Create a new Quartus II project for your system. As shown in Figure 1, we stored our project in a directory named sopc_builder_tutorial and named both the project and its top-level design entity lights. You can choose a different directory or project name, but note that SOPC Builder does not allow spaces in file names (e.g., SOPC Builder tutorial will cause an error).

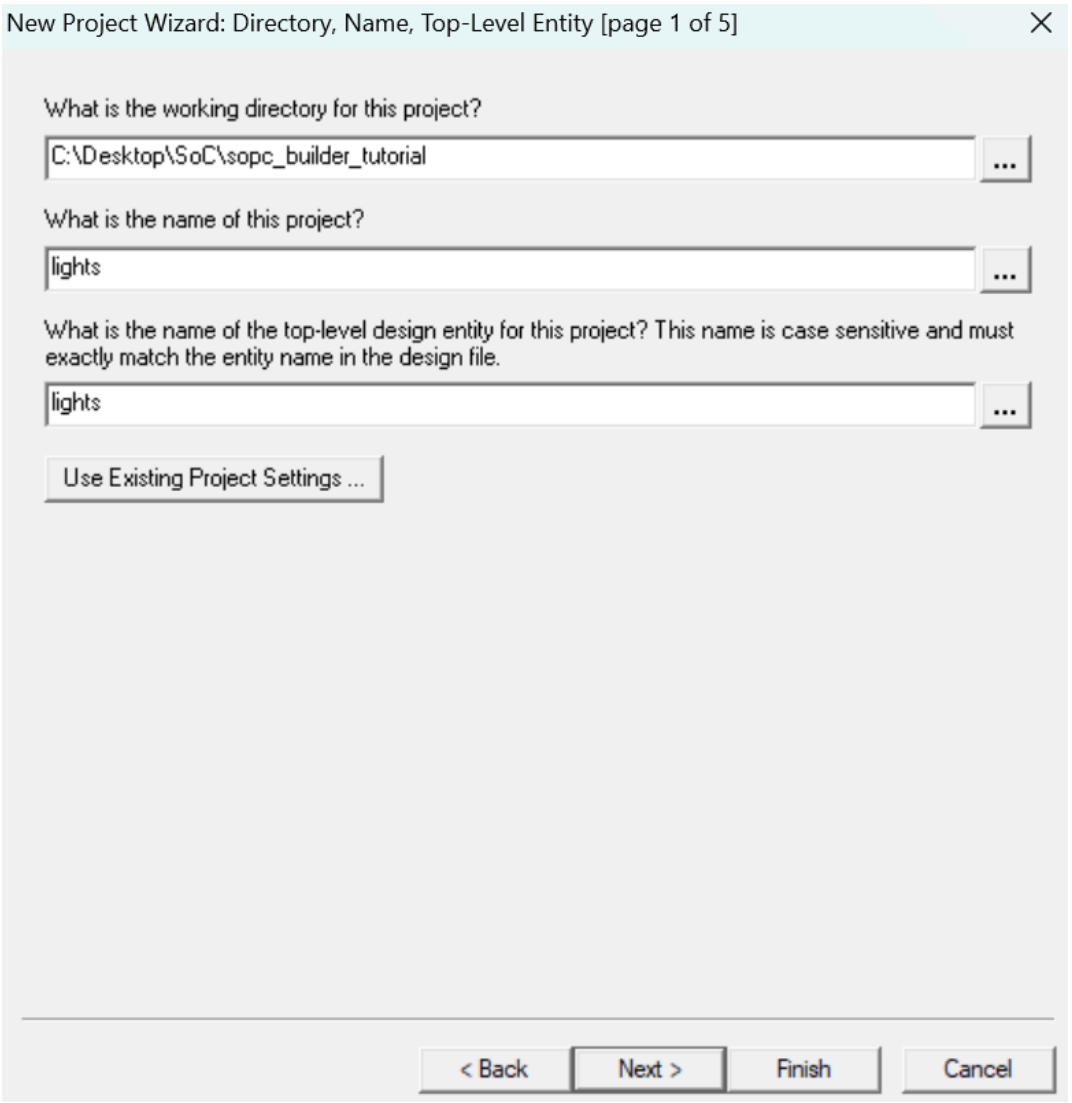


Figure 1. Create a new project.

In your project, we use board DDE:
Select Family → Cyclone II
Available device: EP2C35F672C6

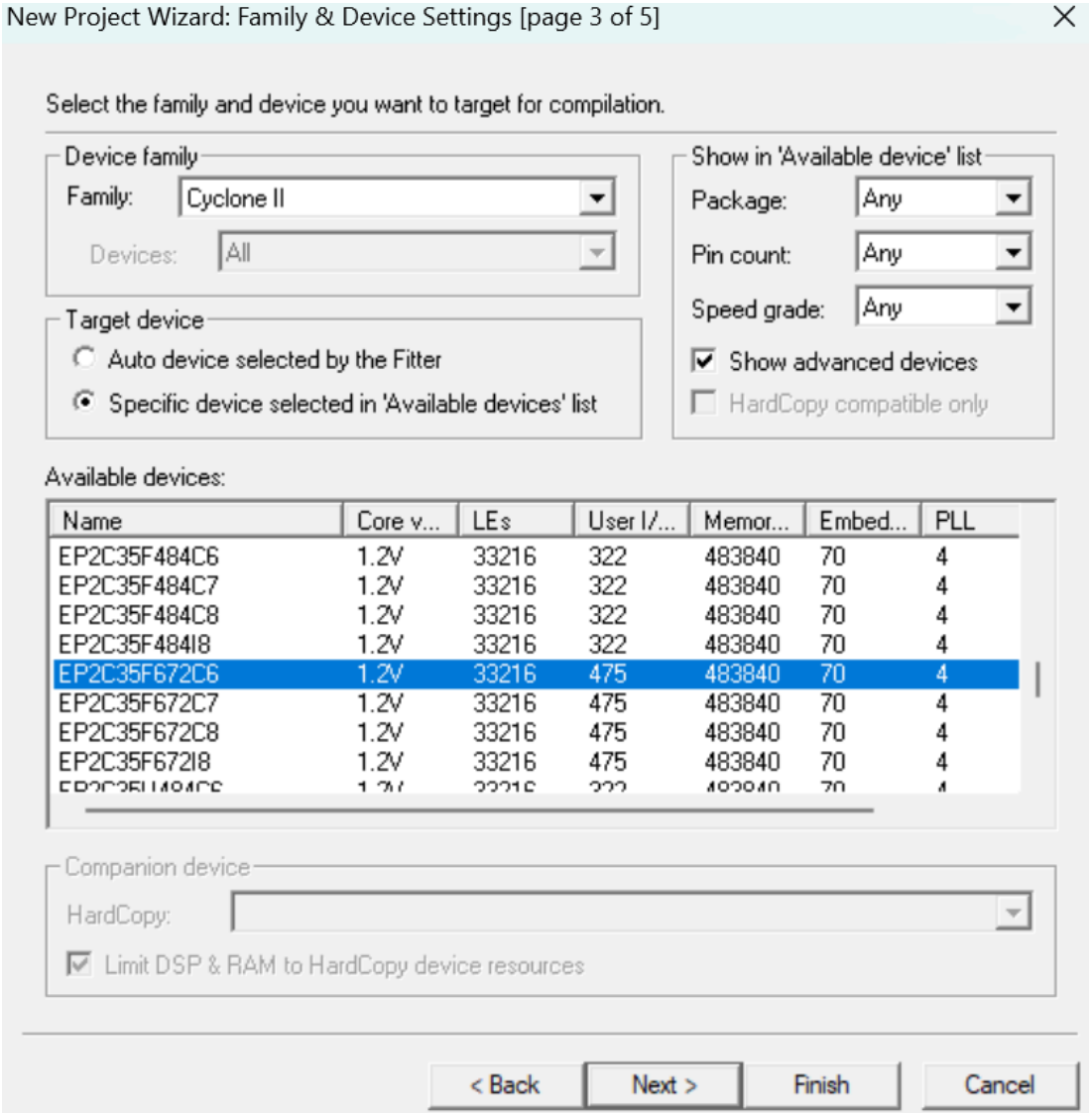


Figure 2. Setting DE-series FPGA devices names.

Select Tools > SOPC Builder to open the pop-up box in Figure 3. Enter nios_system as the system name. Choose Verilog as the target HDL for the system module. Click OK to proceed to the window in Figure 4. If Quartus recommends using Qsys, click OK to continue.

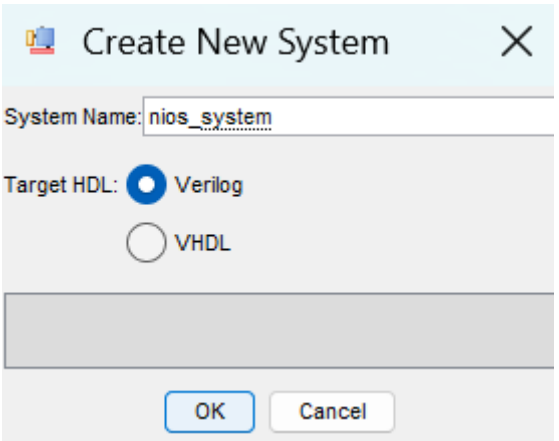


Figure 3. Create a new Nios II system.

Check the setting for the Device Family and ensure that the correct family is Cyclone II. The Nios II processor operates with a clock. In this tutorial, use the 50-MHz clock provided on the DE-series board. In the SOPC Builder display (Figure 5), you can specify clock signal details. If not already listed, add a clock named clk_0, set its source as External, and configure the frequency to 50.0 MHz.

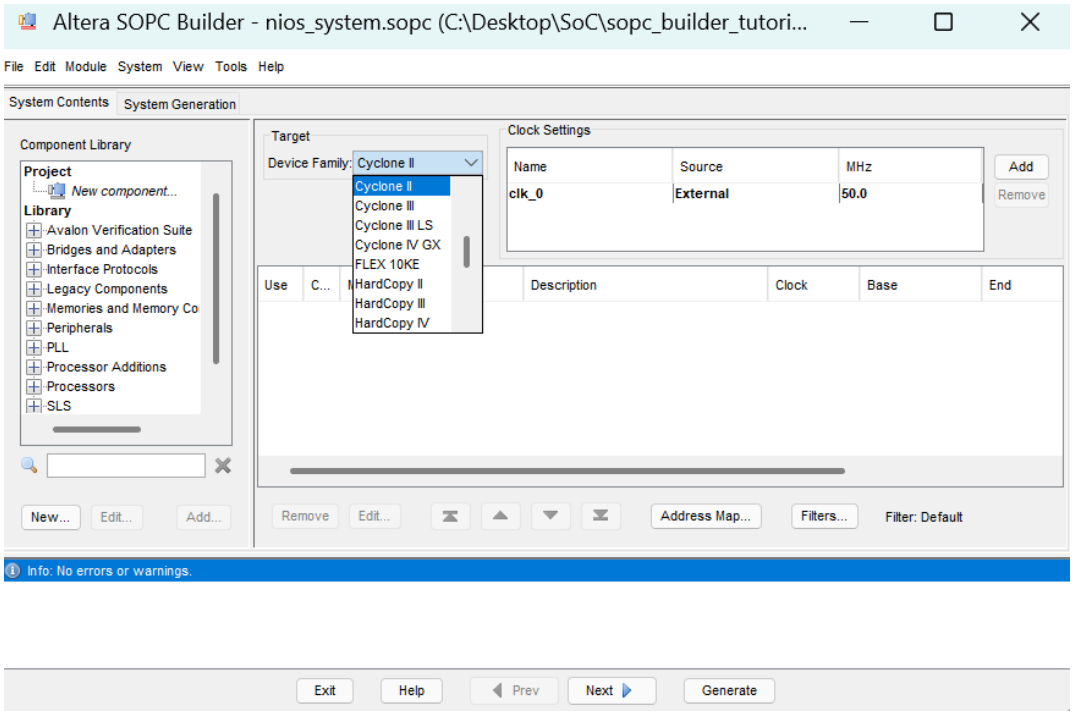


Figure 4. The System Contents tab window on a DE-series board.

