COMPSYS303
Lab1 – Create a Nios II System with Qsys

Building a Nios II System with Qsys on DE2-115 & Software Development

Last Updated: 12th July 2018

1. Hardware design of the DE2-115 Board

1.1 Create a Quartus II project and start a new design

Start "Quartus Prime 16.1", select "New Project Wizard" from the startup screen.
Click "Next" in the introduction page as shown in Figure 1-1.

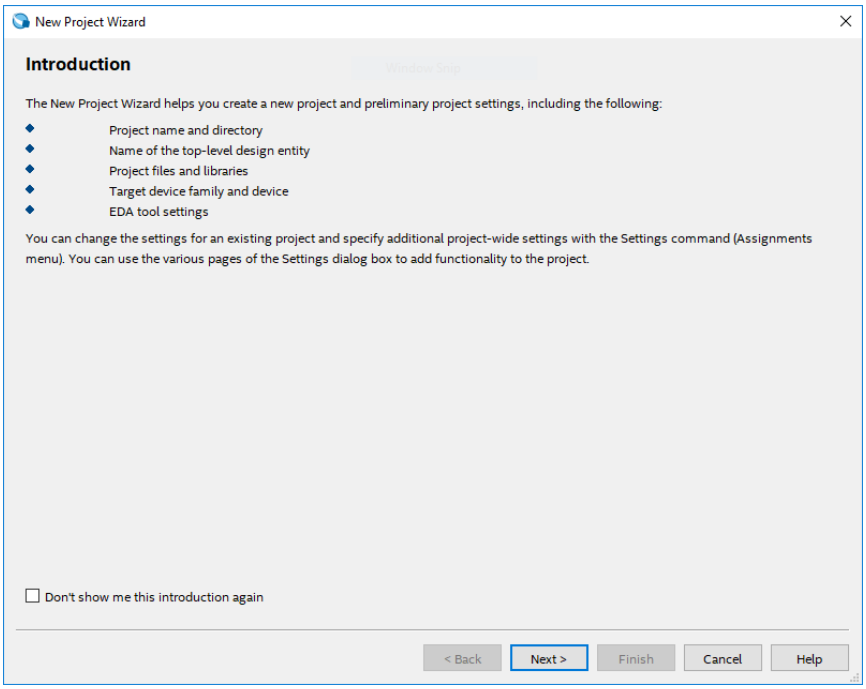

Figure 1-1: The introduction page

In the following step, as shown in Figure 1-2, which let you to decide the name of the directory where the design files will be located, and the name of the top-level entity design. We decided our files will be in "**D:\CS303-Lab1**" (you should pick somewhere on D drive), and the name of the project will be **lab1**. Note that the default name of the top-level design will be the same as the name of project. You are free to change any of these names, we just leave it as default here. Click "Next" after entering the corresponding fields. If the project directory does not exist, the wizard will ask you to create the particular directory, choose "Yes".

The wizard may now ask you whether you want to create an Empty project or a Project template. Just pick an "Empty project".

In the next step, the wizard will ask you to add file(s) to the project in Figure 1-3, click "Next" here whereas the necessary files will be added in the later sections of the tutorial.

Select "**Cyclone IV E**" as the device family in the drop-down box, and "**EP4CE115F29C7**" as the target device in the "Family and Device Settings page" shown in the Figure 1-4. Click "Next" to next wizard page.

Since we are going to use Quartus II mainly for the course, click "Next" to let Quartus II be the synthesis, simulation, and analysis tools as default, this is shown in Figure 1-5. The final page of the wizard will show the project information briefly as in the Figure 1-6. You may click "Finish" to proceed.

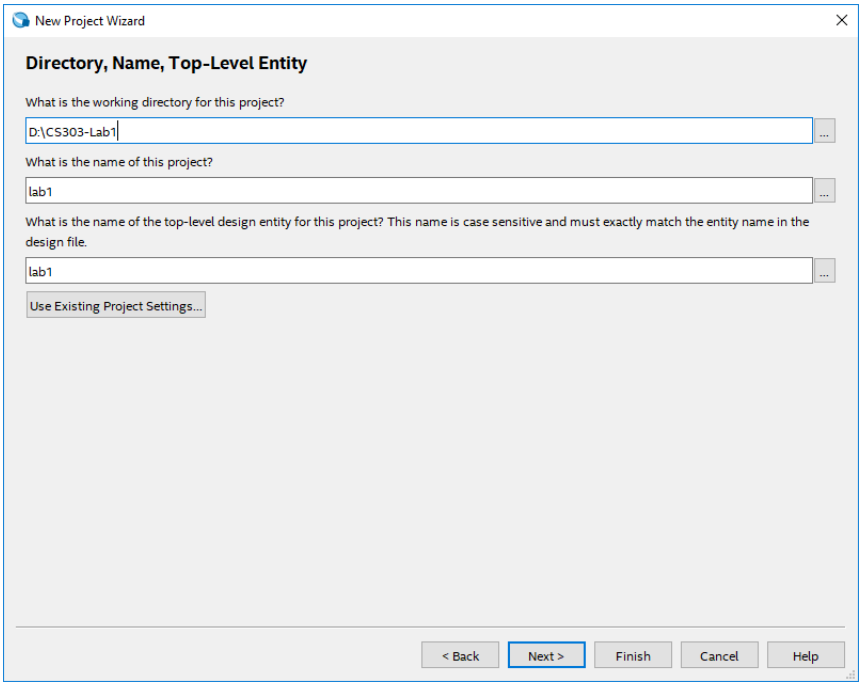

Figure 1-2: Project name and directory and the name of the top-level design

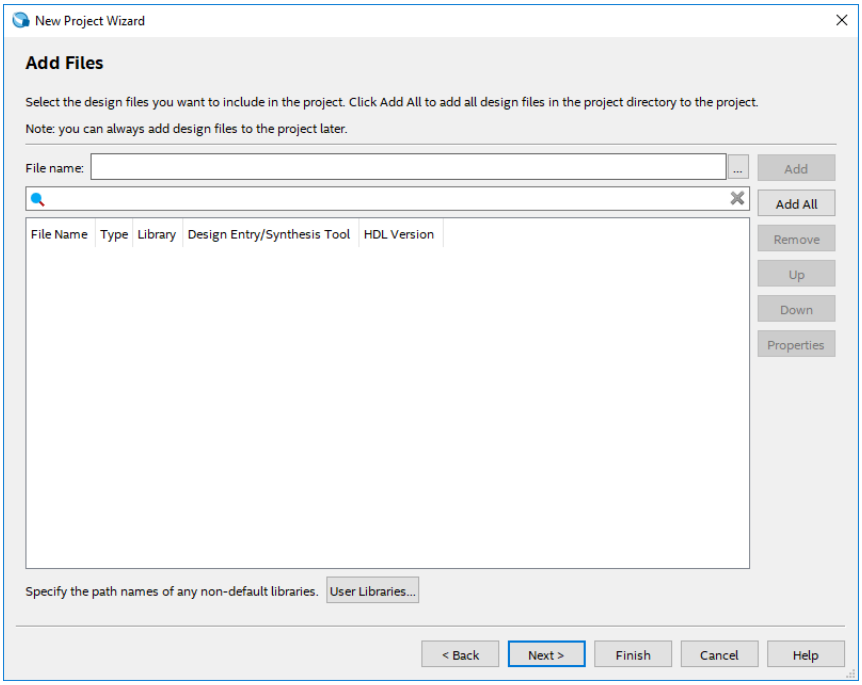

Figure 1-3: Adding files to the project

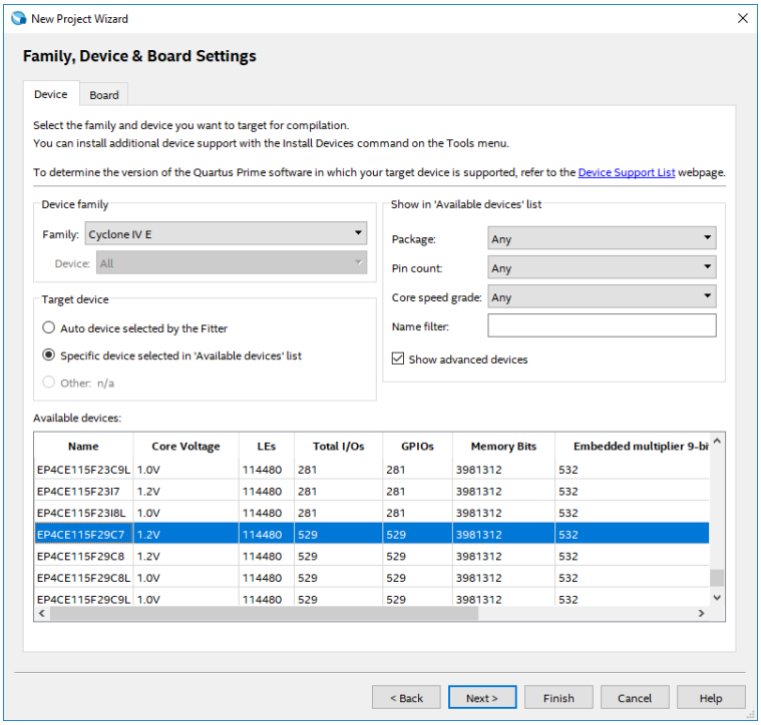

Figure 1-4: Family and Device Settings dialog

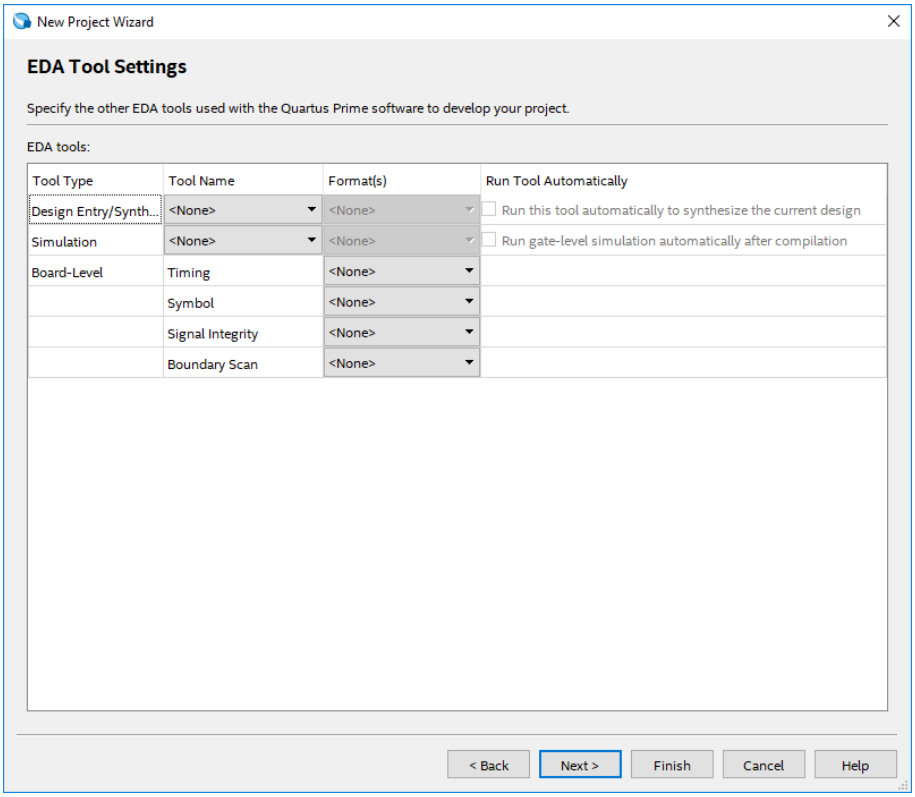

Figure 1-5: EDA tools setting

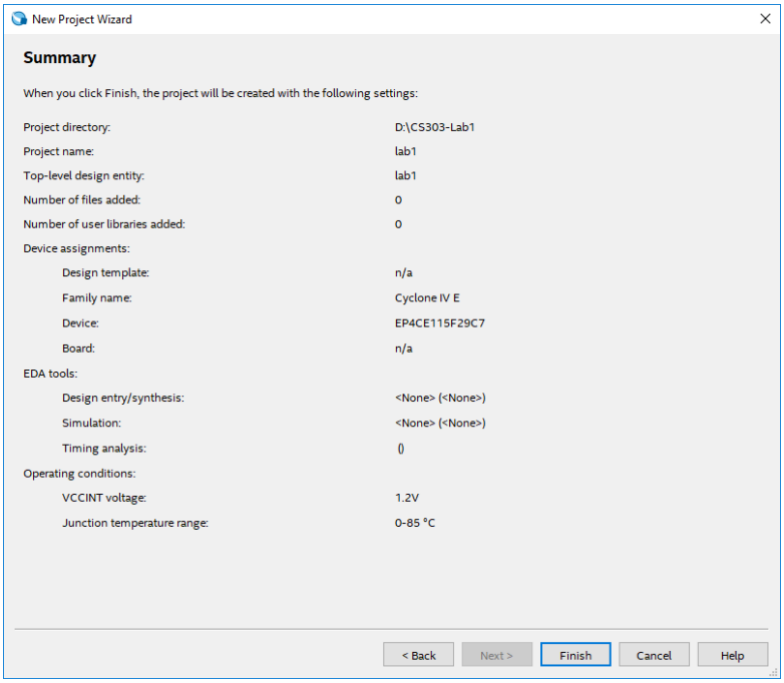

Figure 1-6: Project summary

The next step is to initiate NIOS II processor, which interconnects peripherals through dedicated interfaces. First create a schematic as our top-level design by selecting "New" from the "File" menu. Choose "**Block Diagram/Schematic File (under Design Files)**" and click "OK" as shown in Figure 1-7.

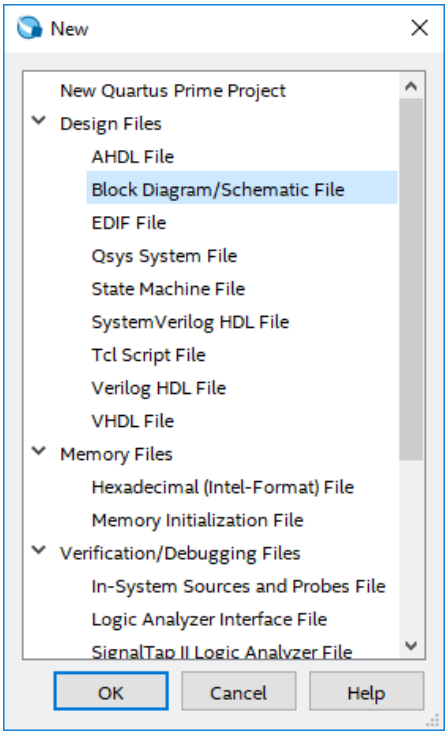

Figure 1-7: Create a new schematic file

The schematic design space will show up in the Quartus II main window, as illustrated in Figure 1-8. Users can carry out design procedures with available symbols of hardware components and obtain integration of each component through wiring. Save the schematic (as **lab1.bdf** in this case) through "File->Save as" and tick "Add file to current project" in the file saving dialog as presented in Figure 1-9.

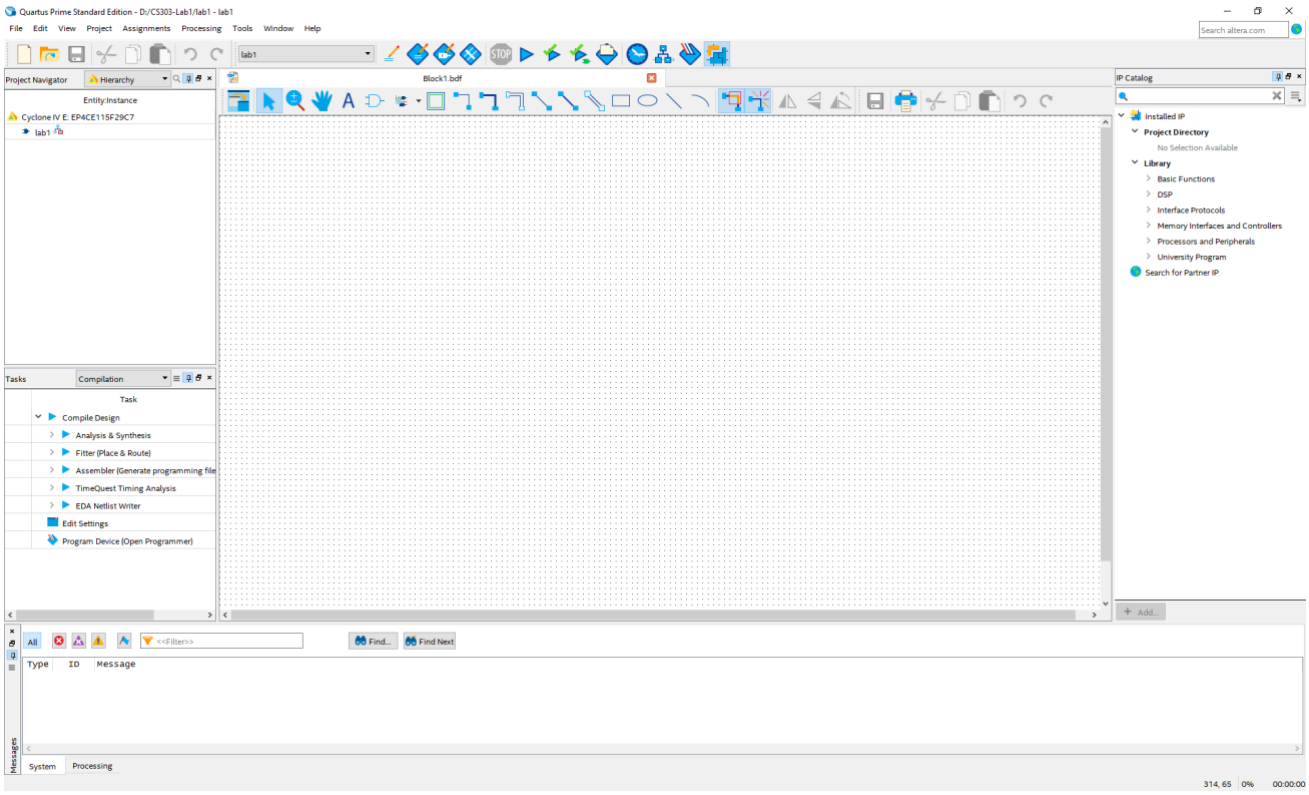

Figure 1-8: Quartus schematic design window

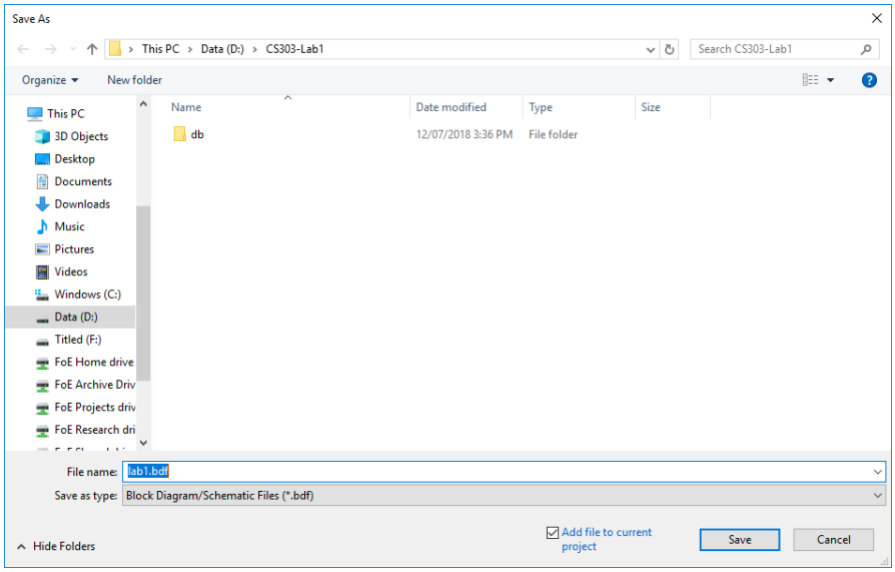

Figure 1-9: Save the schematic and add it to the current project

1.2 Qsys and Nios II system

We start the SOPC design through following steps, with these steps, a Nios II processor will be instantiated, and so will the peripherals and corresponding interfaces. Select "Tools" > "Qsys" to open Qsys. You will see a splash screen as shown in Figure 1-10 as it starts up.

