

Atum-A3-Nano

My First FPGA



**Powering next-level performance for
digital logic and embedded applications !**

| | |
|---|-----------|
| Chapter 1 Hardware Design | 3 |
| 1.1 Design Flow | 3 |
| 1.2 Before You Begin..... | 3 |
| 1.3 What You Will Learn | 5 |
| Chapter 2 Assign FPGA Device | 6 |
| 2.1 Assign the Device | 6 |
| Chapter 3 Design Entry | 9 |
| 3.1 Create a Simple Counter | 9 |
| 3.2 Add a PLL Megafunction..... | 13 |
| 3.3 Create a Multiplexer | 19 |
| 3.4 Instance the Counter, PLL and Multiplexer | 20 |
| 3.5 Assign the Pins..... | 22 |
| Chapter 4 Compile and Verify Design | 26 |
| 4.1 Compile Your Design..... | 26 |
| 4.2 Program the FPGA Device..... | 29 |
| 4.3 Verify the Hardware..... | 32 |
| Additional Information | 33 |

Chapter 1

Hardware Design

This tutorial provides comprehensive information that will help you understand how to create a FPGA design and run it on your Atum A3 Nano board. The following sections provide a quick overview of the design flow, explain what you need to get started, and describe what you will learn.

1.1 Design Flow

Figure 1-1 shows the FPGA design flow block diagram. The standard FPGA design flow starts with design entry using schematics or a hardware description language (HDL), such as Verilog HDL or VHDL. In this step, you can create Verilog HDL file and use existed IP to implement the design. The flow then proceeds through compilation, simulation, programming, and verification in the FPGA hardware.

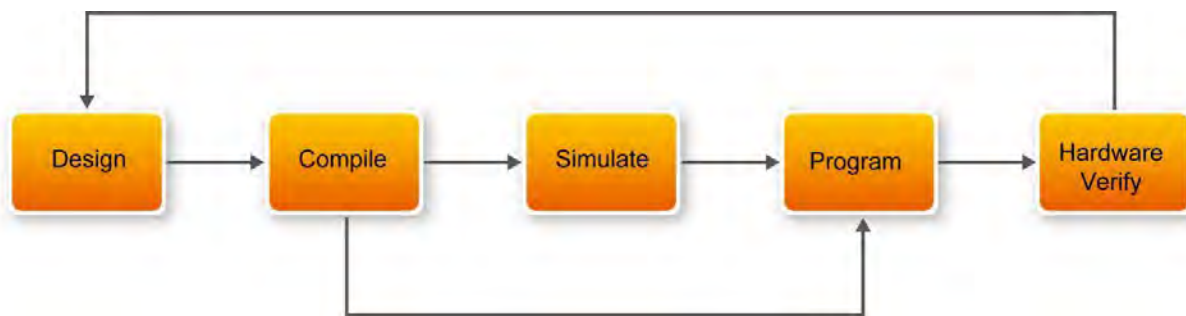


Figure 1-1 Design Flow

This tutorial guides you through all the steps except for simulation. Although it is not covered in this document, simulation is very important to learn, and there are entire applications devoted to simulating hardware designs. There are two types of simulation, Functional simulation and Timing simulation. Functional simulation allows you to verify that your code is manipulating the inputs and outputs appropriately. Timing (or post place-and-route) simulation verifies that the design meets timing and functions appropriately in the device.

1.2 Before You Begin

This tutorial assumes the following prerequisites:

- You generally know what a FPGA is. This tutorial does not explain the basic concepts of programmable logic.
- You are somewhat familiar with Verilog HDL software development.

- You have installed the Quartus Prime Pro 25.1 software on your computer. If you do not have the Quartus Prime Pro software, you can download it from Intel web site at this [link](#).
- You have a Atum A3 Nano board on which you will test your project. Using a development board helps you to verify whether your design is really working.
- You have gone through the Quick Start Guide and/or the Getting Started Guide for your development kit. These documents ensure that you have:
 - **Installed the required software.**
 - **Determined that the development board functions properly and is connected to your computer.**

Next step you need to confirm the USB-Blaster III driver is installed successfully. Use the Type-C USB cable to connect the leftmost Type-C USB connector on the Atum A3 Nano board to a USB port on a computer that runs the Quartus Prime Pro 25.1 software. Plug in the 5V adapter to provide power to the board.

Power on the board, then the computer will recognize the new hardware connected to its USB port as shown in **Figure 1-2**.

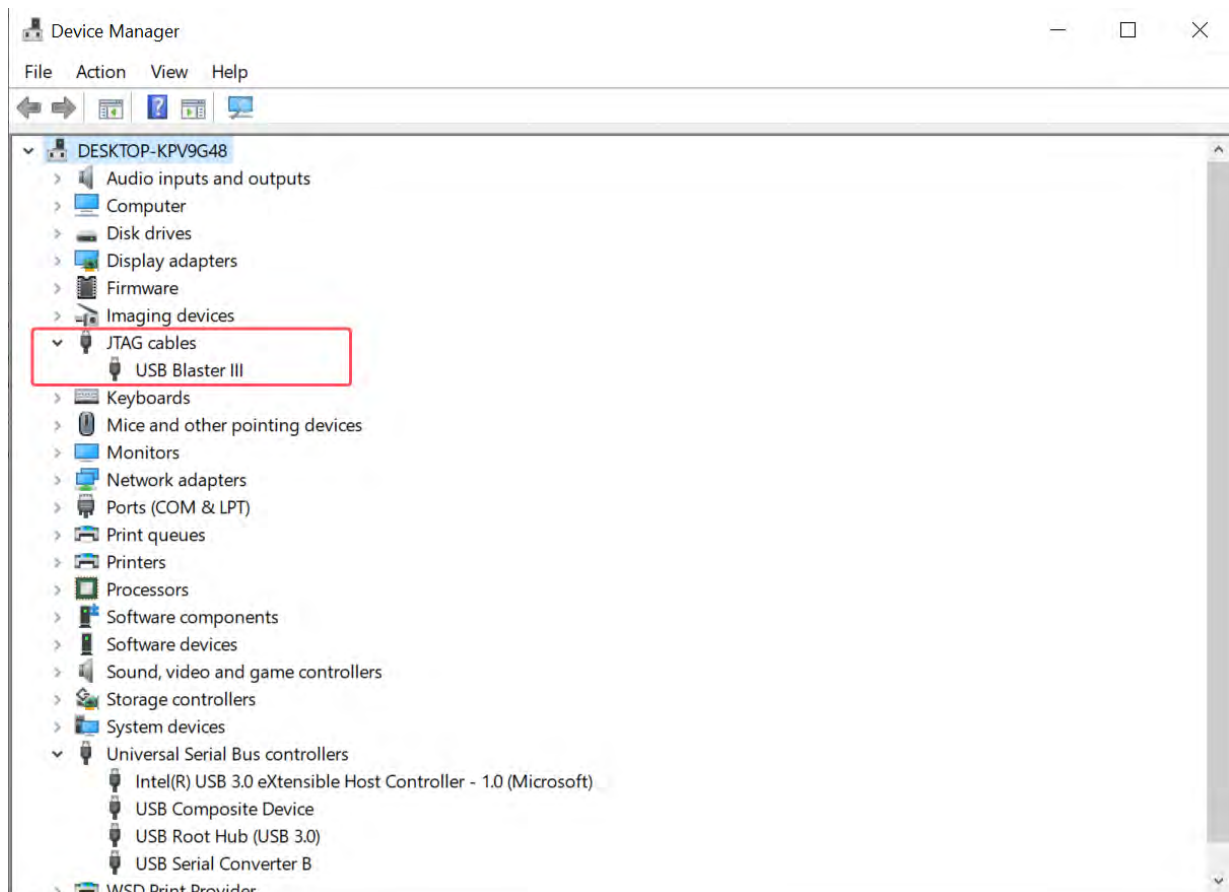


Figure 1-2 USB Blaster III shown in PC Device Manager

1.3 What You Will Learn

In this tutorial you will perform the following tasks:

Create a design that causes LEDs on the development board to blink at a speed that is controlled by an input KEY. This design is easy to create and gives you visual feedback that the design works. Of course, you can use your Atum A3 Nano board to run other designs as well. For the LED design, you will write Verilog HDL code for a simple 32-bit counter and a 2-input multiplexer, add a phase-locked loop (PLL) IP as the clock source. When the design is running on the board, you can press an input KEY to multiplex the counter bits that drive the output LEDs.

Becoming familiar with Quartus Prime Pro design tools—This tutorial will not make you an expert, but at the end, you will understand basic concepts about Quartus Prime Pro projects, such as entering a design using Verilog HDL, compiling your design, and downloading it into the FPGA on your Atum A3 Nano board.

Develop a foundation to learn more about FPGAs—For example, you can create and download digital signal processing (DSP) functions onto a single chip, or build a multi-processor system, or create anything else you can imagine all on the same chip. You don't have to scour data books to find the perfect logic device or create your own ASIC. All you need is your computer, your imagination, and a Terasic Atum A3 Nano board.

Chapter 2

Assign FPGA Device

You begin this tutorial by creating a new Quartus Prime Pro project. A project is a set of files that maintain information about your FPGA design. The Quartus Prime Pro Settings File (.qsf) and Quartus Prime Pro Project File (.qpf) files are the primary files in a project. To compile a design or make pin assignments, you must first create a project.

2.1 Assign the Device

1. In the Quartus Prime Pro software, select **File --> New Project Wizard**. The Introduction page opens as shown in **Figure 2-1**.

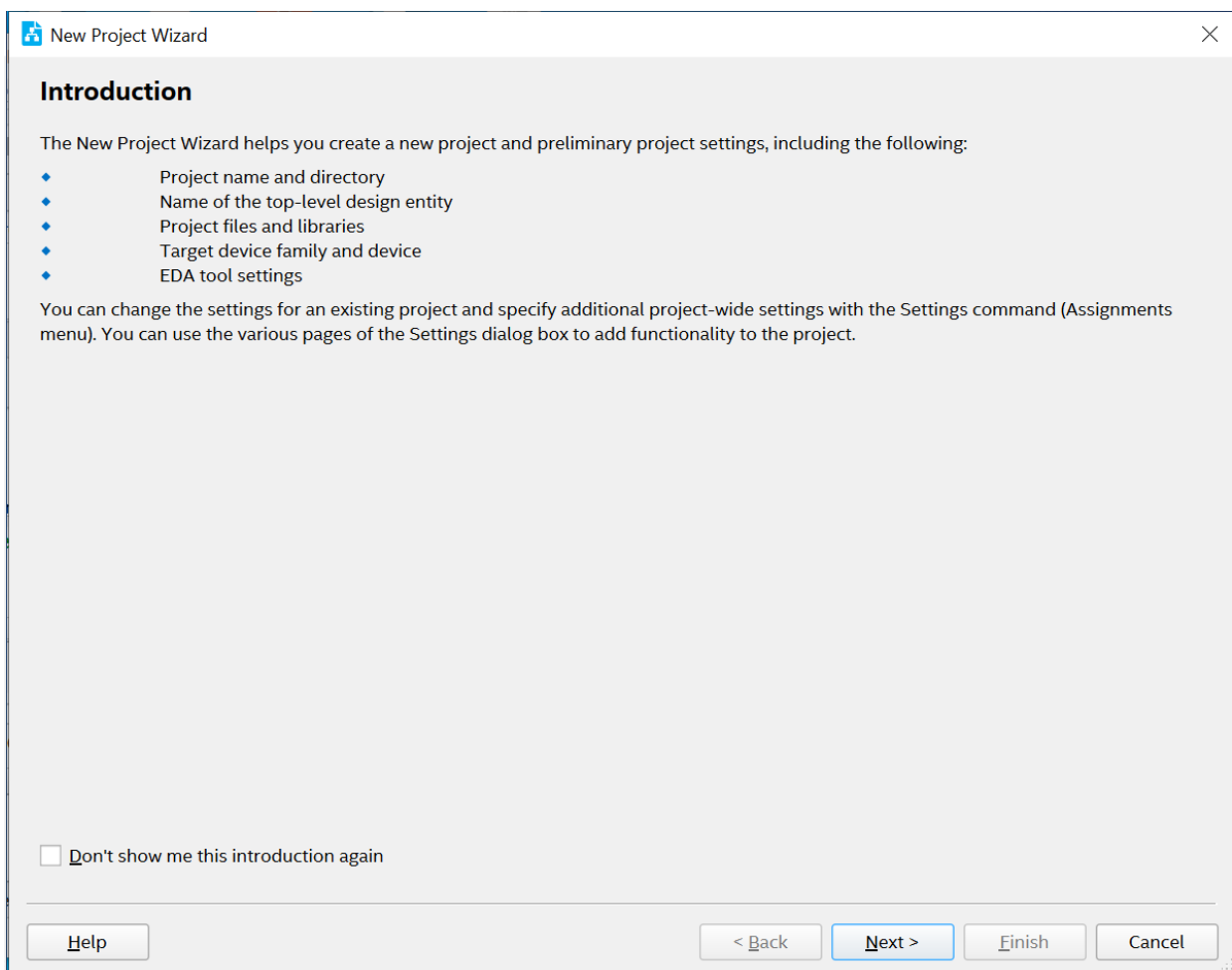


Figure 2-1 New Project Wizard introduction

2. Click the **Next** button, and enter the following information about your project.
 - a. **What is the working directory for this project?** Enter a directory in which you will store your Quartus project files for this design. For example, D:\My_design\my_first_fpga. File names, project names, and directories in the Quartus software cannot contain spaces.
 - b. **What is the name of this project?** Type my_first_fpga.
 - c. **What is the name of the top-level design entity for this project?** Type my_first_fpga.

Figure 2-2 Project information

- d. Click **Next** button, you will assign a specific FPGA device to the design. For Atum A3 Nano board, the FPGA device is A3CZ135BB18AE7S.

