German University in Cairo
Media Engineering and Technology
Assoc. Prof. Dr. Hassan Soubra

Embedded Systems, Winter Semester 2020
Practice Assignment 2

Discussion: 24/10/2020 - 29/10/2020

Exercise 2-1 Nios II

Objective:

In this tutorial, you will realize a system that lights two LEDs from two switches.
You will use the Altera FPGA board and Quartus II program.

It's not so easy to understand from the first time, but at the end you will be proud to achieve this really interesting tutorial.

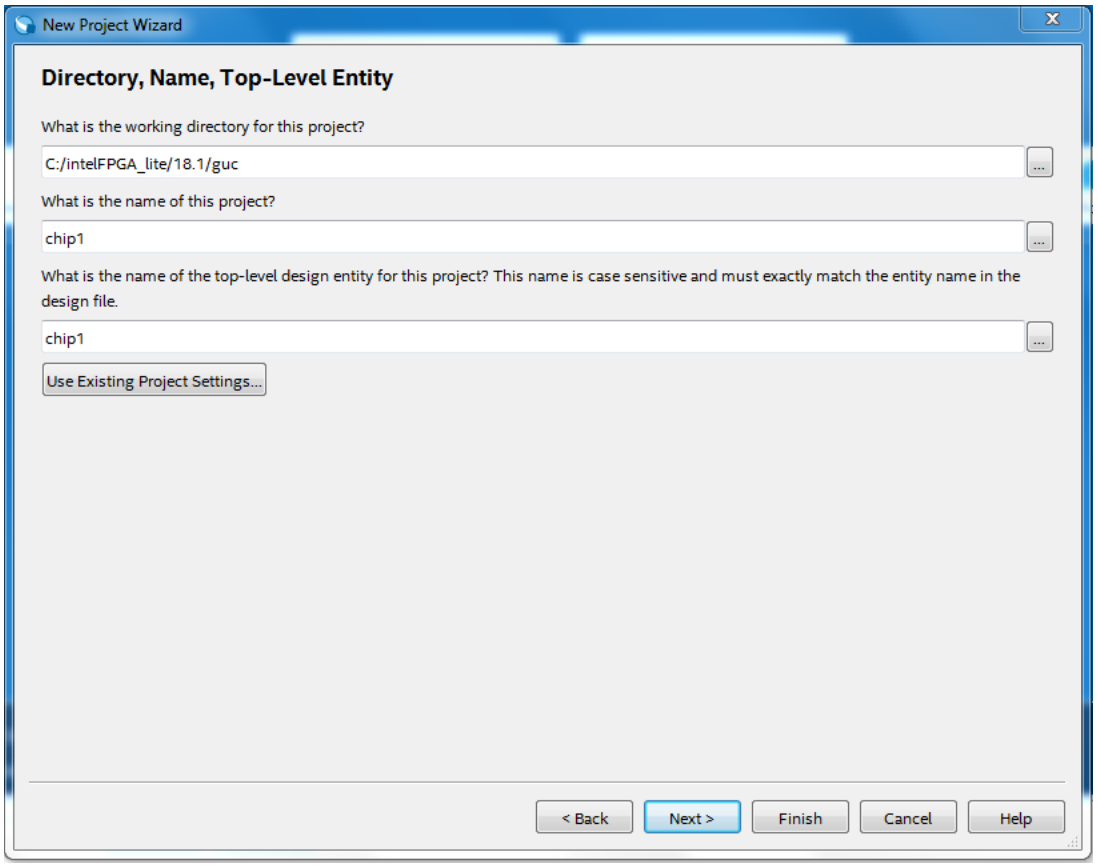
- This project is a mix between hardware and software.
- You're indeed going to choose which parts of hardware you want, realize the design and then create a software library from this design. You will be able to create a program in C language and upload this program into the FPGA board.
- For that, the design hardware will be created with the Platform Designer (Qsys) and a file will be generated, that's de1_blinker.sopcinfo.
- With this file you will be able to create the BSP (Board Support Package) that it will be compiled with our user application in C language.
- The user application and the BSP will be created with SBT (Software Build Tools).
- These tools are composed of:
  a) GCC (GNU Compiler Collection)
  b) A special Nios II GNU C library
  c) A HAL (Hardware Abstraction Layer)
- SBT is a tool based on Eclipse, a well-known open source IDE.
- You do not need to have Eclipse already installed because the Indigo version will be launched for you when you will decide to create your BSP. Indeed a hidden Eclipse was installed with your Quartus installation.
- After the compilation and the linkage of the user application and the BSP, a new file will be added: de1_blinker.elf. At this point, you will send it to the board in order to program it.
- Finally you will see the result directly on the board with red LEDs lit from switches.

Step 1: New Project from Quartus:

a) Quartus → File → New Project Wizard

b) Next

- Project Name: **chip1**
- Project Top-level Entity: **chip1**



Directory, Name, Top-Level Entity

What is the working directory for this project?

C:/intelFPGA_lite/18.1/guc

What is the name of this project?

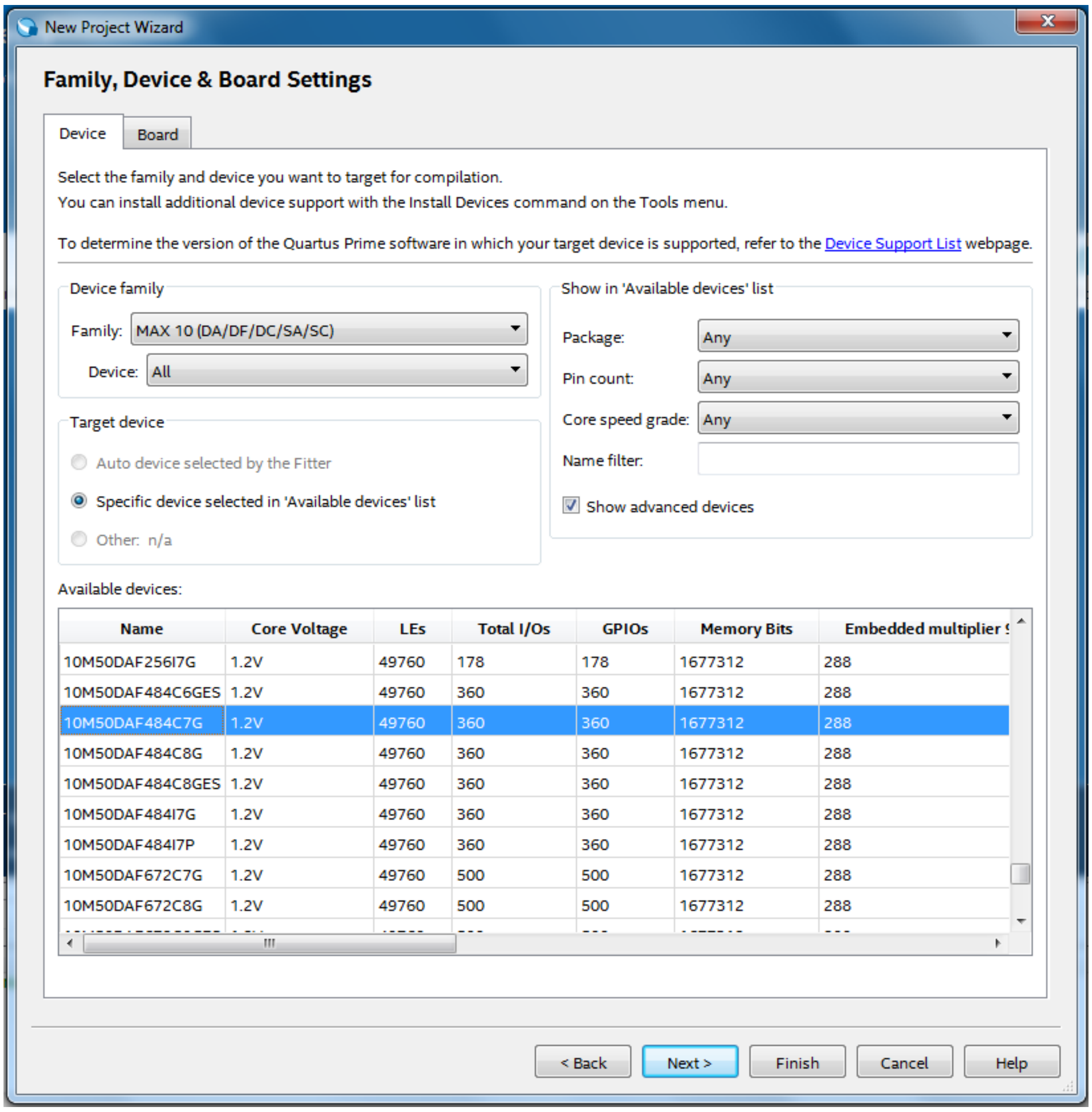
chip1

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

chip1

c) Next → Next → Next

- Family: **MAX 10 (DA/DF/DC/SA/SC)**
- Available devices: **10M50DAF484C7G**


New Project Wizard

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.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family

Family: MAX 10 (DA/DF/DC/SA/SC)

Device: All

Target device

( ) Auto device selected by the Fitter

(●) Specific device selected in 'Available devices' list

( ) Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter:

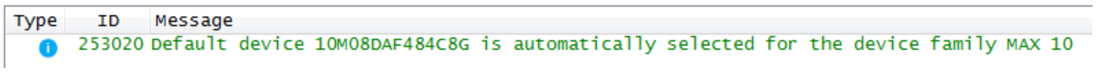
☑ Show advanced devices

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9
10M50DAF256I7G	1.2V	49760	178	178	1677312	288
10M50DAF484C6GES	1.2V	49760	360	360	1677312	288
10M50DAF484C7G	1.2V	49760	360	360	1677312	288
10M50DAF484C8G	1.2V	49760	360	360	1677312	288
10M50DAF484C8GES	1.2V	49760	360	360	1677312	288
10M50DAF484I7G	1.2V	49760	360	360	1677312	288
10M50DAF484I7P	1.2V	49760	360	360	1677312	288
10M50DAF672C7G	1.2V	49760	500	500	1677312	288
10M50DAF672C8G	1.2V	49760	500	500	1677312	288