Figure 1-10: Starting up Qsys

Once Qsys starts up, let's name the Qsys file we're about to create. Go to "File->Save As", and save the file as "nios2_system.qsys" as shown in Figure 1-11.

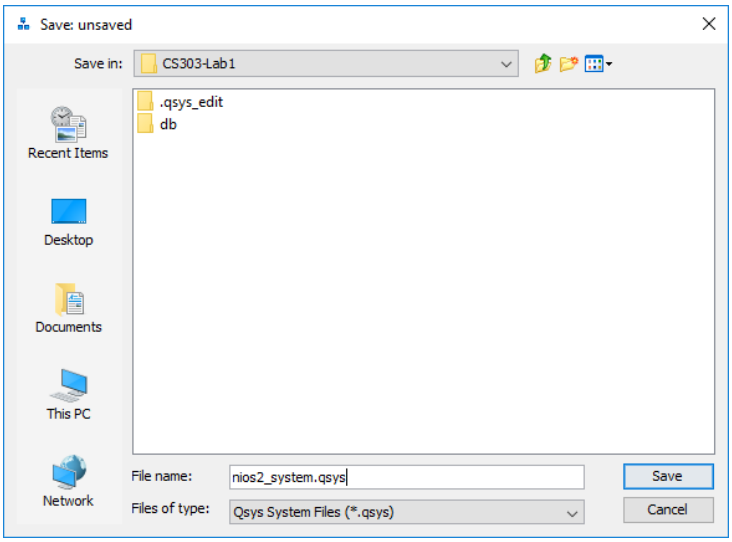

Figure 1-11: Save the file

1.2.1 Add the Nios II processor

Nios II is the centralized unit of our target system, it can be added by selecting "**Nios II Processor**" under the "**Library->Embedded Processors**" and clicking "**Add…**" to include the processor as shown in Figure 1-12. Select **Nios II/f**, which is the most powerful (and consumes the most logic elements among the two Nios configurations) as shown in Figure 1-13. We are going to use the default settings of this processor, click "**Finish**" to go straight ahead.

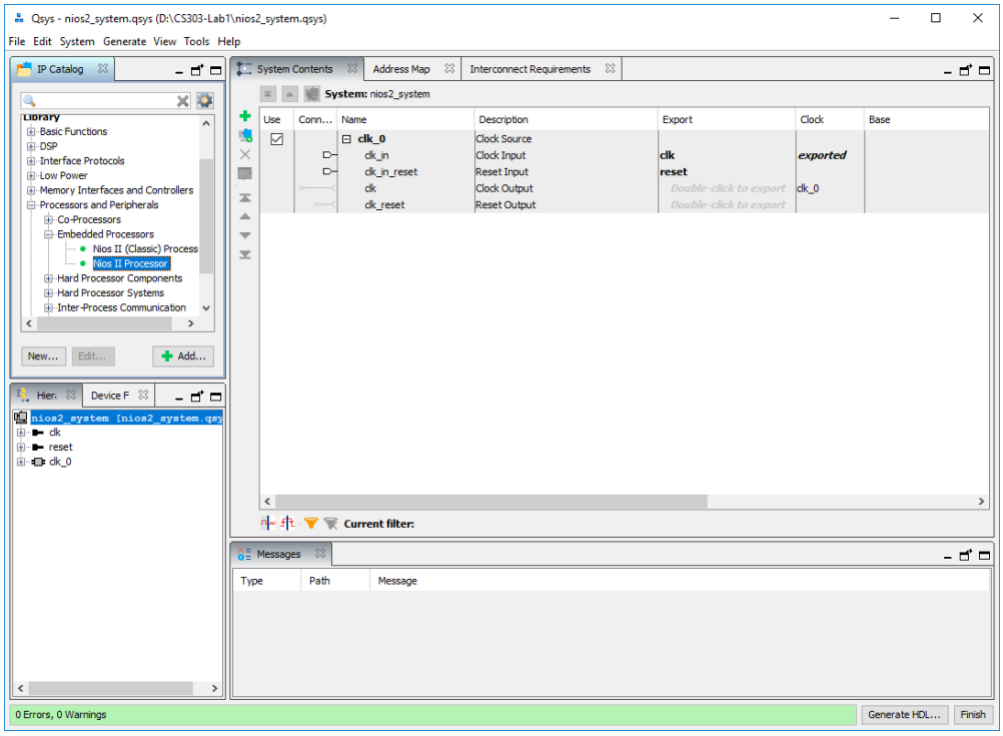

Figure 1-12: Add the Nios II processor

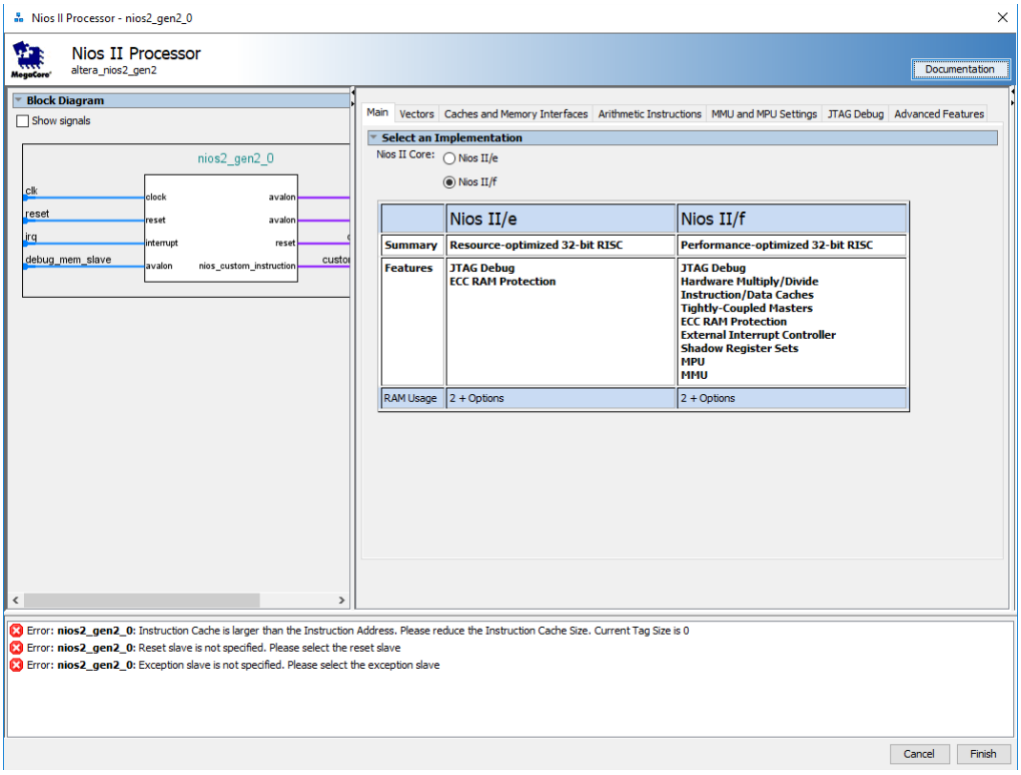

Figure 1-13: Nios II configurations

1.2.3 Add SDRAM PLL

Later on we will be adding some SDRAM memory to the design. However, this type of memory requires a clock with a certain clock delay relative to the main clock. Go under **"University Program->Clock"** and select **System and SDRAM Clocks for DE-series Boards**. This block will easily generate the required clock to drive the SDRAM. The only option you can change here is the specific board that we are using which in our case is the DE2-115 as shown in Figure 1-14. Then click "Finish" to proceed.

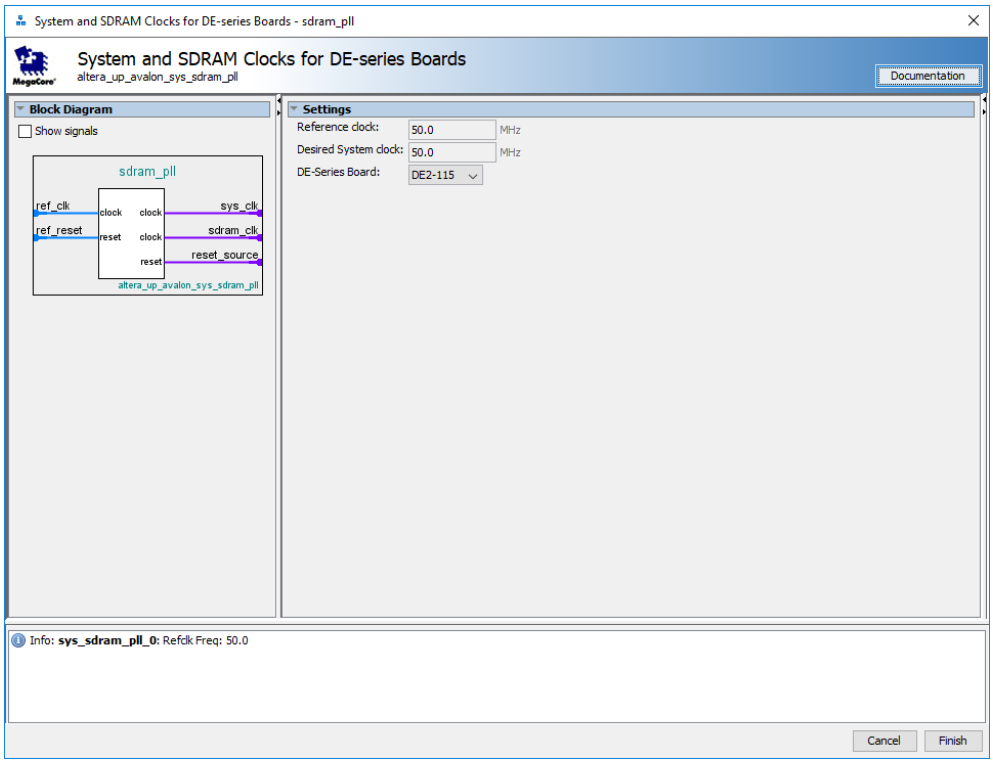

Figure 1-14 Adding SDRAM PLL

1.2.2 Add On-Chip RAM

We can use memory bits **within the FPGA board** to create **on-chip memory** to be used by the processor. Go under "**Library->Basic Functions->On-Chip Memory**" select "**On-Chip Memory (RAM or ROM)**" and click "Add…" as shown in Figure 1-15. Allocate **30KB** (**30,720B**) of RAM **memory size** and change **Data width** to **16**, according to Figure 1-15. Click "Finish" button to proceed. Note that you might want to decrease the size of on chip RAM in your future designs, since memory bits are precious. You might find some errors regarding to memory settings, we will resolve that in the later stage.

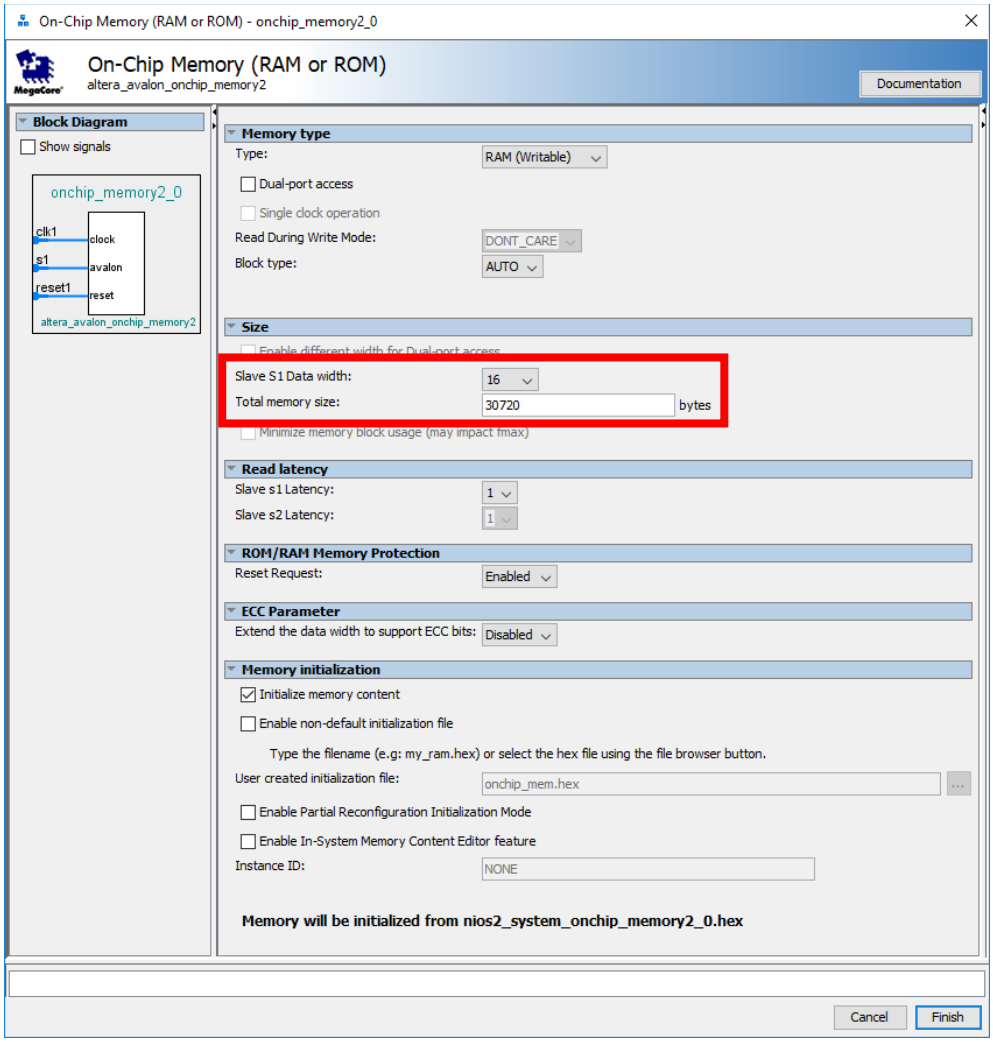

Figure 1-15 Adding On-Chip RAM to the nios2 system

1.2.3 Add the SDRAM module

Memories are used by Nios II to store programs and data. In this design, the program will be loaded to the **8MB SDRAM** on the DE2 board. To use the SDRAM, add the "**SDRAM Controller**" under the "**Library->Memory Interfaces and Controllers->SDRAM**". Change the **Data width** to **16 bits**. The rest of the settings are illustrated in Figure 1-16. Click "Finish" to add the SDRAM module.

