES	1.1V	41910	499	457	9	9

< Back | Next > | Finish | Cancel | Help

You will need to select the correct device for the board you are using, first by selecting a device family, and then by selecting a specific target device. If you followed the recommendation to only install device support for the Cyclone 5 family, then your "Family" choice will default to Cyclone V (*E/GX/GT/SX/ST*), but if you chose to install support for other devices if already had support for other families installed, you may have to select this family yourself from the dropdown.

Now, you need to find the exact device by reading the corresponding code near the middle of the FPGA IC (*Integrated Circuit*), on your DE10-Standard board. Look closely at the board and

make a note of the complete device name for the Cyclone V chip. It is the long string directly below "Cyclone V SoC."

Continuing with the *Family, Device & Board Settings* dialog, we now need to specify the exact device, using the device name you just found. To select the specific device, you could

- scroll through the list at the bottom of the window to find it, or
- start typing the name in the *Name filter* box, or
- if you knew the Package, Pin Count, or Core speed grade, use those dropdowns.

Select the correct device in the bottom list. Press **Next** to continue to the window below.

New Project Wizard

EDA Tool Settings

Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

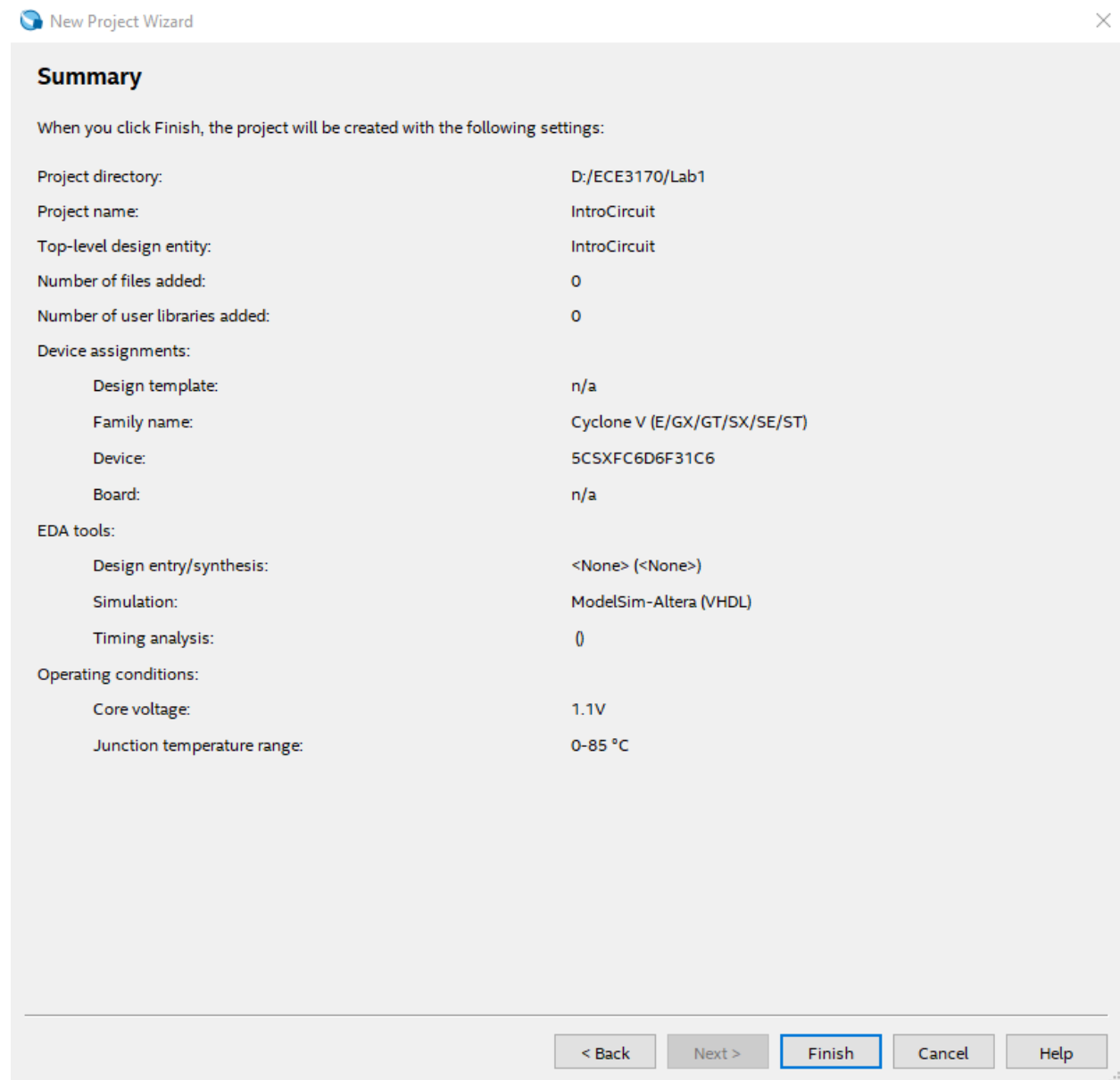
Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synth...	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

< Back Next > Finish Cancel Help

It is common for commercial entities to use third-party tools to develop hardware and software configurations for FPGAs (Field Programmable Gate Array). This window allows Quartus to communicate with and use such tools within the development environment. We will mostly be using the default tools included within Quartus, but we will use a version of a widely supported simulator called ModelSim. To the right of the Simulation,

- select *ModelSim-Altera* (NOT generic *ModelSim*), and
- select *VHDL* in the second dropdown (under *Format(s)*)

Then click **Next** to proceed to the summary window.



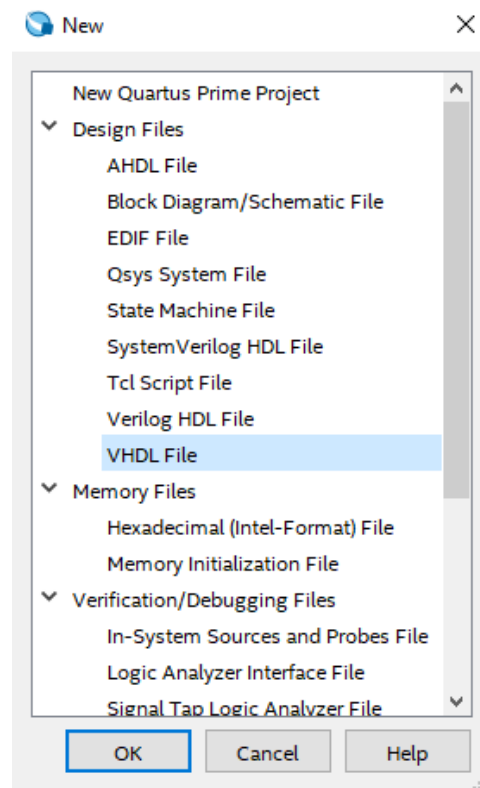
Take the Summary window as screenshot, you will need to submit this for the pre-lab.

With the summary window in view, press **Finish** to close the wizard and create the new project files. You will be returned to the main window of Quartus.

Optionally, this is a good time to take a break. Note that you can choose the menu option **File => Save Project** and then **File => Close Project** to return to a place where you can close Quartus entirely. You would, of course, have to get back to the same point with **File => Open Project**, selecting the *IntroCircuit* project just saved. Note that this is different from **File => Open File**, which would open an isolated file, without context to the project settings, including the device assignments.

Step 5.

Design projects are composed of one or more design files, and this project will be defined by a single VHDL file. Earlier, the concept of a *top-level design entity* was mentioned, to define the one file that contains the overall design definition in some entity (device). To reiterate, we are only going to have one file, containing the one top-level design entity. From the menu, select **File => New...** to bring up the dialog below, and select **VHDL File** as was done here.



Quartus will only let you save it after you type something in it, so go ahead and add the library and use statements that will be used for this file:

```
library ieee;
use ieee.std_logic_1164.all;
```

This will be your top-level design file for this project, so save it using the same name as your project so that Quartus automatically uses it as the top-level design.

Step 6. Device ports

Add this entity and port statement to your VHDL. The entity name should match your file name, because Quartus will be looking for a device with that name.

```
entity IntroCircuit is
port(
    B0      : in  std_logic;
    B1      : in  std_logic;
    B2      : in  std_logic;
    S0      : in  std_logic;
    S1      : in  std_logic;
    S2      : in  std_logic;
    S3      : in  std_logic;
    clk     : in  std_logic;
    LED0    : out std_logic;
    LED1    : out std_logic;
    LED2    : out std_logic;
    LED3    : out std_logic;
);
end IntroCircuit;
```

-- Describes the device from the outside
-- Defines the signals coming into and out of the device

Step 7. Device architecture

We will implement a few simple logic expressions to ensure proper operation of the DE-10 Standard board. Because this program is quite simple, you probably do not need to declare any internal signals. You can assign each output with a single line of VHDL. The architecture name is up to you (here it has been named "behavior"), but it does need to be declared as an architecture *of your* device, so use your device name from the entity statement for that part of the architecture declaration.

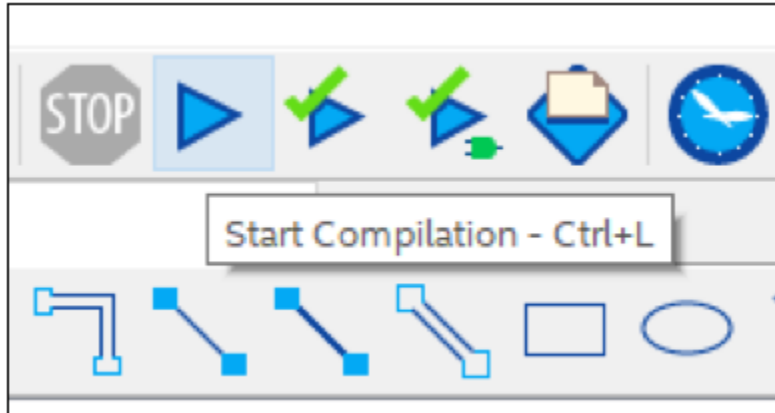
```
architecture behavior of IntroCircuit is
begin
    LED0 <= NOT B0;
    LED1 <= B1 AND S0;
    LED2 <= S1 XOR S2;

    process(clk, B2) is
    begin
        if rising_edge(clk) and B2 = '0' then
            LED3 <= S3;
        end if;
    end process;

end behavior;
```

Step 8.

Save the file and compile the project. With the file saved, compile the design by selecting **Processing => Start Compilation**, or by clicking on the corresponding icon (the triangle to the right of STOP) in the menu bar shown below.



Fix any compilation errors before moving on.

When Quartus reports a compilation error in a VHDL file, double-clicking the error should take you to where it found the error. Note though that it takes you to where Quartus *noticed* the error, not necessarily where the error occurred.

