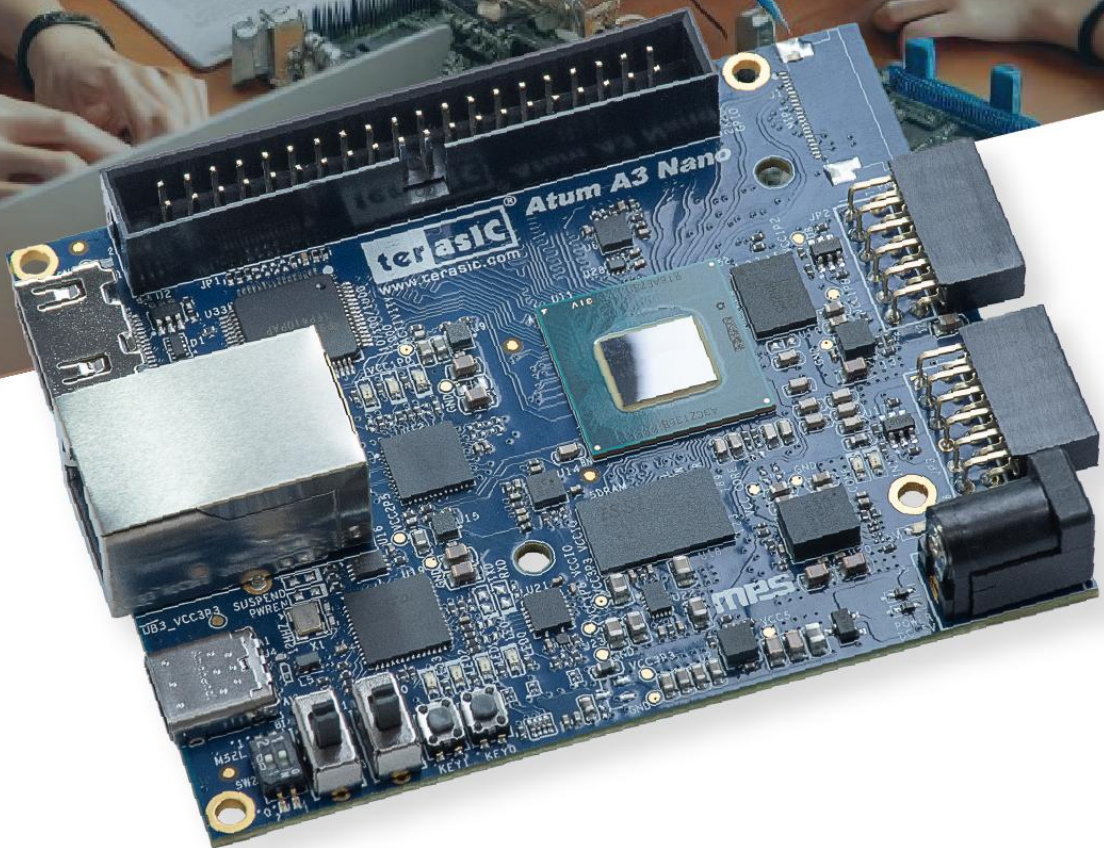


My First Nios V for Terasic Atum A3 Nano



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

CONTENTS

Chapter 1	Overview	1
	1.1 Hardware and Software Requirements	1
Chapter 2	Hardware Design	2
	2.1 Creation of Hardware Design	2
	2.1.1 Creating a New Project	2
	2.1.2 Creating a Platform Designer System	5
	2.1.3 Configuring Assignment and Constraint	27
	2.1.4 Compiling the Quartus Prime Software Project.....	30
	2.2 Download Hardware Design to Target FPGA	32
Chapter 3	Ashling RiscFree IDE Build Flow	35
	3.1 Create the hello_world Example Project	35
	3.2 Build and Run the Program	46
	3.3 Edit and Re-Run the Program	60
	3.4 Why the LED Blinks.....	65
	3.5 Debugging the Application	66
	3.6 Configure System Library	68
Chapter 4	Appendix.....	73
	4.1 Revision History	73
	4.2 Copyright Statement	73

Chapter 1

Overview

This document provides an introductory roadmap for developing a Nios® V processor system. It walks you through creating and running a basic Hello World program and controlling the LED blink feature on an FPGA board. The example setup uses the Terasic Atum A3 Nano development board, with Quartus® Prime software and the Ashling* RiscFree* IDE for Altera FPGAs as the primary development tools.

1.1 Hardware and Software Requirements

Hardware Requirements

- Terasic Atum A3 Nano development board

Software Requirements

- Quartus Prime Pro Edition (This document uses Quartus Prime Pro Edition version 25.1)
- Ashling RiscFree IDE for Altera FPGAs

Chapter 2

Hardware Design

This chapter will guide you through starting the hardware design process by adding the Nios V processor and its peripherals into Platform Designer. After setting up the system assignments and constraints, you will complete the hardware design by successfully compiling the project.

2.1 Creation of Hardware Design

This section describes the flow of how to create a hardware system including Platform Designer feature.

2.1.1 Creating a New Project

1. Launch Quartus Prime then select **File->New Project Wizard**, start to create a new project. See Figure 2-1.

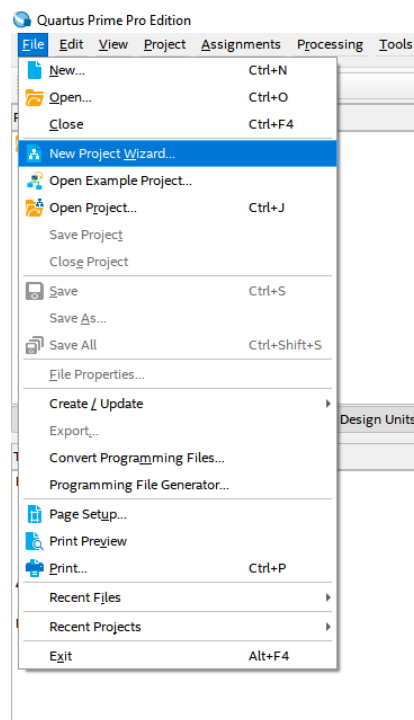


Figure 2-1 Start to Create a New Project

2. Choose a working directory for this project, type project name and top-level entity name as shown in Figure 2-2. Click **Next** to next window.

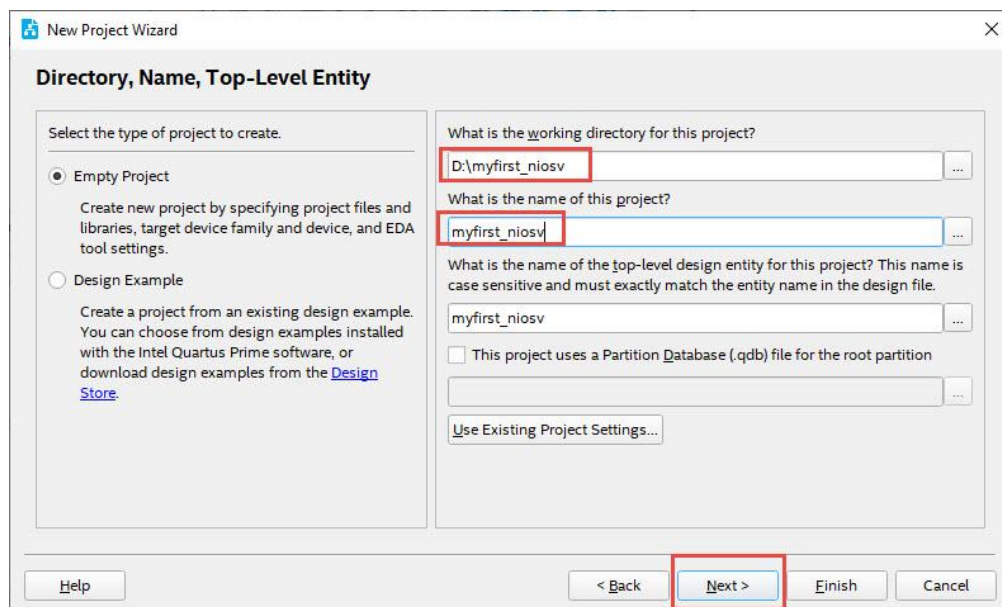
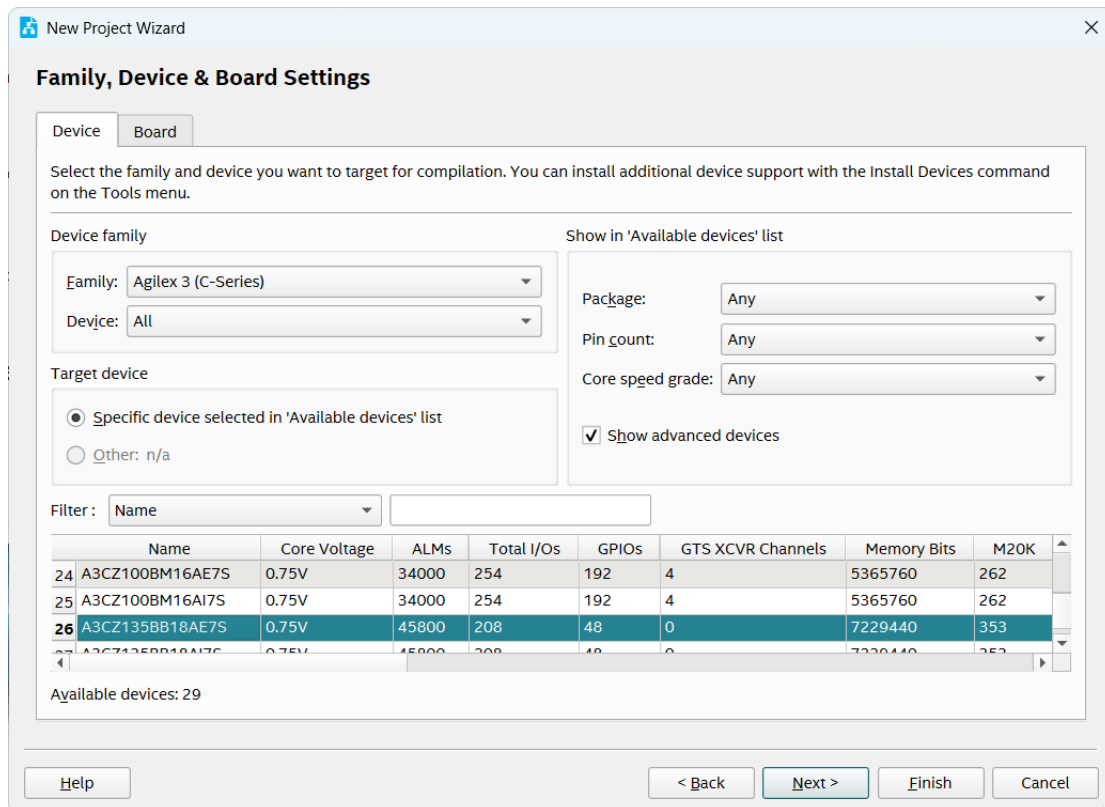


Figure 2-2 Input the working directory, the name of project, top-level design entity

3. We choose device family and device settings. You should choose settings the same as the **Figure 2-3**.

- FPGA Family :Agilex 3(C-Series)
- FPGA Name : A3CZ135BB18AE7S

Then click **Finish** to finish new project. **Figure 2-4** show a new complete project.



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.

Device family

Family: Agilex 3 (C-Series)

Device: All

Target device

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

☒ Show advanced devices

Filter: Name

	Name	Core Voltage	ALMs	Total I/Os	GPIOs	GTS XCVR Channels	Memory Bits	M20K
24	A3CZ100BM16AE7S	0.75V	34000	254	192	4	5365760	262
25	A3CZ100BM16AI7S	0.75V	34000	254	192	4	5365760	262
26	A3CZ135BB18AE7S	0.75V	45800	208	48	0	7229440	353
27	A3CZ135BB18AI7S	0.75V	45800	208	48	0	7229440	353

Available devices: 29

Help < Back Next > Finish Cancel

Figure 2-3 New Project Wizard: Family & Device Settings

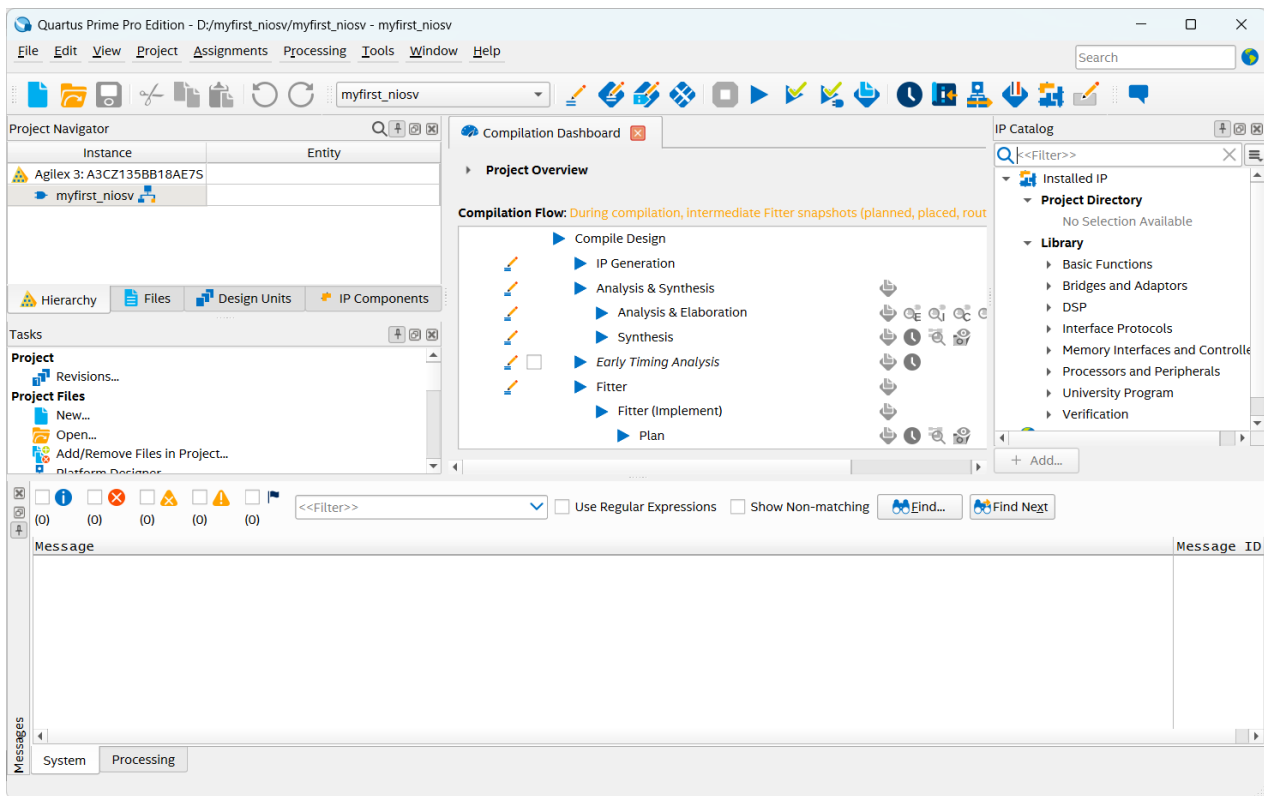


Figure 2-4 A New Complete Project

2.1.2 Creating a Platform Designer System

1. Choose **Tools > Platform Designer** to open new Platform Designer system wizard. See Figure 2-5.

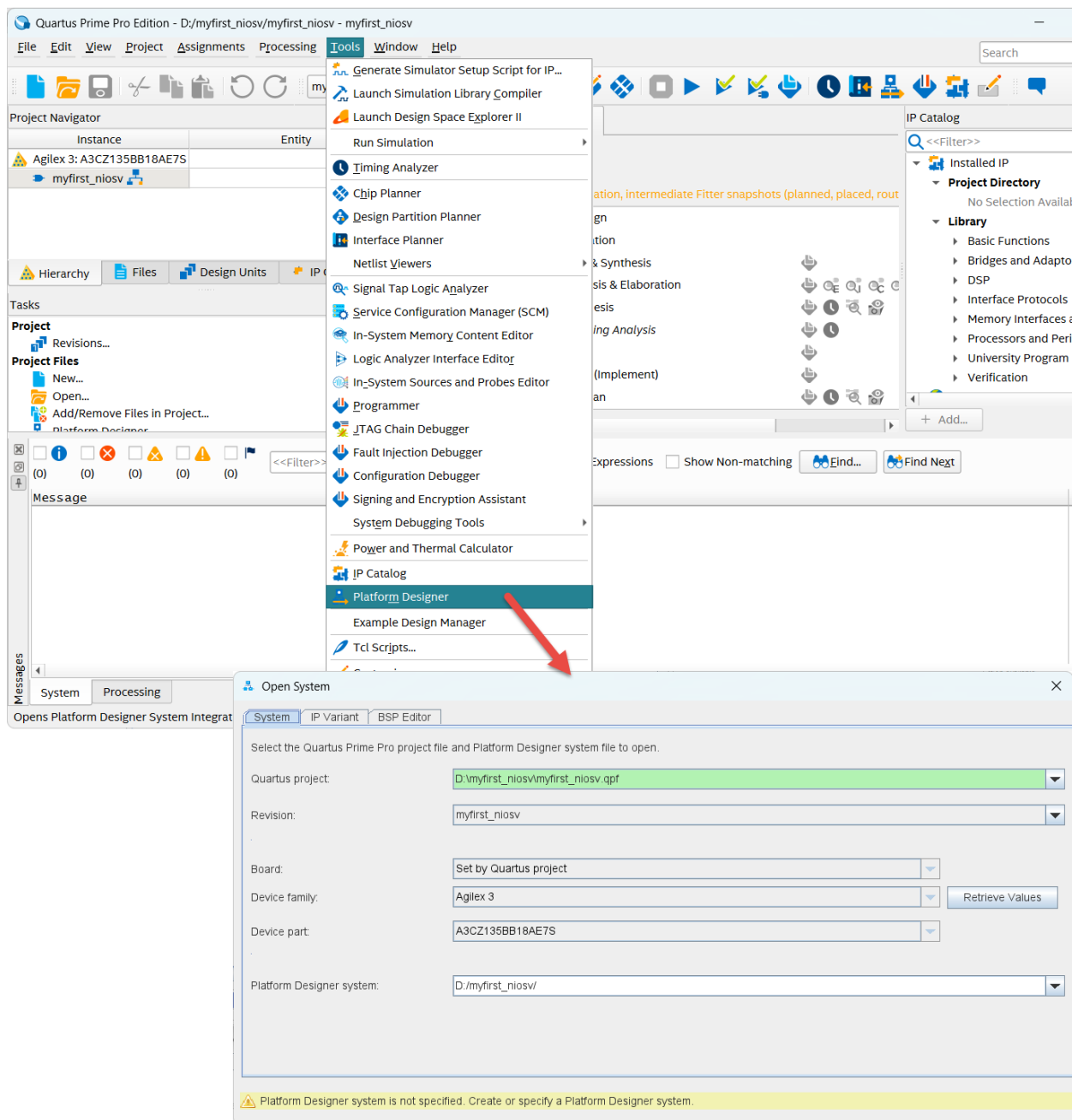


Figure 2-5 Platform Designer Menu

2. Click the Create New Platform Designer System icon and name the new project system "**system**" (refer to Figure 2-6).

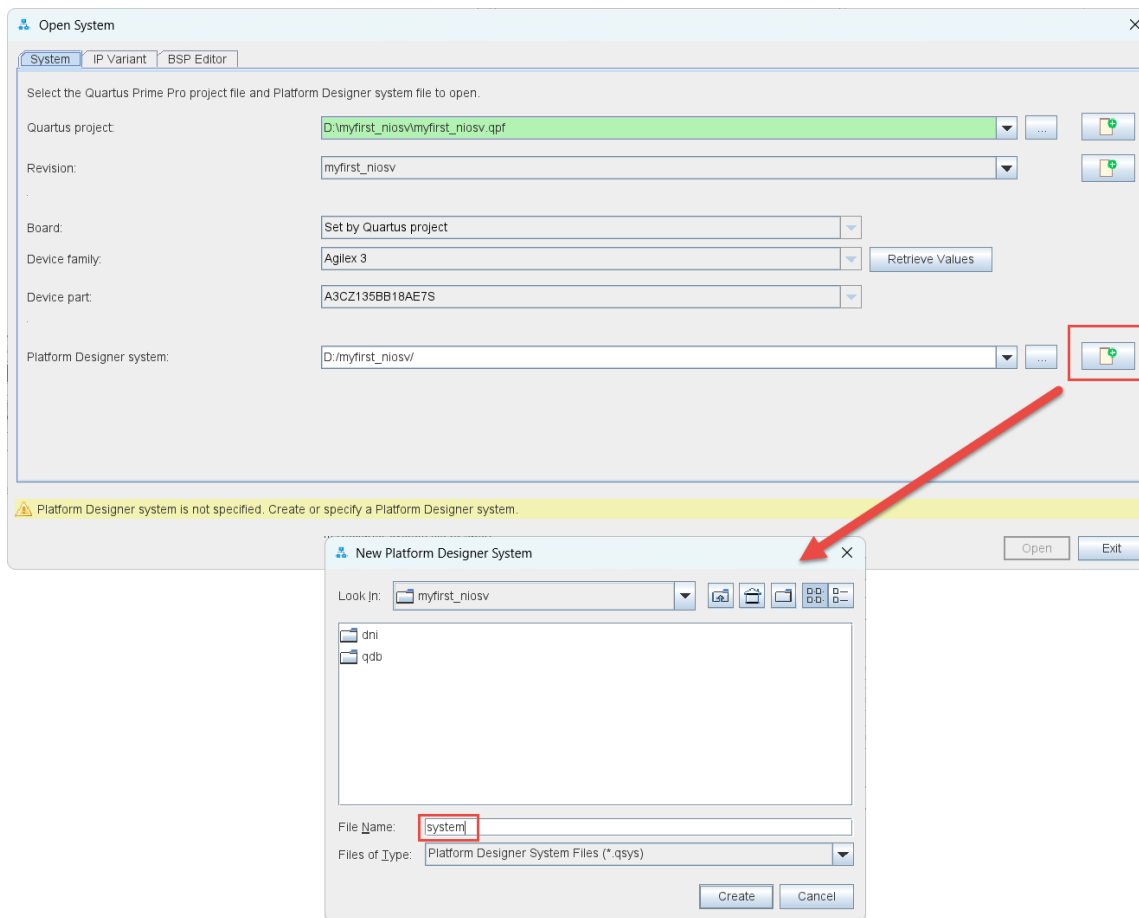


Figure 2-6 Create New System [1]

Click Create and you will see a window as shown in Figure 2-7. Then a Platform Designer window will appear.

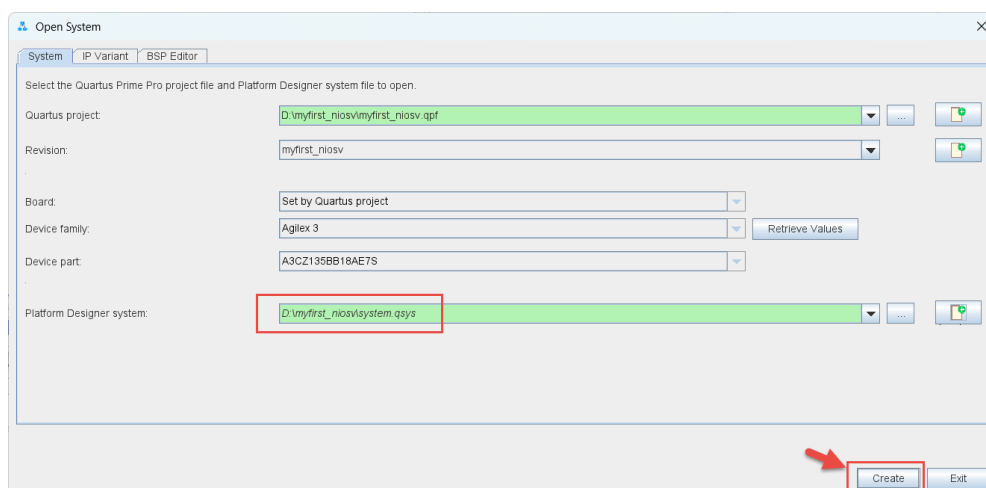


Figure 2-7 Create New System [2]

■ Adding Components

1. Now, we will first add a reset component into the system.

Go to IP tab and choose **Library > Basic Functions > Configuration and Programming > Agilex Reset Release** to open wizard of adding reset component (See Figure 2-8)

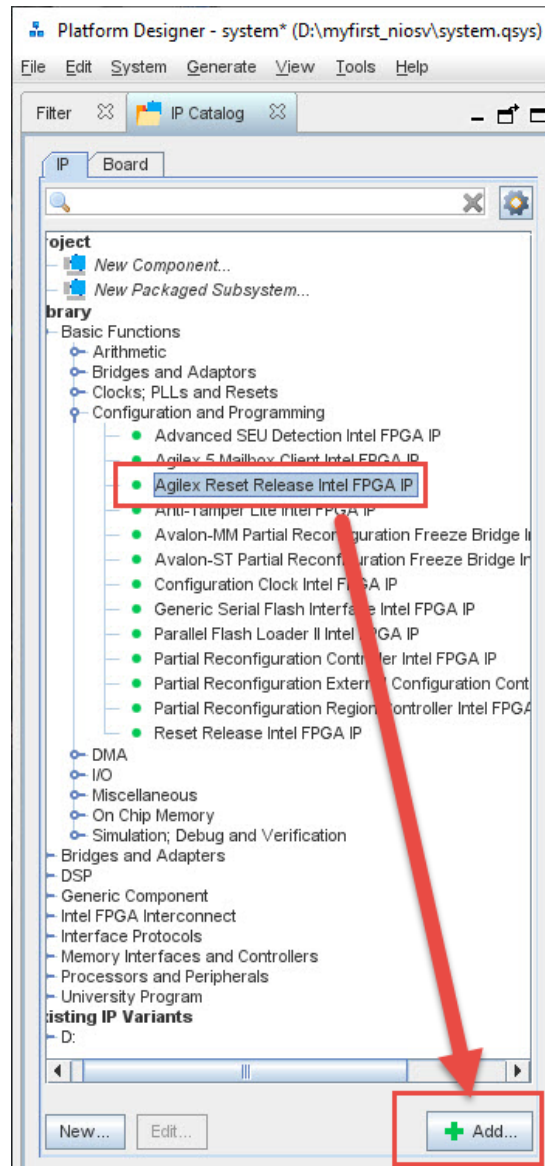


Figure 2-8 Add Agilex Reset Release

See Figure 2-9, In the Agilex Reset Release IP window:

- a. Check **Reset Interface**.

- b. Rename the HDL entity name to “reset_release”
- c. Click **Finish**

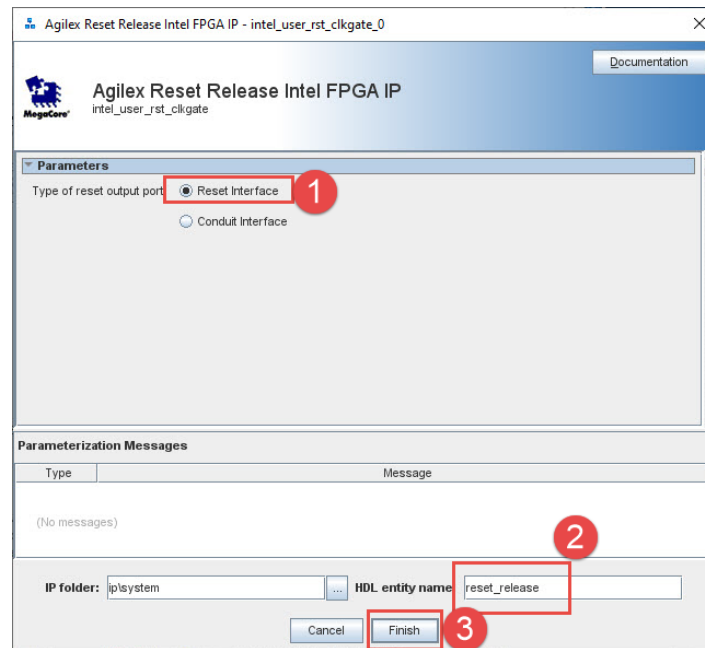


Figure 2-9 Agilex Reset Release

A reset_release component is added in the system as shown in Figure 2-10.

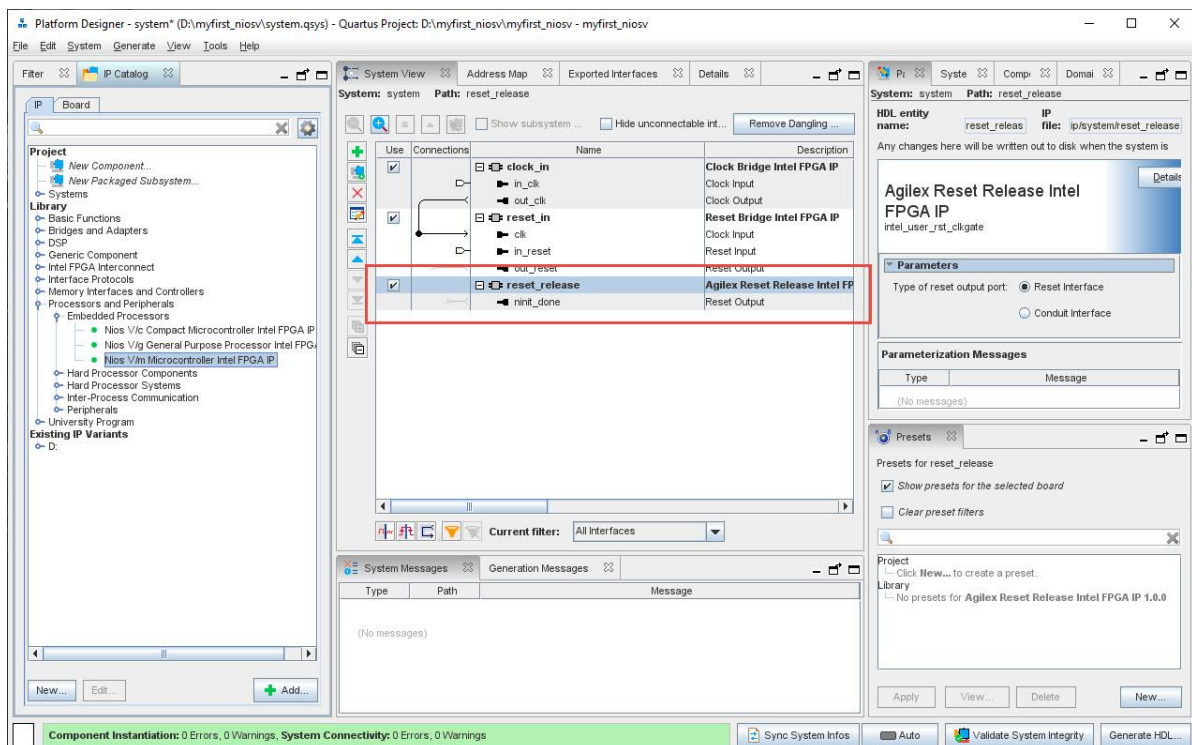


Figure 2-10 Add reset_relese component

2. Then we will add a **Nios V microcontroller** into the system. Choose **Library > Processors and Peripherals > Embedded Processors > Nios V/m Microcontroller** to open wizard of adding cpu component. See Figure 2-11.

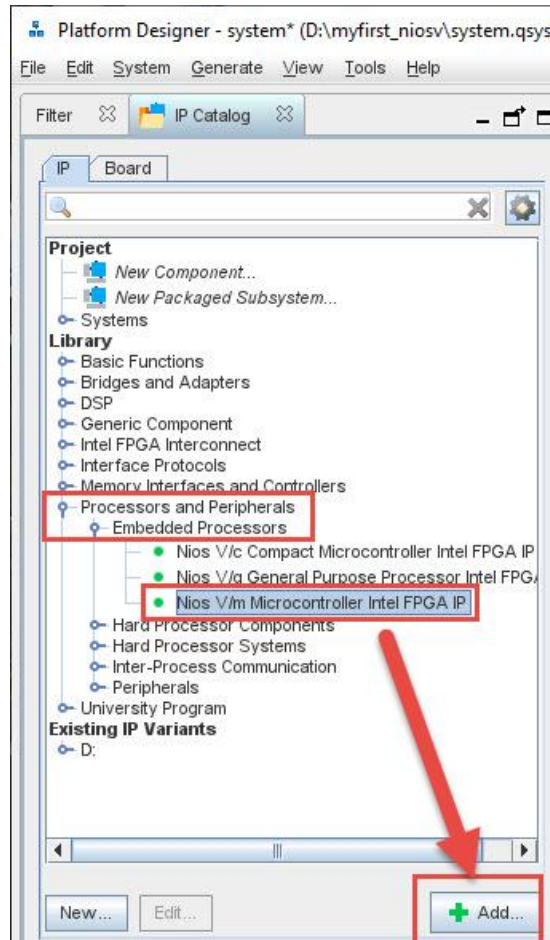


Figure 2-11 Add Nios V/m Processor

See Figure 2-12, n the Nios V/m Microcontroller IP window:

- a. Rename the HDL entity name to “**nios_m**”
- b. Click **Finish**
- c. Figure Figure 2-13 show the **nios_m** component is added in the system.