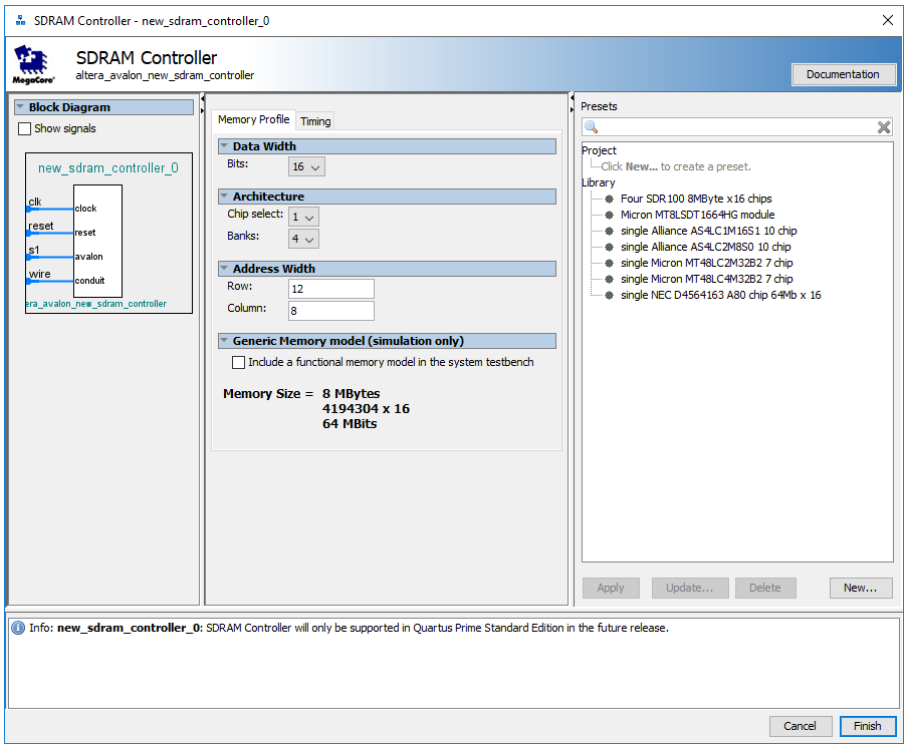

Figure 1-16: The settings of The SDRAM controller

1.2.7 Add the UART port support

To use the UART port on the DE2 development board, simply select "**UART (RS-232 Serial Port)**" under "**Library->Interface Protocols->Serial**", set **Baud rate** to **115200**, **Parity** to **None**, **Data bits** to **8**, **Stop bits** to **1**, **Synchronizer stages** to **2**, and tick the "**Include end-of-packet register**" detailed in Figure 1-17. Flow control is unavailable since the CTS/RTS are not connected on the DE2 board. Click "Finish" to add the component.

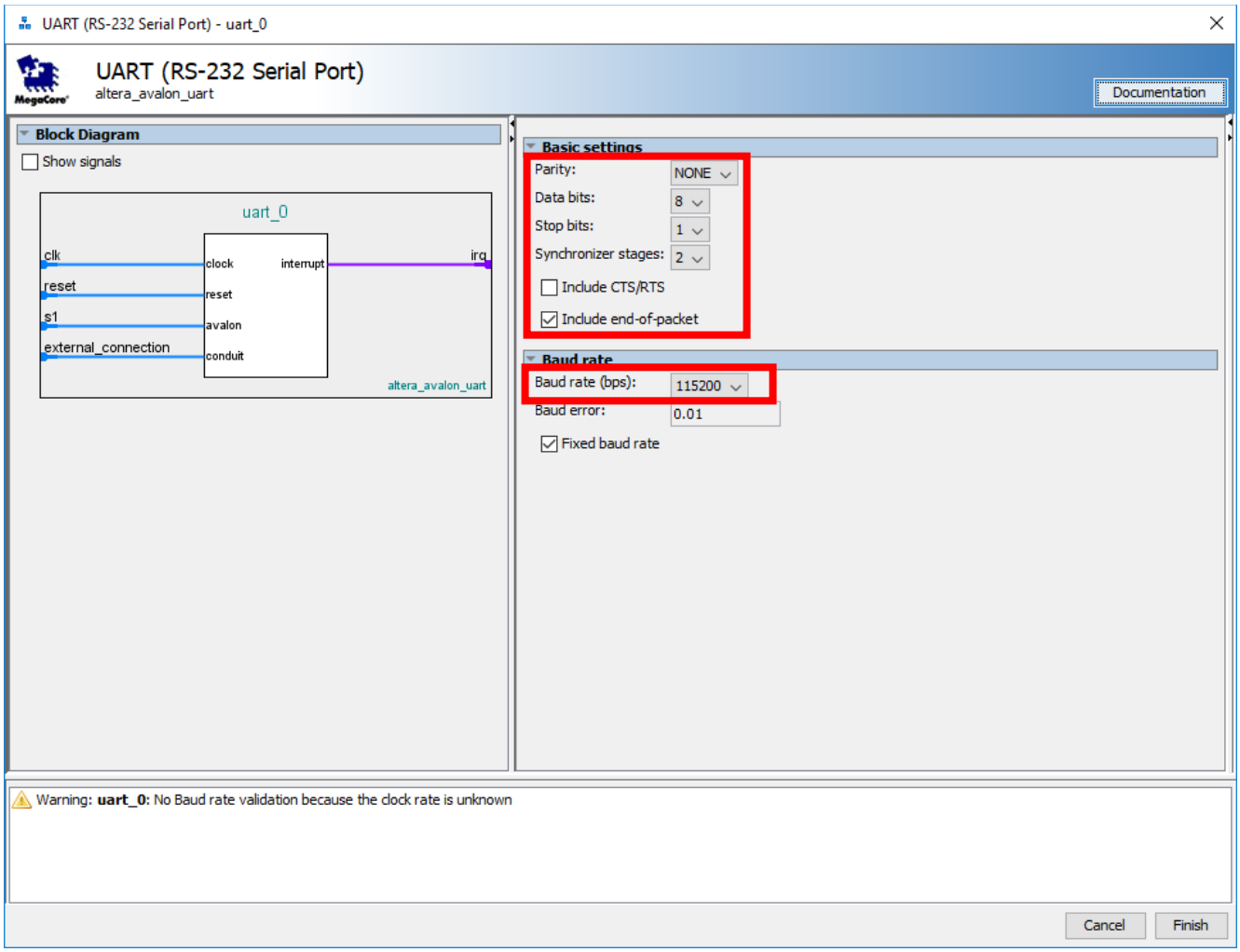

Figure 1-17: Settings of the UART port

1.2.8 Add the JTAG UART support

It is good to add the JTAG UART support to the system, which is very useful in the debugging process of software development on Nios II where the input/output of the Nios II terminal are via this components. "**JTAG UART**" is located under the "**Library->Interface Protocols->Serial**" section, and details of the settings are in Figure 1-18.

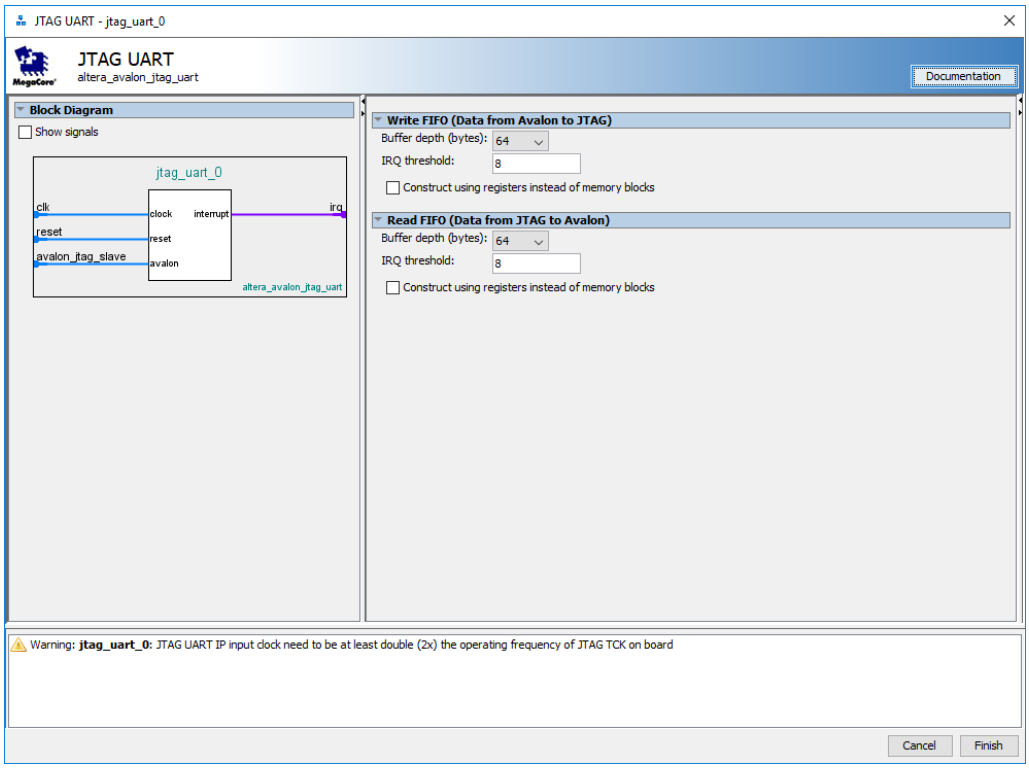

Figure 1-18: Configuration of the JTAG UART

1.2.9 Add the interval timers

Timers are useful for some software, which can be added from "**Library->Processors and Peripherals->Peripherals**" by selecting the "**Interval timer**". **Two** timers of **periods of 1 ms** are added in this case which are shown in Figure 1-19. Remember to add **two** of these.

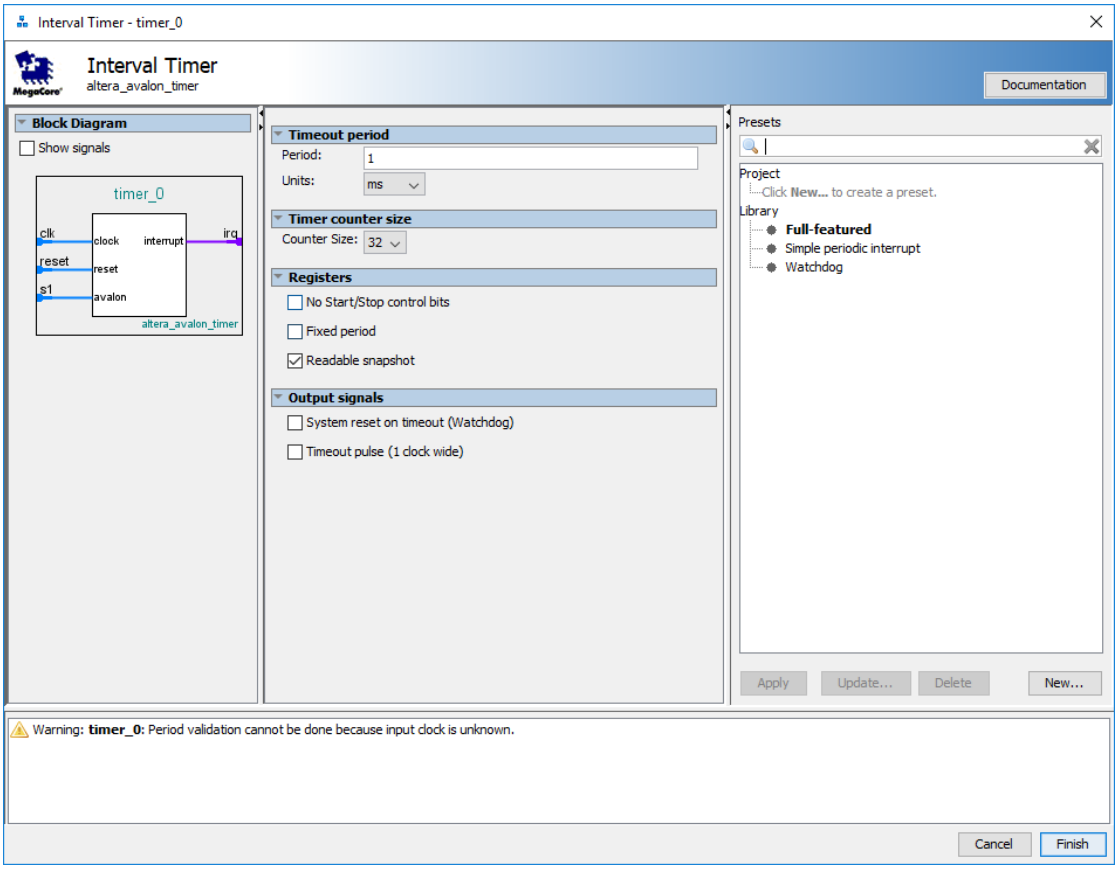

Figure 1-19: Configuration of two interval timers

1.2.10 Add the 3 push button PIOs

4 push buttons on the DE2 development board are accessed through parallel input ports, however we will be using one as a system-wide reset. Each bit represents a single button on the DE2. When a button is pushed, an interrupt will be generated on the falling edge. Selected "**PIO (Parallel I/O)**" from "**Library->Processors and Peripherals->Peripherals**" The settings of the push button ports are as follows, **Width is 3 bits, Input port only.** Select **Synchronously capture at Falling edge, Generate IRQ at Edge**, which are also detailed in Figure 1-20.

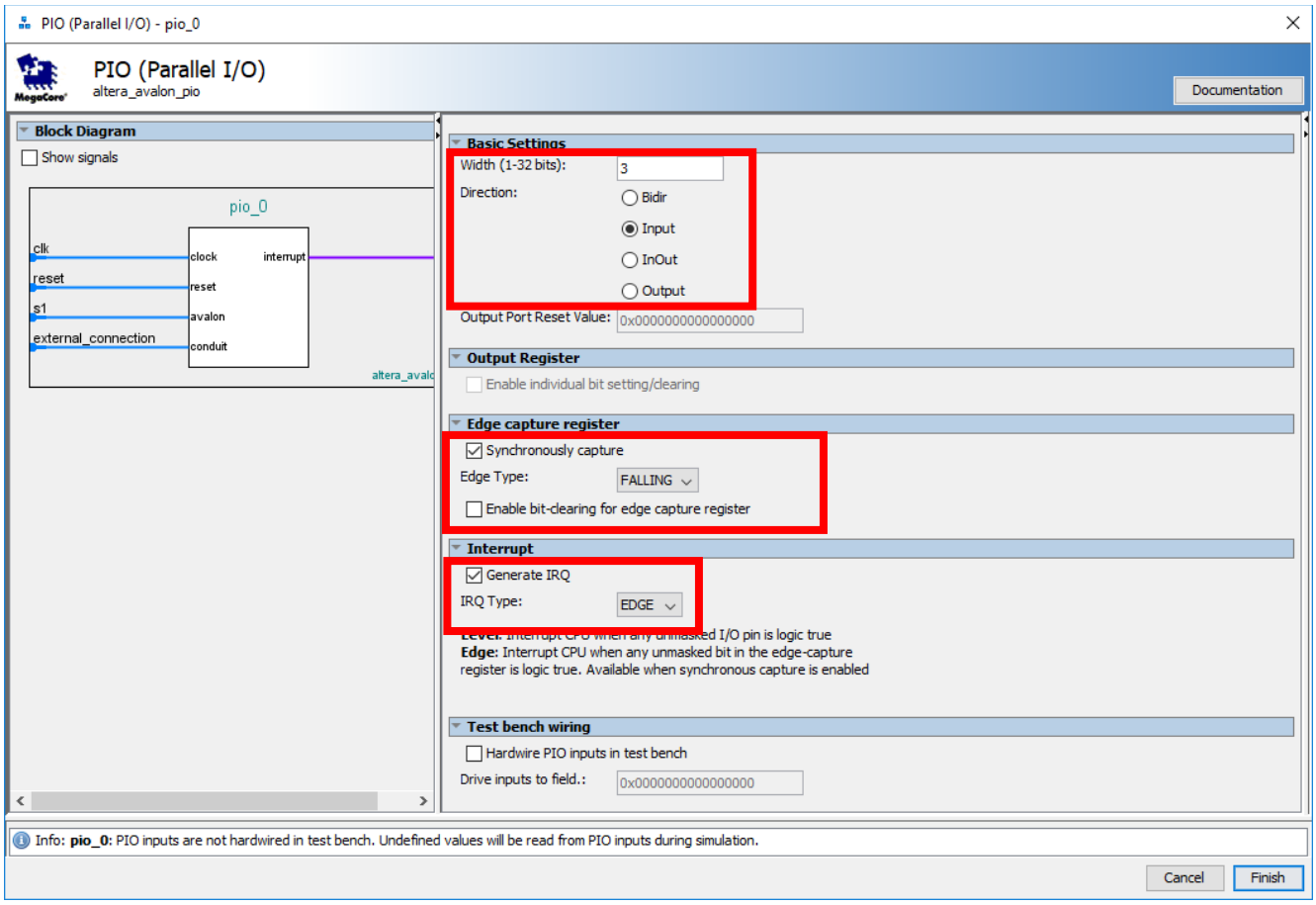

Figure 1-20: Configuration of the 3 push buttons

1.2.11 Add a PIO module to access 18 switches

Similar to push buttons, 18 switches which are left to the push buttons are facilitated as parallel input ports. Configurations of switches are illustrated in Figure 1-21. Once add the component, rename it to **switches**. One of the differences between switches and buttons is that, **interrupts** are enabled in the buttons but **not for switches**.