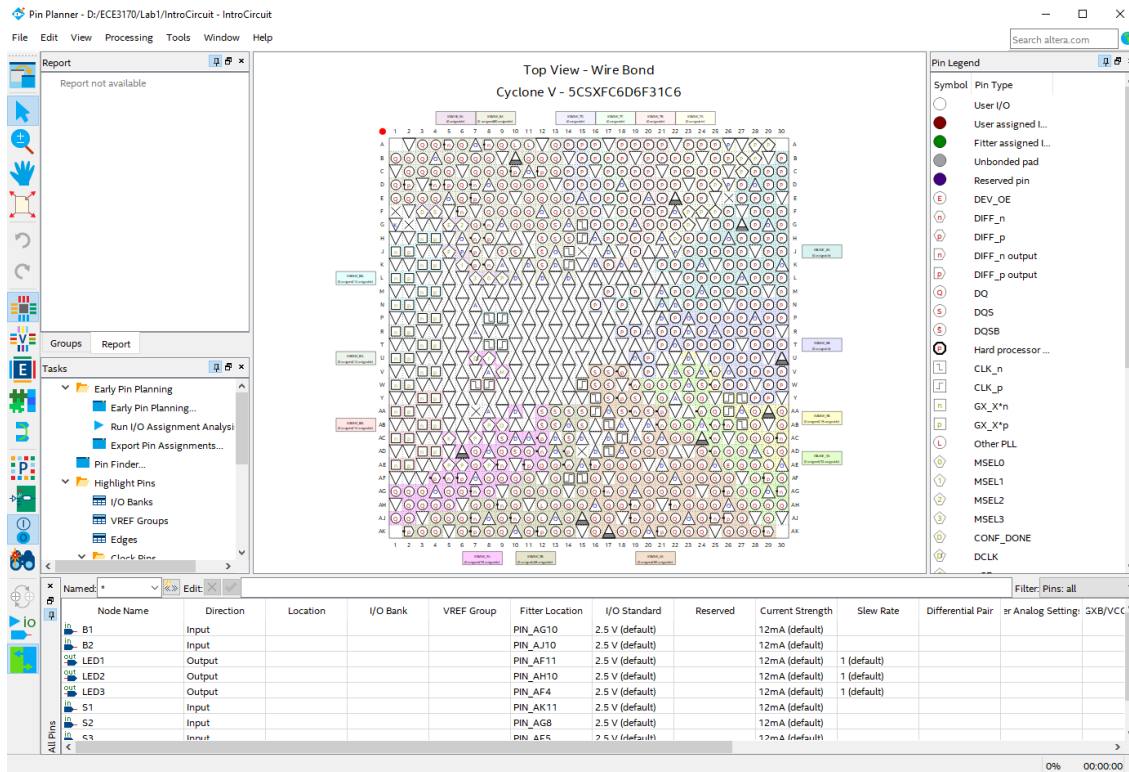
Please take a Screenshot of the Compilation report.

Step 9.

The final step in the design process is to add pin assignments to the design file. We chose input names that correspond to elements of the development board -- the buttons, switches, and LEDs. But Quartus is not designed to work with just our board, and it does not know where those devices connect to pins on the FPGA. That information is in the manual for the board, and the relevant information will be provided below.

Open the pin planner by selecting **Assignments => Pin Planner** (or find the icon in the toolbar), which brings up the window below. Notice that the I/O pins are listed in the table at the bottom of the Pin Planner.

Since the pins in your design must be assigned to pins on the FPGA, the Fitter stage of compilation made arbitrary choices, displayed in the **Fitter Location** column of the Pin Planner (not shown below). These are not correct, since Quartus has no knowledge of the development board and the usage of FPGA pins relative to switches, LEDs, and pushbuttons that we wish to use.



For each I/O pin in the list, double-click on the **Location** cell (not the **Fitter Location** cell) and enter the pin numbers shown below. Note that you can omit "PIN_" as you are typing, if you like, since Quartus will fill in the leading characters. Or you can select from a drop-down list, instead of typing. Again, you would have to search the documentation for the board to know that, for example, the LED called LEDR0 is actually connected to pin AA24 on the FPGA chip.

Node Name	Direction	Location	I/O Bank	VREF Group
in B0	Input	PIN_AJ4	3B	B3B_NO
in B1	Input	PIN_AK4	3B	B3B_NO
in B2	Input	PIN_AA14	3B	B3B_NO
in clk	Input	PIN_AC18	4A	B4A_NO
out LED0	Output	PIN_AA24	5A	B5A_NO
out LED1	Output	PIN_AB23	5A	B5A_NO
out LED2	Output	PIN_AC23	4A	B4A_NO
out LED3	Output	PIN_AD24	4A	B4A_NO
in S0	Input	PIN_AB30	5B	B5B_NO
in S1	Input	PIN_Y27	5B	B5B_NO
in S2	Input	PIN_AB28	5B	B5B_NO
in S3	Input	PIN_AC30	5B	B5B_NO

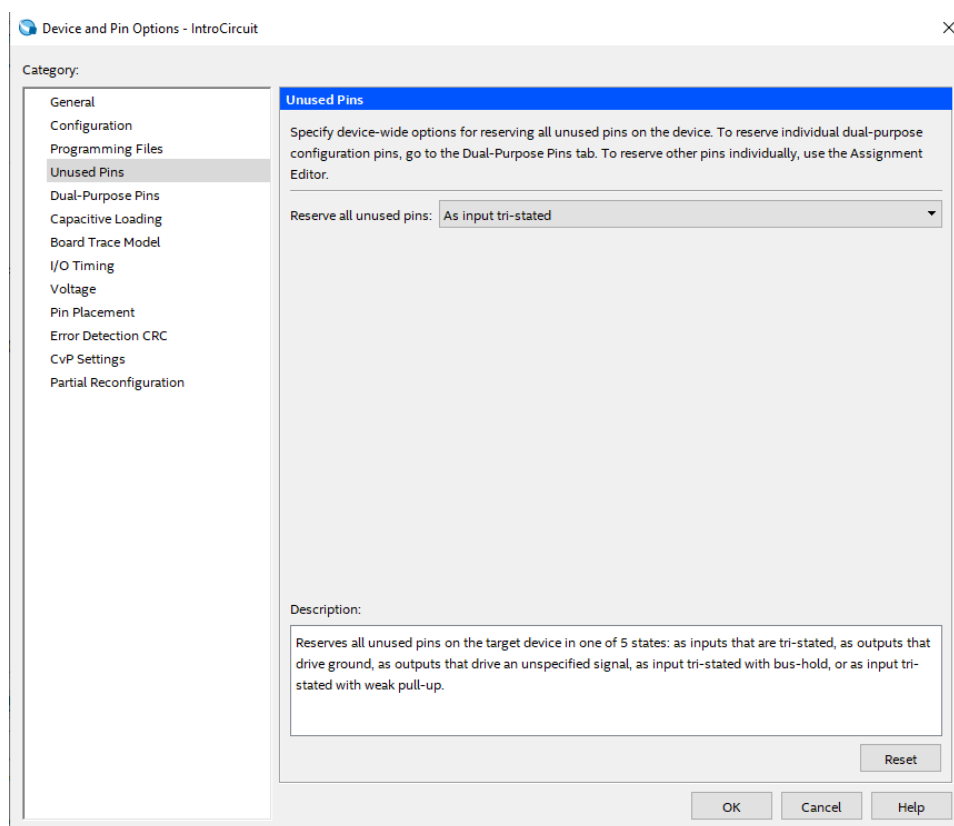
With all of the pin assignments made, close the Pin Planner window. It is very important to note that even though the correct pin numbers have been assigned, the project must be compiled again before these pins will actually be used in any files that can be programmed onto the DE10-Standard board. In this case, we will compile at least one more time after making a change in the next step.

Step 10.

The following changes to default project settings are critical to the correct and safe operation of designs programmed to the DE10-Standard. We will remind you to change them in the first couple labs, but you need to remember to change them whenever you create a new Quartus project.

We have just told Quartus what to do with eight of the many pins that are connected to the Cyclone V FPGA chip. But there are many other pins, and if we are not specific, Quartus may use them for intermediate signals, or drive them deliberately high or deliberately low. This could result in strange patterns on the LEDs or other outputs. It could even damage something on the board that is not meant to be driven.

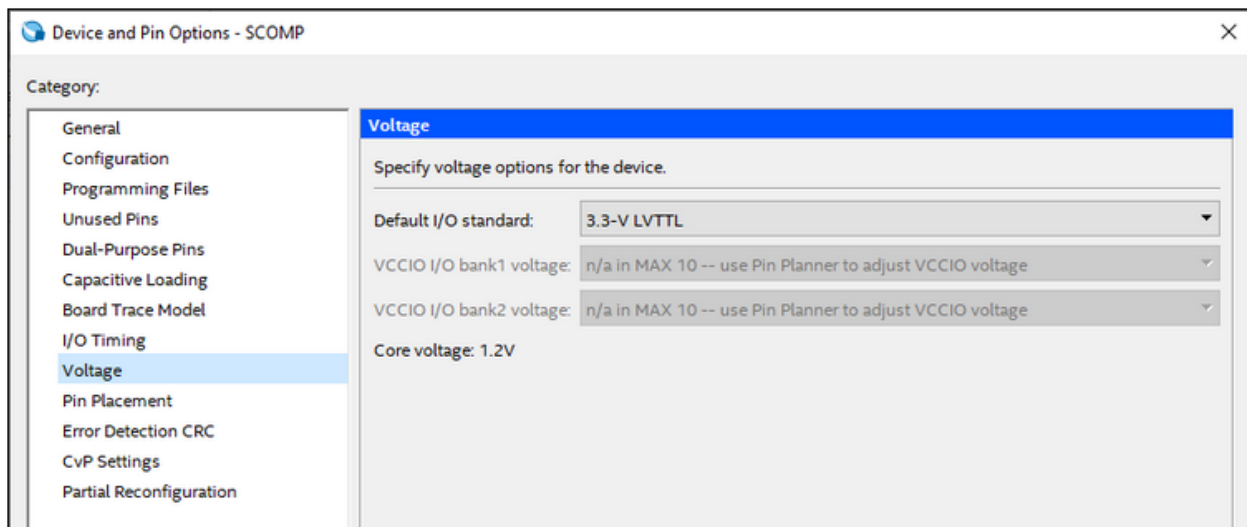
Fortunately, there is an option to define what to do with all of the pins that are not needed in our design. From the main Quartus menu, select **>Assignments => Device...** to bring up the Device settings window. It is the same dialog that we used earlier to define the target FPGA device. Look for the **Device and Pin Options...** button and click it to bring up the window shown below. Select the **Unused Pins** category on the left, and then **as input tri-stated** on the right..



Without getting too technical, this option forces all of the unused pins on the FPGA to be passive inputs. In normal new design situations, this would not matter, since the board design would typically be done after the FPGA is designed. However, the projects in these laboratory

exercises will target your DE10-Standard FPGA board, which has several other ICs and connectors on it in addition to the FPGA. Many of these ICs connect directly to the FPGA I/O pins. To avoid causing erroneous board operation or potential damage to the FPGA or other ICs, it is important to change this setting every time a new project is created. Once complete, press the **OK** button to close this window. Press the **OK** button on the settings window to complete the options change.

