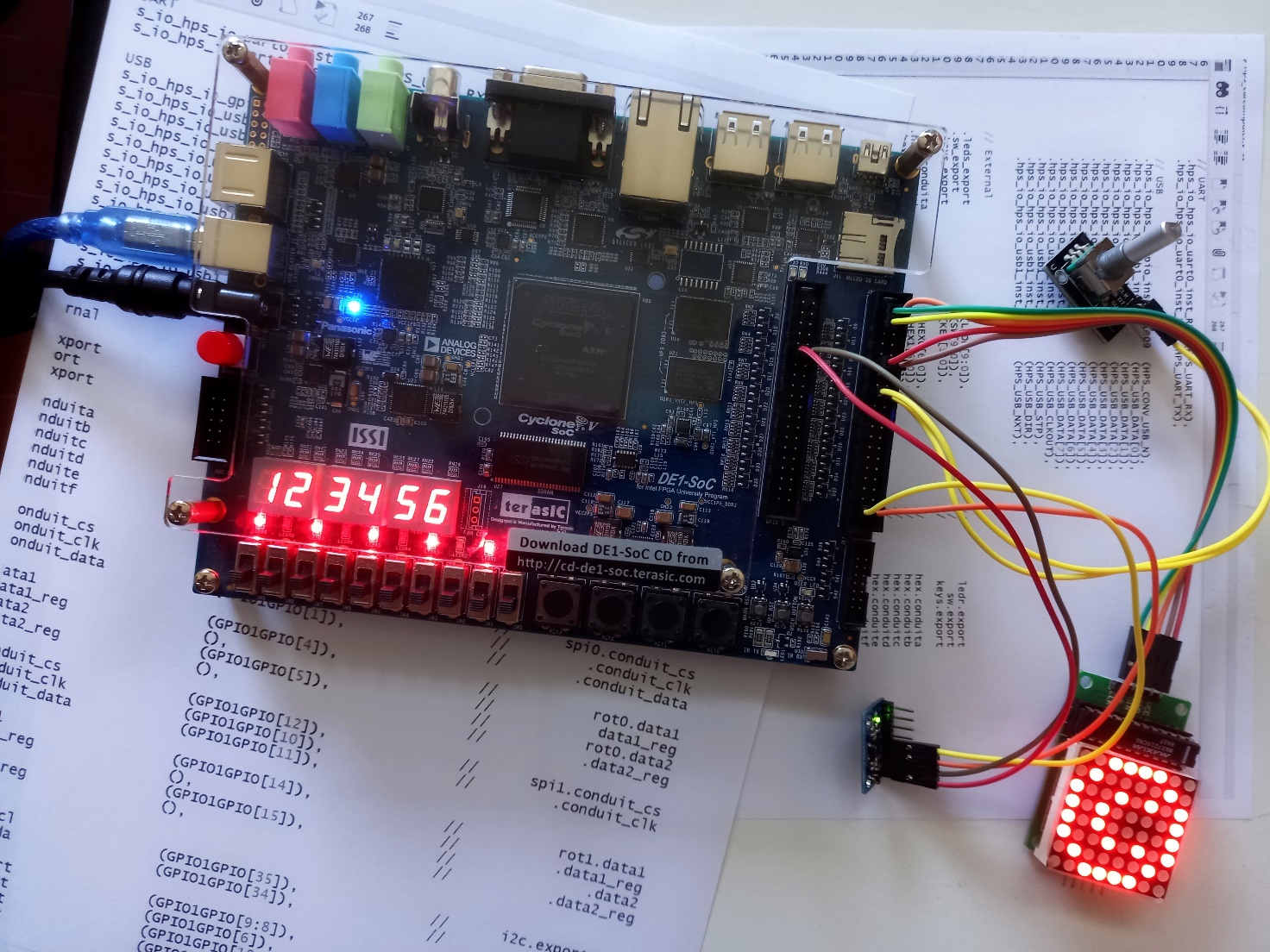
Electronic Systems Design

# Prof. Marsi Stefano University of TriesteAcademic Year 2025/26

Tutorial 5



**Development of a simple processor with some peripherals based on NIOS II.**  
**Hardware** used: Terasic DE1-SoC Board  
**Software** used: Quartus 22.1

Tutorial 5

Creation of a NIOS II processor, equipped with peripherals.

Description: In this tutorial you will create a simple system consisting of a NIOS2 processor, a memory and some peripherals.

Purpose: The aim is to see how an entire computing system can be created within a single FPGA and how this system can be used to interact with the outside world through peripherals already developed by third parties or even by developing the most suitable peripheral to create a certain interface to the outside.

Expected learning:

* Learning of the Quartus tool "Platform Designer" to create a processor together with its interface system
* Configuring the processor and third-party peripherals
* Creation of "ad hoc" peripherals
* Processor programming and code execution in a Codesign "Hardware-Software" process

# Premise

The internal resources of an FPGA are sufficient to create a processor that can then be interfaced with the outside world and programmed to perform various operations.

This practice is so developed that most modern FPGAs already contain both embedded processors made in Hardware (Hard-Processors), and some of the most common interfaces (I2C, SPI, IrDA, UART, ...) . The CycloneV is no less and it too already contains a dual-processor ARM and various peripherals, but to practice how a processor can be made, configured, interfaced and programmed in this first tutorial we will create what is commonly called a "Soft-Processor", that is a processor made using the "Generla Purpose" resources inside the FPGA such as logic blocks and memories. The potential of the processor that will be created will be extremely limited, but the purpose of the Tutorial is to familiarize with the development tool. For the use of the Embedded processor (Hard-Processor) please refer the reader to the following Tutorials.

The realization of an entire processor starting from primitive logic blocks would be a task too vast and arduous to be contained in a single tutorial, therefore we will use the system developed by Quartus (Platform Designer) which allows you to define and configure a processing platform with its entire network of interfaces in a rather simple way.

# Realization of the architecture

Start by creating the "skeleton" of a system using the "System Builder" tool that involves the use of

* Leds
* 7-segment display
* Swithes
* Keys
* Clock
* GPIO1 (in default mode)

Not all of these peripherals will be used at first, but their presence will be useful later without having to reconsider the project from the beginning.