On the left side of the window, expand Processors, select Nios II Processor and click Add. Select Nios II/e, the simplest processor version. Click Finish to return to the window in Figure 4, which now displays the Nios II processor as shown in Figure75.

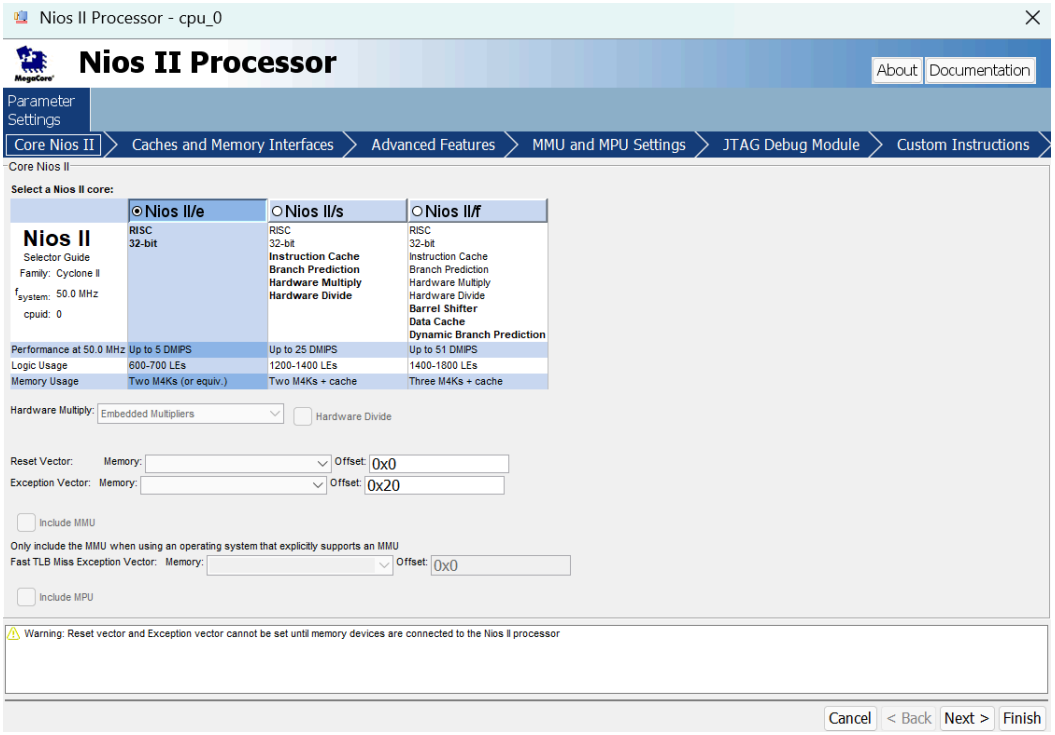


Figure 5. Create a Nios II processor.

There may be some warnings or error messages displayed in the SOPC Builder Messages window (at the bottom of the screen), because some parameters have not yet been specified. Ignore these messages as we will provide the necessary data later.

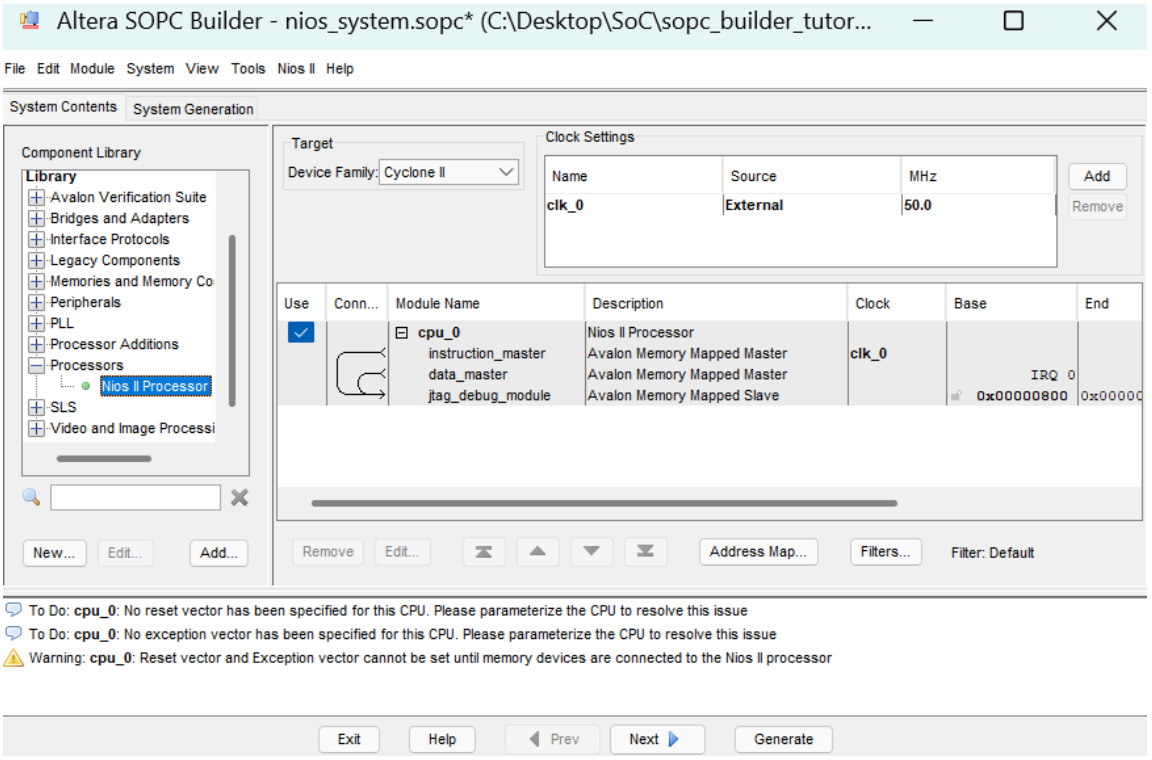


Figure 6. The defined processor on a DE-series board.

To specify the on-chip memory, follow these steps:

- Select Memories and Memory Controllers > On-Chip > On-Chip Memory (RAM or ROM) and click Add.
- In the On-Chip Memory Configuration Wizard (Figure 7), set the Data width to 32 bits and the Total Memory Size to 4 Kbytes (4096 bytes).
- Keep the other default settings unchanged.
- Click Finish to return to the System Contents tab as shown in Figure 8.

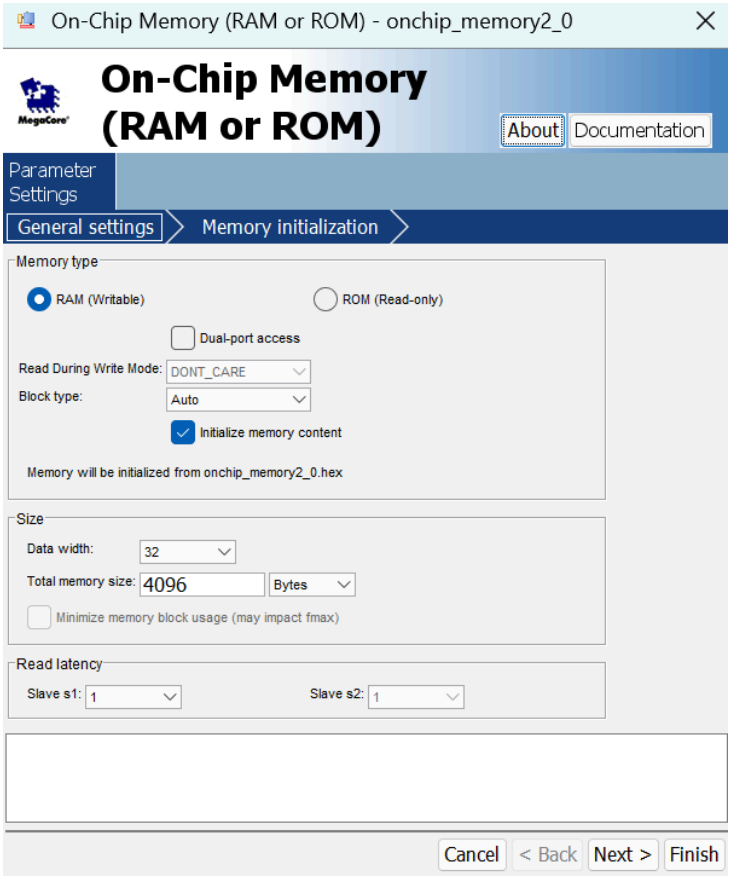


Figure 7. Define the on-chip memory

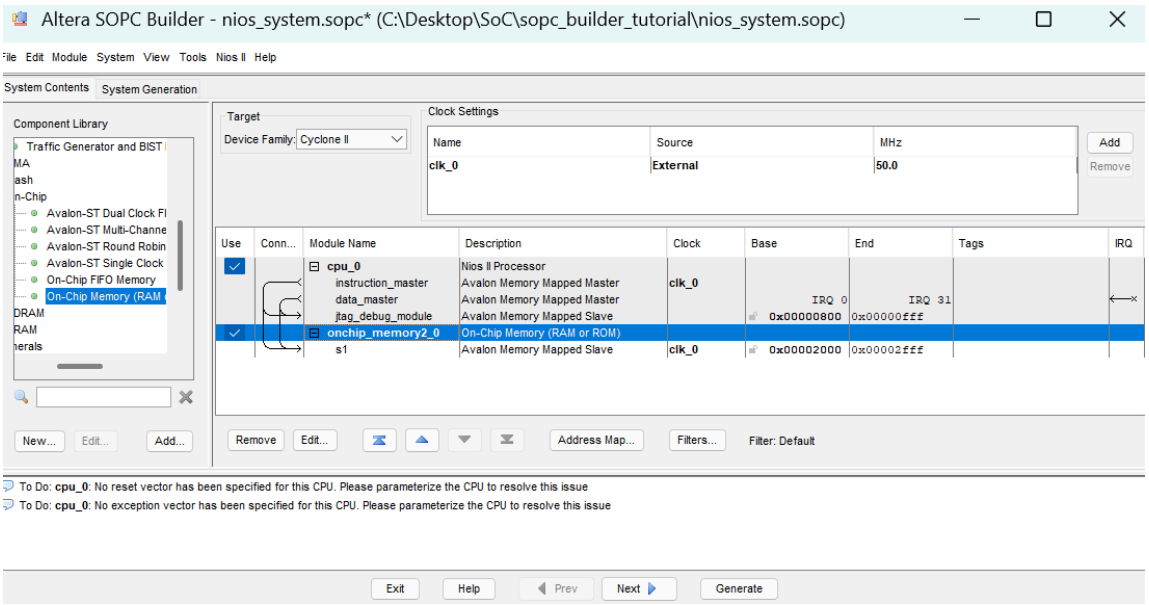


Figure 8. The on-chip memory included on a DE-series board.

To specify the input parallel I/O interface:

- Select Peripherals > Microcontroller Peripherals > PIO (Parallel I/O) and click Add to open the PIO Configuration Wizard (Figure 9).

- Set the port width to 8 bits and the direction to Input, as shown. For a DE0-Nano board, set the width to 4 bits.
- Click Finish to return to the System Contents tab (Figure 10).