In the same Device and Pin Options dialog, go to the Voltage tab, and change the Default I/O Standard to "3.3-V LVTTTL", as shown here:

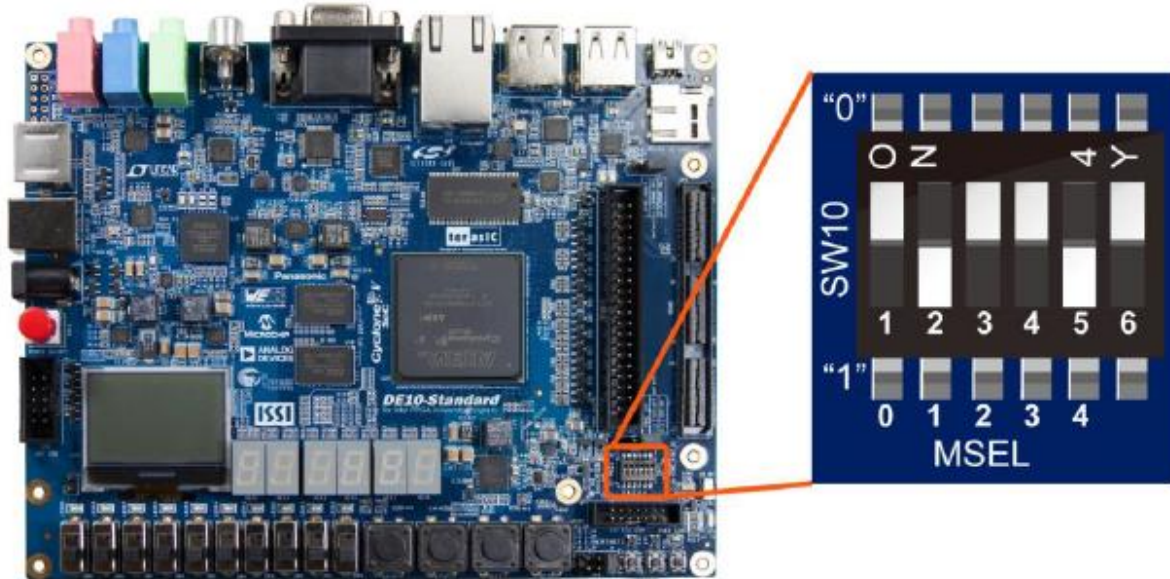


The FPGA on the DE10-Standard can configure its pins to use different voltages, and the majority of devices on the board require 3.3 V. Changing this default settings will save you from needing to individually configure each pin (and from the design not working if you forget to change one).

Compile your project one more time (because you just changed some settings that need to be incorporated into the programming file), either from the toolbar button (the triangle) or the **Processing** menu. If you have any errors that you cannot figure out, use online resources to ask instructors or TA.

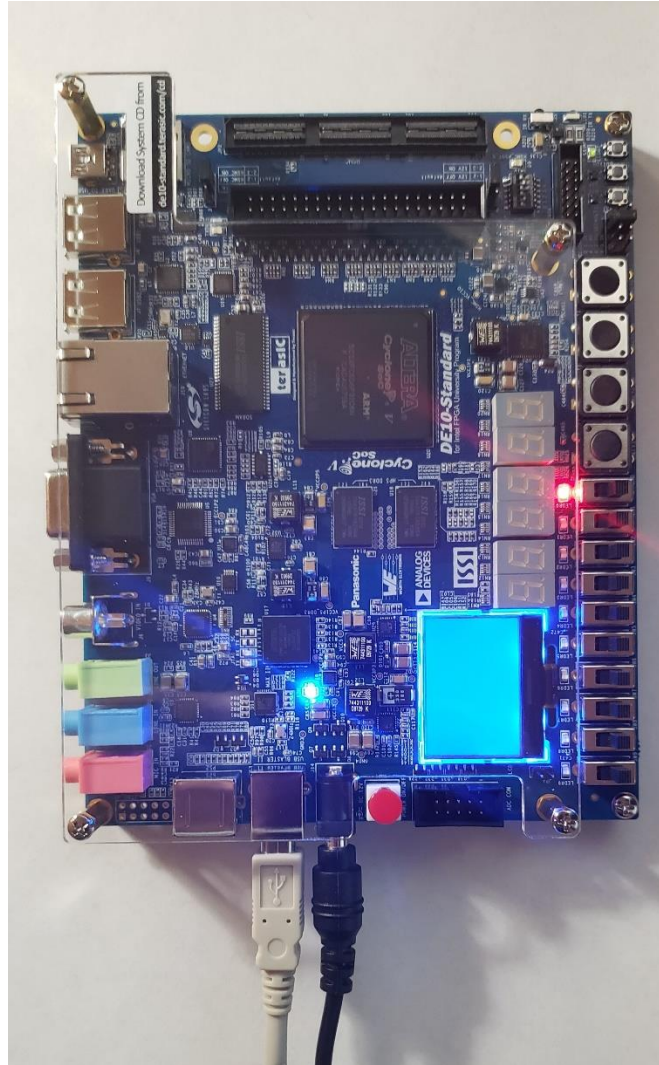
Step 11.

Normally in this course, we will simulate our design and possibly fix some errors. But for this simple project, we will go straight to a hardware test.



For programming of the FPGA using JTAG, the above shown switch positions are required (MSEL [4:0] set to "10010"). **NOTE: These positions will change in later labs when we program the HPS (Arm Processor). They will need to be set appropriately depending on if we are programming the FPGA or HPS.**

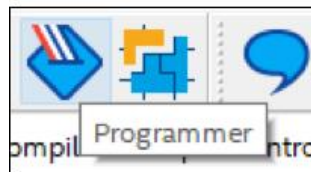
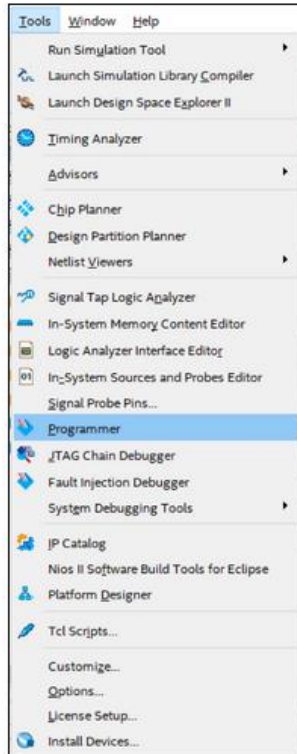
Connect your computer to your target development board. This should require only a USB cable between the computer and the DE10-Standard board, using one of the cables supplied with the board as shown below.



The term "Programming" is used by Quartus, and we will use it as well to minimize confusion. But you should always think of this as "configuring" the FPGA to connect its logic, flip-flops, and other hardware internally. There is no program "running" on the board -- it will be implementing hardware.

Before configuring/programming the board with your design file, open the device setting window by selecting **Assignments => Device...** from the menu. Verify that the correct FPGA device is still selected. If the wrong device is shown, select the correct FPGA and press the **OK** button. This could require you to go back and reassign pin numbers, but should not be necessary if you completed earlier steps with the correct device assignment.

Open the programmer tool, by selecting **Tools => Programmer** from the menu or selecting the icon from the toolbar.

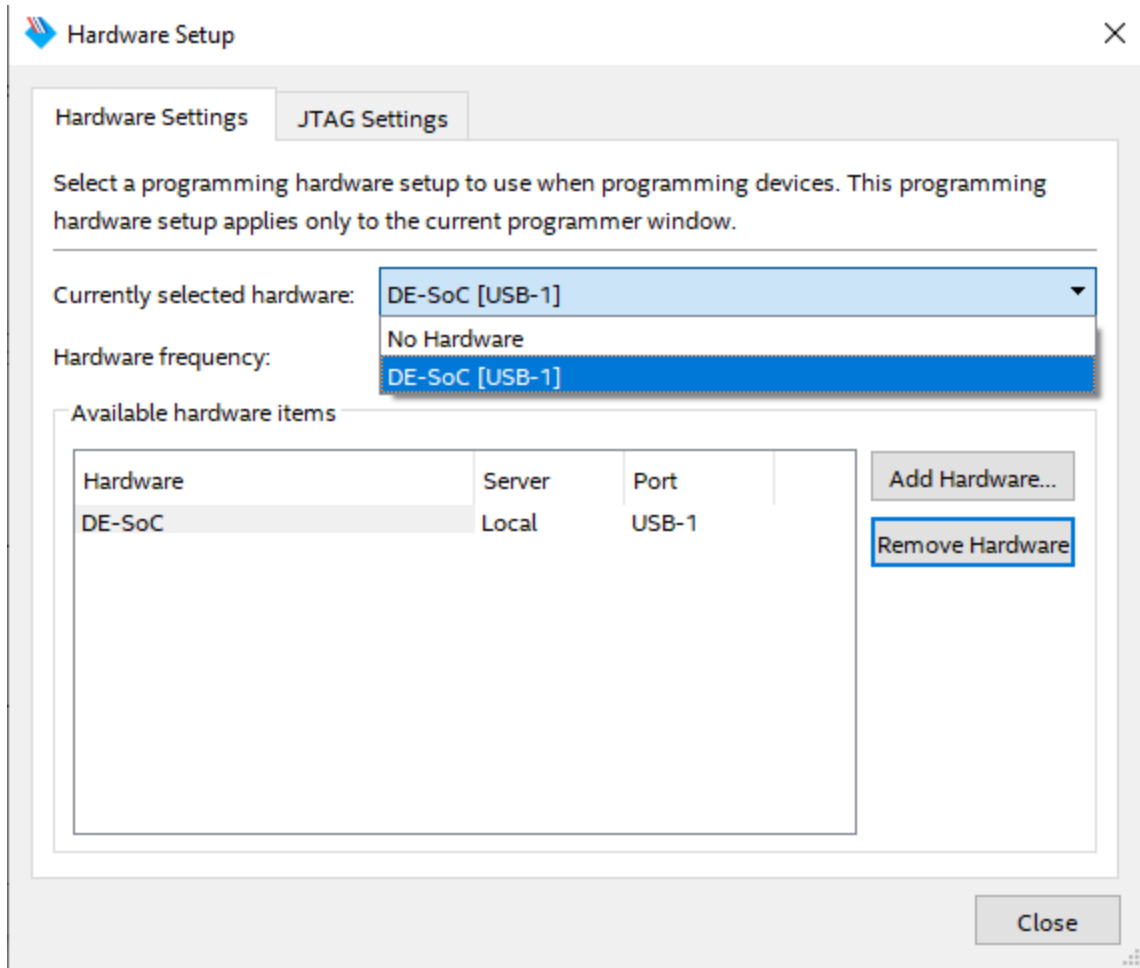


NOTE: This next section is NOT the same as in ECE 2031.