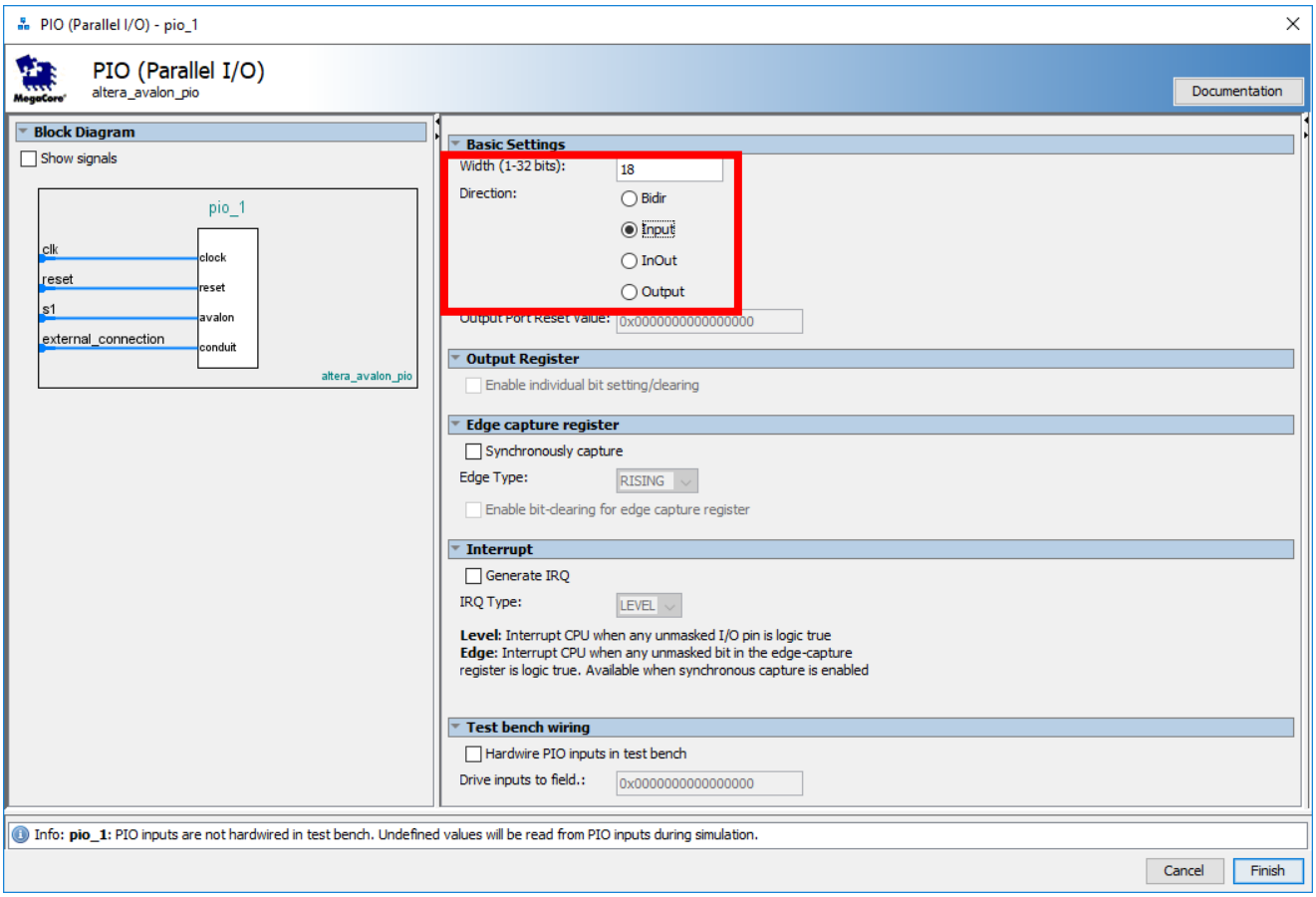

Figure 1-21: Configurations of 18 switches on DE2

1.2.12 Add the LED modules as parallel output

The 18 red and 9 green LEDs are parts of parallel inputs and outputs of the system, which can be included by adding "**PIO (Parallel I/O)**" under the "**Library->Processors and Peripherals->Peripherals**" section. The red and green LEDs are **output port only** as detailed in Figure 1-22.

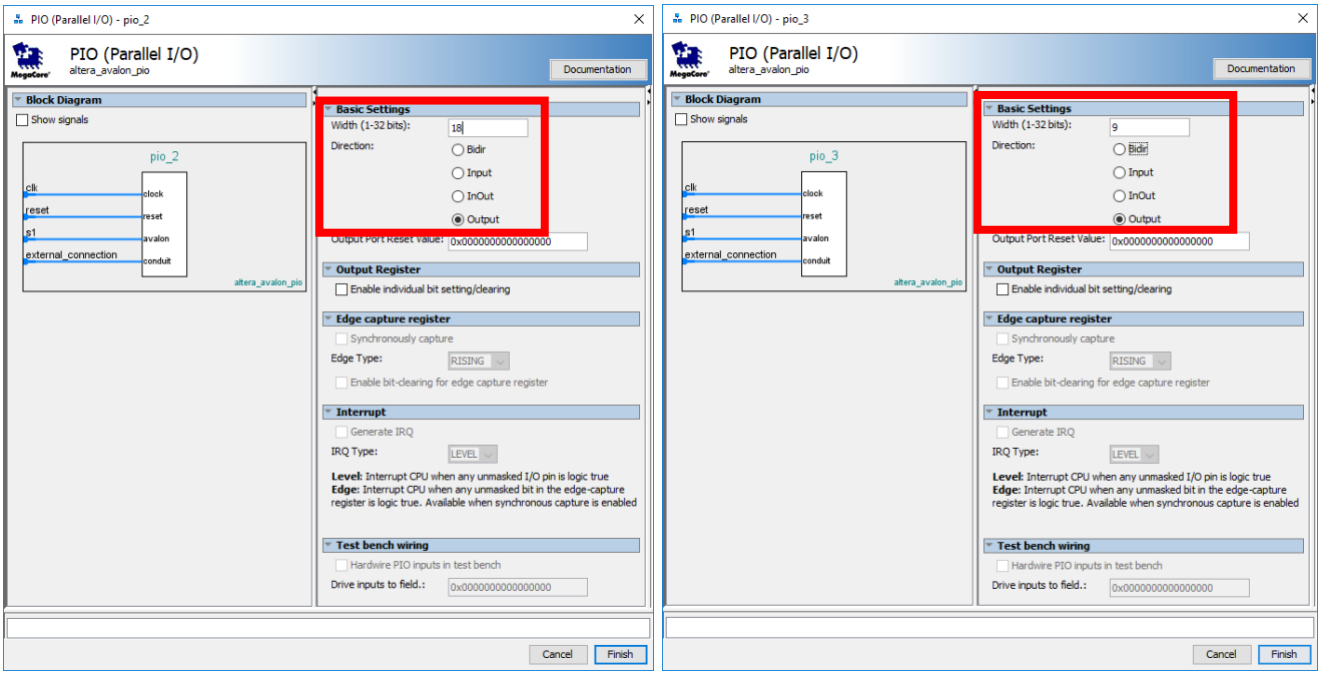

Figure 1-22: Settings of the LEDs

1.2.13 Add the Character LCD display

To work with the 2 lines character LCD display on the DE2-115, add "**Library->Processors and Peripherals->Peripherals->Altera Avalon LCD 16207**" where a 16x2, Optrex 16207 will be added to the system.

1.2.14 Renaming and Connecting components

In order to connect all the components, click the dots on the joins between two vertical and horizontal lines. The **clk** input should go through the **SDRAM PLL** to each component (via the **system_clk** signal) while the **reset** line can simply connect to all components. All components should be connected to the **data_master** of the Nios II CPU, while the **On-Chip Memory** and **SDRAM Controller** should also be connected to the **instruction_master**.

For any components with an IRQ wire, also connect this to the **irq** input on the CPU.

Any wires with a type of "Conduit" should also be exported to the outside of the CPU, by double clicking the "Export" column.

The final setup should be as shown in Figures 1-23 through 1-25. **Make sure the names are the same as the diagram below.**

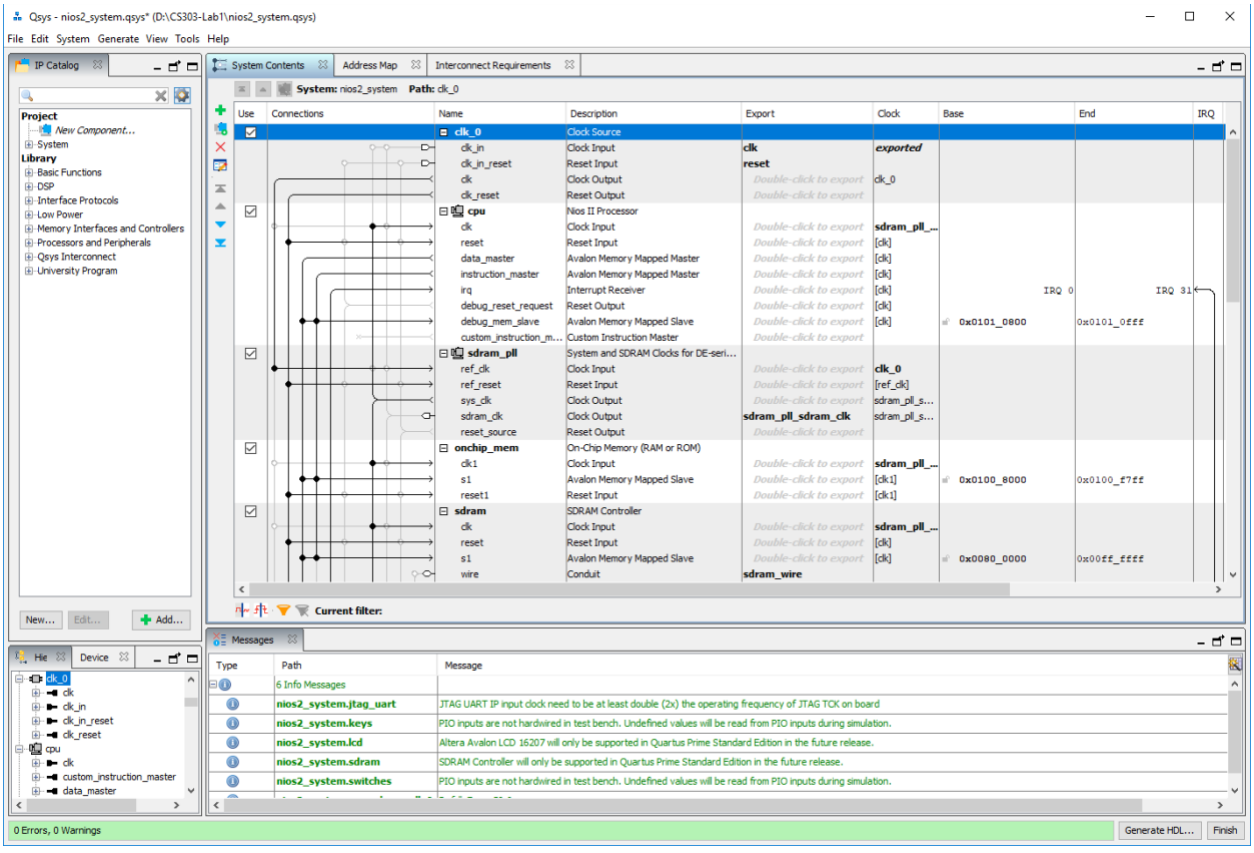

Figure 1-23: Connections to the CPU

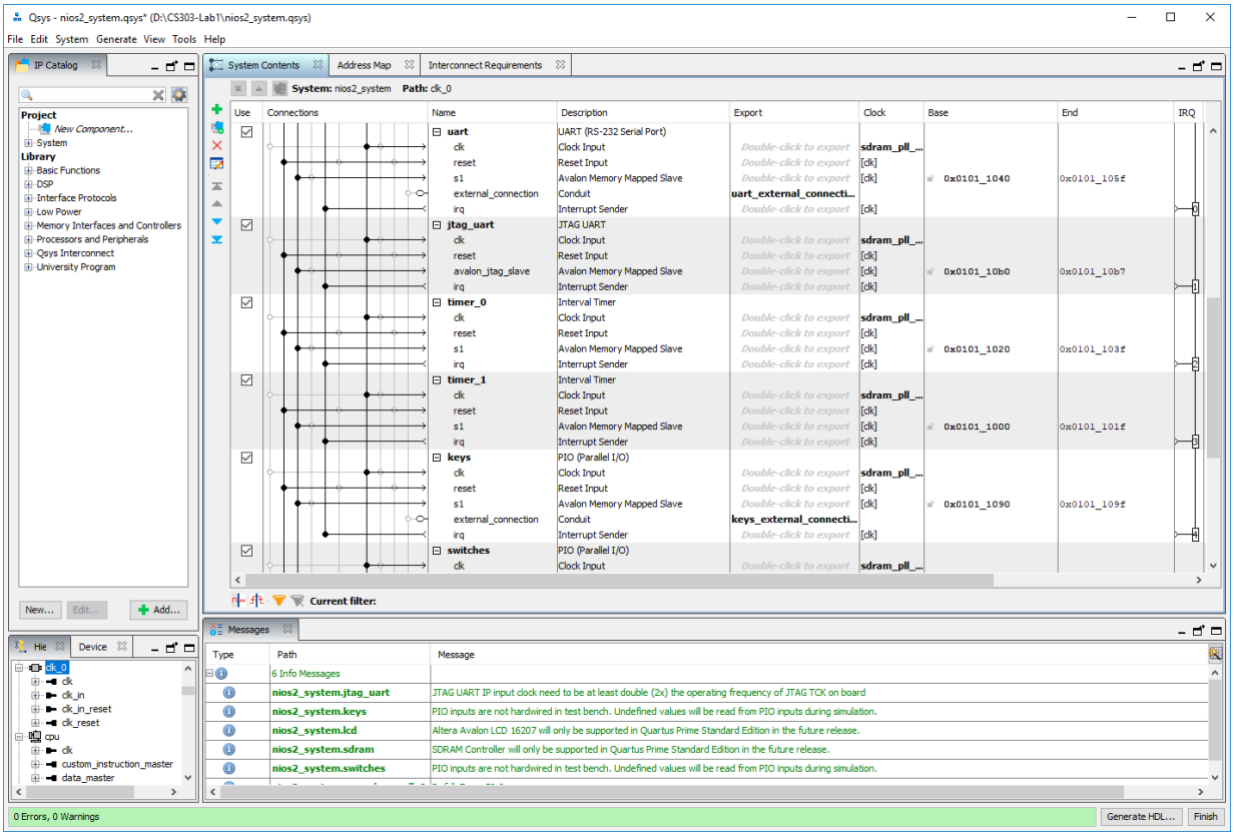

Figure 1-24: Connections to the CPU

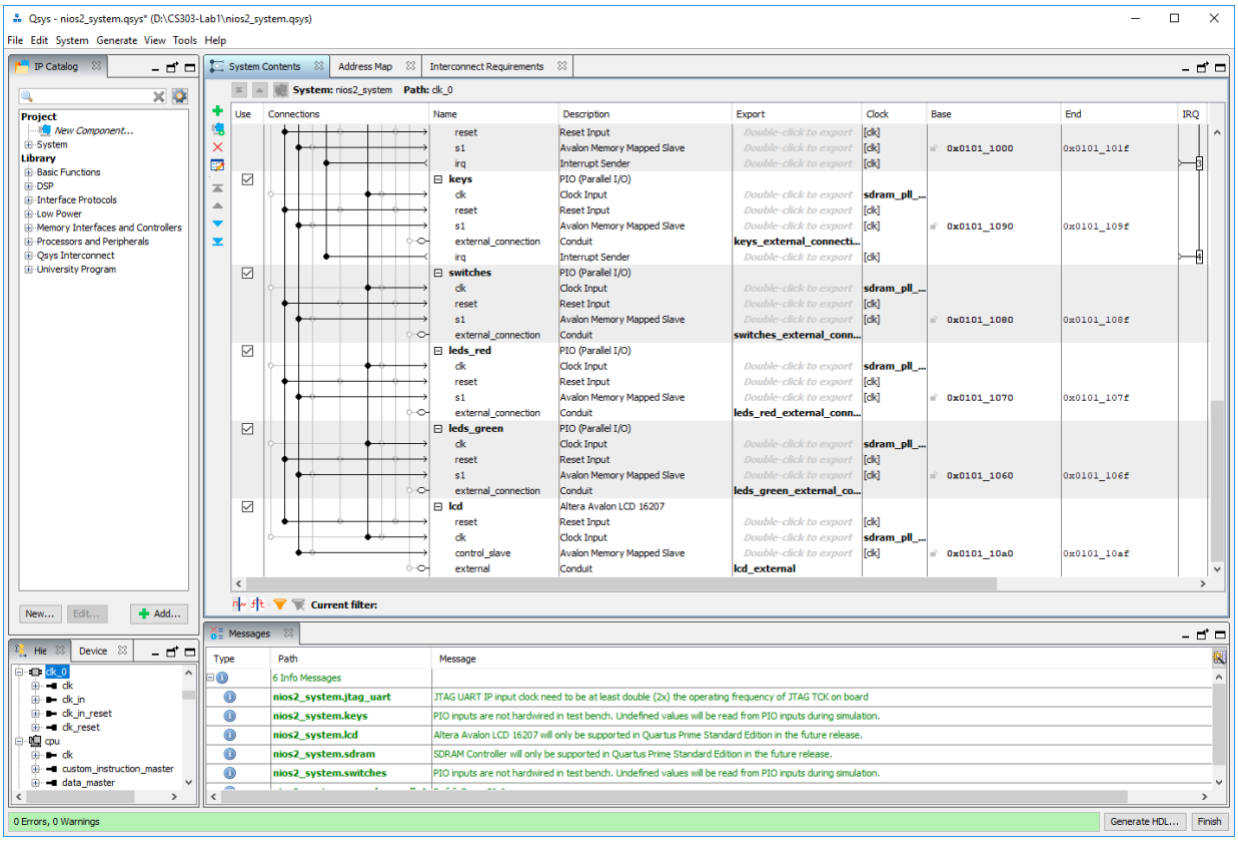

Figure 1-25: Connections to the CPU

1.2.15 Resolve conflicts of base addresses

After adding up the required components, we have to ensure the addresses and IRQ number of each components do not conflict each other. It is achieved through menu "**System->Assign Base Addresses**", you should find out that a lot of error messages are suppressed. Now we can sort out the IRQ conflicts in the similar fashion through "**System->Assign Interrupt Numbers**". You can assign IRQ manually depending on the priorities/importance of reactivity of a device. Now all the conflicts should be resolved as shown in Figure 1-26 (by right clicking and selecting "Collapse All"), the address might not be the same).