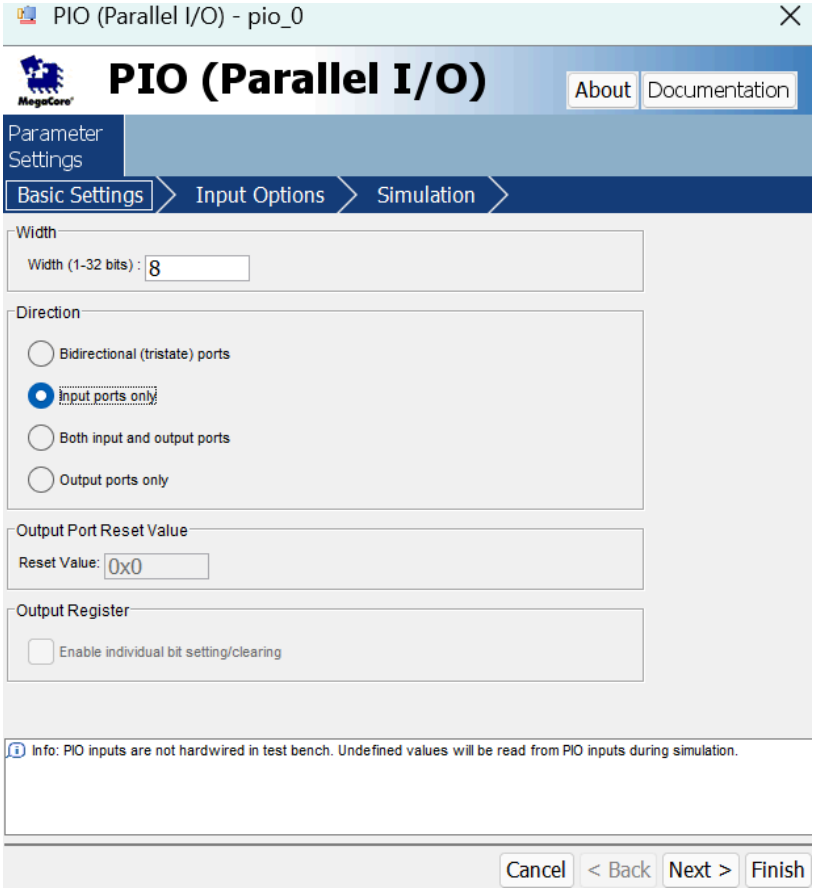


Figure 9. Define a parallel input interface.

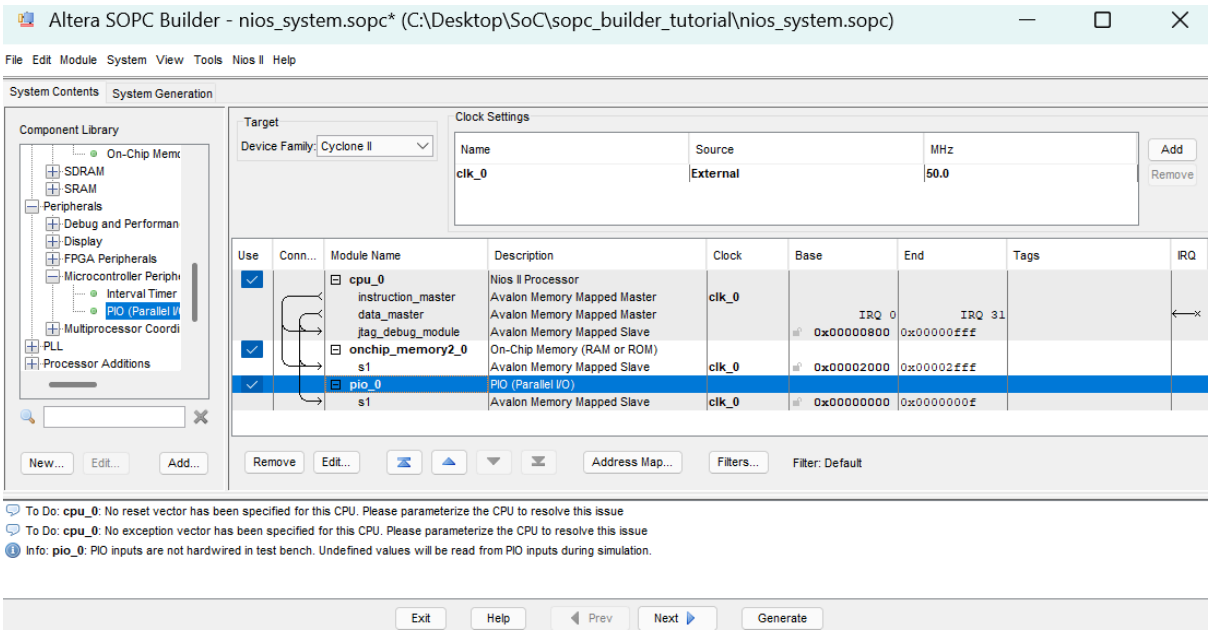


Figure 10. The parallel input interface is included on a DE-series board.

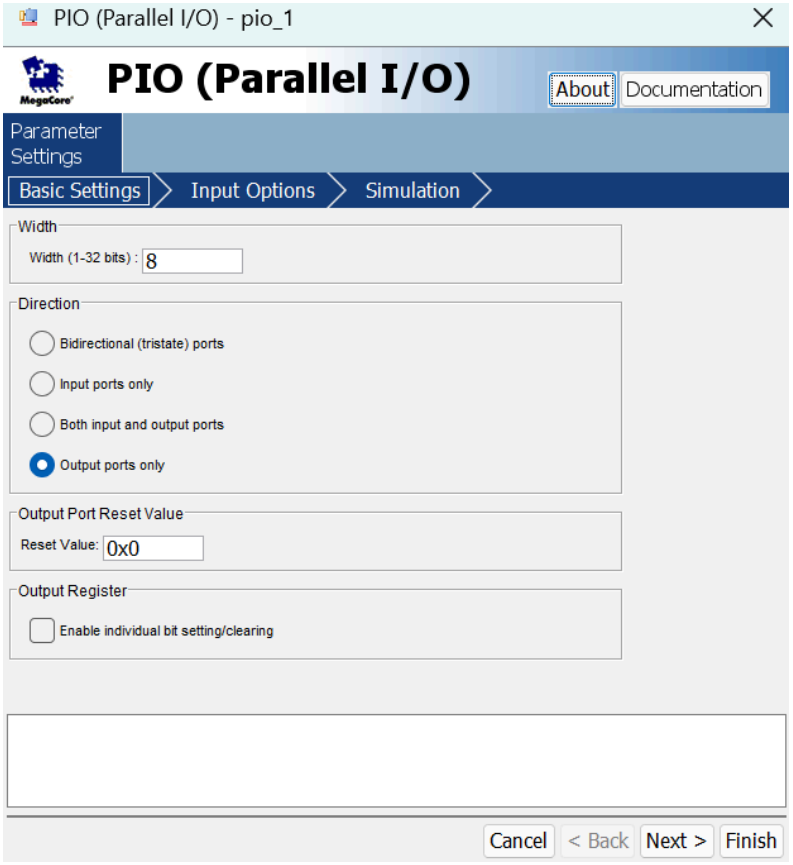


Figure 11. Define a parallel output interface.

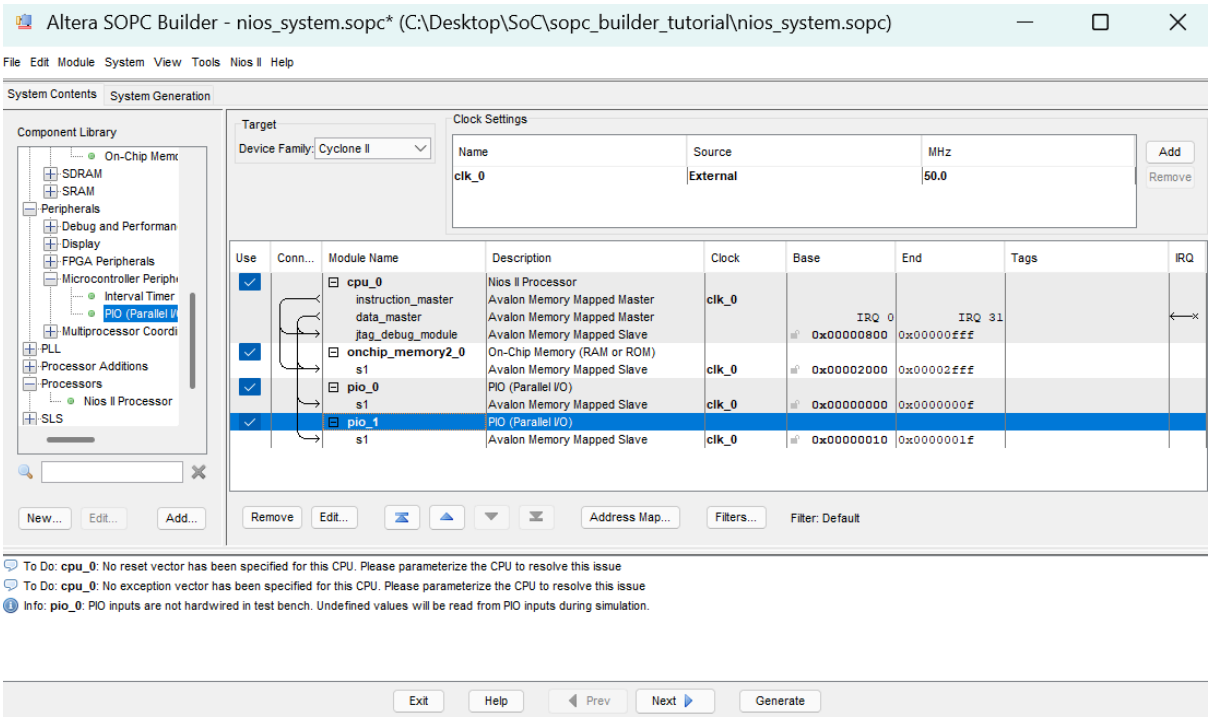


Figure 12. The parallel output interface included on a DE-series board.

We wish to connect to a host computer and provide a means for communication between the Nios II system and the host computer. This can be accomplished by instantiating the JTAG UART interface as follows:
• Select Interface Protocols > Serial > JTAG UART and click Add to reach the JTAG UART Configuration Wizard in Figure 13.
• Do not change the default settings
• Click Finish to return to the System Contents tab

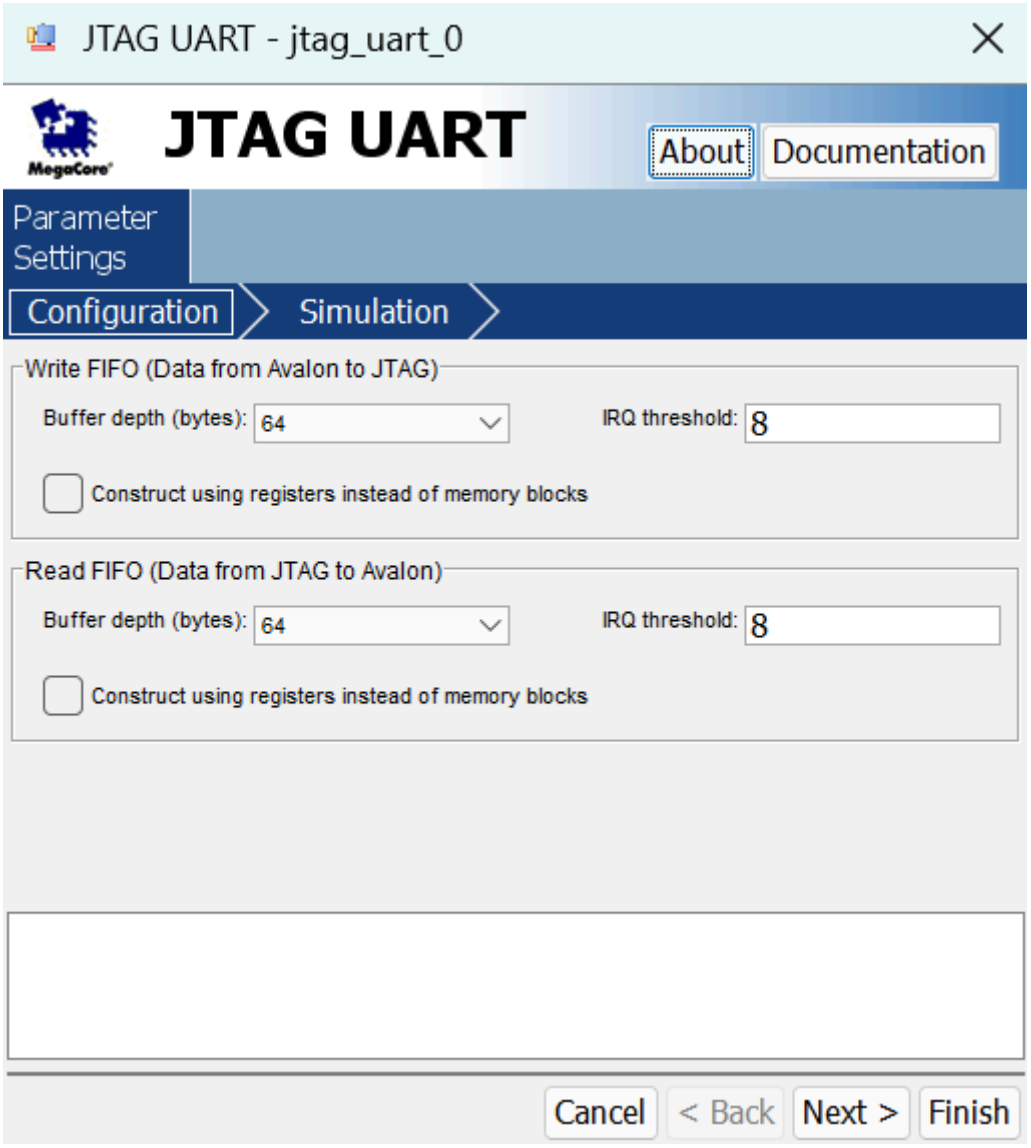


Figure 13. Define the JTAG UART interface.