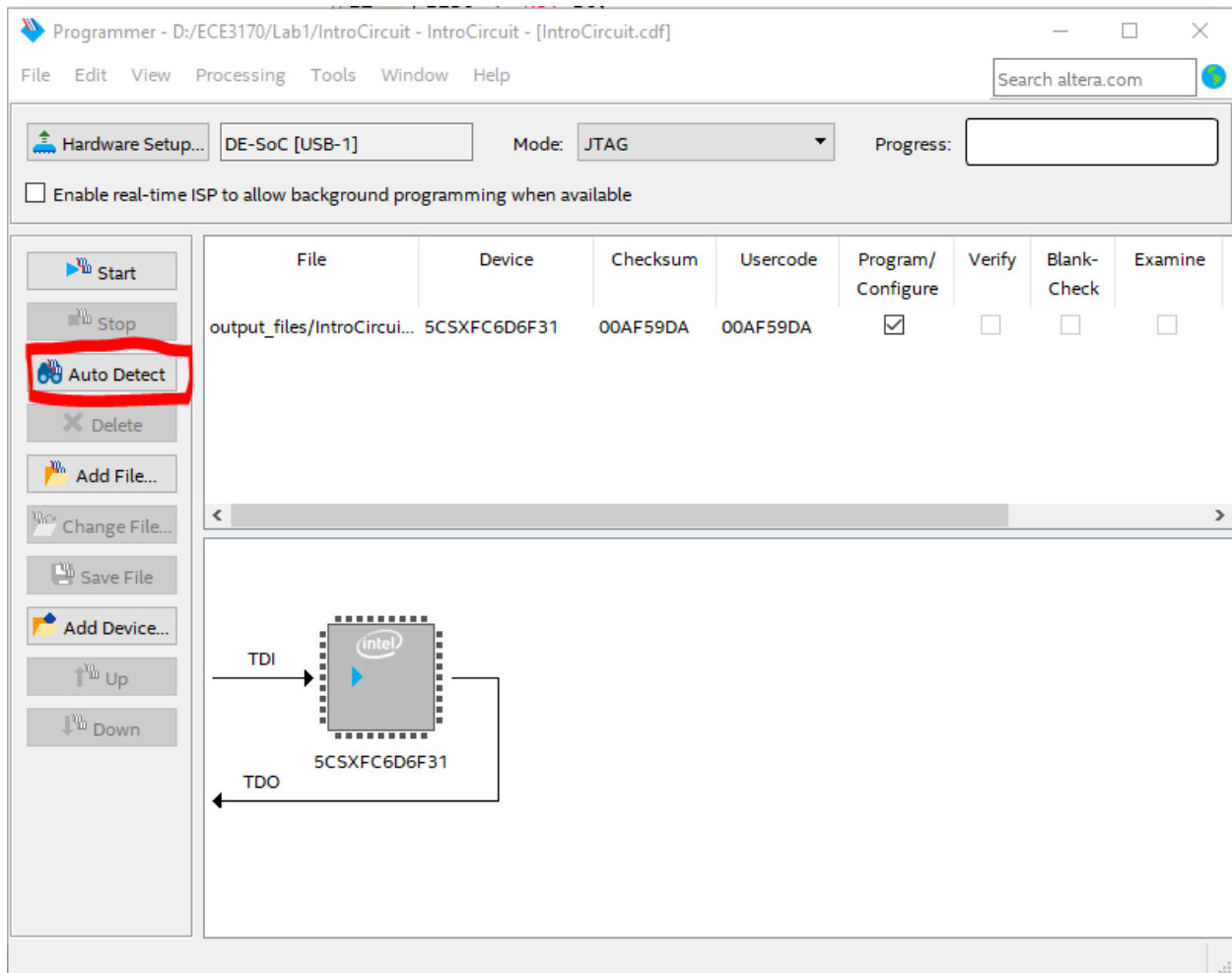
You should see a window like the one below. When you open the Programmer while working on a project that has been compiled, it should correctly select the file holding the logic (top windowpane, left column) and the correct device (second column). It also shows a diagram of the expected connection to the device, which for our case is not the correct connection.

We will program the board using the JTAG chain. This is in volatile memory and will not be stored after a power off of the device.

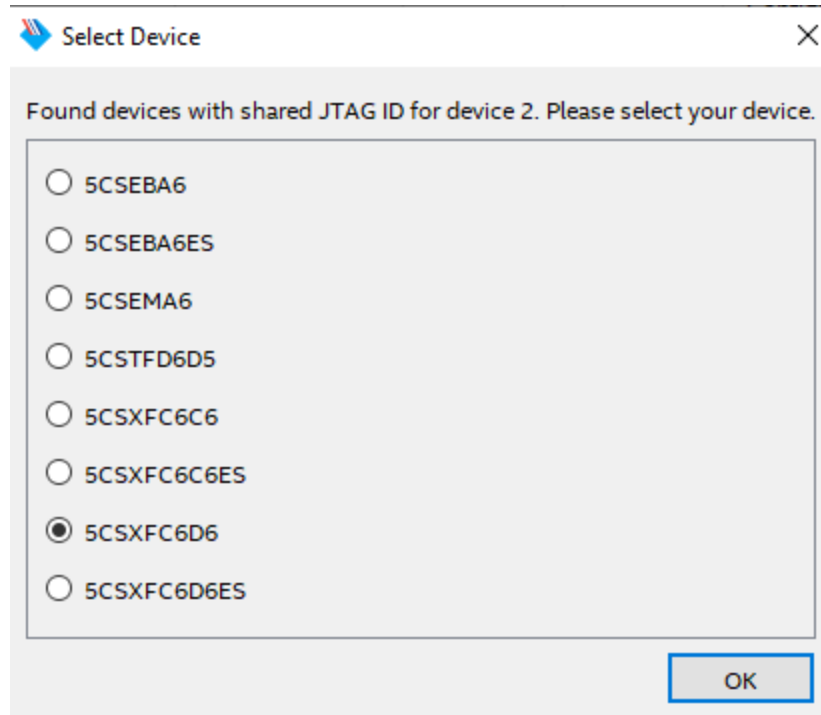
The **Hardware Setup** refers to the programming hardware between your computer and the chip, including the USB port, the cable, and some interface hardware on the board. If the **Hardware Setup** is listed as **No Hardware**, which is normally the case for first-time use, click on the **Hardware Setup...** button to display the window below.



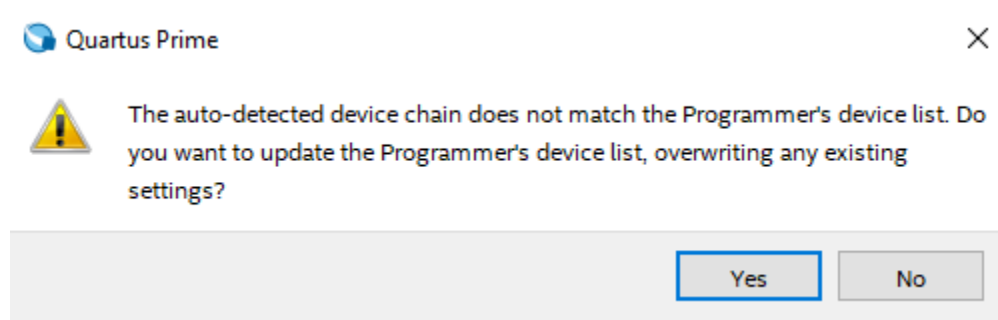
Under the **Currently selected hardware** option, choose **DE-SoC [USB-X]** and close the window. If no programming hardware is listed, make sure that the USB cable is in place, and possibly try the **Add Hardware** button. If it still does not work, go back to the steps taken when Quartus was installed, because those steps included an initial test of the USB Blaster setup. Close the Hardware Setup window after it correctly shows the USB-Blaster as the currently selected hardware.



Select "Auto Detect" shown in the image above.



Select the detected device associated with the board as shown above.



Press "Yes" when you receive the above pop-up.

You should now see what is shown below:

Programmer - D:/ECE3170/Lab1/IntroCircuit - IntroCircuit - [IntroCircuit.cdf]*

File Edit View Processing Tools Window Help

Search altera.com

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress:

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<none>	5CSXFC6D6	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Start Stop Auto Detect Delete Add File... Change File... Save File Add Device... Up Down

Right click on the FPGA (the chip on the right) device and open the .sof file to be programmed by navigating to **Edit...Change File**

If **Change File** is grayed out, make sure to left-click once on the FPGA (chip on the right), which will cause a dashed border to appear around the chip image.

Programmer - D:/ECE3170/Lab1/IntroCircuit - IntroCircuit - [IntroCircuit.cdf]*

File Edit View Processing Tools Window Help

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: (Failed)

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
output_files/IntroCircui...	5CSXFC6D6F31	00AF59DA	00AF59DA	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File
Add Device...
Up
Down

Replaces a selected programming file

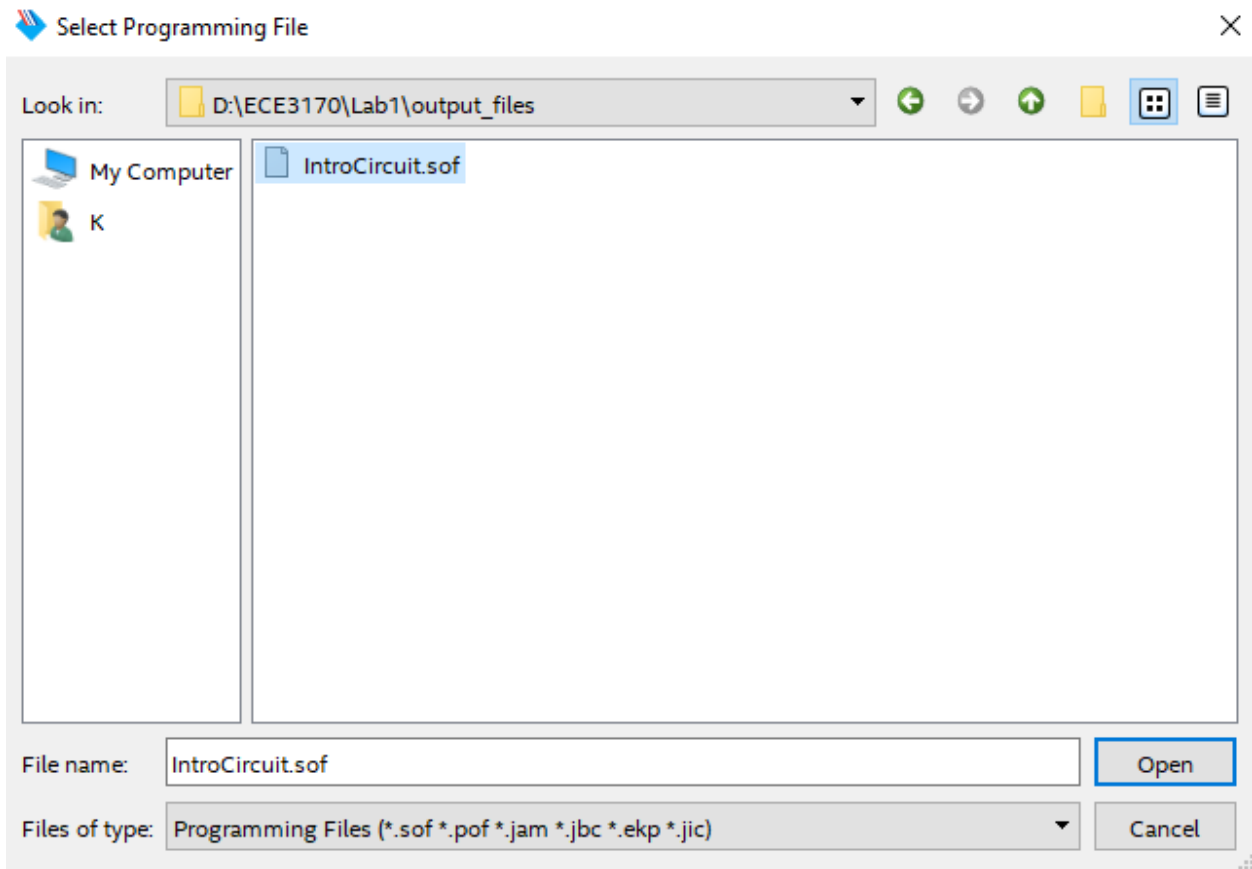
Message

```

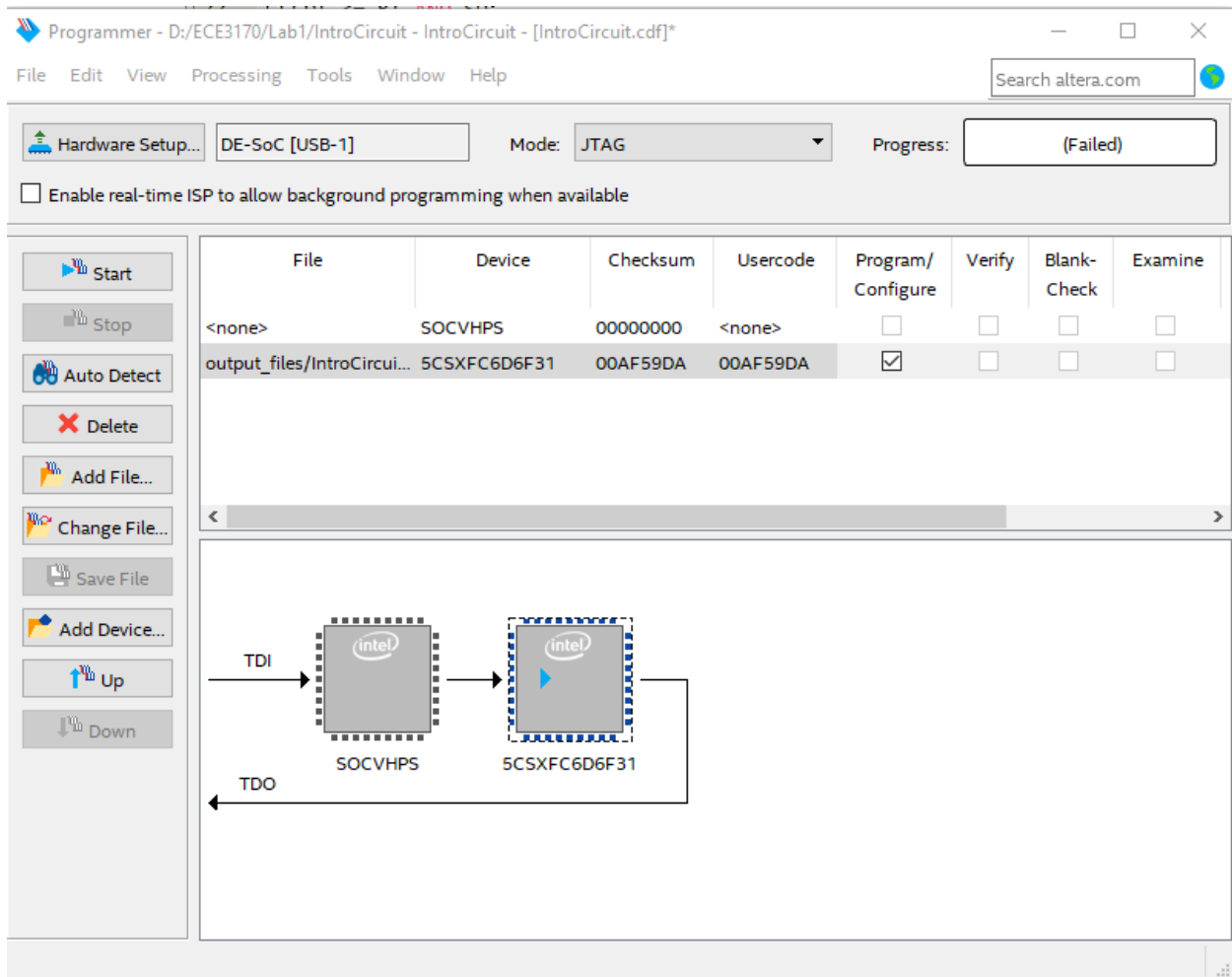
2 Design is not fully constrained for setup requirements
2 Design is not fully constrained for hold requirements
Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
*****
Running Quartus Prime EDA Netlist Writer