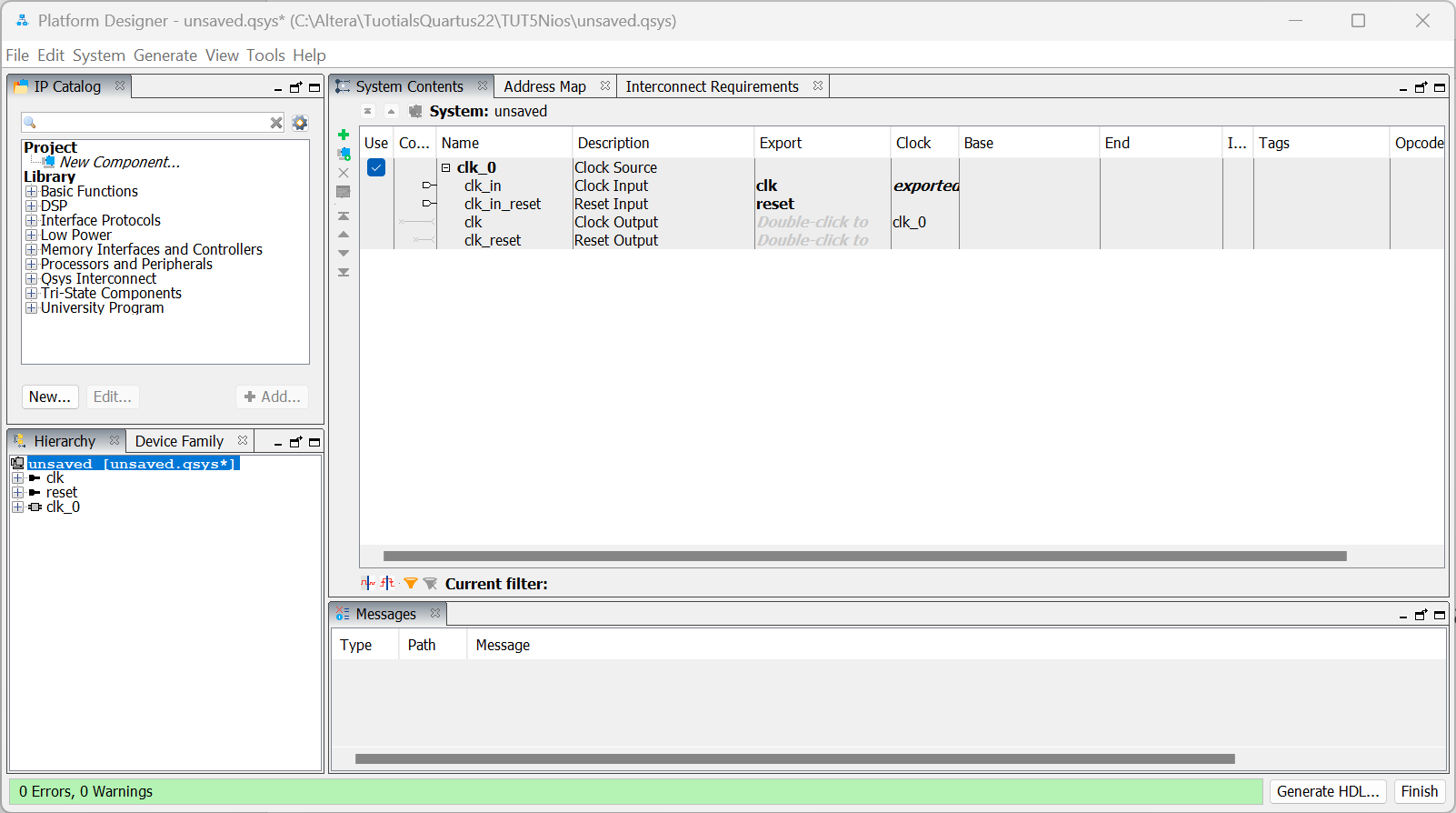
Open the project thus generated within Quartus.

Inside Quartus:

Tools > Platform Design

This opens up a system that allows you to configure the architecture based on blocks already developed by third parties.

The first block is already present and is a block useful for managing the clock and reset

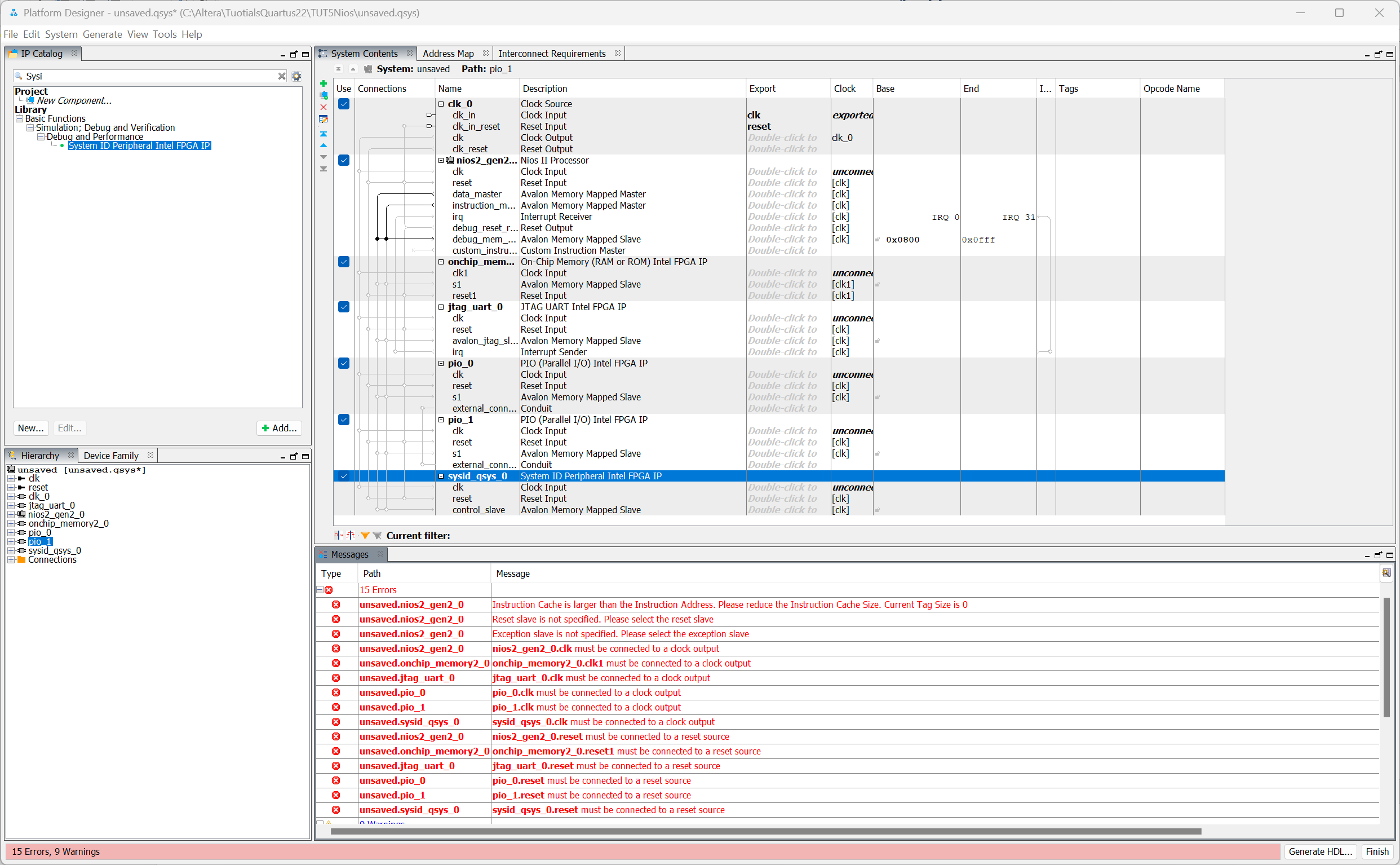


In the left window, organized in submenus, there are several blocks already developed by third parties.