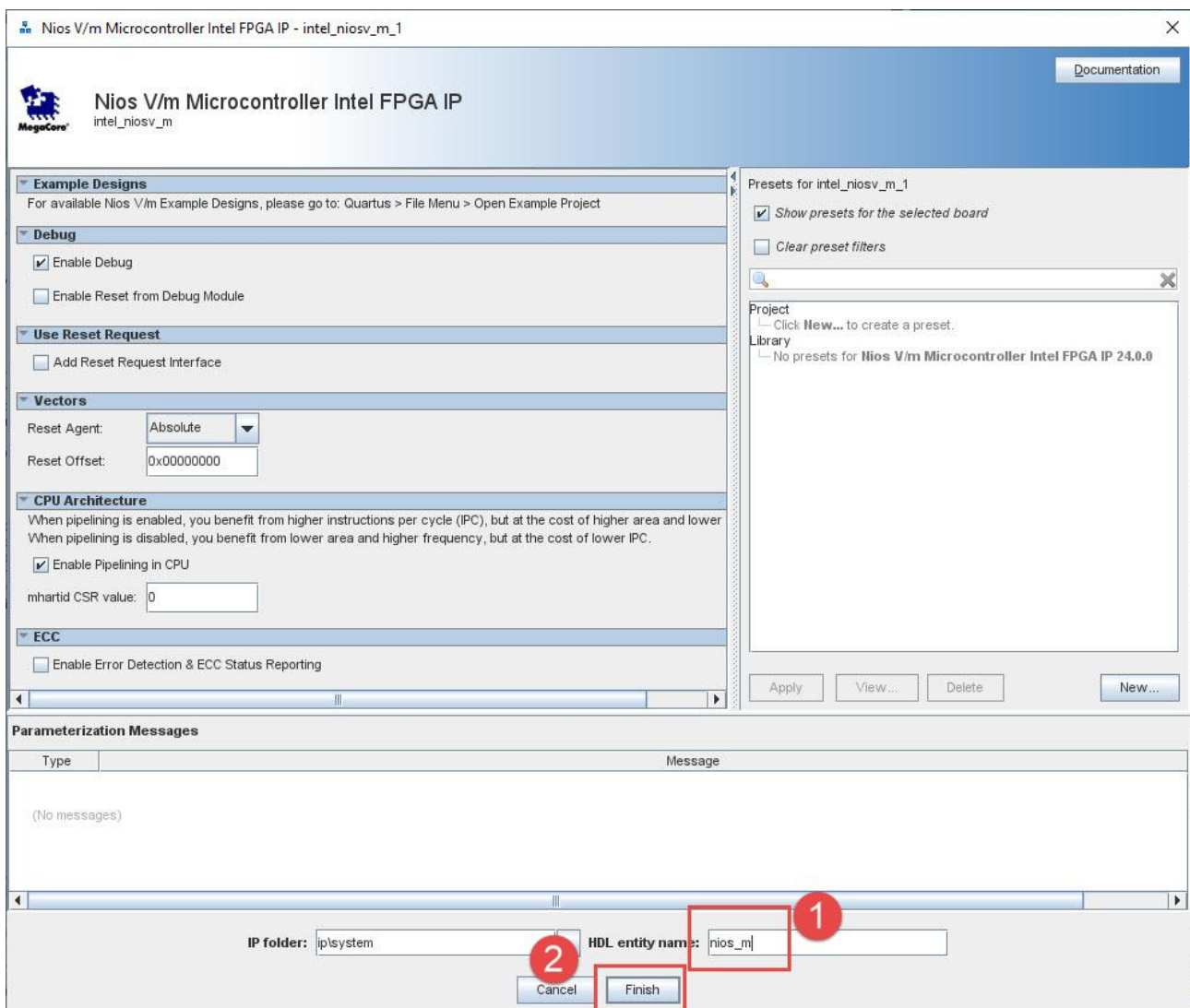


Figure 2-12 Nios V/m Processor

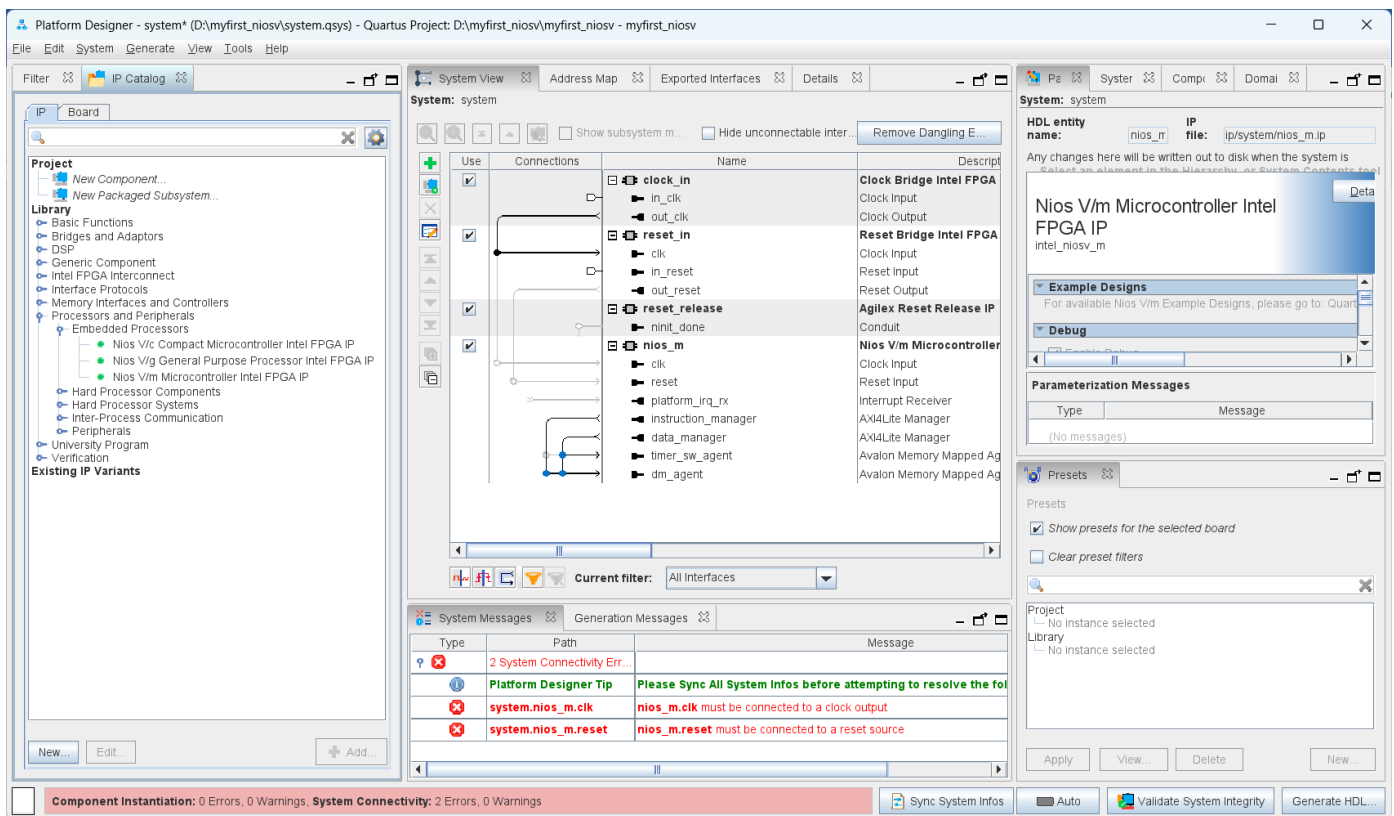


Figure 2-13 Add Nios V CPU completely

3. Choose **Library > Interface Protocols > Serial > JTAG UART Intel FPGA IP** to open wizard of adding JTAG UART as shown in Figure 2-14.

In the JTAG UART IP window:

- a. Rename the HDL entity name to “**jatg_uart**”
- b. Click **Finish**

4. Figure Figure 2-15 show the **jtag_usrt** component is added in the system.

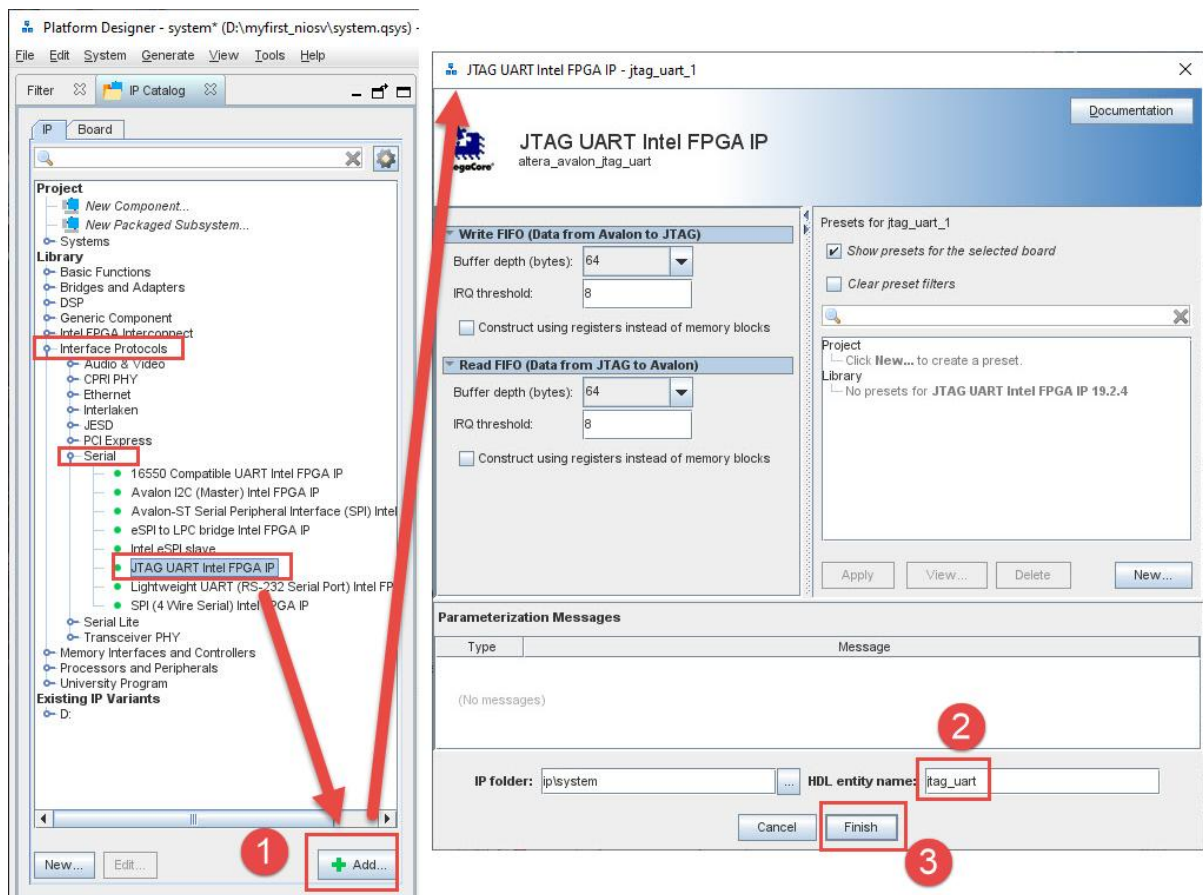


Figure 2-14 Add JTAG UART (1)

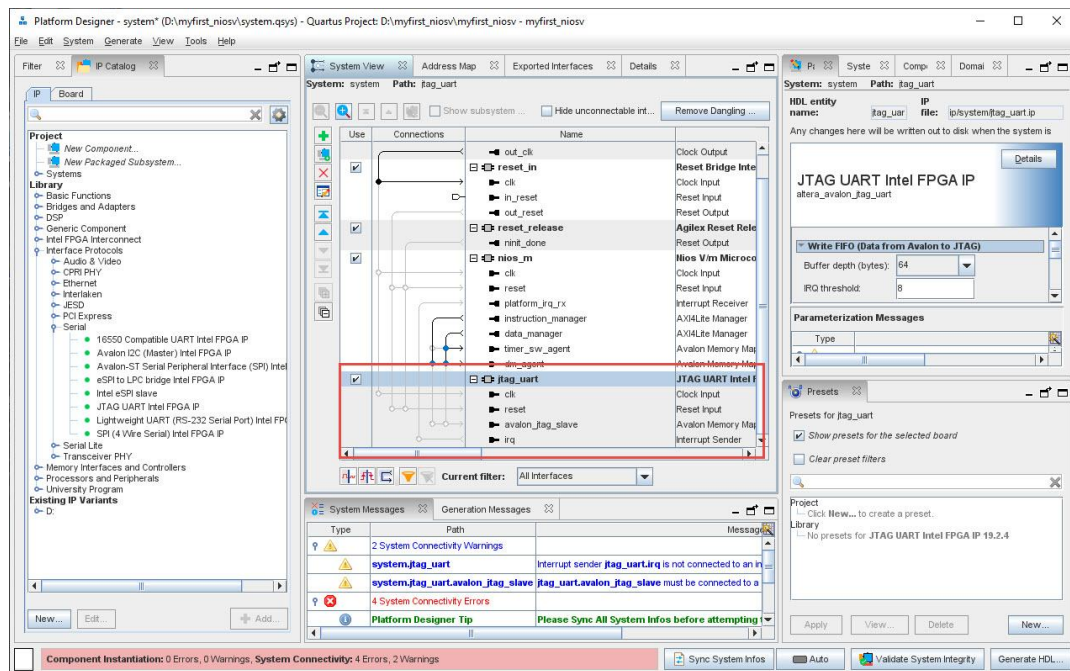


Figure 2-15 JTAG UART

5. Choose **Library > Basic Functions > On-Chip Memory > On-Chip Memory II (RAM or ROM) Intel FPGA IP** to open wizard of adding On-Chip memory. See Figure 2-16.

In the On-Chip Memory II IP window:

- a. Change memory size to “**262144**”.
- b. Rename the HDL entity name to “**onchip_memory**”
- c. Click **Finish**

6. Figure 2-17 show the **onchip_memory** component is added in the system.

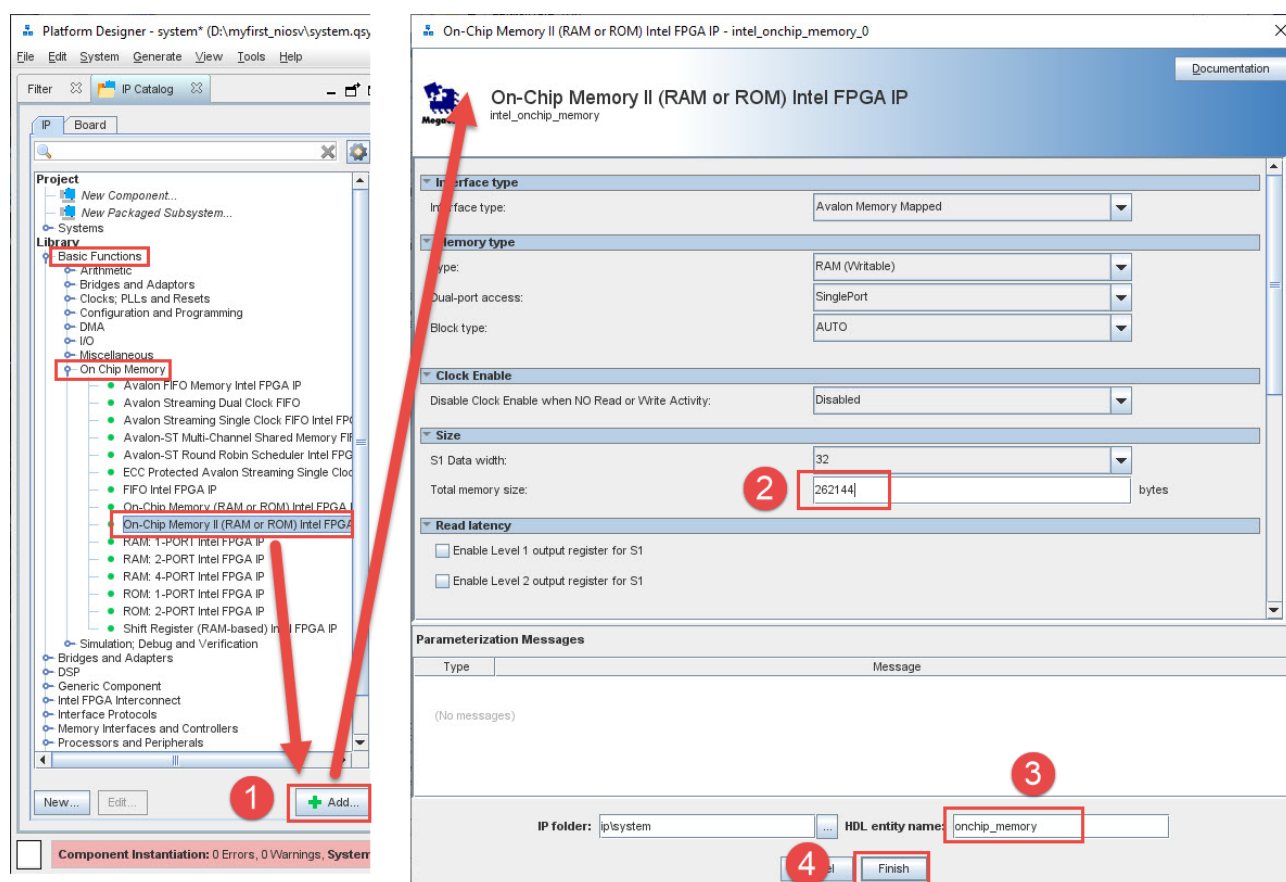


Figure 2-16 Add On-Chip Memory

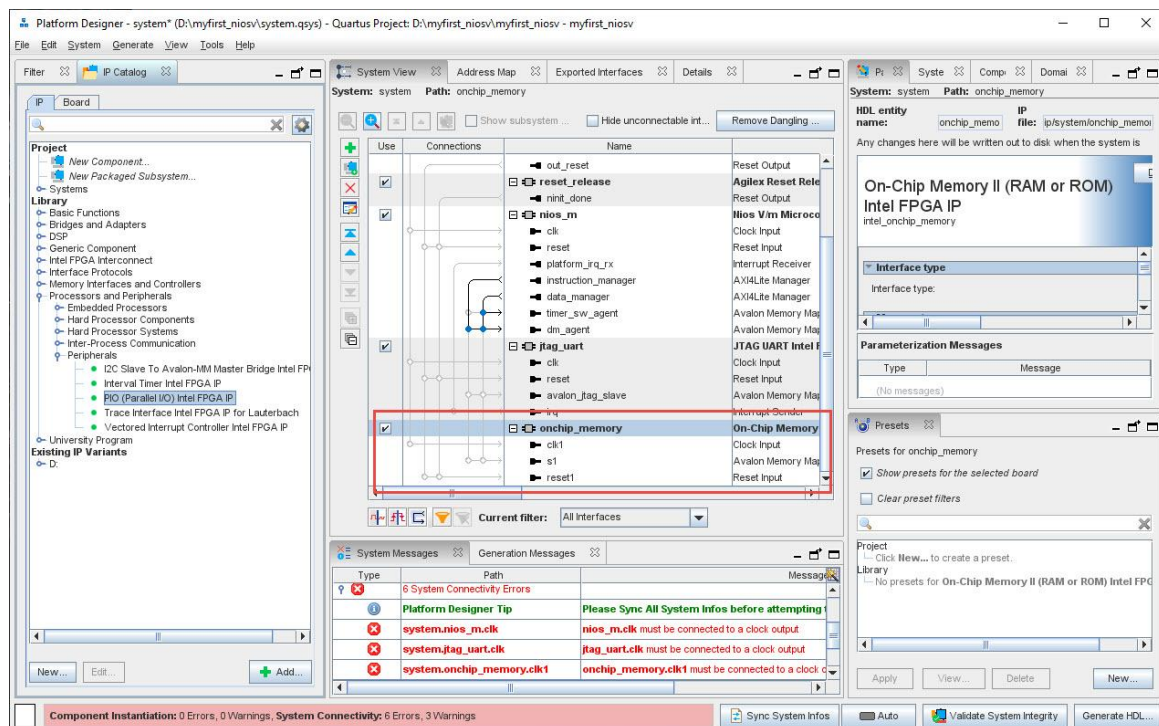


Figure 2-17 Add On-Chip memory Completely

7. Choose **Library > Processors and Peripherals > Peripherals > PIO (Parallel I/O)** to open wizard of adding PIO. See Figure 2-18.

In the PIO(Parallel I/O) IP window:

- a. Rename the HDL entity name to “**prio_led**”
- b. Click **Finish**

8. Figure 2-19 show the **prio_led** component is added in the system.

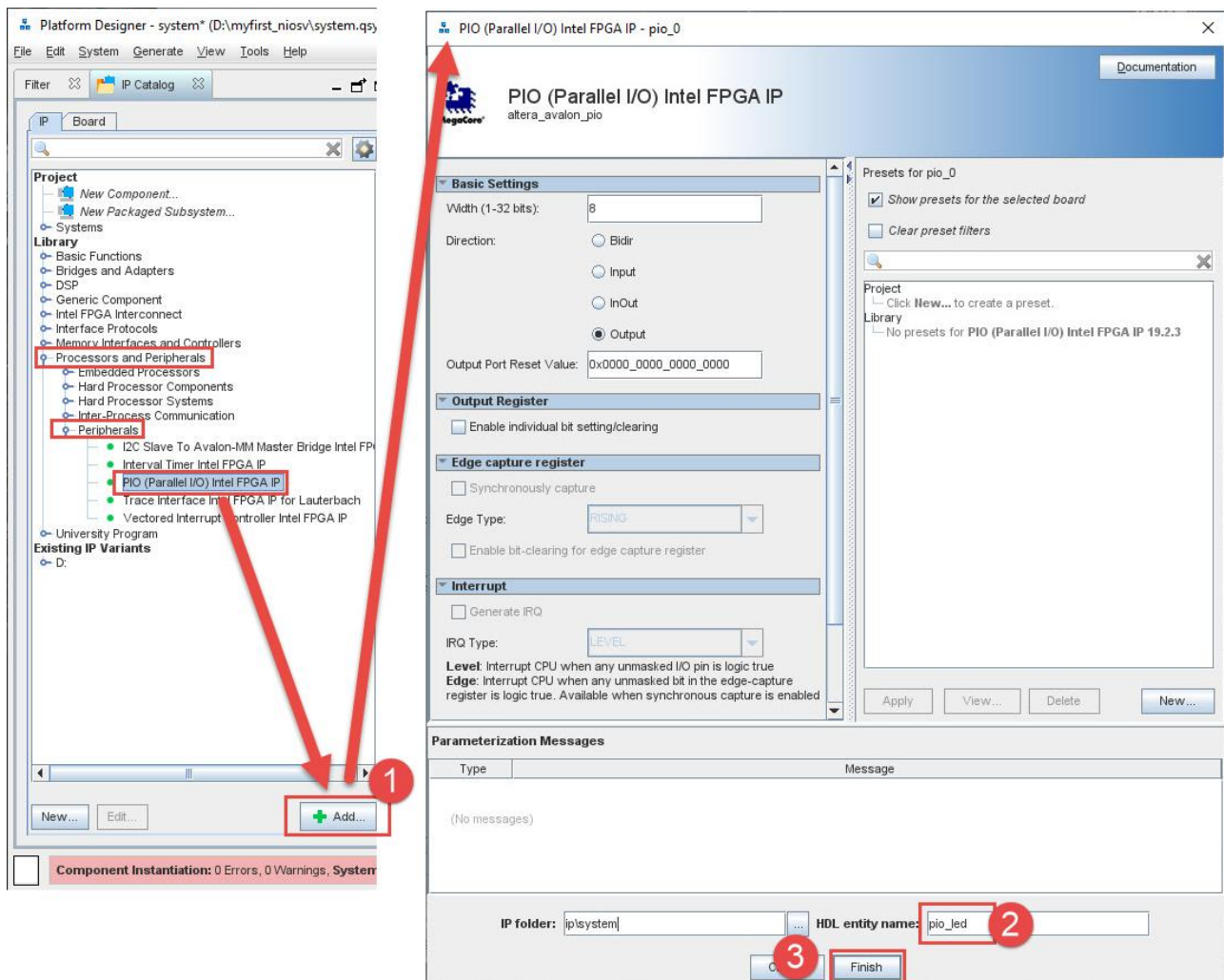


Figure 2-18 Add PIO

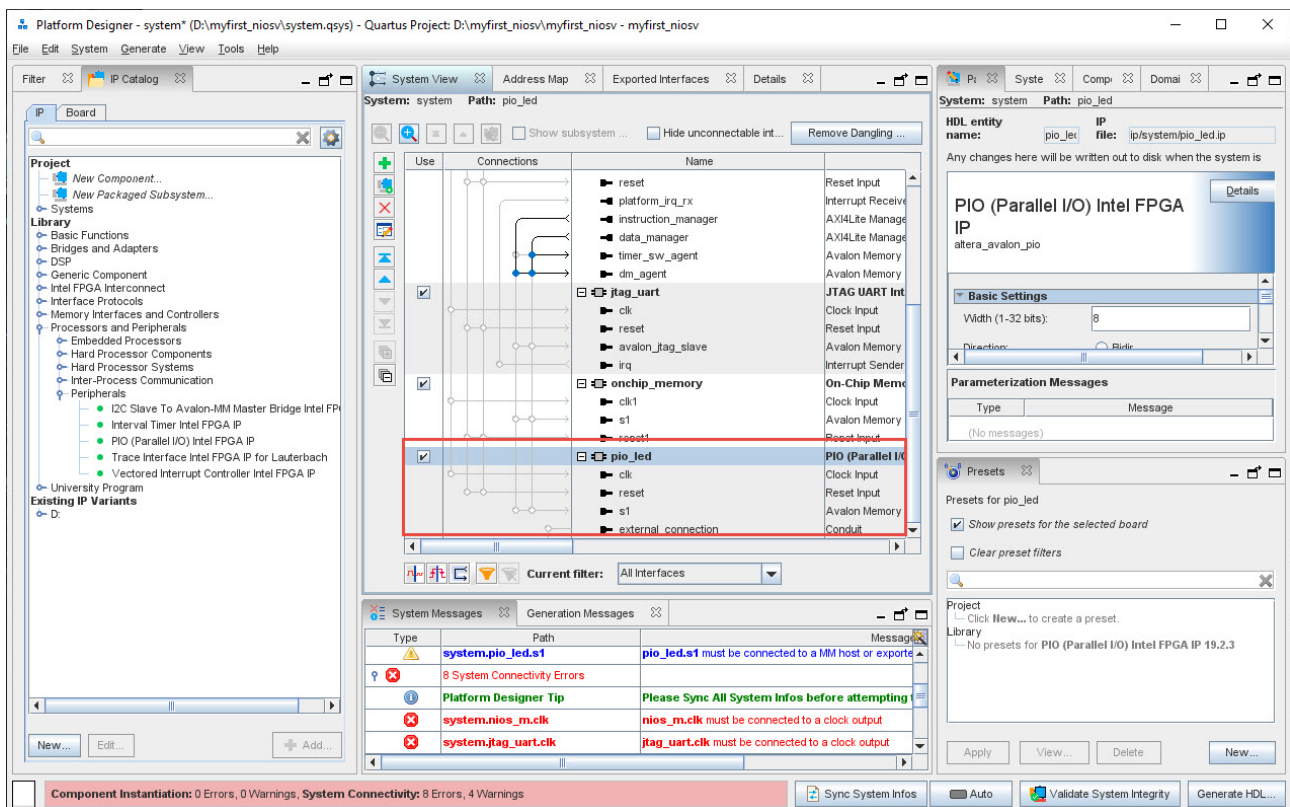


Figure 2-19 PIO

■ Connect Interfaces and Signals

This section explains two ways to connect components in Platform Designer.

- Method 1: Click on an open connection circle to link compatible interfaces. Once connected, the line turns black and the circle is filled. Clicking a filled circle will remove the connection.

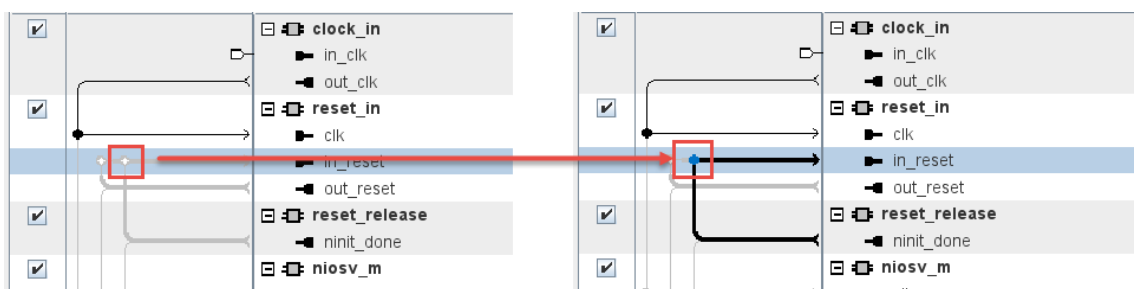


Figure 2-20 Connect interfaces and signals (1)

- Method 2:

In the Platform Designer window (see Figure 2-21), right-click **ninit_done** of the

reset_release component, then select:

Connections: reset_release.ninit_done > reset_in.in_reset

This connects reset_release.ninit_done to reset_in.in_reset, as shown in the figure.

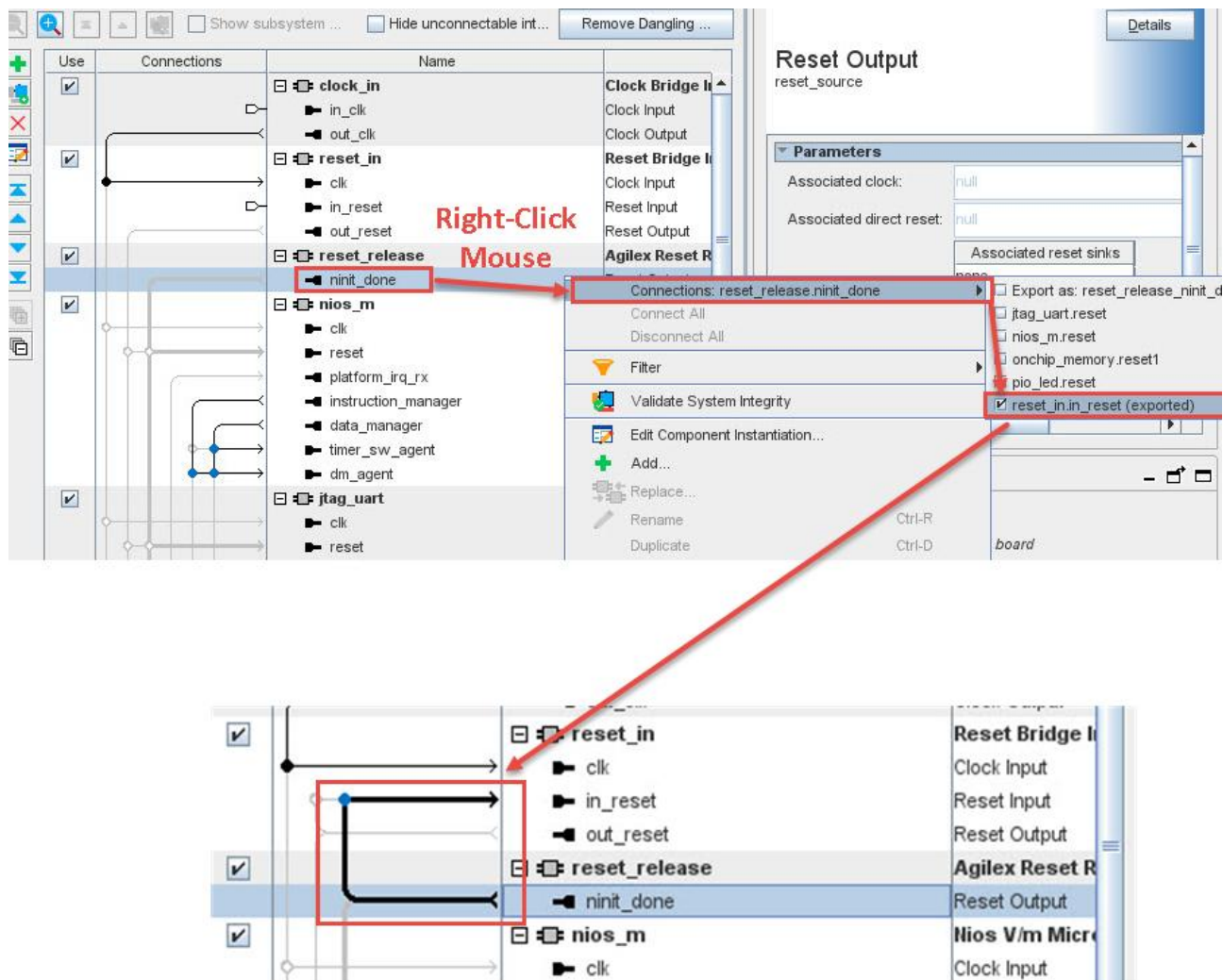


Figure 2-21 Connect interfaces and signals (2)

Use any of the methods above to connect the components in the system as specified in the table below.