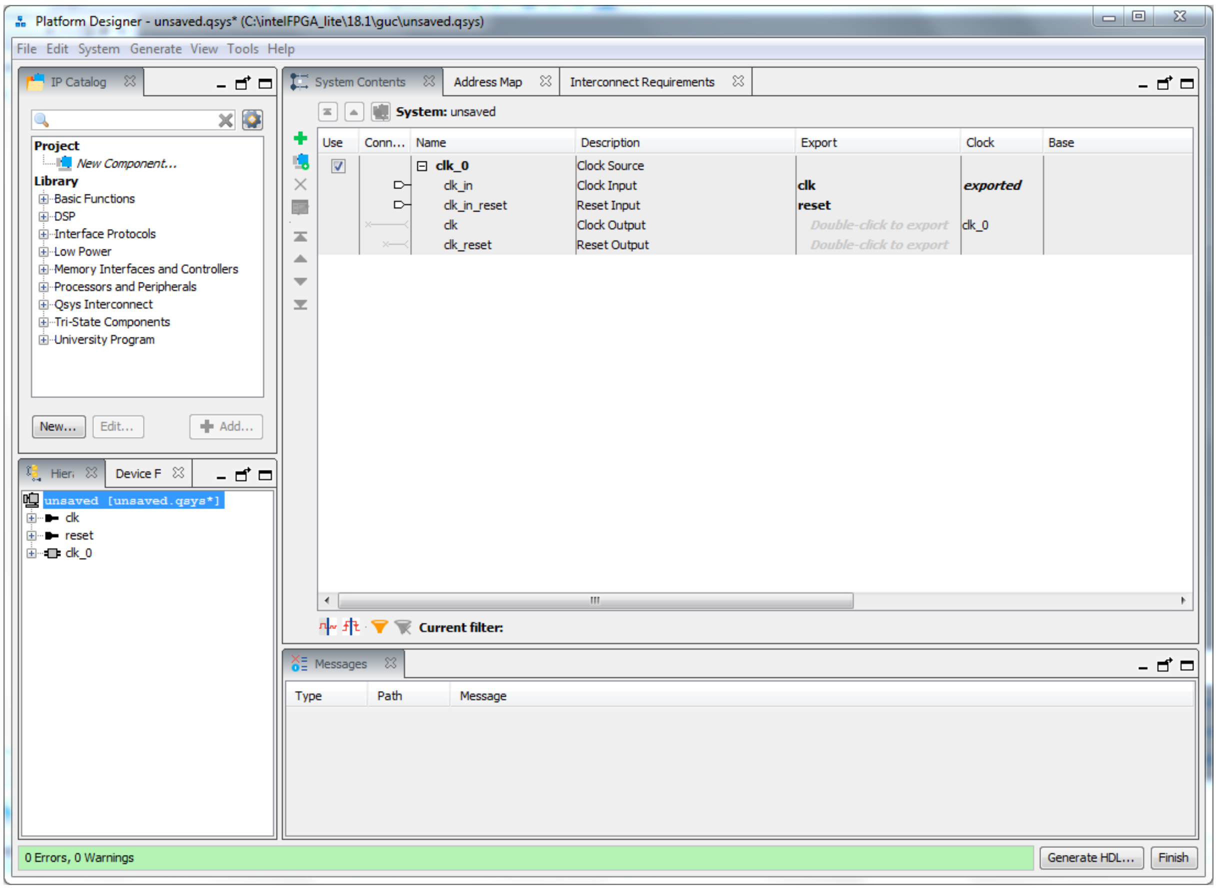
< Back | Next > | Finish | Cancel | Help

d) Next → Next → Finish


```
Type   ID     Message
       253020 Default device 10M08DAF484C8G is automatically selected for the device family MAX 10
```

Step 2: Creating Components with Platform Designer:

a) Quartus → Tools → Platform Designer

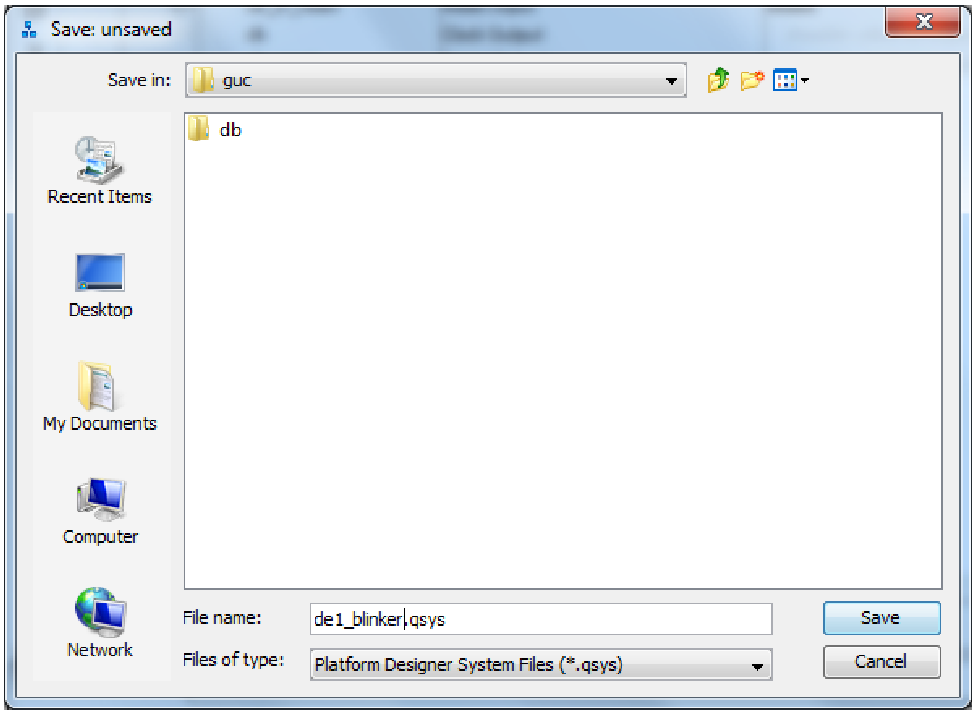


Use	Conn...	Name	Description	Export	Clock	Base
✓		clk_0	Clock Source			
		clk_in	Clock Input	clk	exported	
		clk_in_reset	Reset Input	reset		
		clk	Clock Output	Double-click to export	clk_0	
		clk_reset	Reset Output	Double-click to export		

Type	Path	Message

b) File → Save

- File Name: **de1_blinker.qsys**.


Save: unsaved

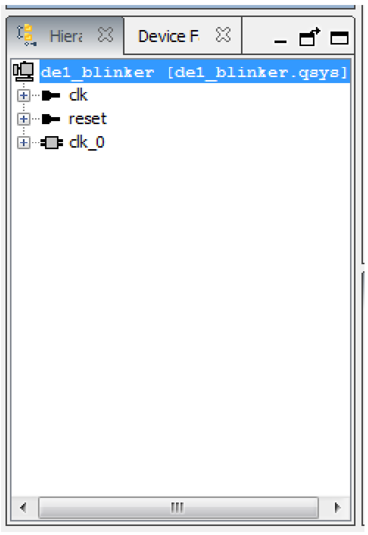
Save in: guc

db

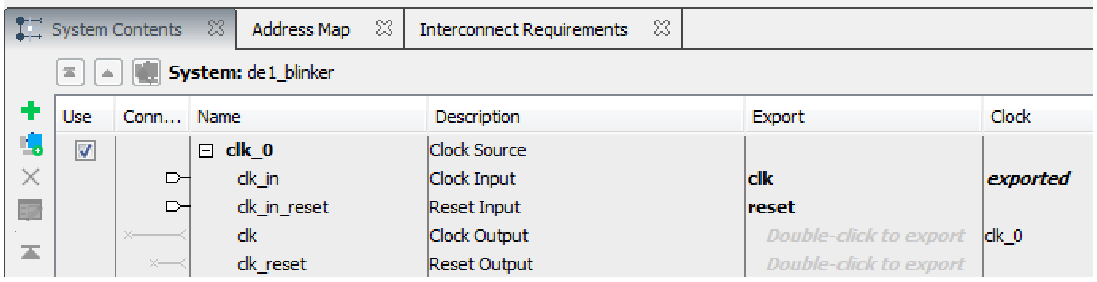
File name: de1_blinker.qsys

Files of type: Platform Designer System Files (*.qsys)

c) In your project directory you can now see a new file: **de1_blinker.qsys**.


```
de1_blinker [de1_blinker.qsys]
  clk
  reset
  clk_0
```

d) On the right you can see the **clk_0** clock in the **System Contents** tab.



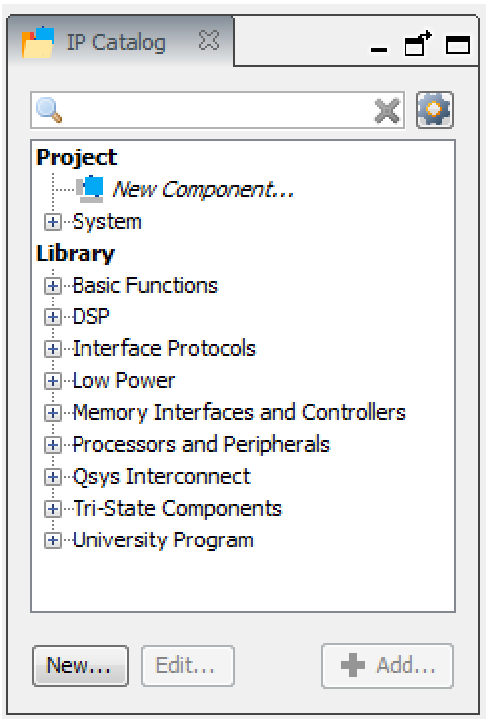
System: de1_blinker

Use	Conn...	Name	Description	Export	Clock
✓		clk_0	Clock Source		
		clk_in	Clock Input	clk	exported
		clk_in_reset	Reset Input	reset	
		clk	Clock Output	Double-click to export	clk_0
		clk_reset	Reset Output	Double-click to export	

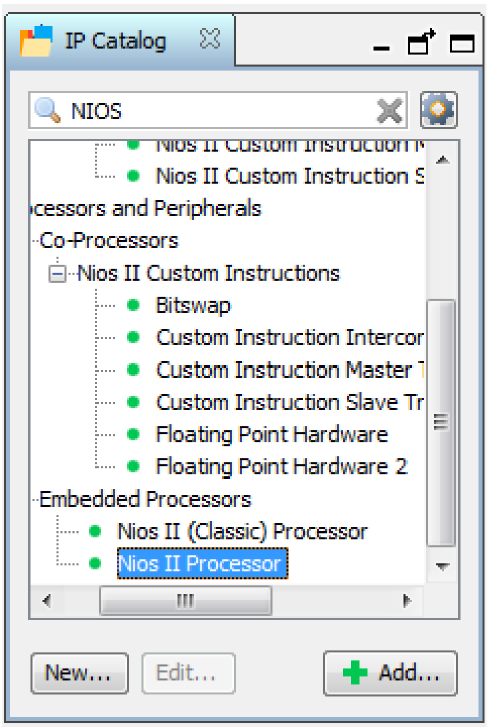
e) Right click on **clk_0** → Rename → Rename it to **clock_main**

Step 3: Adding Nios II Processor:

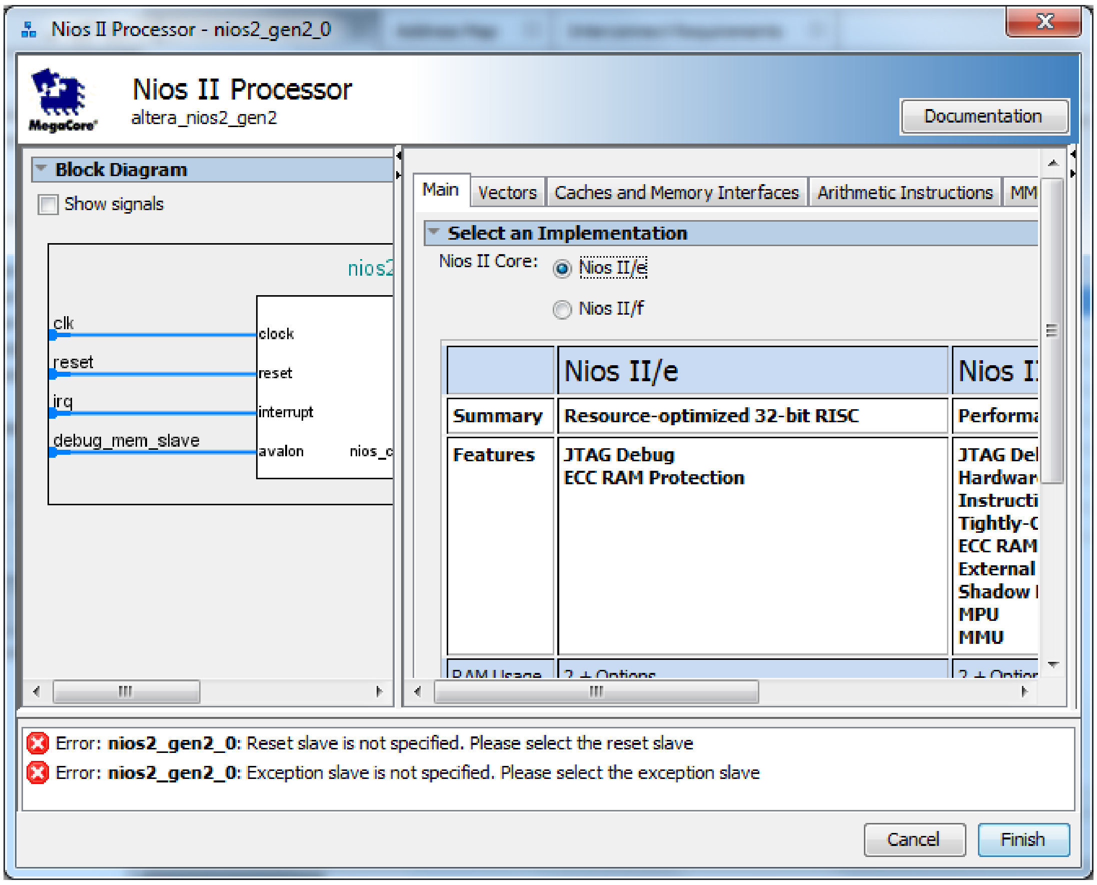
a) Look for the IP catalog tab in the top left of the Platform Designer (Qsys) window. Below the IP catalog tab, you can search for the various components you want to add to your Platform Designer (Qsys) based system.



b) Enter **'Nios'** in the search tab and select the **Nios II Processor** (not the Classic Nios II) from the library be double clicking.


IP Catalog
NIOS
Nios II Custom Instruction S
cessors and Peripherals
Co-Processors
Nios II Custom Instructions
Bitswap
Custom Instruction Intercor
Custom Instruction Master
Custom Instruction Slave Tr
Floating Point Hardware
Floating Point Hardware 2
Embedded Processors
Nios II (Classic) Processor
Nios II Processor
New...
Edit...
Add...

c) A configuration window will appear, in this select the **Nios II/e** processor. The 'e' stands for economy and the 'f' stands for fast. You will use the **economy** version in this tutorial.



Nios II Processor - nios2_gen2_0

Nios II Processor
altera_nios2_gen2

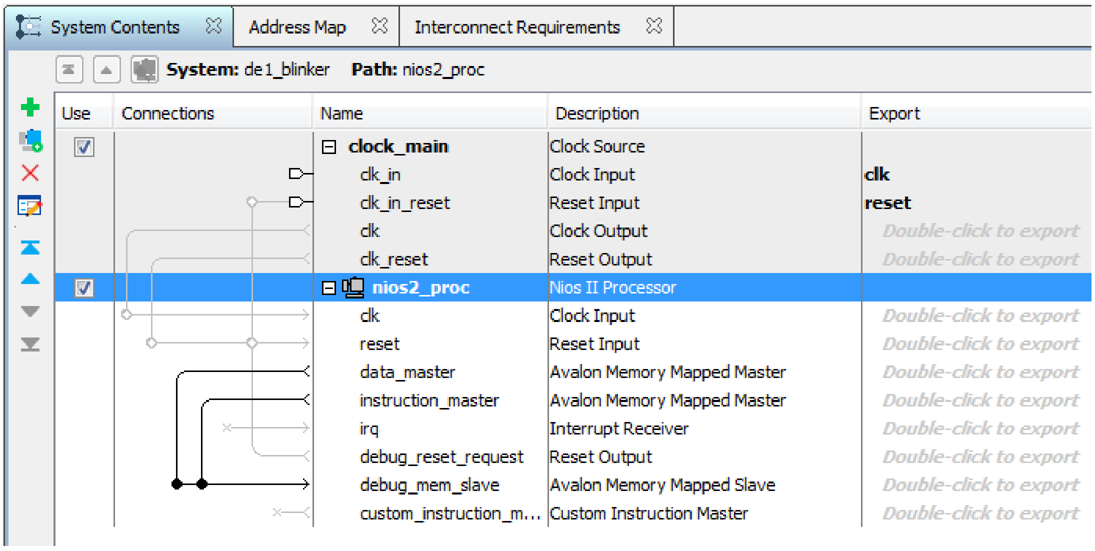
Select an Implementation

Nios II Core: (●) Nios II/e ( ) Nios II/f

	Nios II/e	Nios I
Summary	Resource-optimized 32-bit RISC	Performa
Features	JTAG Debug ECC RAM Protection	JTAG Deb Hardwar Instructi Tightly-C ECC RAM External Shadow I MPU MMU
RAM Usage	2 + Options	2 + Optior

Error: **nios2_gen2_0**: Reset slave is not specified. Please select the reset slave
Error: **nios2_gen2_0**: Exception slave is not specified. Please select the exception slave

d) The Nios II core is added on the **System Contents** with the name **nios2_gen_0**. Let's rename it to **nios2_proc** by right clicking on the name, and selecting Rename.



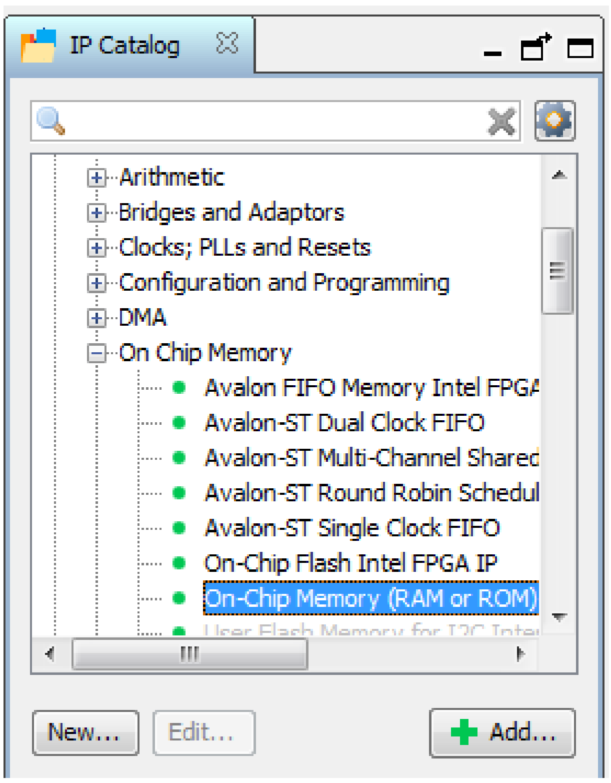
System: de1_blinker **Path:** nios2_proc

Use	Connections	Name	Description	Export
✓		clock_main	Clock Source	
		clk_in	Clock Input	clk
		clk_in_reset	Reset Input	reset
		clk	Clock Output	Double-click to export
		clk_reset	Reset Output	Double-click to export
✓		nios2_proc	Nios II Processor	
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		data_master	Avalon Memory Mapped Master	Double-click to export
		instruction_master	Avalon Memory Mapped Master	Double-click to export
		irq	Interrupt Receiver	Double-click to export
		debug_reset_request	Reset Output	Double-click to export
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export
		custom_instruction_m...	Custom Instruction Master	Double-click to export

e) You can see that the **data_master** and **instruction_master** are already linked to **debug_mem_slave** with a black line passing by a tiny black round.
For now don't worry about the system errors reported.

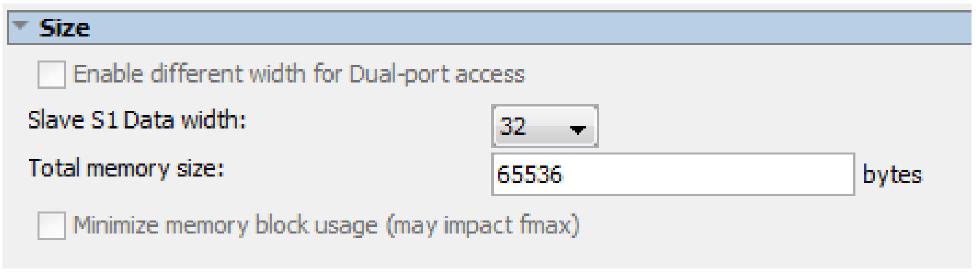
Step 4: Adding a Memory:

a) IP Catalog → Library → Basic Functions → On Chip Memory → On-Chip Memory (RAM or ROM) → Add


IP Catalog
Arithmetic
Bridges and Adaptors
Clocks; PLLs and Resets
Configuration and Programming
DMA
On Chip Memory
Avalon FIFO Memory Intel FPGA
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared
Avalon-ST Round Robin Schedul
Avalon-ST Single Clock FIFO
On-Chip Flash Intel FPGA IP
On-Chip Memory (RAM or ROM)
New...
Edit...
Add...

b) The On-Chip Memory (RAM or ROM)'s module window appears and in the Size section, you will find the total memory size is 4096 bytes.

c) This size is a bit short for our first example, so let's replace it by **65536**.


Size

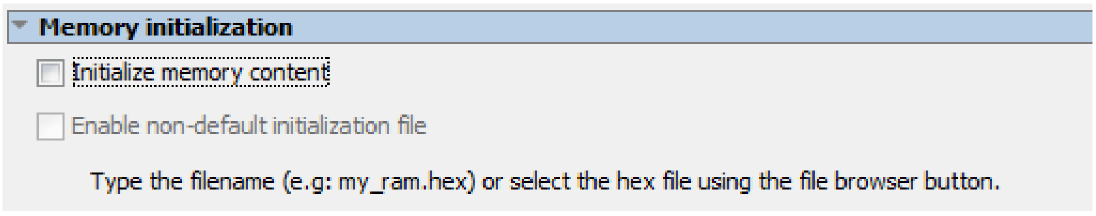
Enable different width for Dual-port access

Slave S1 Data width: 32

Total memory size: 65536 bytes

Minimize memory block usage (may impact fmax)

d) Uncheck **'Initialize memory content'**. This feature includes the software executable in the hardware image. If you do not, you will have a bug when trying to upload to the FPGA.