Among these we will search (perhaps using the filter function) and add the following Blocks. For each of them a configuration window will open, but initially for each of them accept the default configuration through the "Finish" button.

* **NIOS II Processor** (the soft processor)
* **On-chip Memory (RAM or ROM) Intel FPGA IP** (the memory where data and code will reside)
* **JTAG UART Intel FPGA IP** (The STDIO serial interface to communicate with the processor)
* Two Intel FPGA IP Parallel I/O (PIO) **interfaces**
* **System ID Peripheral Intel FPGA IP** (System Identifier – SysID)

The window with the architecture should look more or less like this:



Where in red are indicated architectural errors that we will gradually fix.

The various columns in the "System Contents" tab are used to define:

1. Links between blocks
2. The name of the I/O blocks and signals
3. The description of the precedents
4. The signals that the blocks export to the outside of the architecture (I/O)
5. The Clock Domain
6. The Basic Address
7. The final address
8. The level of interupt

…

By clicking on the various blocks using the "CTRL-R" command you can define the name of the most suitable block, while using "CTRL-E" you can define the specifications. By clicking on the black or white "dots" of the various connections, they can be activated or deactivated.

Rename the various blocks appropriately and configure the connections as follows:

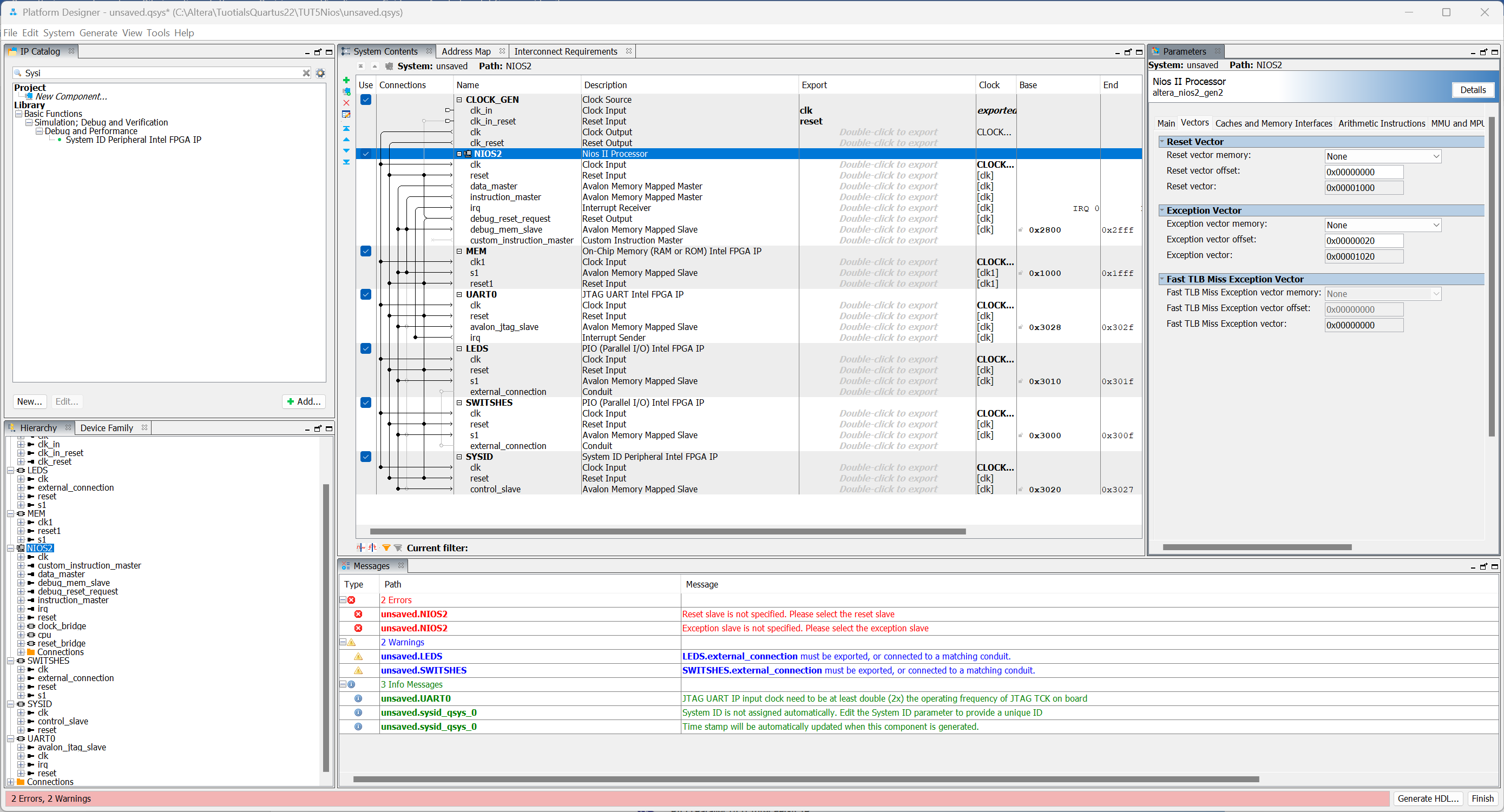
* The clock signal output from the "Clock generator" is connected to all blocks
* The Reset signal output from the clock generator is connected to all blocks

System > Create Global Reset Network

* The **data\_master** line of NIOS2 reaches all blocks   
  (i.e. the NIOS reads/writes data from all blocks)
* the Line **instruction\_master** reaches the memory block   
  (i.e. the instructions for in Nios reside only in memory)
* The **UART0 interrupt** is connected to the processor

We are now going to define appropriate memory addresses for the various blocks. The automatic procedure can be useful in this sense

System > Assign Base Address

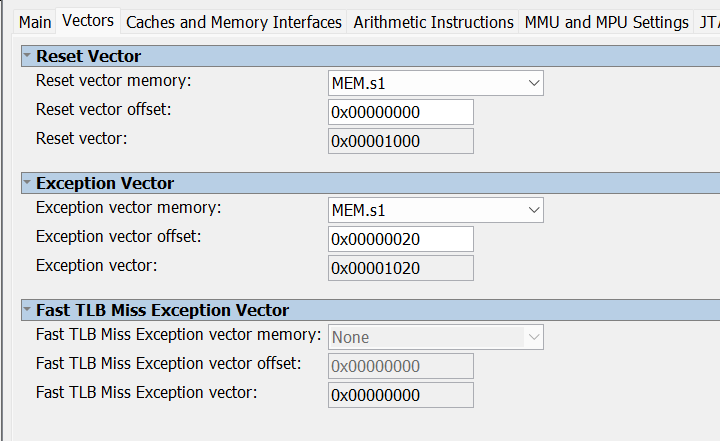


The architecture could look more or less like this with still some errors.