Component Name	Signal Name	Connected Component	Connected Signal
reset_release	ninit_done	reset_in	reset_in.in_reset
nios_m	clk	clock_in	clock_in.out_clk
	reset	reset_in	reset_in.out_reset
jtag_uart	clk	clock_in	clock_in.out_clk
	reset	reset_in	reset_in.out_reset
	avalon_jtag_slave	niosv_m	niosv_m.data_manager
	irq	niosv_m	niosv_m.platform_irq_rx
onchip_memory	clk	clock_in	clock_in.out_clk
	s1	niosv_m	niosv_m.data_manager niosv_m.instruction_manager
	reset	reset_in	reset_in.out_reset
pio_led	clk	clock_in	clock_in.out_clk
	reset	reset_in	reset_in.out_reset
	s1	niosv_m	niosv_m.data_manager
	external_connection		Export as: pio_led_external_connection

Choose **external_connection** of **pio_led** and right-click then choose **Connections > Export as: pio_led_external_connection**. (See Figure 2-22)

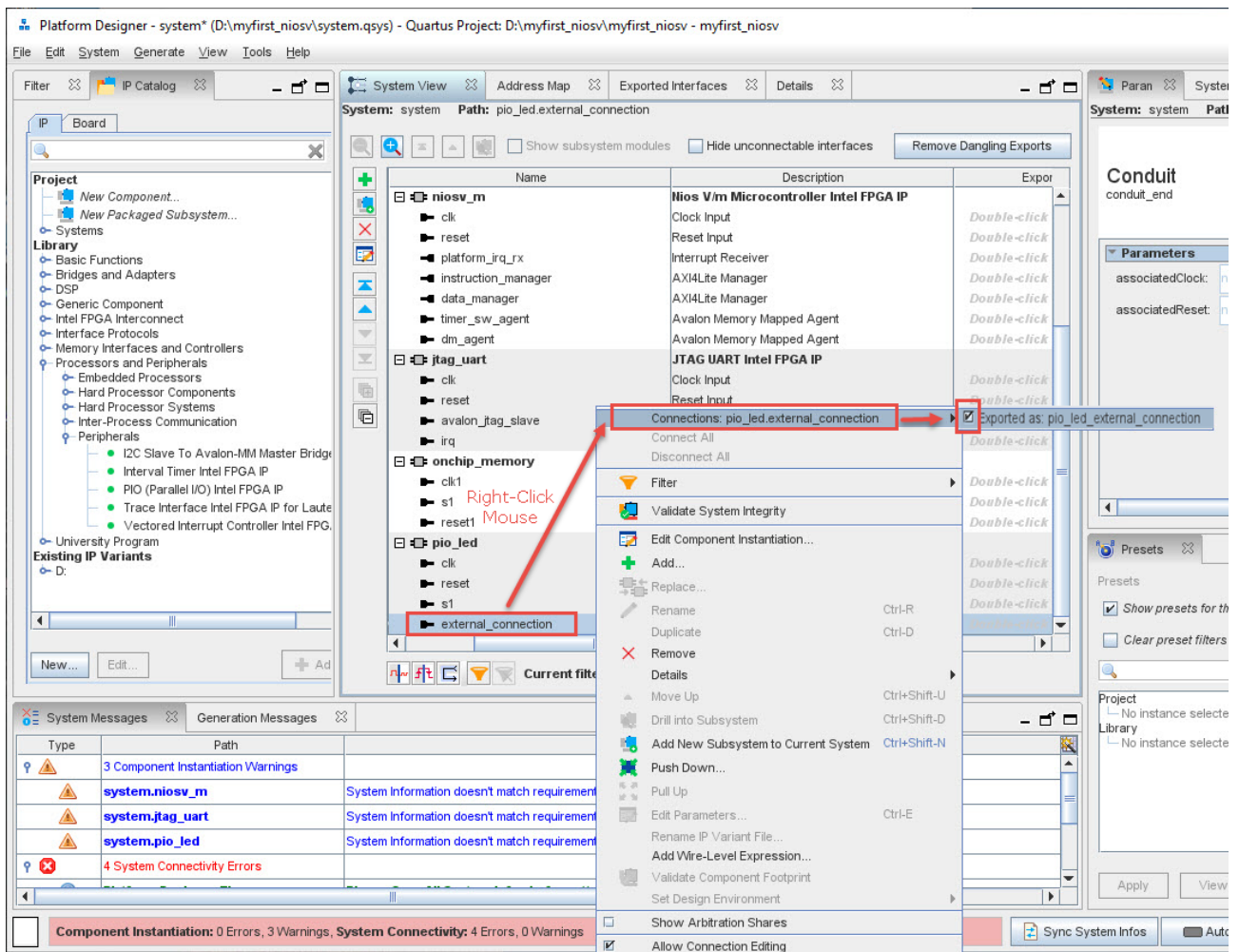


Figure 2-22 Setting external_connection of pio_led

Refer to Figure 2-23 for the final connection configuration.

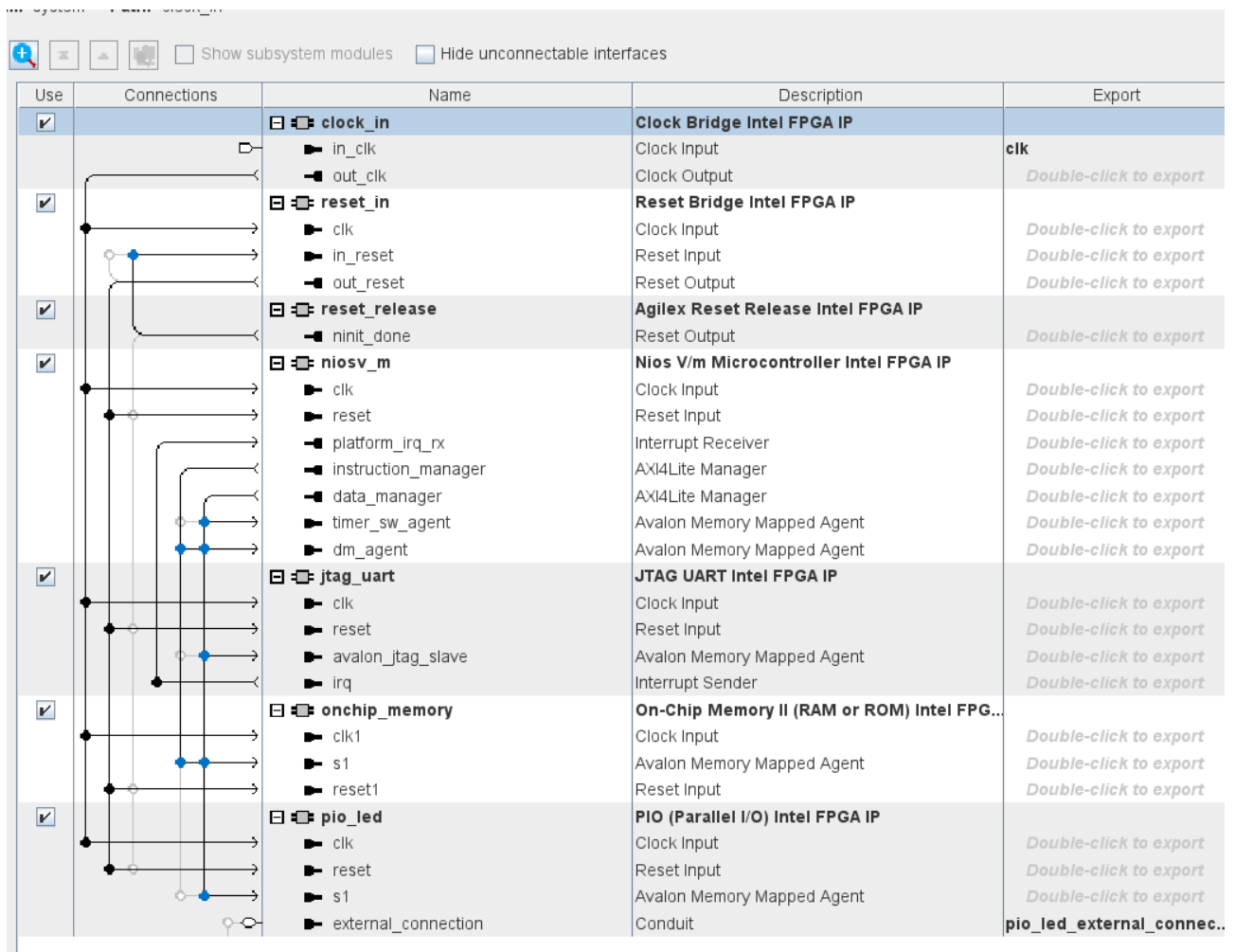
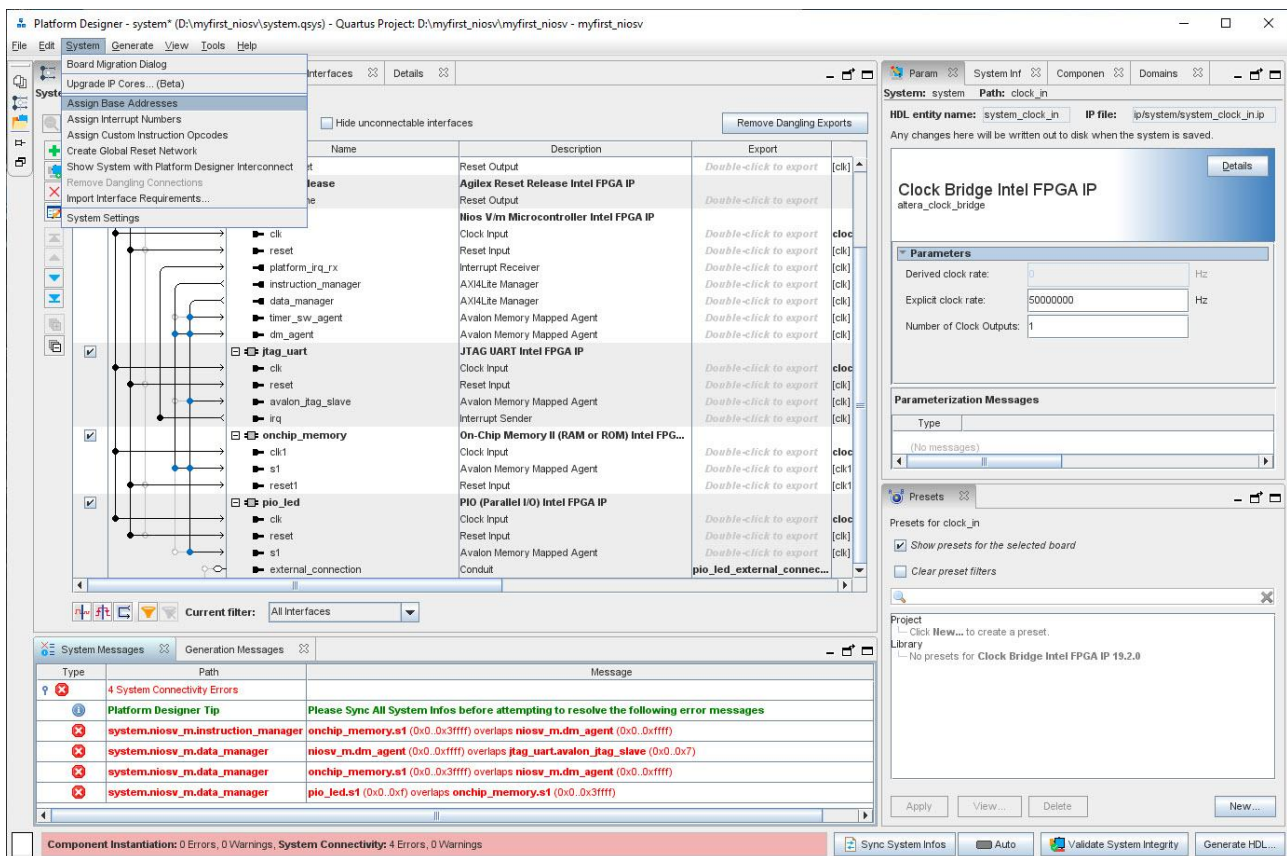


Figure 2-23 final connection configuration

■ Clear System Warnings and Errors

1. Choose **System > Assign Base Addresses** as shown in Figure 2-24. After that, you will find that there is no error in the message window as shown in Figure 2-25.



The screenshot shows the Platform Designer - system* interface. The 'System' menu is open, and 'Assign Base Addresses' is selected. The 'Clock Bridge Intel FPGA IP' configuration panel is visible on the right, showing parameters for the clock input. The 'System Messages' panel at the bottom displays several error messages related to system connectivity and address overlaps.

Type	Path	Message
System Connectivity Errors	4 System Connectivity Errors	Please Sync All System Infos before attempting to resolve the following error messages
Platform Designer Tip		Please Sync All System Infos before attempting to resolve the following error messages
system.niosv_m.instruction_manager	onchip_memory.s1 (0x0..0x3ffff) overlaps niosv_m.dm_agent (0x0..0xffff)	
system.niosv_m.data_manager	niosv_m.dm_agent (0x0..0xffff) overlaps jtag_uart.avalon_jtag_slave (0x0..0x7)	
system.niosv_m.data_manager	onchip_memory.s1 (0x0..0x3ffff) overlaps niosv_m.dm_agent (0x0..0xffff)	
system.niosv_m.data_manager	pio_jed.s1 (0x0..0xf) overlaps onchip_memory.s1 (0x0..0x3ffff)	

Figure 2-24 Assign Base Addresses

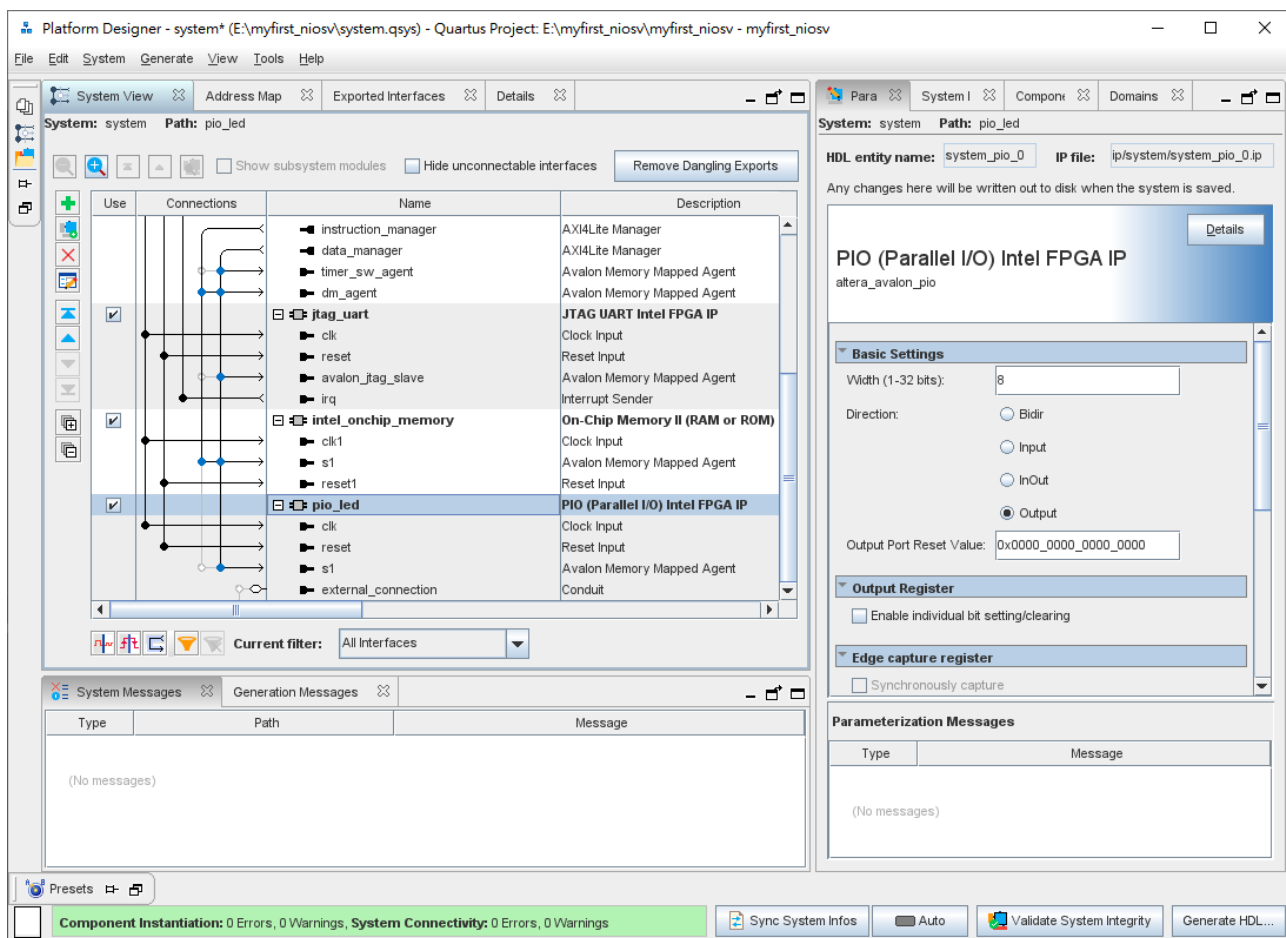


Figure 2-25 No errors or warnings

■ Configuring the Reset Vector of the Nios V Processor

- Click **niosv_m** in the component list on the right part to edit the component. Update **Reset Agent** and **Vector Offset** as shown in Figure 2-26.

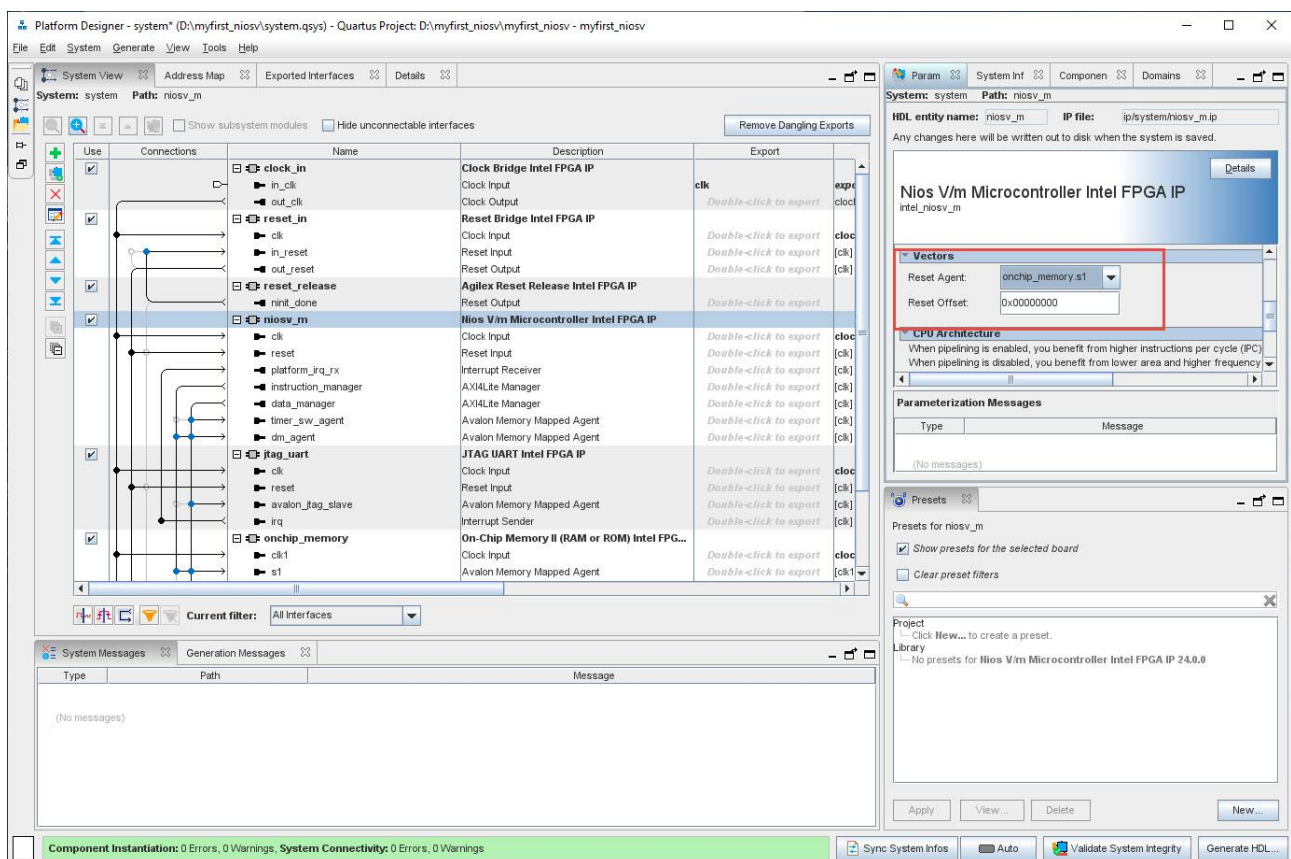


Figure 2-26 Update CPU settings

■ Saving and Generating System HDL

3. Click **Generate** > **Generate HDL** and then pop a window as shown in Figure 2-27. Click **Generate** and then pop a window as shown in Figure 2-28. Click **Save** and the generation start.

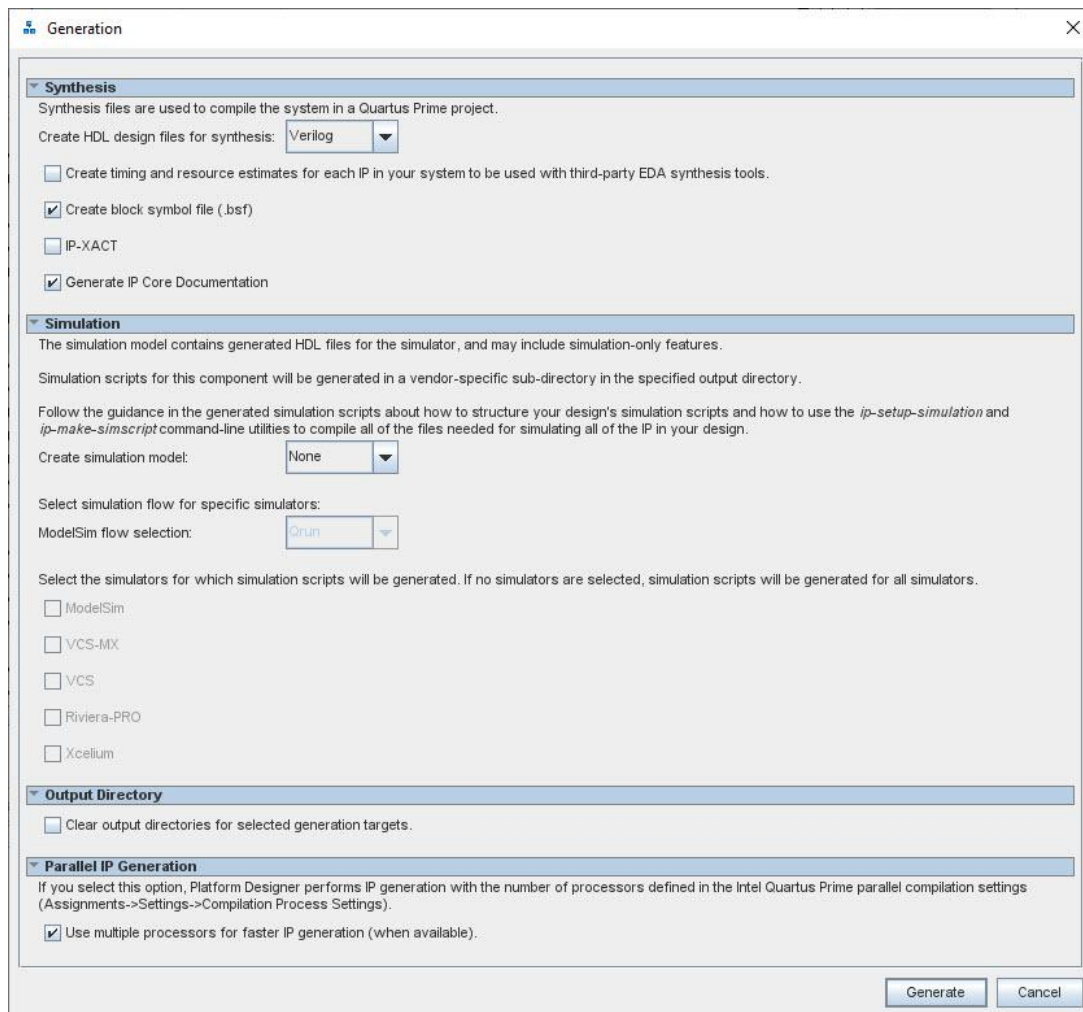


Figure 2-27 Generate Platform Designer

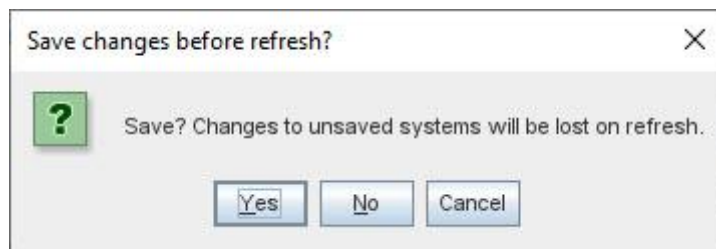


Figure 2-28 Save changes

4. Figure 2-29 shows the generate process. If there is no error in the generation, the window will show successful as shown in Figure 2-30.

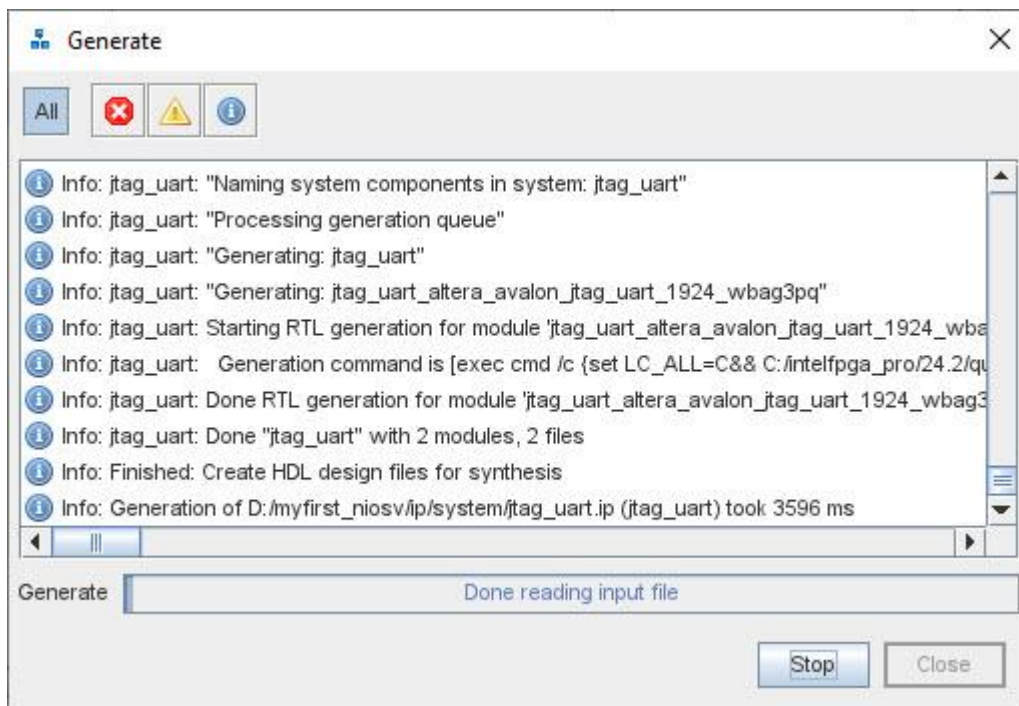


Figure 2-29 Generate Platform Designer

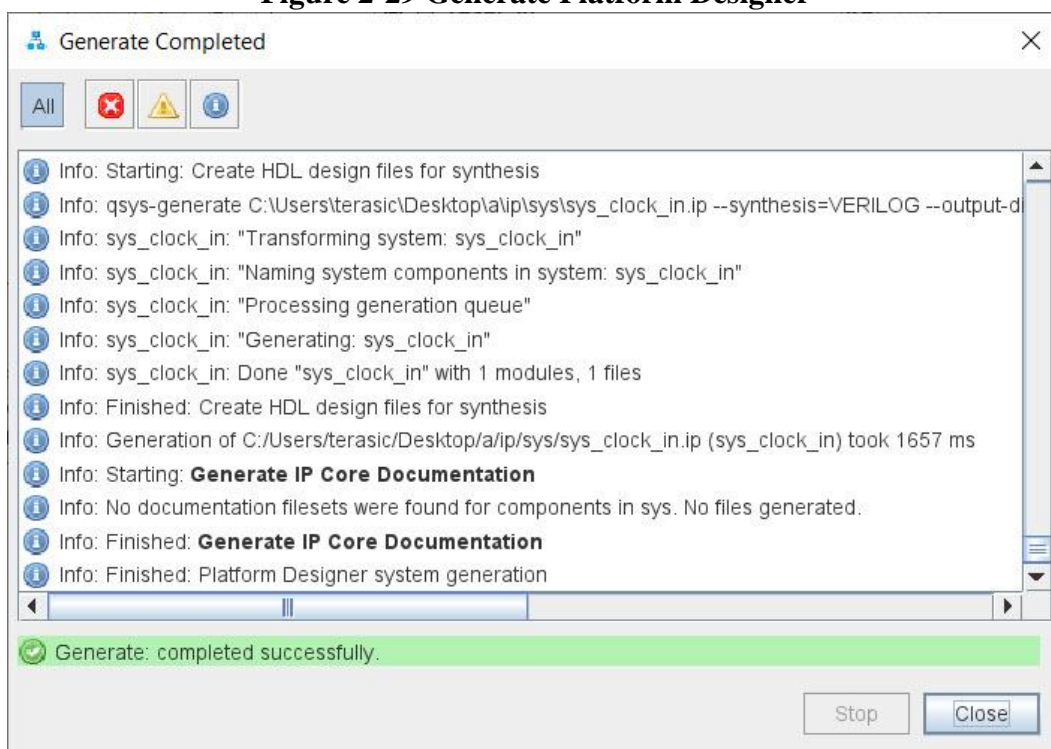


Figure 2-30 Generate Platform Designer Completely

5. Click **Close** and **File > Exit** to exit the Platform Designer and return to the window

2.1.3 Configuring Assignment and Constraint

1. In **Quartus**, choose **File > New** to open new files wizard. See Figure 2-31.

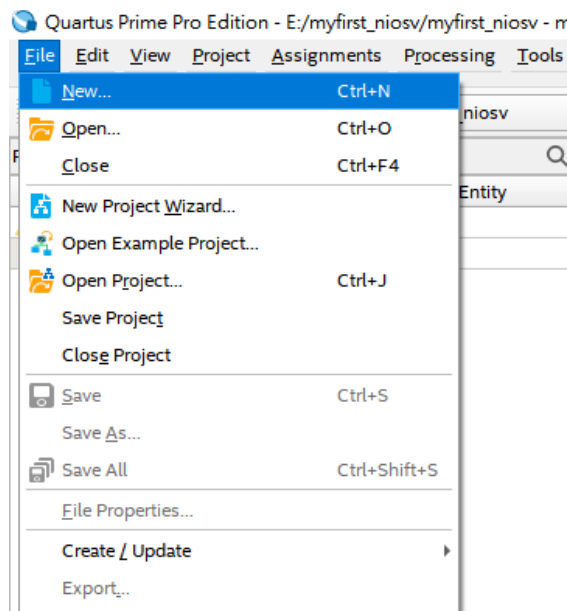


Figure 2-31 New Verilog file

2. Choose **Verilog HDL File** (Figure 2-32.) and click **OK** to return to the window as shown in Figure 2-33. Figure 2-33 show a blank verilog file.