The complete system is shown in Figure 14. SOPC Builder automatically assigns names to components, which may not be descriptive enough for the target design but can be changed. In Figure 2, the parallel input and output interfaces are named *Switches* and *LEDs*, respectively, and these names can be used in the system. Right-click on *pio_0*, select Rename, and change it to *Switches*. Similarly, rename *pio_1* to *LEDs*.
The base and end addresses of system components can be set manually or automatically by SOPC Builder. Choose the automatic option by selecting System > Assign Base Addresses from the SOPC Builder menu, resulting in the assignment shown in Figure 15.

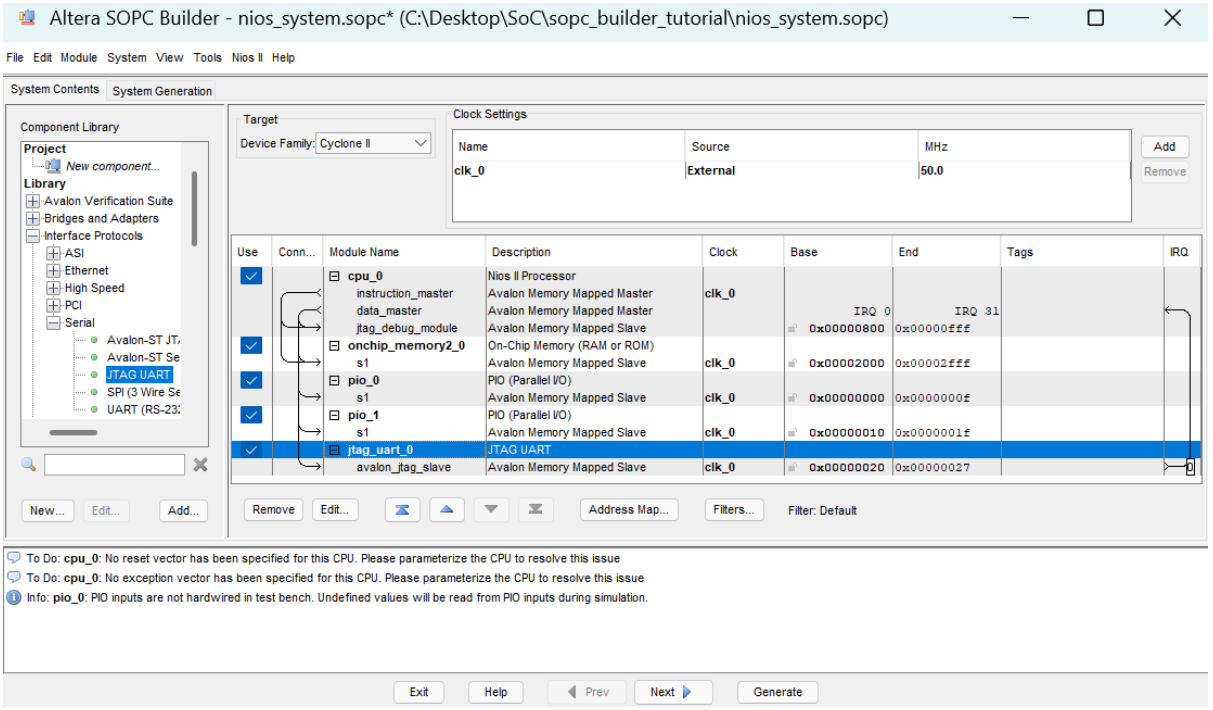


Figure 14. The complete system on a DE-series board.

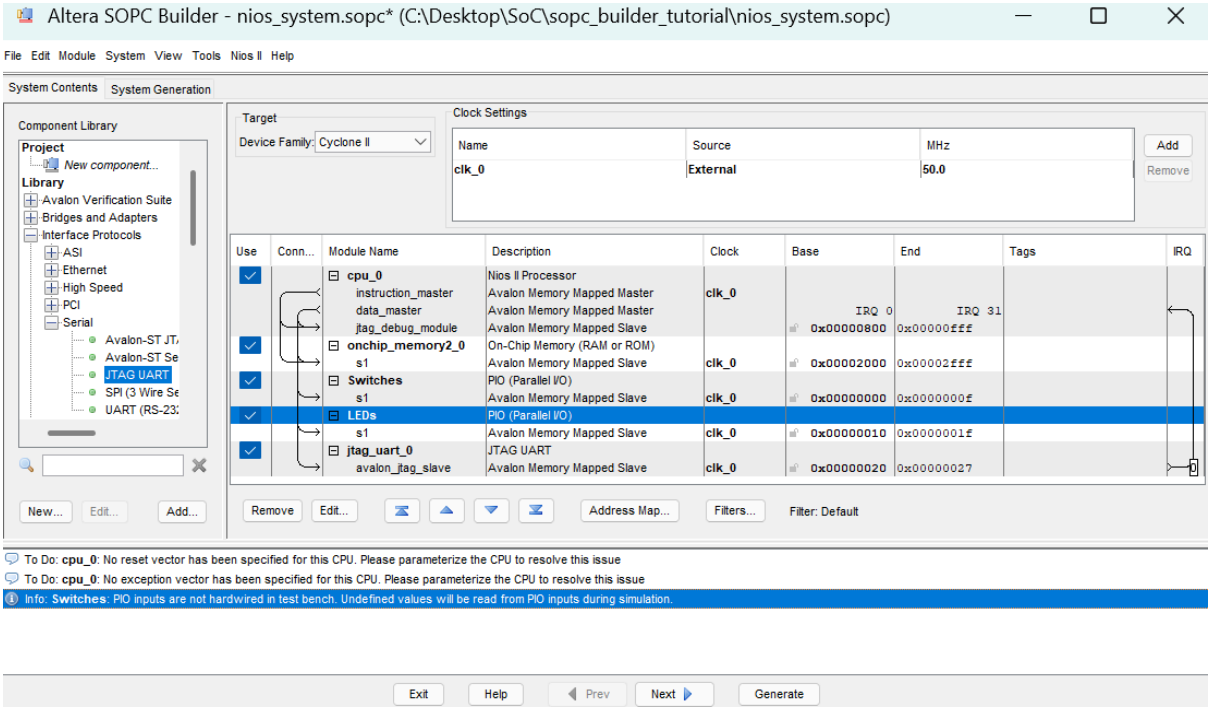


Figure 15. The final specification on a DE-series board.

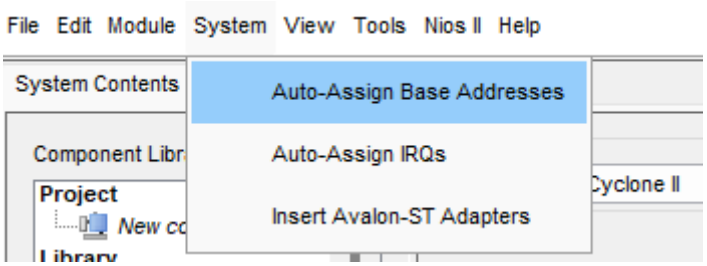


Figure 16. Assign base addresses.

The Nios II processor's reset behavior is defined by its reset vector, the memory location it fetches the next instruction from upon reset. The exception vector is the memory address it accesses during an interrupt. To set these parameters:

- Right-click on *cpu_0* and select Edit to open the window in Figure 15.
- Choose *onchip_memory2_0* as the memory device for both the reset vector and exception vector, as shown in Figure 15.
- Keep the default offset setting unchanged.
- Click Finish to return to the System Contents tab.

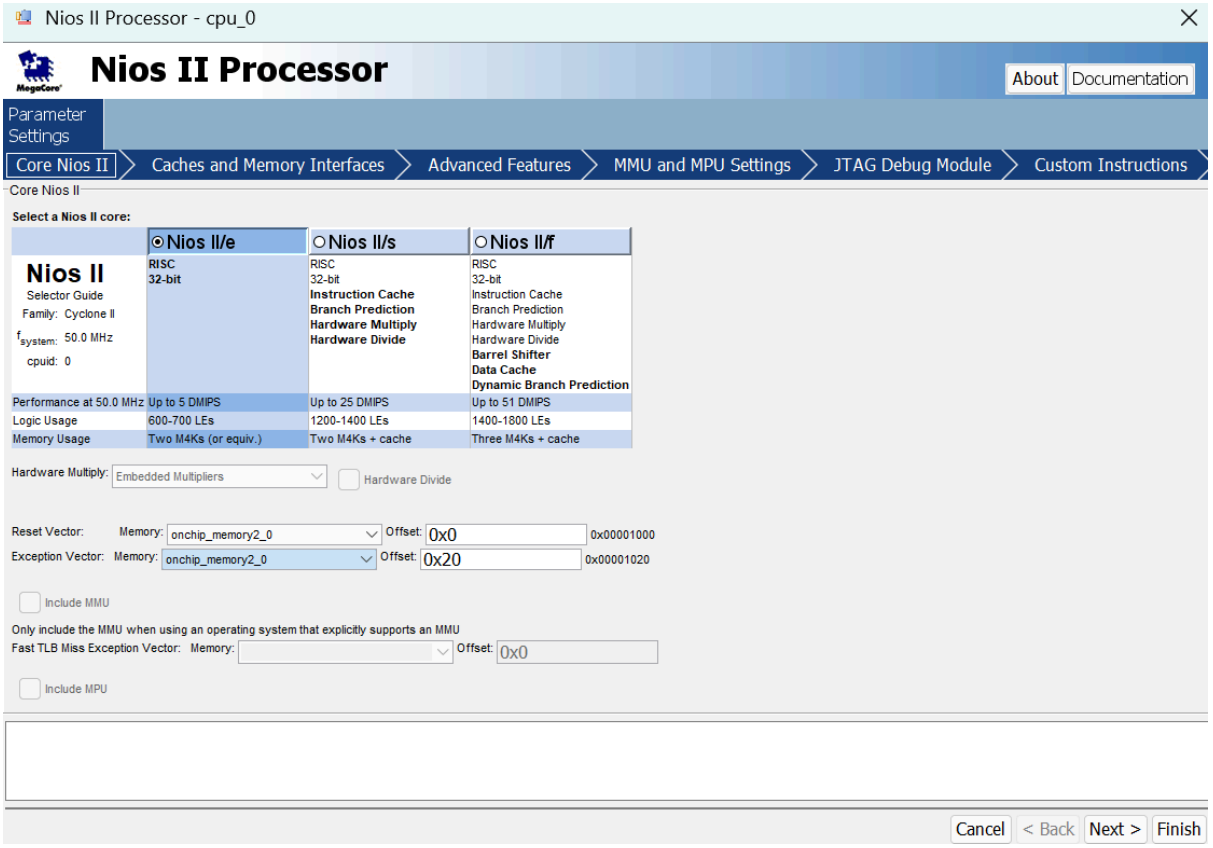


Figure 17. Define the reset vector and exception vector

Select the System Generation tab, which leads to the window in Figure 16. Turn off Simulation - Create project simulator files, because in this tutorial we will not deal with the simulation of hardware. Click Generate on the bottom of the SOPC Builder window. The SOPC Builder may prompt you to save changes to *.sopc* file. Click Save to continue. The generation process produces

the messages displayed in the figure. When the message “SUCCESS: SYSTEM GENERATION COMPLETED" appears, click Exit to return to the main Quartus II window.