```

You should see a file IntroCircuit.sof located in the “output_files” folder. Select this file.



Select the checkbox for Program/Configure on the FPGA as shown below, and then press **Start** to program the FPGA. Take a screenshot for your lab report showing the programmer has successfully programmed the FPGA. (Essentially the below screenshot but with a success instead of "Failed" which is shown now)



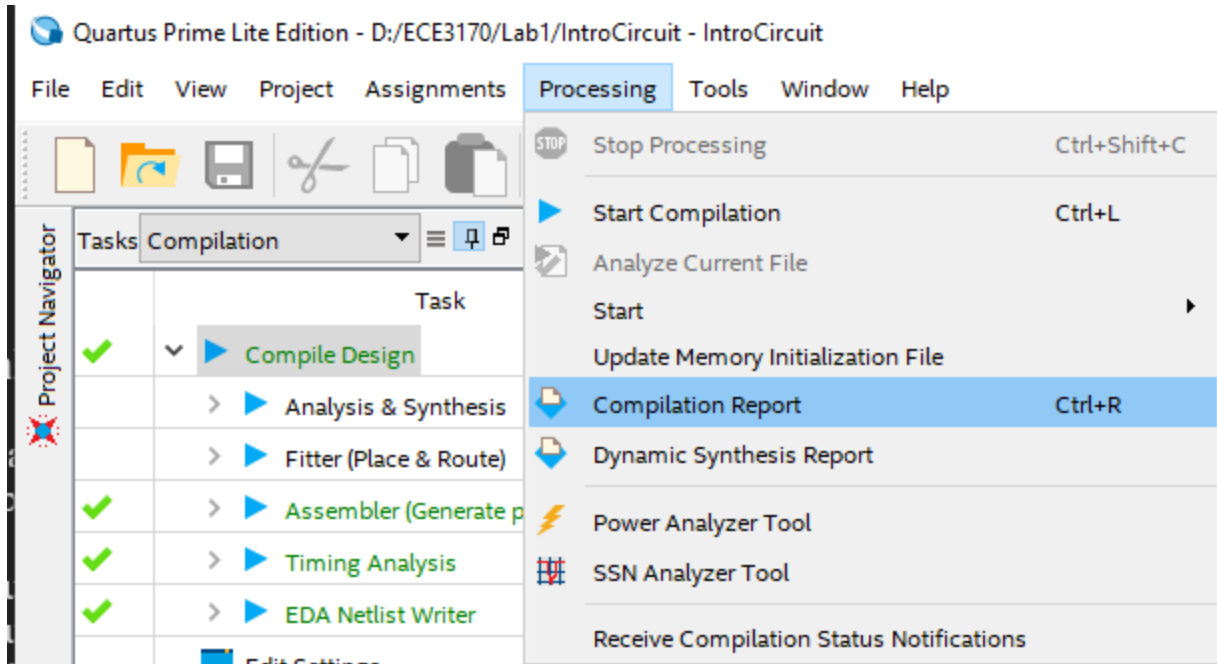
Verify that the LEDs operate as expected with usage of the push buttons and switches. Note that the push-buttons generate a low logic level when pressed (active low).

Examining the compilation report

Step 1.

After compiling, the compilation report should automatically open in Quartus

If it does not, you can open it by **Processing -> Compilation Report**



Step 2.

With the Compilation Report open, navigate to the table of contents on the left, and open **Analysis & Synthesis -> Resource Usage Summary**. Here you can see a basic summary of the resources utilized for the project. We will require this summary for later lab portions.

ModelSim Tutorial and Functional Simulation of SHA2 VHDL Code

When programming in any language, it is useful to debug, test, or simulate your code to verify its functionality. When programming in VHDL, the convention is to have functional VHDL code and a testbench which tests the code. In this section, you will simulate the provided testbench (sha256_test.vhd) with the given VHDL code in ModelSim. This code is present in lab1/sha256_vhdl.

To better understand the structure of the VHDL code, you are given the hierarchy of the provided VHDL files below:

- sha256_test.vhd -- testbench (for simulation only)
 - gv_sha256.vhd -- top level
 - sha256_control.vhd
 - sha256_padding.vhd
 - sha256_msg_sch.vhd
 - sha256_hash_core.vhd
 - sha256_regs.vhd

- sha256_Kt_rom.vhd
- sha256_Ki_rom.vhd

To simulate the code, proceed as follows:

- Run ModelSim. Once ModelSim is open, create a new project by clicking on: File --> New --> Project...
- Verify that the project location is set to lab1/sha256_vhdl and give the project a name, say "sha256", and click OK.
- After creating the project, we have to add all the VHDL files to it. Do so by clicking on: Add Existing File --> Browse...
- Choose all the VHDL files in the directory. Click Open --> OK.
- Once you are done adding ALL the VHDL files, click Close.
- Now we have to compile the design files. To do so click on: Compile --> Compile All. Check the transcript window to make sure all files were successfully compiled.
 - The first time you compile, you may get errors. That is because ModelSim has not figured out the hierarchy of the VHDL files and incorrectly attempts to compile the top-level file first.
 - To fix this, just Compile -> Compile All again.
- Before we start the simulation process, let us take a quick look at the testbench file which is responsible for running the simulation. Open the testbench file by double-clicking on "sha256_test.vhd" in the Project window.
 - The documentation present at the beginning of the file describes the functionality of this specific implementation of the SHA256. It helps in understanding how the input control signals need to be exercised and how the VHDL module can be tested.
 - Read through the documentation (comments) to have a better understanding of the operation of the SHA256 module.
 - Notice that a testbench usually contains the following main sections:
 - An empty testbench entity with no port declarations.
 - A testbench architecture that contains the following:
 - Signal declarations (to be used to connect the unit under test (UUT) to the testbench logic)
 - Component declaration section (usually for the UUT)
 - Component instantiations (usually for the UUT)
 - A clocking process (to create the simulation clock)
 - A testing process (to create the input stimuli and check for expected behavior)
 - In this lab, our unit under test (UUT) is the SHA256 top level entity. So in the testbench, you will see that this component is instantiated and connected the input stimuli. In addition, you will see that the output ports are checked for expected behavior using the assert and report statements.

- **Describe the logic behind the provided testbench as you understand it in your written report.**
- Now we can start the simulation by clicking on: Simulate --> Start Simulation...
In the Start Simulation window, make sure you are on the design tab, expand the work library, choose the testbench for this code named: "testbench", and click OK.
- ModelSim will change view into simulation mode and a couple of other windows show up.
- Our next step is to add some signals of interest to a wave window to monitor their changes as simulation proceeds. Before doing so, let us create a dump file that will be storing the results of our simulation as we perform it. To do so, type the following in the command line of the "Transcript" window:

```
vcd file sim_results.vcd
vcd add testbench/*
vcd add testbench/Inst_sha_256_dut/*
```
- Now let us add our signals of interest to the wave window to monitor their changes as simulation proceeds. To do so, go to the "sim" window and click on the instance named "testbench" to add the testbench signals. Next, open the "Objects" window and choose all the signals (inputs, outputs, and internal). Right-click on the selection and click on Add Wave. Check that the signals have been successfully added to the "Wave" window.
- Our final step is to run the simulation for a specific time. For this testbench, running the simulation for 17200 ns should be enough. To do so, type **run 17200 ns** in the command line of the "Transcript" window.
- Navigating back to the "Wave" window will now show you the result of the simulation for all the signals that we added. To better read the values, select all the signals and right-click, then change the Radix to Hexadecimal. Also, towards the bottom left of the signals pane (to the left of where it says "Now"), there is a blue button that has a description of "Toggle leaf names <-> full names" if you hover the mouse over it. Click on that button to show the signal names only without the hierarchy.
- Look through the wave window and try to understand how the signals are changing values with respect to the simulation time. Specifically, look for the output signals that show the resultant hash value. Notice that the hash value is only considered valid when the signal "dut_do_valid" is asserted.
- Now that we have run the simulation, make sure that you set the zoom of the wave window to show the results at least between 0 ns and 1500 ns. To do so, right-click anywhere in the wave window and use the Zoom Range option. Export an image of your simulated waveform by clicking on File --> Export --> Image... and save it as an image.
Include this image in your submission.
- **In your report, briefly explain each of the test cases in the testbench indicating whether the test case passed or failed. For cases that fail, if any, mention why you think that happened.** (HINT: read the comments in the testbench carefully).
- Before ending the simulation, open the transcript window and verify that the no reports are generated by the testbench indicating a failure of any of the test cases.