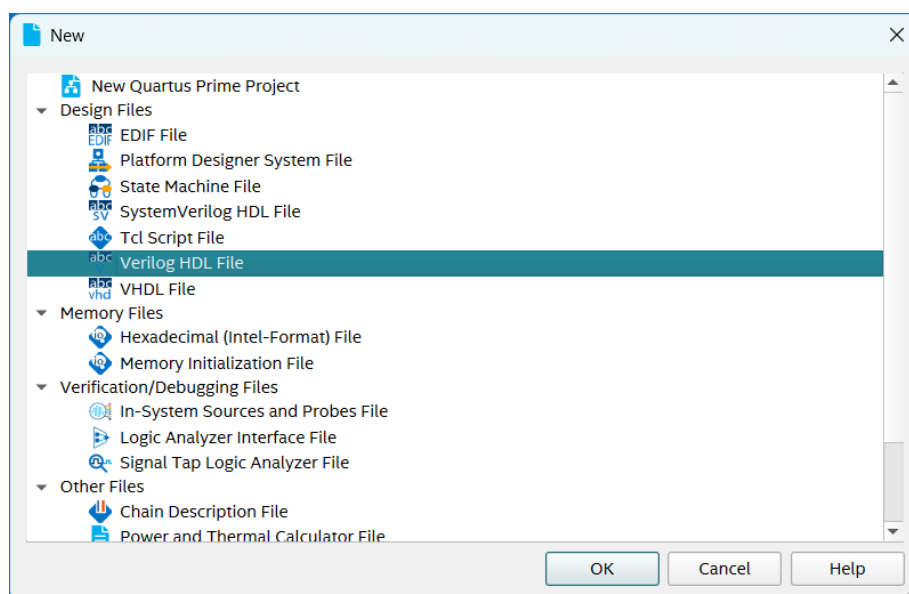


Figure 2-32 New Verilog File

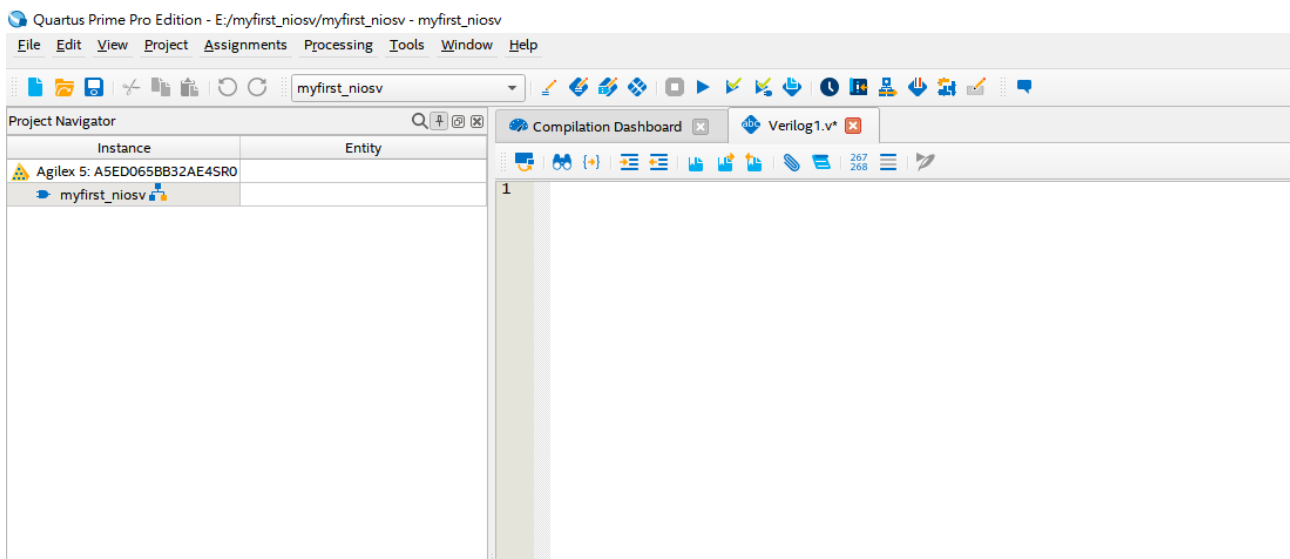


Figure 2-33 A blank verilog file

3. Type verilog the following script as shown in Figure 2 44.

```
module myfirst_niosv
(
    CLOCK_50,
    LED
);
input    CLOCK_50;
output [3:0] LED;
wire [3:0] pio_led;

assign LED = ~pio_led;

system system_inst (
    .clk_clk (CLOCK_50),
    .pio_led_external_connection_export (pio_led)
);

endmodule
```

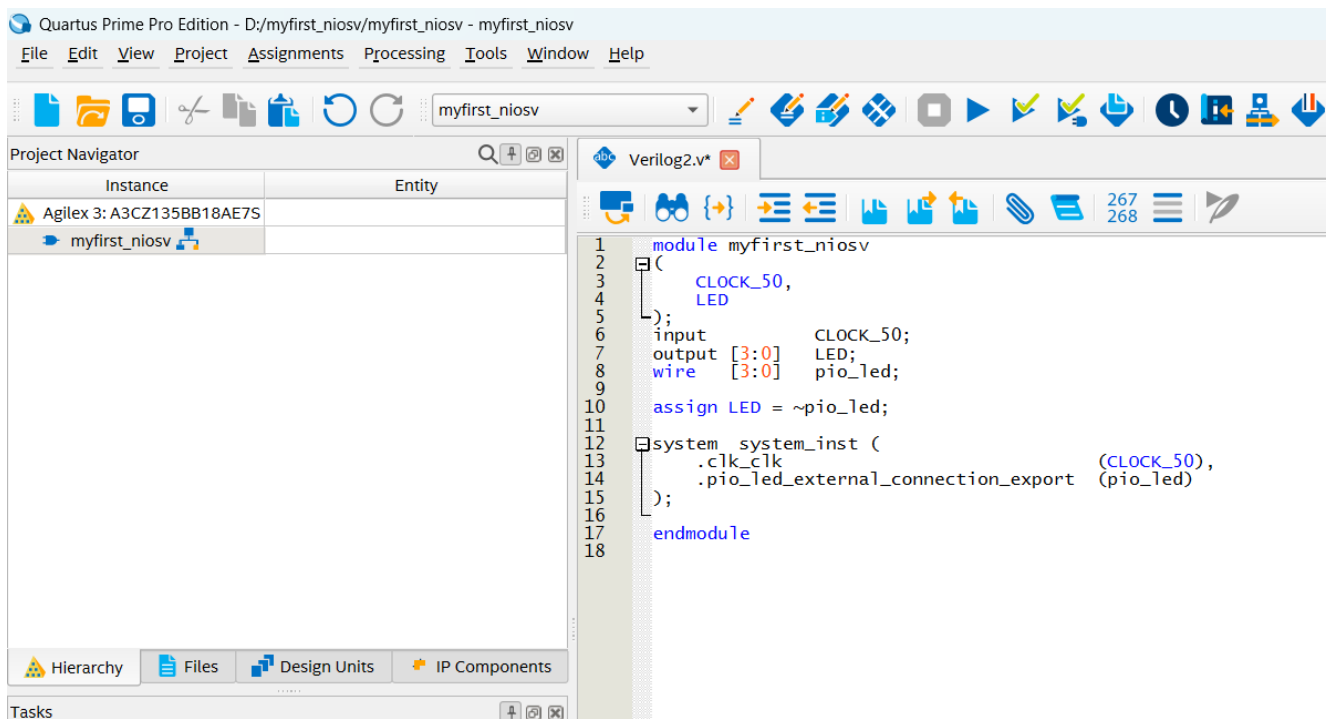


Figure 2-34 Input verilog Text

4. Choose **Save** Icon in the tool bar. There will appear a window as shown in Figure 2-35. Make sure the file name is **myfirst_niosv.v** and click **Save**.

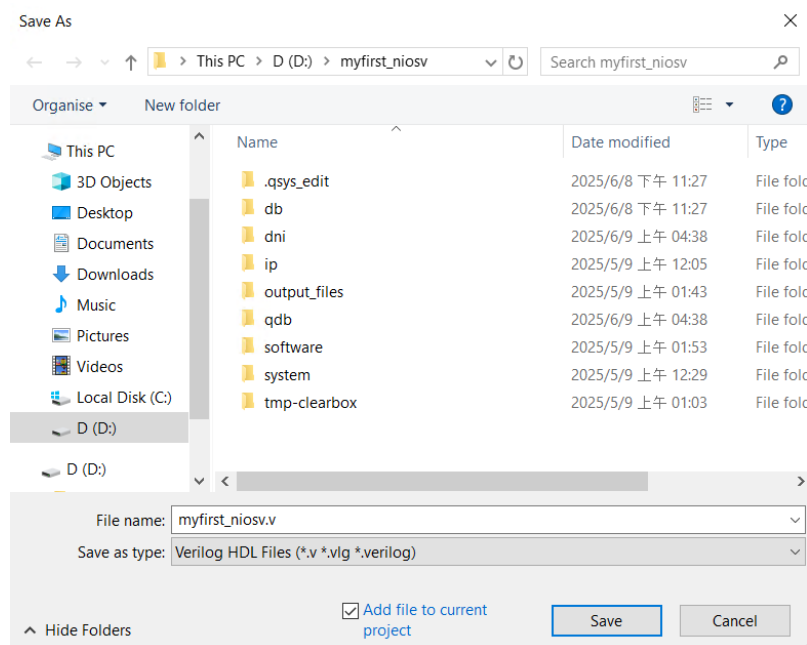


Figure 2-35 Save Verilog file

2.1.4 Compiling the Quartus Prime Software Project

1. Choose **Processing > Start Compilation** as shown in Figure 2-36. Figure 2-37 shows the compilation process.

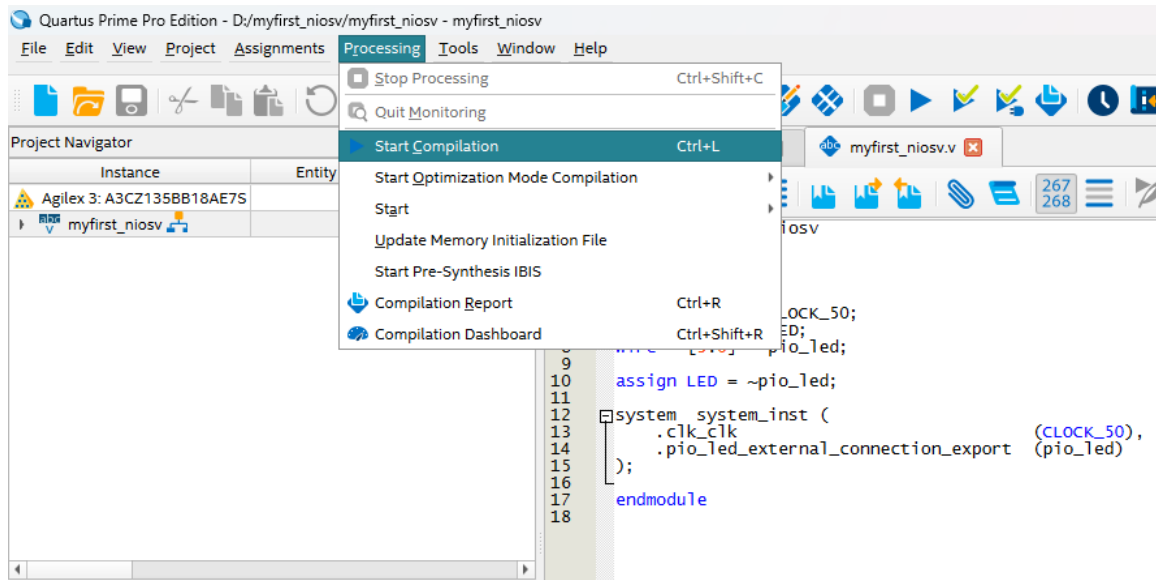


Figure 2-36 Start Compilation

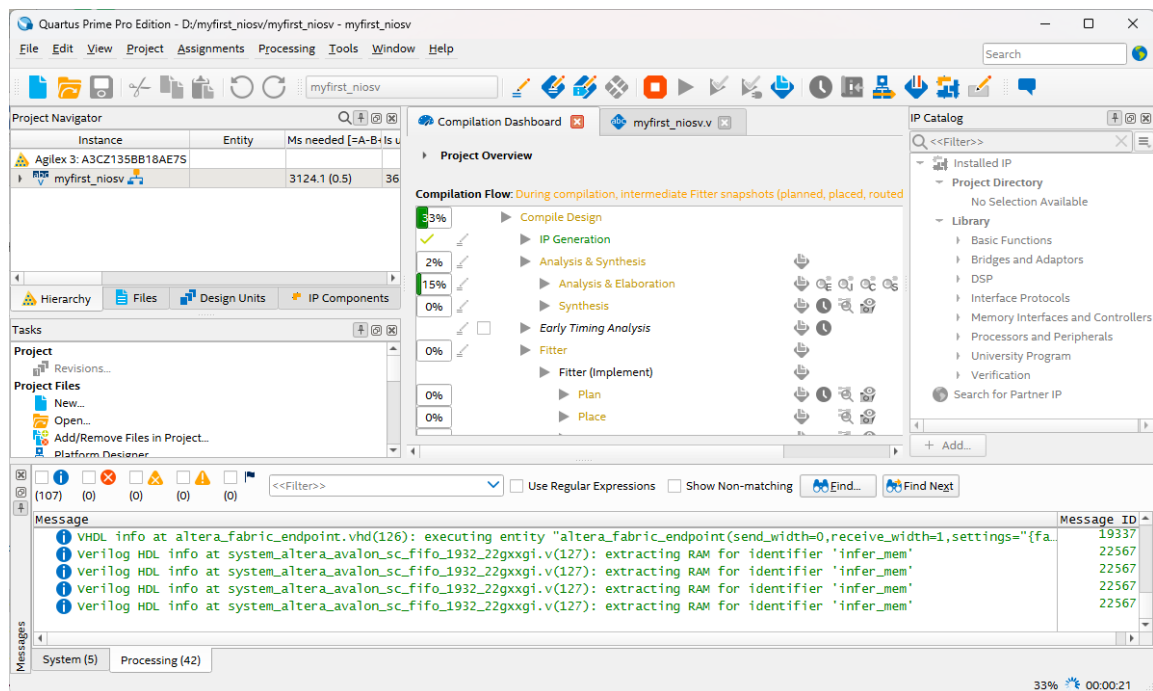


Figure 2-37 Execute Compilation

2. A window that shows successfully will appear as shown in Figure 2-38.

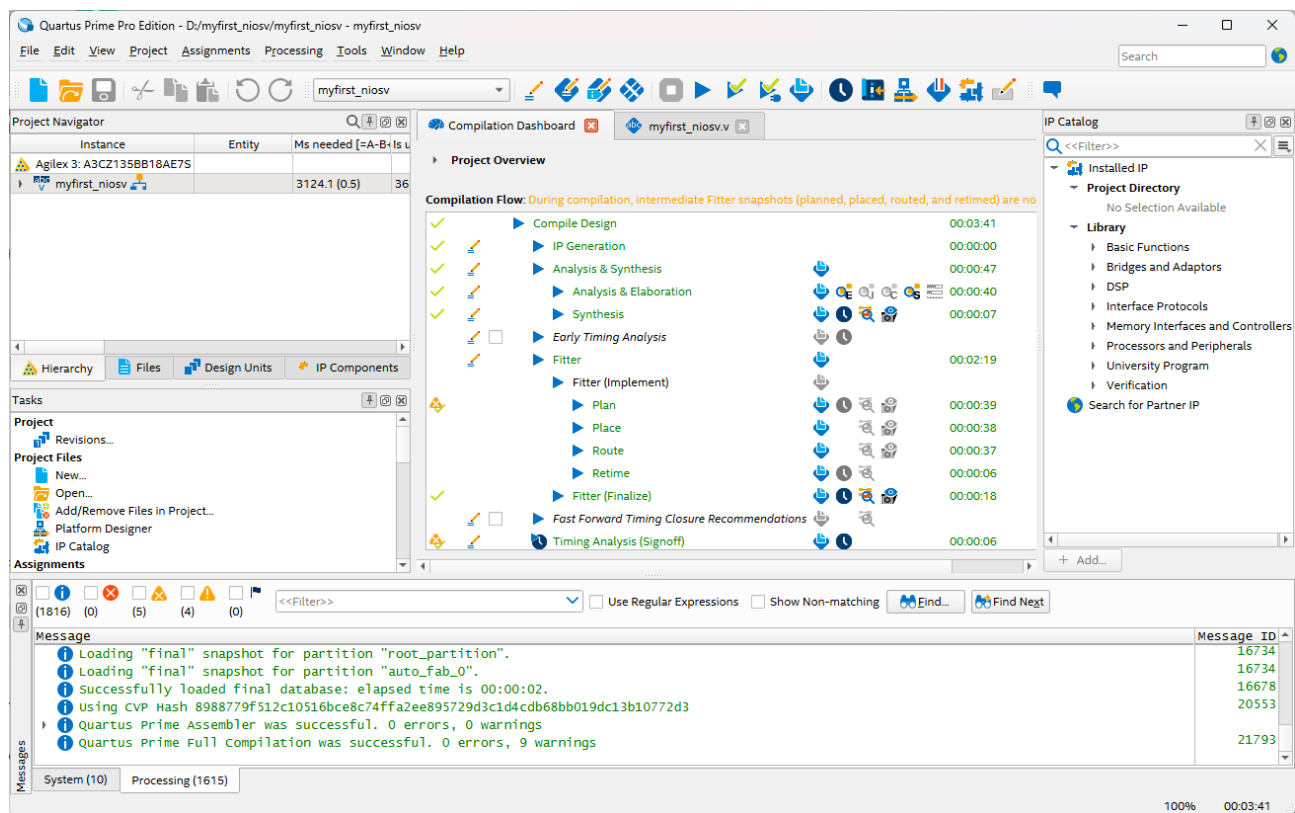


Figure 2-38 Compilation project completely

3. Choose **Assignments** > **Pin Planner** to open pin planner as shown in Figure 2-39. Figure 2-40 show blank pins.

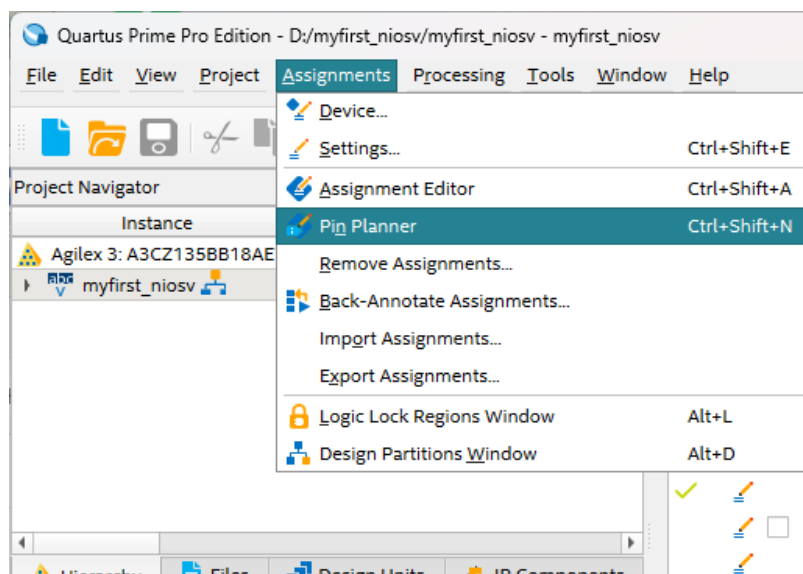
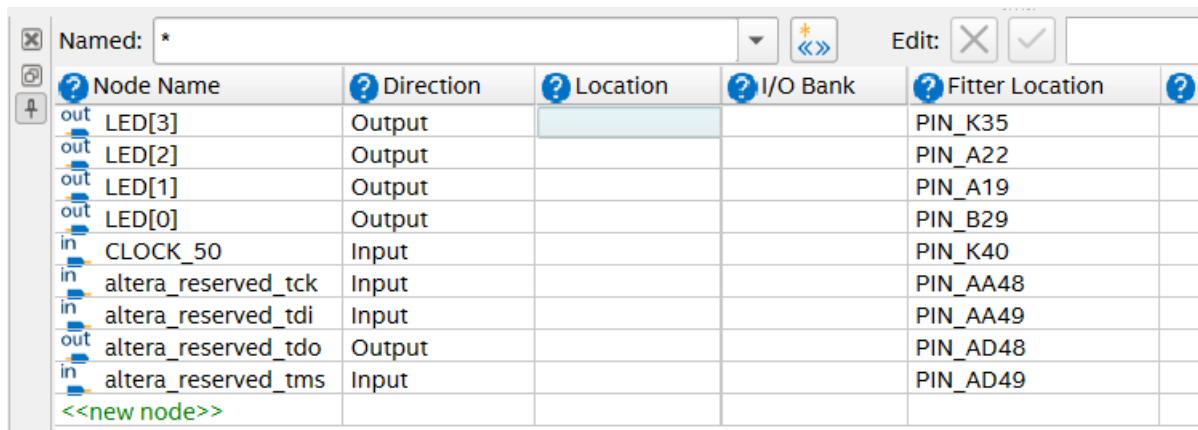


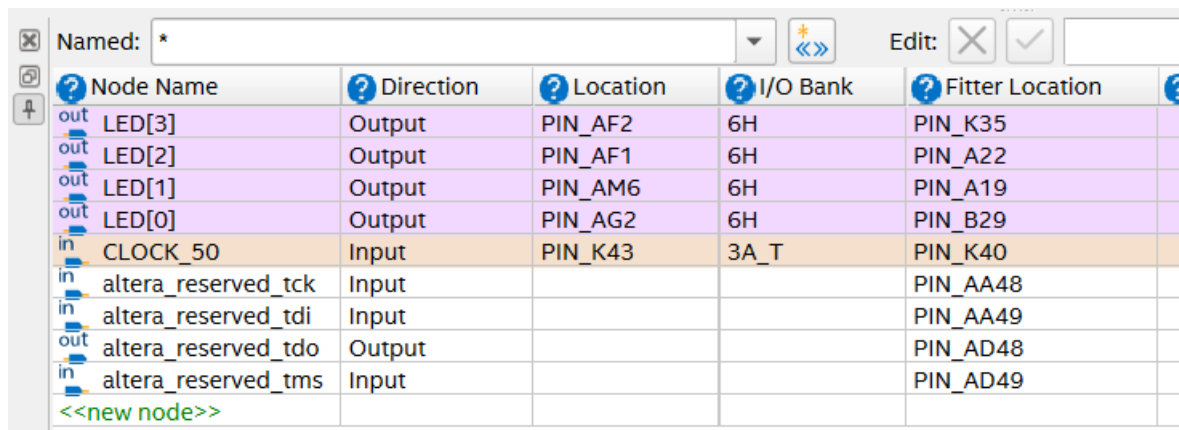
Figure 2-39 Pins menu



Node Name	Direction	Location	I/O Bank	Fitter Location
out LED[3]	Output			PIN_K35
out LED[2]	Output			PIN_A22
out LED[1]	Output			PIN_A19
out LED[0]	Output			PIN_B29
in CLOCK_50	Input			PIN_K40
in altera_reserved_tck	Input			PIN_AA48
in altera_reserved_tdi	Input			PIN_AA49
out altera_reserved_tdo	Output			PIN_AD48
in altera_reserved_tms	Input			PIN_AD49
<<new node>>				

Figure 2-40 Blank Pins

- Input Location value as shown in Figure 2-41.



Node Name	Direction	Location	I/O Bank	Fitter Location
out LED[3]	Output	PIN_AF2	6H	PIN_K35
out LED[2]	Output	PIN_AF1	6H	PIN_A22
out LED[1]	Output	PIN_AM6	6H	PIN_A19
out LED[0]	Output	PIN_AG2	6H	PIN_B29
in CLOCK_50	Input	PIN_K43	3A_T	PIN_K40
in altera_reserved_tck	Input			PIN_AA48
in altera_reserved_tdi	Input			PIN_AA49
out altera_reserved_tdo	Output			PIN_AD48
in altera_reserved_tms	Input			PIN_AD49
<<new node>>				

Figure 2-41 Set Pins

- Close the Pin Planner.
- Restart compilation the project.**

2.2 Download Hardware Design to Target FPGA

This section describes how to download the configuration file to the board.

Download the FPGA configuration file (i.e. the SRAM Object File (.sof) that contains the NIOS V standard system) to the board by performing the following steps:

- Connect the board to the host computer via the USB download cable.
- Apply power to the board.

5. Choose **Tools** -> **Programmer** in Quartus.

6. Click **Hardware Setup** in the top, left corner of the Programmer window. The Hardware Setup dialog box appears.

7. Select **Atum A3 Nano** from the **Currently selected hardware** drop-down list box.

Note: If the appropriate download cable does not appear in the list, you must first install drivers for the cable. Refer to Quartus Prime Pro Help for information on how to install the driver. See Figure 2-42.

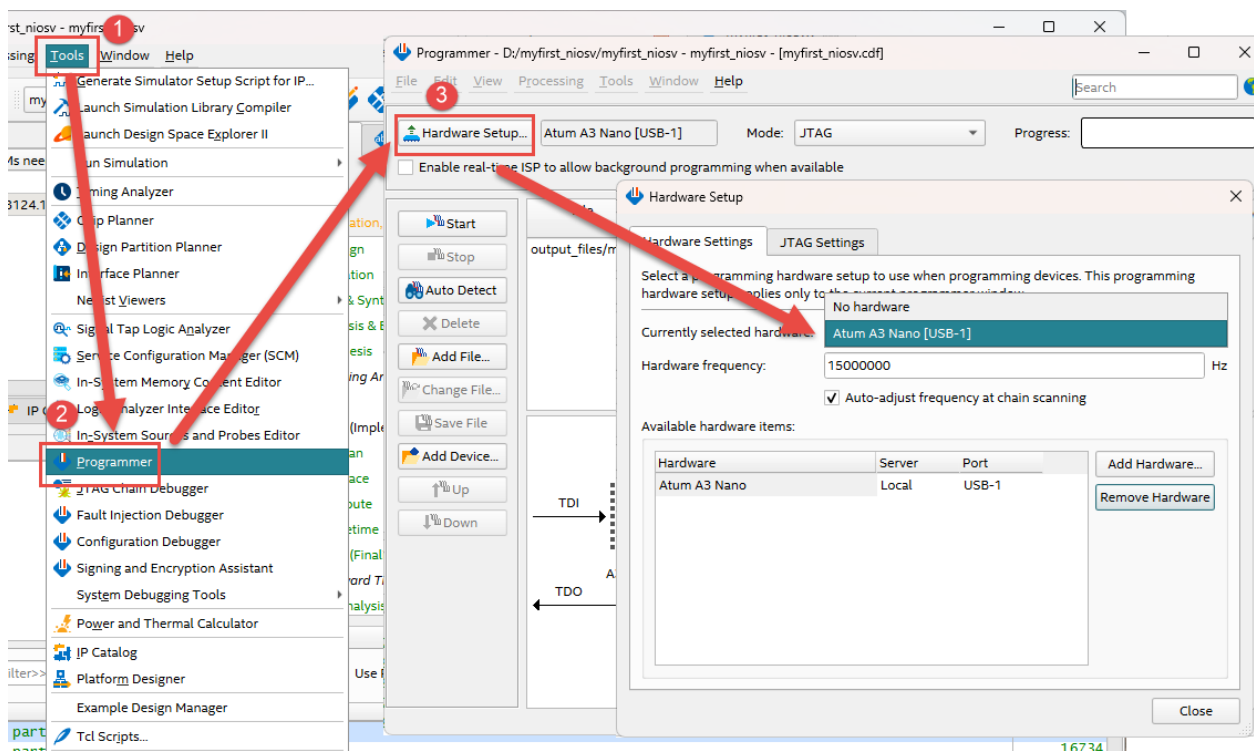


Figure 2-42 Hardware Setup Window

15. Turn on the **Program/Configure** option for the programming file. (See Figure 2-43 for an example).

16. Click **Start**.

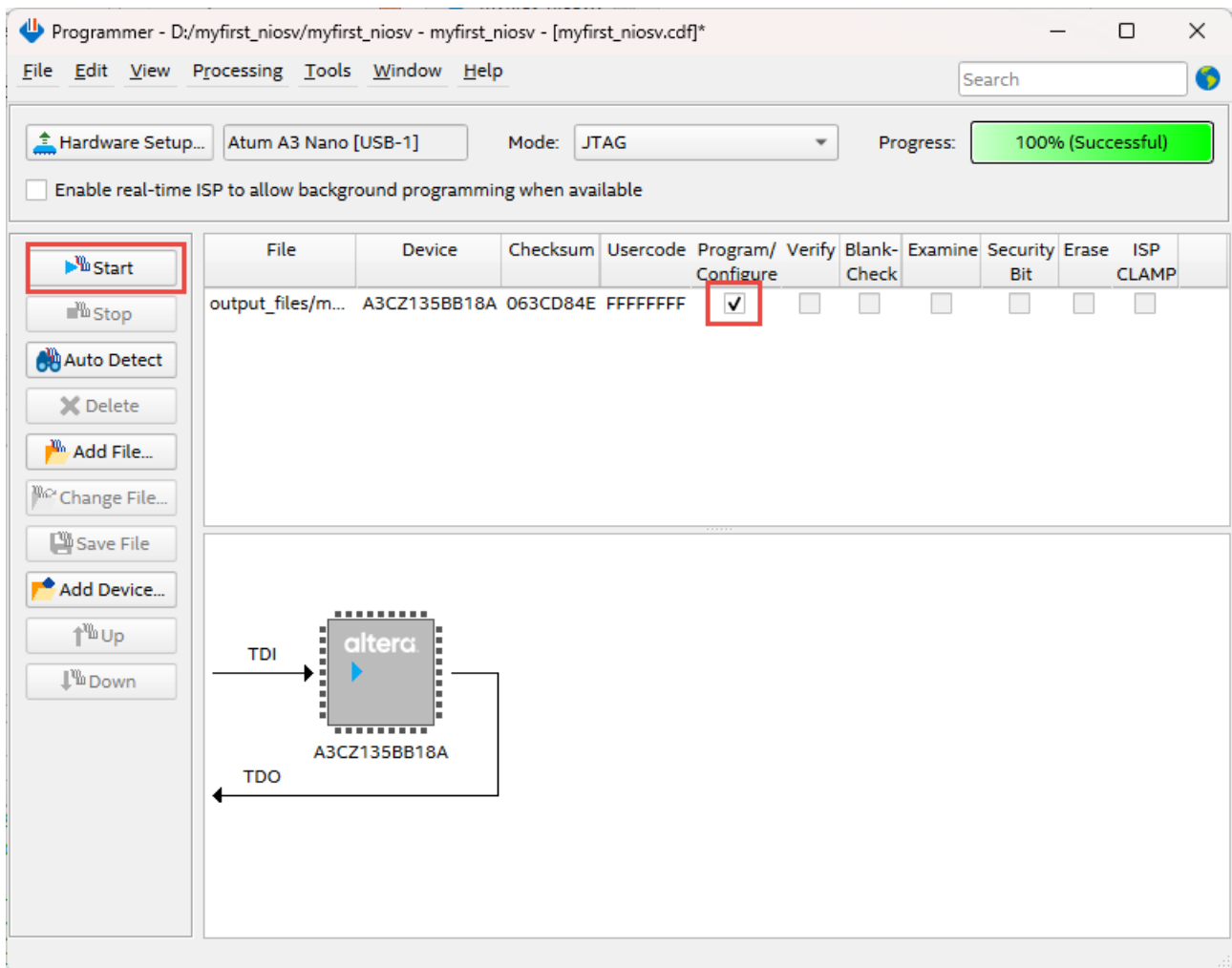


Figure 2-43 Programmer

The Progress meter sweeps to 100% after the configuration finished. When configuration is complete, the FPGA is configured with the Nios V system, but it does not yet have a C program in memory to execute.