Then go to configure (CTRL-E) on the various blocks

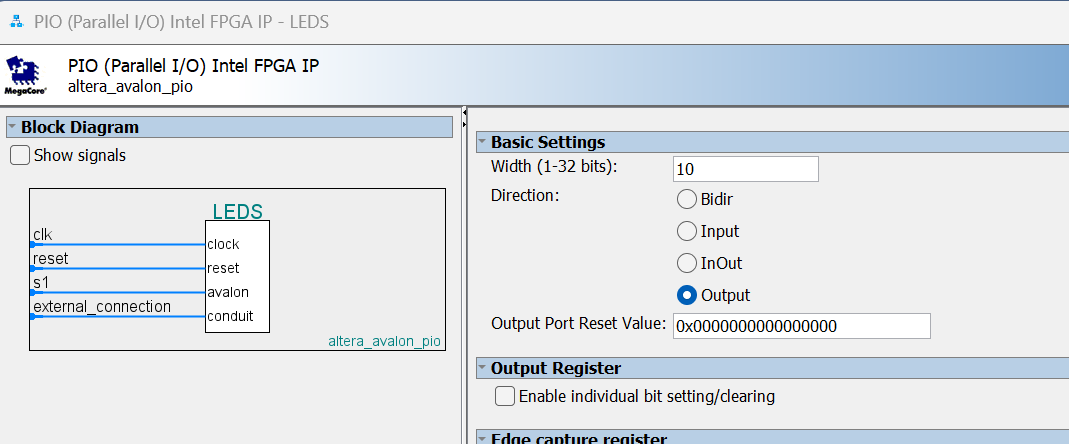
**Nios:**

* Type Nios II/e
* Reset Vector Memory must refer to "On-chip Memory"
* Exception Vector Memory must refer to "On-chip Memory"



Leds

* Have a length of 10 bits and are configured as an output

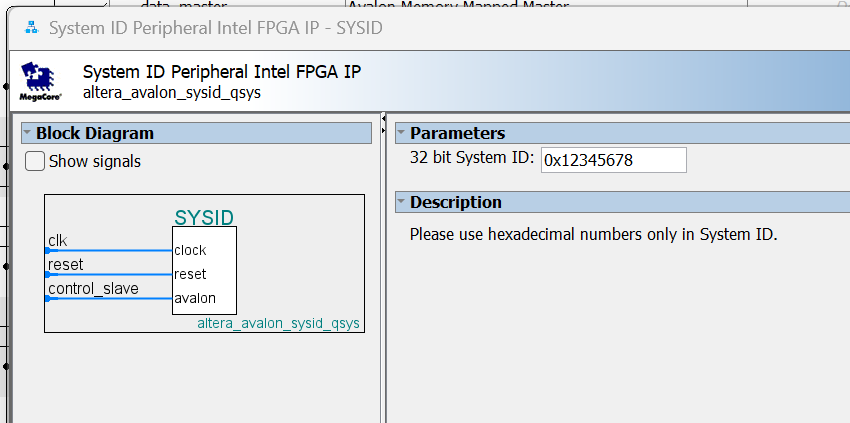


SWITCHES

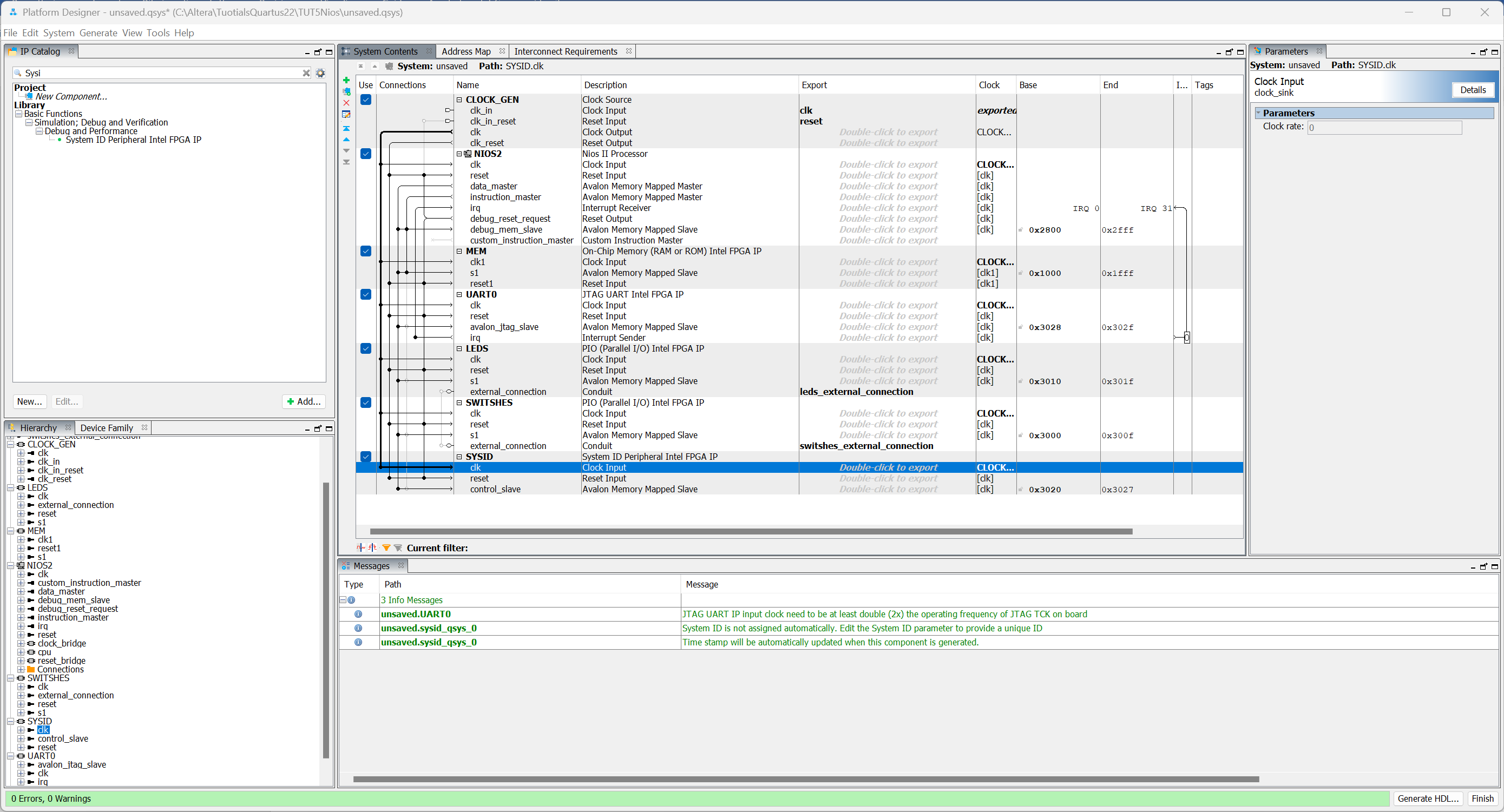
* Have a length of 10 bits and are configured as input

SYSID

* Provide a unique identifier value



Finally, with a double click on the "conduit" signal of "leds" and "switches" you can define the name with which these blocks interface with the outside of the architecture.