Use	C...	Name	Description	Export	Clock	Base	End	IRQ
☑		clk_0	Clock Source		*exported*			
☑		cpu	Nios II Processor		sdram_pll_...	0x0101_0800	0x0101_0fff	
☑		sdram_pll	System and SDRAM Clocks for DE-seri...		clk_0			
☑		onchip_mem	On-Chip Memory (RAM or ROM)		sdram_pll_...	0x0100_8000	0x0100_f7ff	
☑		sdram	SDRAM Controller		sdram_pll_...	0x0080_0000	0x00ff_ffff	
☑		uart	UART (RS-232 Serial Port)		sdram_pll_...	0x0101_1040	0x0101_105f	0
☑		jtag_uart	JTAG UART		sdram_pll_...	0x0101_10b0	0x0101_10b7	1
☑		timer_0	Interval Timer		sdram_pll_...	0x0101_1020	0x0101_103f	2
☑		timer_1	Interval Timer		sdram_pll_...	0x0101_1000	0x0101_101f	3
☑		keys	PIO (Parallel I/O)		sdram_pll_...	0x0101_1090	0x0101_109f	4
☑		switches	PIO (Parallel I/O)		sdram_pll_...	0x0101_1080	0x0101_108f	
☑		leds_red	PIO (Parallel I/O)		sdram_pll_...	0x0101_1070	0x0101_107f	
☑		leds_green	PIO (Parallel I/O)		sdram_pll_...	0x0101_1060	0x0101_106f	
☑		lcd	Altera Avalon LCD 16207		sdram_pll_...	0x0101_10a0	0x0101_10af	

Figure 1-26: The components and their IRQ and addresses in current nios2 system

1.2.16 Checking the module names

Module names for every design are not necessarily to be the same, for saving some troubles and simplifying this lab, module names are asked to be fixed. You can change the name of the component by right clicking on the name of the module and choosing "rename" to input the preferred name. The names of modules are as follows (order is not important, but names have to be the same):

1. cpu – Nios II Processor
2. sdram_pll – SDRAM PLL
3. onchip_mem – On-Chip Memory (RAM or ROM)
4. sdram – SDRAM controllers
5. uart – UART (RS-232 Serial Port)
6. jtag_uart – JTAG UART
7. timer_0 – Interval Timer
8. timer_1 – Interval Timer
9. keys – PIO
10. switches – PIO
11. leds_red – PIO
12. leds_green – PIO
13. lcd – Character LCD

1.2.17 More about Nios II processor settings and generation of the system

Since currently two memories (on-chip RAM and SDRAM) are available, we will need to decide the usages of them. For instance, the place to store RESET address, or interrupt vector table. Double click on the Nios II CPU from the component list. It will show the details of the Nios II processor as in Figure 1-13. Select the "Vectors" tab. Reset address is used when reset happens, the program will start from the reset address once the **program is downloaded** or **the processor is reset**. We set the **Reset vector at offset 0x0 of the sdram**, which is going to store the program as well, and set **Exception vector to 0x20 of the sdram**, as illustrated in Figure 1-27. Click "Finish" to proceed.

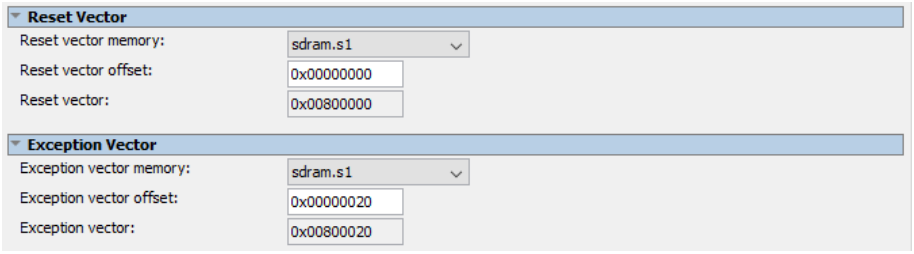

Figure 1-27: The setting of the vectors

1.2.18 Generating the CPU

Select "Generate" -> "Generate HDL" From the menu, you will be presented with a dialog as in Figure 1-28. Make sure that no simulation model is being created, and no testbench system, and ensure that a "bsf" file will be created. Click the "Generate" button (at the bottom) to start generating the configured system. Click "Save" if it says the system has been modified. This may take a while.
Click "**Close**" after the system is generated successfully as shown in Figure 1-29.

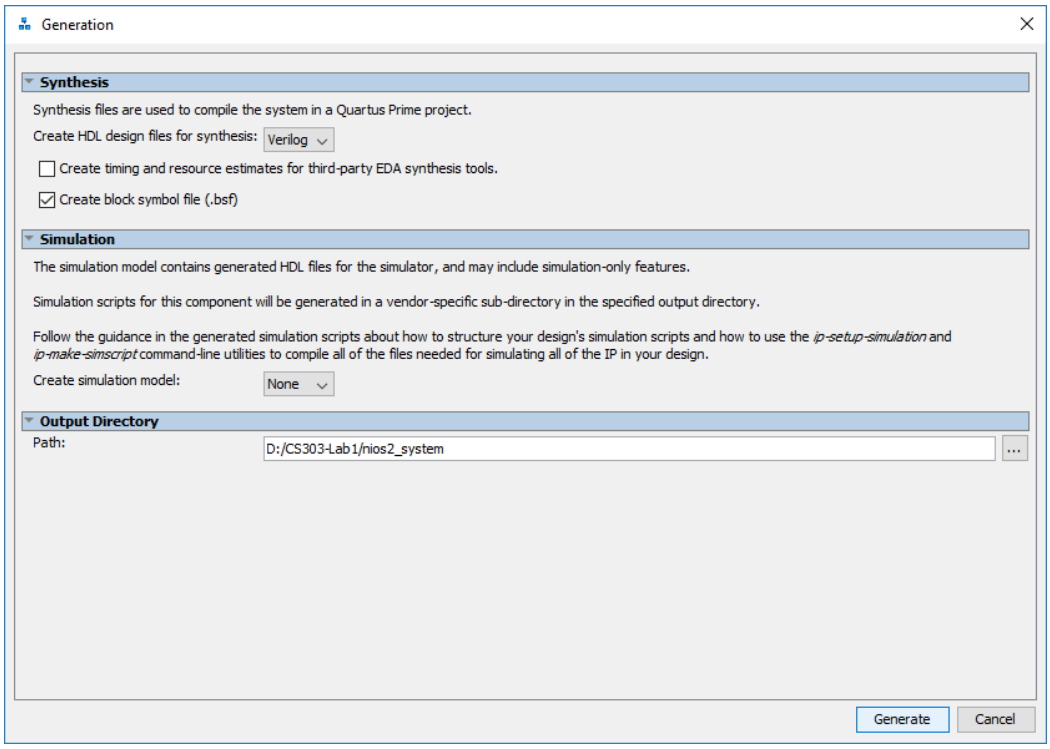

Figure 1-28: Generating the Nios II system

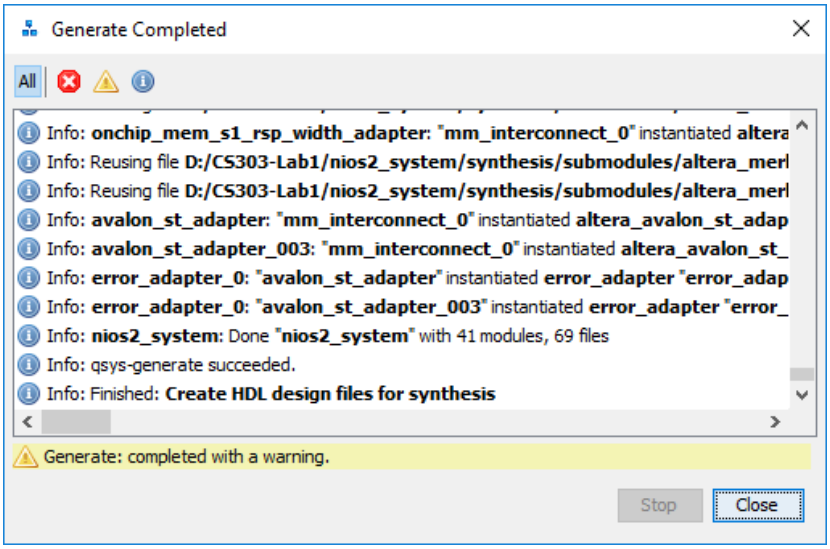

Figure 1-29: Generating the Nios II system

1.2.18 Adding the Nios II processor to Quartus

We must now add the ".qip" (Quartus II IP File) to our Quartus project so that the CPU can be compiled. Go to **Project->Add/Remove Files in Project** in the menu bar. A dialog like that shown in Figure 1-30 will appear.

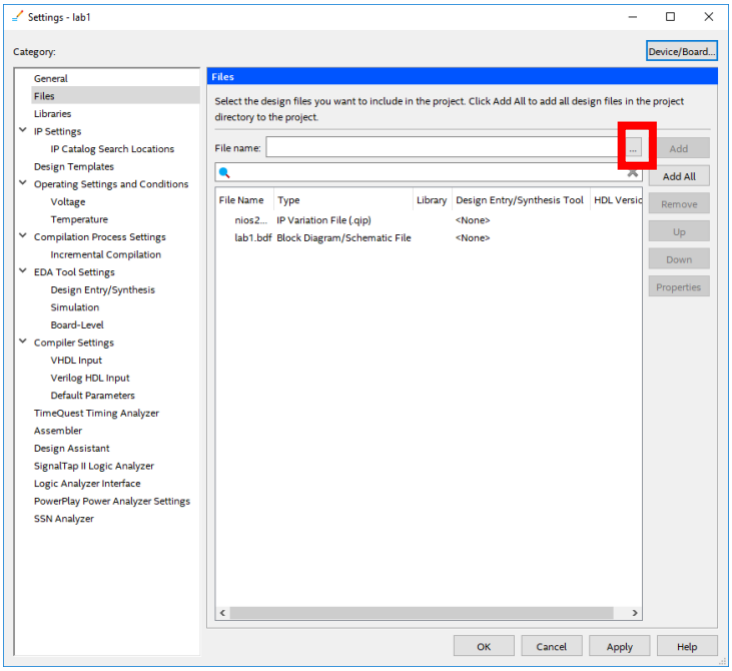

Figure 1-30: Add/Remove Files dialog

Click on the "**…**" button next to the "File Name" box, change the type to "Script Files (*.tcl, *.sdc, *.qip, *.sip)" and navigate to **D:\CS303-Lab1\nios2_system\synthesis**. If you changed the location or name of any of your files this path may differ. Select the "nios2_system.qip" file and Add it.

Now, double click on any empty space in the "bdf" file. A symbol picker will show as shown in Figure 1-31. Click on the "…" button next to "Name" and find the "bsf" file that was generated in **D:\CS303-Lab1\nios2_system**. Double click it, and then click "OK".

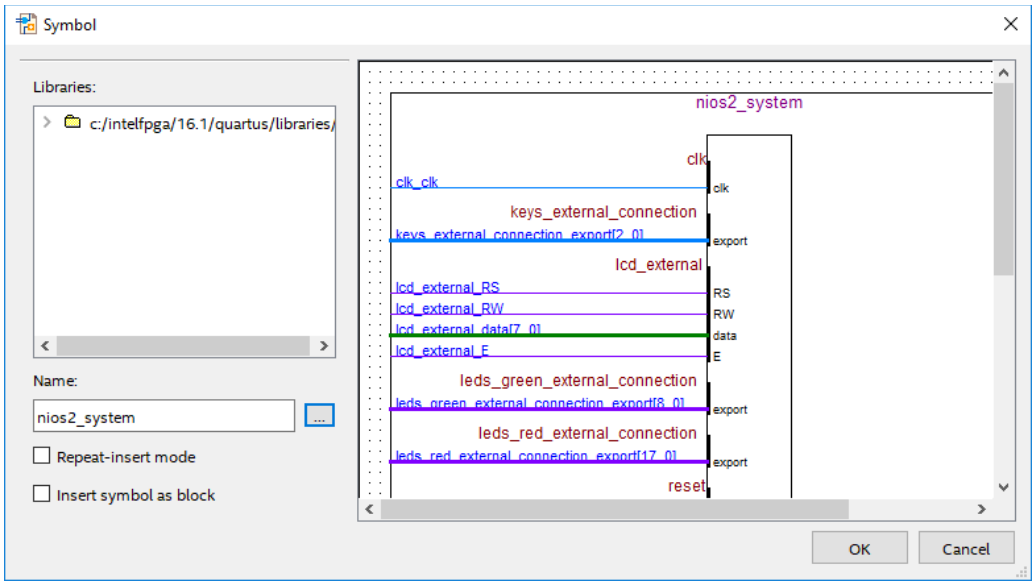

Figure 1-31: Nios II system as an available component

From the schematic, you can tell the characteristics of any particular input output port. The blue lines indicate input ports, while **thicker** ones are **buses** and thinner ones are bit lines. The purple lines are output ports, where thicker and thinner lines mean the same thing. The green lines indicate bi-directional ports, which is used for both inputs and outputs.

1.3 Integrating the Nios II system with other hardware components

Now we have to create inputs and outputs on the top level design and interconnections between the nios2_system and its environment. **Double click any space on schematic again**, you can see 3 kinds of pins – **bidir** (bi-directional), **input**, and **output** under the **primitives**, as shown in Figure 1-32. Press "**Cancel**" and return to your design.

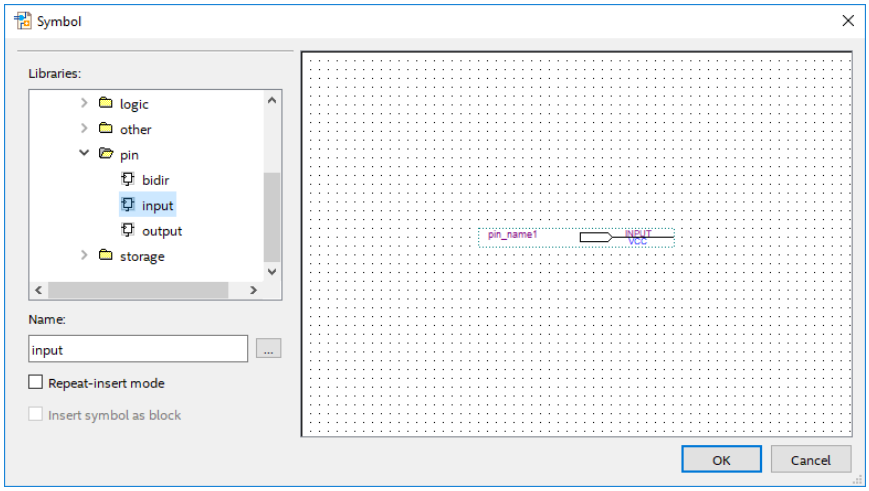

Figure 1-32: Input pin from primitives

You can add each pin manually, but the fastest way to do this is via "Generate Pins for Symbol Ports". Right click on each symbol in your design and select "**Generate Pins for Symbol Ports**" as shown in Figure 1-33. Repeat this for all other blocks in your design.

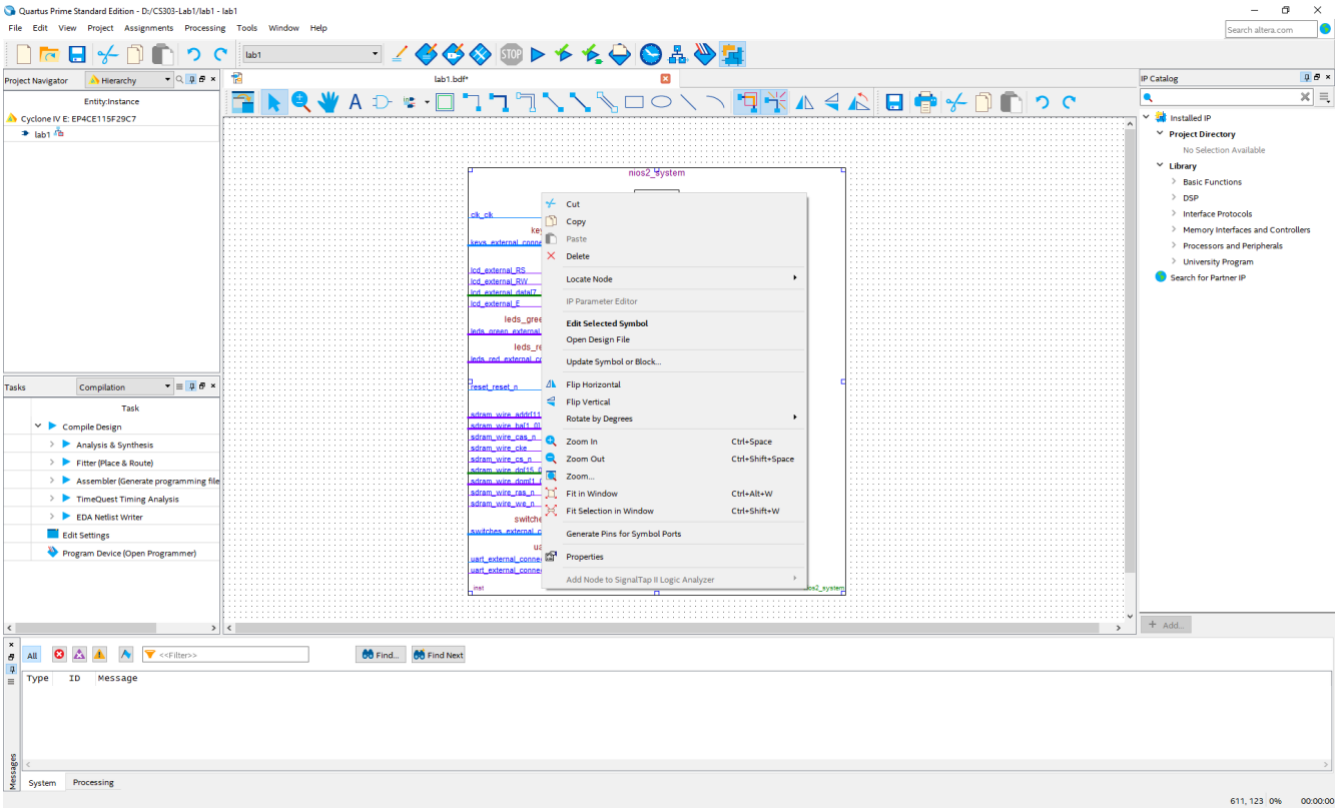

Figure 1-33: Generate pins for Symbol Ports

Next, we need to rename the pins (by double clicking on the pin name) to the following:
(you should end up with something similar to Figure 1-34.