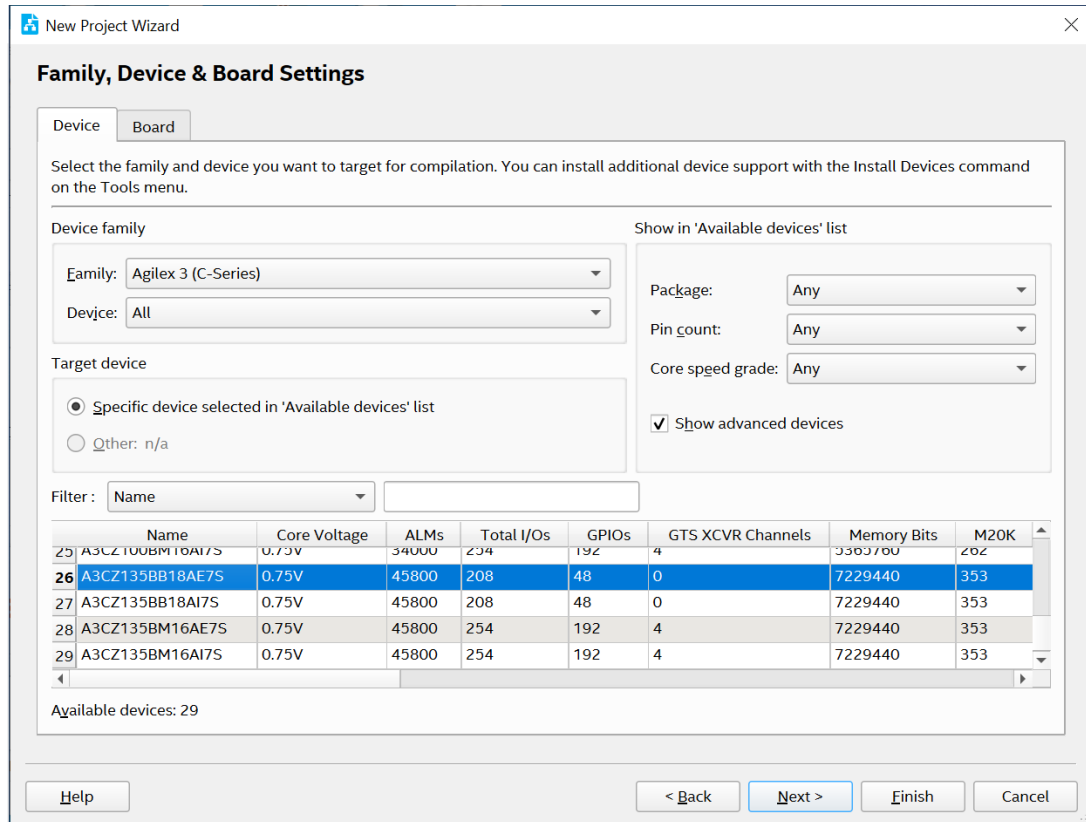


Figure 2-3 Specify the FPGA Device

- e. Click the **Finish** button, create the my_first_fpga project directory. You just created your first Quartus FPGA project, as shown in **Figure 2-4**.

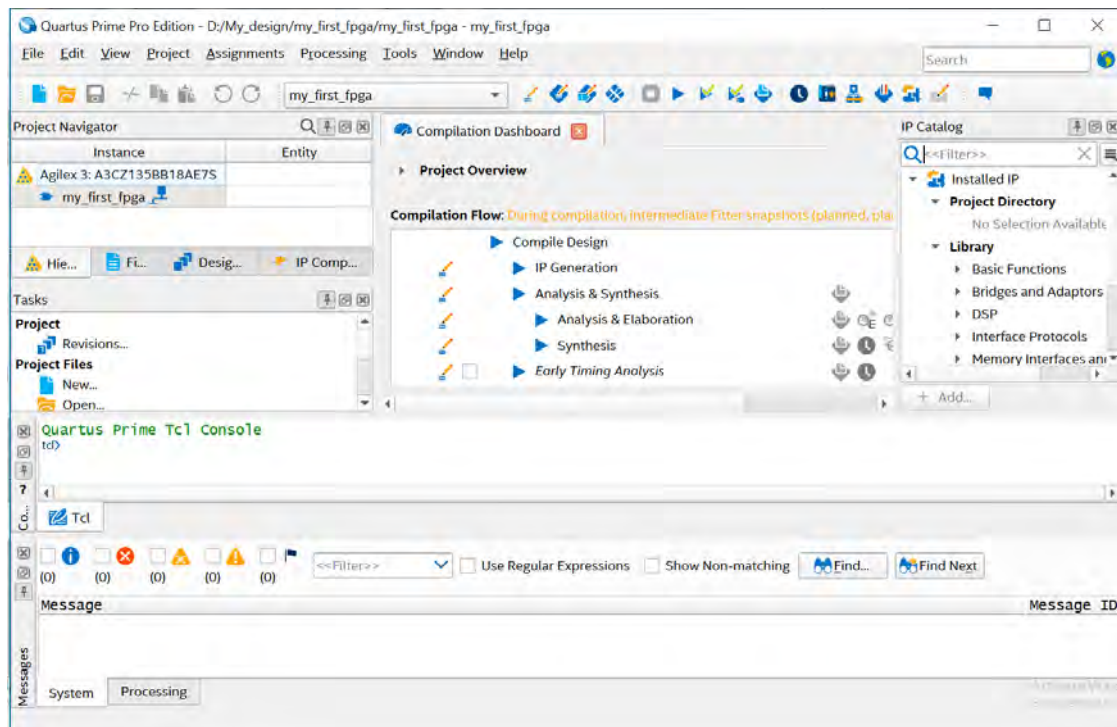


Figure 2-4 my_first_fpga project

In the design entry step you create a Verilog HDL .v file that is the top-level design. You will add library of parameterized modules (LPM) functions and use Verilog HDL code to add two logic blocks. When creating your own designs, you can choose any of these methods or a combination of them.

3.1 Create a Simple Counter

In this section, we will introduce to create a Verilog HDL code which implements a 32-bit simple counter.

1. In the Quartus tools bar, choose **File --> New --> Verilog HDL File**.

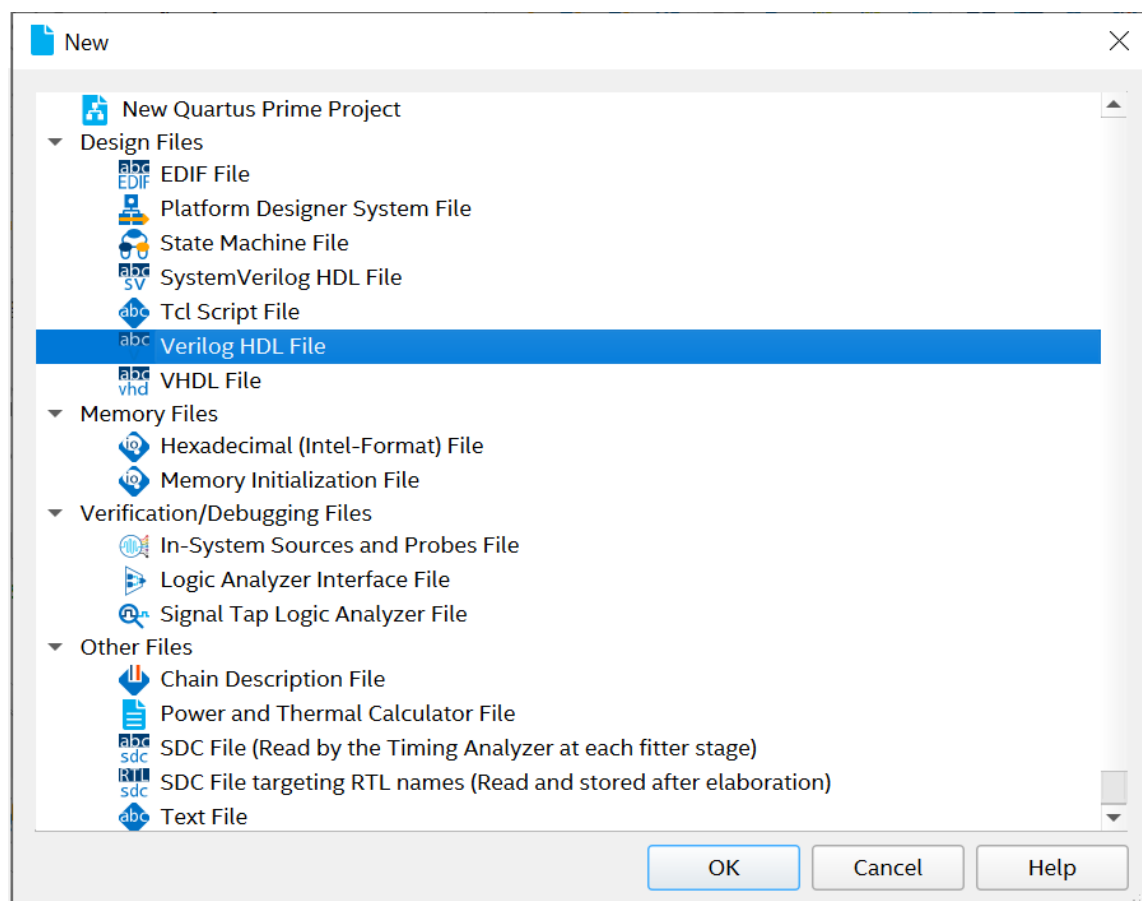


Figure 3-1 Choose Verilog HDL File

2. Click **OK** button to create a new Verilog1.v file. This empty file is ready for you to enter the Verilog HDL code.

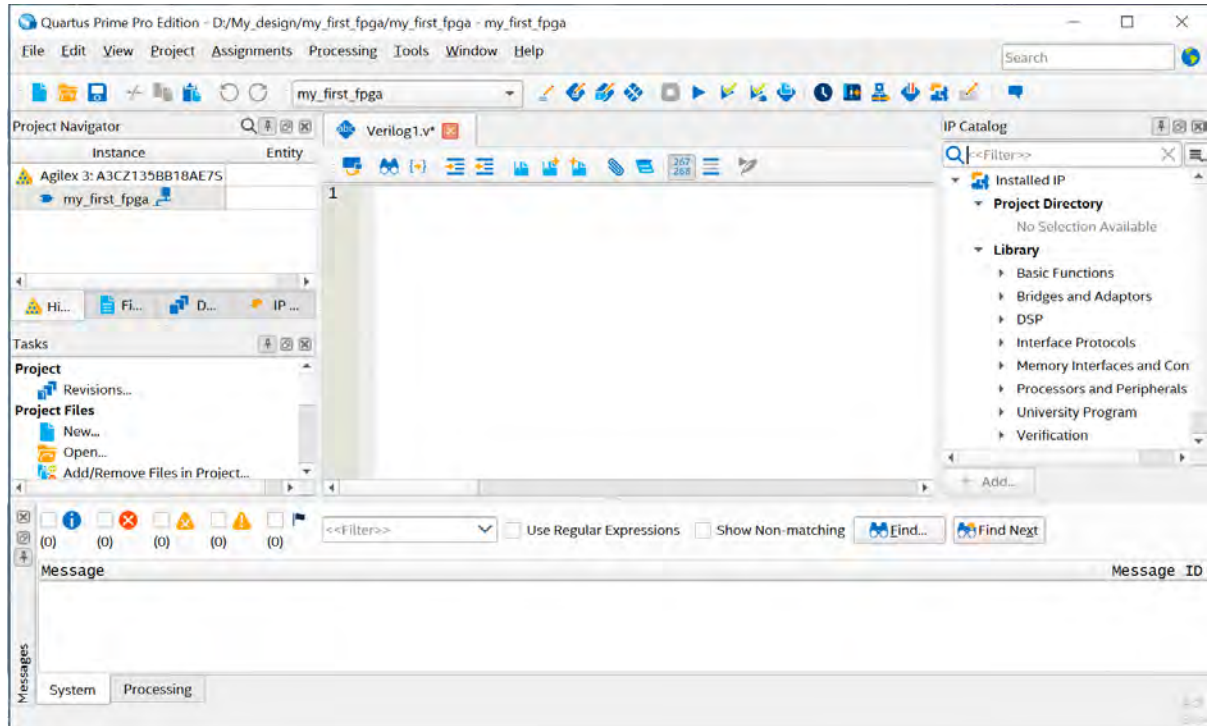


Figure 3-2 Create a new empty .v file

3. Type the following Verilog HDL code into the empty file.

```
//It has a single clock input and a 32-bit output port
module simple_counter (
    input                clk,
    output reg [31:0]    counter_out
);

always @ (posedge clk)    // on positive clock edge
begin
    counter_out <= #1 counter_out + 1;    // increment counter
end

endmodule                // end of module counter
```

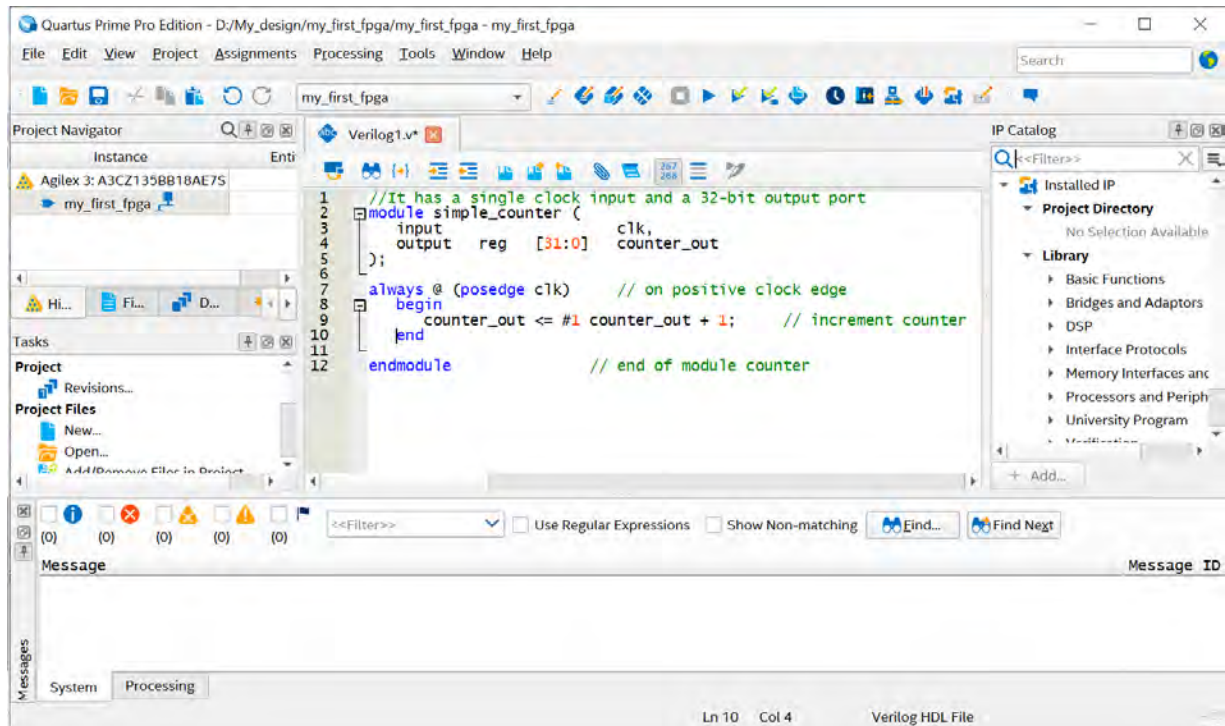


Figure 3-3 The Verilog File of simple_counter.v

4. Save the file by choosing **File --> Save**, in the **Save as** window, click the **New folder** to create a new folder and named it as **v**.