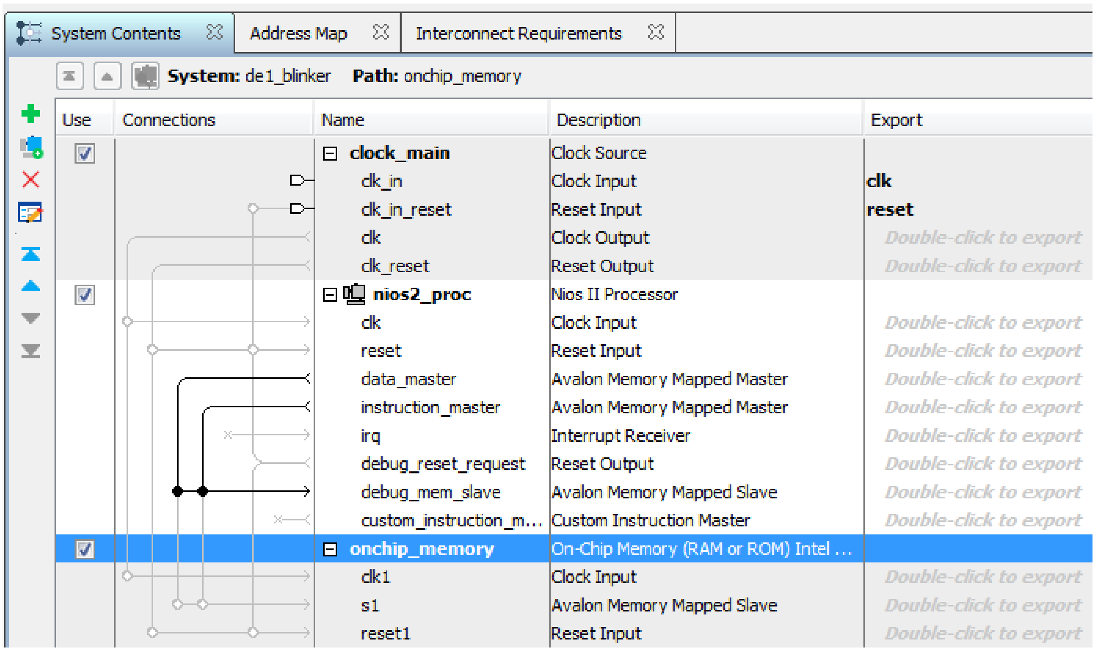
Memory initialization

☐ Initialize memory content

☐ Enable non-default initialization file

Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.

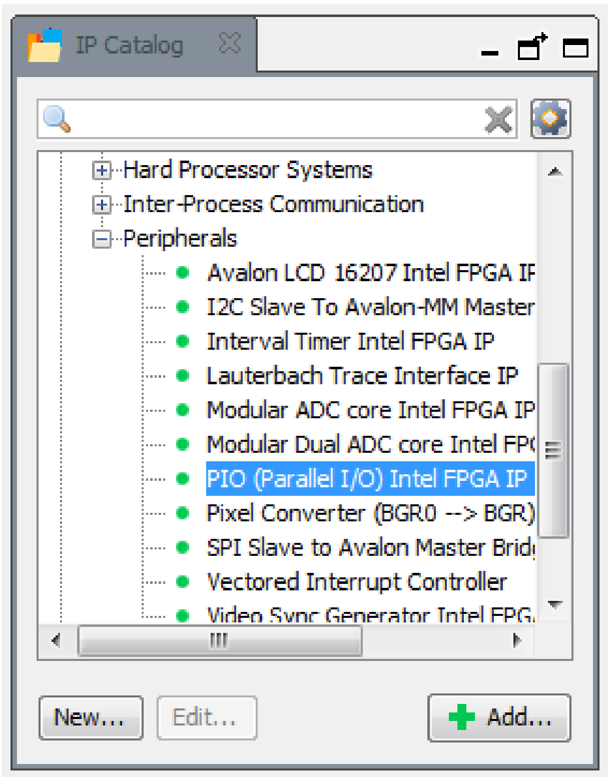
e) Click Finish. The **onchip_memory2_0** is added to the **System Contents**. Rename it to **onchip_memory**.



Use	Connections	Name	Description	Export
✓		clock_main	Clock Source	
		clk_in	Clock Input	clk
		clk_in_reset	Reset Input	reset
		clk	Clock Output	Double-click to export
		clk_reset	Reset Output	Double-click to export
✓		nios2_proc	Nios II Processor	
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		data_master	Avalon Memory Mapped Master	Double-click to export
		instruction_master	Avalon Memory Mapped Master	Double-click to export
		irq	Interrupt Receiver	Double-click to export
		debug_reset_request	Reset Output	Double-click to export
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export
		custom_instruction_m...	Custom Instruction Master	Double-click to export
✓		onchip_memory	On-Chip Memory (RAM or ROM) Intel ...	
		clk1	Clock Input	Double-click to export
		s1	Avalon Memory Mapped Slave	Double-click to export
		reset1	Reset Input	Double-click to export

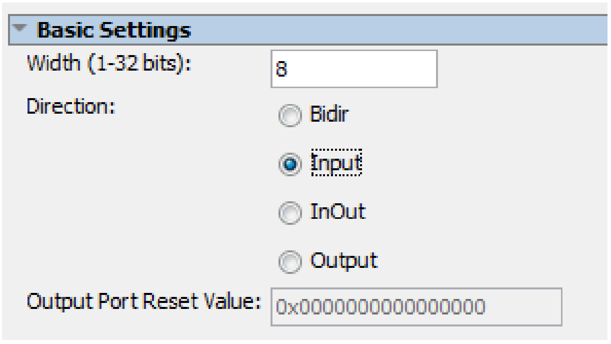
Step 5: Adding a First Parallel I/O (PIO) for Switches:

a) IP Catalog → Library → Processors and Peripherals → Peripherals → PIO (Parallel I/O) → Add


IP Catalog
Hard Processor Systems
Inter-Process Communication
Peripherals
Avalon LCD 16207 Intel FPGA IP
I2C Slave To Avalon-MM Master
Interval Timer Intel FPGA IP
Lauterbach Trace Interface IP
Modular ADC core Intel FPGA IP
Modular Dual ADC core Intel FP
PIO (Parallel I/O) Intel FPGA IP
Pixel Converter (BGR0 --> BGR)
SPI Slave to Avalon Master Brid
Vectored Interrupt Controller
Video Sync Generator Intel FPG
New...
Edit...
Add...

b) In the **Basic Settings** → **Width (1-32 bits)** → there is already **8** written. Even if you need only 2 switches, let it like that. It will allows us to show how to assign location to the pins later.

c) Still from the **Basic Settings** → **Direction** → select **Input** → **Finish**.

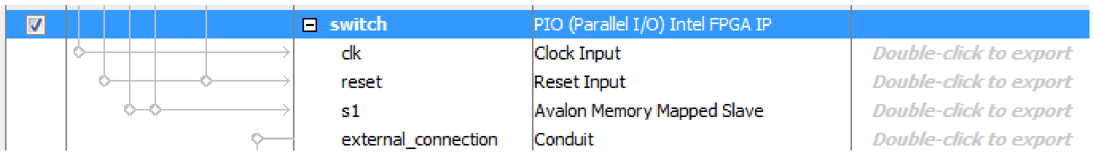


Basic Settings

Width (1-32 bits):	8
Direction:	Bidir [✓ Input] InOut Output
Output Port Reset Value:	0x0000000000000000

d) The PIO (Parallel I/O) module input has just been added to the System Contents.

e) Rename it from **pio_0** to **switch**.



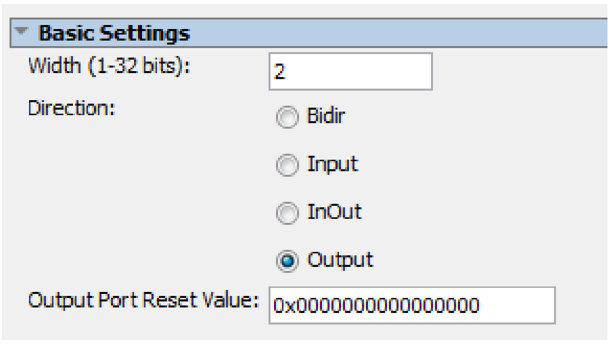
⊟ switch	PIO (Parallel I/O) Intel FPGA IP	
clk	Clock Input	Double-click to export
reset	Reset Input	Double-click to export
s1	Avalon Memory Mapped Slave	Double-click to export
external_connection	Conduit	Double-click to export

Step 6: Adding a Second Parallel I/O (PIO) for LEDs:

a) IP Catalog → Library → Processors and Peripherals → Peripherals → PIO (Parallel I/O) → Add - *Same steps as part (a) in Step 5*

b) In the **Basic Settings** → **Width (1-32 bits)** → there is already **8** written. This time change it to **2**.

c) Still from the **Basic Settings** → **Direction** → select **Output** → **Finish**.



Basic Settings

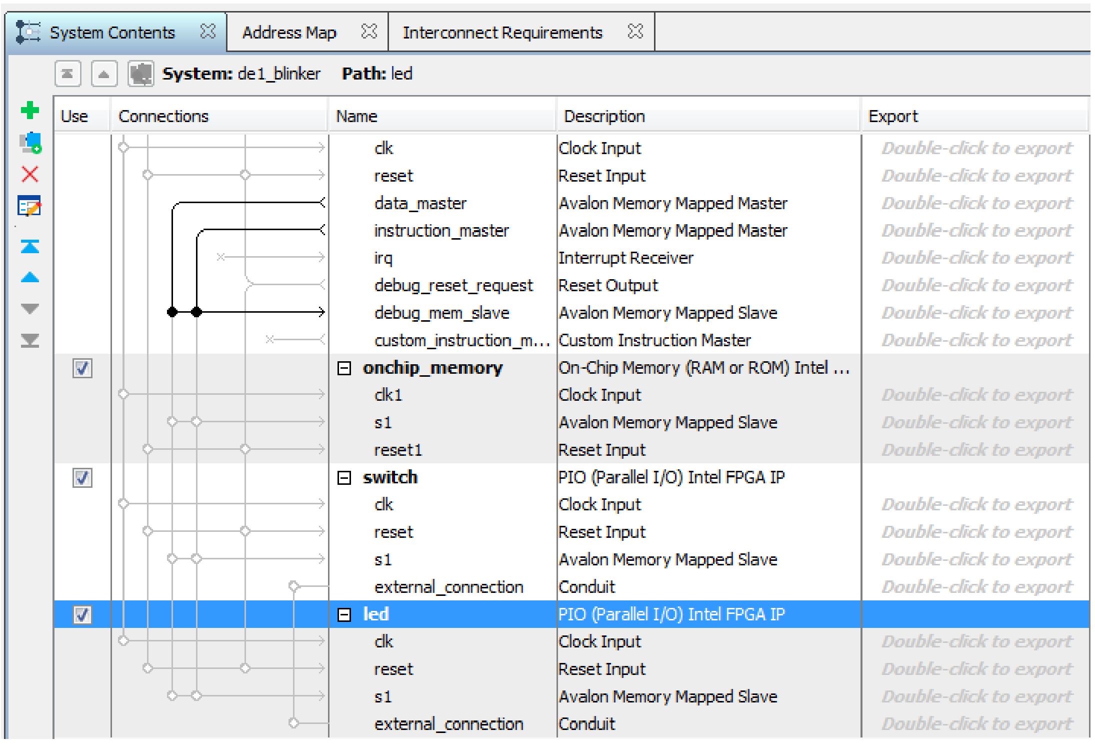
Width (1-32 bits): 2

Direction: Bidir Input InOut [✓ Output]

Output Port Reset Value: 0x0000000000000000

d) The PIO (Parallel I/O) module output has just been added to the System Contents.

e) Rename it from **pio_0** to **led**.