Chapter 3

Ashling RiscFree IDE

Build Flow

This chapter covers the build process for C software programs targeting the Nios V processor. The Ashling RiscFree IDE for Altera FPGAs provides an easy-to-use graphical user interface (GUI) that automates the build process and the management of CMakeLists.txt files. The IDE also integrates a text editor and a debugger.

In this chapter, you will use the Ashling RiscFree IDE for Altera FPGAs to compile a simple C example program, which will run on a standard Nios V system configured on the FPGA of your development board. You will create a new software project, build it, and execute it on the target hardware. You will also learn to edit the project, rebuild it, and initiate a debugging session.

3.1 Create the hello_world Example Project

In this section you will create a new NIOS V C/C++ application project. To begin, perform the following steps:

■ **Creating a Board Support Package**

1. In Quartus Prime, select **File > Open** to open `system.qsys`, as shown in the figure below.

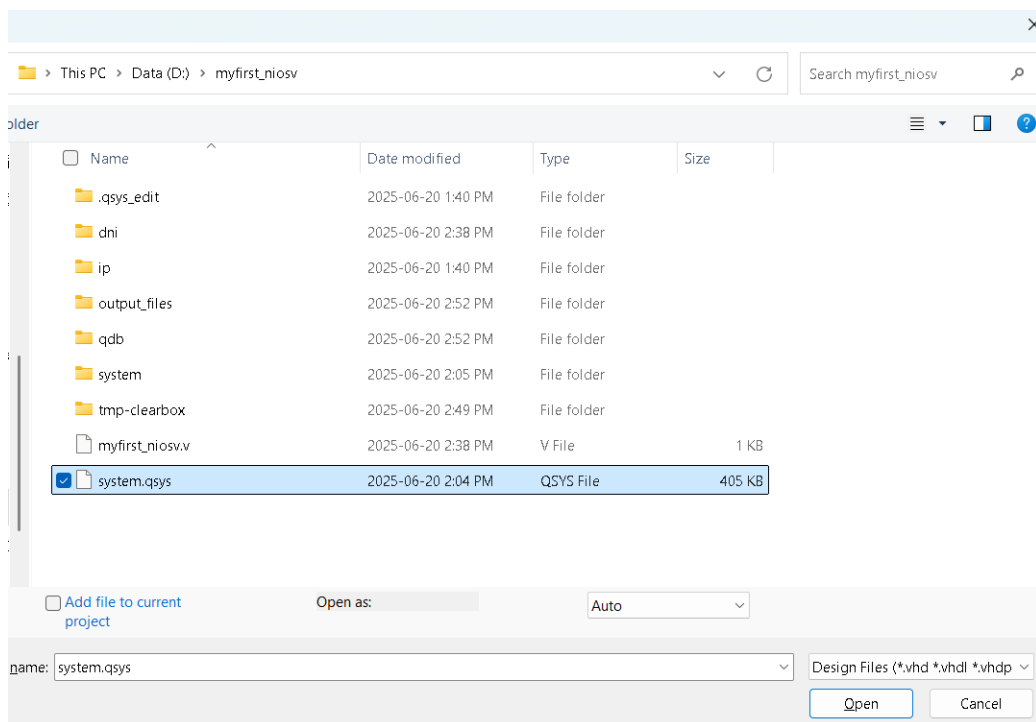


Figure 3-1 Open .qsys

2. Click File > New BSP to create a new BSP, as shown in the figure below.

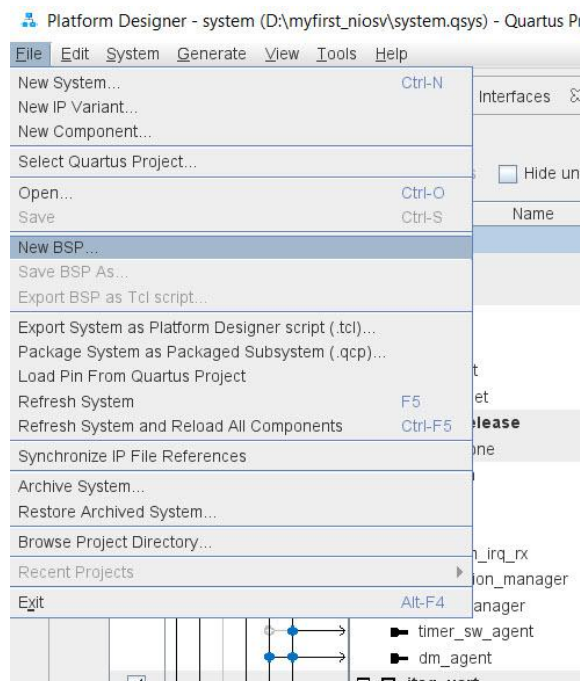


Figure 3-2 Creating a new bsp

3. Create a **software** folder and a **bsp** subfolder within it. Name the file **settings.bsp** and click Create, as shown below.

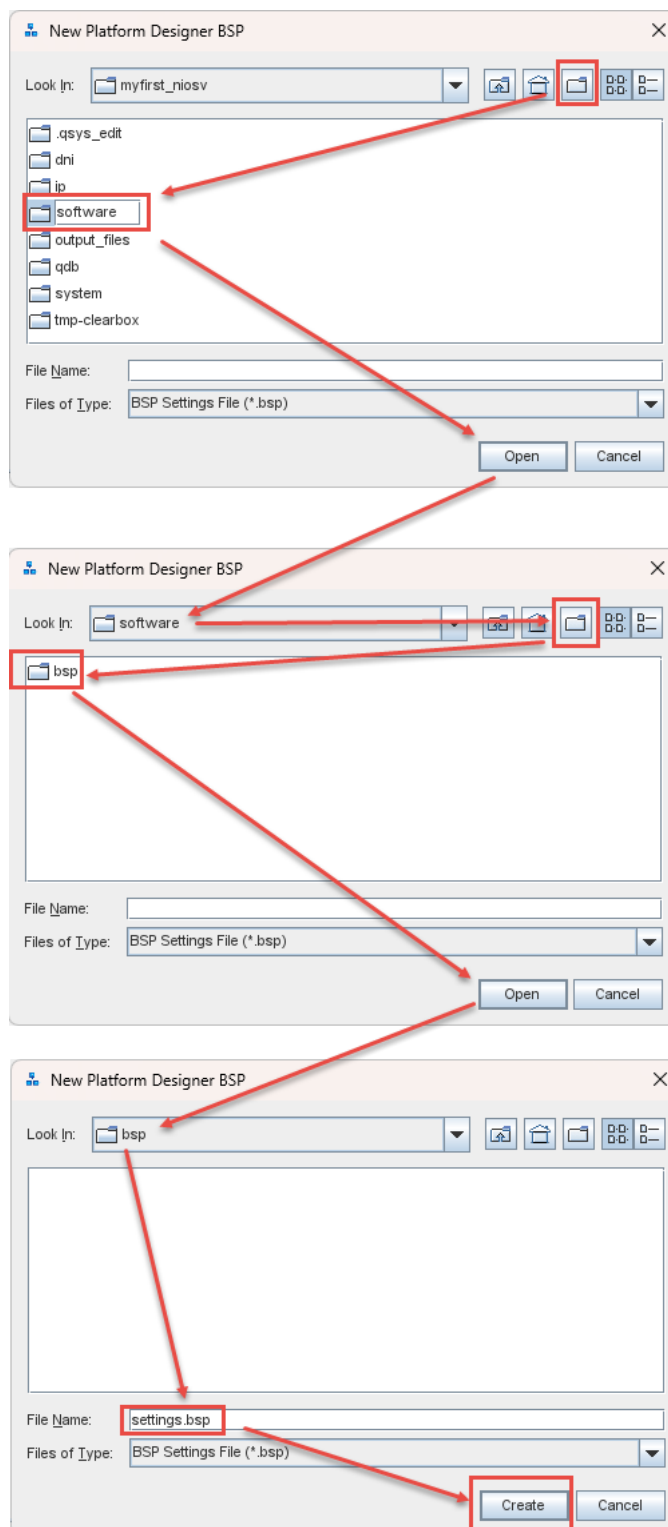


Figure 3-3 Creating a new bsp

4. In the **System File** field, browse to and select **system.qsys**, as shown below.

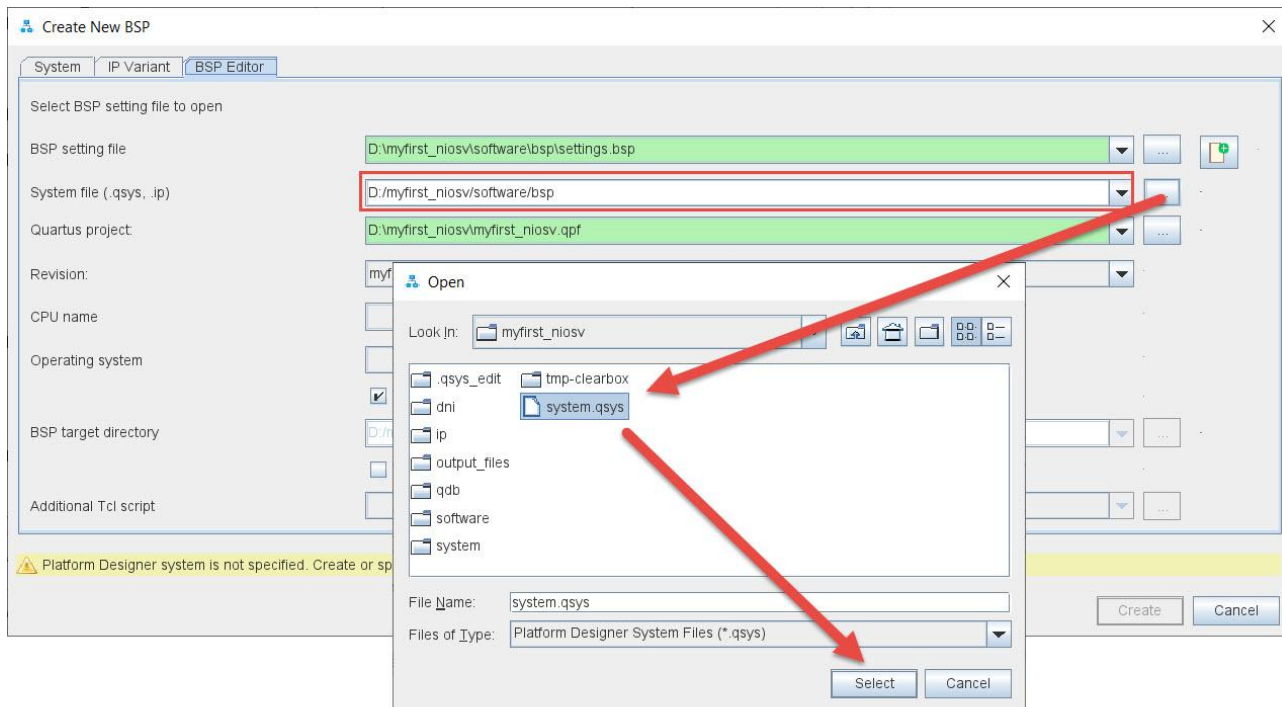


Figure 3-4 Select system file

5. Click **Create** to generate the BSP.

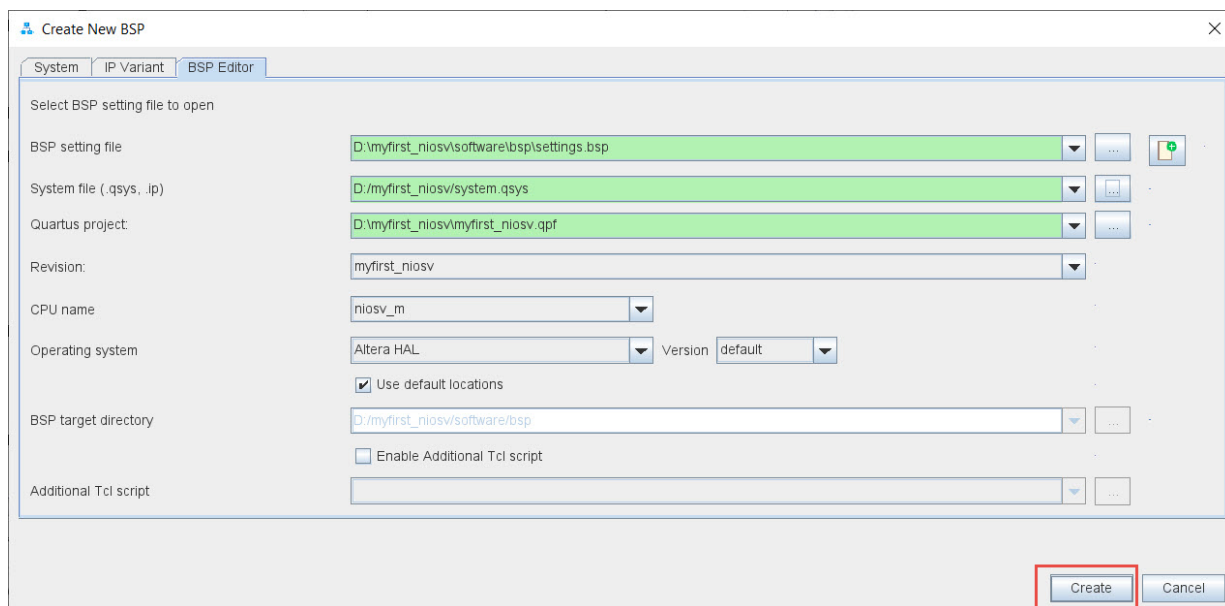


Figure 3-5 Creating bsp

- Double-click the **BSP Editor** tab to maximize it, as shown below. Click **Generate BSP**.

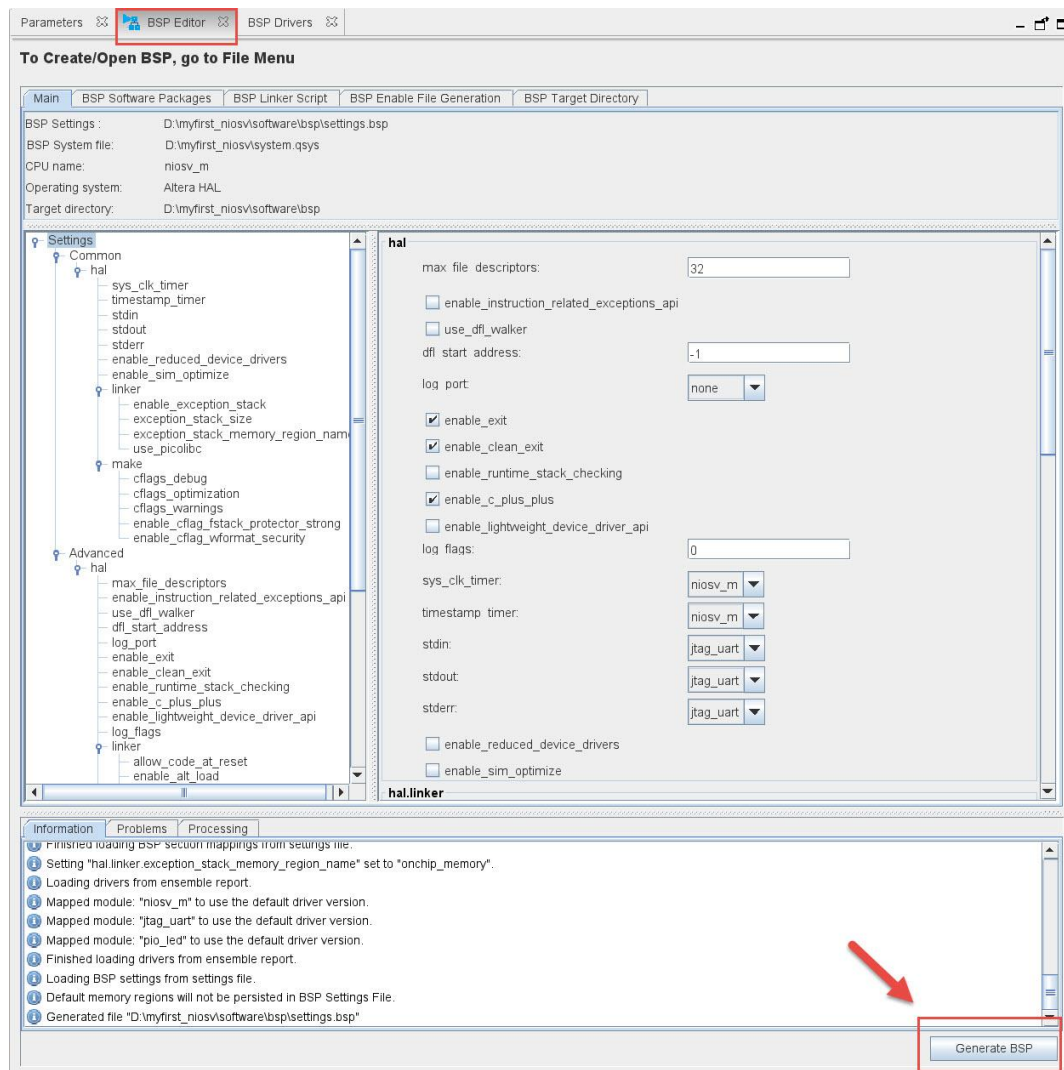


Figure 3-6 Creating bsp

- Launch **Nios V Command Shell** from the Start Menu.

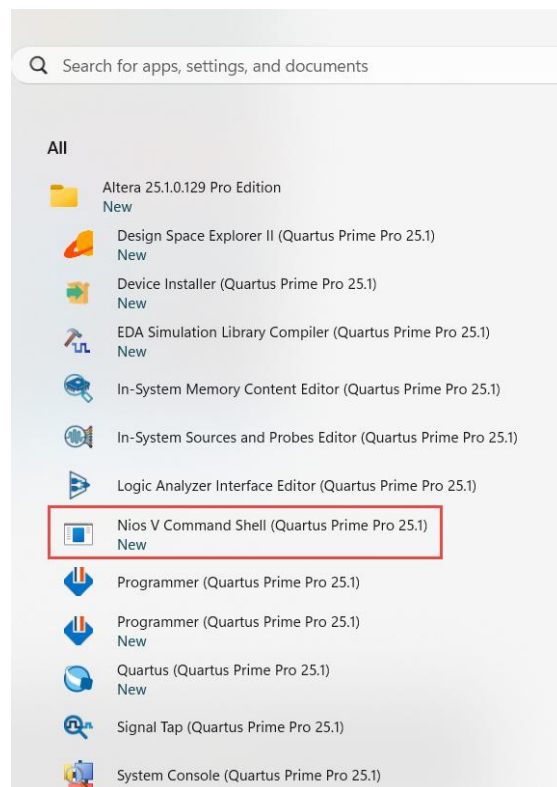


Figure 3-7 Launch Nios V Command Shell

8. Run the command below to switch to the working directory (see Figure 3-8).

```
cd /d <myfirst_niosv path>
```

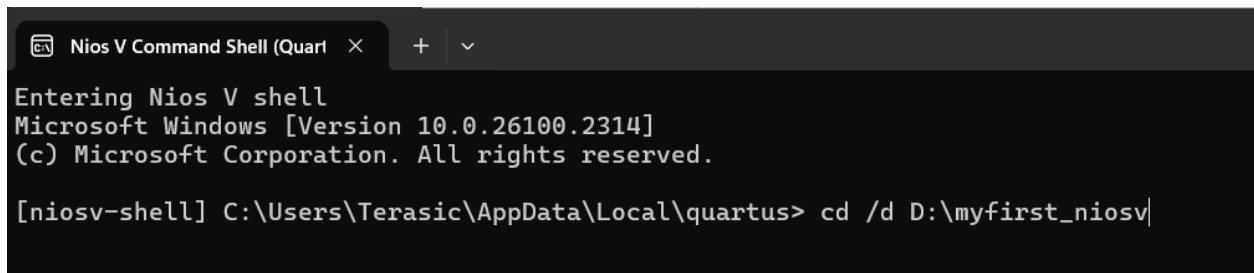


Figure 3-8 Switch working directory

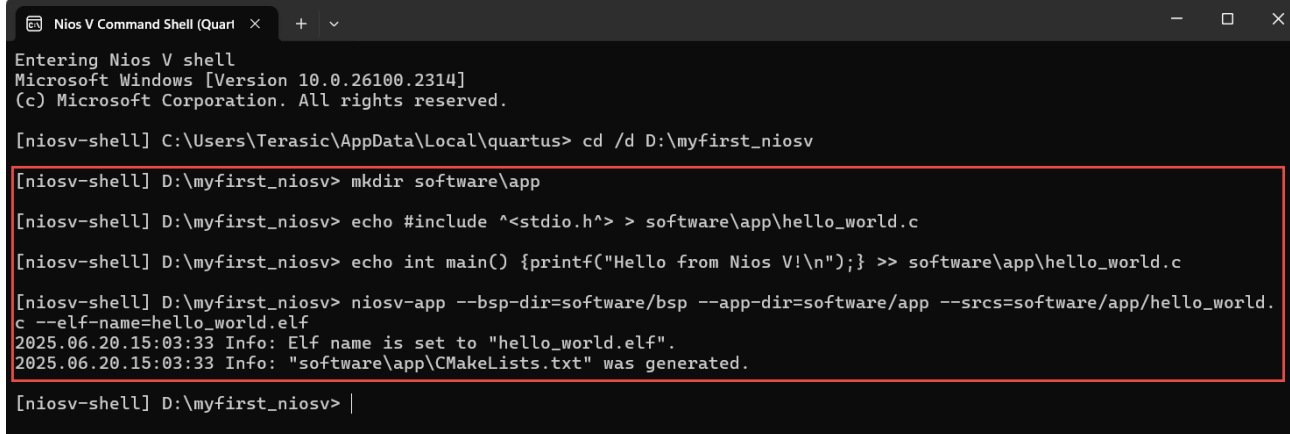
9. Run the commands shown to create the application (see Figure 3-9).

```
mkdir software\app

echo #include ^<stdio.h^> > software\app\hello_world.c

echo int main() {printf("Hello from Nios V!\n");} >> software\app\hello_world.c
```

```
niosv-app --bsp-dir=software/bsp --app-dir=software/app --srcs=software/app/hello_world.c --elf-name=hello_world.elf
```



```

Nios V Command Shell (Quartus)
Entering Nios V shell
Microsoft Windows [Version 10.0.26100.2314]
(c) Microsoft Corporation. All rights reserved.

[niosv-shell] C:\Users\Terasic\AppData\Local\quartus> cd /d D:\myfirst_niosv

[niosv-shell] D:\myfirst_niosv> mkdir software\app

[niosv-shell] D:\myfirst_niosv> echo #include ^<stdio.h^> > software\app\hello_world.c

[niosv-shell] D:\myfirst_niosv> echo int main() {printf("Hello from Nios V!\n");} >> software\app\hello_world.c

[niosv-shell] D:\myfirst_niosv> niosv-app --bsp-dir=software/bsp --app-dir=software/app --srcs=software/app/hello_world.c --elf-name=hello_world.elf
2025.06.20.15:03:33 Info: Elf name is set to "hello_world.elf".
2025.06.20.15:03:33 Info: "software\app\CMakeLists.txt" was generated.

[niosv-shell] D:\myfirst_niosv>
  
```

Figure 3-9 Creating application

■ Importing Projects into Ashling RiscFree IDE for Altera FPGAs

1. User need to install WSL in your windows system.
2. Launch **Ashling RiscFree IDE** for Intel FPGAs from the Start Menu.

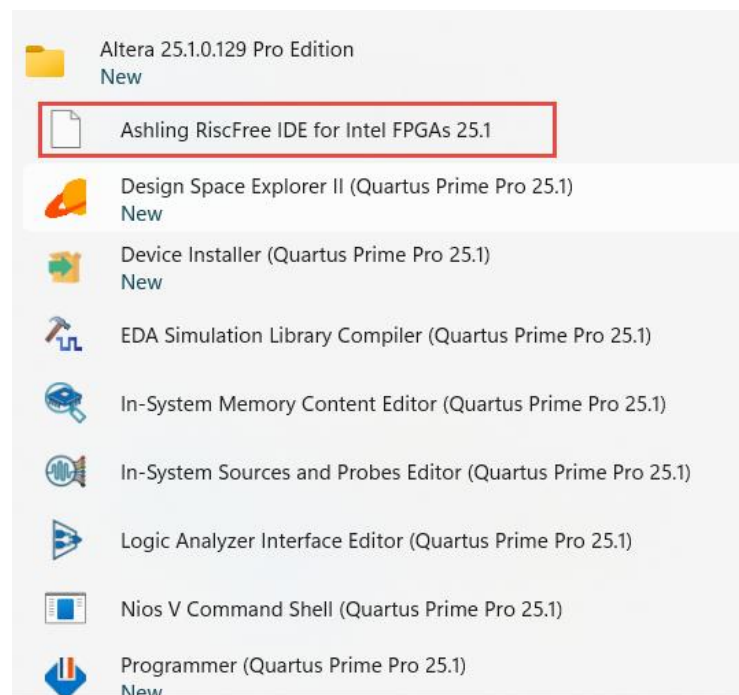


Figure 3-10 Select Workspace

Note: If the user does not see the Ashling RiscFree IDE Shortcut in the Quartus 25.1 Pro folder of the Windows Start Menu, please refer to the link below to resolve the issue : [Resolving the Missing](#)

[Ashling RiscFree IDE Shortcut in the Quartus 25.1 Pro Start Menu](#)

- Then click Launch Set the workspace as shown in Figure 3-11, then click Launch.

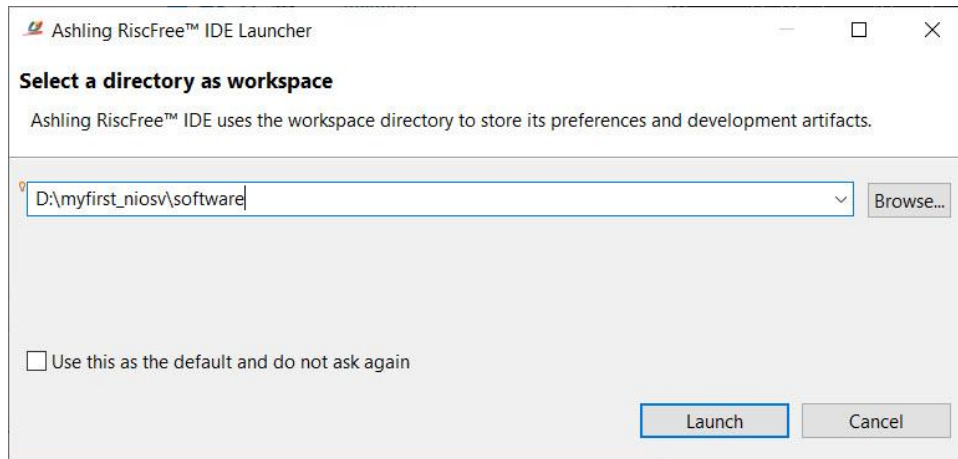


Figure 3-11 Select Workspace

- Choose **File->Import Nios V CMake Project** to open the Import Window.

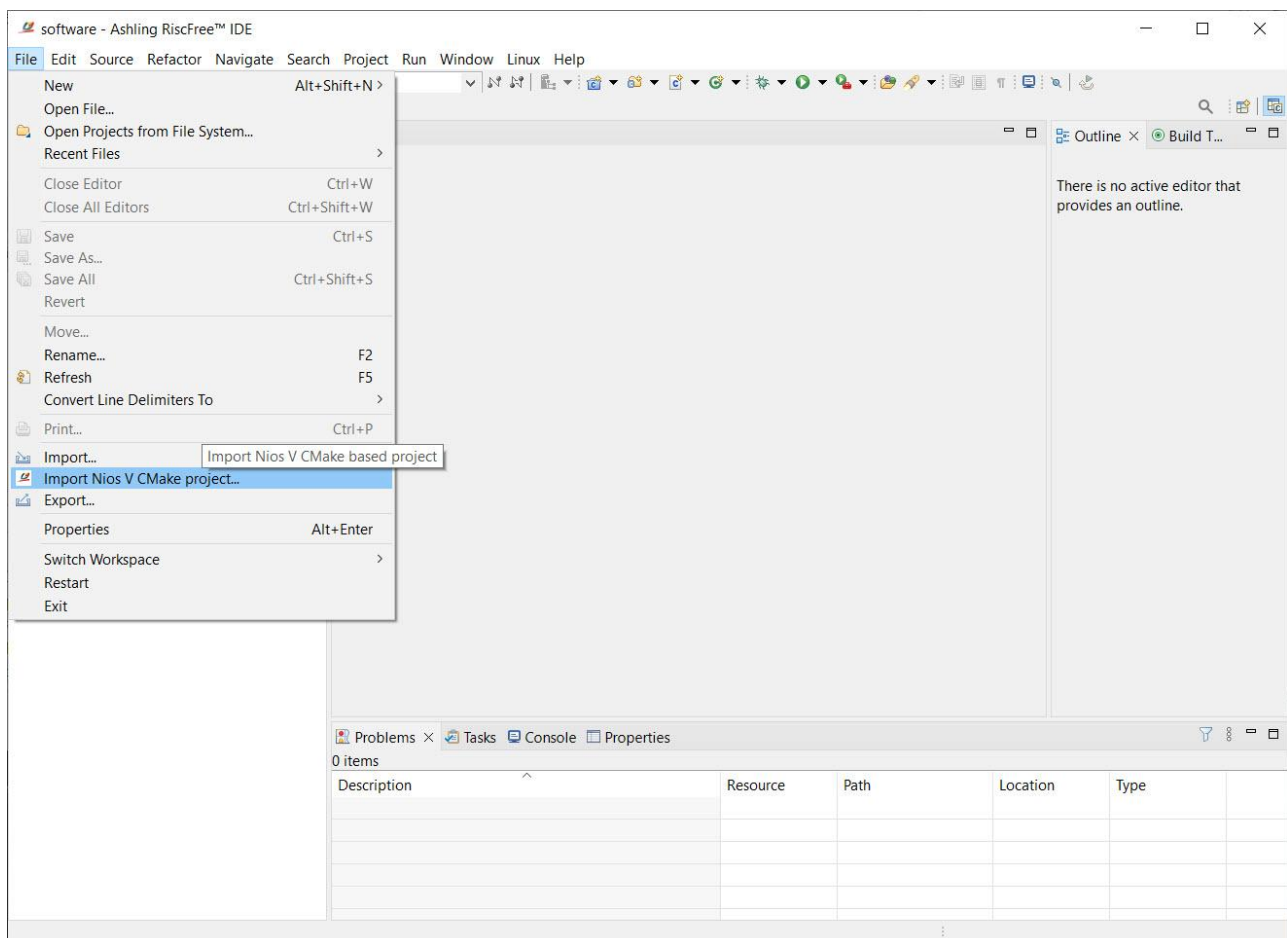


Figure 3-12 Import Nios V CMake Project

5. In the Import Window, make sure the following things:

- Browse and select the **bsp** project folder.
- Give the project a name. (**bsp** is default name)

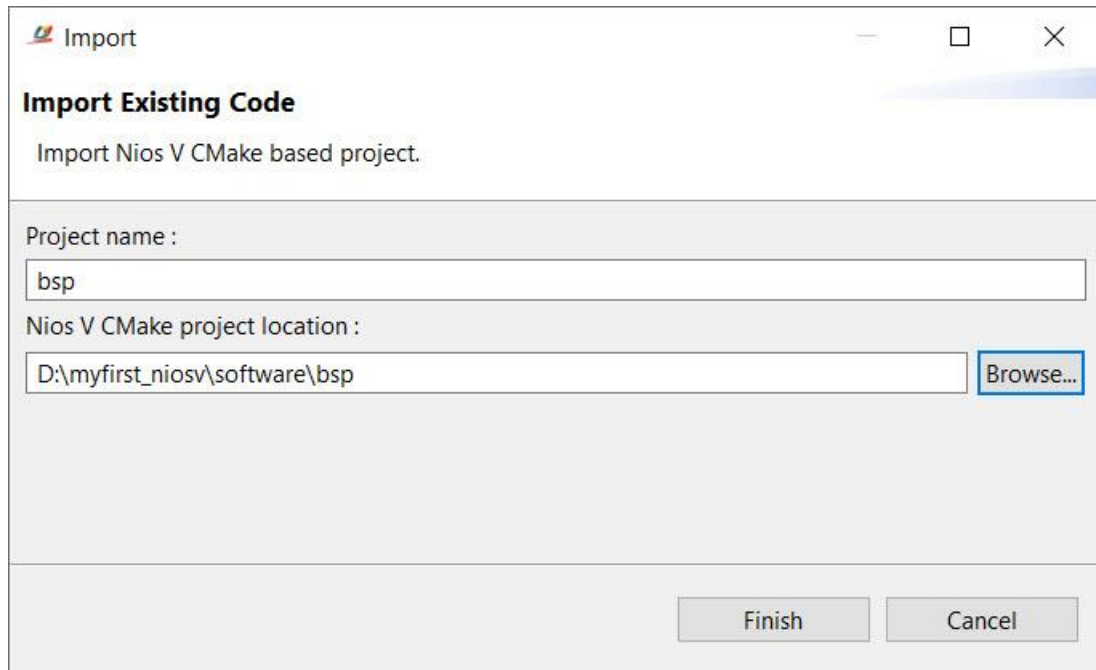


Figure 3-13 Ashling RiscFree IDE Import Project Wizard

6. Click **Finish**. The Ashling RiscFree IDE import the **bsp** project and returns to the Nios V C/C++ project perspective. See Figure 3-14.