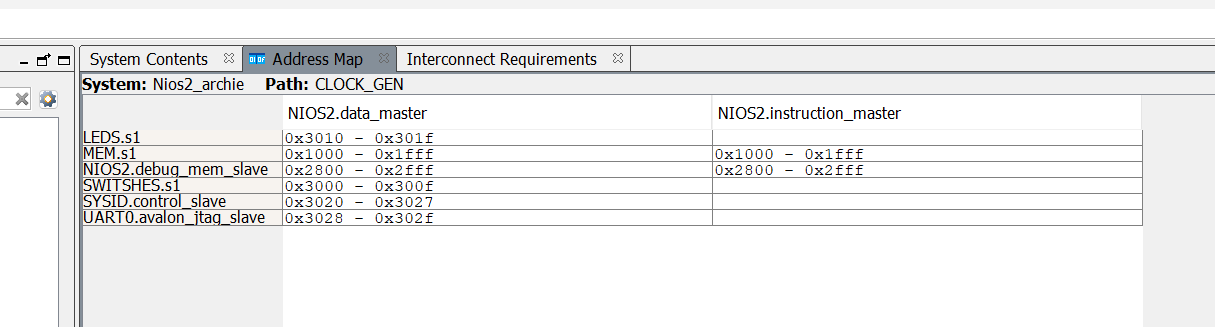
The architecture is fully defined and saved with an appropriate name (e.g. "Nios2\_archie")

and its respective HDL code can be generated, using in this case the "Verilog HDL" choice

When the build is completed

< Close >

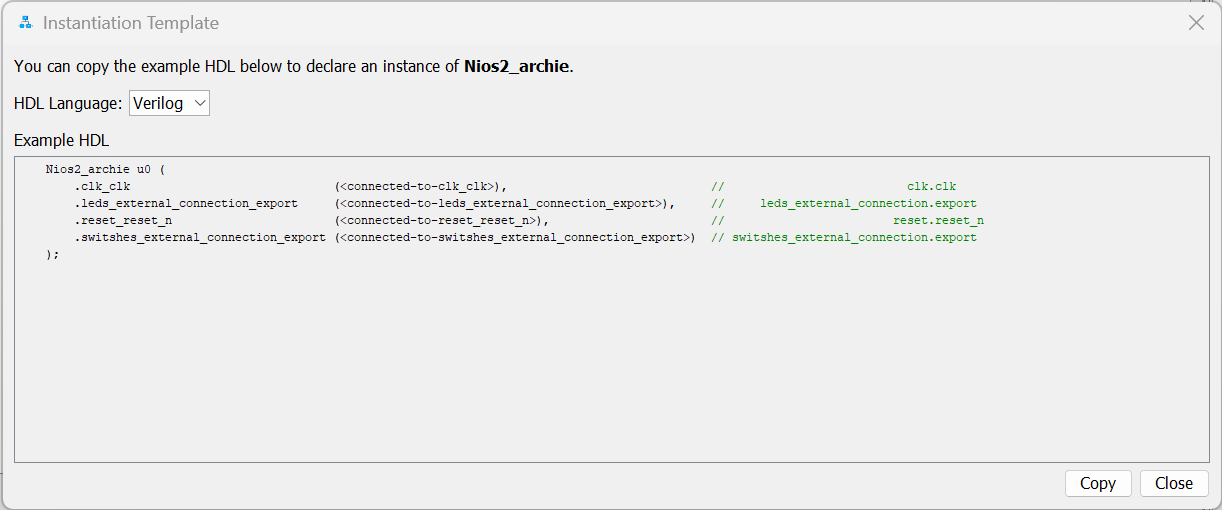
Before closing the procedure, take note of the memory locations of the various devices



And it might be useful to see an example of how to instantiate the component:

Generate > Show Instanstation Template

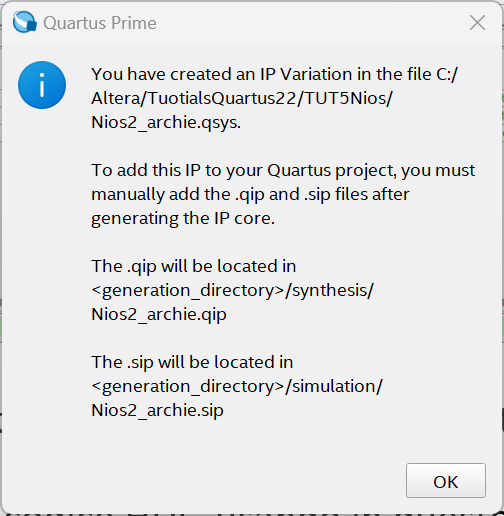
Copy the code



< Copy >

< Finish >

A similar window will probably appear reminding you where the files to be manually included in the project have been stored.

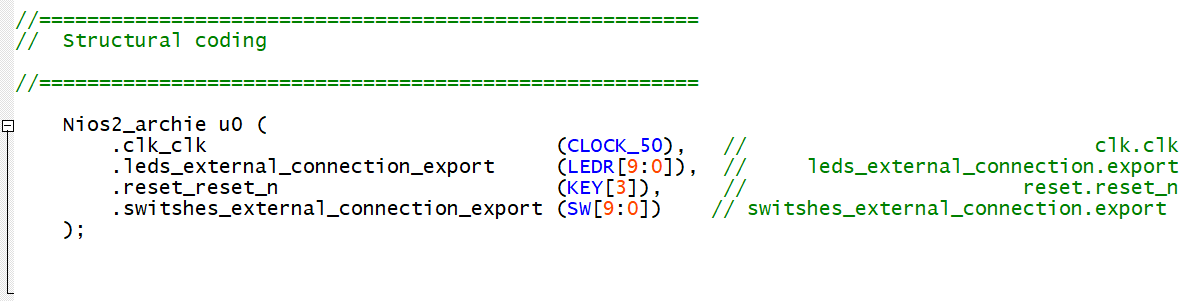


# Instantiation of the architecture within the project

The architecture generated in the previous step must now be imported at the global system level and interfaced with any other blocks and signals.

For example, it is necessary to ensure that the clock and an appropriate reset signal arrive at the architecture and its I/O signals related to LEDs and switshs connect to the corresponding pins inside the DE0-SoC board

At the level of the "Top Level Entity" generated with the "system builder" tool at the beginning of the tutorial, taking inspiration from the newly generated instantiation model, you can integrate the code with:



While at the system level you still need to include the generated ".qip" file in the project.

Project > Add/Remove files in project

Click on the three dots

Go to find the file where it was previously indicated by the dialog box and import it.

<OK>

At this point the entire system can be compiled and loaded onto the DE1-SoC board.