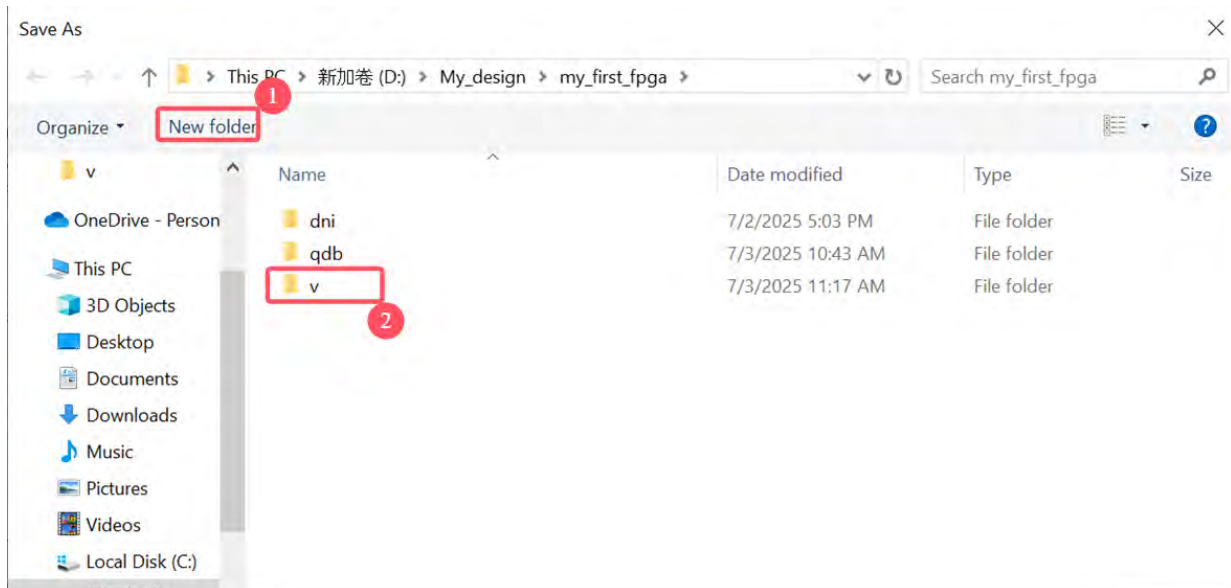


Figure 3-4 Create a new folder

5. Double click the **v** folder to enter it, change the **File name** to **simple_counter.v**, and choose the **Save as type: Verilog HDL File (*.v, *.vlg, *.verilog)**. Click the **Save** button to save this file.

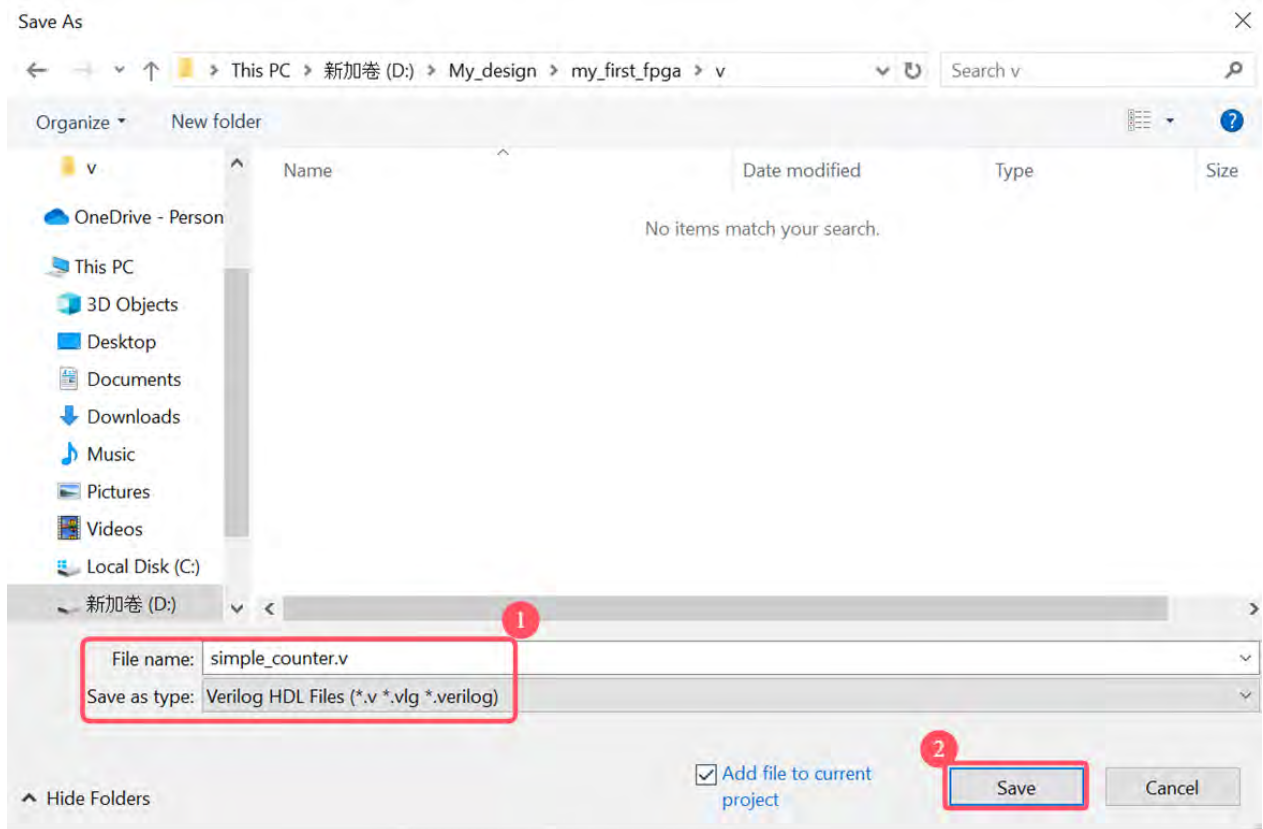


Figure 3-5 Save the Verilog HDL file

6. Now the first Verilog HDL file is created successfully.

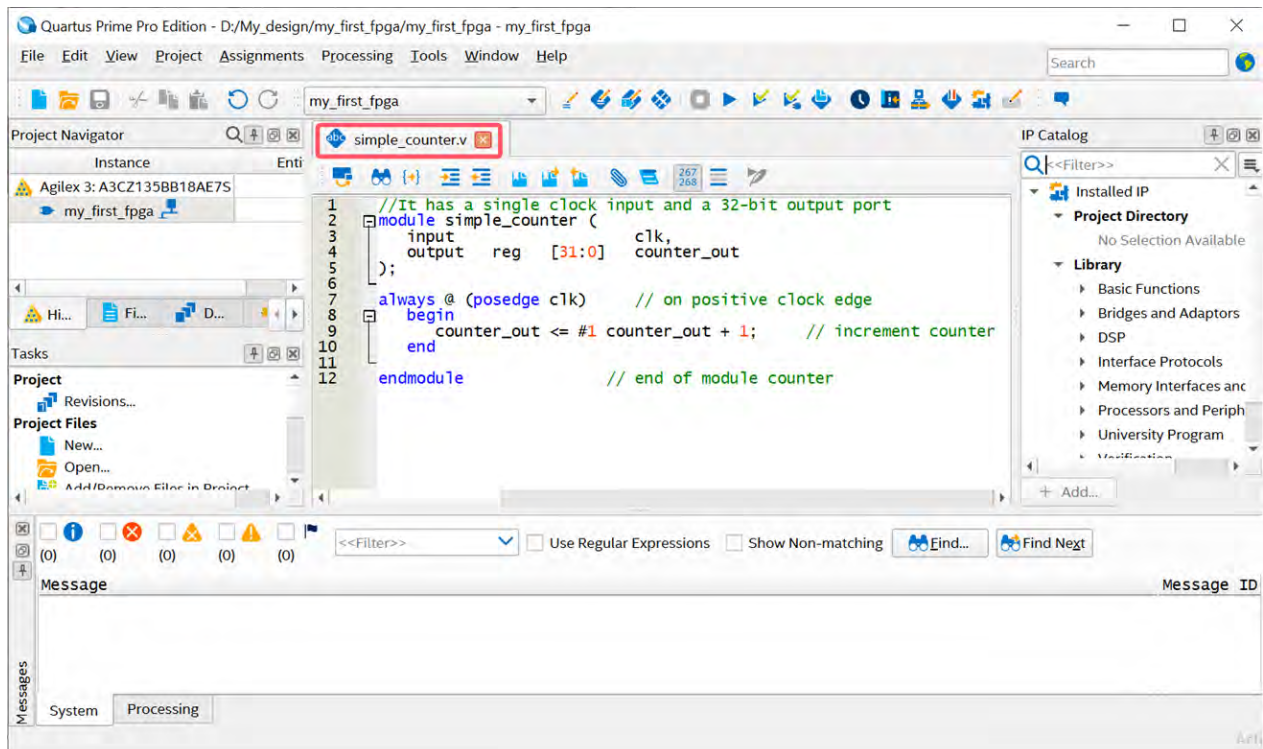


Figure 3-6 Created the simple_counter.v file successfully

3.2 Add a PLL Megafunction

There is an IP catalog which can easily customize and integrate IP cores into your project. Use the IP Catalog and parameter editor GUI to select, customize, and generate files representing your custom IP variation. You can increase efficiency by using a megafunction instead of writing the function yourself. Altera also provides more complex functions, called Megacore functions, which you can evaluate for free but require a license file for use in production designs.

In this section, we introduce to use a PLL clock source to drive the simple counter. A PLL uses the on-board 50MHz oscillator to create a constant clock frequency as the input to the counter. To create the clock source, you will add a pre-built IP core named IOPLL FPGA IP.