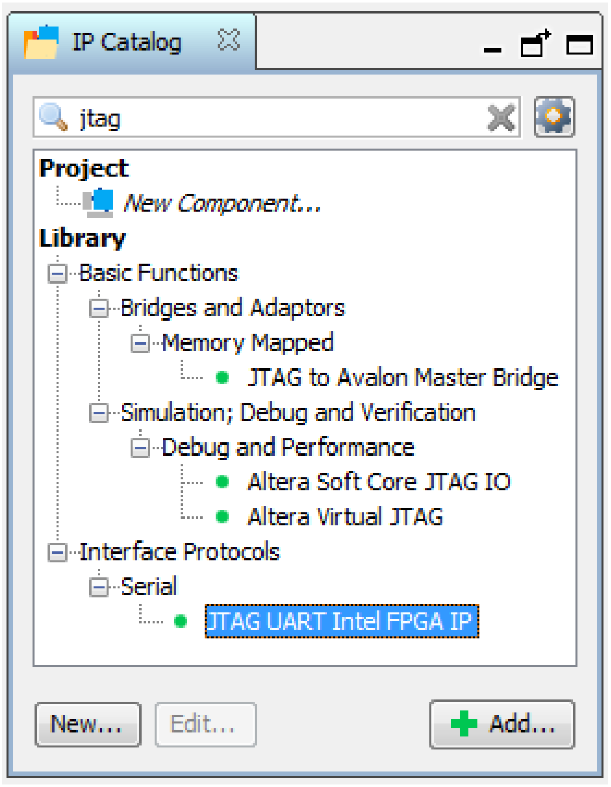
System Contents | Address Map | Interconnect Requirements

System: de1_blinker **Path:** led

Use	Connections	Name	Description	Export
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		data_master	Avalon Memory Mapped Master	Double-click to export
		instruction_master	Avalon Memory Mapped Master	Double-click to export
		irq	Interrupt Receiver	Double-click to export
		debug_reset_request	Reset Output	Double-click to export
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export
		custom_instruction_m...	Custom Instruction Master	Double-click to export
✓		⊟ **onchip_memory**	On-Chip Memory (RAM or ROM) Intel ...	
		clk1	Clock Input	Double-click to export
		s1	Avalon Memory Mapped Slave	Double-click to export
		reset1	Reset Input	Double-click to export
✓		⊟ **switch**	PIO (Parallel I/O) Intel FPGA IP	
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		s1	Avalon Memory Mapped Slave	Double-click to export
		external_connection	Conduit	Double-click to export
✓		⊟ **led**	PIO (Parallel I/O) Intel FPGA IP	
		clk	Clock Input	Double-click to export
		reset	Reset Input	Double-click to export
		s1	Avalon Memory Mapped Slave	Double-click to export
		external_connection	Conduit	Double-click to export

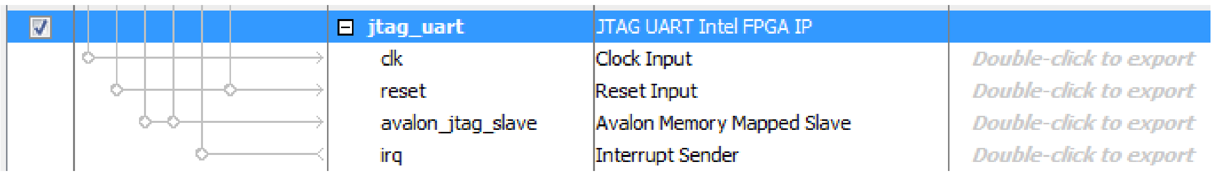
Step 7: Adding JTAG UART Component:

a) Search for **'JTAG'** in the IP Catalog, locate the **JTAG UART** → Add.



b) Keep the default settings and click **Finish**.

c) Rename it from **jtag_uart_0** to **jtag_uart**.



☑	jtag_uart	JTAG UART Intel FPGA IP	
	clk	Clock Input	Double-click to export
	reset	Reset Input	Double-click to export
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export
	irq	Interrupt Sender	Double-click to export

Step 8: Connecting the Qsys System Components Together:

a) Connect the **clk_main.clk** with each components' clocks.

- clk_main.clk → nios2_proc.clk
- clk_main.clk → onchip_memory.clk1
- clk_main.clk → switch.clk
- clk_main.clk → led.clk
- clk_main.clk → jtag_uart.clk

b) Connect the **clk_main.clk_reset** with each components' resets inputs.

- clk_main.clk_reset → nios2_proc.reset
- clk_main.clk_reset → onchip_memory.reset1
- clk_main.clk_reset → switch.reset
- clk_main.clk_reset → led.reset
- clk_main.clk_reset → jtag_uart.reset

c) Connect the **nios2_proc.data_master** with the following:

- nios2_proc.data_master → onchip_memory.s1
- nios2_proc.data_master → switch.s1
- nios2_proc.data_master → led.s1
- nios2_proc.data_master → jtag_uart.avalon_jtag_slave

- *The nios2_proc.data_master is already connected to nios2_proc.debug_memory_slave*

d) Connect the **nios2_proc.instruction_master** with the following:

- nios2_proc.instruction_master → onchip_memory.s1

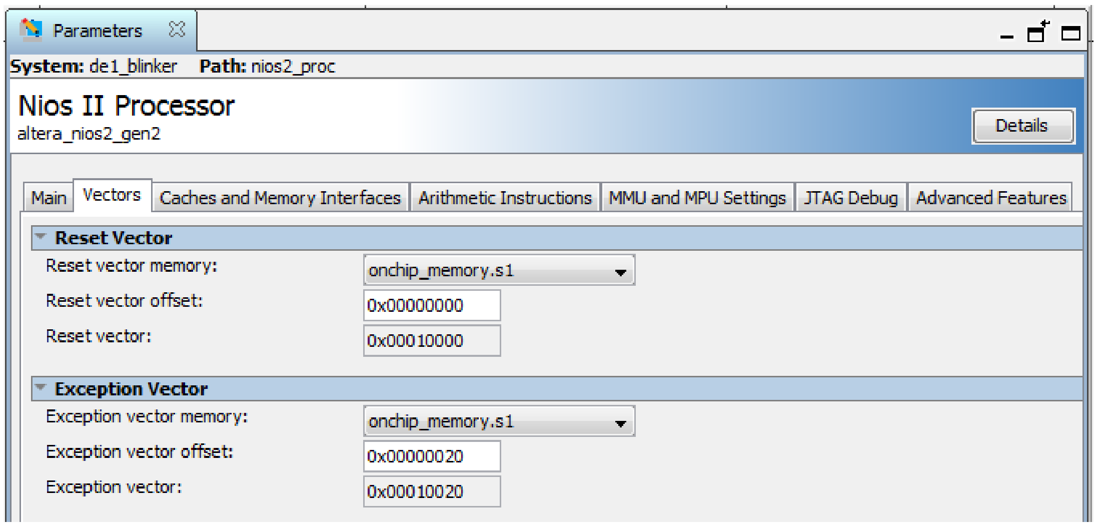
- *The nios2_proc.instruction_master is already connected to nios2_proc.debug_memory_slave*

e) Let's assign base addresses. From **System** → **Assign Base Addresses**.

f) Double click on **nios2_proc** from the **System Contents** tab.

g) **Vectors** → **Reset Vector** → **Reset vector memory** → Select **onchip_memory.s1**.

h) **Vectors** → **Exception Vector** → **Exception vector memory** → Select **onchip_memory.s1**.



System: de1_blinker **Path:** nios2_proc

Nios II Processor
altera_nios2_gen2

Reset Vector

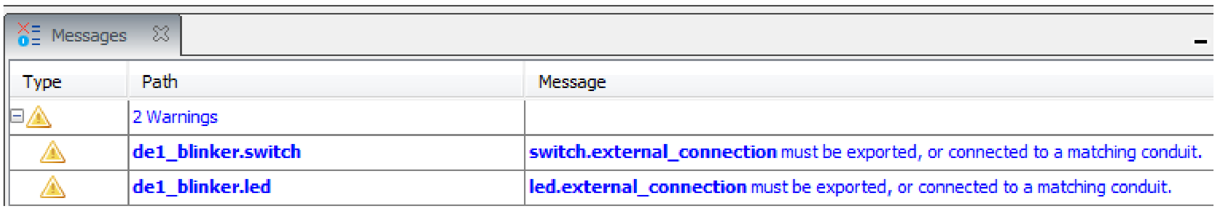
Reset vector memory: onchip_memory.s1
Reset vector offset: 0x00000000
Reset vector: 0x00010000

Exception Vector

Exception vector memory: onchip_memory.s1
Exception vector offset: 0x00000020
Exception vector: 0x00010020

i) To finish the system, you have to add connections to external connections.

j) As you can see the **switch.external_connection** is in a not defined state.


Messages

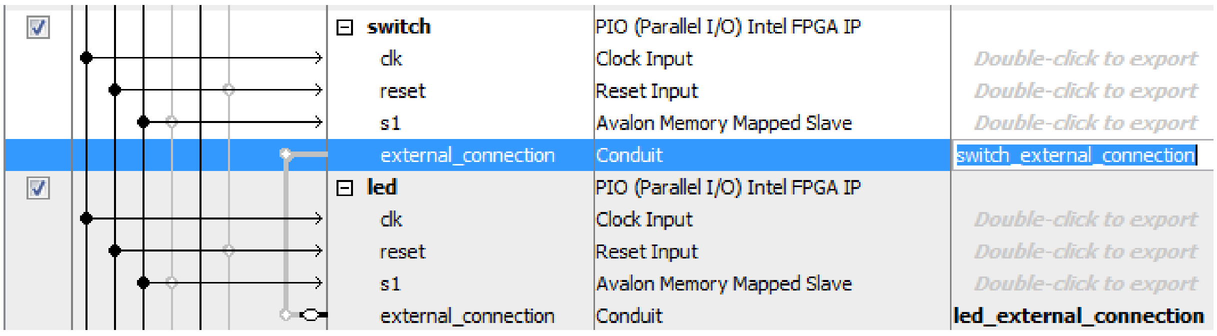
Type	Path	Message
⚠	2 Warnings	
⚠	**de1_blinker.switch**	**switch.external_connection** must be exported, or connected to a matching conduit.
⚠	**de1_blinker.led**	**led.external_connection** must be exported, or connected to a matching conduit.

k) From the **System Contents** tab → the **switch** section → **external_connection** → go to **Export** column → click on **Double-click to export**

l) The new **Export** becomes **switch_external_connection**.

m) Do the same for the **led** → **external_connection** in the **Export** column.

n) The new **Export** becomes **led_external_connection**.