- Finally, to end the simulation and correctly save the results in the VCD file, click on: Simulation --> End Simulation. **Submit the generated VCD dump file electronically.**
- We are done with VHDL simulation. You can close the ModelSim window.

To verify whether the generated results of our inputs are correct, we will run the data of the first 4 test cases on a software implementation of SHA256. For this lab, we will use either the Linux "sha256sum" command or the Windows "Get-FileHash" command. Documentation on the two commands can be found in the following links:

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-5.1>

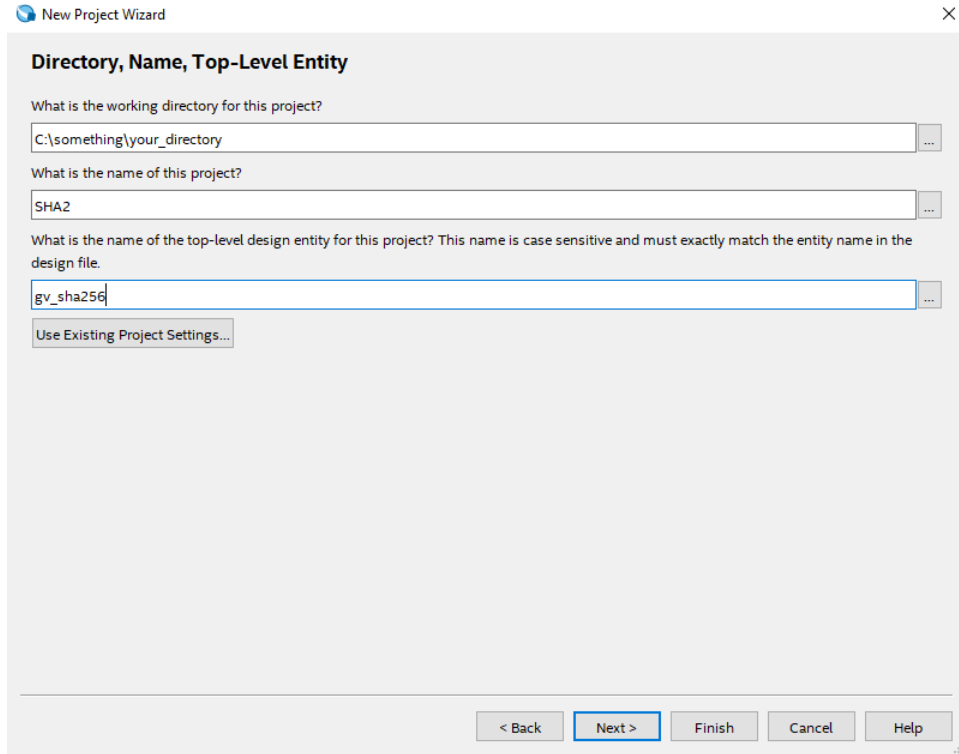
https://docs.oracle.com/cd/E36784_01/html/E36870/sha256sum-1.html

Generate the hashes of the first 4 test cases using either command by placing each test case in a file and running the command on that file. For each test case, compare the software generated hash value with the generated value in your VHDL simulation. **In your written report, mention whether each of the test cases had the correct hash and provide the correct hash value for the cases that did not, if any.**

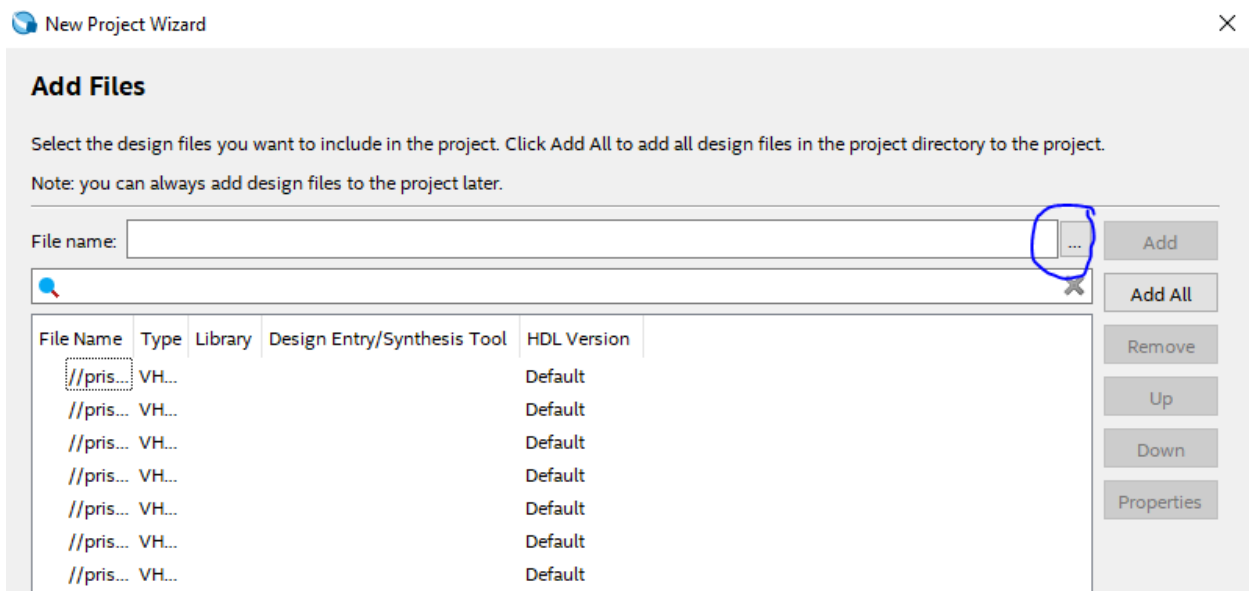
HINT: When passing your input to the "sha256sum" command or the "Get-FileHash" command, make sure the input does not contain any new line or end-of-file characters since that would generate a completely different and incorrect hash.

SHA2 VHDL Synthesis

1. In Quartus, start a new project. Name the project "SHA2", with the top-level entity as "gv_sha256" as shown below.



- Then, select **Next** to advance to a window which asks you to choose between an "Empty project" or a "Project template." Choose the "Empty project" and select **Next** to advance.
- On the below screen, press the button circled in blue, and the provided SHA2 VHDL files in the sha256_vhdl directory – **except do not add the test bench "sha256_test.vhd"** – to the project and select **Next** to advance.



- As before, select the appropriate FPGA device and select **Next**.

5. As before, select **ModelSim-Altera** and **VHDL** and then **Finish** after verifying the project options are correct.

6. Compile the project. Take a screenshot of the **Resource Utilization Summary**.

7. Report any **red** errors generated through the compilation process. Explain in your own words the reason(s) for the errors. Additionally, include a possible solution to the errors that will work for the Cyclone V SoC. You do not need to actually implement your solution in this lab.

In this lab we will not be loading the SHA-256 VHDL onto the Cyclone V SoC. In a future lab we will explore implementing the VHDL on the FPGA with an interface through the SoC's processor.

PreLab1 Report

Please type all your answers and include a cover sheet with your first and last name, GT ID number, and GT username.

Simulation:

1. Brief description of the test bench and its major components.
2. The exported waveform image of the simulated design.
3. Brief explanation of each of the test cases and whether they passed or failed (reason for failure, if any).
4. The dump file sim_results.vcd of the simulated design.
5. Comparison with software hash values.

Synthesis and Implementation:

1. Utilization results for all design components.
2. Compilation Report
3. Error generated during implementation and reason behind it.
4. Your ideas for a possible solution.

Canvas Submission of PreLab1 Files

1. Place all files into a single folder and submit your work on Canvas.

LAB 1

In this lab, you will first embed the VHDL implementation of the widely used Secure Hash Algorithm SHA2 into a top-level design. You will be given a VHDL code that implements the algorithm. You will then add a new VHDL module to your top-level design. Finally, you will create a test bench to set up and simulate your new design. Bare-bones design files are provided to you. The files are missing hardware logic that implements the needed components of this lab.

After achieving correct functionality by testing your design in ModelSim, you will synthesize and implement a file to implement your design on the DE-10 Standard board using Quartus. When synthesizing and implementing the new design, you will verify that some parts of the process that failed and generated errors in Prelab got resolved. In this lab, you will be asked to report how you think these errors are now resolved.

Please type your answers to the questions in this lab into a lab report.

This lab will not reiterate the steps explained in Prelab. If you need a reminder of the functionality of Quartus, please refer to PreLab.

I. VHDL/ DE-10 Help

There is plenty of documentation available on how to write good VHDL. Some good simple examples can be found [here](#). Some good YouTube videos introducing VHDL basics can be found [here](#).

Documentation and resources for the DE-10 Standard board can be found at:

<https://rocketboards.org/foswiki/Documentation/DE10Standard>

II. VHDL Code Modification and ModelSim Simulation

In this section, you will first modify the given VHDL code to implement the needed logic. You will then create a testbench based on the Prelab testbench file and simulate your new VHDL code in ModelSim.

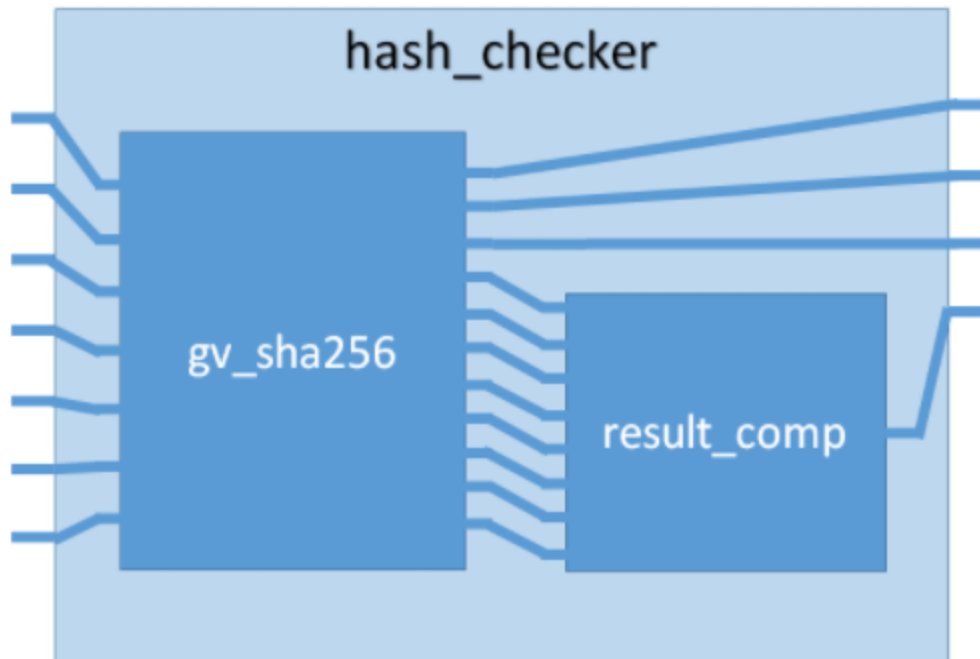
To better understand the structure of the VHDL code, you are given the hierarchy of the provided VHDL files below:

- hash_checker_test.vhd -- testbench (for simulation only). You will modify the Prelab version of this file
 - hash_checker.vhd -- The new top level design file
 - result_comp.vhd -- The hash comparator result component instantiated in the top level
 - gv_sha256.vhd -- SHA256 component instantiated in the top level
 - sha256_control.vhd
 - sha256_padding.vhd
 - sha256_msg_sch.vhd
 - sha256_hash_core.vhd
 - sha256_regs.vhd
 - sha256_Kt_rom.vhd
 - sha256_Ki_rom.vhd

HINT: If you need more details about the GV_SHA256 design, you can find the VHDL project on OpenCores at the following [link](#).