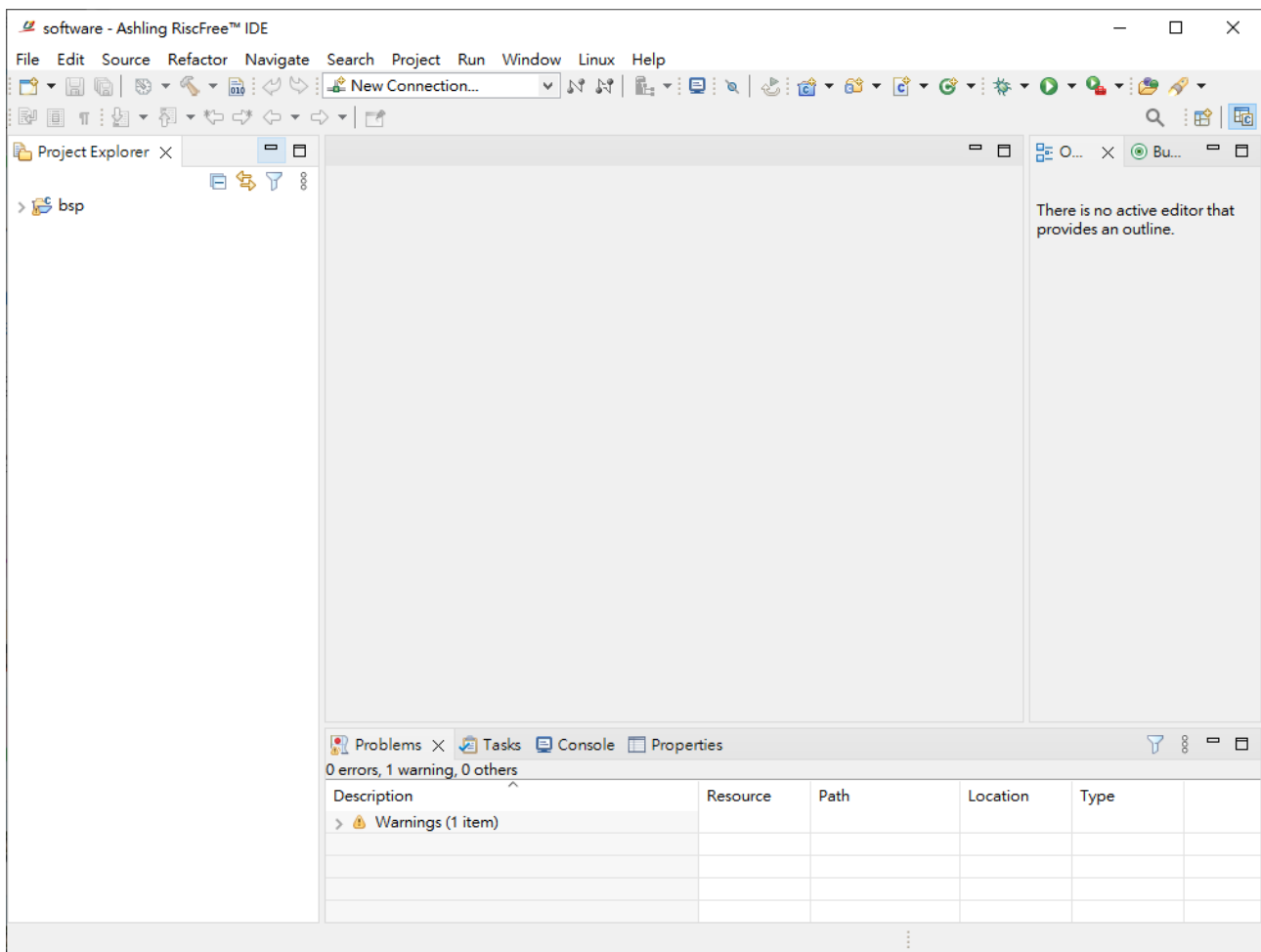


Figure 3-14 Ashling RiscFree IDE C++ Project Perspective for bsp

7. Choose **File->Import Nios V CMake Project** to open the Import Window.

8. In the Import Window, make sure the following things:

- Browse and select the **app** project folder.
- Give the project a name. (**app** is default name)

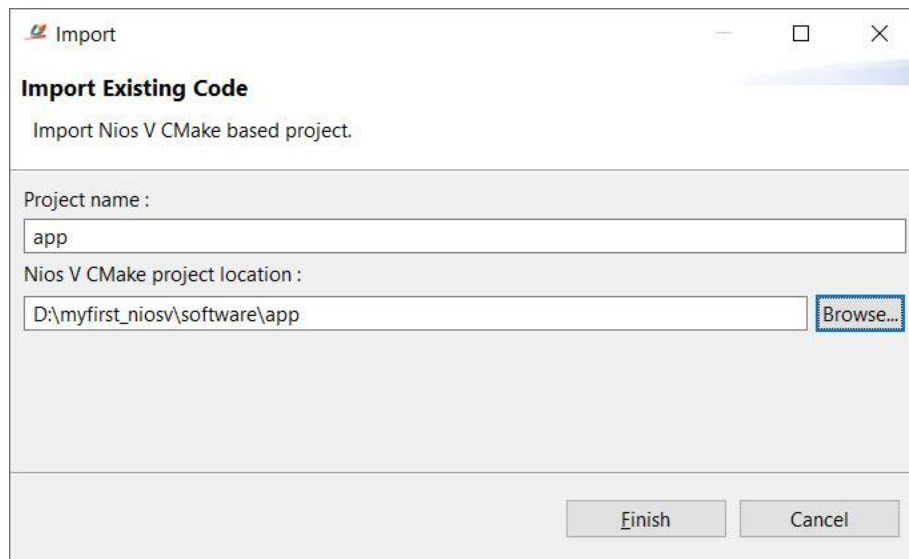


Figure 3-15 Ashling RiscFree IDE Import Project Wizard

9. Click **Finish**. The Ashling RiscFree IDE import the **app** project and returns to the Nios V C/C++ project perspective. See Figure 3-16.

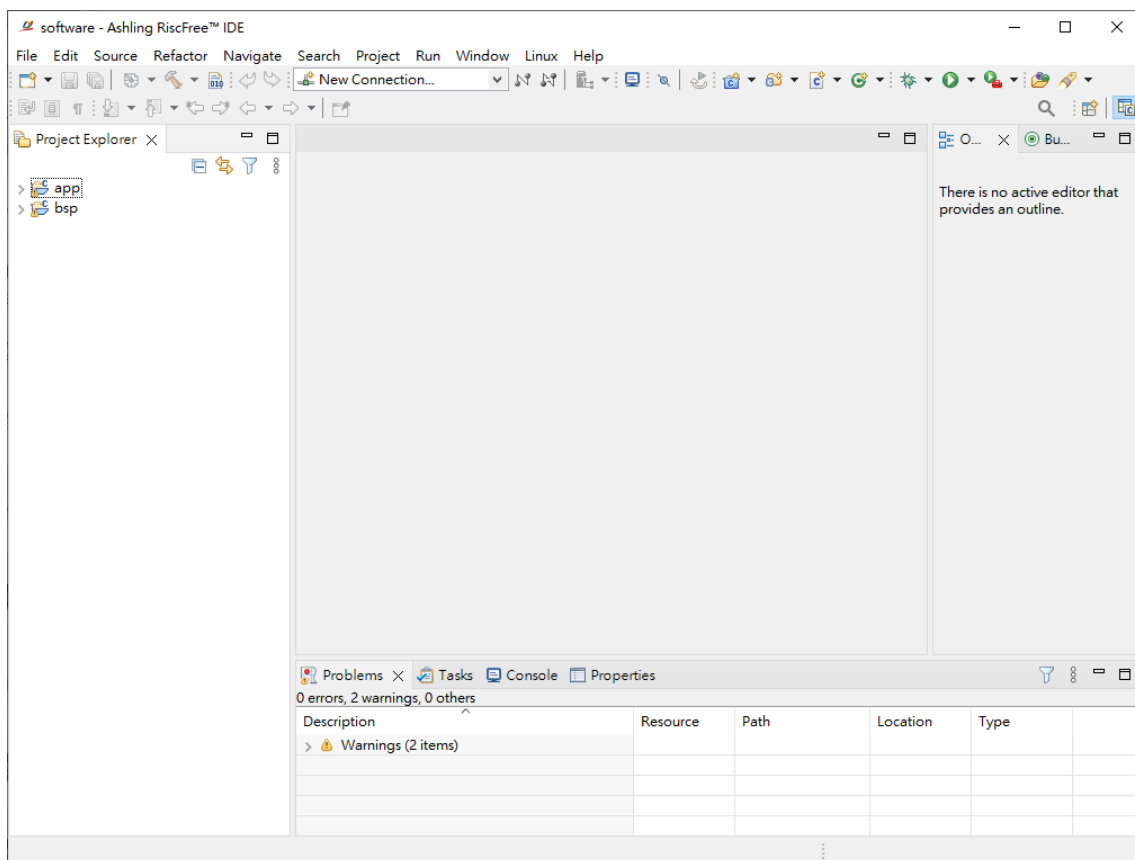


Figure 3-16 Ashling RiscFree IDE C++ Project Perspective for bsp and app

When you import the project, the Ashling RiscFree IDE add two new projects in the Projects Explorer tab:

- **app** is your C/C++ application project. This project contains the source and header files for your application.
- **bsp** is a system library that encapsulates the details of the Nios V system hardware.

3.2 Build and Run the Program

In this section you will build and run the program to execute the compiled code.

1. Before building the project, we would like to confirm with you that the .sof file (hardware design file) in the Quartus project has been downloaded (refer to Section 2.2).
2. To build the program, right-click the **app** project in the Projects Explorer tab and choose **Build Project**. The **Build Project** dialog box appears and the IDE begins compiling the project.

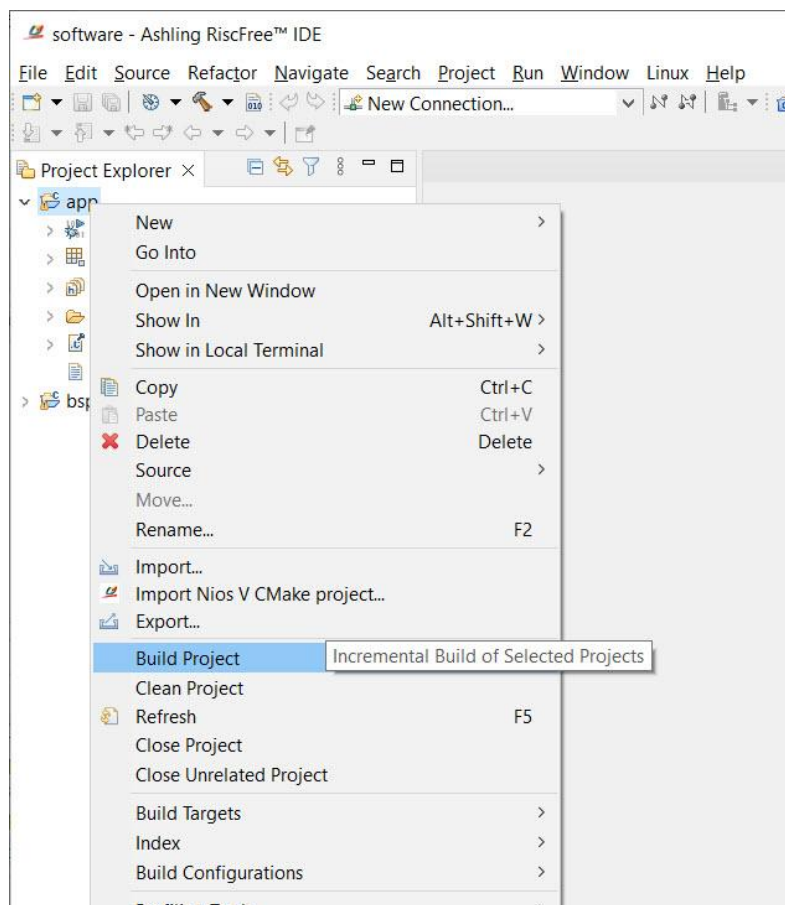


Figure 3-17 Ashling RiscFree IDE C++ Project Perspective for bsp and app

- When the compilation is complete, a **Build Finished** message will appear in the **CDT Build Console**. The compilation time will vary depending on your system. Refer to Figure 3-18 for an example.

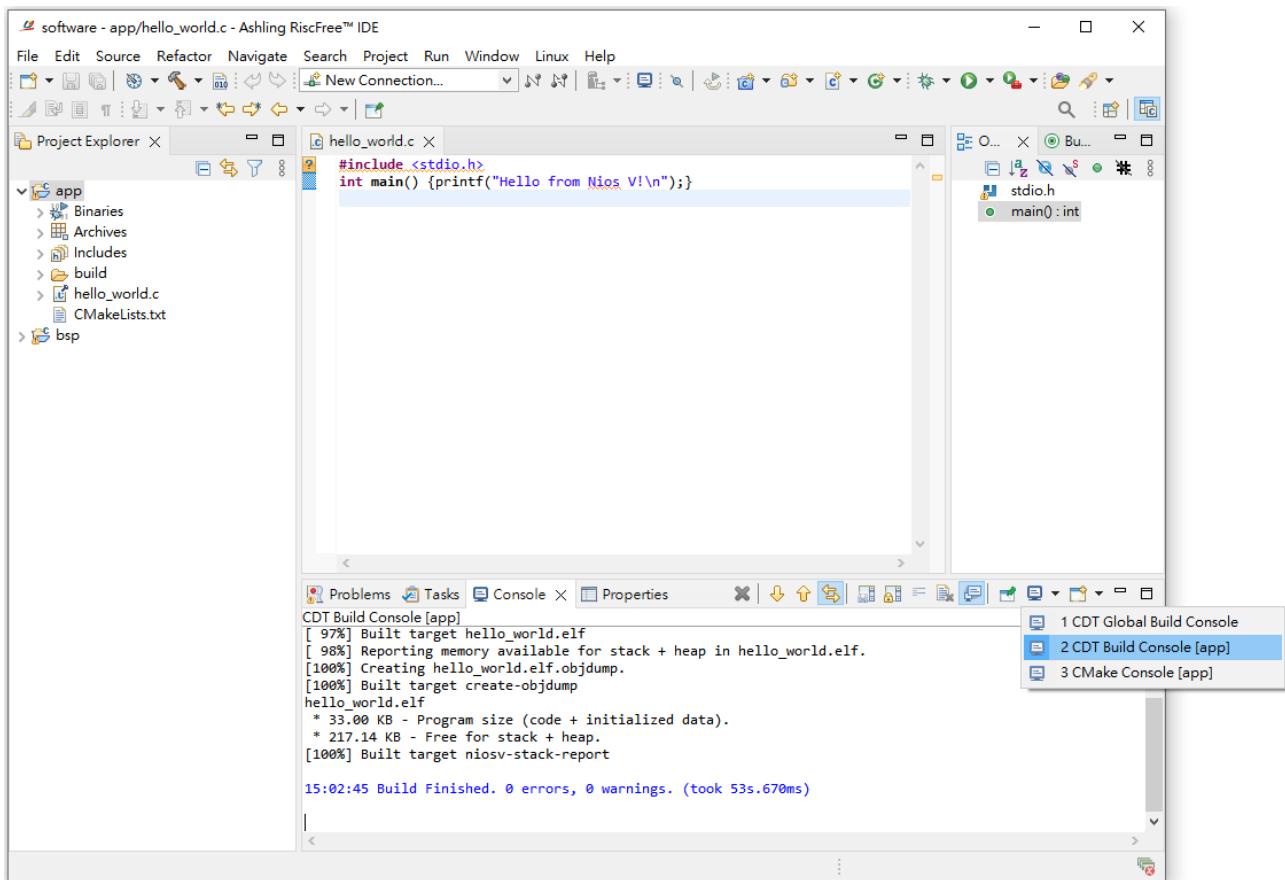


Figure 3-18 Ashling RiscFree IDE app Build Completed

4. After compilation complete, right-click the **app** project, choose **Run As**, and choose **Ashling RISC-V Hardware Debugging**.

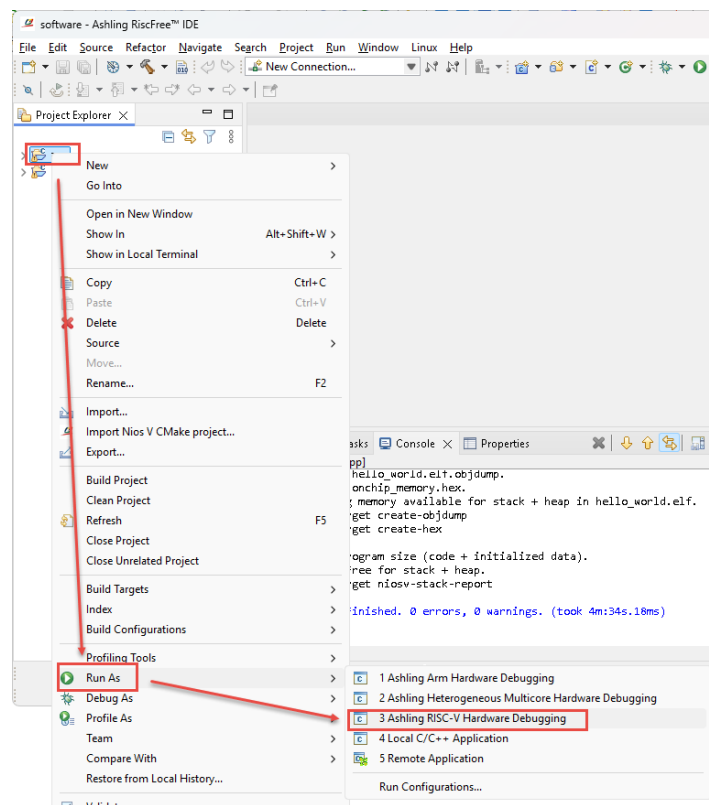


Figure 3-19 Ashling RiscFree IDE Import Project Wizard

5. In the **C Local Application** window, select `hello_world.elf` and then click OK.

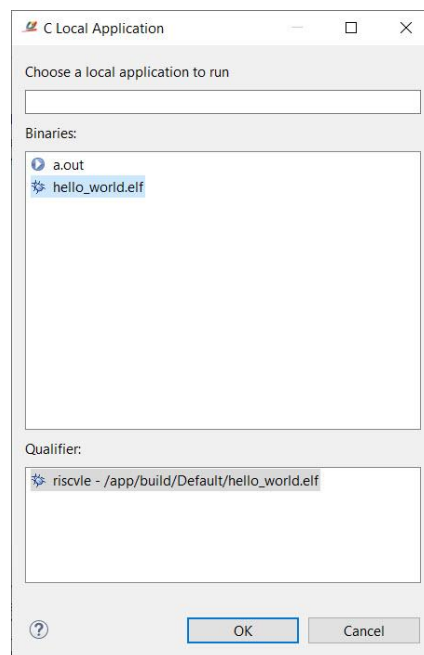


Figure 3-20 Ashling RiscFree IDE Import Project Wizard

6. In the Edit Configuration window:

- Select the **Debugger** tab.
 - For Debug Probe, select **Atum A3 Nano Development board**.
 - Click **Auto-detect Scan Chain**.
 - Click **Run**.

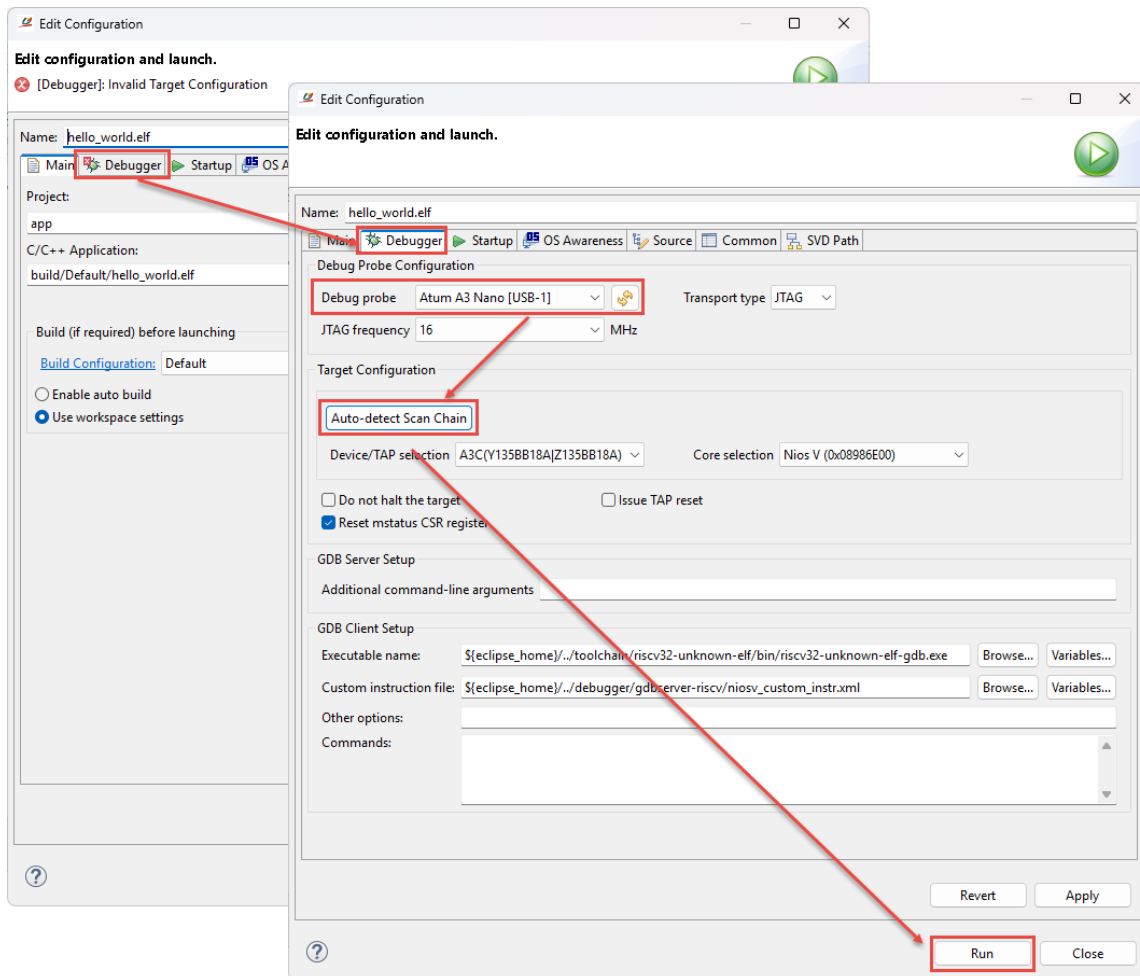


Figure 3-21 Ashling RiscFree IDE Import Project Wizard

The IDE begins to download the program to the target FPGA development board and begins execution.

When the target hardware begins executing the program, a message ‘Got a debugger connection’ will appear in the Console tab (switch to **4. hello_world.elf[Ashling RISC-V Hardware Debuggin]**).

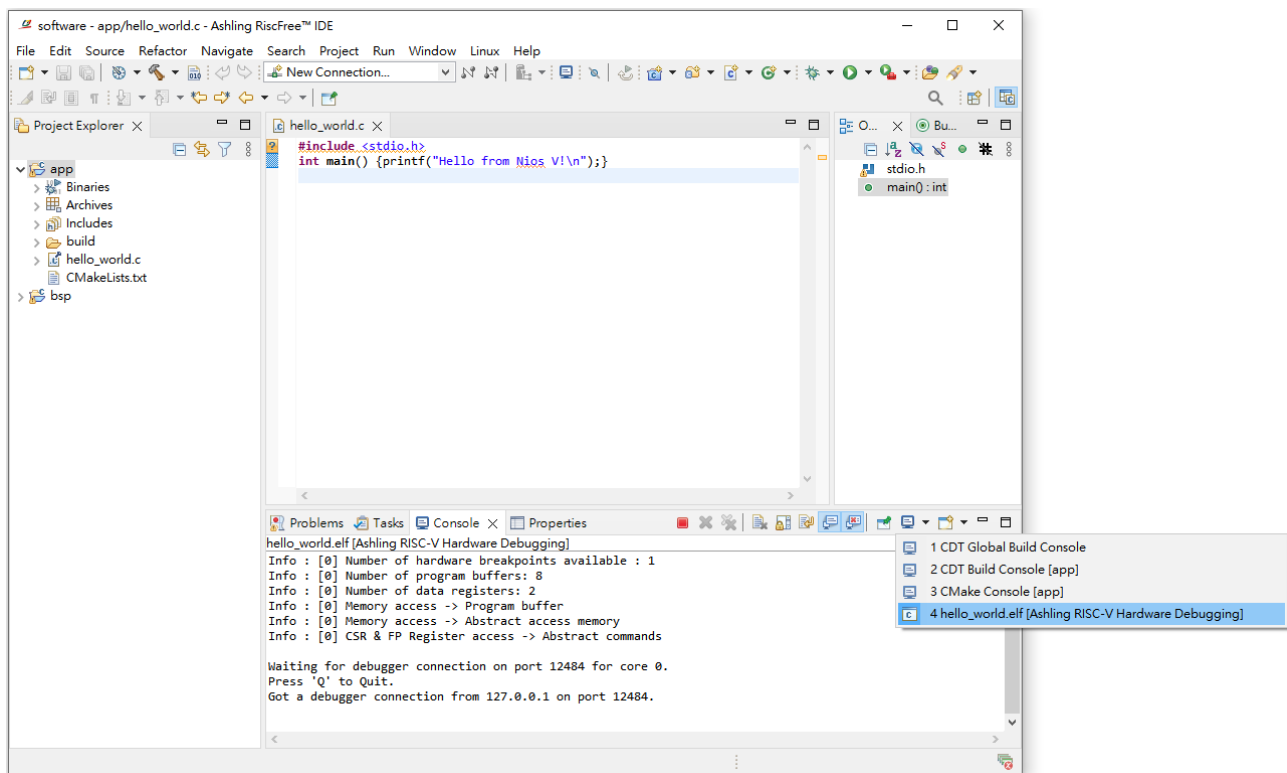


Figure 3-22 Debugger connection message in the Console

You can configure the Ashling RiscFree IDE to integrate external tools. As an example, the following steps detail how to set up the juart-terminal tool to display a "Hello World" message via the JTAG UART IP.