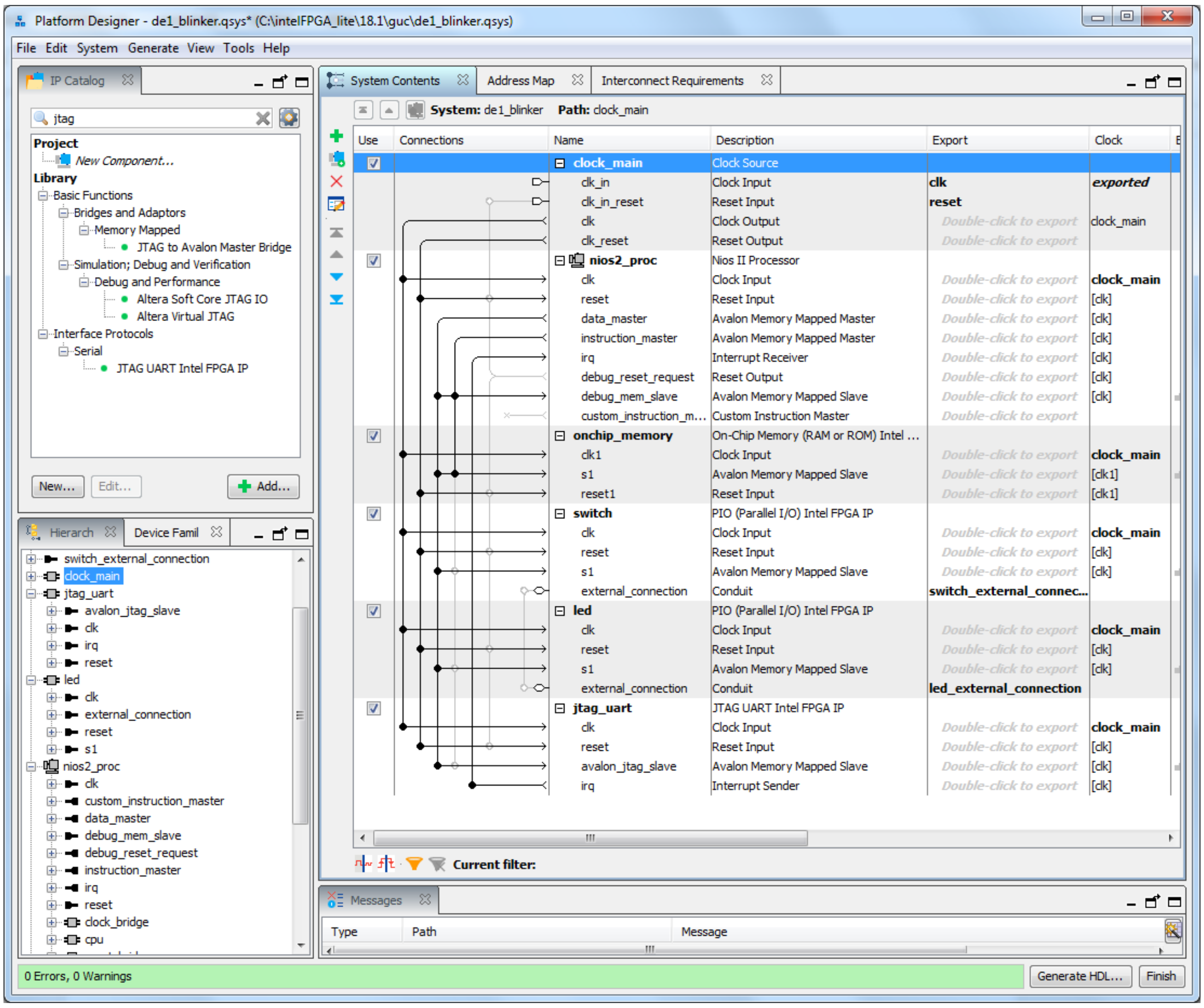


☑ **switch**	PIO (Parallel I/O) Intel FPGA IP	
clk	Clock Input	*Double-click to export*
reset	Reset Input	*Double-click to export*
s1	Avalon Memory Mapped Slave	*Double-click to export*
external_connection	Conduit	switch_external_connection
☑ **led**	PIO (Parallel I/O) Intel FPGA IP	
clk	Clock Input	*Double-click to export*
reset	Reset Input	*Double-click to export*
s1	Avalon Memory Mapped Slave	*Double-click to export*
external_connection	Conduit	**led_external_connection**

o) Connect the **nios2_proc.irq** with the following:

- nios2_proc.irq → jtag_uart.irq

p) You are done for the clicking black rounds part. You should have something that looks like this:


Platform Designer - de1_blinker.qsys* (C:\intelFPGA_lite\18.1\guc\de1_blinker.qsys)
File Edit System Generate View Tools Help
IP Catalog
System Contents
Address Map
Interconnect Requirements
New... Edit... Add...
Hierarch
Device Famil
Messages
Current filter:
Generate HDL...
Finish

jtag

Project
- New Component...

Library
- Basic Functions
  - Bridges and Adaptors
    - Memory Mapped
      - JTAG to Avalon Master Bridge
  - Simulation; Debug and Verification
    - Debug and Performance
      - Altera Soft Core JTAG IO
      - Altera Virtual JTAG
- Interface Protocols
  - Serial
    - JTAG UART Intel FPGA IP

- switch_external_connection
- clock_main
- jtag_uart
  - avalon_jtag_slave
  - clk
  - irq
  - reset
- led
  - clk
  - external_connection
  - reset
  - s1
- nios2_proc
  - clk
  - custom_instruction_master
  - data_master
  - debug_mem_slave
  - debug_reset_request
  - instruction_master
  - irq
  - reset
  - clock_bridge
  - cpu

System: de1_blinker Path: clock_main

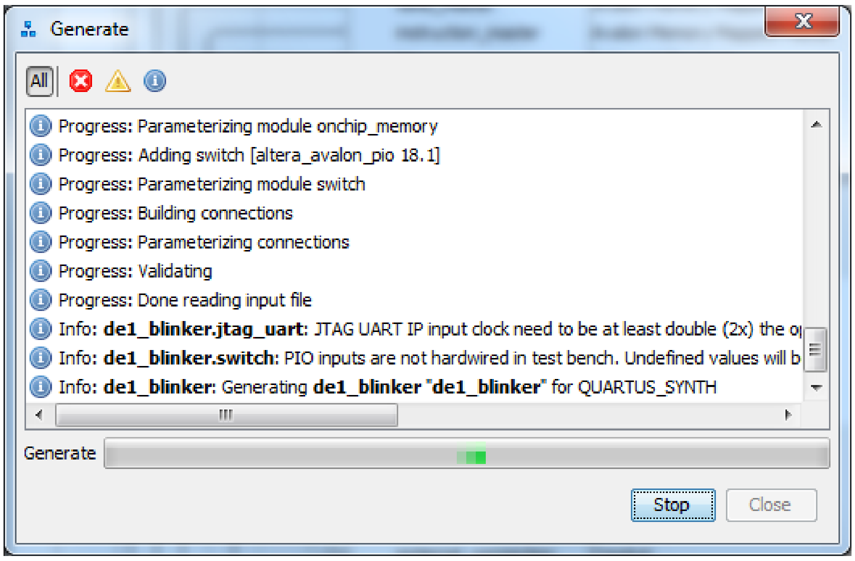
Use	Name	Description	Export	Clock
✓	**clock_main**	Clock Source		
	clk_in	Clock Input	**clk**	***exported***
	clk_in_reset	Reset Input	**reset**	
	clk	Clock Output	*Double-click to export*	clock_main
	clk_reset	Reset Output	*Double-click to export*	
✓	**nios2_proc**	Nios II Processor		
	clk	Clock Input	*Double-click to export*	**clock_main**
	reset	Reset Input	*Double-click to export*	[clk]
	data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]
	instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]
	irq	Interrupt Receiver	*Double-click to export*	[clk]
	debug_reset_request	Reset Output	*Double-click to export*	[clk]
	debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	custom_instruction_m...	Custom Instruction Master	*Double-click to export*	
✓	**onchip_memory**	On-Chip Memory (RAM or ROM) Intel ...		
	clk1	Clock Input	*Double-click to export*	**clock_main**
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]
	reset1	Reset Input	*Double-click to export*	[clk1]
✓	**switch**	PIO (Parallel I/O) Intel FPGA IP		
	clk	Clock Input	*Double-click to export*	**clock_main**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	external_connection	Conduit	**switch_external_connec...**	
✓	**led**	PIO (Parallel I/O) Intel FPGA IP		
	clk	Clock Input	*Double-click to export*	**clock_main**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	external_connection	Conduit	**led_external_connection**	
✓	**jtag_uart**	JTAG UART Intel FPGA IP		
	clk	Clock Input	*Double-click to export*	**clock_main**
	reset	Reset Input	*Double-click to export*	[clk]
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	irq	Interrupt Sender	*Double-click to export*	[clk]

Type	Path	Message

0 Errors, 0 Warnings

Step 9: Generating HDL Code:

a) **Generate** → **Generate HDL** → Click on the **Generate** button.
- *The Generate window appears with plenty of generating lines.*



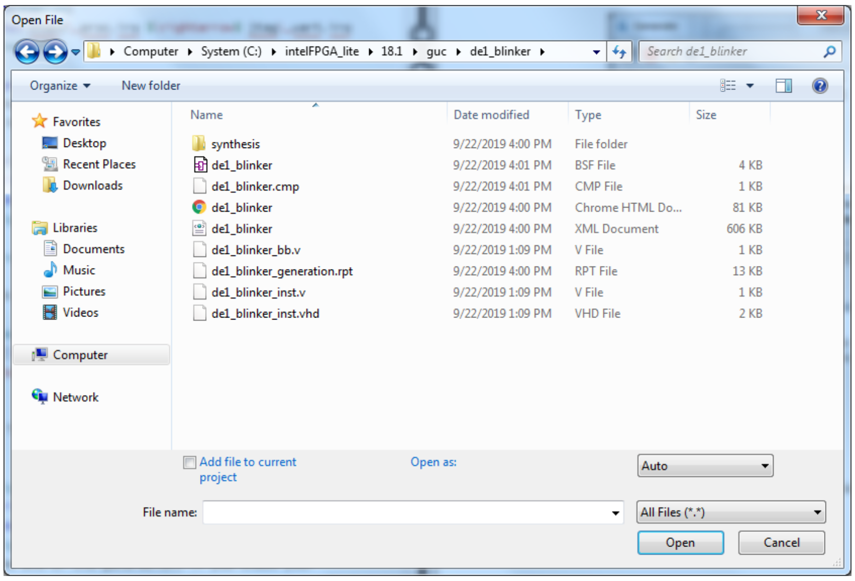
```
Progress: Parameterizing module onchip_memory
Progress: Adding switch [altera_avalon_pio 18.1]
Progress: Parameterizing module switch
Progress: Building connections
Progress: Parameterizing connections
Progress: Validating
Progress: Done reading input file
Info: de1_blinker.jtag_uart: JTAG UART IP input clock need to be at least double (2x) the o
Info: de1_blinker.switch: PIO inputs are not hardwired in test bench. Undefined values will b
Info: de1_blinker: Generating de1_blinker "de1_blinker" for QUARTUS_SYNTH
```

b) At the end of the generation, if you check your sys-on-prog-chip directory, you should see a new de1_blinker folder and 5 new files in addition to de1_blinker.sys:

- de1_blinker.bsf
- de1_blinker.cmp
- de1_blinker.html
- de1_blinker.sopcinfo
- de1_blinker.generation.rpt

c) Inside the de1_blinker folder there is another folder called **Synthesis**.

d) Inside the **Synthesis** folder another called **Submodules** with a lot of files.