Block: nios2_system

Inputs:

1. **CLOCK_50** – clk_clk
2. **KEY[3]** – reset_reset_n
3. **KEY[2..0]** – keys_external_connection[2..0]
4. **SW[17..0]** – switches_external_connection[17..0]
5. **UART_RXD** – uart_external_connection_rxd

Outputs/ Bidirectional:
LCD_EN – lcd_external_E

LCD_RS – lcd_external_RS

LCD_RW – lcd_external_RW

LCD_DATA[7..0] – lcd_external_data[7..0]

LEDG[8..0] – leds_green_external_connection[8..0]

LEDR[17..0] – leds_red_external_connection[17..0]

DRAM_ADDR[11..0] – sdram_wire_addr[11..0]

DRAM_BA[1..0]– sdram_wire_ba[1..0]

DRAM_CAS_N – sdram_wire_cas_n

DRAM_CKE – sdram_wire_cke

DRAM_CS_N – sdram_wire_cs_n

DRAM_DQ[15..0] – sdram_wire_dq[15..0]

DRAM_DQM[1..0] – sdram_wire_dqm[1..0]

DRAM_RAS_N – sdram_wire_ras_n

DRAM_WE_N – sdram_wire_we_n

UART_TXD – uart_external_connection_txd

Some of the peripherals require constant high signal values to work properly. These could be pins such as RESET or ENABLE. In our design following pins need to be asserted all the time. We need to connect them to the "vcc". Double click on the bdf, search and add **vcc** and **output** in the symbol library to achieve Figure 1-35.

LCD_ON – vcc (LCD on)

LCD_BLON – vcc

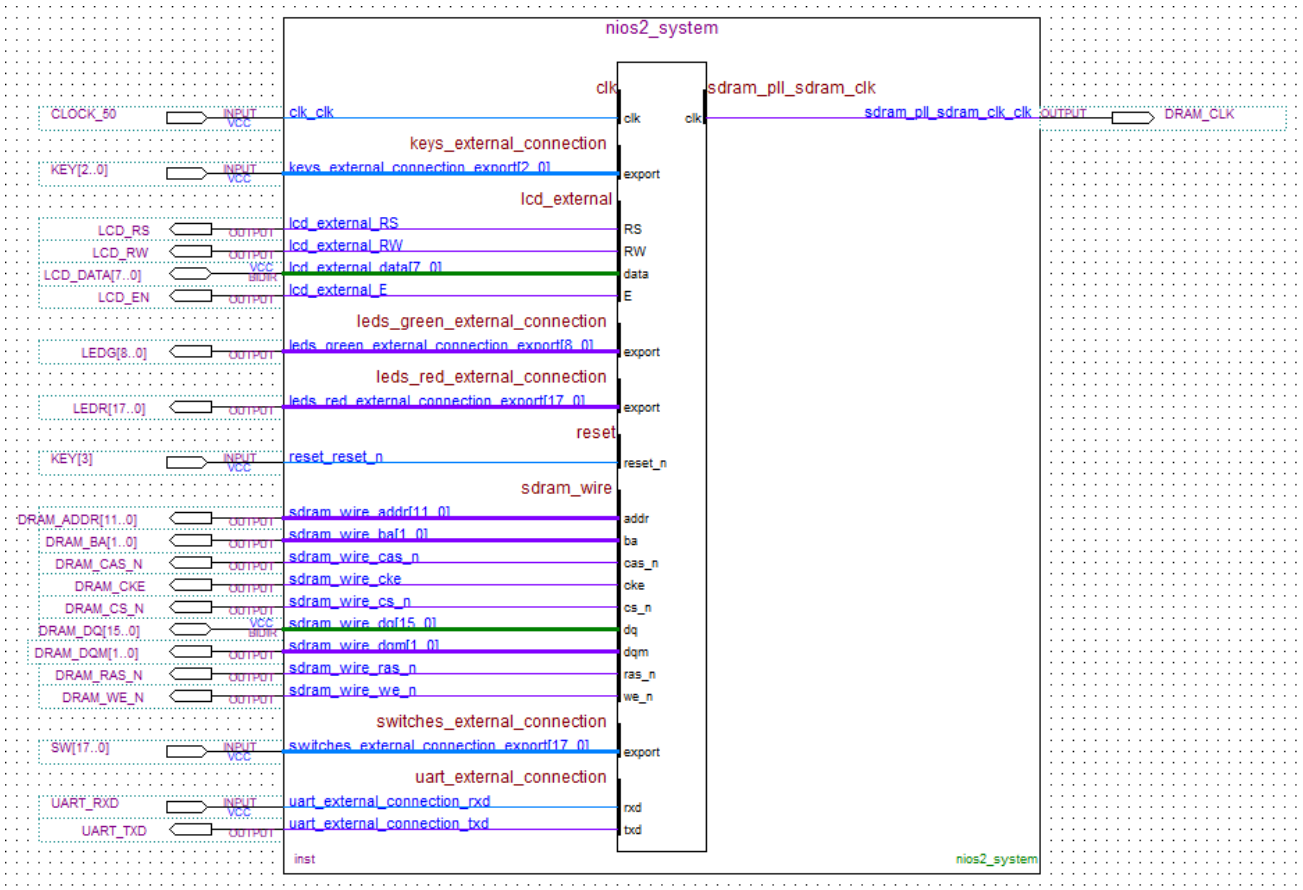

Figure 1-34: Final input/output pins for the design

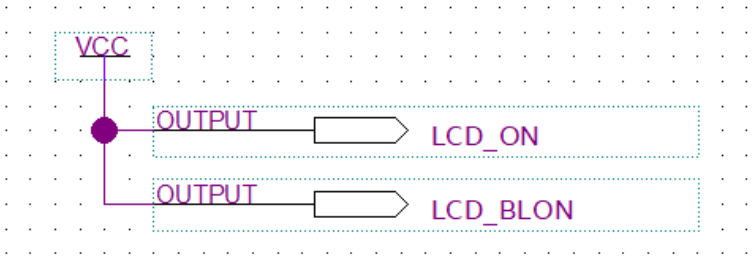

Figure 1-35: always high

We have so far defined the pins; we now need to connect them to the DE2 hardware. First we will do the pin assignments and later inter connect modules.

To save your time, we provided an assignment file (**DE2_115_pin_assignments.csv**) available to be imported. First clear all existing assignments. Then from Quartus menu "**Assignments->Import assignments...**" and select the input file **DE2_115_pin_assignments.csv** (provided to you) followed by clicking "OK" as shown in Figure 1-36.

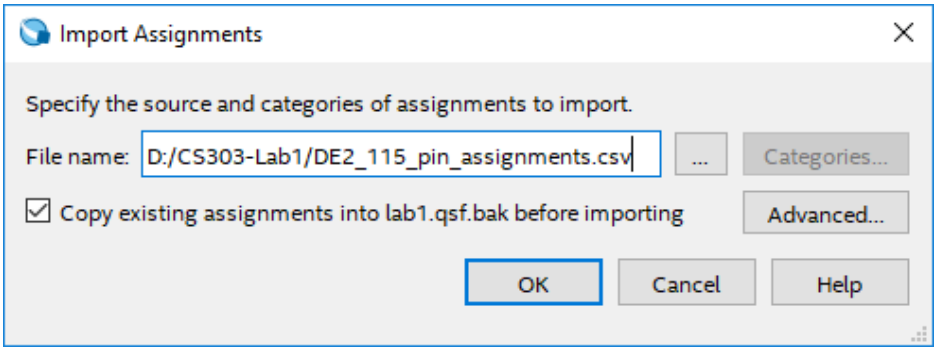

Figure 1-36: Importing an existing assignment file

Now the design is ready to be compiled, select "Start Compilation" from the "Processing" menu of Quartus II, or just simply click the purple arrow button to start the compilation process. After a successful compilation, a report will be generated as shown in Figure 1-37. If you run into compilation issues due to instance names, simply go to the properties of the block causing the issues (right-click menu) and change the instance name.

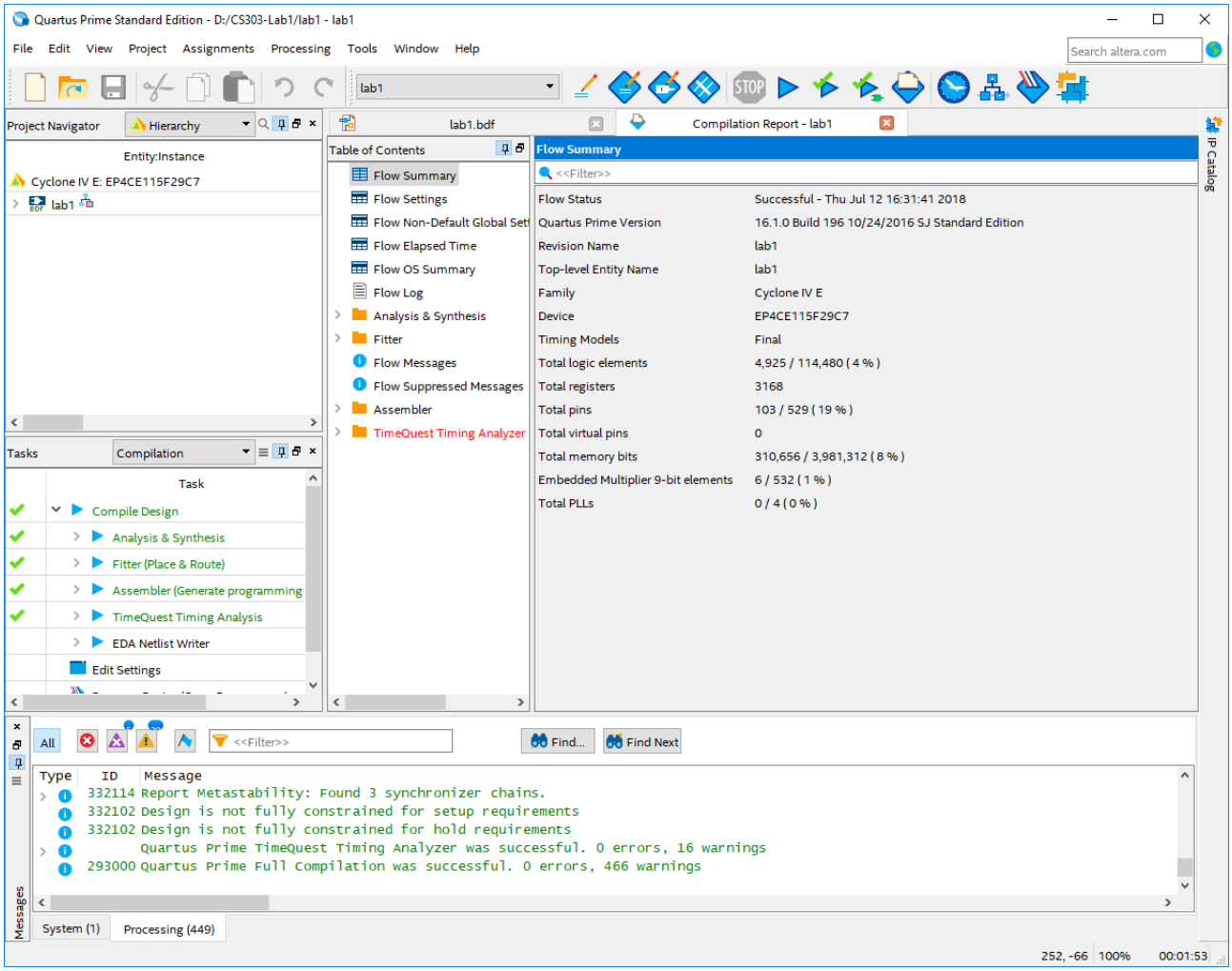

Figure 1-37: Successful compilation of the hardware design

1.4 Download the lab1 digital design to the FPGA chip

Connect the DE2 board with power, and Blaster port to the workstation via USB cable, and select "**Programmer**" from the "**Tools**" menu to open up the programmer as shown in Figure 1-38. Click the "**Hardware Setup**" button to make sure the current selected hardware is **USB Blaster** as shown in Figure 1-39. If not, add one up through "**Add Hardware**" button beside. By clicking the "Add File" or "Change File" button at the left to include "lab1.sof" and tick the box under the "Program/Configure". Click "**Start**" below the "**Hardware Setup**" to download the hardware design to the DE2 board. You will find the progress bar to be 100% when the process is completed. Now you are ready to start the software implementation of the Nios II system.

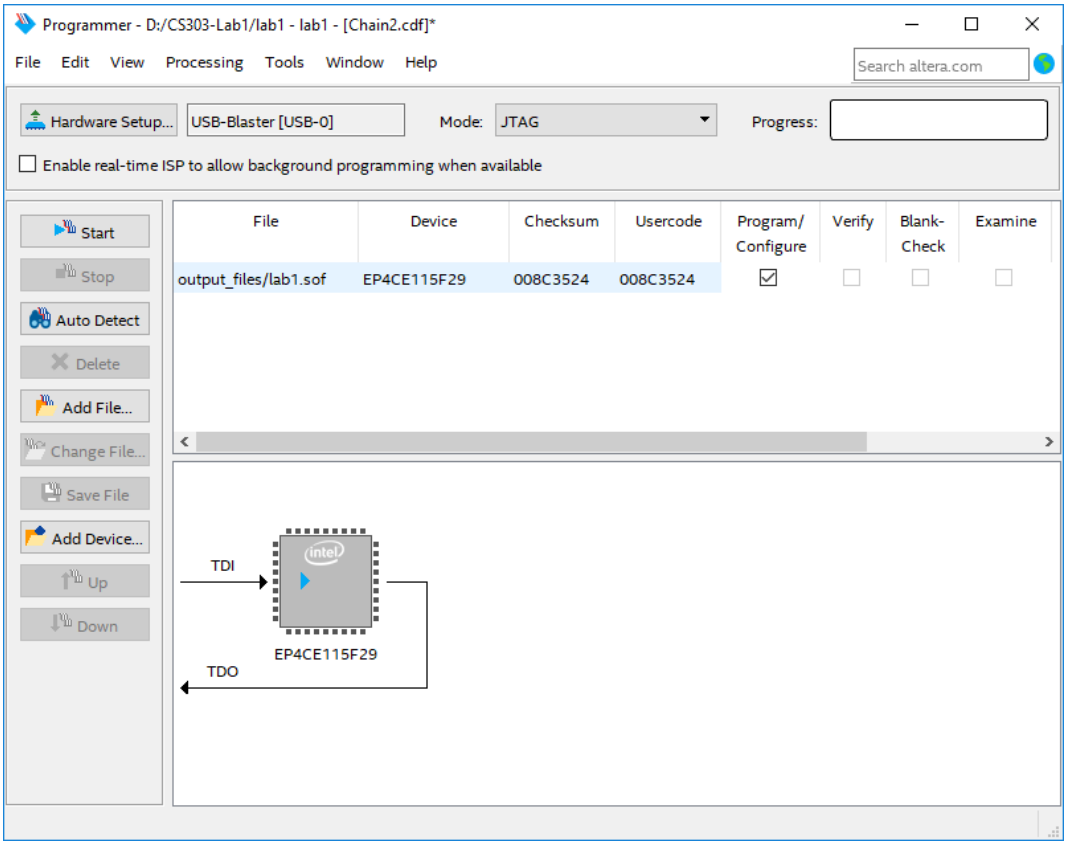

Figure 1-38: Programmer of Quartus II

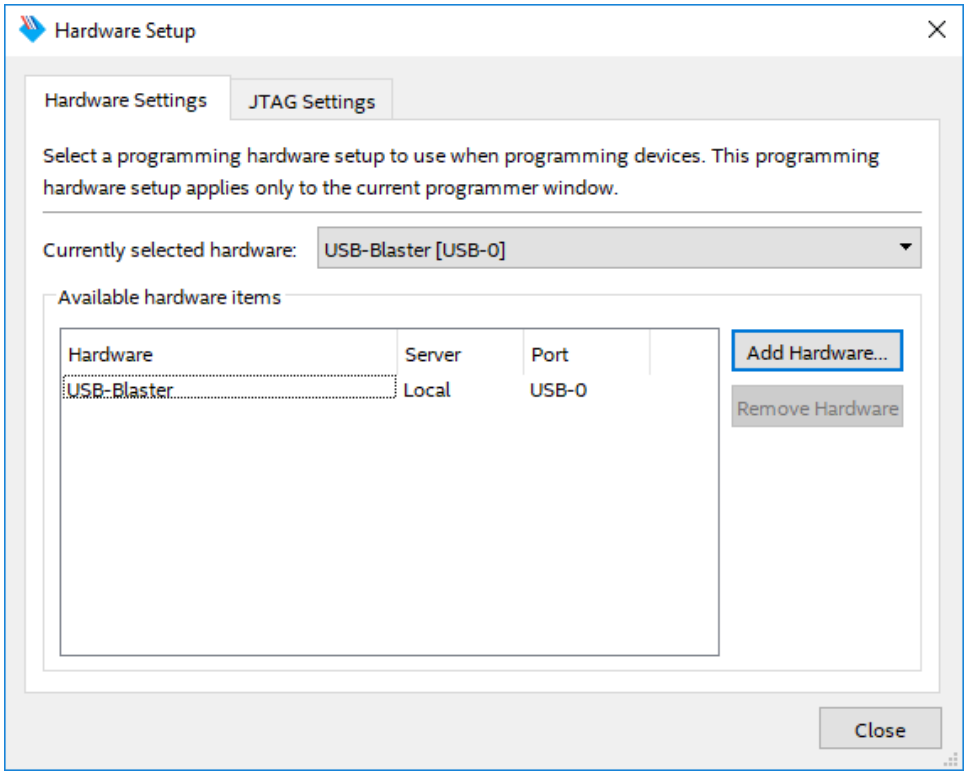

Figure 1-39: Hardware settings of the download cable

2. Software implementation on the DE2 with dedicated hardware design

2.1 Developing software with Nios II IDE

Start "Nios II 16.1 Software Build Tools for Eclipse", the main IDE window as illustrated in Figure 2-1.