To perform external tool configuration for juart-terminal, follow these steps:

1. Go to **Run > External Tools > External Tools Configurations**.

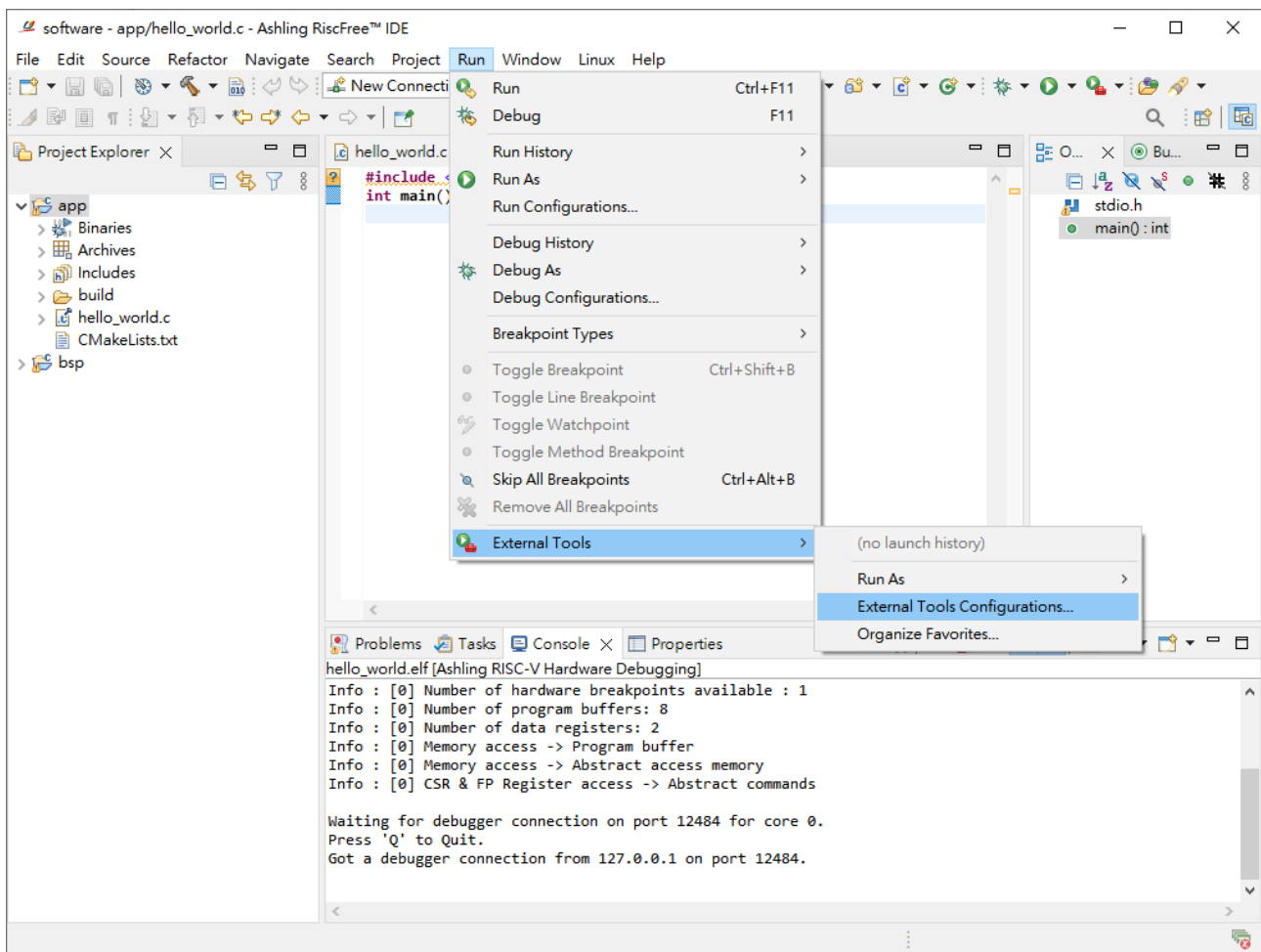


Figure 3-23 Navigating to External Tools Configurations

2. Double click **Program** to open a **New_configuration** window.

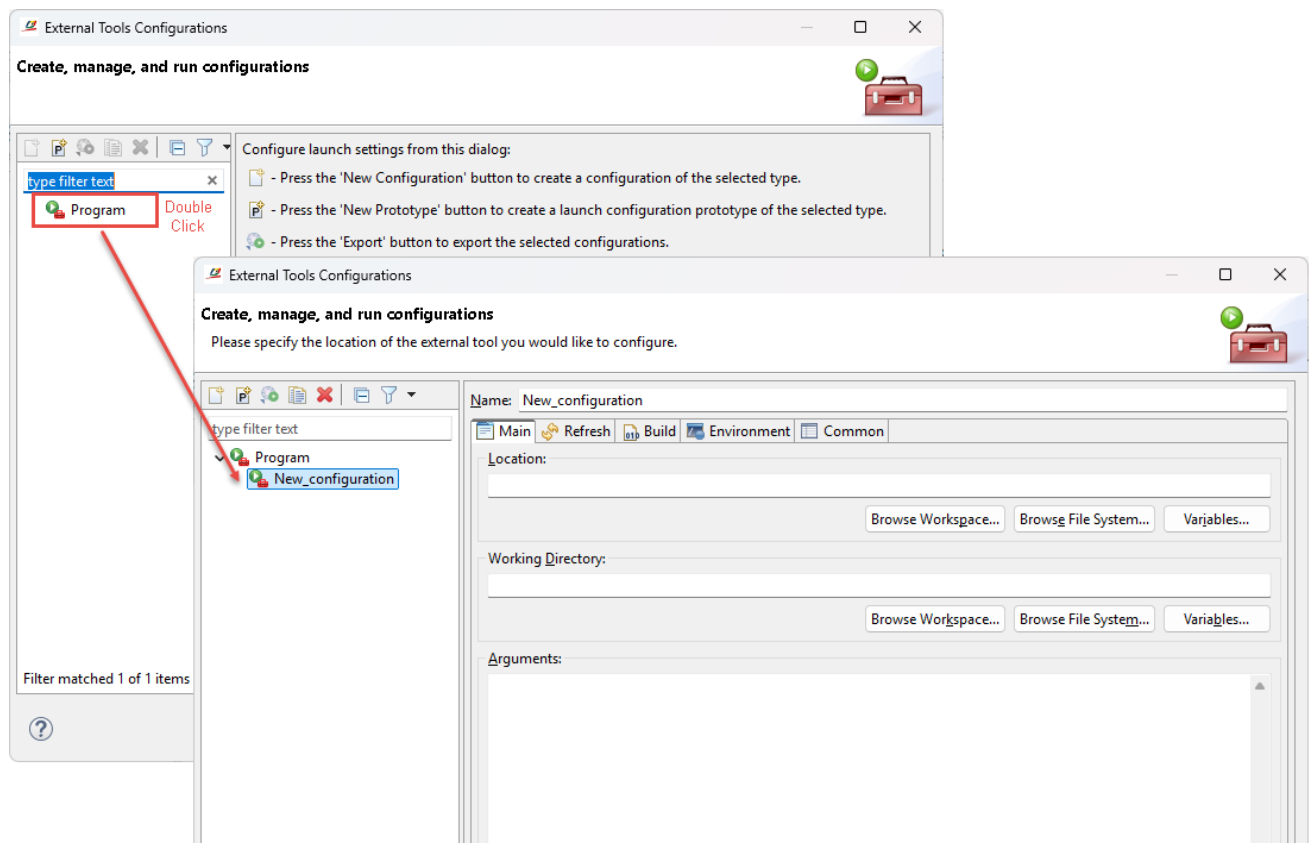


Figure 3-24 Creating a new Program configuration

3. Rename the configuration as **Nios V JTAG UART Output**.

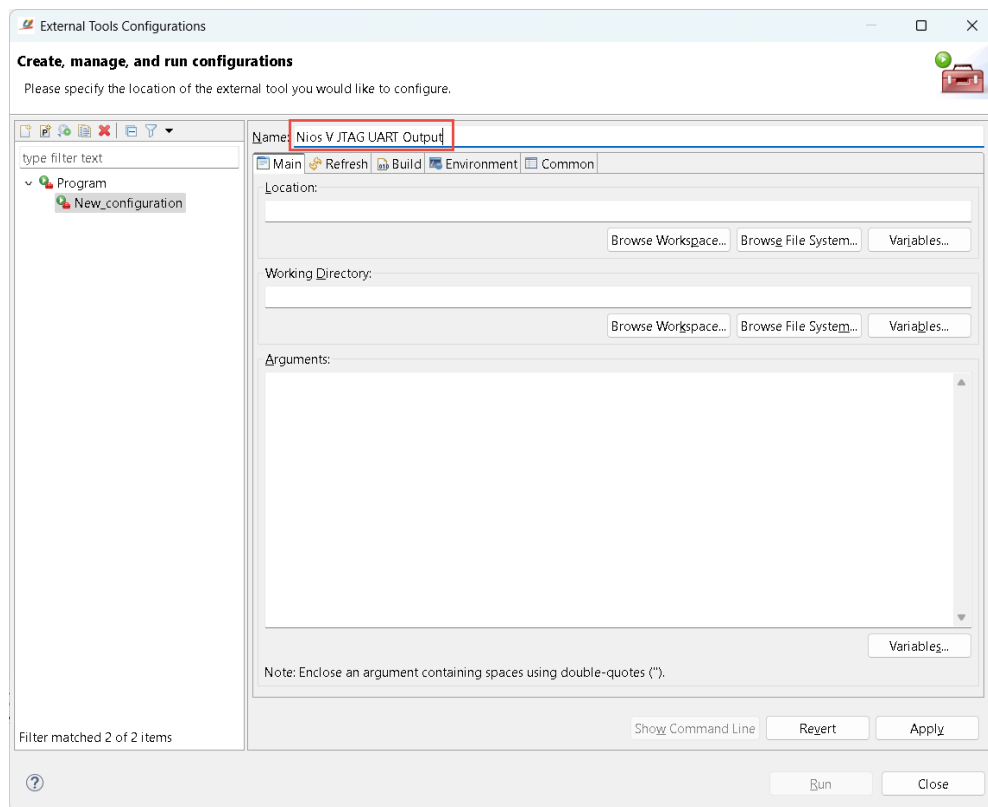


Figure 3-25 Renaming the configuration to 'Nios V JTAG UART Output'

4. In Location, click **Browse File system**.

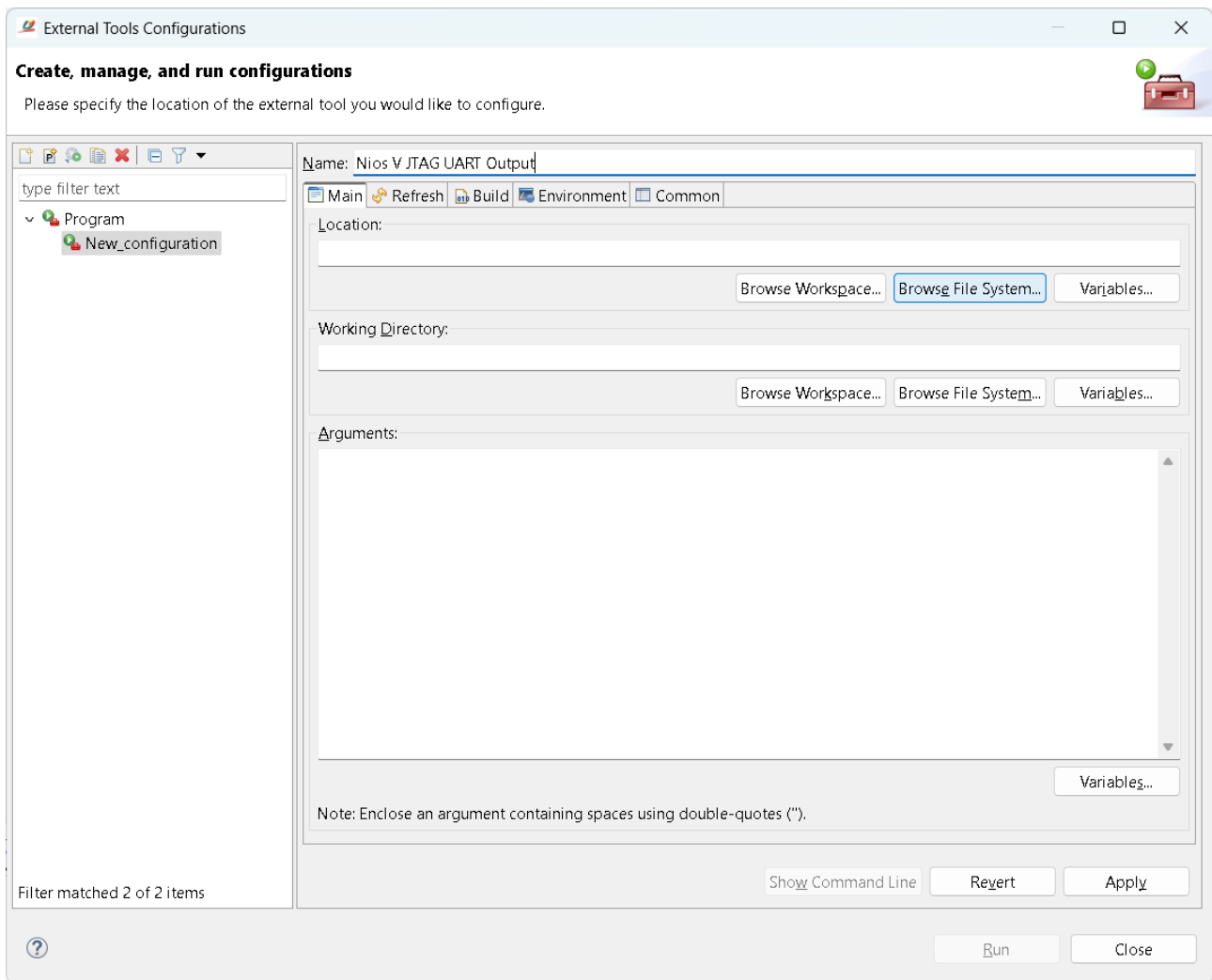


Figure 3-26 Browse for the executable file in the Location field

5. Browse and select the juart-terminal file in the following paths:

<Intel Quartus Prime installation directory>/quartus/bin64/juart-terminal.exe

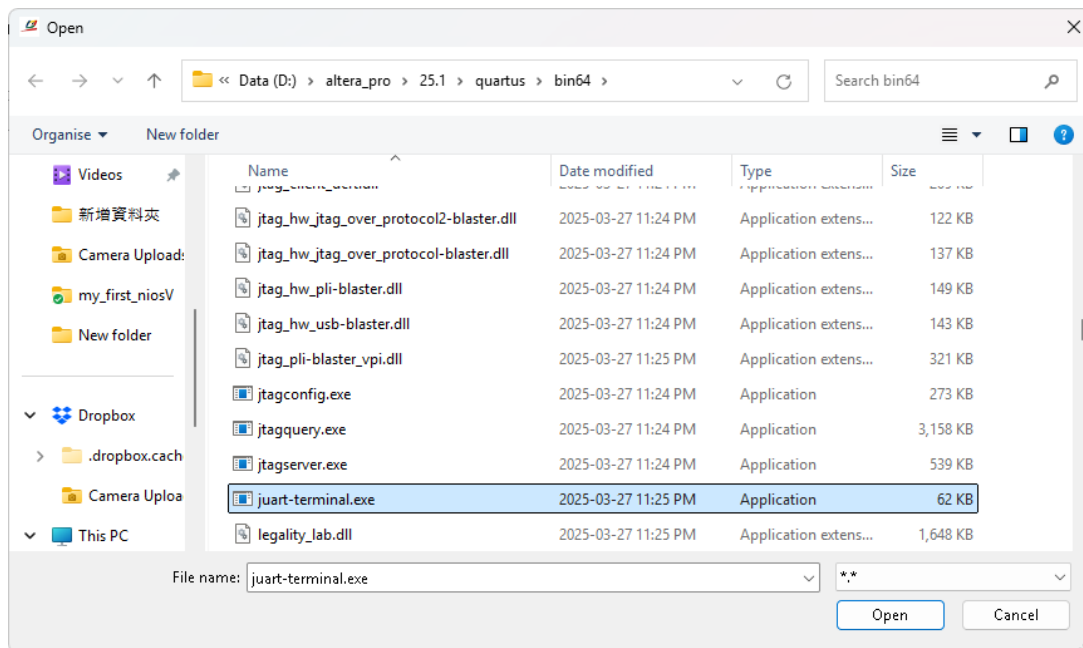


Figure 3-27 Selecting 'juart-terminal.exe'

- Set the **Arguments** as “-c 1 -d 1 -i 0”. This configures the JTAG UART connection is towards the JTAG UART IP at cable 1, device 1, and instance 0.

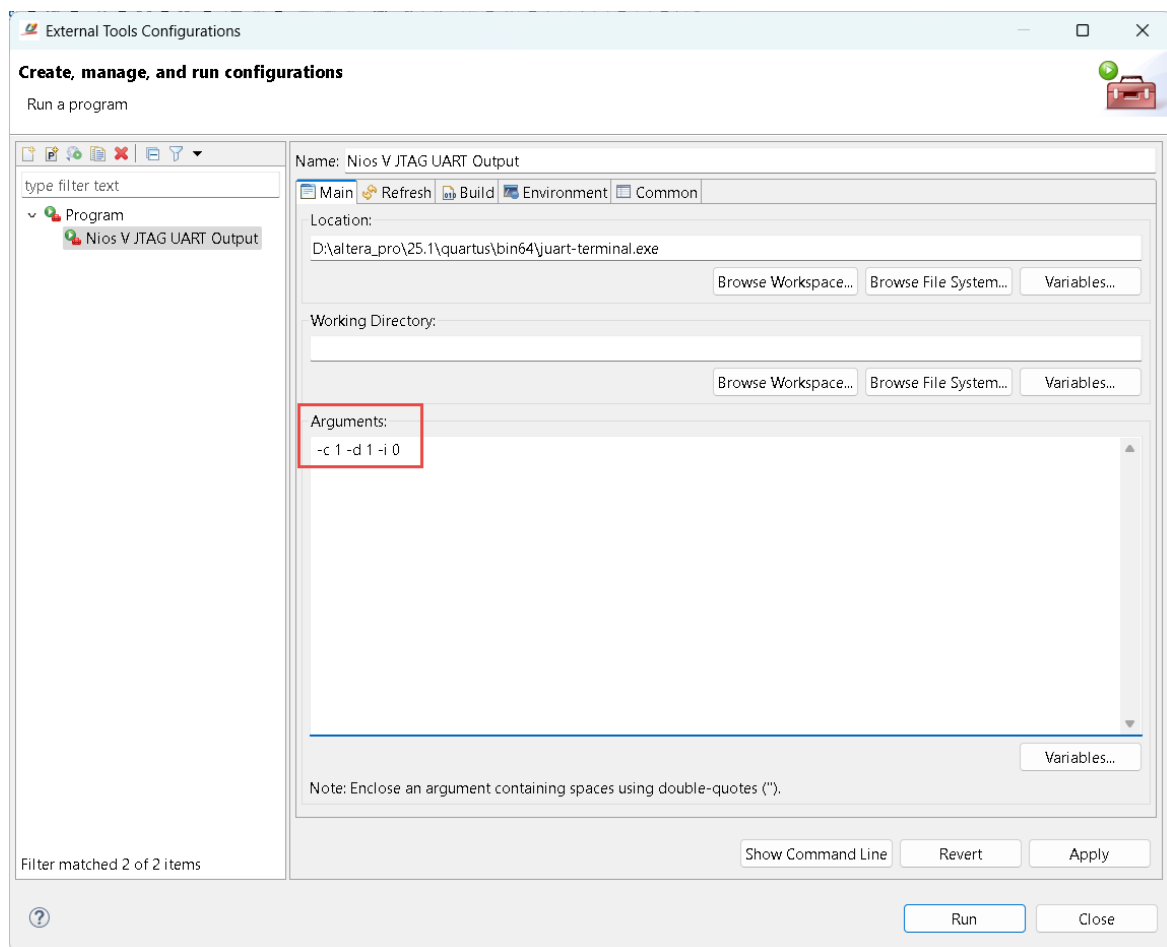


Figure 3-28 Setting arguments for the JTAG UART

7. Click **Apply**, then click **Run**

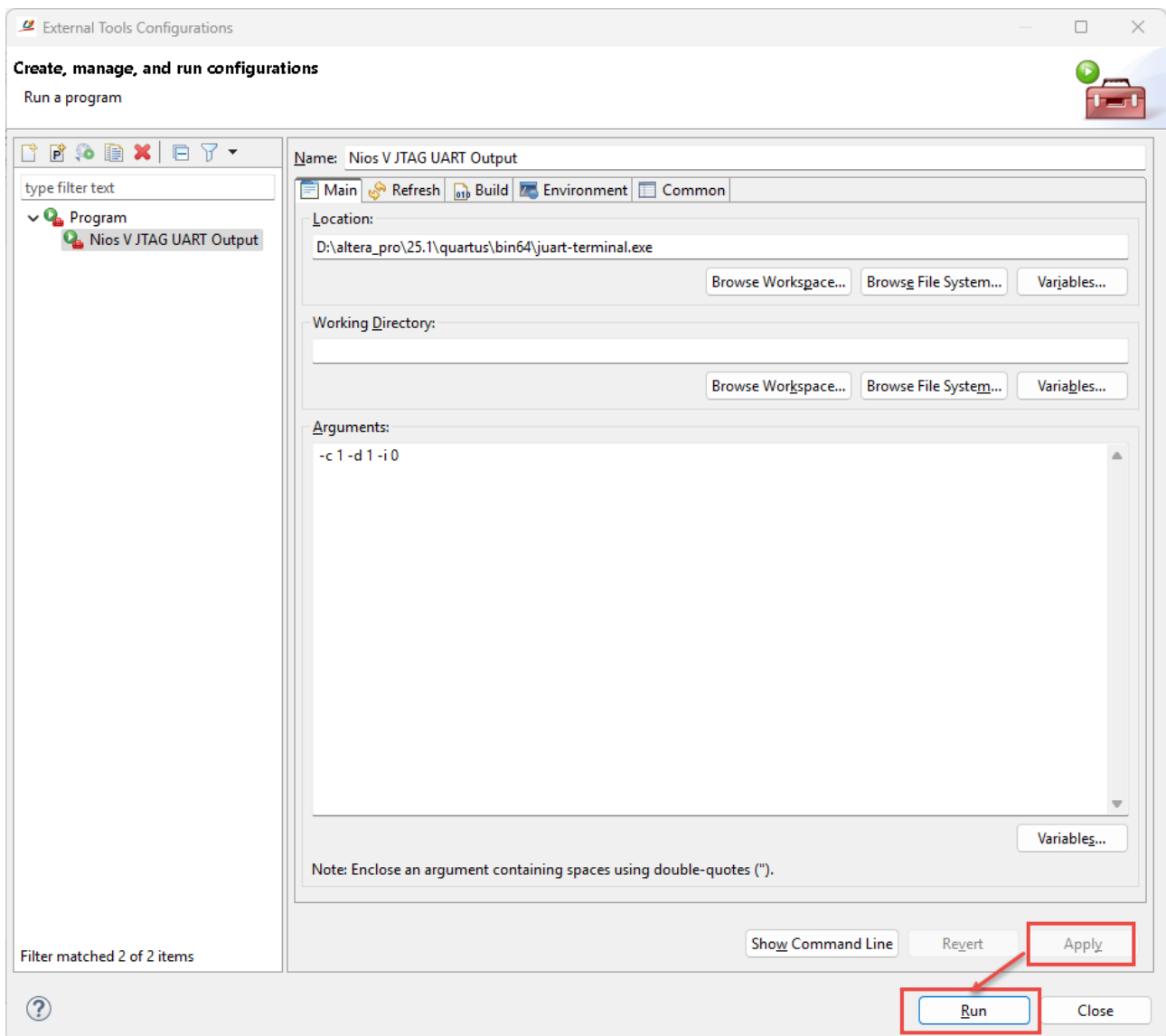


Figure 3-29 Applying and running the external tool

8. The message 'Hello from Nios V!' appears in the **Nios V JTAG UART Output** tab.

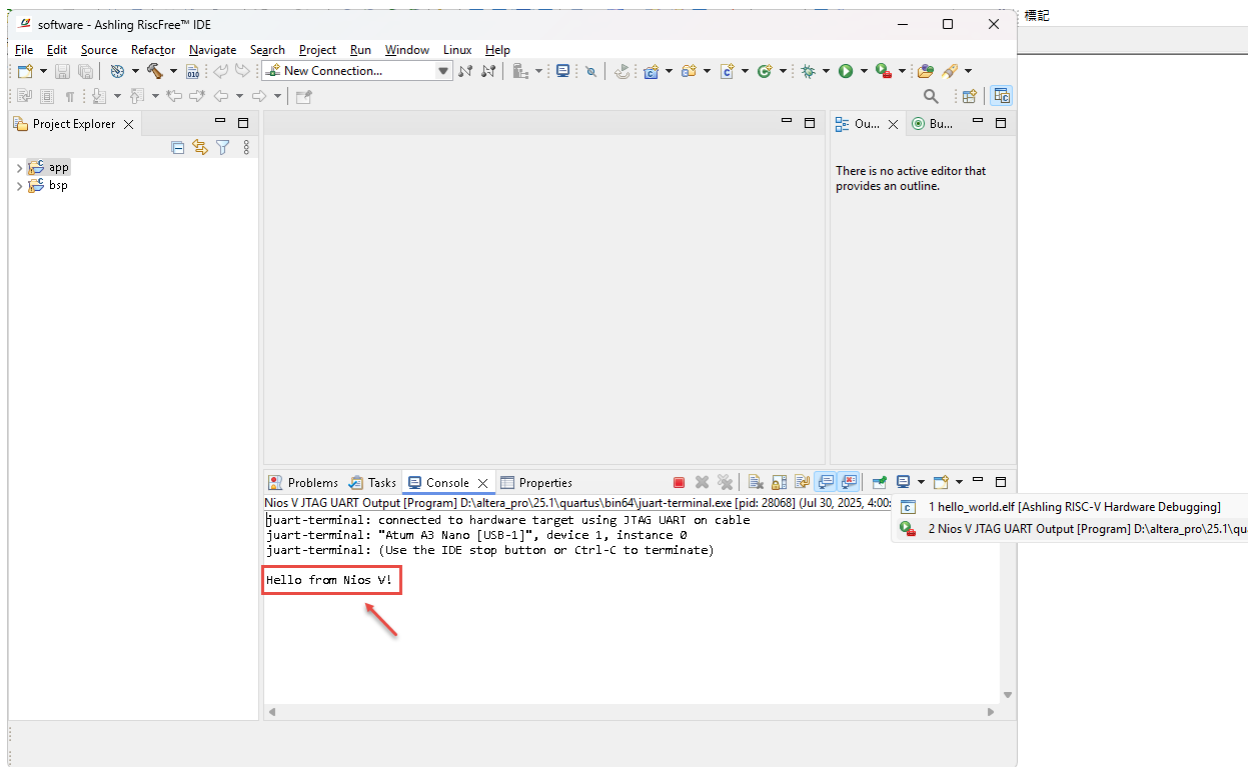


Figure 3-30 "Hello from Nios V!" message in the output tab.

9. Before proceeding to the next step, the user needs to switch to the `hello_world.elf` console and stop `Terminate` . This will ensure that the next step can be executed without any issues.

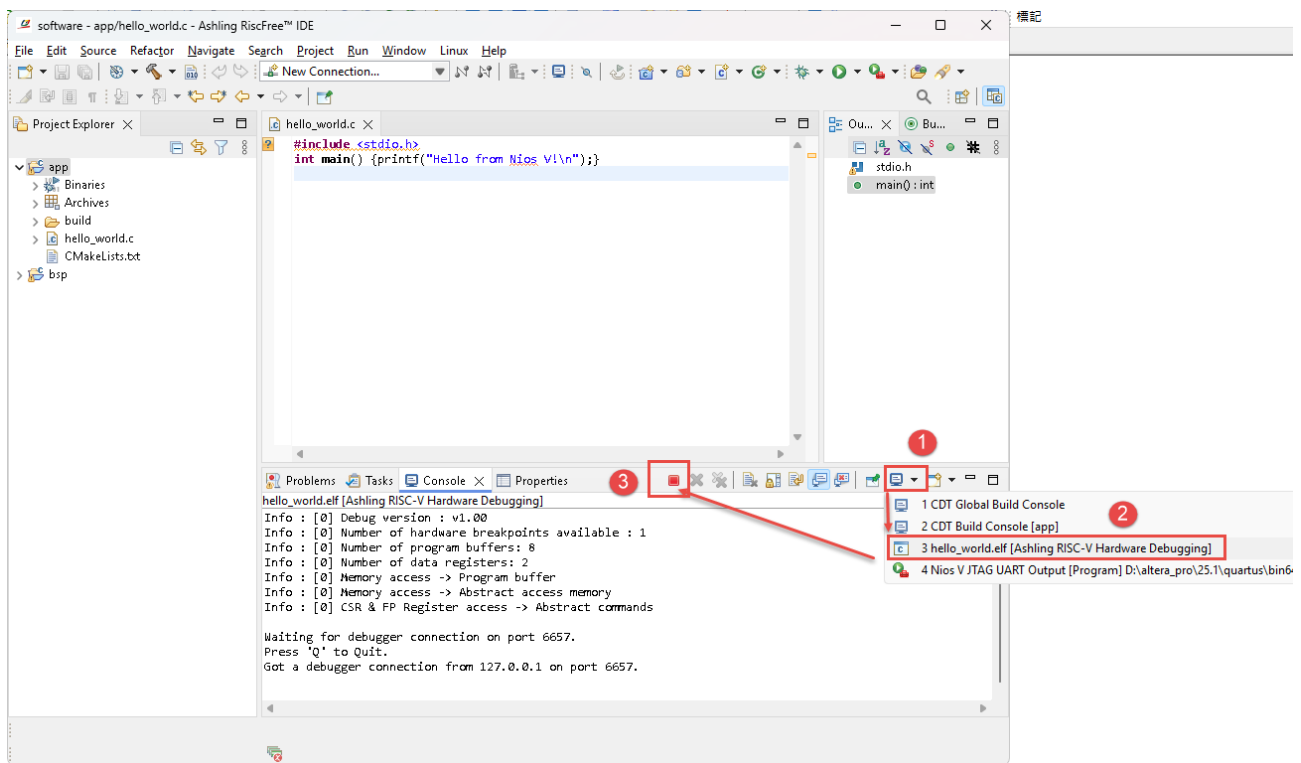


Figure 3-31 Stop Terminate.

Now you have created, compiled, and run your first software program based on NIOS V. And you can perform additional operations such as configuring the system properties, editing and re-building the application, and debugging the source code.

3.3 Edit and Re-Run the Program

You can modify the **hello_world.c** program file in the IDE, build it, and re-run the program to observe your changes executing on the target board. In this section you will add code that will make LEDG blink.

Perform the following steps to modify and re-run the program:

1. In the hello_world.c file, modify the text shown in the example below:

```
#include <stdio.h>
#include <system.h>
#include <altera_avalon_pio_regs.h>

int main()
{
    printf("Hello from Nios V!\n");
    int count = 0;
    int delay;
    while (1) {
        IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED_BASE, count & 0x01);
        delay = 0;
        while (delay < 2000000) {
            delay++;
        }
        count++;
    }
    return 0;
}
```

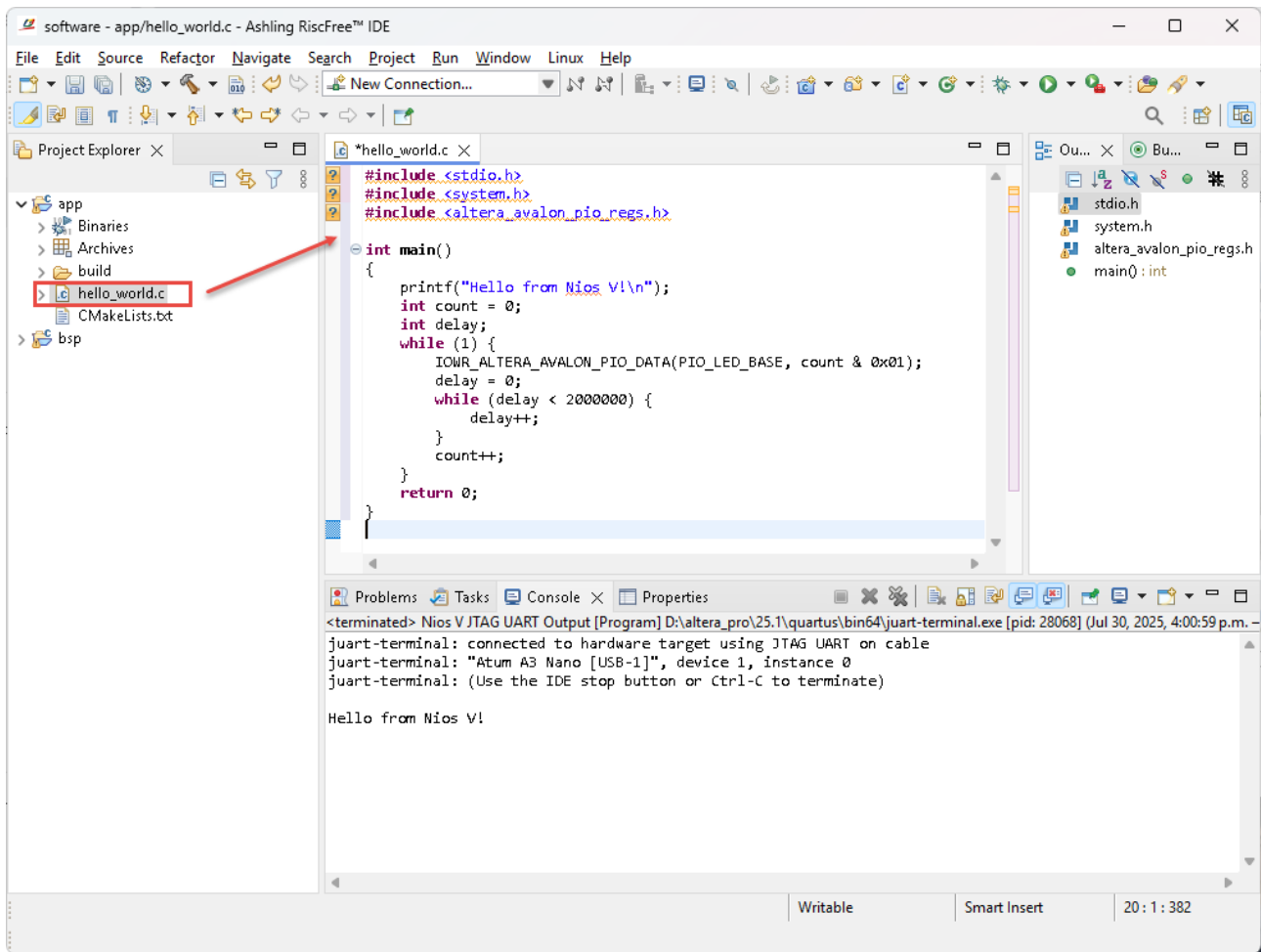


Figure 3-32 Adding LED blink code to 'hello_world.c'

2. Save the project.
3. Recompile the file by right-clicking **app** in the Projects Explorer tab and choosing **Build Project**