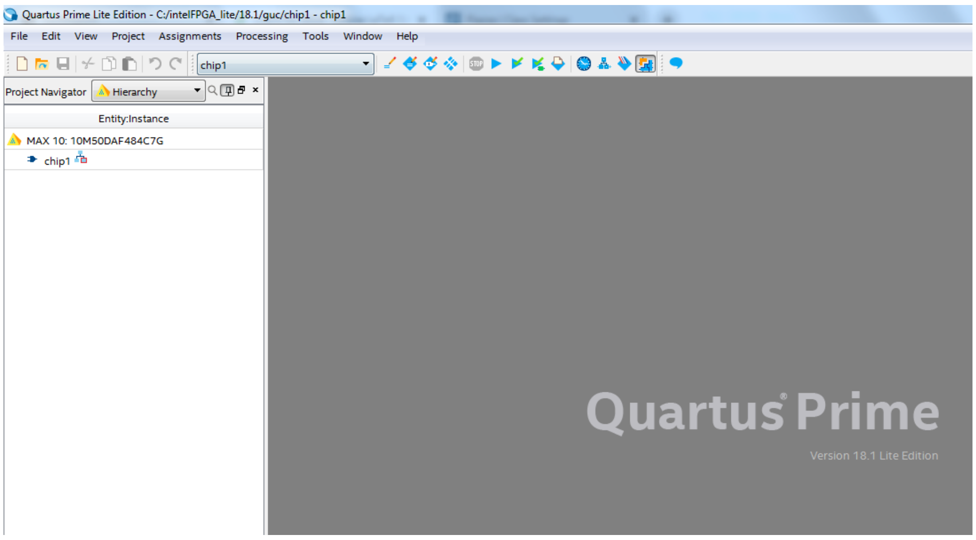


Name	Date modified	Type	Size
synthesis	9/22/2019 4:00 PM	File folder	
de1_blinker	9/22/2019 4:01 PM	BSF File	4 KB
de1_blinker.cmp	9/22/2019 4:01 PM	CMP File	1 KB
de1_blinker	9/22/2019 4:00 PM	Chrome HTML Do...	81 KB
de1_blinker	9/22/2019 4:00 PM	XML Document	606 KB
de1_blinker_bb.v	9/22/2019 1:09 PM	V File	1 KB
de1_blinker_generation.rpt	9/22/2019 4:00 PM	RPT File	13 KB
de1_blinker_inst.v	9/22/2019 1:09 PM	V File	1 KB
de1_blinker_inst.vhd	9/22/2019 1:09 PM	VHD File	2 KB

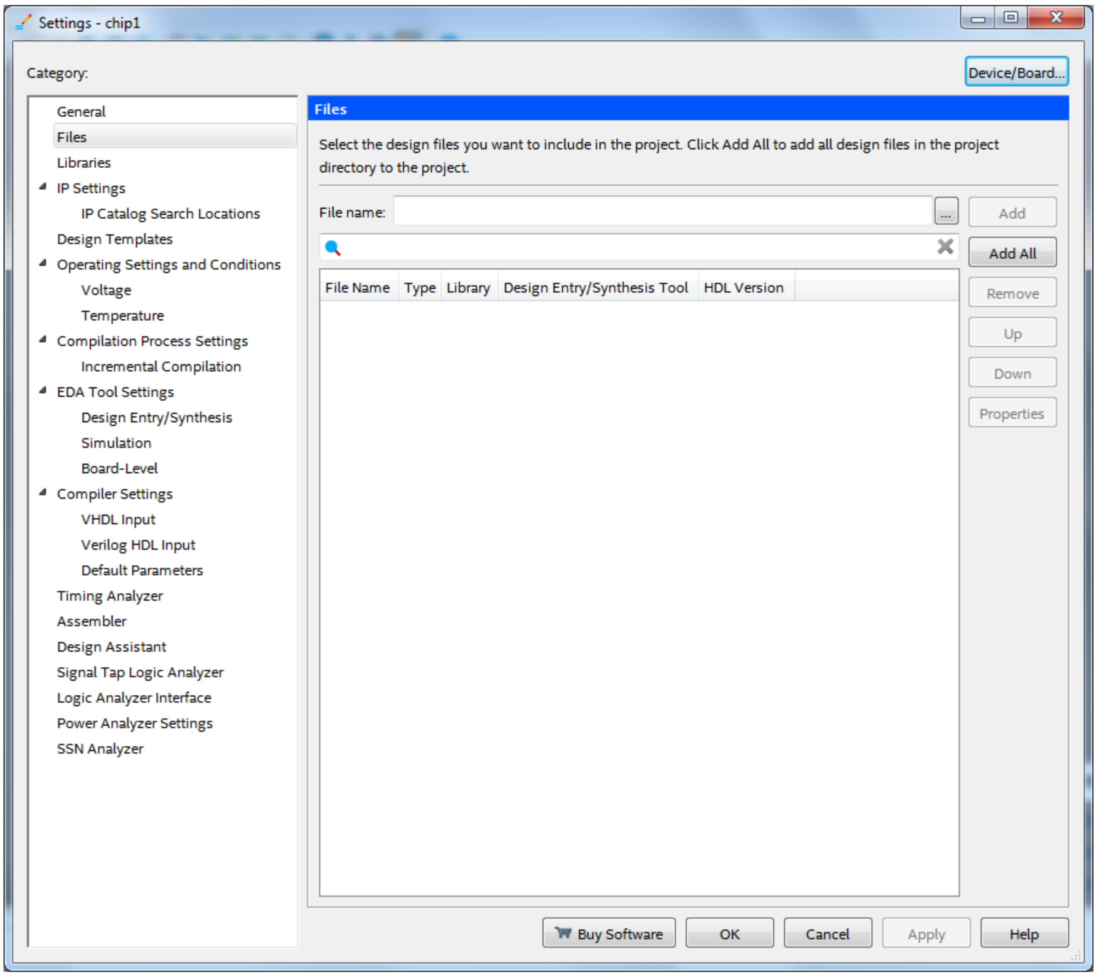
Step 10: Adding Files:

a) We now move back to the Quartus main window (Not the Platform Designer).

b) **Project → Add/Remove Files in Project**.

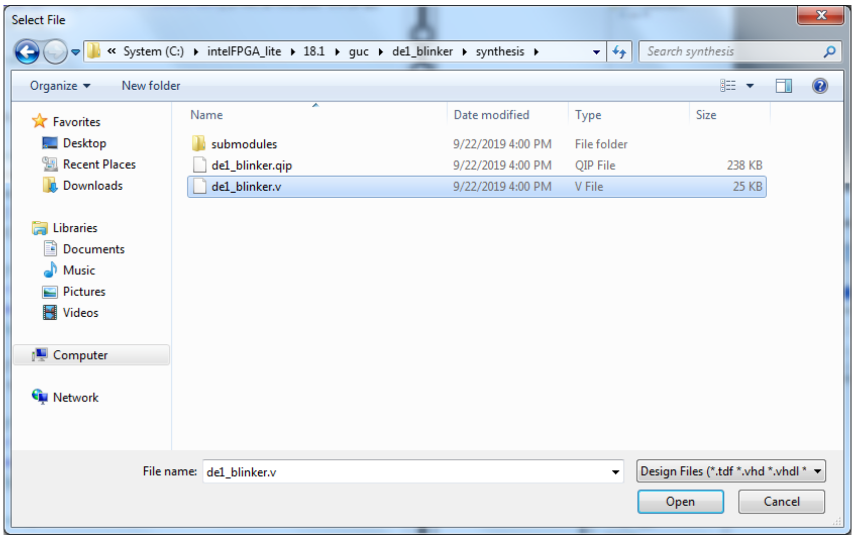

Quartus Prime Lite Edition - C:/intelFPGA_lite/18.1/guc/chip1 - chip1
File Edit View Project Assignments Processing Tools Window Help
chip1
Project Navigator
Hierarchy
Entity:Instance
MAX 10: 10M50DAF484C7G
chip1
Quartus Prime
Version 18.1 Lite Edition

c) Click the '...' button at the right of the **File name** text input.



Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

d) From the open window that appeared, select **de1_blinker** folder → **synthesis** → **de1_blinker.v**



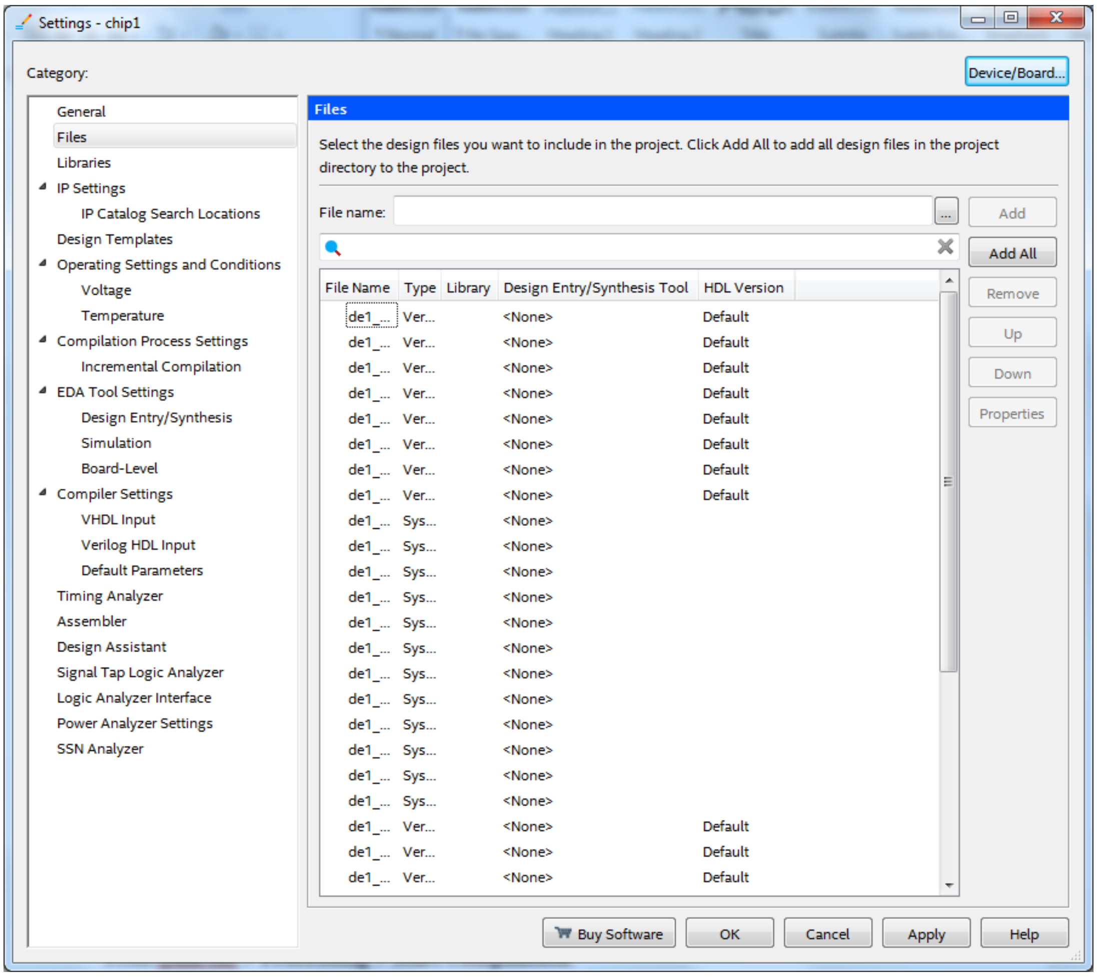
Name	Date modified	Type	Size
submodules	9/22/2019 4:00 PM	File folder	
de1_blinker.qip	9/22/2019 4:00 PM	QIP File	238 KB
de1_blinker.v	9/22/2019 4:00 PM	V File	25 KB

File name: de1_blinker.v
Design Files (*.tdf *.vhd *.vhdl *

e) Let's continue by adding all other files.

f) Select all 36 files in the **Submodules** folder.