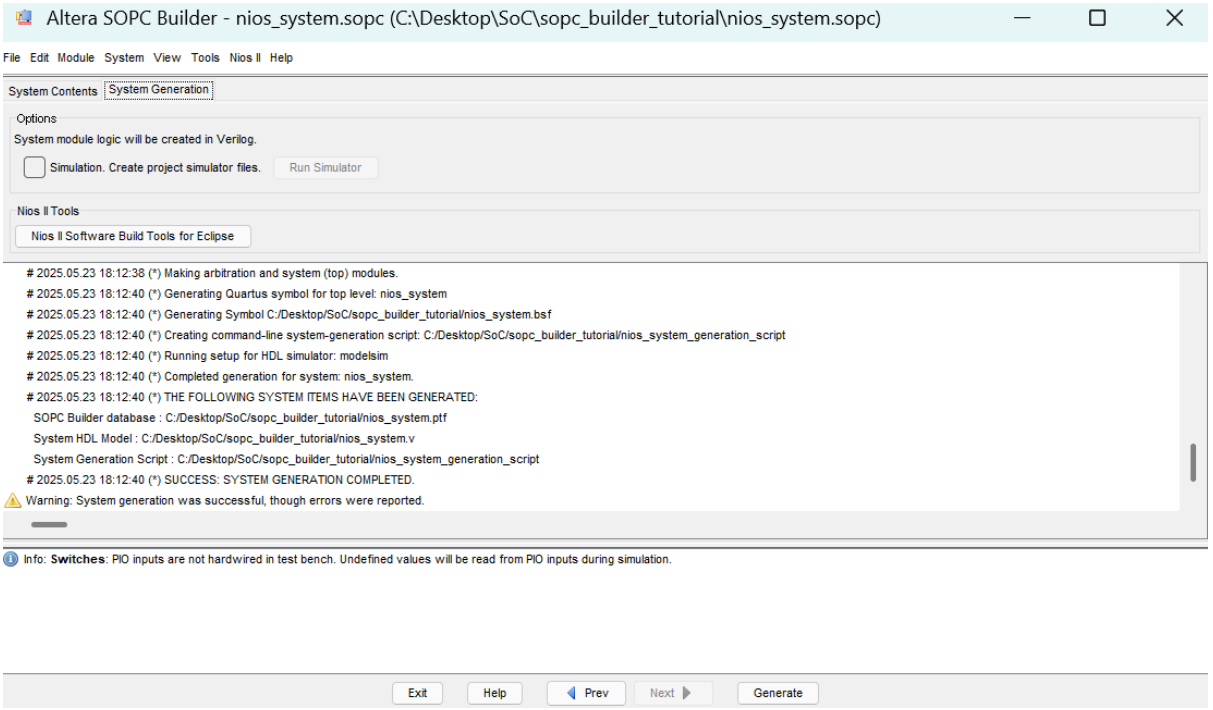


Figure 18. Generation of the system.

The Verilog module generated by SOPC Builder is located in the file nios_system.v within the project directory, with the module name matching the system name specified initially. The Verilog code is extensive. Figure 17 shows the section defining the input and output signals for the nios_system module: the 8-bit input vector for the parallel port Switches is named in_port_to_the_Switches, the 8-bit output vector is out_port_from_the_LEDs, the clock signal is clk_0, and the reset signal is reset_n. The reset_n signal, added automatically by SOPC Builder, is active low.

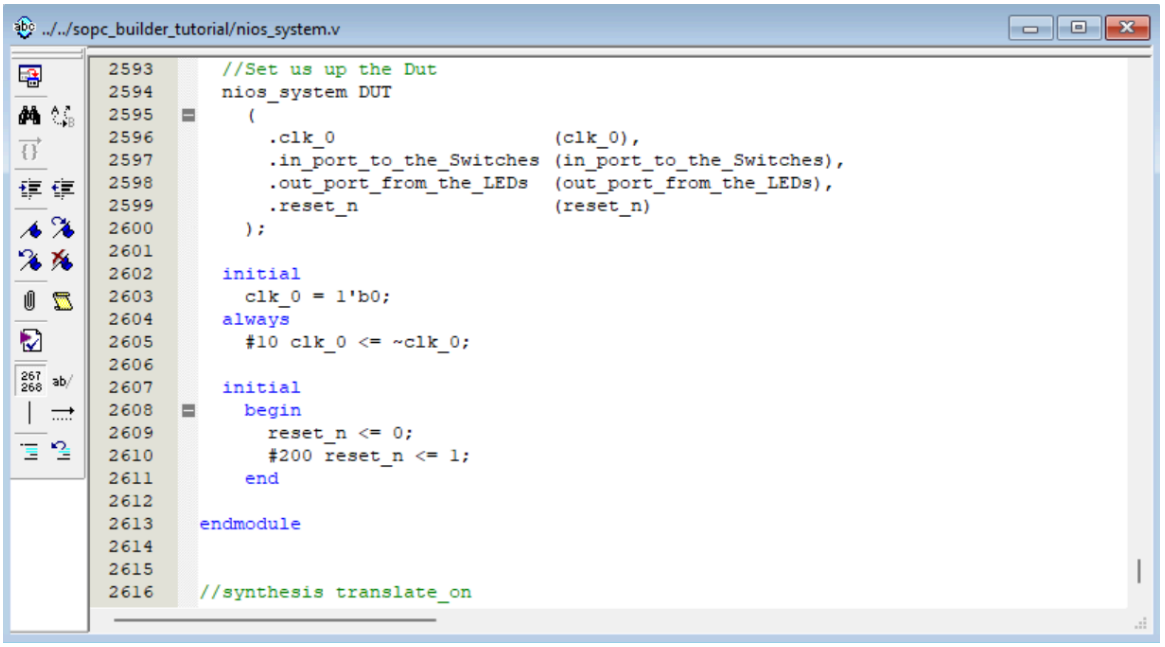


Figure 19. A part of the generated Verilog module.

2. Running the Application Program

After completing hardware configuration on Board DE2, we will create and execute an application program that performs the desired operation. We will program the Switches to control LEDs. If switch 0-7 is closed, the corresponding LEDs will turn on.

We write a program in the C language on the Nios II IDE, which provides libraries and functions specially for the Nios II processor.

A parallel I/O interface generated by the SOPC Builder is accessible by means of registers in the interface.

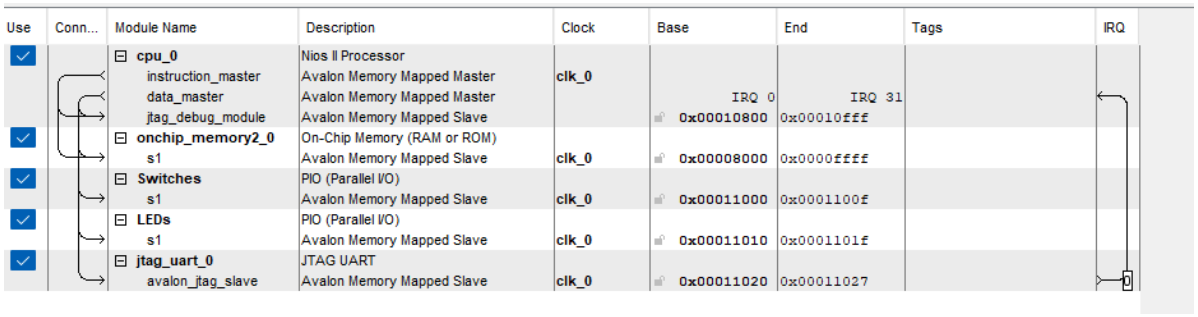
Depending on how the PIO is configured, there may be as many as four registers. One of these registers is called the Data register.

In a PIO configured as an input interface, the data read from the Data register is the data currently present on the PIO input lines.

In a PIO configured as an output interface, the data written (by the Nios II processor) into the Data register drives the PIO output lines.

If a PIO is configured as a bidirectional interface, then the PIO inputs and outputs use the same physical lines. In this case, a Data Direction register is included, which determines the direction of the input/output transfer.

In our unidirectional PIOs, it is only necessary to have the Data register.The addresses assigned by the SOPC Builder are 0x00011000 for the Data register in the PIO called Switches and 0x00011010 for the Data register in the PIO called LEDs, as indicated in Figure .



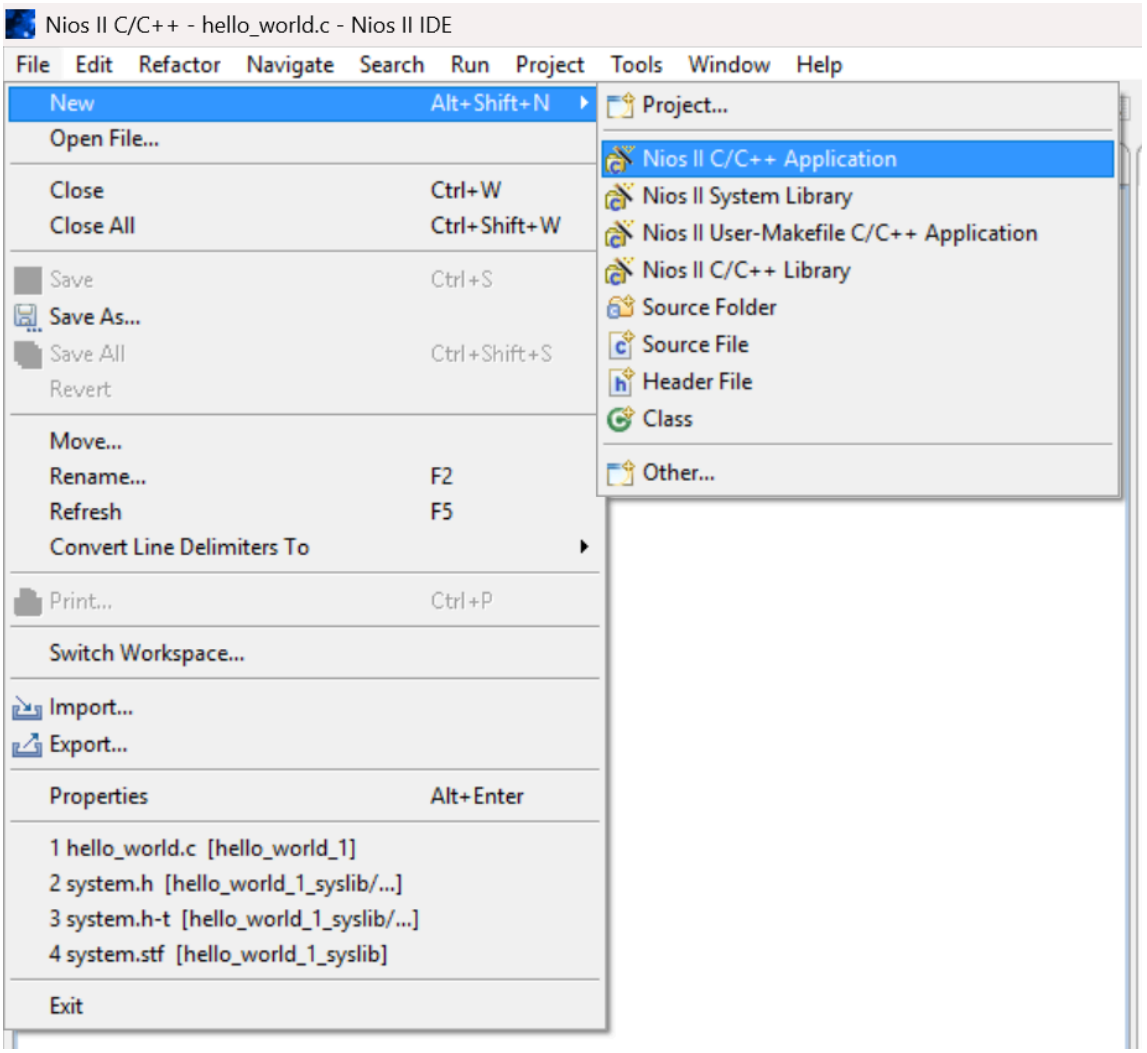
Use	Conn...	Module Name	Description	Clock	Base	End	Tags	IRQ
✓		cpu_0	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave		0x00010800	0x00010fff		
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x0000ffff		
✓		Switches	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	clk_0	0x00011000	0x0001100f		
✓		LEDs	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	clk_0	0x00011010	0x0001101f		
✓		jtag_uart_0	JTAG UART					
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00011020	0x00011027		