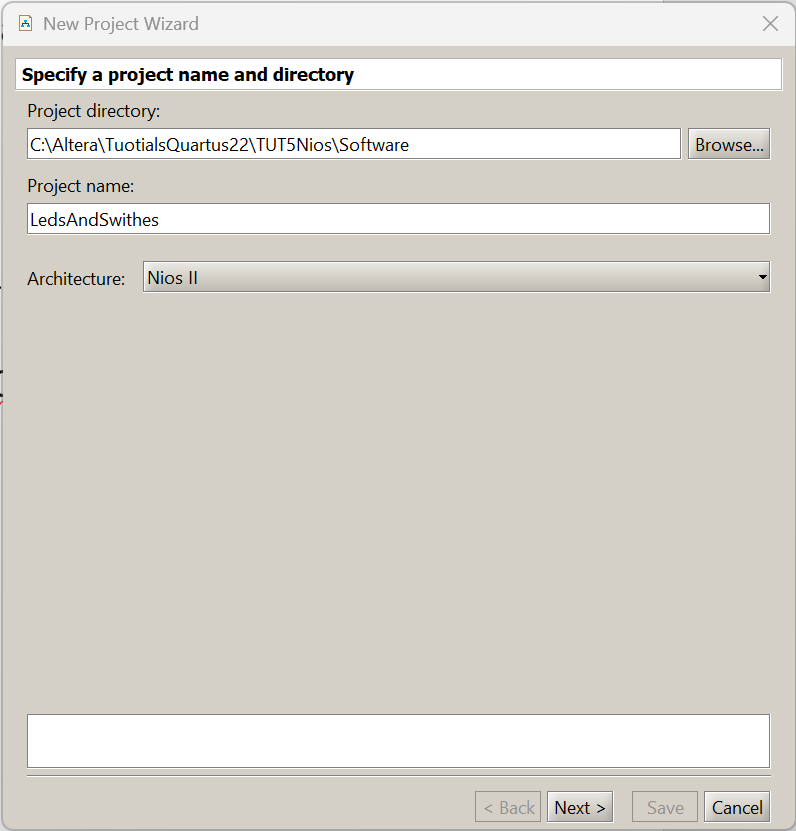
The Hardware Part inherent in the realization of the system is finished and you can ask for Quartus.

# Intel FPGA Monitor Program

Launch "Intel FPGA Monitor Program"

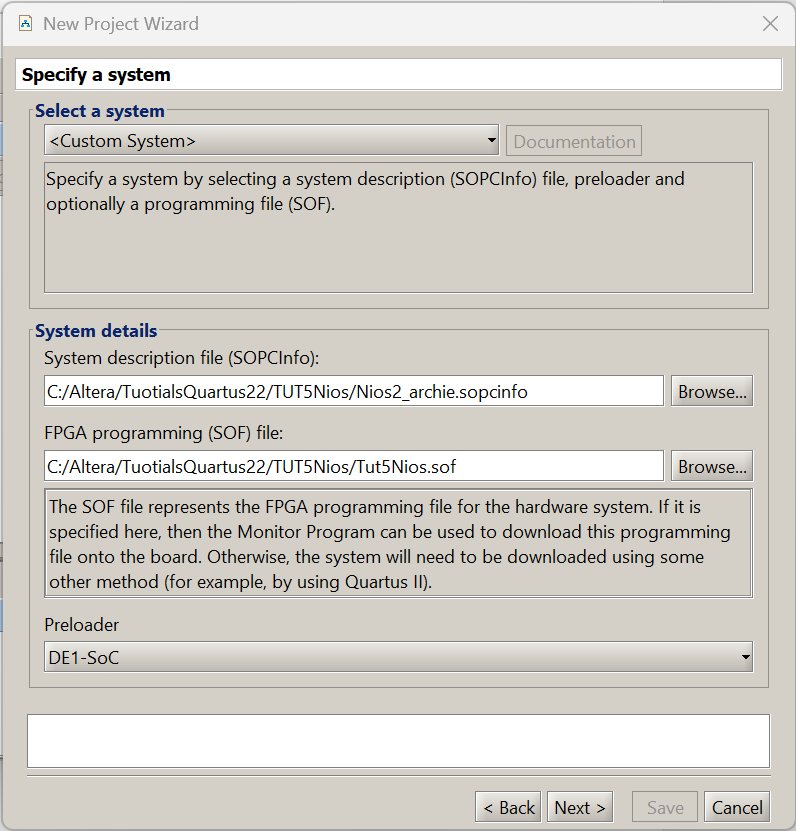
File > New Project

Define an appropriate directory, a name for the project, and choose which NiosII processor



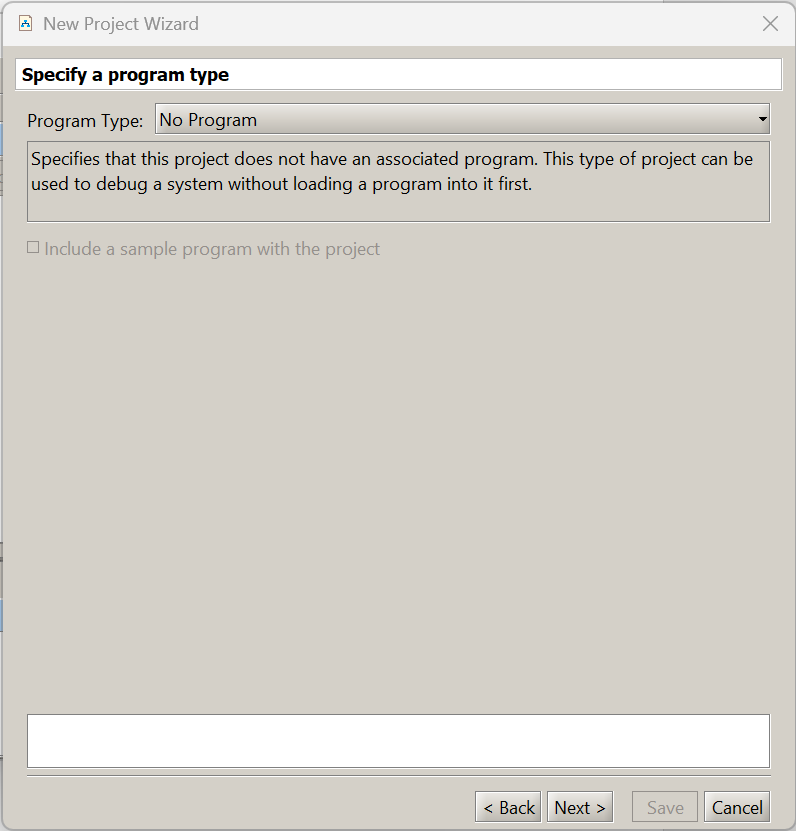
> Next

As the type of architecture, choose <Custom>, then indicate the files that characterize the hardware system .sopcinfo and .sof located in the directory where the hardware of the project resides, as a pre-loader the one relating to the DE1-Soc card