1. In the IP Catalog, find the IOPLL FPGA IP in the Library/Basic Functions/Clocks, PLLs and Resets/PLL.

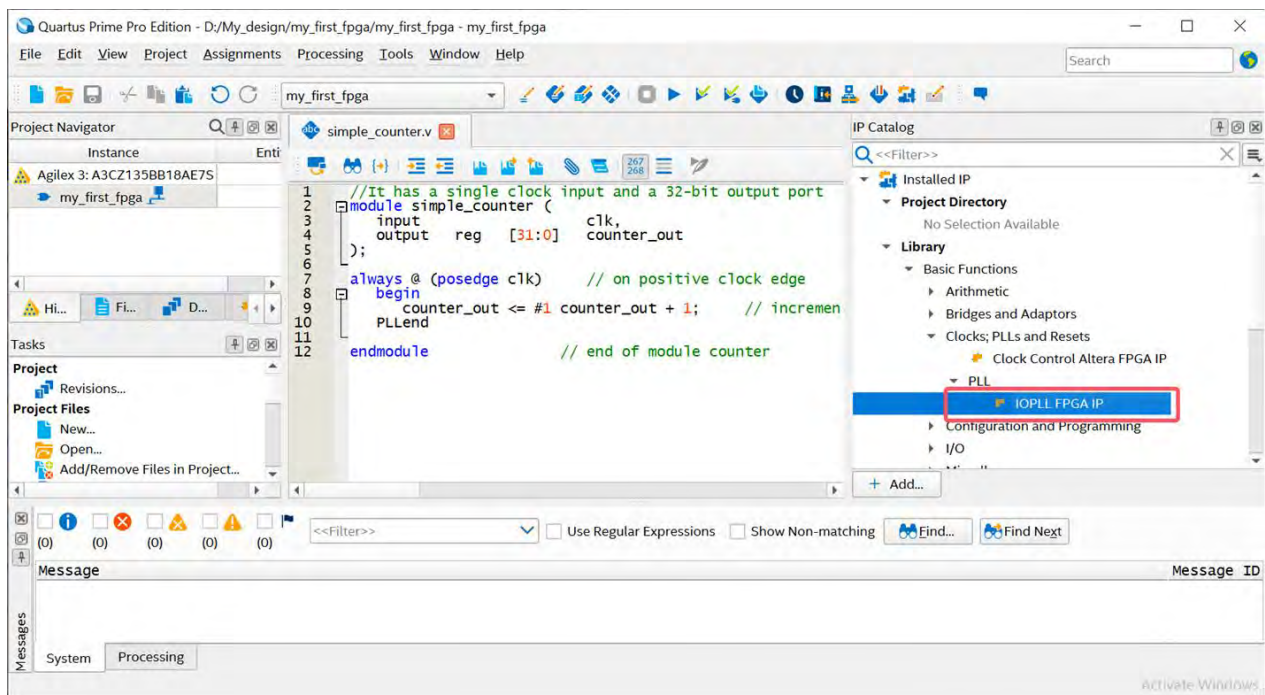


Figure 3-7 Find the PLL IP

2. Double click the IOPLL FPGA IP, a New IP Variant window pops up.

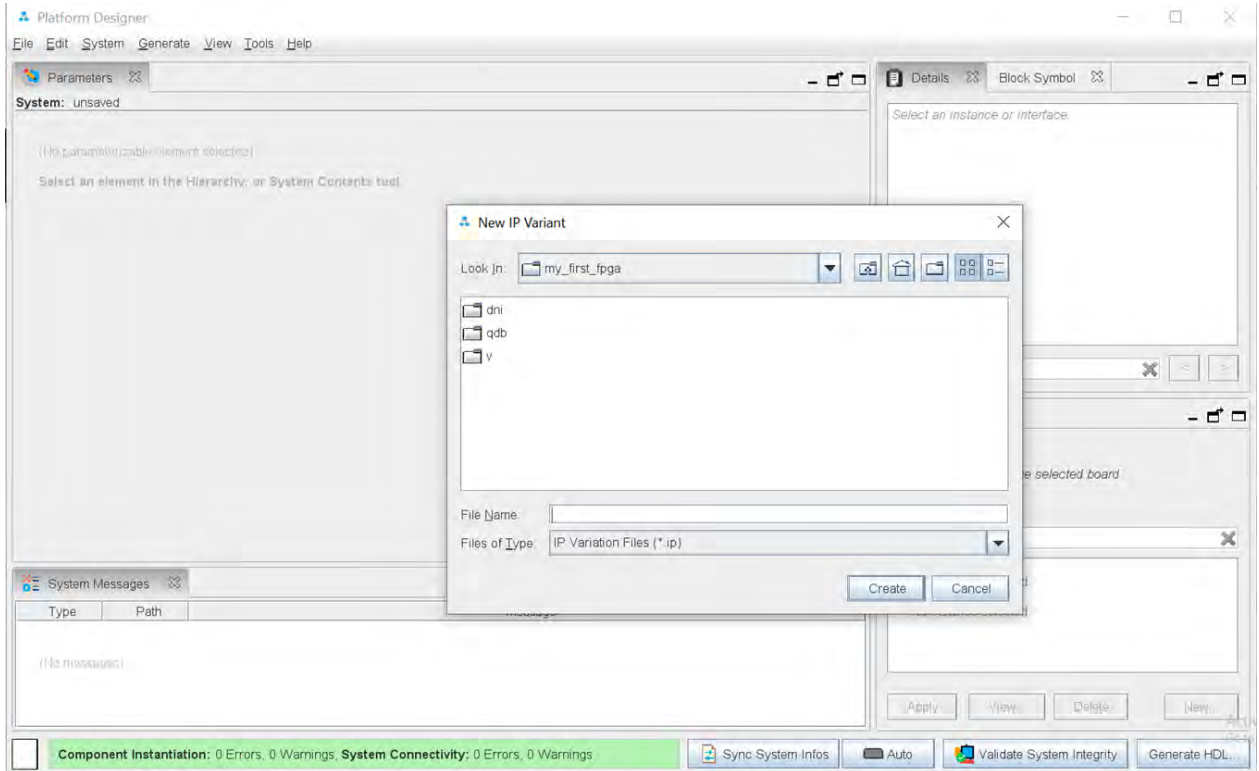


Figure 3-8 New IP Variant window

3. Click the **Create New Folder** icon to create a new folder and named it as **ip**.

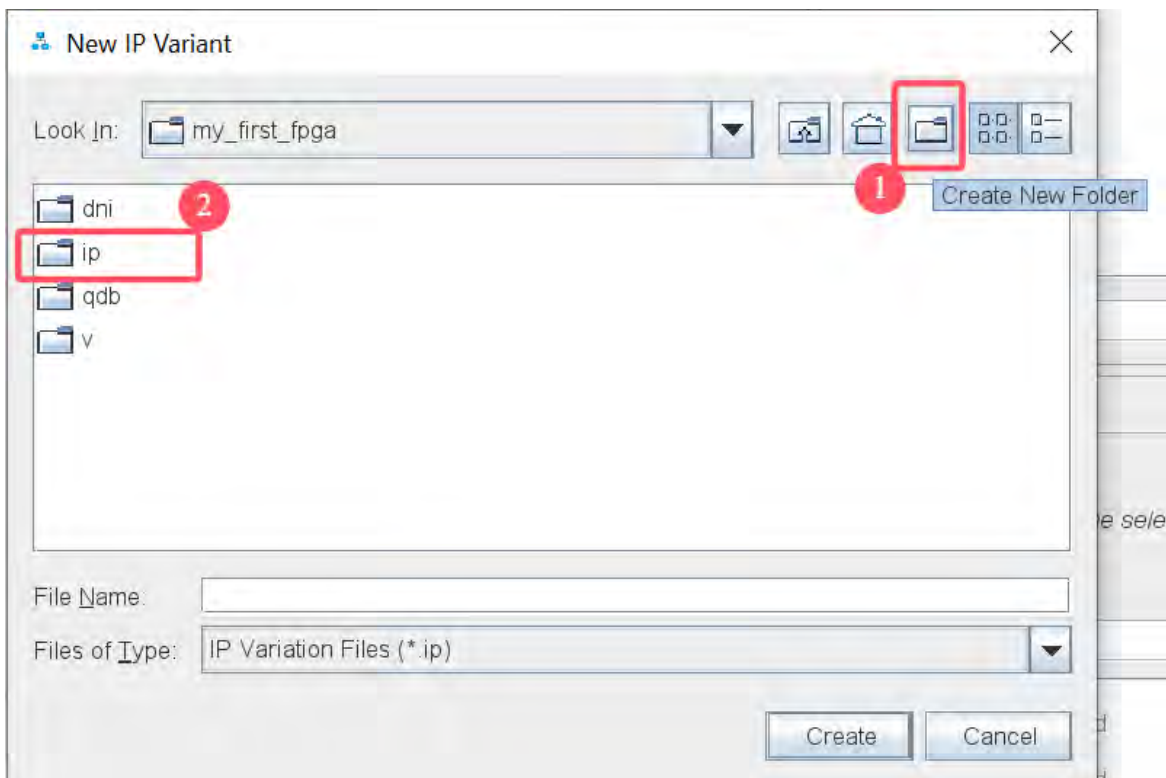


Figure 3-9 Create the new ip folder

- Double click the ip folder to enter it, enter pll to the File name, then click the **Create** button.

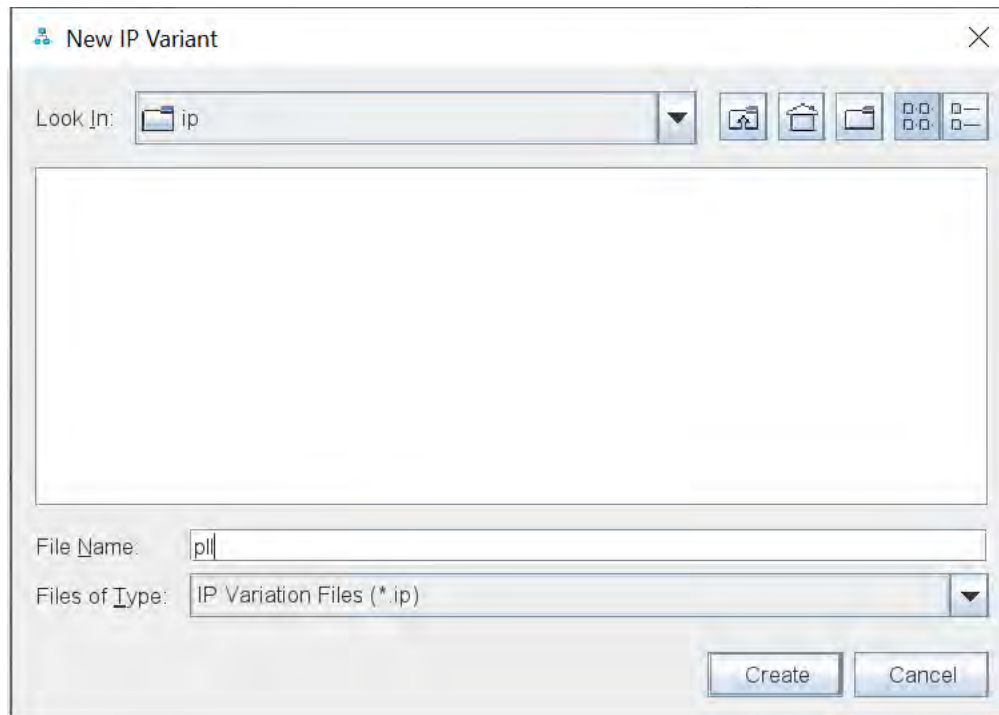


Figure 3-10 Create pll IP

- The IP Parameter Editor window pops up.

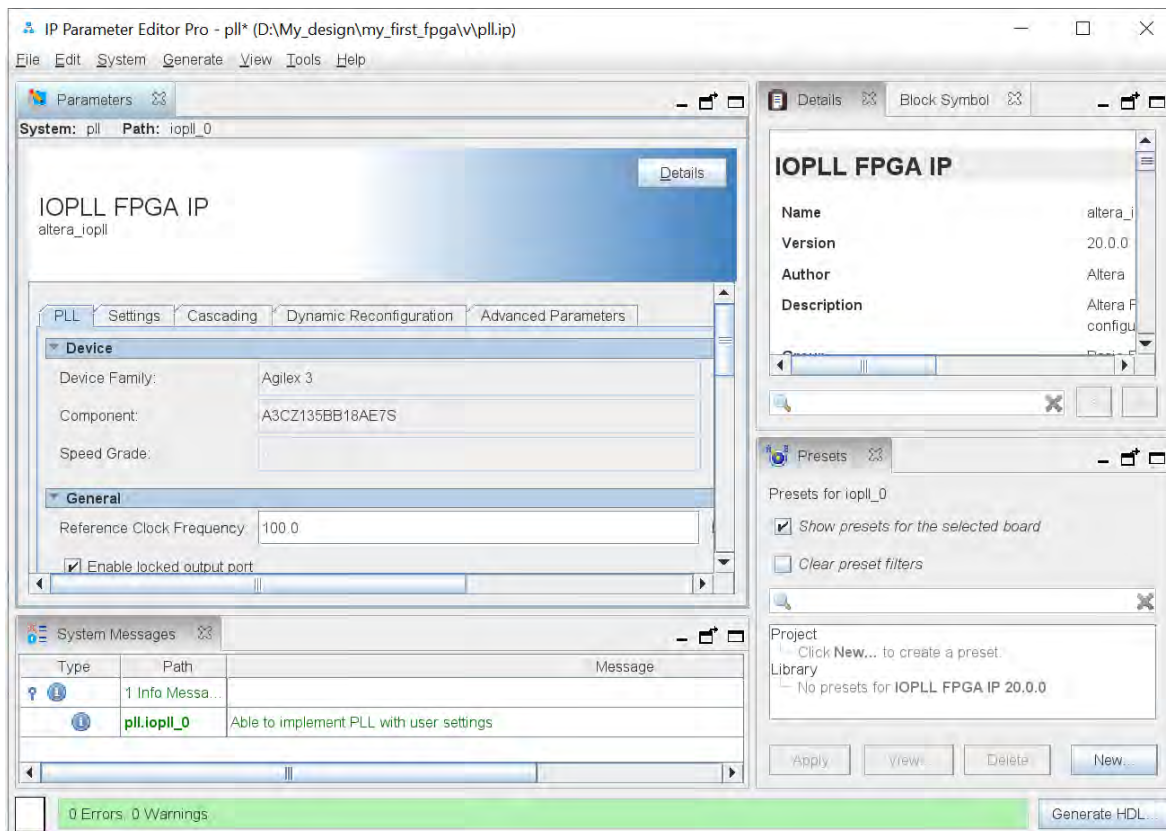


Figure 3-11 PLL IP Parameter Editor window

6. In the PLL tab, modify the parameters below.
 - a. **Reference Clock Frequency:** 50MHz
 - b. Uncheck the Enable locked output port
 - c. **Compensation Mode:** Direct
 - d. **Number of Clocks:** 1
 - e. **Desired Frequency:** 5MHz