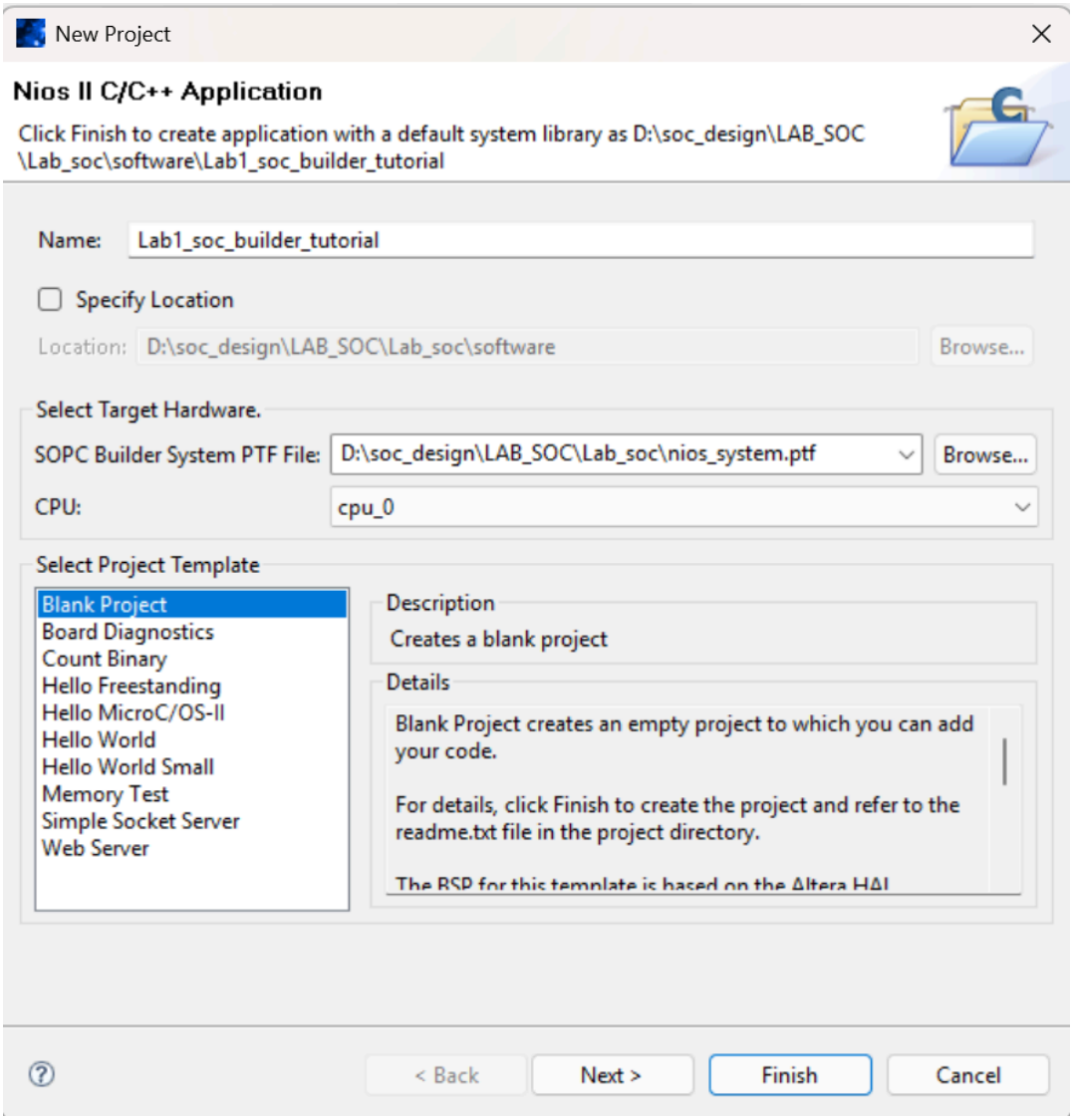
Figure: The complete Nios II system on Soc builder tools.

a. Create the Application Program

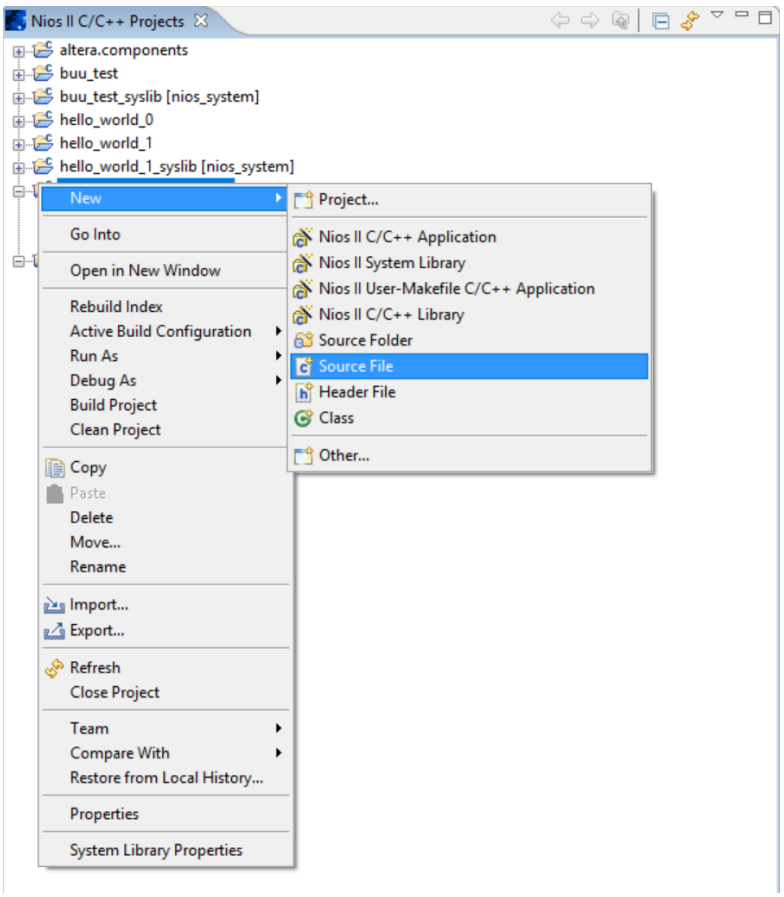
Step1. Open Nios II IDE and choose File → New → Nios II C/C++ Application (figure).

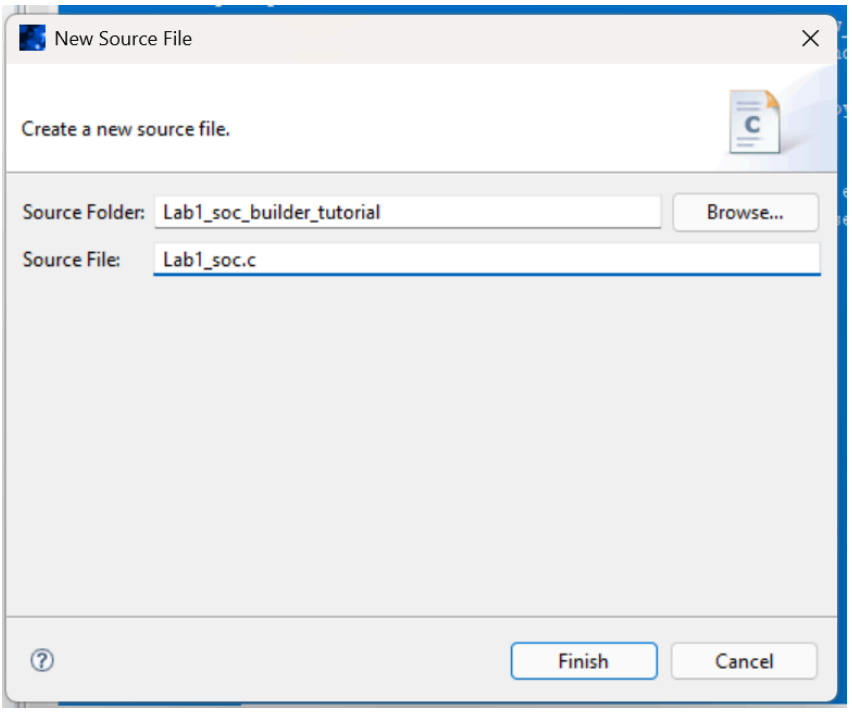
Then, set a name for the application and click the browser to select the SOPC builder systems PTF file (figure). After selection, in the CPU, it is the CPU name that we choose in the SOPC builder, and then choose Blank Project, and finish.





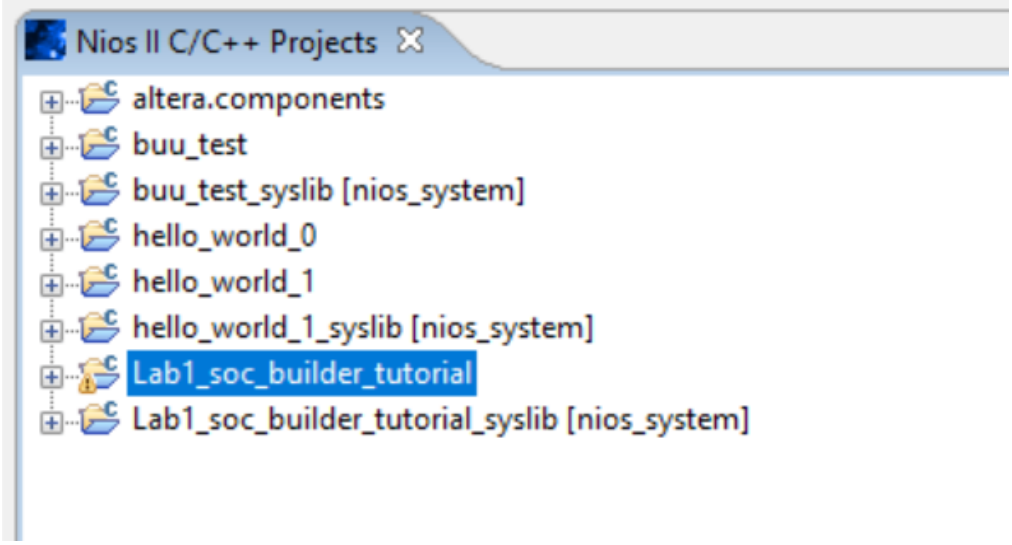
Step2. Add a C source file to the Project
Right click on Lab1_soc_builder_tutorial → New → Source file
We set the name file is Lab1_soc.c and then finish