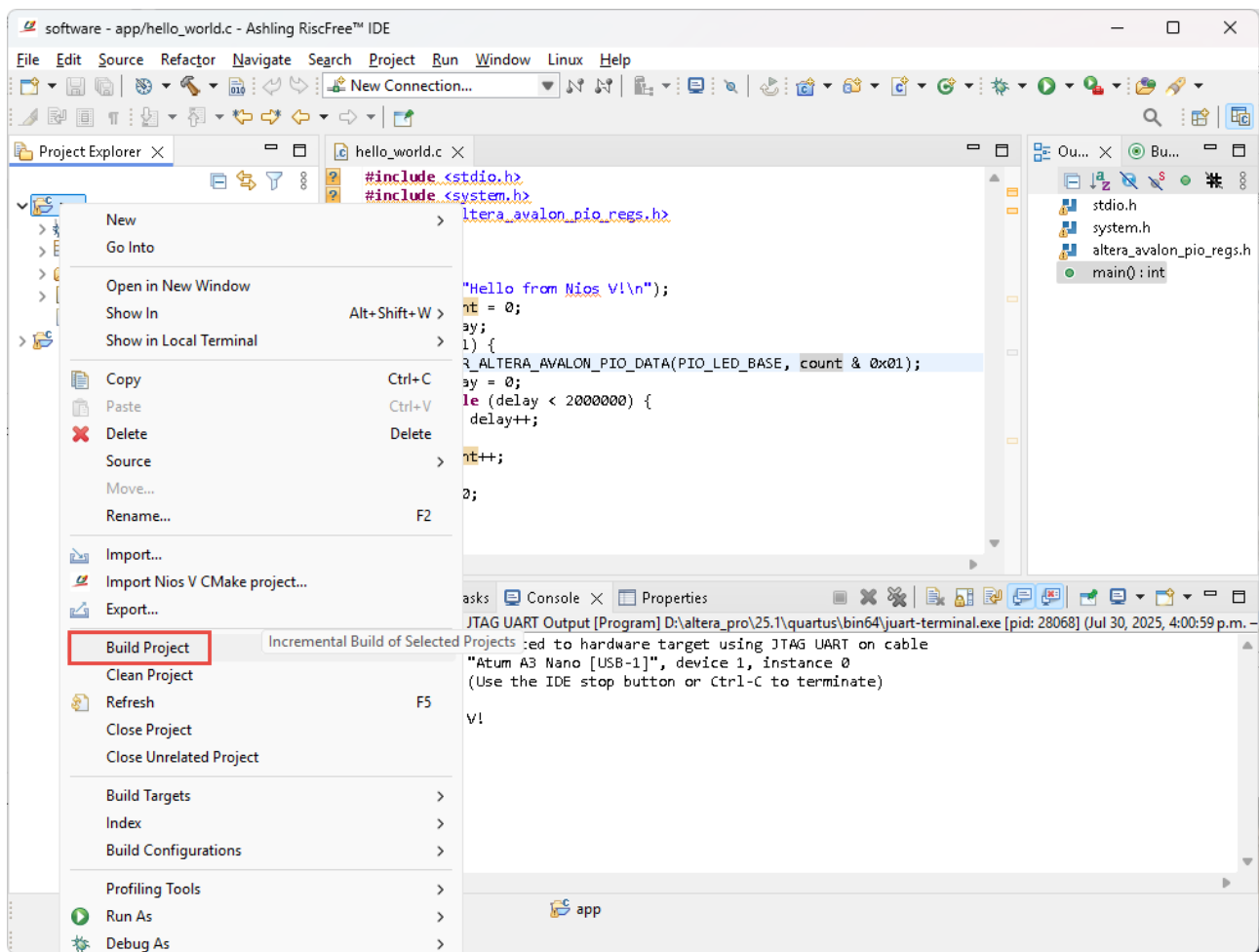


Figure 3-33 Rebuilding the project after code changes

4. After **Build Project** done, **Run > Run As > Ashling RISC-V Hardware Debugging**.

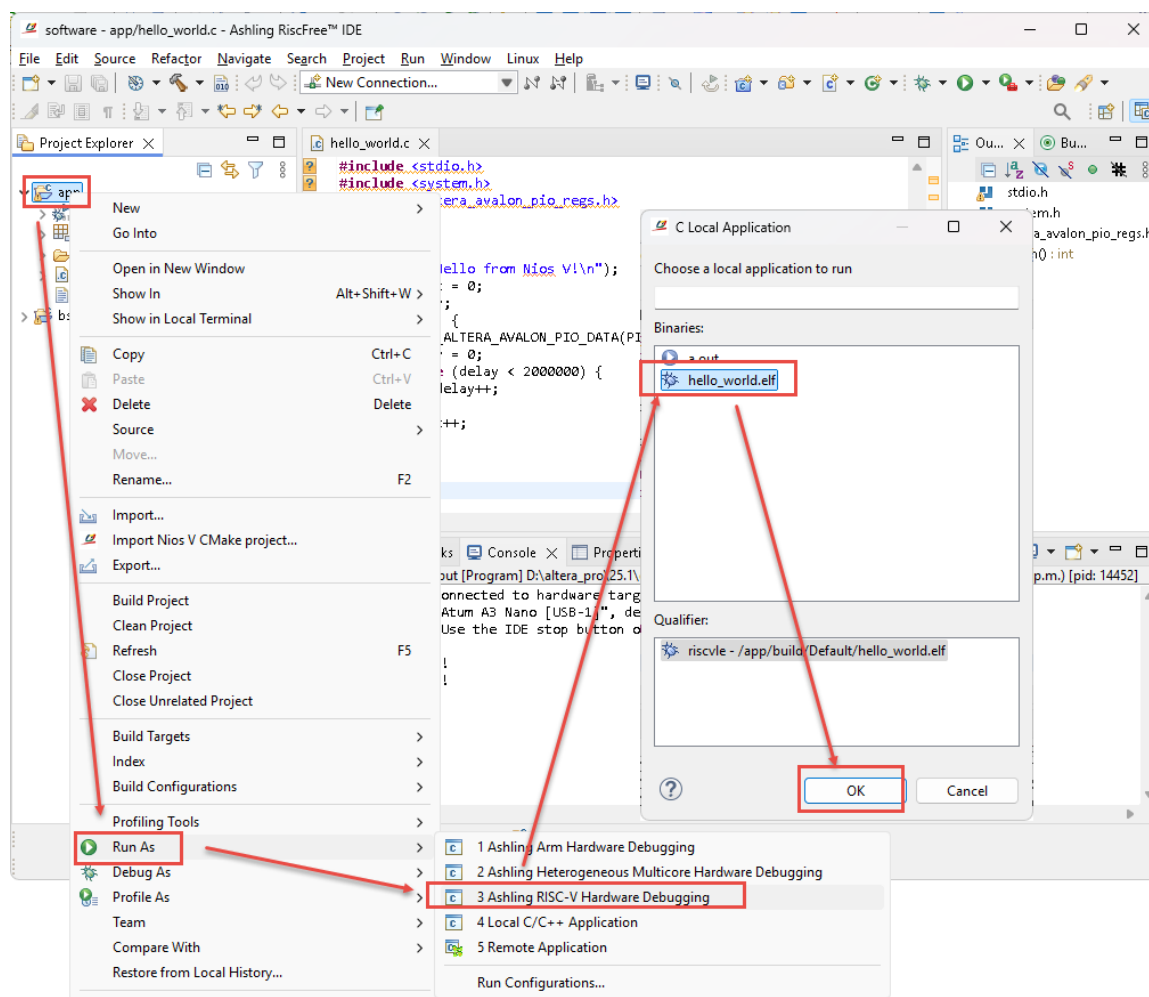


Figure 3-34 Run program

5. Orient your development board so that you can observe LEDG blinking.



Figure 3-35 LEDG

3.4 Why the LED Blinks

The Nios V system description header file, **system.h**, contains the software definitions, name, locations, base addresses, and settings for all of the components in the Nios V hardware system. The **system.h** file is located in the **bsp/system.h** directory as shown in Figure 3-36.

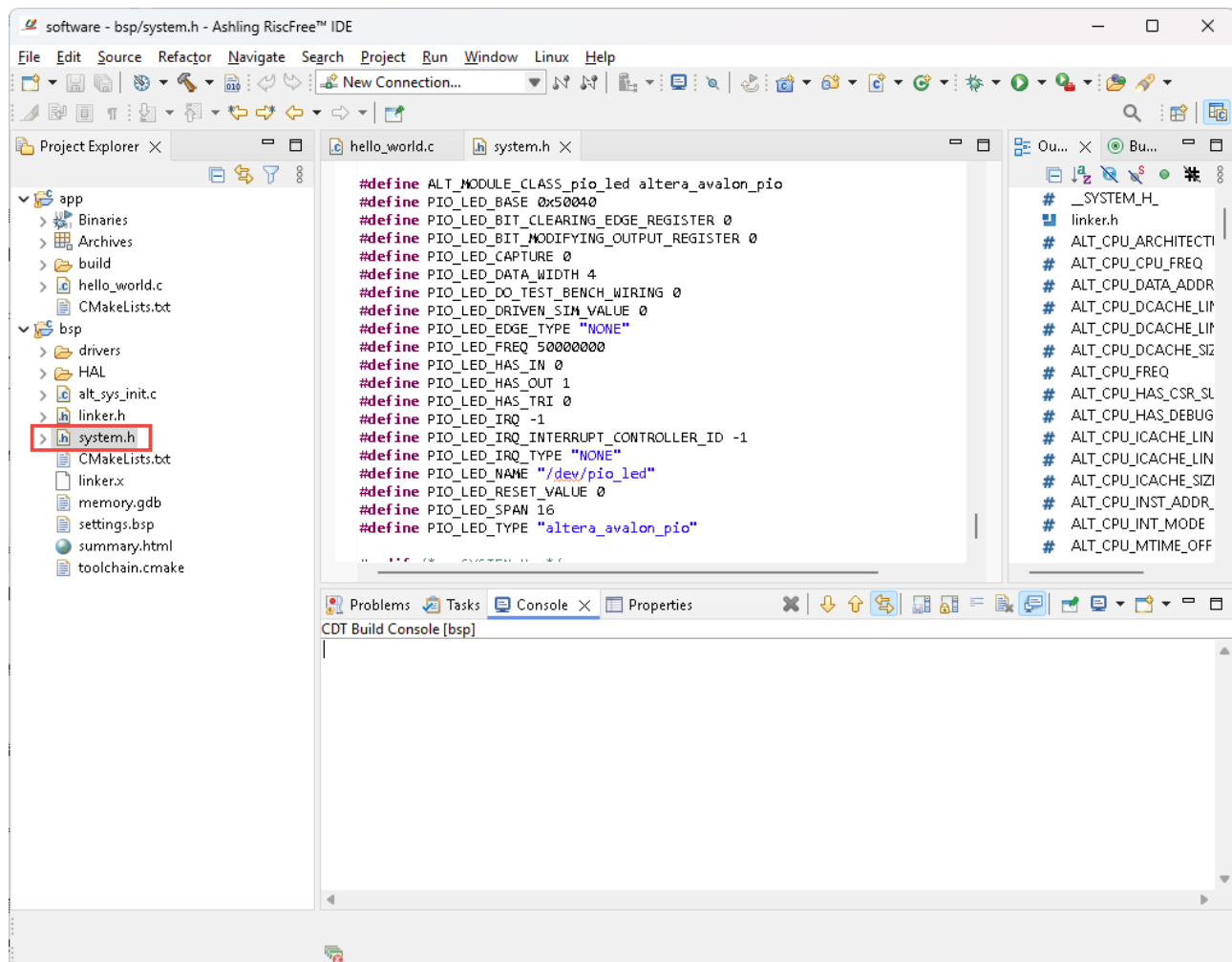


Figure 3-36 system.h Location

If you look at the **system.h** file for the Nios V project example used in this tutorial, you will notice the **pio_led** function. This function controls the LED. The Nios V processor controls the PIO ports (and thereby the LED) by reading and writing to the register map. For the PIO, there are four registers: **data**, **direction**, **interrupt mask**, and **edge capture**. To turn the LED on and off, the application writes to the PIO data register.

The PIO core has an associated software file **altera_avalon_pio_regs.h**. This file defines the core's register map, providing symbolic constants to access the low-level hardware.

The **altera_avalon_pio_regs.h** file is located in **bsp\drivers\inc**.

When you include the **altera_avalon_pio_regs.h** file, several useful functions that manipulate the PIO core registers are available to your program. In particular, the function

IOWR_ALTERA_AVALON_PIO_DATA (base, data)

can write to the PIO data register, turning the LED on and off. The PIO is just one of many Platform Designer peripherals that you can use in a system. To learn about the PIO core and other embedded peripheral cores, refer to Embedded Peripherals IP User Guide.

When developing your own designs, you can use the software functions and resources that are provided with the Nios V HAL. Refer to the Nios V Processor Software Developer Handbook for extensive documentation on developing your own Nios V processor-based software applications.

3.5 Debugging the Application

Before you can debug a project in the Ashling RiscFree IDE, you need to create a debug configuration that specifies how to run the software. To set up a debug configuration, perform the following steps:

1. In the **hello_world.c**, double-click the front of the line which is needed to set breakpoint. See Figure 3-37.

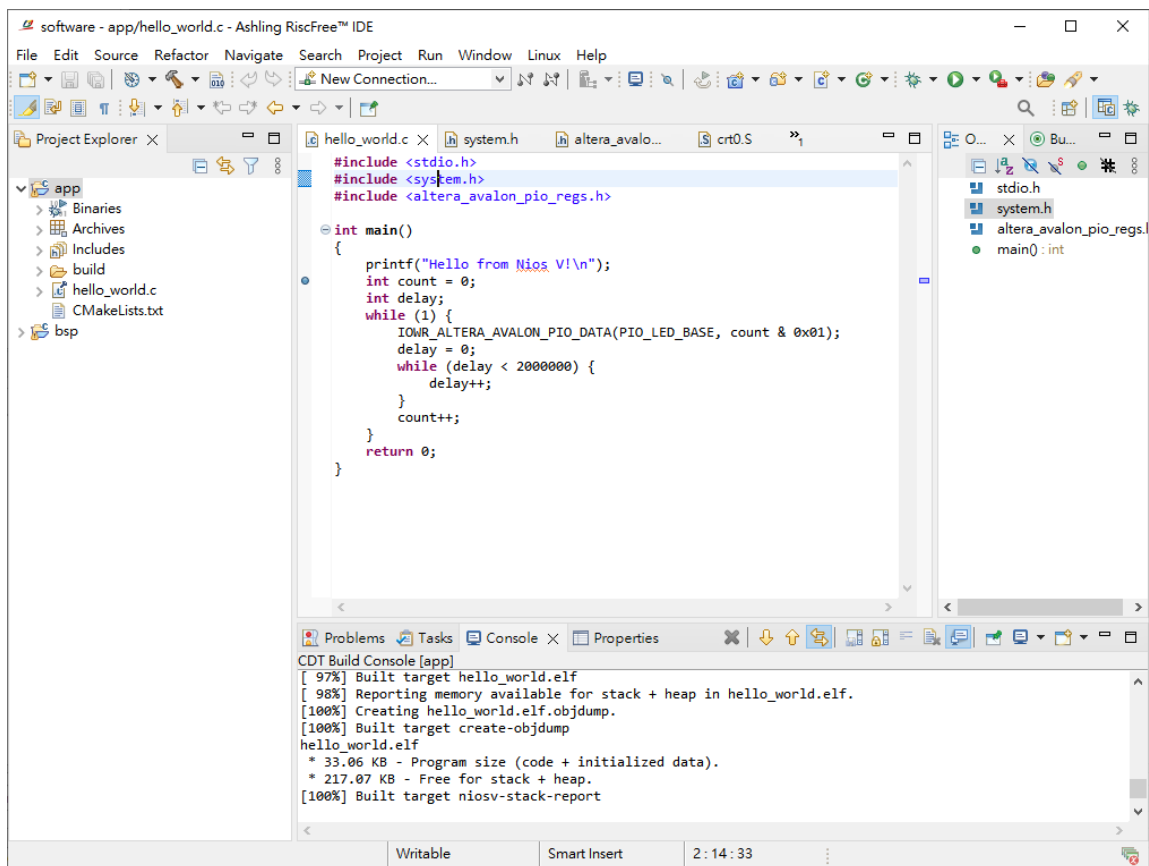


Figure 3-37 Set Breakpoint

2. To debug your application, right-click the application (**app** by default) and choose **Debug as > Ashling RISC-V Hardware Debugging**.
3. If the **Confirm Perspective Switch** message box appears, click **Switch**.

After a moment, the `main()` function appears in the editor. A blue arrow next to the first line of code indicates that execution stopped at that line.

4. Choose **Run-> Resume** to resume execution.

When debugging a project in the Ashling RiscFree IDE, you can pause, stop or single step the program, set breakpoints, examine variables, and perform many other common debugging tasks.

Note: To return to the C/C++ project perspective from the debug perspective, click the C/C++ icon in the top right corner of the GUI.

3.6 Configure System Library

In this section you will learn how to configure some advanced options about the target memory or other things. By performing the following steps, you can change all the available settings:

1. In Quartus Prime, select File > Open, and open system.qsys, as shown in the figure below.

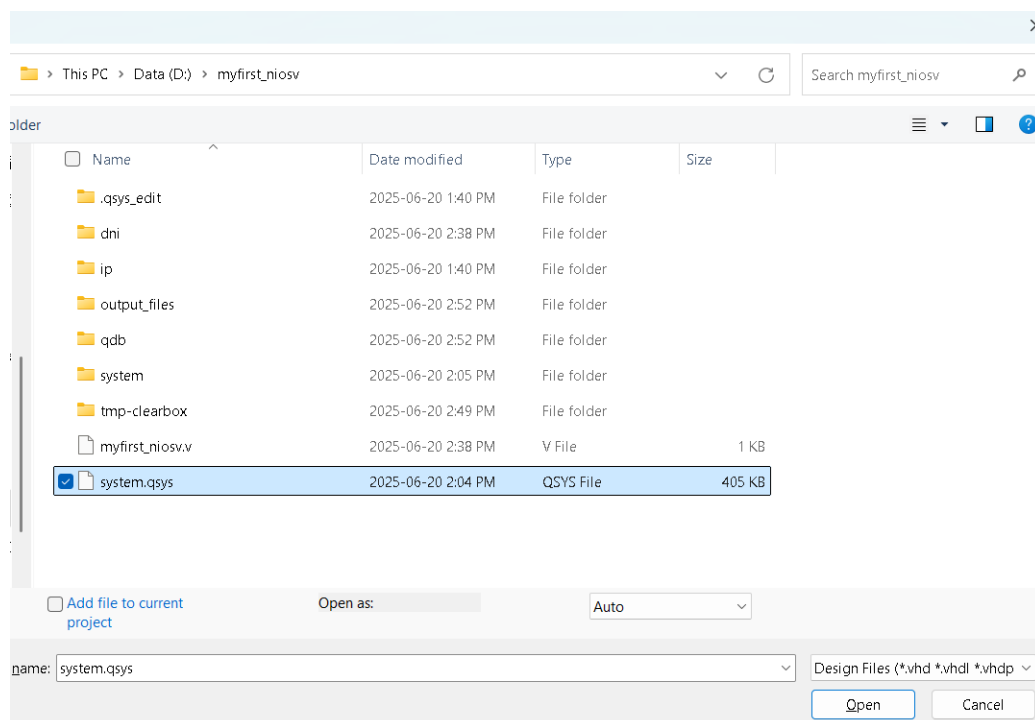


Figure 3-38 Opening 'system.qsys' in Quartus Prime

2. Click File > Open to open the BSP, as shown in the figure below.

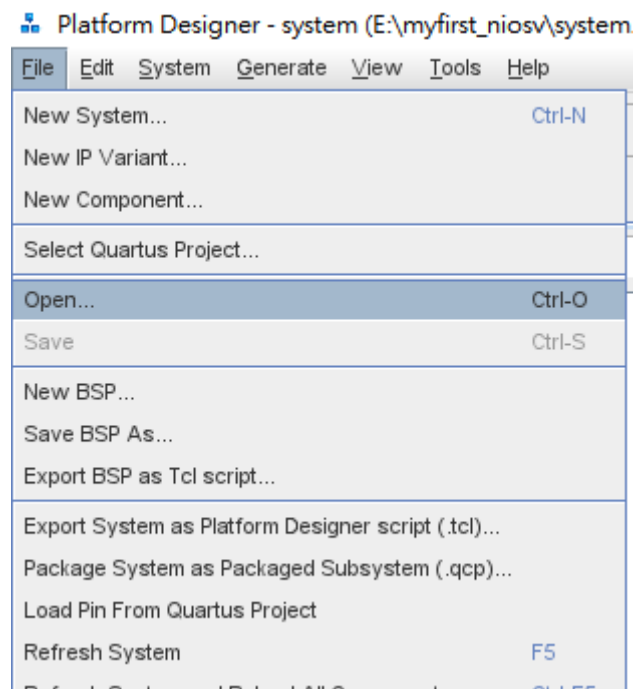


Figure 3-39 Opening the BSP file

3. Select the **BSP Editor** tab.

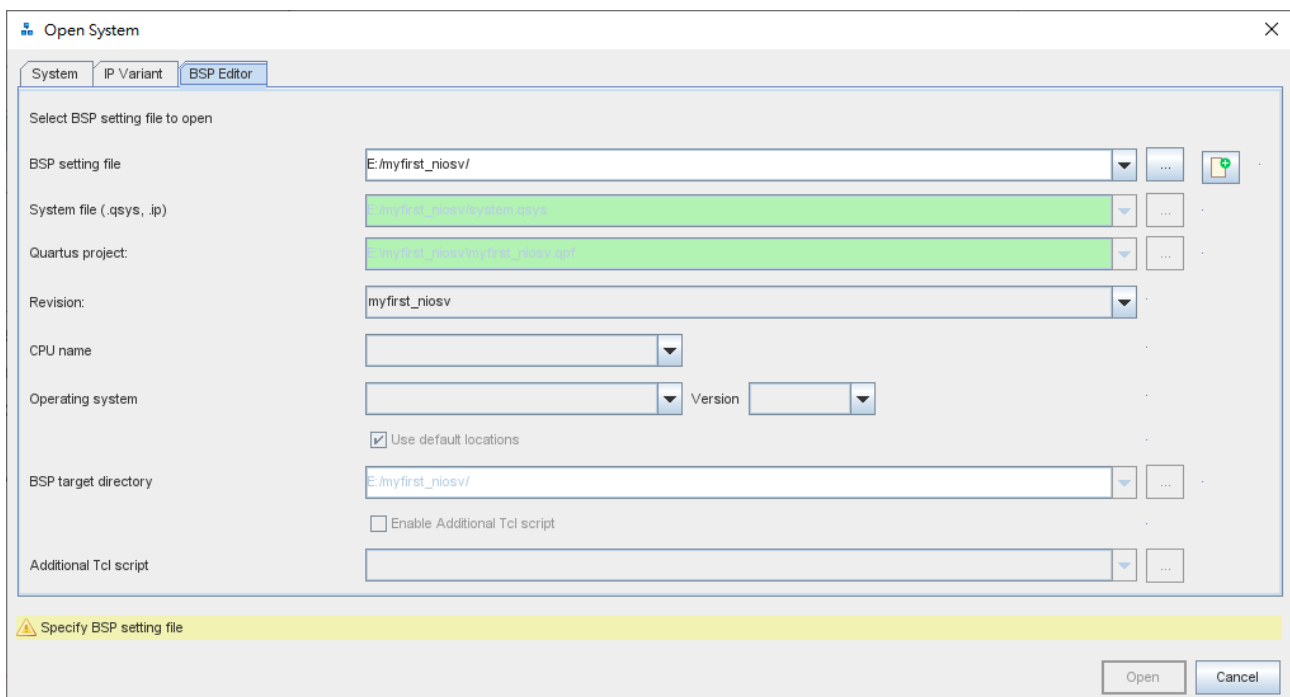


Figure 3-40 Selecting the 'BSP Editor' tab

4. Browse to and open software/bsp/settings.bsp.
5. Click Open to open the BSP Editor.

6. Double-click the BSP Editor Tab to maximize the tab, as shown in the figure below.

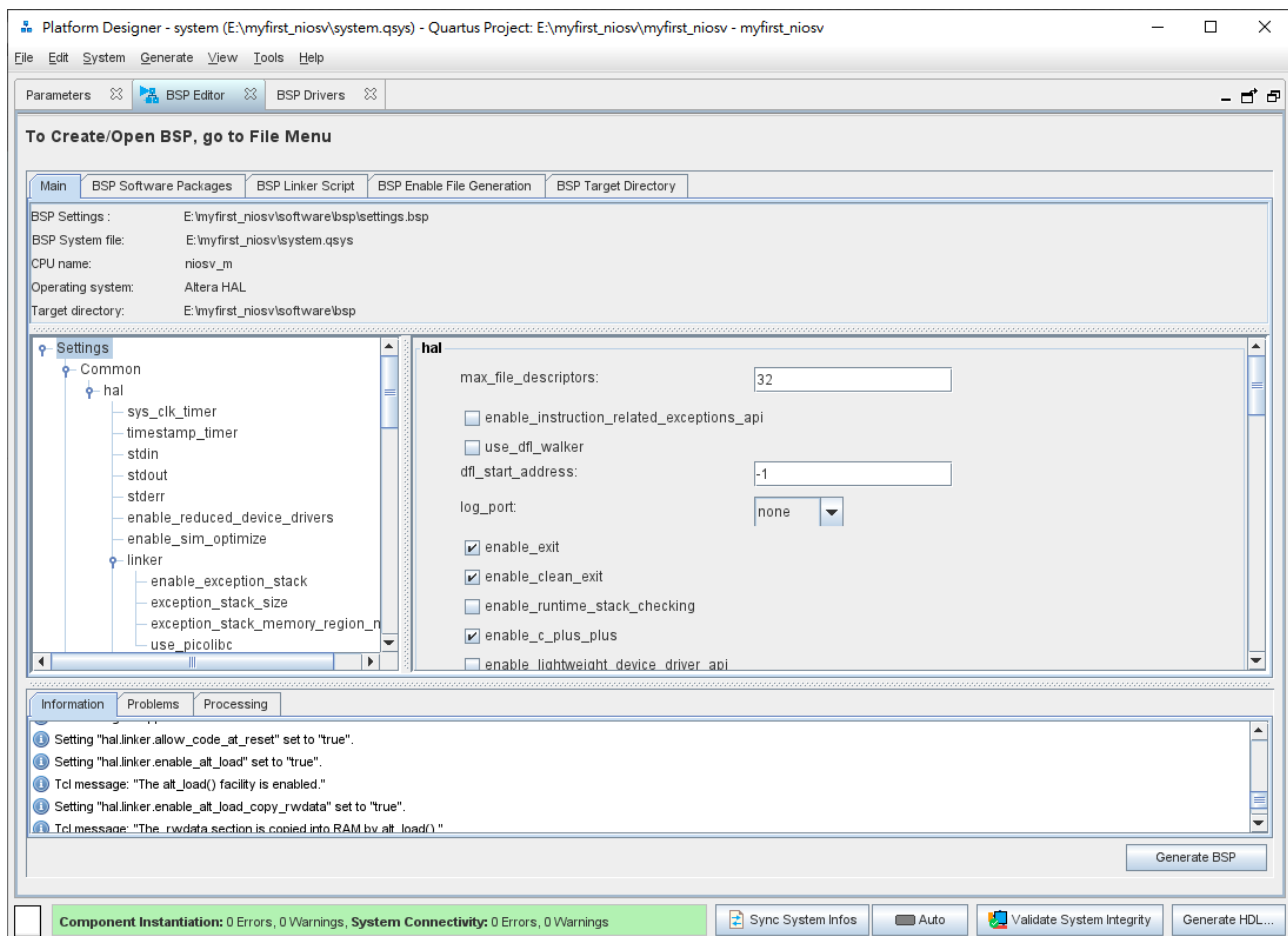


Figure 3-41 Maximizing the 'BSP Editor' tab.

7. Click Generate BSP

The following content will introduce how to configure a software project to use hardware memory. By modifying the System Library Properties in the Ashling RiscFree IDE, you can precisely configure the memory mapping.

1. In the Ashling RiscFree IDE, right-click **app** and choose **System Library Properties**. The **Properties for app_syslib** dialog box opens.
2. Click **System Library**. The **System Library** page contains settings related to how the program interacts with the underlying hardware. The settings have names that correspond to the targeted NIOS V hardware.
3. In the **BSP Linker Script** tab, observe which memory has been assigned for **Block Started by Symbol** memory (.bss), exception handler memory (.exceptions), Heap memory, Read-only

data memory (.rodata), Read/write data memory (.rwdata), Stack memory and Program memory (.text), see Figure 3-42. These settings determine which memory is used to store the compiled executable program when the example **app** programs runs. In the **Main** tab, You can also specify which interface you want to use for stdio, stdin, and stderr. You can also configure build options to support C++, reduced device drivers, etc.

4. Choose **onchip_memory** for all the memory options in the **BSP Linker Script** tab. See Figure 3-42 for an example.

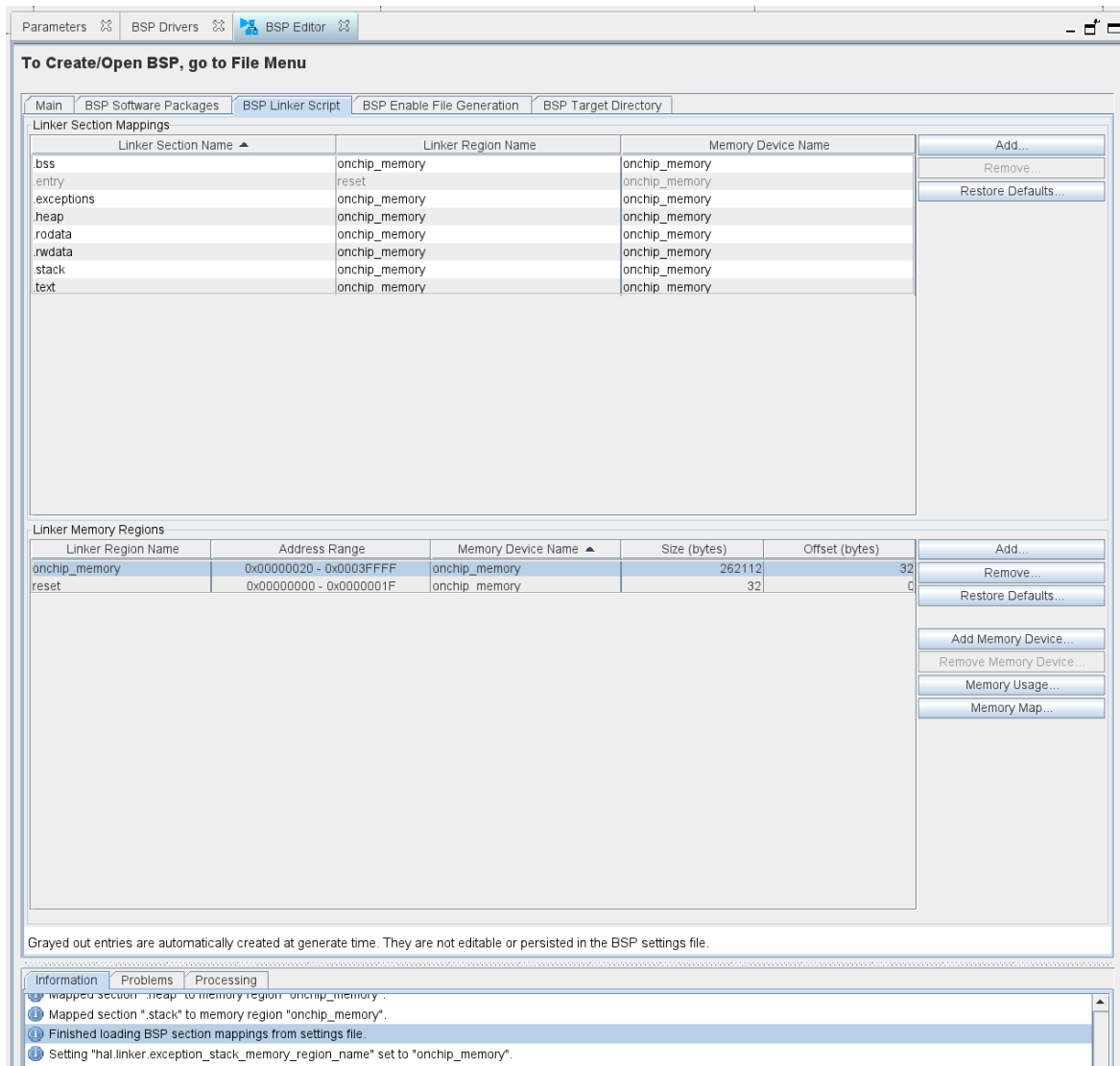


Figure 3-42 Configuring System Library Properties

5. Click **Generate BSP** to regenerate BSP. Close the **Platform Designer** Windows and return to the IDE workbench.

Note: If you make changes to the system properties you must rebuild your project. To rebuild, right-click the **app** project in the **Projects Explorer** tab and choose **Build Project**.

Chapter 4

Appendix

4.1 Revision History

<i>Version</i>	<i>Change Log</i>
V1.0	Initail version

4.2 Copyright Statement

Copyright © Terasic Inc. All Rights Reserved.