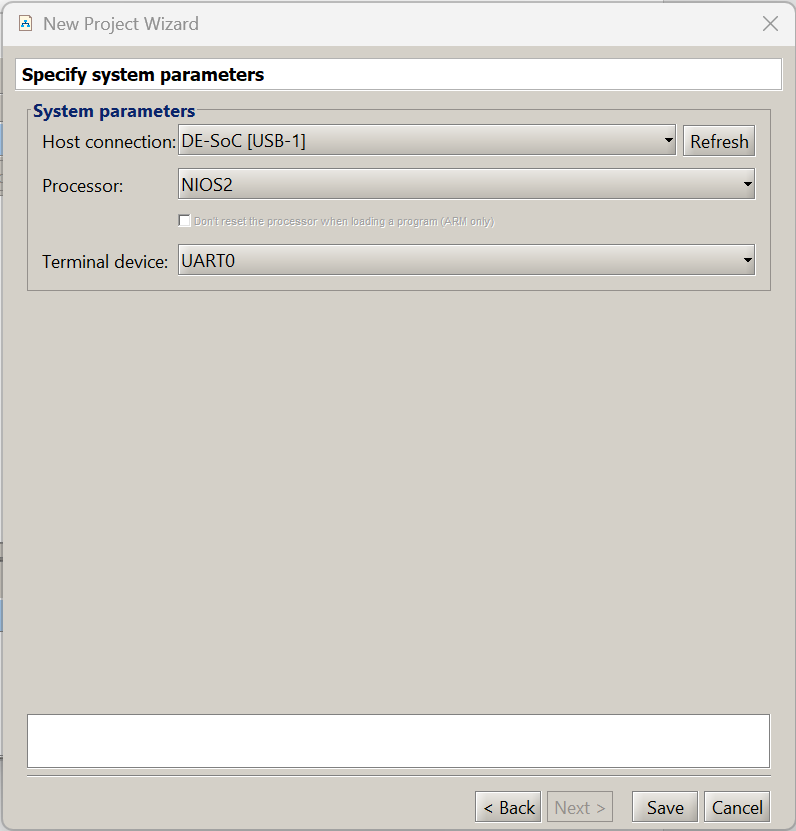
> Next >

As a type of programming at the moment choose <No Program>



> Next >

Then define the Processor and communication interface



> Save

The system will ask if you want to load the configuration file on the board, but if you have a small bug of the system it is better to skip this point and use Quartus to program the board (as already done).

Action > Connect to System

Access the Memory tab

Goto Address go to the addresses where LEDs, Swithes and SysIDs have been mapped (e.g. 0x3000), check "Query Device"

> GO

At this point you can see the various peripherals through their memory addresses:

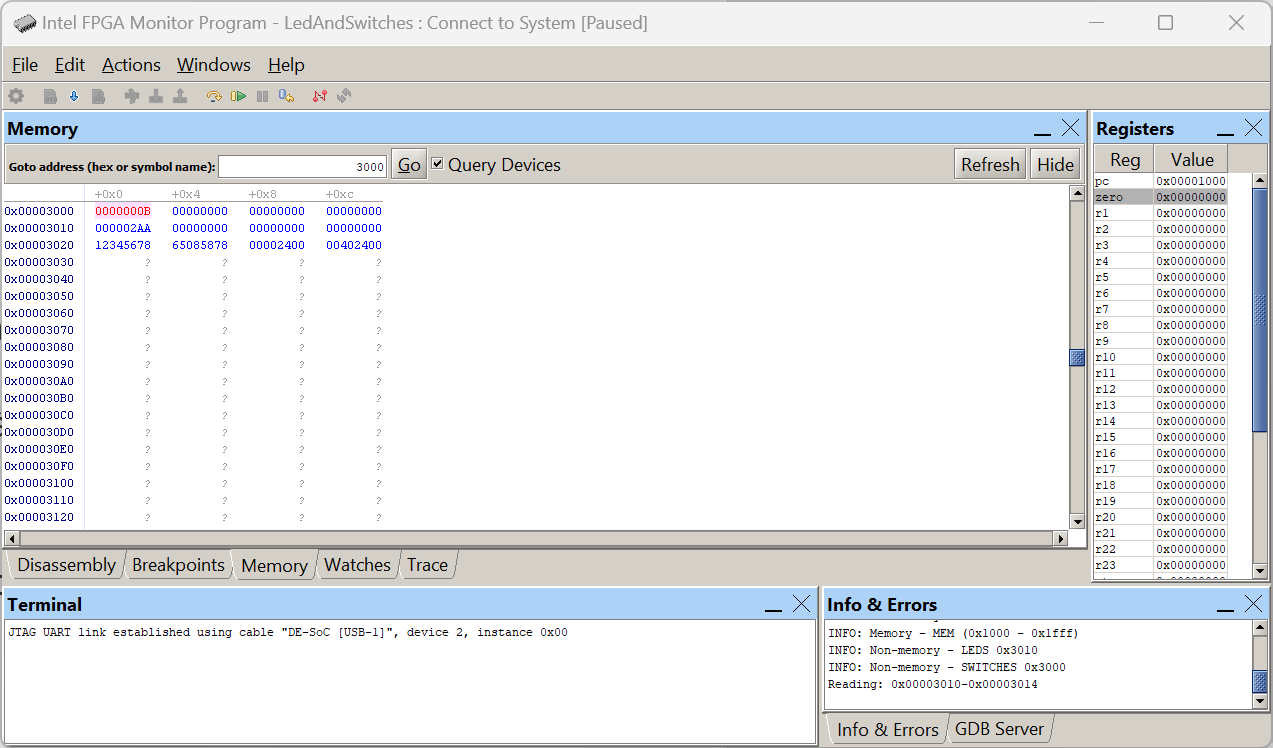
- by writing some data (e.g. 0x2AA to the relative location of the LEDs) you will see the LEDs light up alternately)

- by activating a configuration on the switches and clicking <Refresh> you will see this configuration mapped in memory

- the SysID location will be mapped to the value set via Hardware during the architecture definition phase

- By pressing the Reset button (Key[3]) the LEDs will turn off

The values changed since the previous reading are shown in red.

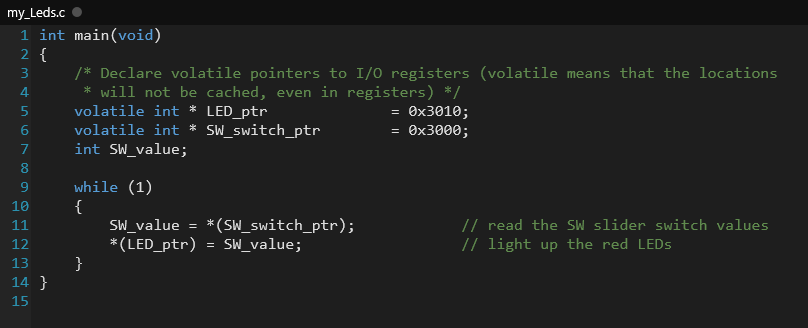


So far, the read and write operations to the peripherals have been done manually through the JTAG interface and the processor has not taken part in it.