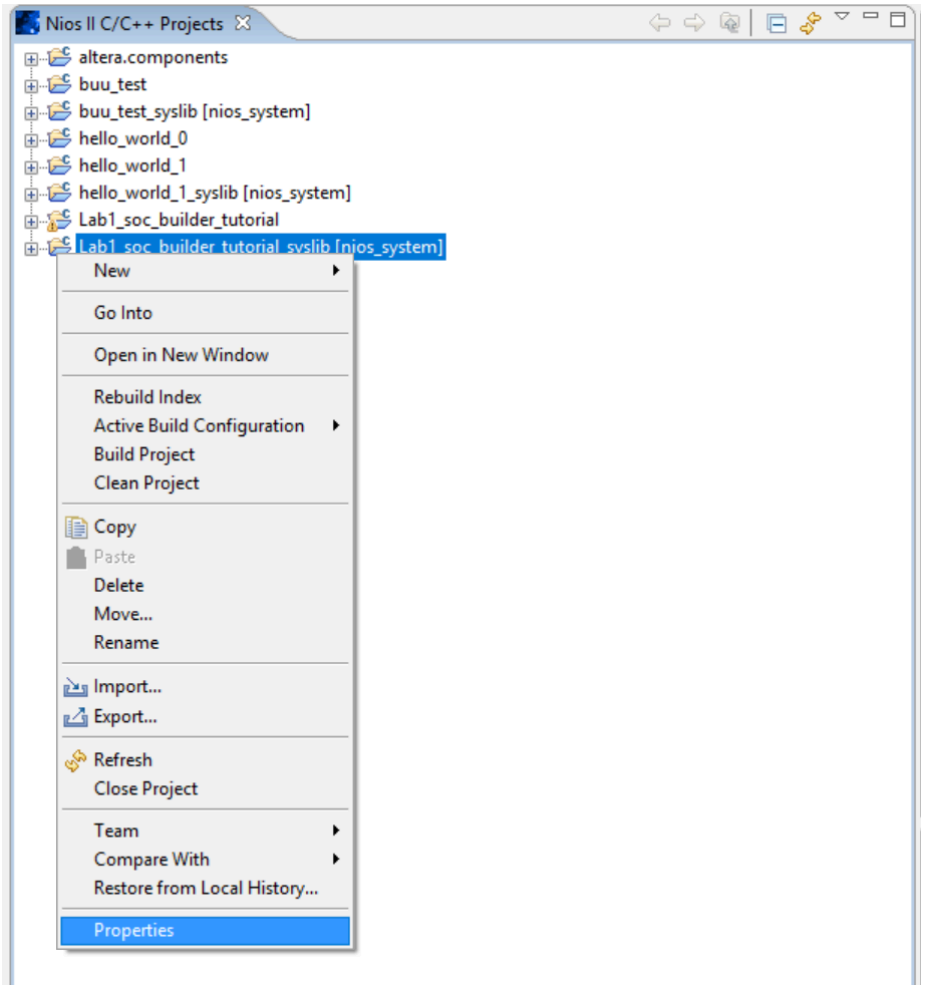


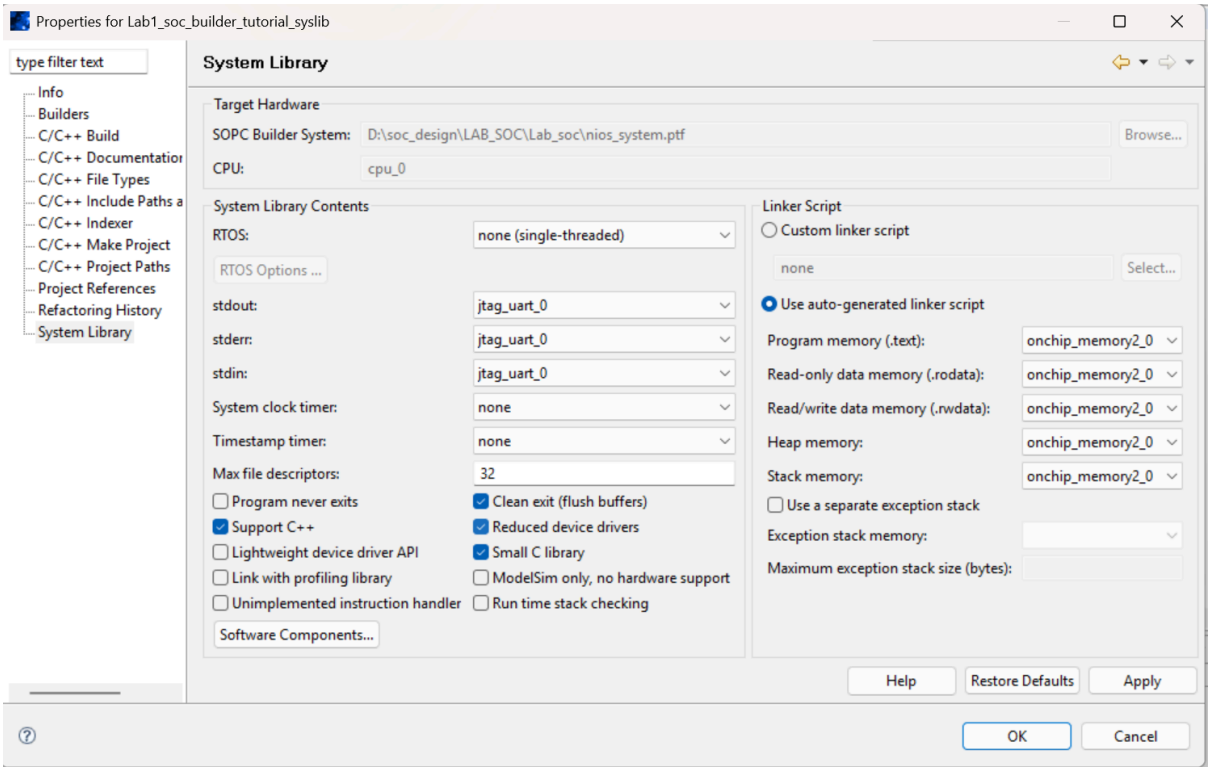


After this stage, there are two folders in the project list, including Lab1_soc_builder_tutorial and Lab1_soc_builder_tutorial_syslib.
To reduce the size of the project after debugging to be suitable for the size of the Memory on chip of the Nios II processor, which is 32 kbytes. Right click to

Lab1_soc_builder_tutorial_syslib → in the system library, tick to choose Reduce device drivers and the Small size library.







Step3. Write a C program to link switches and LEDs and display which switch is selected on the console screen.

```
#include <system.h>       // Contains base address definitions for peripherals
#include <alt_types.h>    // Defines fixed-width types like alt_u8
#include <io.h>           // Contains IORD and IOWR macros
#include <stdio.h>        // Standard I/O for printf

int main() {
   alt_u8 switch_value;  // 8-bit variable to hold switch input

   while (1) {  // Infinite loop
      switch_value = IORD(SWITCHES_BASE, 0);      // Read switch values
      IOWR(LEDS_BASE, 0, switch_value);           // Output to LEDs
      printf("Switch value: 0x%02X\n", switch_value); // Print switch value in hex
   }

   return (0); // Never reached
}
```

In this code, use the function IORD to read the value of switches in the FIFO by using the base address of the Switches and store it in the variable switch_value, and the function IOWR that reads values from the variable switch_value and writes them into the FIFO of LEDs through the LEDs' base address.

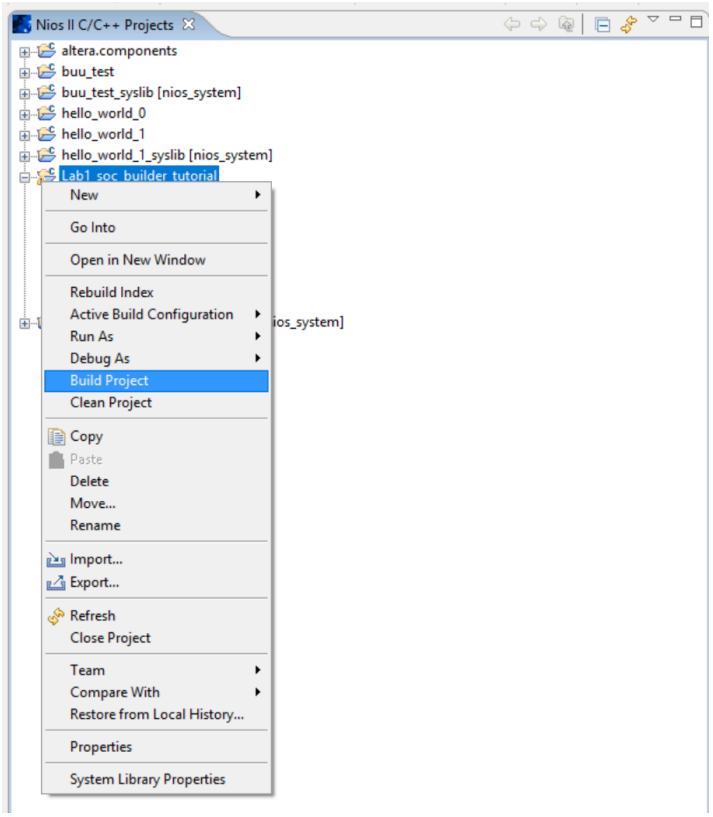
That means one switch controls one LED. And all function is supported by the Nios II library in the Nios II IDE.
All base addresses of components are automatically declared in the system.h file by the Nios II IDE.
And we also use the printf function to display with value in the console window.

```
/*
 * LEDs configuration
 *
 */

#define LEDS_NAME "/dev/LEDs"
#define LEDS_TYPE "altera_avalon_pio"
#define LEDS_BASE 0x00011010
#define LEDS_SPAN 16
#define LEDS_DO_TEST_BENCH_WIRING 0
#define LEDS_DRIVEN_SIM_VALUE 0
#define LEDS_HAS_TRI 0
#define LEDS_HAS_OUT 1
#define LEDS_HAS_IN 0
#define LEDS_CAPTURE 0
#define LEDS_DATA_WIDTH 8
#define LEDS_RESET_VALUE 0
#define LEDS_EDGE_TYPE "NONE"
#define LEDS_IRQ_TYPE "NONE"
#define LEDS_BIT_CLEARING_EDGE_REGISTER 0
#define LEDS_BIT_MODIFYING_OUTPUT_REGISTER 0
#define LEDS_FREQ 50000000
#define ALT_MODULE_CLASS_LEDs altera_avalon_pio
```

```
/*
 * Switches configuration
 *
 */

#define SWITCHES_NAME "/dev/Switches"
#define SWITCHES_TYPE "altera_avalon_pio"
#define SWITCHES_BASE 0x00011000
#define SWITCHES_SPAN 16
#define SWITCHES_DO_TEST_BENCH_WIRING 0
#define SWITCHES_DRIVEN_SIM_VALUE 0
#define SWITCHES_HAS_TRI 0
#define SWITCHES_HAS_OUT 0
#define SWITCHES_HAS_IN 1
#define SWITCHES_CAPTURE 0
#define SWITCHES_DATA_WIDTH 8
#define SWITCHES_RESET_VALUE 0
#define SWITCHES_EDGE_TYPE "NONE"
#define SWITCHES_IRQ_TYPE "NONE"
#define SWITCHES_BIT_CLEARING_EDGE_REGISTER 0
#define SWITCHES_BIT_MODIFYING_OUTPUT_REGISTER 0
#define SWITCHES_FREQ 50000000
#define ALT_MODULE_CLASS_Switches altera_avalon_pio
```

Figure: LEDs and Switches base addresses are declared in the system.h file.

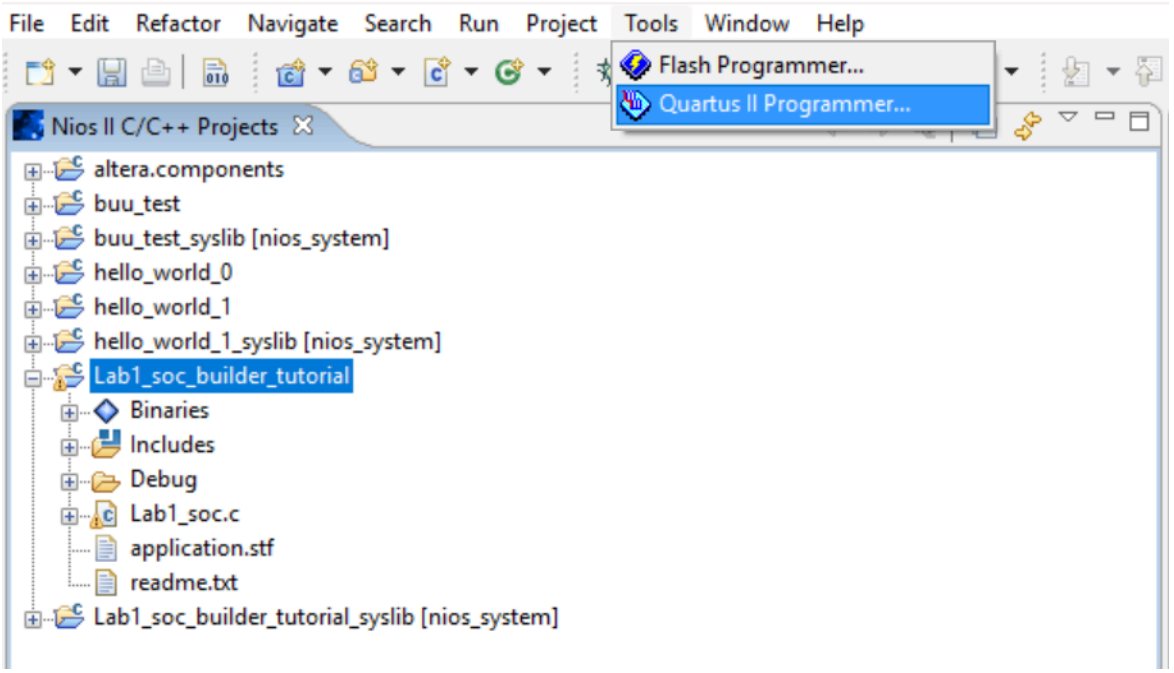
b. Running on Nios II hardware
Step1. Right click in Lab1_soc_builder_tutorial → Build Project