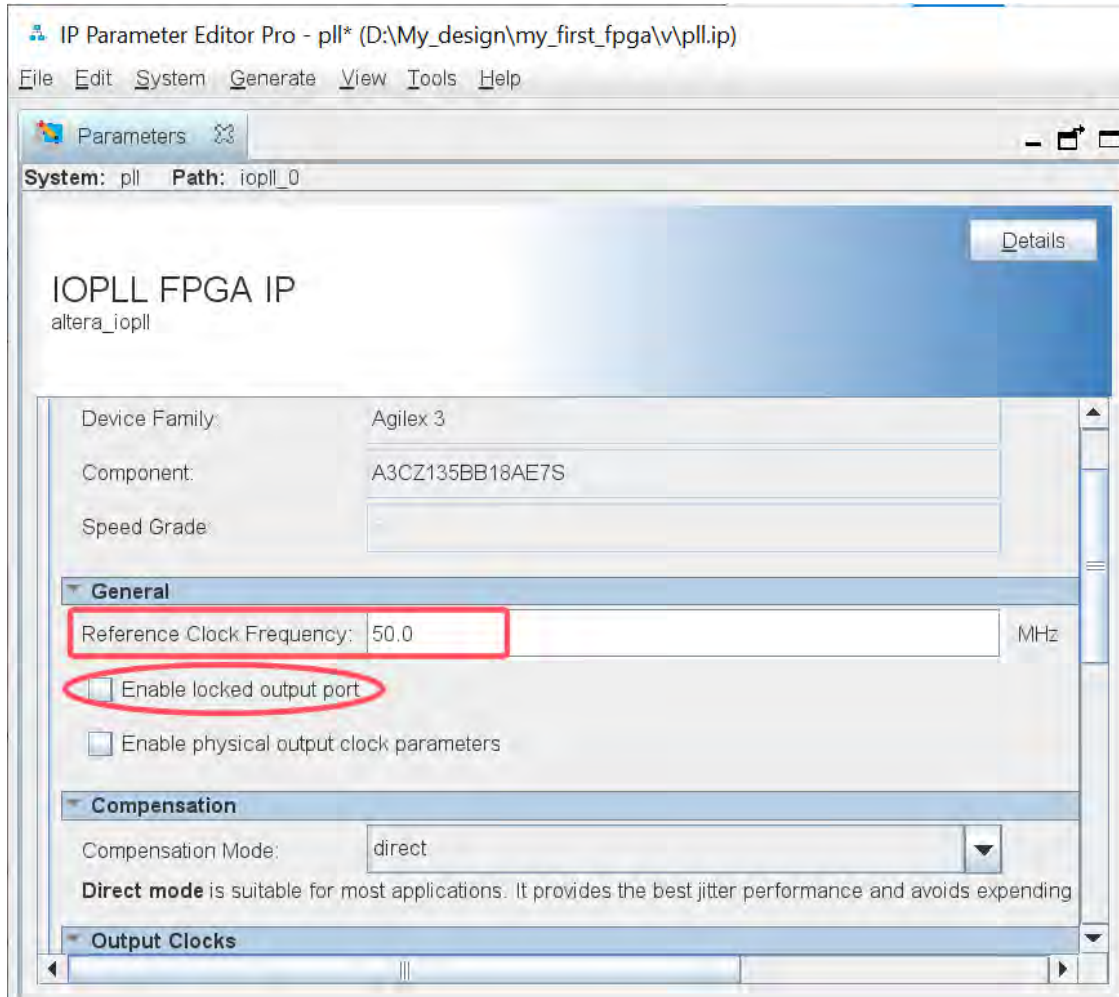


Figure 3-12 Modify IP parameters

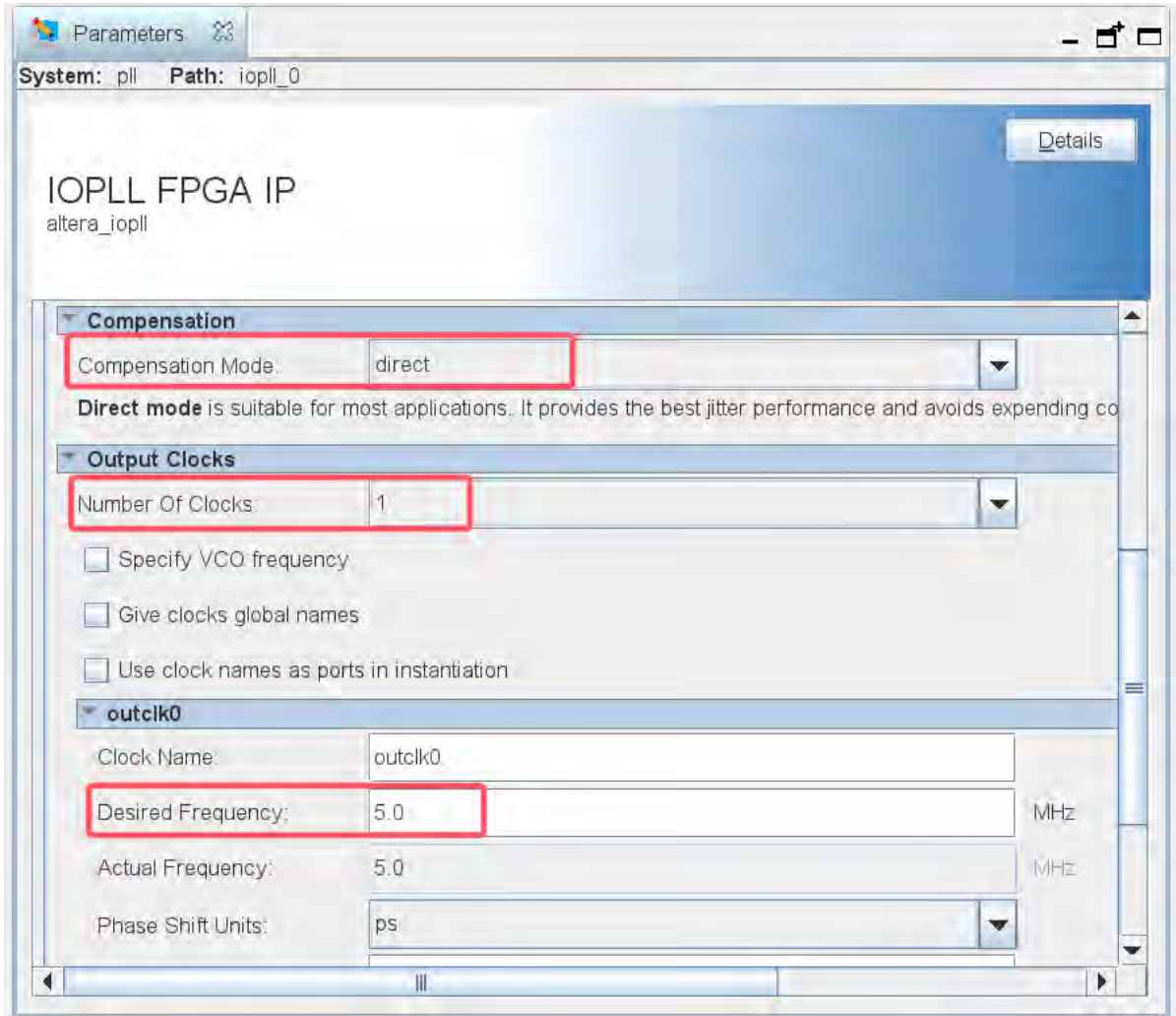


Figure 3-13 Modify IP parameters

7. Click **File-->Save** to save the PLL IP settings.

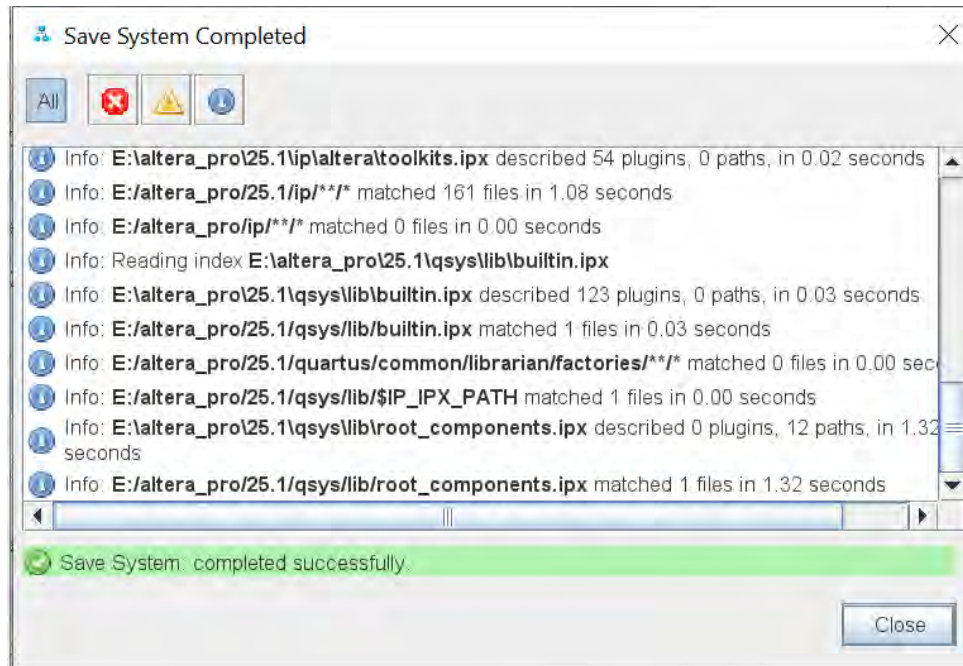


Figure 3-14 Save the IP settings

8. Click the **Generate-->Generate HDL**, in the Generation window, make sure the Create HDL design files for synthesis is selected to Verilog. Click the **Generate** button.

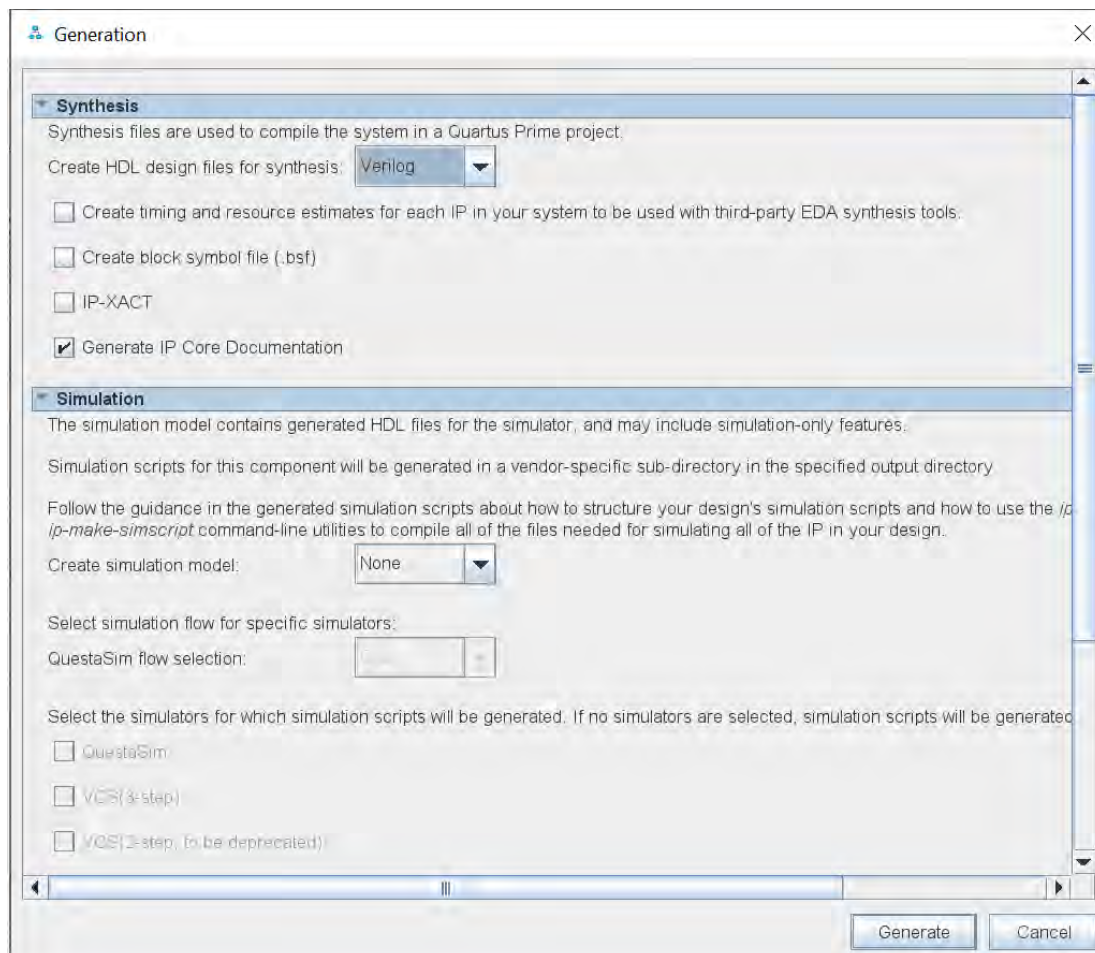


Figure 3-15 Generate HDL design

9. After generating successfully, the PLL IP will be added to the design.

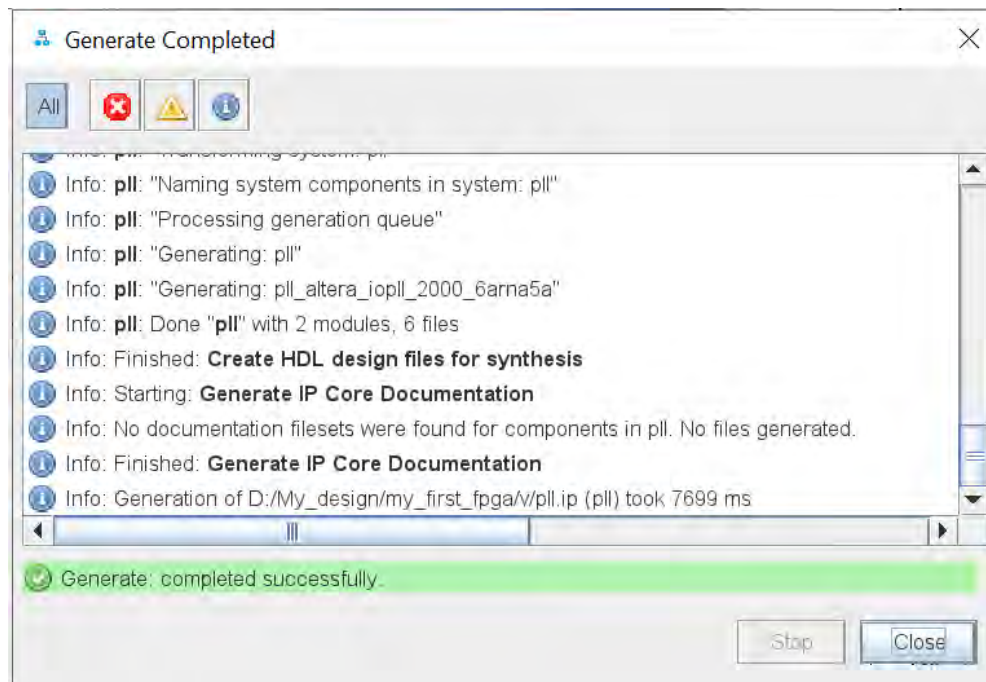


Figure 3-16 Generate completed successfully

3.3 Create a Multiplexer

This design uses a 4-bit 2-to-1 multiplexer to route the simple_counter output to the LEDs on the Atum A3 Nano board. You will create a Verilog HDL code to implements the multiplexer.

Please refer to the steps in section 3.1 Create a Simple Counter to create a **counter_bus_mux.v** file. Type the following Verilog HDL code into the empty file and save it.

```
module counter_bus_mux(
    input [3:0] data0,
    input [3:0] data1,
    input sel,
    output [3:0] result
);

assign result = (sel==1'b1)?data1:data0;

endmodule
```

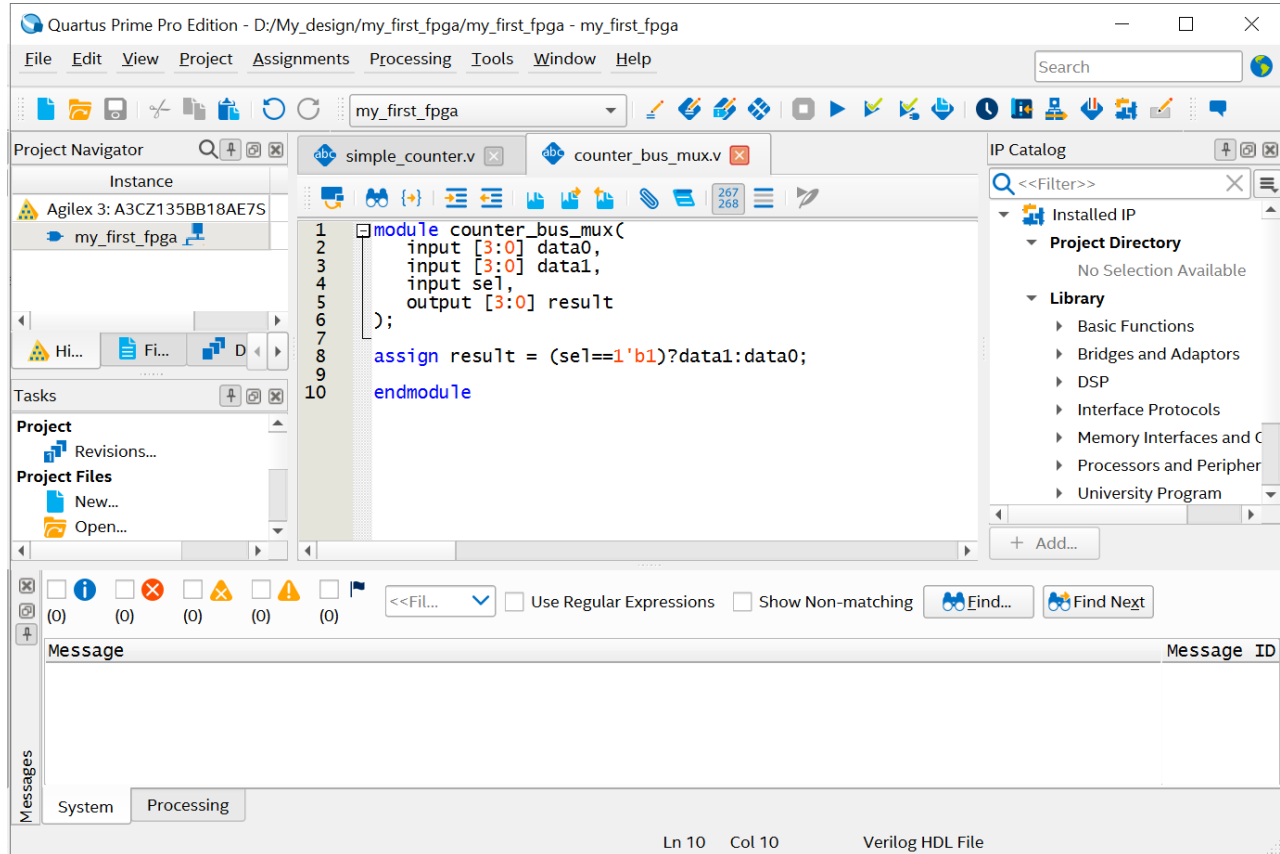


Figure 3-17 Create multiplexer Verilog HDL file

3.4 Instance the Counter, PLL and Multiplexer

In this section, we will introduce to instance the simple_counter, PLL and multiplexer in the top-level file of this design.