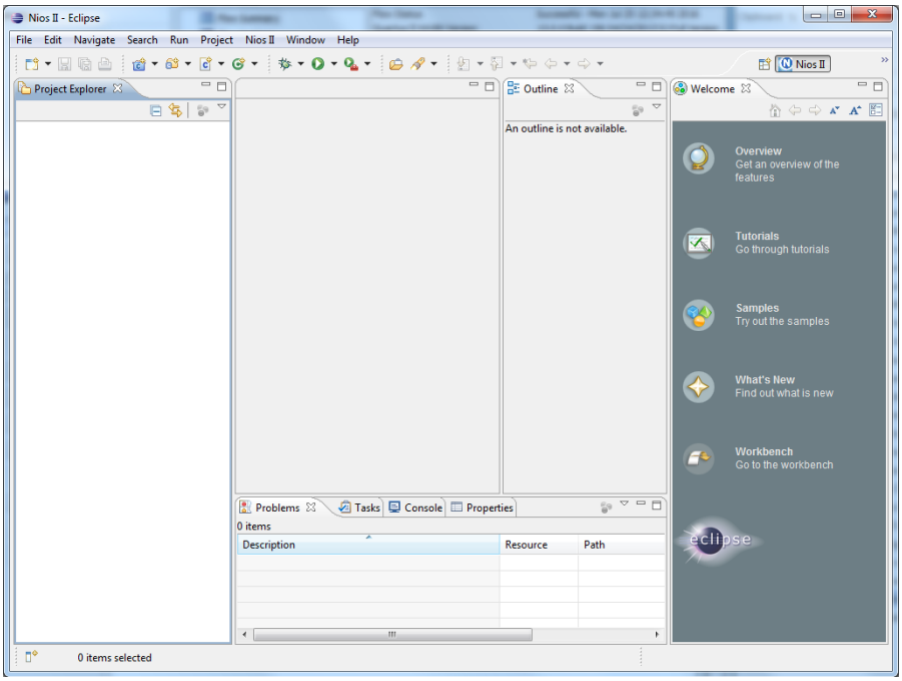

Figure 2-1 Nios II IDE

2.2 Construct an application

Software managed by the Nios II IDE are stored in a top level hierarchy called **workspace**. Generally application software created based on the same hardware configurations are stored in the same workspace for simpler management. A workspace can be switched after you've started from the File menu (**File->Switch Workspace...**). Browse and create a directory (named **workspace**) under the current project directory , therefore the directory path will be "**D:\CS303-Lab1\workspace**" in this case as detailed in Figure 2-2.

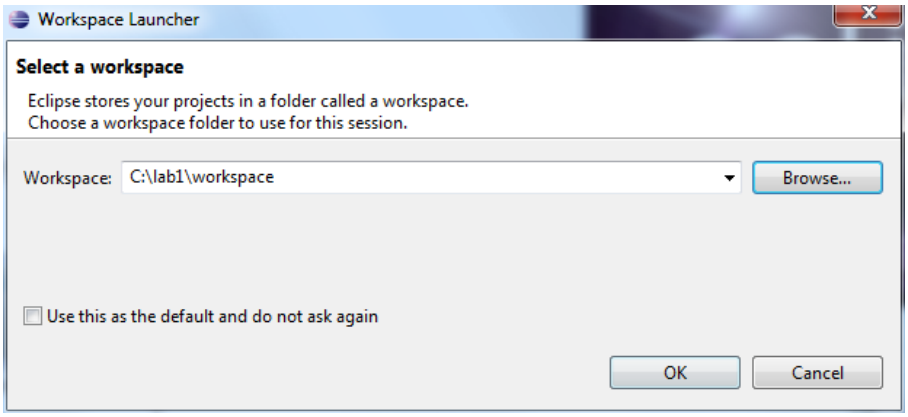

Figure 2-2: Switch workspace to just created directory

To create an application, select "**File->New->Nios II Application and BSP from Template"** from the main menu. For "**SOPC Information File Name**" click and browse to "nios2_system.sopcinfo" file under lab1. This is the SOPC you have created earlier in this lab. You will notice under "CPU name" field you should have the processor instance **nios2_qsys_0** selected automatically (this will reflect whatever you entered for the CPU name in Qsys.

Next type in the project name, which is the application name which will be **hello_lab1** in this tutorial. Untick the **default location** and select the directory to store our software (**D:\CS303-Lab1\software** in this case). Choose "Hello World" for the project template as shown in Figure 2-3.

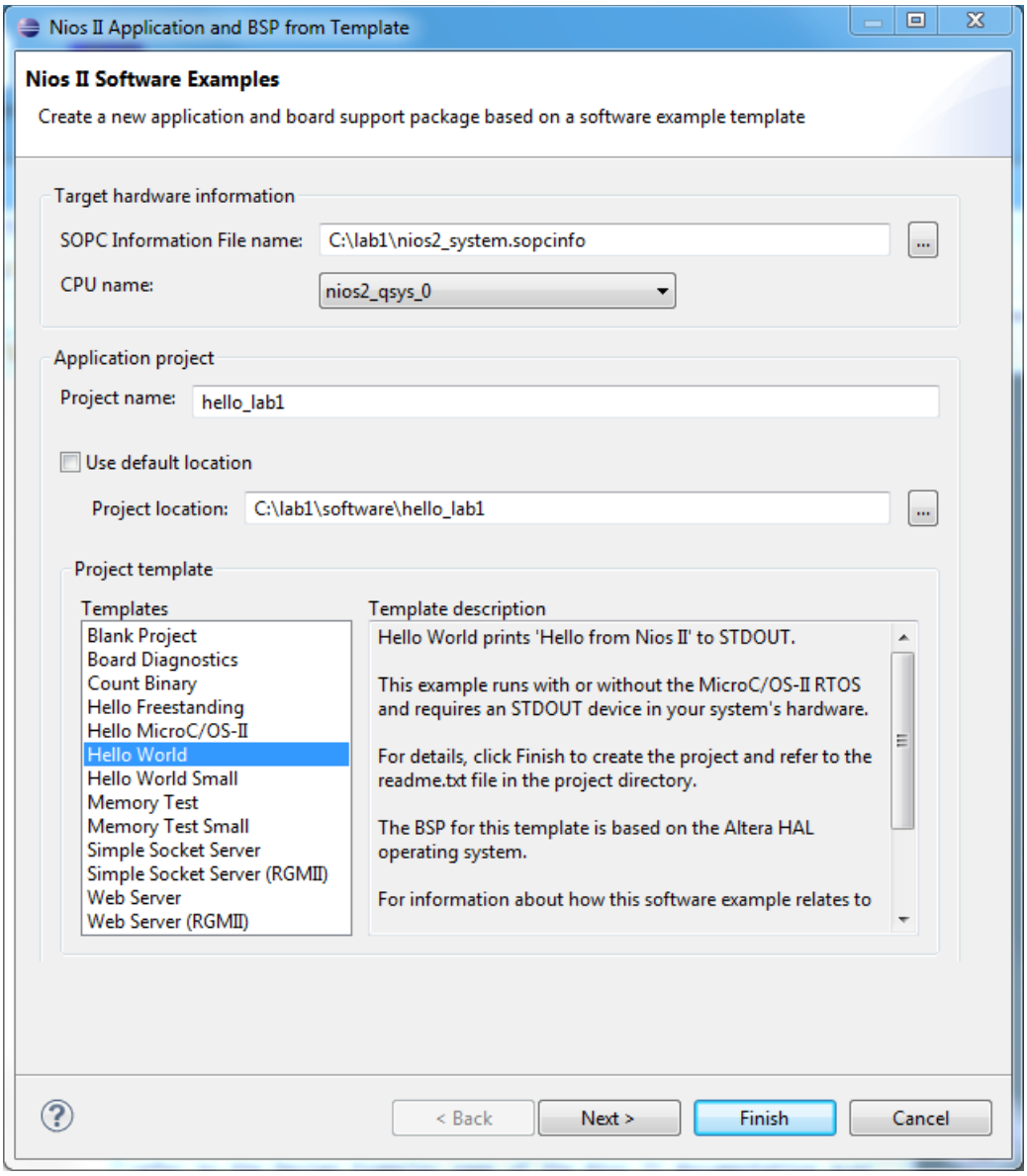

Figure 2-3: Creating Hello World example

Click “Next”. You will see Figure 2-4, here IDE generate system libraries. Used default settings and simply click “Finish”.

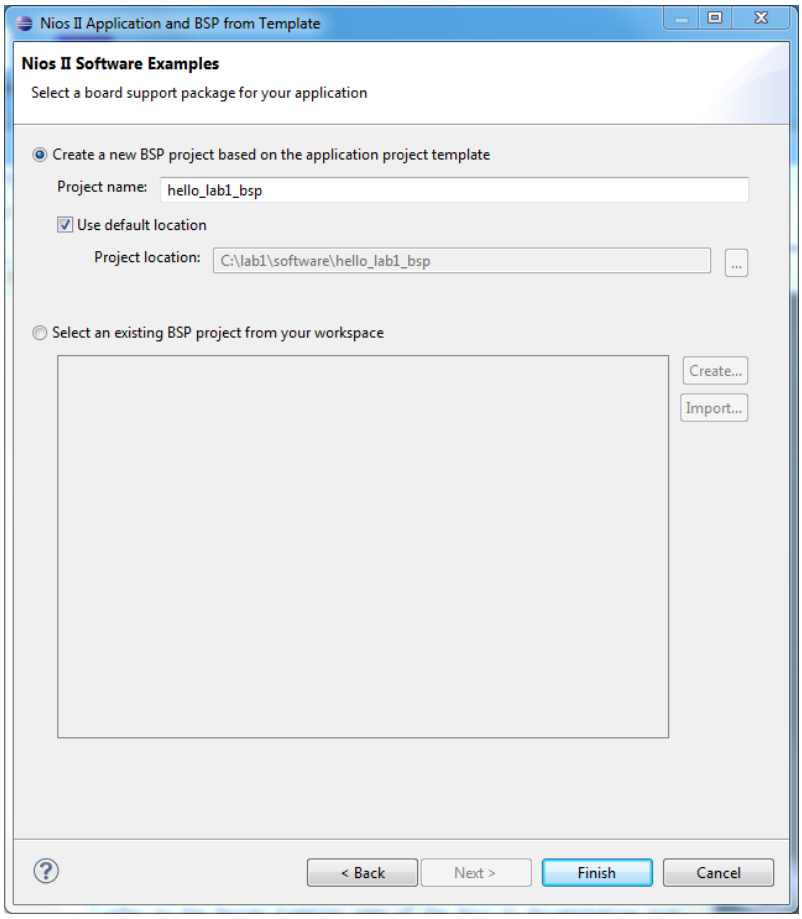

Figure 2-4: Creating libraries for Hello World example

You will see two projects in your project explorer. One is “**hello-lab1**” other is “**hello-lab1_bsp**”. “hello-lab1” contains your application (“Hello World” example) and “hello-lab1_bsp” contains information about the hardware.

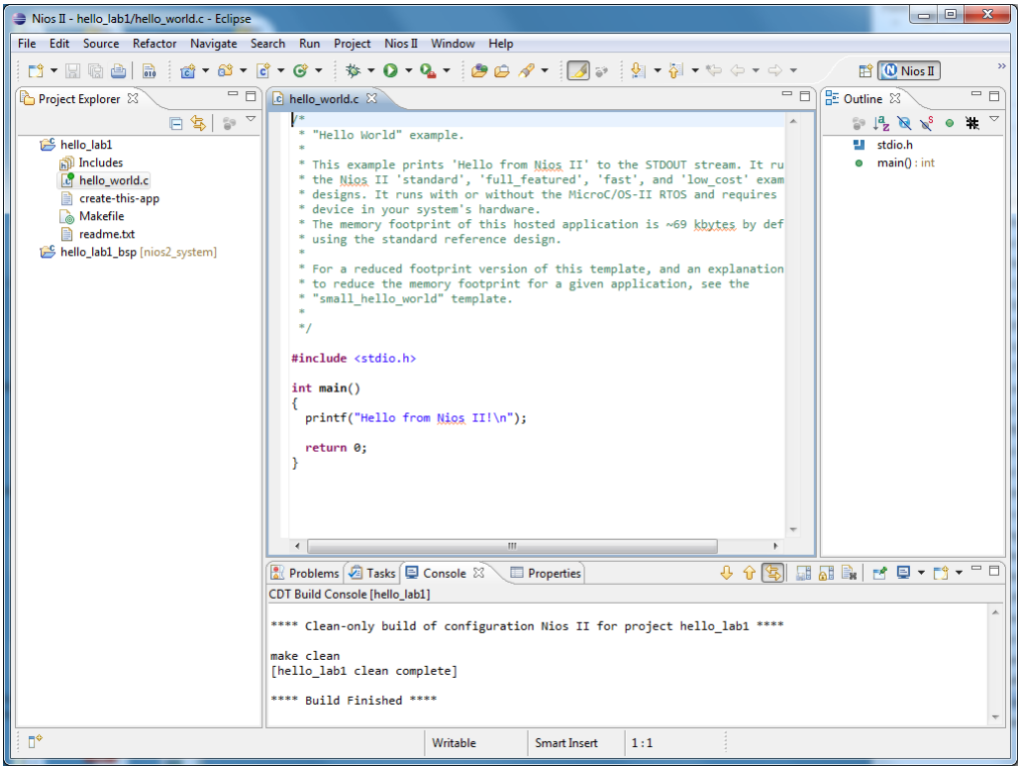

Figure 2-5: Hello World project files

2.3 Access modules added in the SOPC Builder

Refer to Figure 1-26 or section 1.2.15 in the section 1 (**Note: base addresses of your NIOS II system may differ from this tutorial, please refer to your design**), which is the list of modules in the Nios II system. The base address can be treated as identity of each module, therefore the control of each modules can be achieved by providing corresponding base address to the function calls. There is also a way to access the peripherals without directly typing up their address for portability, which is using their s**ymbolic names**. Symbolic names can be found in the **system.h**, which will be generated after building the system library. Right click on "hello-lab1_bsp" project and select "Build project" as shown in Figure 2-6.

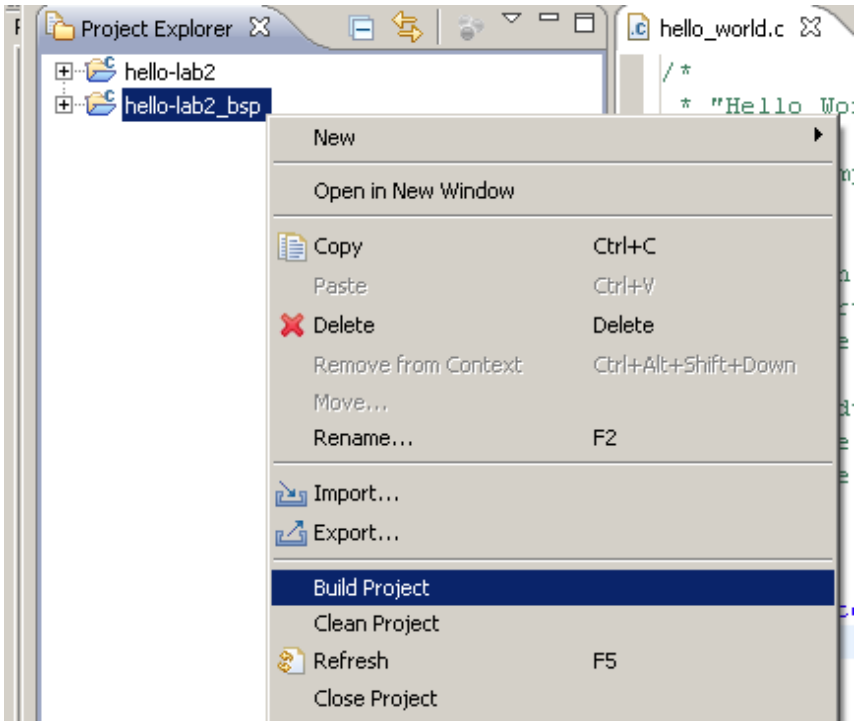

Figure 2-6: Start building bsp library

For this lab we are interested in "**system.h**" and the **drivers** folder, highlighted in Figure 2-7. Double click to open it up "system.h". Entries are in **#define** fashion, for instance, the base address of push buttons is 0x019120600 (might be different for your case, but it does not matter), and is assigned with a symbolic name as "**KEYS_BASE**". You can use KEYS_BASE in the application code instead of hard-coded the based address to increase the portability of the program from one's configuration to another (Note that the names of modules are the same but the address assigned are different). The 'drivers" folder contain the C functions for the hardware peripherals. This is how your C application will communicate to the hardware. If you open the "inc" folder under "drivers", you will see all the interfaces to your peripherals such as timers, LED, LCD, PIO, and lots more.