In this section, you will modify 3 of the VHDL files shown in the file design hierarchy above, namely: *result_comp.vhd*, *hash_checker.vhd* and *hash_checker_test.vhd*. All 3 design files are provided in the lab files. The entity sections in all 3 files have been written for you. Please do not modify these sections in the VHDL code. You will be responsible for modifying and adding to the architecture sections of these files.

Recall from Prelab that the major problem that we faced when implementing the SHA256 module was an excessive number of I/O pins. To circumvent this problem, we will embed the SHA256 module (*gv_sha256*) along with a new comparator module (*result_comp*) inside a new top-level file (*hash_checker*). The picture diagram below helps explain our new design.



Looking at the new design, you can realize that the number of I/O pins used in the top level has significantly decreased. Remember that the signals routed into the `result_comp` module are the 256-bit signals holding the generated hash value.

III.A. Implementing the Comparison Logic

We will first start by modifying the newly generated module (`result_comp`). Open `result_comp.vhd` and carefully read the comments in the file. The comments will guide you through as you write up your VHDL code. Please make sure to keep the entity section intact. Your modifications in this file should be limited to the architecture section. **Do not forget to COMMENT your code. In your written report, summarize the modifications/additions you have done to this file, including any VHDL statements you used.**

Your implemented logic in this file should implement the following behavior. Your code should read in the hash value coming as an input to this module and it should compare the value to a certain set of values (the values are present in the comments in the file). If the input hash value matches any of the valid hashes, you should assert an output signal (set its value to a logic '1'), otherwise the output signal should be de-asserted.

III.B. Implementing the Top-Level Logic

Second, we will modify the top-level file (`hash_checker.vhd`). The comments in the VHDL file will guide you through the needed modifications and additions. Please make sure you do not modify the entity section of this file too. The modifications in this file should be limited to the architecture section. Specifically, you should be instantiating the two modules that will be

needed in this design and connecting their input and output pins. **Do not forget to COMMENT your code. In your written report, summarize the modifications/additions that you have done to this file including any VHDL statements that you used.**

III.C. Modifying the Testbench

Finally, you will modify the Prelab test bench to match the new top-level design. Open the testbench file `hash_checker_test.vhd`. Notice that the logic inside this test bench has not been changed from the one used in prelab. Your job is to modify it as needed for this lab. Feel free to change any line in this file. Remember the major change in terms of running the simulation using the new testbench is that in this lab, we are no longer checking the generated hash value against an expected value in the testbench. All we are interested in is to check whether the comparison result generated by the *result_comp* module is indicating a valid hash or not. Specifically, your testbench should include an assert statement checking the expected behavior of the comparison result signal (*expected_behavior_o*). **Do not forget to COMMENT your code. In your written report, summarize the modifications/additions that you have done to this file including any VHDL statements that you used.**

III.D. Simulating and Verifying Your New VHDL Design

To simulate your new code, follow the same procedure that we used in prelab.

- Start ModelSim, create a project and add your design and testbench files to the project.
- Compile the design files and check for any errors. If you have any syntax errors, go back to the VHDL modification stage of this lab, and fix the errors.
- Next, start the simulation using your modified testbench file.
- Add the testbench signals and the top-level signals of the design to the wave window and to a VCD dump file.
- Run the simulation and examine the results in the transcript and wave windows. Make sure your new design is behaving as expected.

Remember, you should only be looking at the output signals when the signal "dut_do_valid" is asserted.

- Now that you have run the simulation, export your results in the form of two waveform images. Make sure the first wave window shows the results between 0 ns and 1500 ns. Remember to use the Zoom Range option and the Export Image tool that we used in prelab. For the second wave window, show the results between 1500 ns and 3000 ns. **Include both images in your submission.**
- Finally, end the simulation and correctly save the results in the VCD file by clicking on: Simulation --> End Simulation. **Submit the generated VCD dump file electronically.**
- We are done with VHDL simulation. You can close the ModelSim window.

III. VHDL Code Modification for the DE-10 Board

Now that you are done with simulation, we will synthesize and implement the *hash_checker* file to work on the DE-10 Board using Quartus. This will require the new file hierarchy as seen below:

- DE_10_hash_checker.vhd -- The new top level design file. You will now modify this file
 - hash_checker.vhd
 - result_comp.vhd -- The hash comparator result component
 - gv_sha256.vhd -- SHA256 component instantiated in the top level
 - sha256_control.vhd
 - sha256_padding.vhd
 - sha256_msg_sch.vhd
 - sha256_hash_core.vhd
 - sha256_regs.vhd
 - sha256_Kt_rom.vhd
 - sha256_Ki_rom.vhd

The DE_10_hash_checker.vhd file is found in the “DE_10 Testbench” folder of the provided lab files. You will now modify this file.

Previously we utilized a test bench (essentially software) to test the implementation of the SHA-256 function. Now we want the design to function as a standalone unit in hardware on the DE-10 Board. To this aim the DE_10_hash_checker.vhd file consists of a simple state machine to coordinate the loading of four predefined input words. The file additionally will route status signals to LEDs on the DE-10 board and take switch positions as input to decide which of the four predefined input words will be utilized for the hash generation.

The provided DE_10_hash_checker.vhd file has a basic framework, but you are not required to utilize the code already provided if you do not want to. If you desire, you can separate the state machine into a separate entity. If you do not have experience with creating state machines, I highly recommend you just use the provided framework.

Information on making a state machine in VHDL can be found here:

[VHDL Templates for State Machines \(intel.com\)](#)

[How to create a Finite-State Machine in VHDL - VHDLwhiz](#)

Advice on how to approach the state machine generation is to examine the hash_checker_test.vhd file (or the testbench from prelab). The first test vector (~lines 142 to 163) approximates the expected appearance of the required states.

```
test_case <= 1;
dut_ce <= '0';
dut_di <= (others => '0');
dut_bytes <= b"00";
dut_start <= '0';
dut_end <= '0';
dut_di_wr <= '0';
wait until pclk'event and pclk = '1';
dut_ce <= '1';
dut_start <= '1';
dut_di <= x"61626300";
dut_bytes <= b"11";
wait until pclk'event and pclk = '1';
dut_start <= '0';
dut_di_wr <= '1';
if dut_di_req = '0' then
    wait until dut_di_req = '1';
end if;
dut_end <= '1';
wait until pclk'event and pclk = '1';
dut_end <= '0';
dut_di_wr <= '0';
```

All the inputs are at '0' initially. On the next clock cycle (i.e., after the "wait until" line), four different signals are written. Then on the next clock cycle, two inputs are written. There is then a branch and the test_bench loops until it receives a specific input. All this can be directly translated into a state machine.

Common Errors and Issues (assuming you use the two-process state machine template provided in DE10_Hash_Checker.vhd):

- You must incorporate a final hold state in the state machine (in other words, the final state where the behavior of the LED, on/off, should be held, do not go back to the initial state)
- When conditioning on hash_selector_switch to determine the value of di_i, also assign the correct value to bytes_i, since each of the 4 messages has a different value for bytes_i
- Sometimes, Quartus incorrectly categorizes signals as clocks. In the past, we have seen Quartus classify the reset signal and the state variables (s0,...s3) as clocks. To address this, use the assignment editor to assign the "not a clock" option to the pins corresponding to those variables that were incorrectly categorized as clocks. Also, in the

future steps, the Timing Analyzer is used to generate a .sdc file for the clock. Make sure that this .sdc file does not contain references to non-clock signals.

- When debugging your state machine on the DE10 board, it may help to use a very slow clock that allows for real-world debugging of each state (a clock period of 1-2s). Instructions on how to change the clock period are in the subsequent portions of this assignment.
- When debugging individual states, it may help to illuminate various LEDs on the board corresponding to each state. This way, you will know exactly what state the state machine is in. LED pin assignments:

Table 3-8 Pin Assignment of LEDs

<i>Signal Name</i>	<i>FPGA Pin No.</i>	<i>Description</i>	<i>I/O Standard</i>
LEDR[0]	PIN_AA24	LED [0]	3.3V



LEDR[1]	PIN_AB23	LED [1]	3.3V
LEDR[2]	PIN_AC23	LED [2]	3.3V
LEDR[3]	PIN_AD24	LED [3]	3.3V
LEDR[4]	PIN_AG25	LED [4]	3.3V
LEDR[5]	PIN_AF25	LED [5]	3.3V
LEDR[6]	PIN_AE24	LED [6]	3.3V
LEDR[7]	PIN_AF24	LED [7]	3.3V
LEDR[8]	PIN_AB22	LED [8]	3.3V
LEDR[9]	PIN_AC22	LED [9]	3.3V

- It may help to include di_req_o in the sensitivity list of the second process. In other words, the declaration for the process should look like process (state, di_req_o).

Your final file will utilize the following IO pins on the DE-10 board.

- Clock signal
- Two switches to choose from four different hash inputs (Switches in positions 00, 01, 10, 11)

- One push button to start a new hash function
- One push button as a general reset
- One LED to indicate the hash is valid

You should utilize the following pins in the pin planner:

- Clk: PIN_AF14
- Switches: PIN_AB30 and PIN_Y27. These are SW0 and SW1, the two rightmost switches when viewing the board with the switches on the bottom.
- Push Buttons: PIN_AJ4 and PIN_AA15. These are KEY0 and KEY3, the rightmost and leftmost button respectively when viewing the board with the buttons on the bottom. Be aware that the buttons are **ACTIVE LOW**.
- LED: PIN_AA24. This is the rightmost LED, immediately above the switches when viewing the board with the switches on the bottom.

To summarize, the desired functionality of the file is as follows:

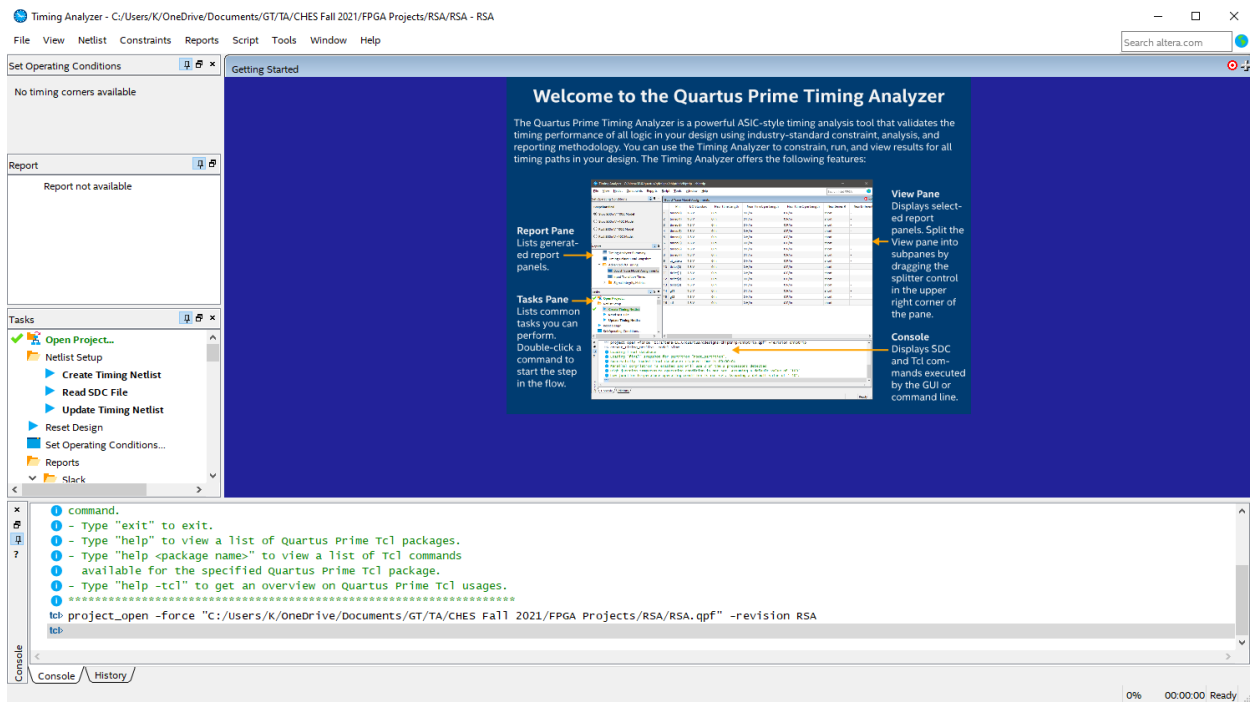
- When you press the start button, one of four hashes are calculated, based on the positions of two switches. If the calculated hash is valid in accordance with the hash checker, one LED turns on, otherwise the LED turns off. The LED should stay in that status until you start a new hash calculation.