The on-board 50MHz clock is used as the PLL clock source, PLL creates a constant clock frequency as the input to the simple_counter, the on-board KEY1 is used as the reset of PLL. Select the counter output bits [26..23] to one of the input four bits of the multiplexer, select the counter output bits [24..21] to another input four bits of the multiplexer, the on-board KEY0 is used as the sel input of the multiplexer. The output of the multiplexer will be connected to the on-board four LEDs.

Create an empty Verilog HDL file, type the following Verilog HDL code into the empty file and save it as my_first_fpga.v file, it's the top-level file of this design.

```
module my_first_fpga(  
    input CLOCK0_50,  
    input [1:0] KEY,  
    output [3:0] LED  
);  
  
wire clk_50;  
wire [31:0] counter;  
wire [3:0] mux_result;  
  
pll u_pll(  
    .refclk(CLOCK0_50),  
    .rst(~KEY[1]),  
    .outclk_0(clk_50)  
);  
  
simple_counter u_simple_counter(  
    .clk(clk_50),  
    .counter_out(counter)  
);  
  
counter_bus_mux u_counter_bus_mux(  
    .data0(counter[24:21]),  
    .data1(counter[26:23]),  
    .sel(KEY[0]),  
    .result(mux_result)  
);  
  
assign LED=~mux_result;  
endmodule
```

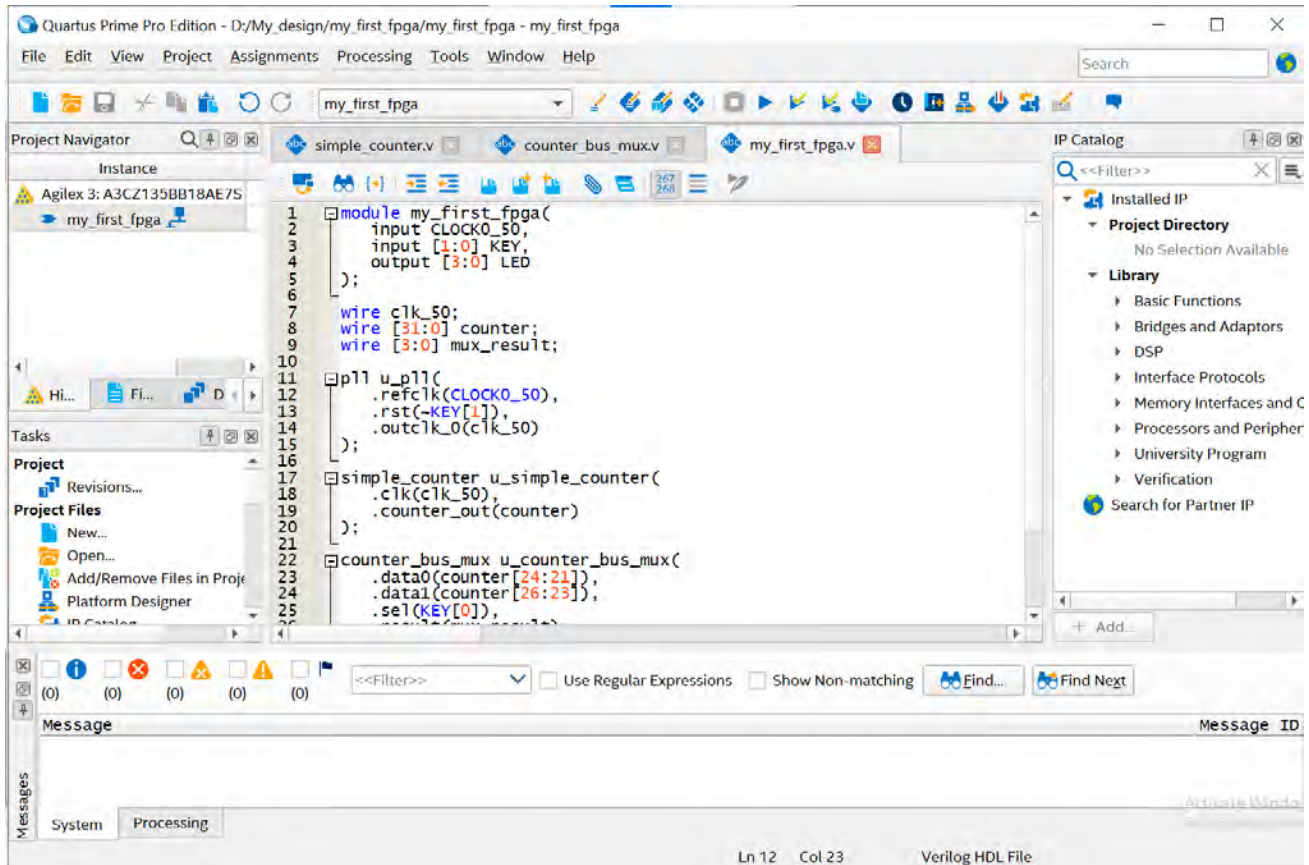


Figure 3-18 Create top-level file

3.5 Assign the Pins

In this section, you will make pin assignments. To make pin assignments that correlate to the KEY[1:0], Clock_50 input pins and LED[3:0] output pins, perform the following steps:

1. Click **Processing --> Start --> Start Analysis & Synthesis**, the message will appear after it's completed.

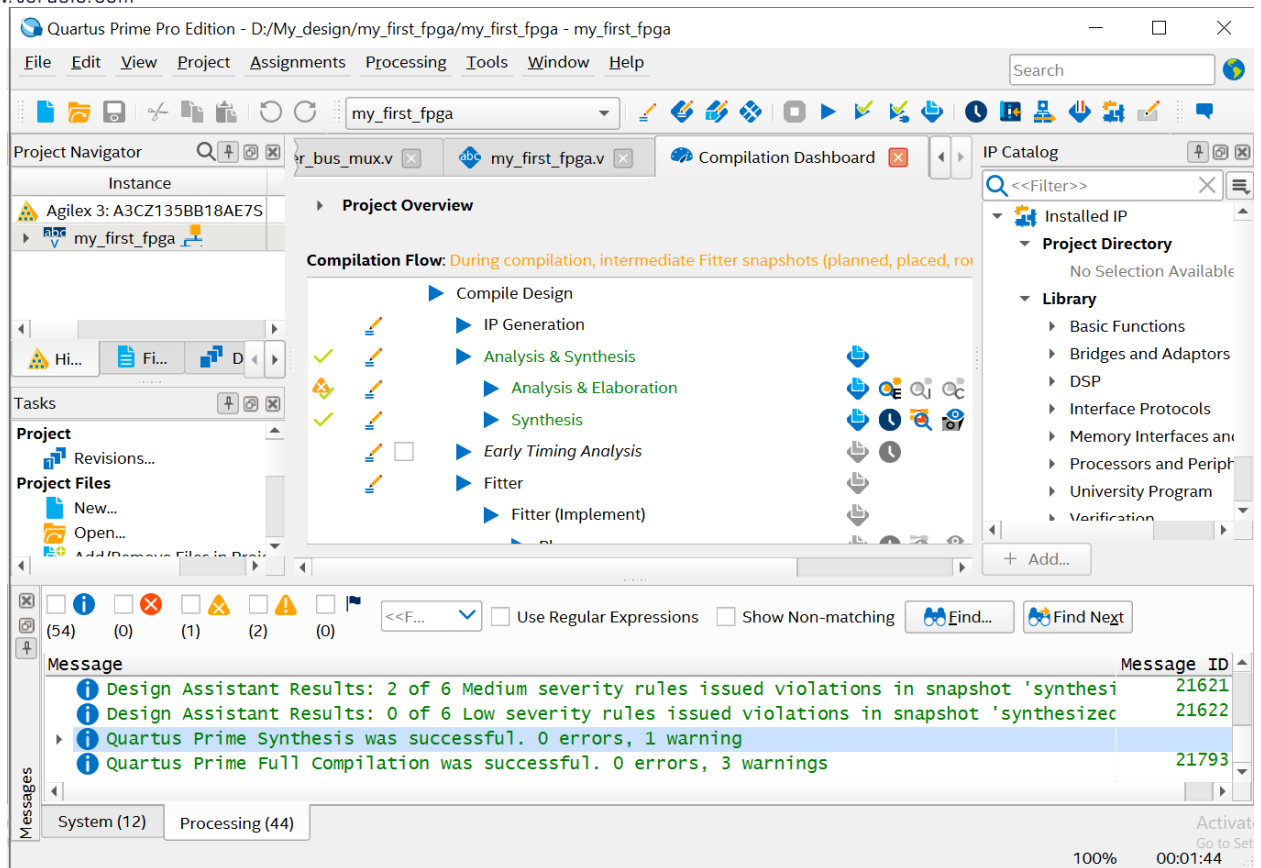


Figure 3-19 Start Analysis & Synthesis

2. Click **Assignments-->Pin Planner** to open the Pin Planner window, it shows the design's seven pins.

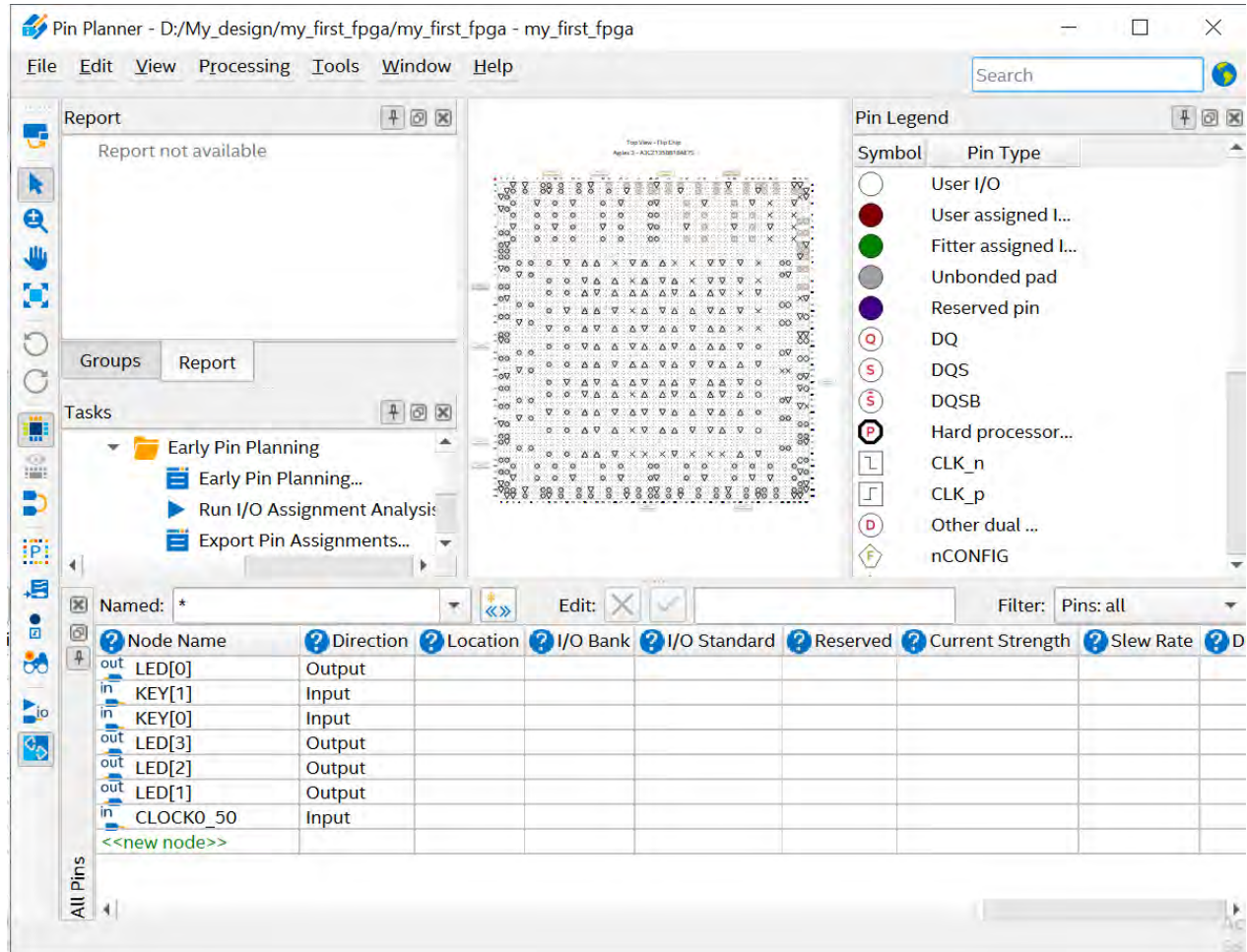


Figure 3-20 Pin Planner

3. In the Location column, double click the blank space next to each of the Node name, open a drop-down list, then you can choose the pins from the list for each node. The software also keeps track of corresponding FPGA data such as the I/O bank and VREF Group. Each bank has a distinct color, which corresponds to the top-view wire bond drawing in the upper right window. The seven pins assignment of this design is listed in [Table 3-1](#).