# Software

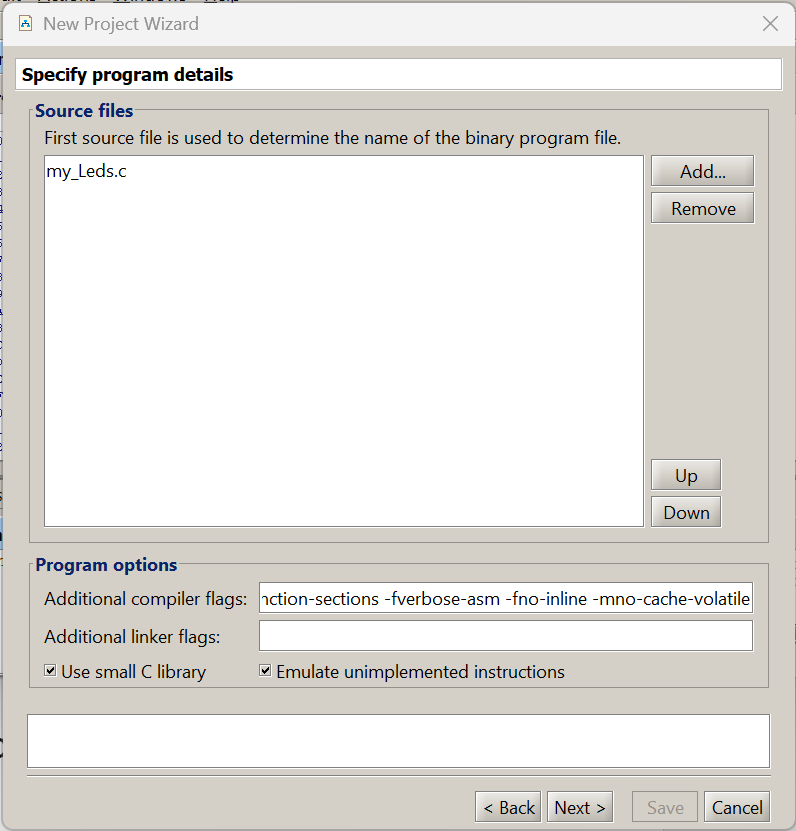
Now let's write a C program that by performing an infinite loop reads the data from the switches and copies them to the LEDs.

A good code might be the following



Obviously changing the locations of the pointers appropriately

Create a new project within the Intel FPGA Monitor Program, but this time at the tab for the programming type choose "C program" and at the next tab import the program previously written through a text editor and select "Use Small C Libraries"



The other tabs that characterize the software configuration of the project can remain unchanged.

Once the configuration is complete

Action > Compile & Load (F5)

The software is compiled and loaded into the processor's memory

Action > Continue (F3)

It starts the program that continuously reads the data from the swithes and copies them to the leds.

If you press the Reset button (KEY[3]) on the card, the cycle stops and the LEDs go out

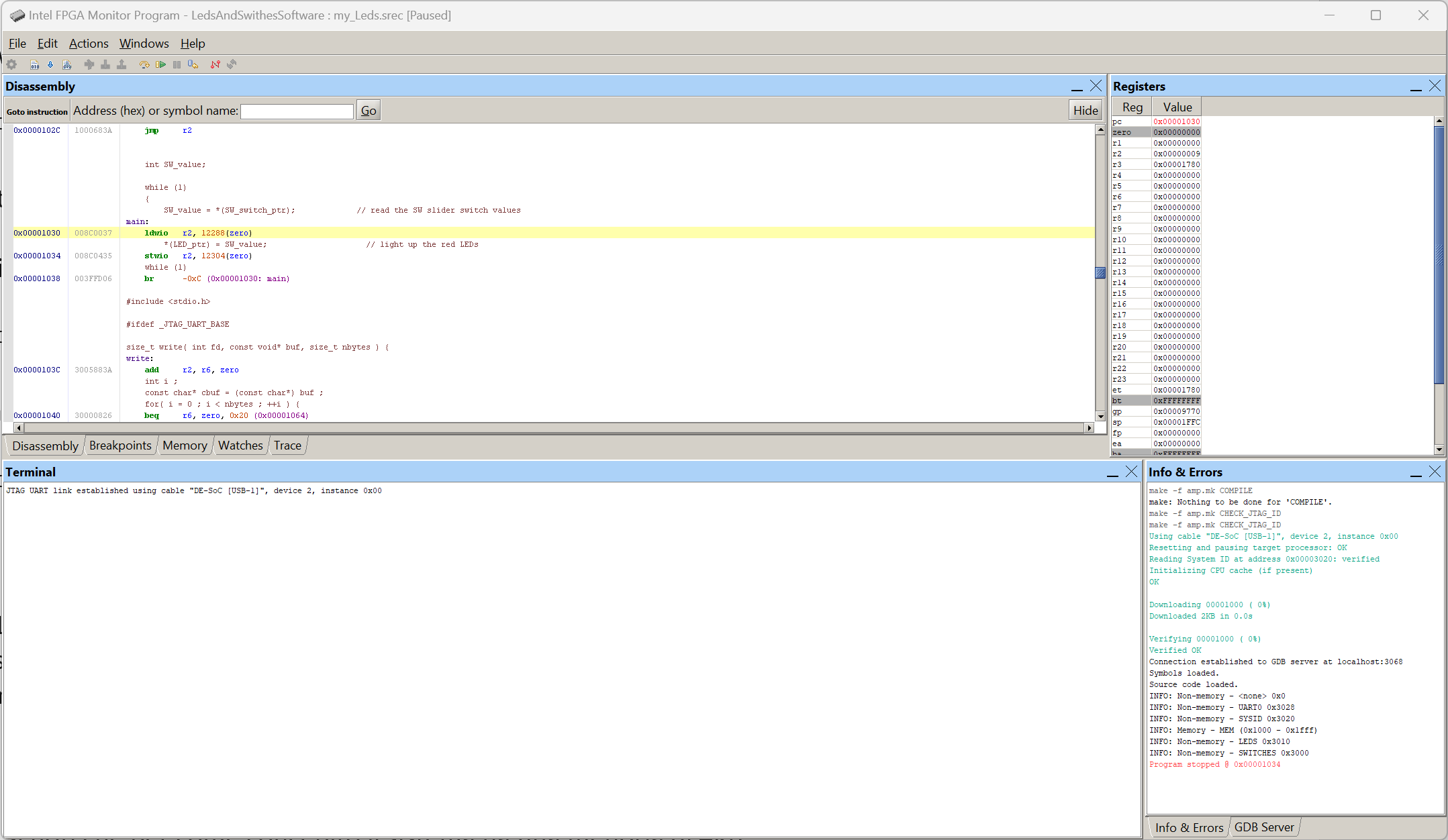
Action > Continue (F3)

Restart the system

Action > Stop (F4)

It stops the processor that no longer performs the prescribed cycle but does NOT reset the LEDs.

**NOTE1**: Interesting is the use of the "Disassembly" window which proposes the code executed by the processor in "Assembly" form and which can also be executed one step at a time by displaying the position within the code as well as the contents of all the registers.



**Tip**: Change the definition of pointers from "volatile int" to "int", what changes at the execution level ? and at the assembly code level ?

**NOTE2**: note that as the system is designed it has a memory of 4Kb, extremely small for any program, if you tried to write a code even slightly more complex than the present one or that required the use of libraries the system would NOT be able to support it. The size of the ON-Chip Memory could be increased, but the blocks made available inside the FPGA are still quite limited, so the only solution is to interface with external memories of the FPGA itself using appropriate interfaces.