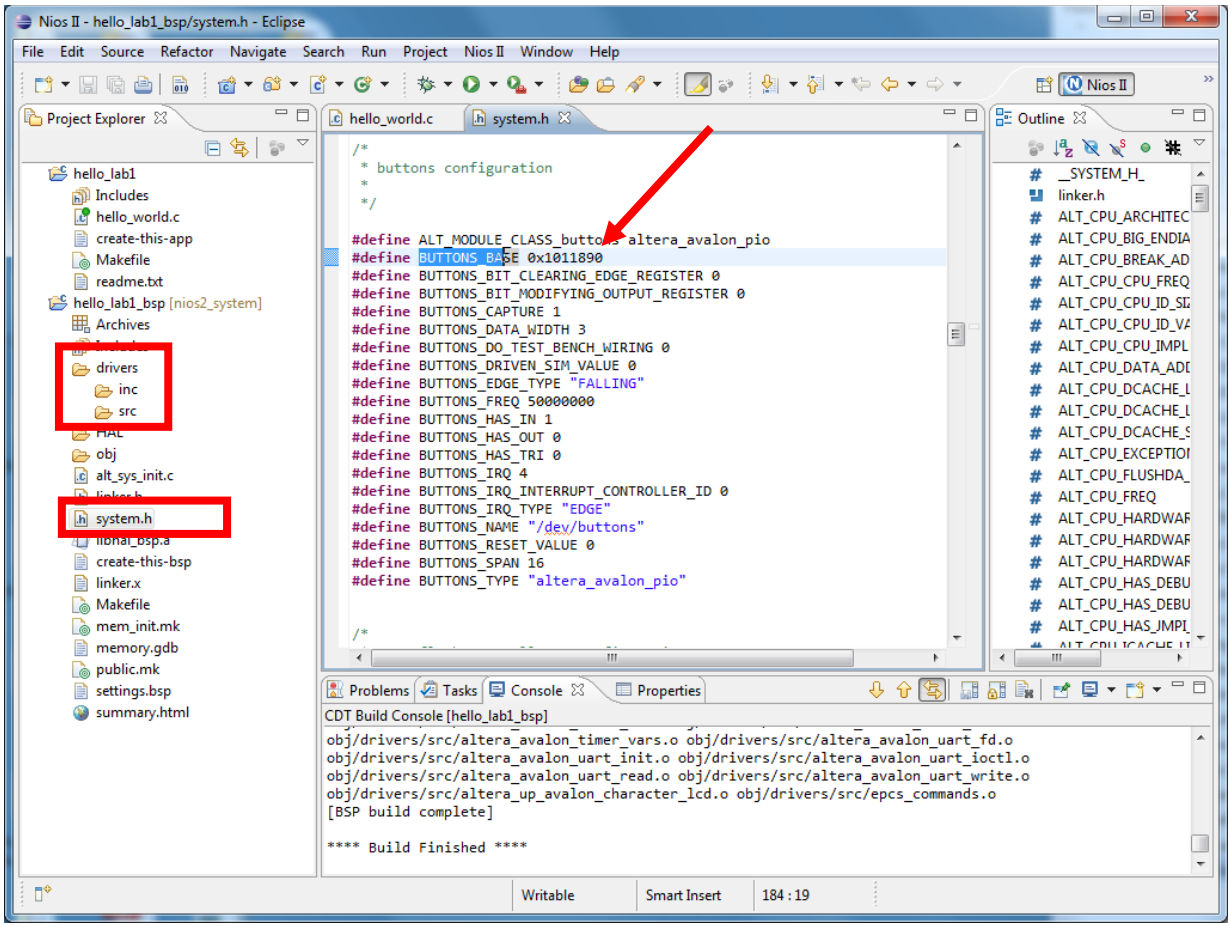

Figure 2-7: System library

In order to use symbolic name, add **#include <system.h>** in your application code (in this case, **hello_world.c** within the hello_lab1 project). Function to write values to the PIO registers are in altera_avalon_pio_regs.h, therefore including them too by adding **#include <altera_avalon_pio_regs.h>** as well.

Before using the peripherals on the DE2 board, we can roughly divided them into three groups according to the way of controlling them with function calls:

1. Output only PIOs – such as green and red LEDs, and 8 seven-segments displays. Generally writing values to these PIOs is the only operation tended to be done. **IOWR_ALTERA_AVALON_PIO_DATA(BASE_NAME, value)** is called to achieve that.
2. Input only PIOs – such as 18 switches and 3 push buttons. Values of them are read through assigning **IORD_ALTERA_AVALON_PIO_DATA(BASE_NAME)** to a variable (most of time unsigned int or unsigned long, depending on the architecture).
3. Character peripherals – such as character LCD and the UART port. File descriptor (or symbolic name) is used to initialize (through **fopen**) the use of the peripherals. **fprintf** will be used for output, more details will be introduced later.

The following code (you can save them to the hello_world.c to see what this piece of codes does) with line numbers illustrate an example of programming Nios II to control peripherals on the DE2 board.

The descriptions of the code are as follows:

1. Line 1 is to enable the uses of the symbolic names such as for the substitution of the base addresses.
2. Line 2 is to provide PIO register writing and reading operations
3. Line 3 is included for simple print out operations such as **printf**
4. Line 7 to line 9 introduces variables to store values read from PIOs
5. Line 10 declares a file descriptor which represents the character LCD
6. Line 12 uses the printf provided by the header file in line 3 to print **Hello from Nios II** to the stdout defined in the system library
7. Line 14 write the hex value of 0xaa (10101010 in binary) to the green LEDs, whereas the bit value 1 will light the LED, and 0 will dim it. **LED_GREEN_BASE** is the symbolic (substituted) name for the base address of the green LEDs, which is located in the system.h included at Line 1.
8. Line 16 uses **fopen** provided in **stdio.h** to initialize the pointer to the character LCD. The first argument is the symbolic name of the LCD described in system.h. The second parameter is attribute/usage of the file descriptor, whereas "**w**" indicates writing to the device. For other modes please refer to the software development manual of the Nios II processor.
9. Line 17 makes the program enter an infinite loop for the no-stop running behavior of this application. Note that sometimes **for(;;)** is encouraged to be used for generating better codes by the compiler.
10. Line 19 reads the value of the 18 switches (which will be a 18 bits value, stored in a 32 bits wide integer, where the higher 14 bits will be stuffed with zero) through calling **IORD_ALTERA_AVALON_PIO_DATA**, provided by the **altera_avalon_pio_regs.h**. **SWITCHES_BASE** is the symbolic name (in system.h) of the switches.
11. Line 20 writes the value of the 18 switches to the red LEDs through **IOWR_ALTERA_AVALON_PIO_DATA**. Line 22 and 23 just keep a copy of the button values before reading in new ones.
12. Line 24 checks if the character LCD was open successfully before using it.
13. Line 26 only updates the LCD display when the button values change, preventing unnecessary

LCD refresh.

14. Line 30 and 31 shows **fprintf** is used to write the characters/string to the LCD. The defined name **ESC** and **CLEAR_LCD_STRING** are commands to the LCD for clear the display.
15. Line 36 illustrates a **fclose** to the LCD is required once finish using the LCD.
16. Line 38 terminates the program

```
1   #include <system.h>
2   #include <altera_avalon_pio_regs.h>
3   #include <stdio.h>
4
5   int main()
6   {
7         unsigned int uiSwitchValue = 0;
8         unsigned int uiButtonsValue = 0;
9         unsigned int uiButtonsValuePrevious = 0;
10        FILE *lcd;
11
12        printf("Hello from Nios II!\n");
13
14        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_GREEN_BASE, 0xaa);
15
16        lcd = fopen(LCD_NAME, "w");
17        while(1)
18        {
19              uiSwitchValue = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);
20              IOWR_ALTERA_AVALON_PIO_DATA(LEDS_RED_BASE, uiSwitchValue);
21
22              uiButtonsValuePrevious = uiButtonsValue;
23              uiButtonsValue = IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE);
24              if(lcd != NULL)
25              {
26                    if(uiButtonsValuePrevious != uiButtonsValue)
27                    {
28                          #define ESC 27
29                          #define CLEAR_LCD_STRING "[2J"
30                          fprintf(lcd, "%c%s", ESC, CLEAR_LCD_STRING);
31                          fprintf(lcd, "BUTTON VALUE: %d\n", uiButtonsValue);
32                    }
33              }
34        }
35
36        fclose(lcd);
37
38        return 0;
39  }
```

Now you can right click on the **hello_lab1** project and select "**Build Project**" to start compiling the application. Then, to run the application, right click on the project under "Run As" select "Nios II Hardware" as shown in Figure 2-8.

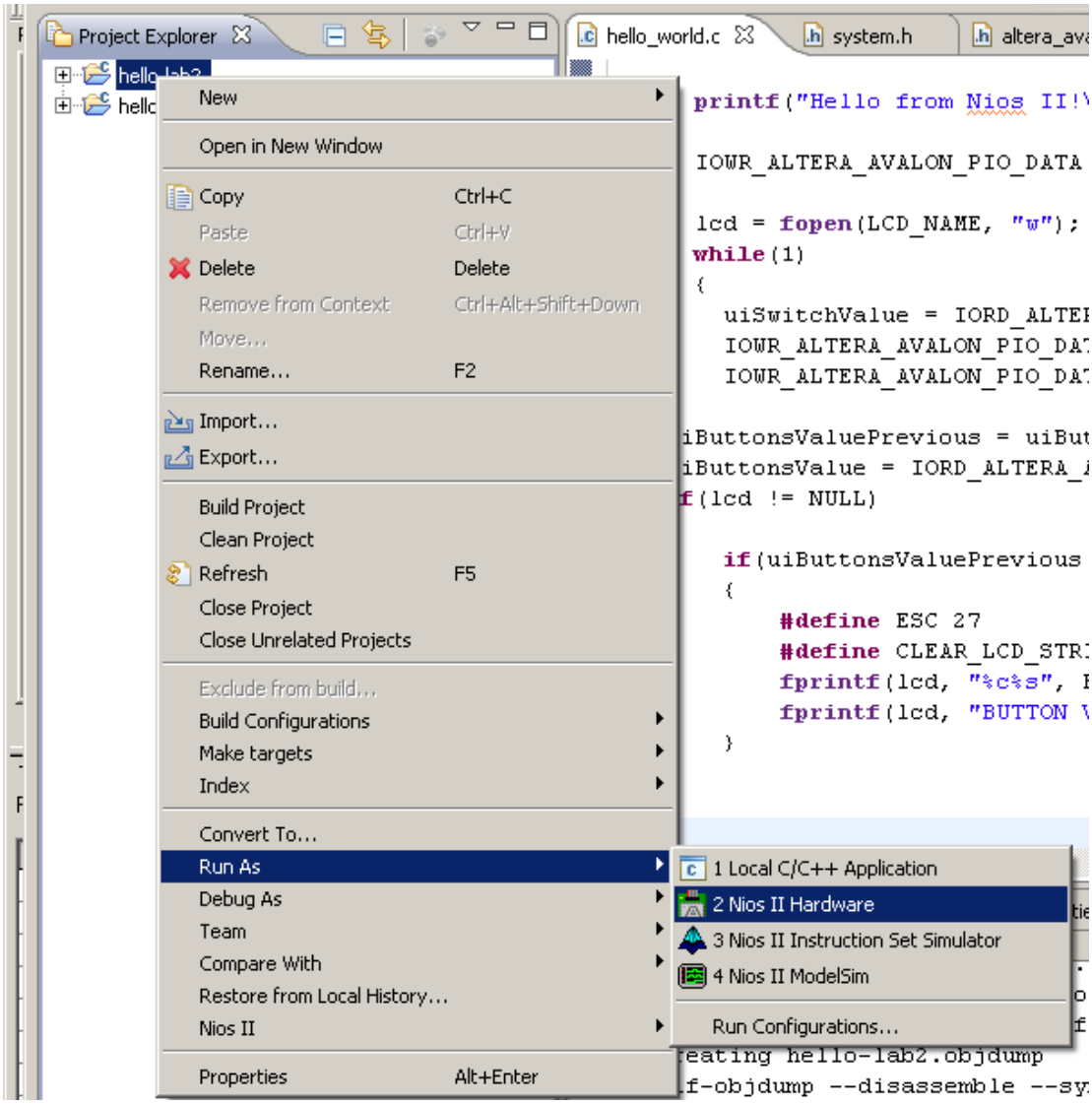

Figure 2-8: Running on the Nios II hardware

You should have the following settings, first check the project tab then the “Target Connection” as shown in the Figure 2-9.

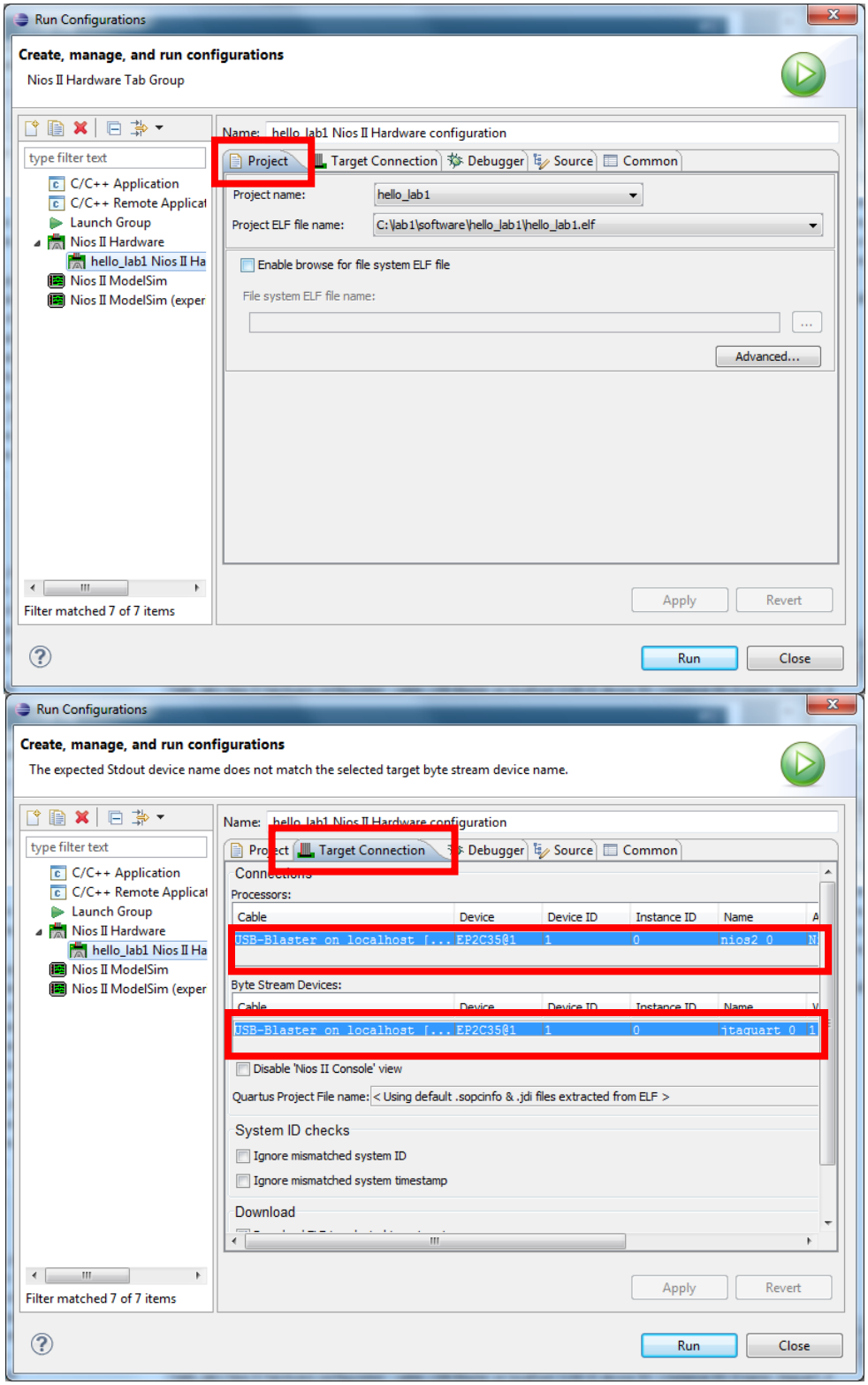

Figure 2-9: Setting up the run configuration

Now click "**Run**" on the lower right corner. Sometimes a compilation of the project might be carried out again if the source code is changed. You should soon be able to see "Hello from Nios II" from the console in the IDE as in Figure 2-10. You will also notice alternate green LED's are lit as a result of line 14 for your source code.

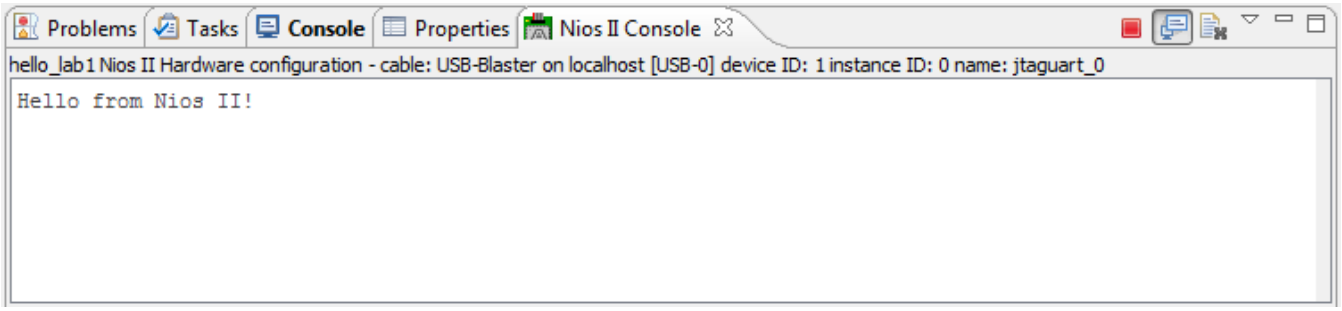

Figure 2-10: The console output in the Nios II IDE

END of the Lab