Table 3-1 Pins assignment

| Node Name | FPGA pin Location |
|-----------|-------------------|
| CLOCK0_50 | K43 |
| KEY[0] | E2 |
| KEY[1] | K3 |
| LED[0] | AG2 |
| LED[1] | AM6 |
| LED[2] | AF1 |
| LED[3] | AF2 |

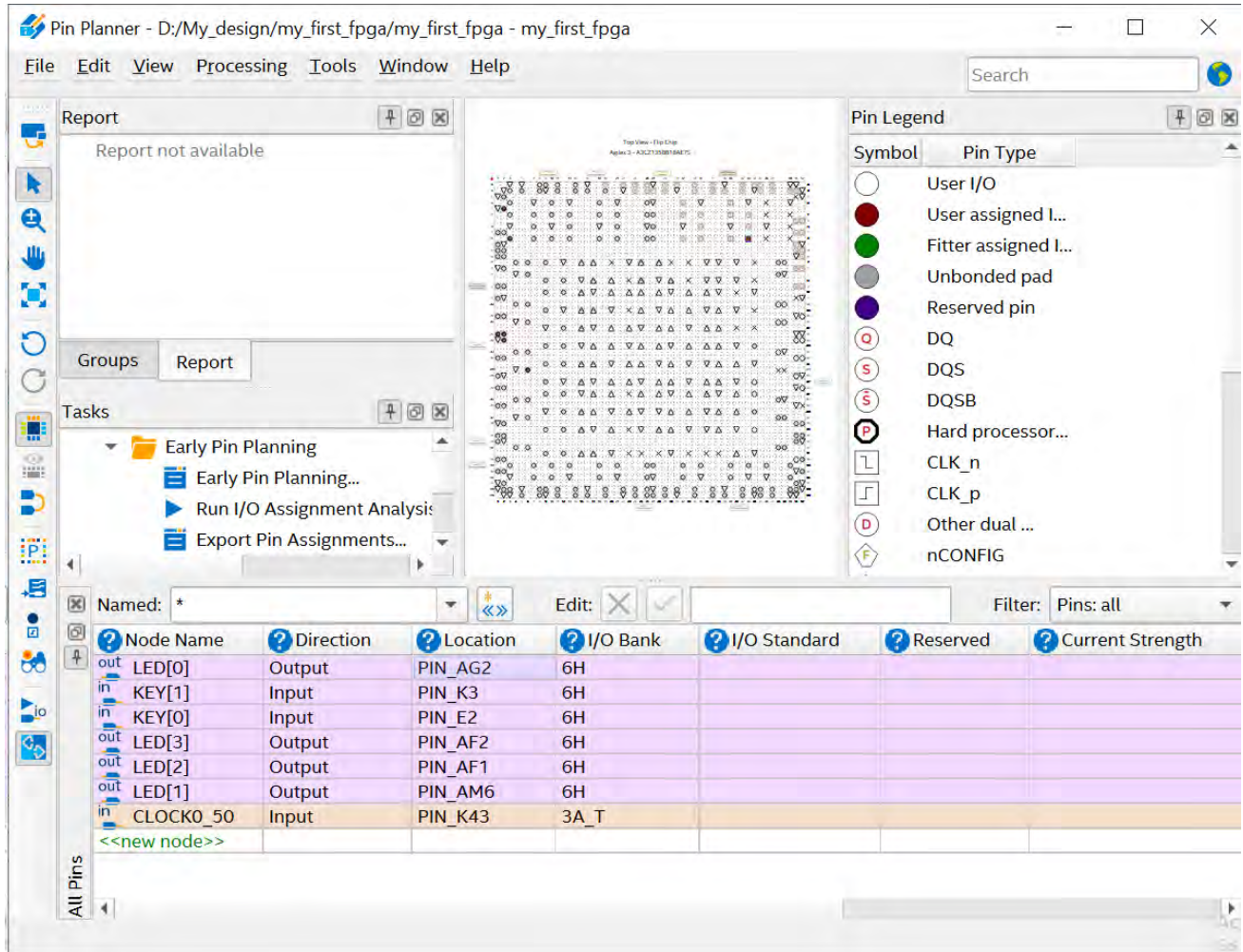


Figure 3-21 Complete the pins assignment

4. Now, you are finished creating your Quartus project design!

Chapter 4

Compile and Verify Design

After creating your design, you must compile it. Compilation converts the design into a bitstream that can be downloaded into the FPGA. The most important output of compilation is an SRAM Object File (.sof), which is used to program the device. The software also generates other report files that provide information about your code as it compiles.

4.1 Compile Your Design

If you want to store .sof in memory device (such as flash or EEPROMs), you must first convert the .sof to a file type specifically for the targeted memory device.

Now that you have created a complete Quartus project and assigned all assignments, you can compile the design. In the Processing menu, choose Start Compilation or click its icon button on the toolbar.

While compiling your design, the Quartus software provides useful information about the compilation on the Compilation Dashboard.

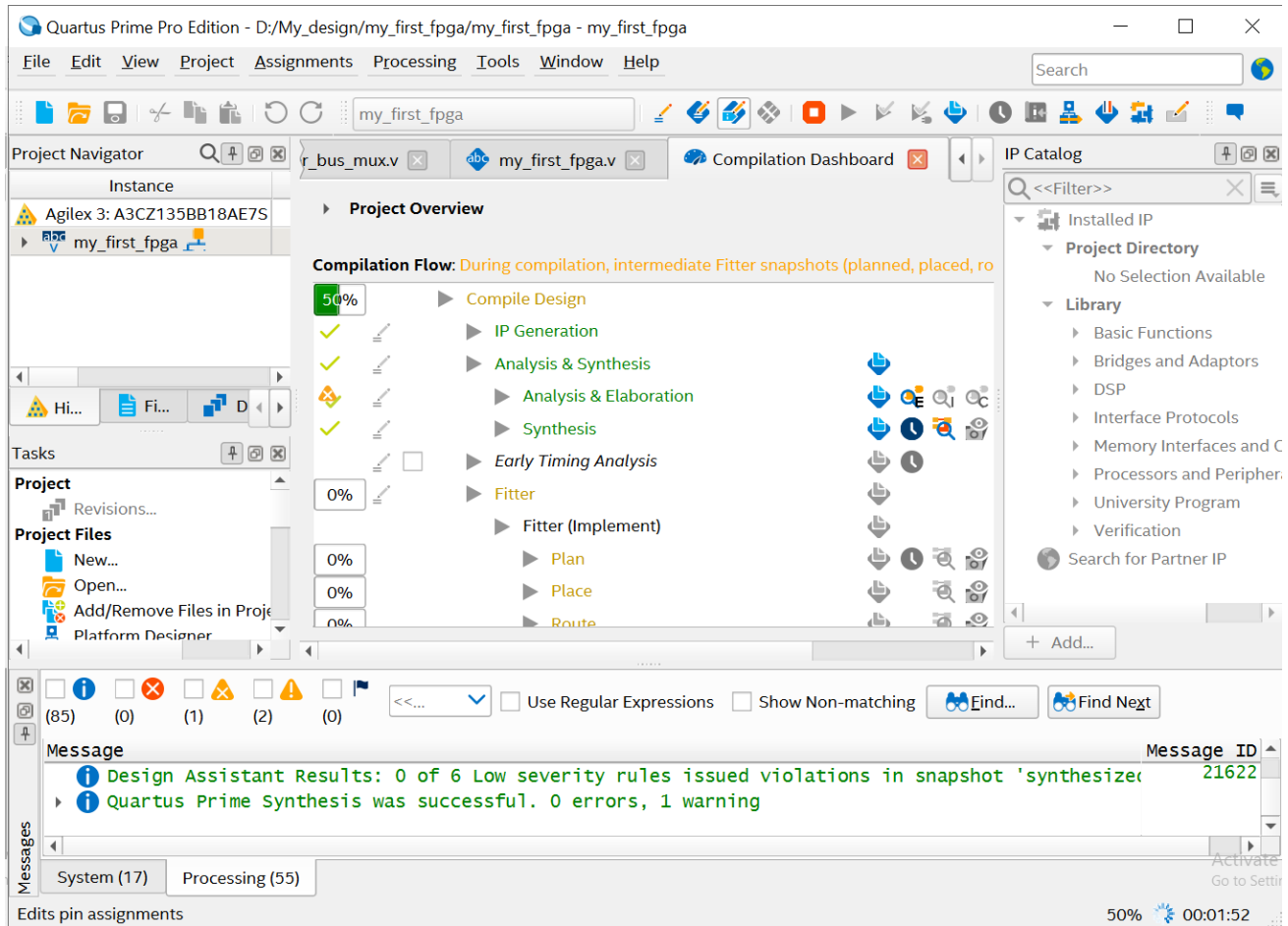


Figure 4-1 Compilation process

The Quartus Messages window displays many messages during compilation. It should not display any critical warnings; it may display a few warnings that indicate that the device timing information is preliminary or that some parameters on the I/O pins used for the LEDs were not set. The software provides the compilation results in the Compilation Report tab as shown in **Figure 4-2**.

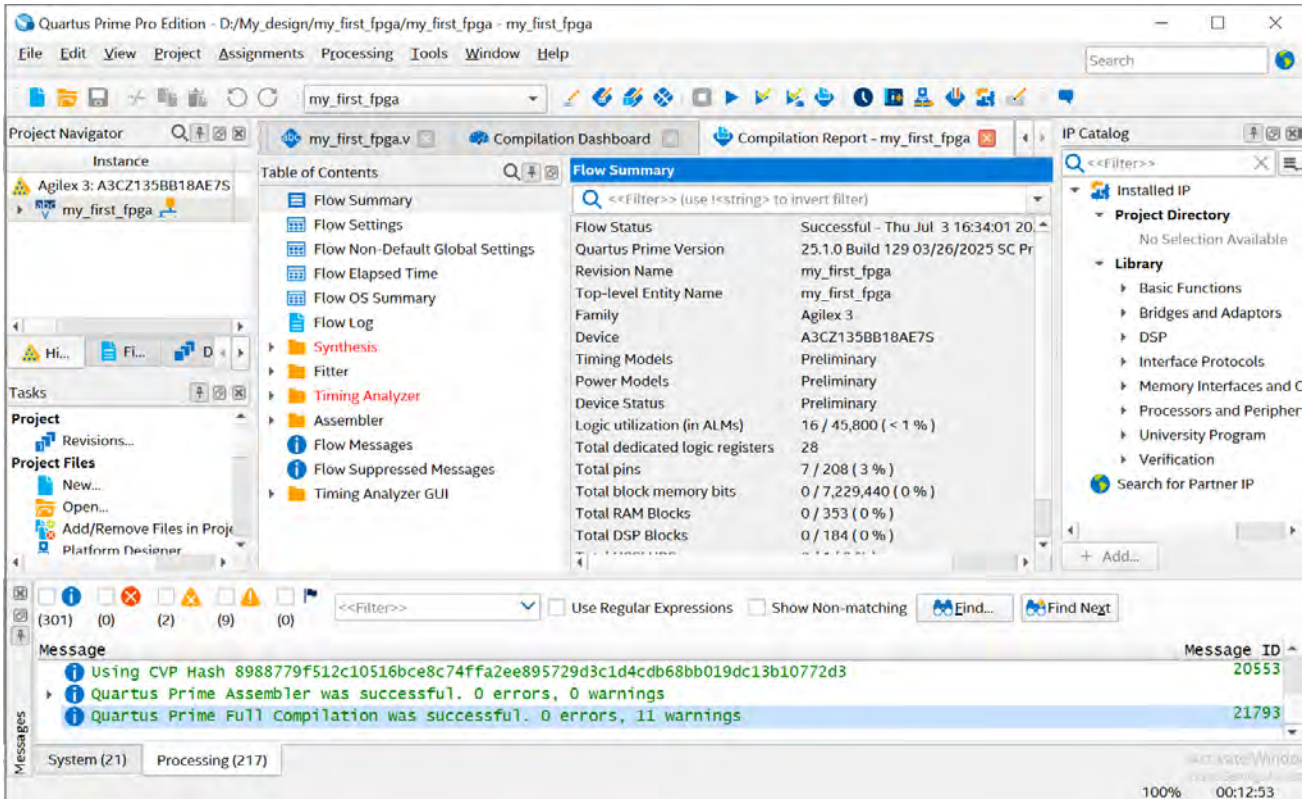


Figure 4-2 Compilation report

After the compilation is completed, the .sof file of the design is generated in the output_files folder.

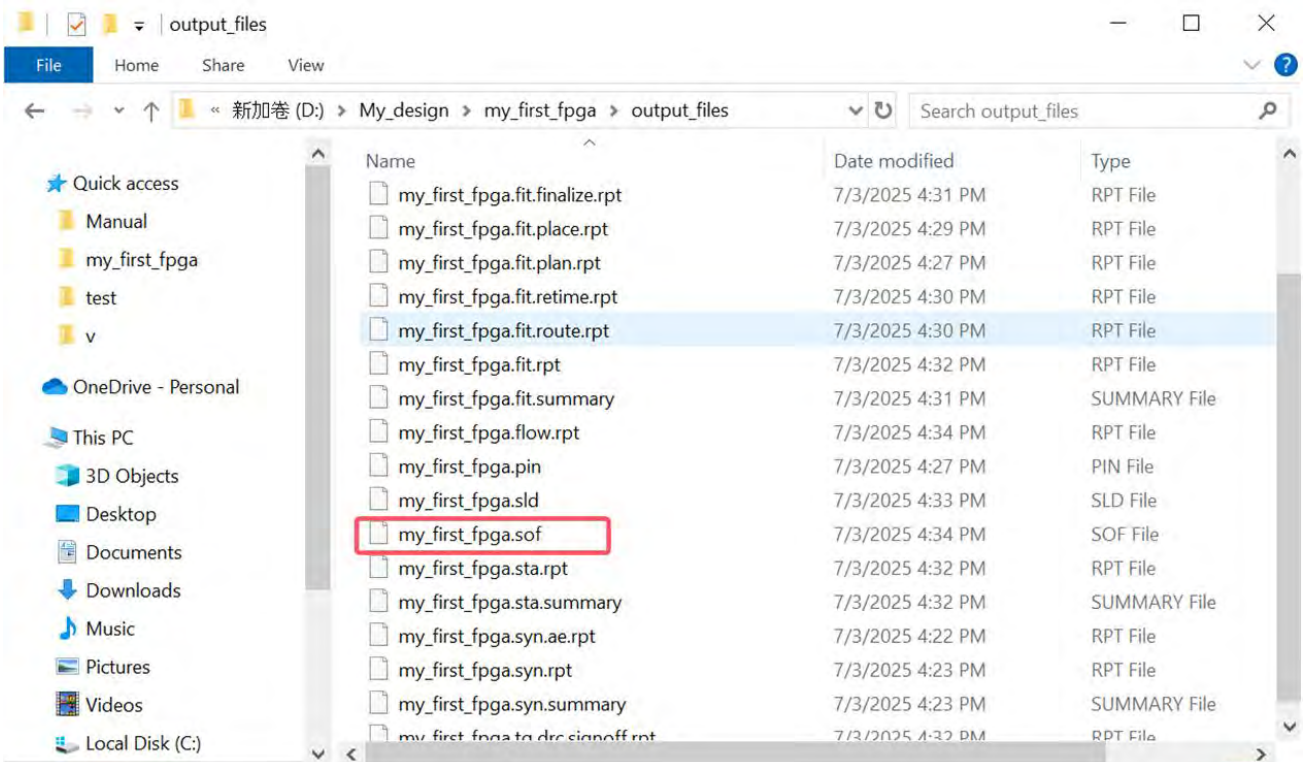


Figure 4-3 Generate the .sof file

4.2 Program the FPGA Device

Follow the steps below to program the FPGA device with the generated my_first_fpga.sof.

1. Connect your computer to the Atum A3 Nano board by plugging the Type-C USB cable into the USB Blaster III connector (J4) of Atum A3 Nano and power up the board.
2. Open the Quartus Prime software and select **Tools --> Programmer**. The Programmer window will appear as shown in **Figure 4-4**.