IV. Obtaining Timing Results

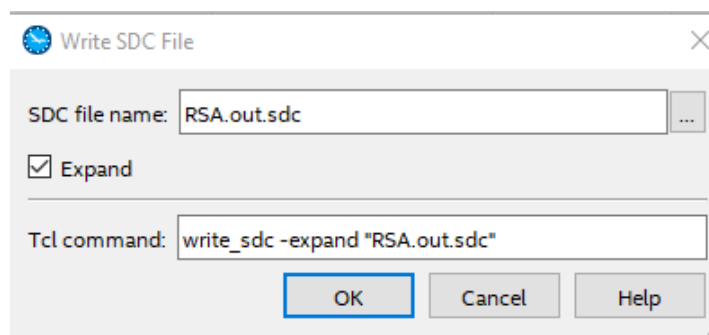
With your design completed obtain timing results as follows:

Create a project in Quartus as in prelab, with the DE-10_hash_check file as the top-level file.

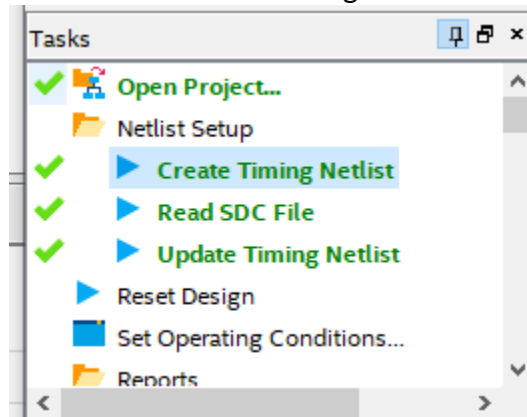
- Make sure you choose the correct FPGA as your target device.
- Next compile design with the default options and wait for the task to be completed.
- Now open **Tools -> Timing Analyzer**. You will see the below screen



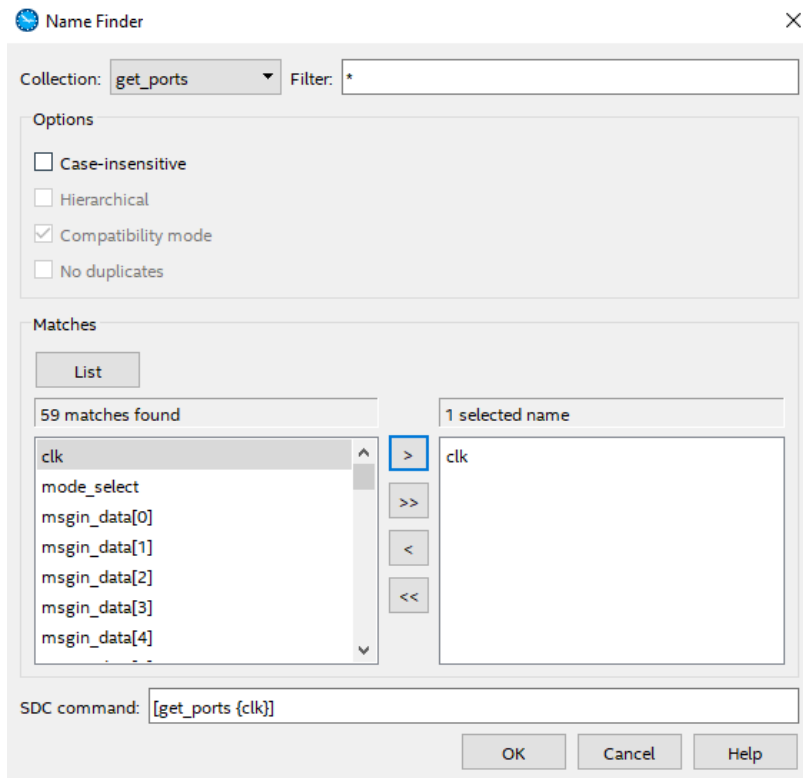
- Double click **“Create Timing Netlist”** on the left under the task bar. It should turn green.
- Run **Constraints -> Write SDC File**. Press **“OK”** on the opened dialogue box.



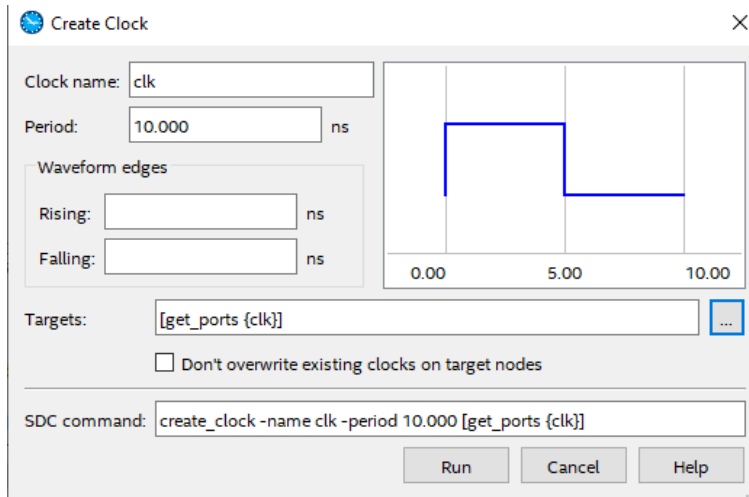
- You should now see the following in the Tasks window



- Now run **Constraints -> Create Clock**. In the pop-up window, put “clk_i” for the Clock Name and hit the “...” next to “Targets”. On the new pop up window click “list” and move “clk_i” to the right as in the example shown below. Hit OK.



- Now hit run in the below window



- Double click “Update Timing Netlist” in the Tasks window of the Timing Analyzer.
- Back in the main Quartus screen, click the arrow next to Timing Analysis in the Tasks window and open “Edit Settings”. In the Settings window, click the “...” and add the .out.sdc file you just created to the project.
- Now re-run the full Compilation task in Quartus. After the compilation is complete look for the Timing Analysis results. The results can be seen in the “Timing Analyzer” section of the Compilation Report
- Next, find the best clock period at which your new design runs by changing your clock constraint. **Report the new timing values shown in the “Clocks” tab inside “Timing**

Analyzer” in the Compilation Report.

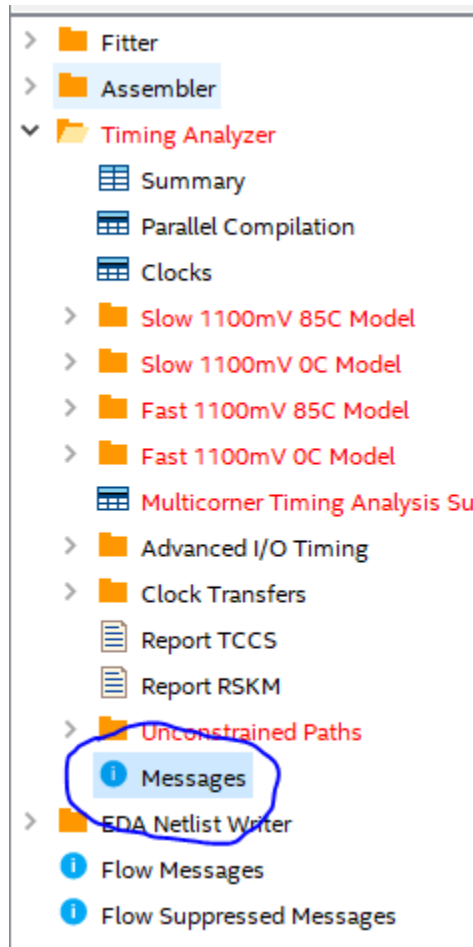
The screenshot displays the 'Compilation Report - RSA' window. On the left is a 'Table of Contents' pane with a tree view. The 'Timing Analyzer' folder is expanded, showing sub-items like 'Summary', 'Parallel Compilation', 'Clocks', and four models: 'Slow 1100mV 85C Model', 'Slow 1100mV 0C Model', 'Fast 1100mV 85C Model', and 'Fast 1100mV 0C Model'. These four models are highlighted in red, indicating they failed timing requirements. The 'Flow Summary' pane on the right shows the following details:

Flow Summary	
Flow Status	Successful - Mon Jan 24 18:01:37 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	RSA
Top-level Entity Name	rsa_core_top
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	373 / 41,910 (< 1 %)
Total registers	268
Total pins	59 / 499 (12 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

- In the above example, you can see that the four Models (Slow 85C, Slow 0C, Fast 85C, Fast 0C) are in the red. This is due to failing to meet timing requirements. They will be in black when the timing constraints are met.
- Adjust the clock frequency and rerun the Timing Analysis to try to find the maximum clock frequency. The clock frequency for the Timing Analysis can be adjusted by editing the SDC file created by the Timing Analyzer. This file is located in your project folder with an .out.sdc extension. In the file you will see the following line.

```
38 #*****
39 # Create Clock
40 #*****
41
42 create_clock -name {clk} -period 1.000 -waveform { 0.000 0.500 } [get_ports {clk}]
43
44
```

- The 1.000 after period is the time period in ns. The numbers in the brackets define when the clock changes from '1' to '0' and '0' to '1'. To change this value to something else, such as 5.000, the new line will be
 - “..... -period 5.000 -waveform { 0.000 2.5000 }”
- Save the file. You can now rerun just the “Timing Analyzer” task without needing to redo full compilation.
- Report the fastest clock achieved and take a screenshot showing the contents of the Messages tab of the “Timing Analyzer” under the compilation report.



Lab Report

Please type all your answers and include a cover sheet with your first and last name, GT ID number, and GT username.

Code Modification and Simulation Section:

1. All design and simulation files including commented and modified code (VHDL files).
2. Code modifications and/or additions summary of all 4 design files.
3. The two exported waveform images of the simulated design.
4. The VCD dump file of the simulated design.

Synthesis and Implementation:

1. Fastest clock frequency achieved.
2. "Messages" output from the Timing Analyzer
3. Resource Usage Summary for the project with DE-10_hash_checker.vhd as the top level.

Canvas Submission of Lab 1 Files

1. Place all files into a single folder and submit your work on Canvas.