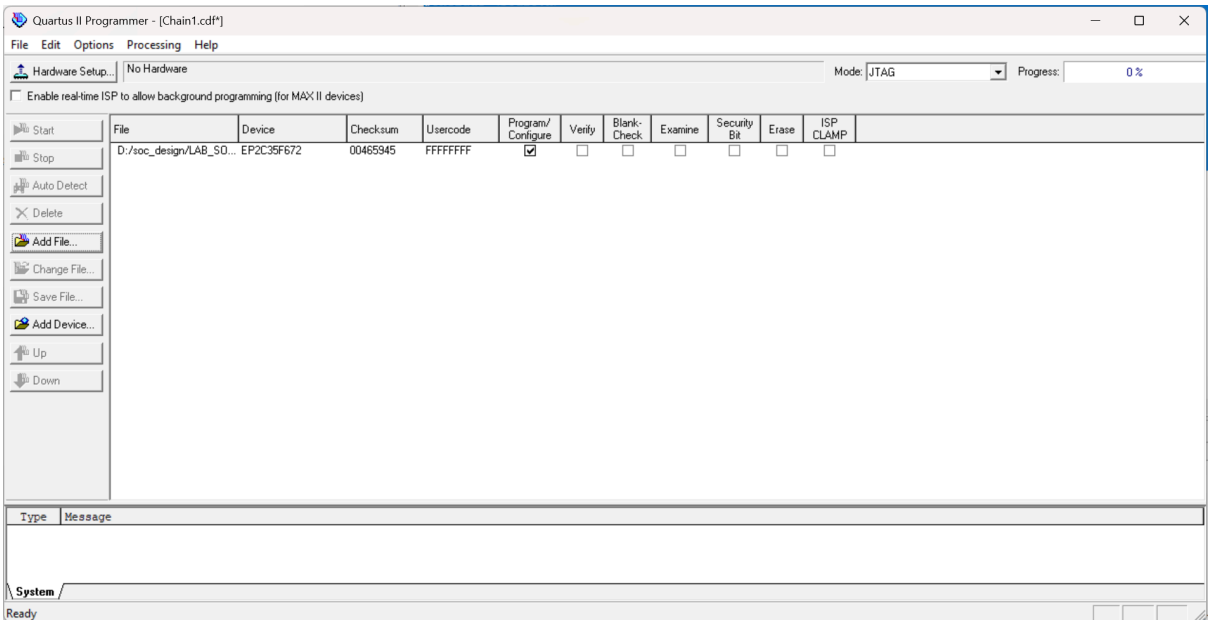


To run the program on Nios II:

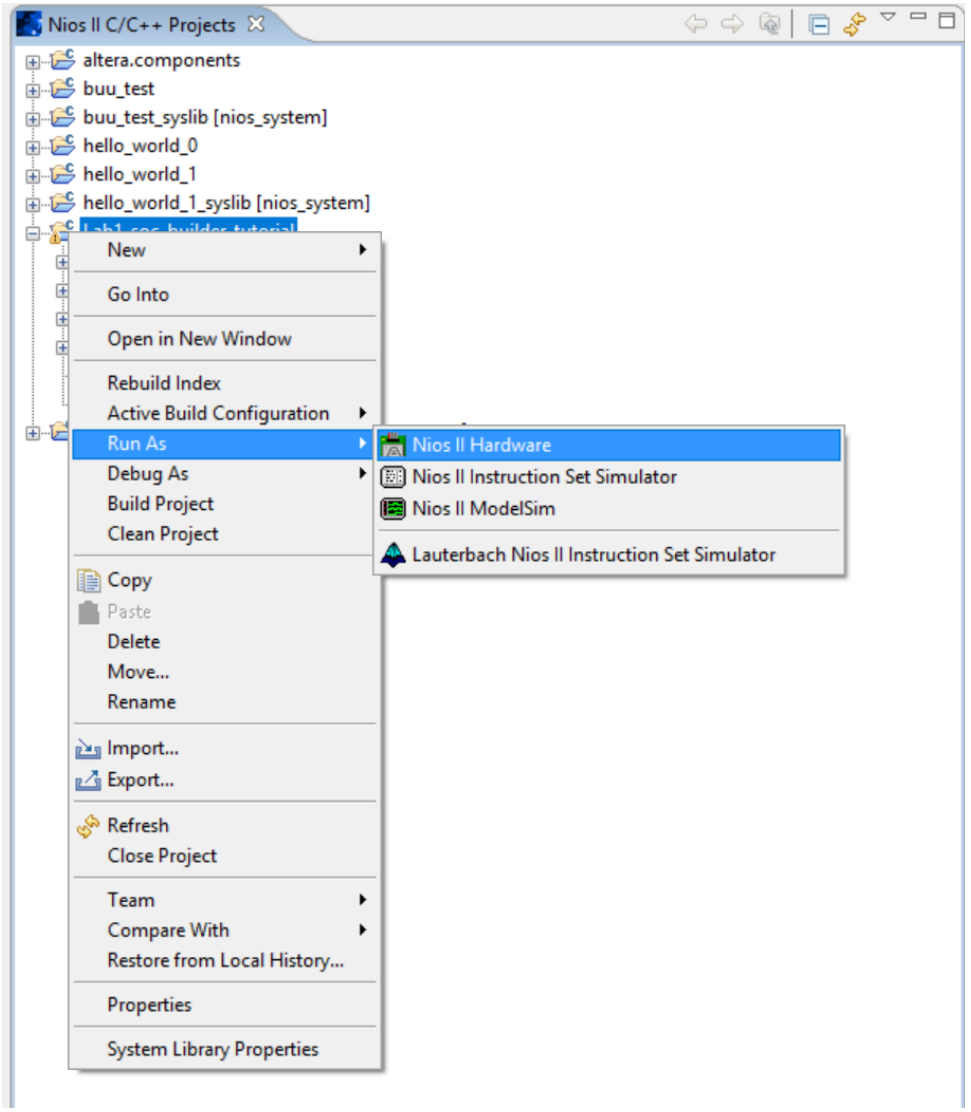
Step2. We create the Nios II processor on board DE2.

Tools → Quartus II programmer and choose the file light.sof and start.





Step3 .we right-click in Lab1_soc_builder_tutorial → Run as → Nios II hardware.



c. Result

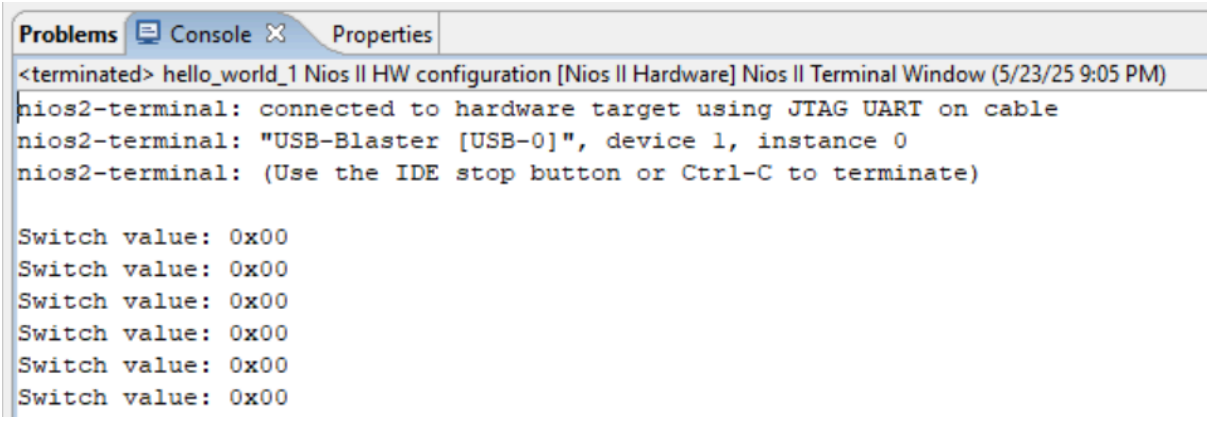


Figure: The Console window displays the Ox00 when no Switch is closed.

```
Switch value: 0x01
Switch value: 0x01
Switch value: 0x01
Switch value: 0x01
Switch value: 0x01
Switch value: 0x01
Switch value: 0x01
Switch value: 0x03
Switch value: 0x03
Switch value: 0x03
```

Figure: The Console window displays the 0x01 and 0x03 corresponding to 1 and 3 switches that are closed.

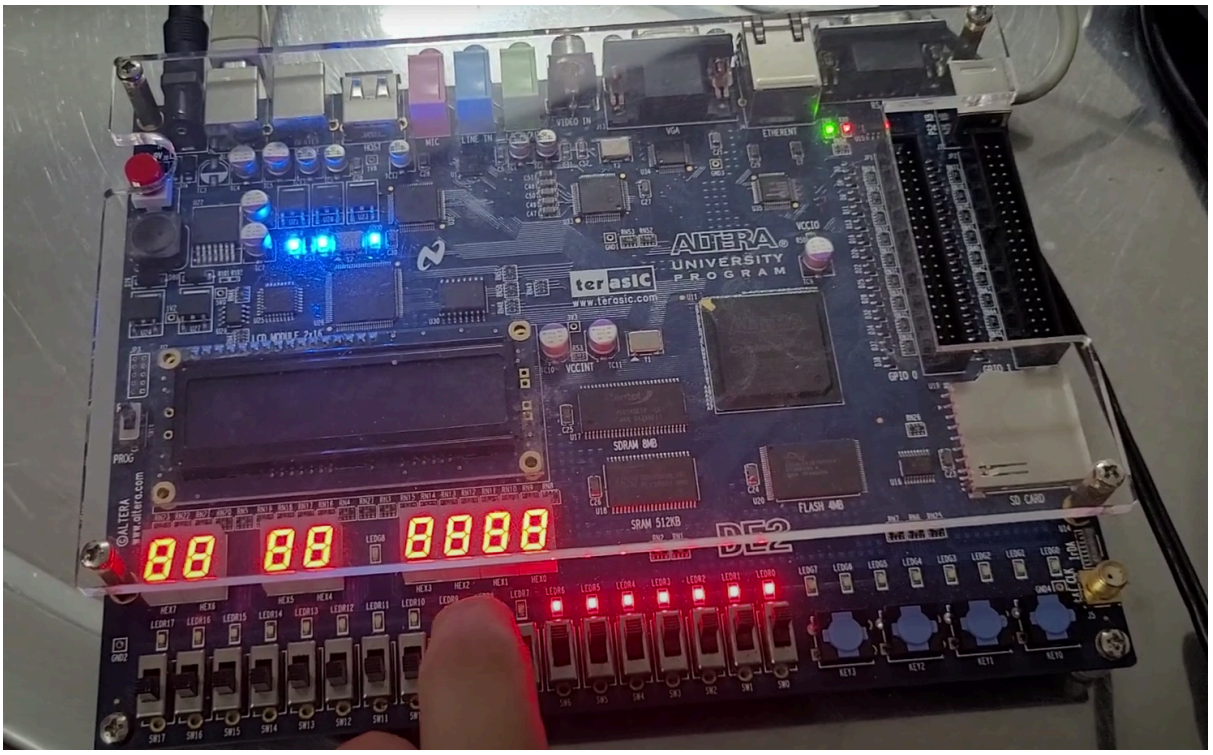


Figure: The number of Leds turned on corresponds number of Switches closed.