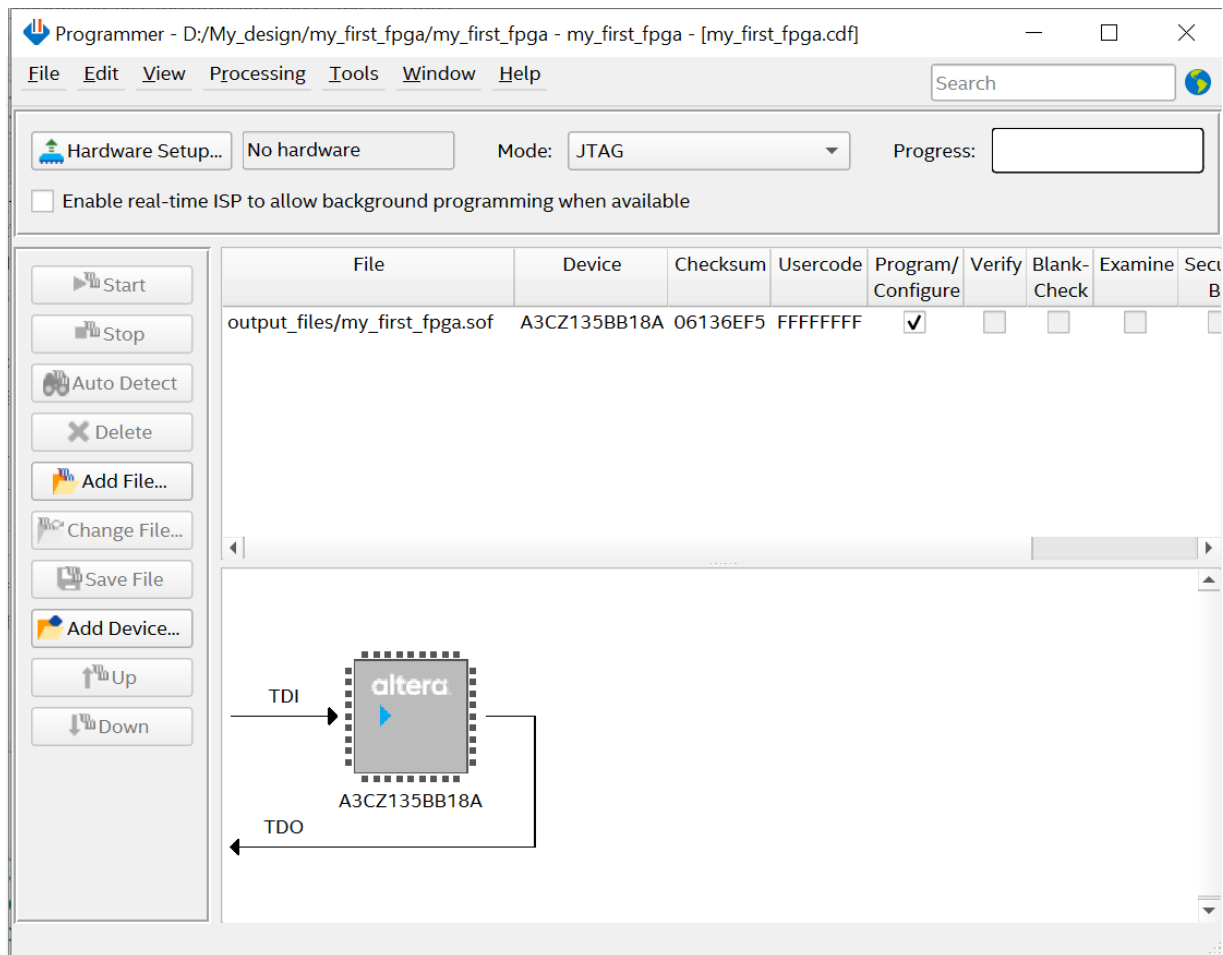


Figure 4-4 Quartus Programmer window

3. Click **Hardware Setup**.
4. Select **Atum A3 Nano[USB-1]** under **Currently selected hardware**, and click **Close** as shown in **Figure 4-5**.

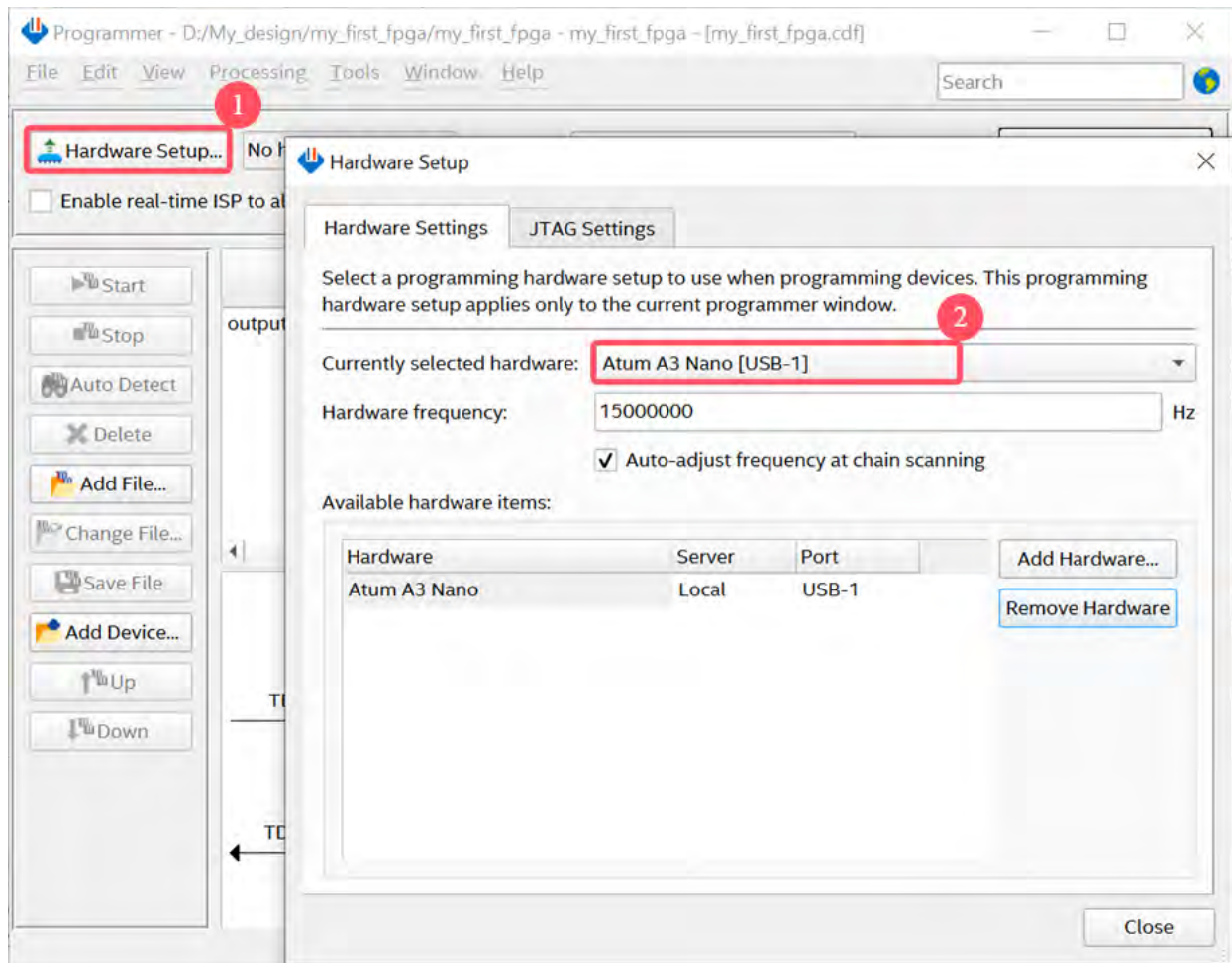


Figure 4-5 Hardware Setup

- As the FPGA device and the .sof file are default selected, you can click “Start” button directly to download .sof file into FPGA, as shown in [Figure 4-6](#). The .sof file is download successfully as shown in [Figure 4-7](#).

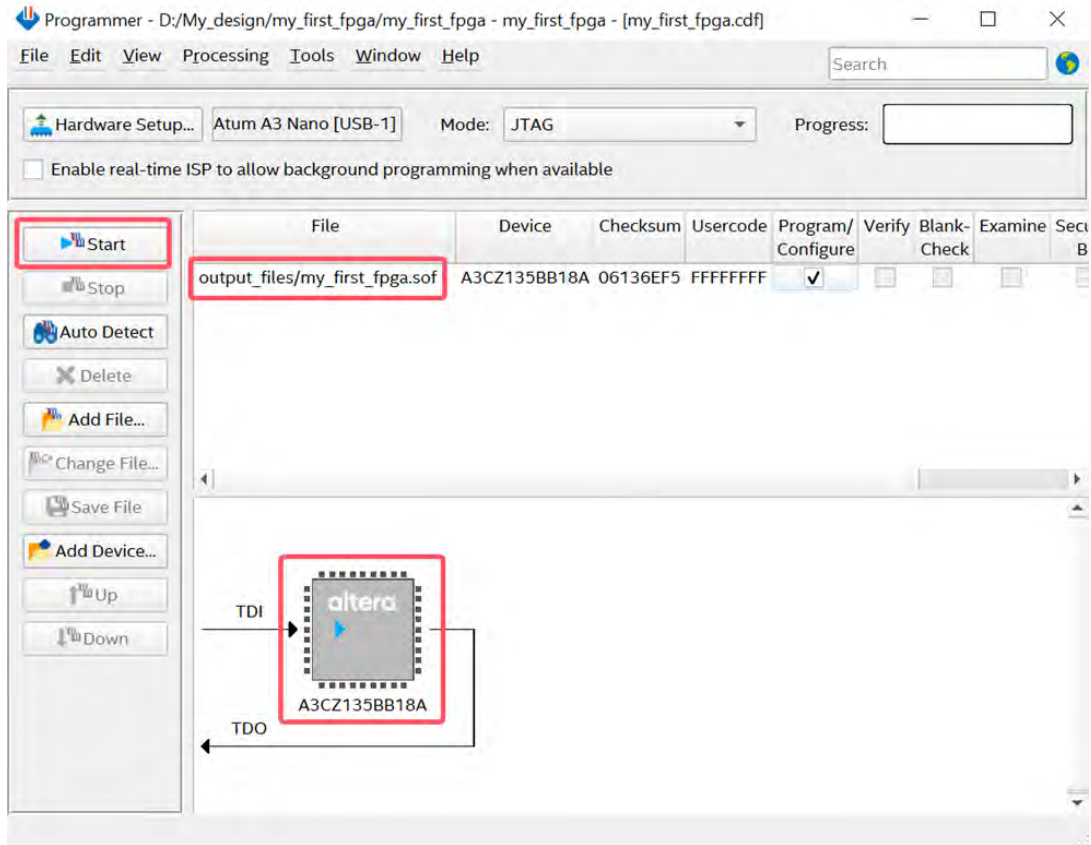


Figure 4-6 Download .sof file

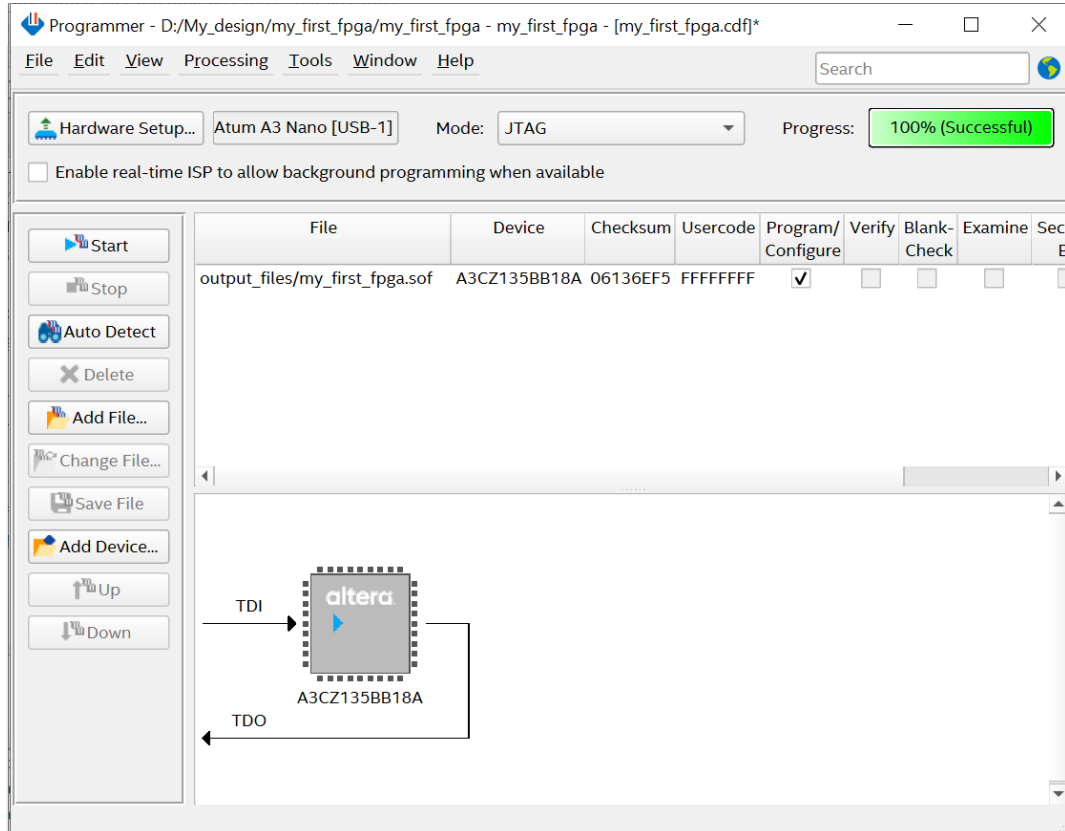


Figure 4-7 Download .sof successfully

4.3 Verify the Hardware

When you verify the design on Atum A3 Nano board, you observe the runtime behavior of the FPGA hardware design and ensure that it is functioning appropriately. Verify the design by performing the following steps:

1. Observe that the four development board LEDs appear to be advancing slowly in a binary count pattern, which is driven by the `simple_counter` bits [26..23].
The LEDs are active high, therefore, when counting begins, all the four LEDs are turned off (the 0000 state).
2. Press and hold KEY[0] on the board, observe that the four LEDs advance more quickly. Pressing this KEY[0] causes the design to multiplex using the faster advancing part of the counter (bits [24..21]).

Additional Information

Contact Terasic

Users can refer to the following table for technical support and more information of Terasic and our product:

| Contact Method | Address |
|----------------|--|
| Email | support@terasic.com/sales@terasic.com |
| Tel | +886-3-575-0880 |
| Website | www.terasic.com |
| Address | 9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan, 30070 |

Revision History

| Date | Version | Changes |
|---------|---------|---------------|
| 2025.07 | V1.0 | First Version |
| | | |
| | | |
| | | |