- **de1_blinker** folder → **synthesis** → **submodules** → Select all and click **open**.



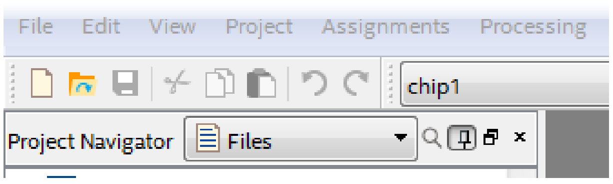
Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

File name:

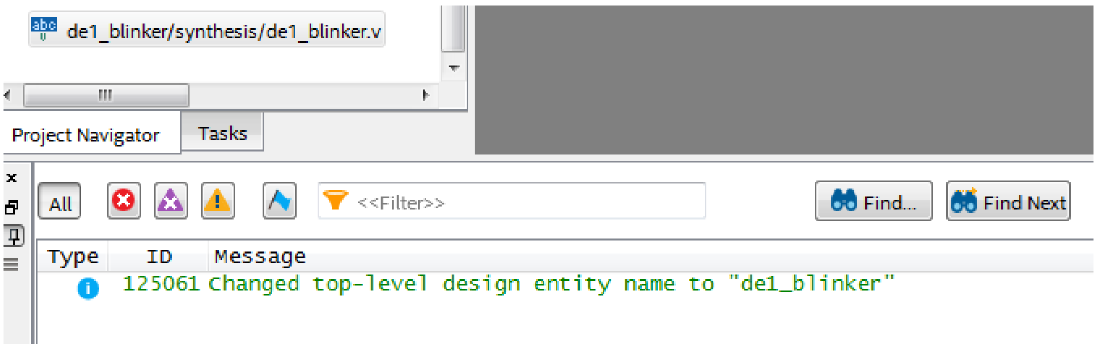
File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Sys...		<None>	
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default
de1_...	Ver...		<None>	Default

g) **Project Navigator** → click the **Files** tab.

h) In the list of files, select **de1_blinker/synthesis/de1_blinker.v**



i) **Right click** → **Set as Top-Level Entity**.

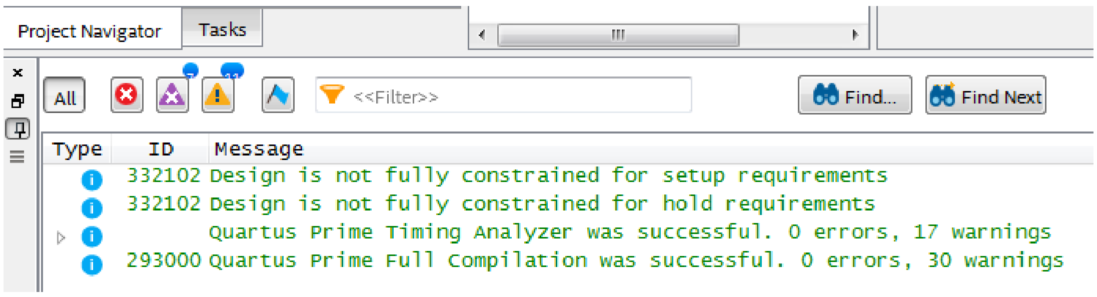


```
125061 Changed top-level design entity name to "de1_blinker"
```

Step 11: Compilation:

a) **Quartus** → **Processing** → **Start Compilation**.

b) Normally the compilation ends with no errors (only warnings). The first compilation was to tell Quartus which pins you would like to use.

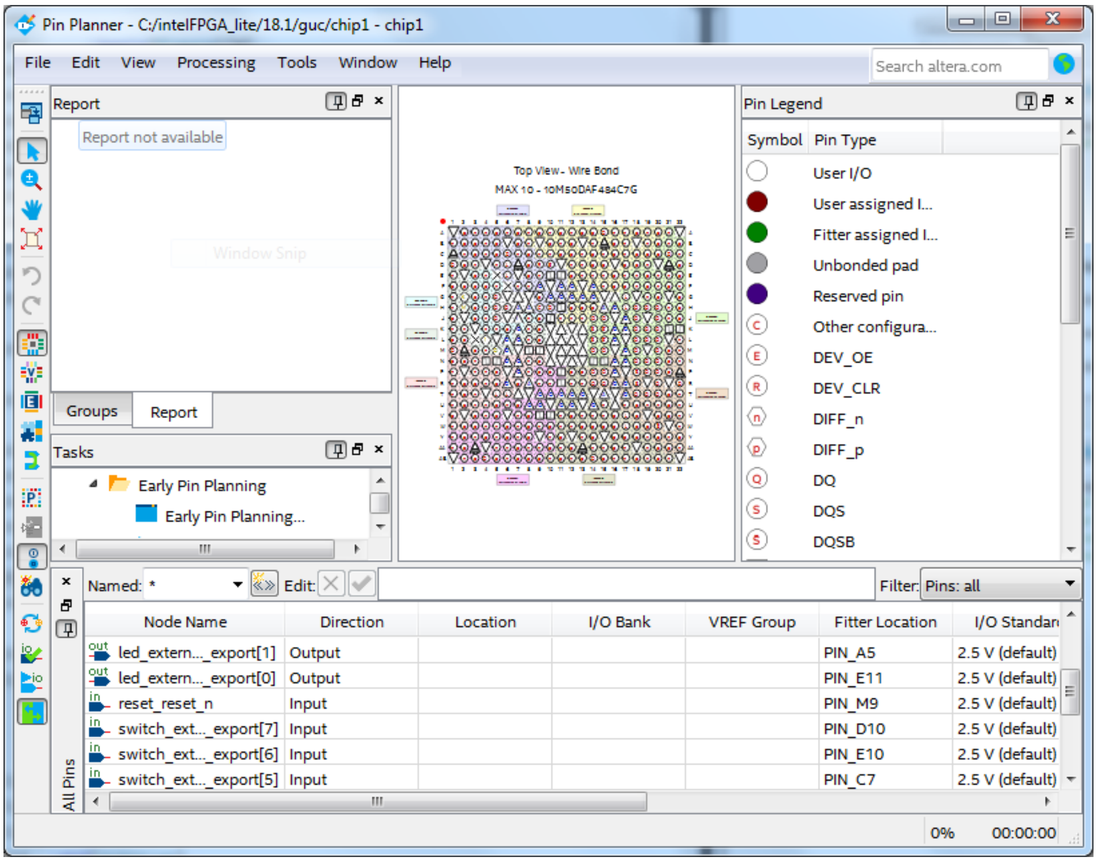


```
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
       Quartus Prime Timing Analyzer was successful. 0 errors, 17 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 30 warnings
```

c) Let's now assign the pins to its correct locations.

d) **Quartus** → **Assignments** → **Pin Planner**.

e) At the bottom, in the **All Pins** area, you should see **Node Name** but without their **Location** nor their **I/O Bank** and **VREF Group**.



Pin Planner - C:/intelFPGA_lite/18.1/guc/chip1 - chip1

File Edit View Processing Tools Window Help

Report

Report not available

Groups Report

Tasks

Early Pin Planning

Early Pin Planning...

Top View - Wire Bond
MAX 10 - 10M50DAF484C7G

Pin Legend

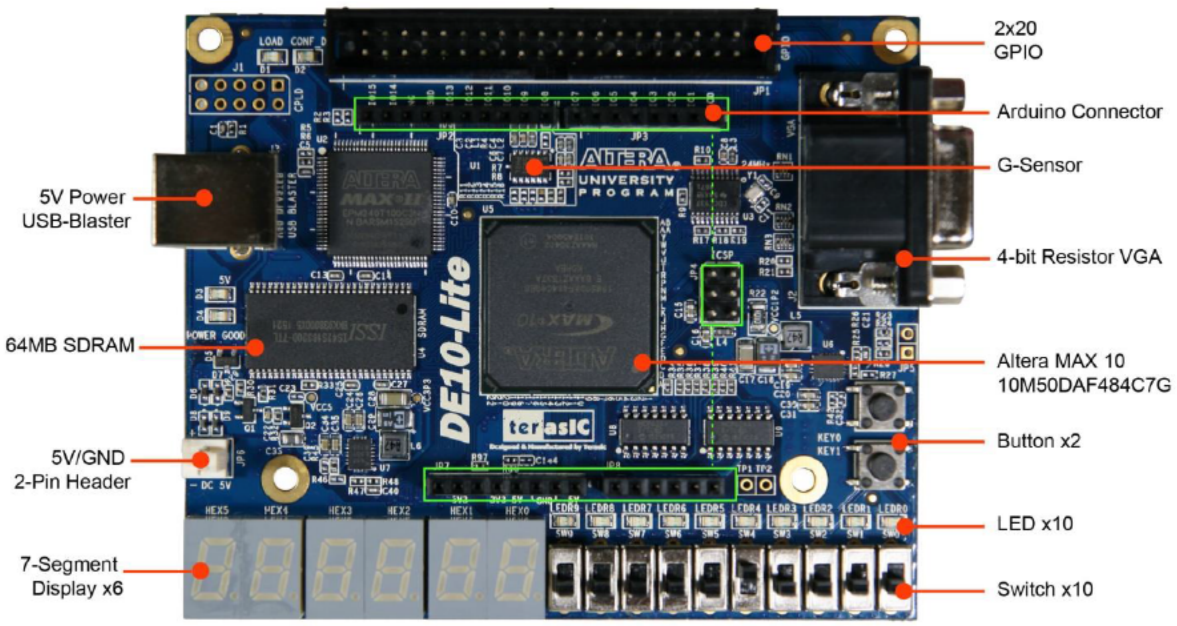
Symbol	Pin Type
○	User I/O
●	User assigned I...
●	Fitter assigned I...
●	Unbonded pad
●	Reserved pin
Ⓒ	Other configura...
Ⓔ	DEV_OE
Ⓡ	DEV_CLR
ⓝ	DIFF_n
ⓟ	DIFF_p
Ⓠ	DQ
Ⓢ	DQS
Ⓢ	DQSB

Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standar
led_extern..._export[1]	Output				PIN_A5	2.5 V (default)
led_extern..._export[0]	Output				PIN_E11	2.5 V (default)
reset_reset_n	Input				PIN_M9	2.5 V (default)
switch_ext..._export[7]	Input				PIN_D10	2.5 V (default)
switch_ext..._export[6]	Input				PIN_E10	2.5 V (default)
switch_ext..._export[5]	Input				PIN_C7	2.5 V (default)

All Pins

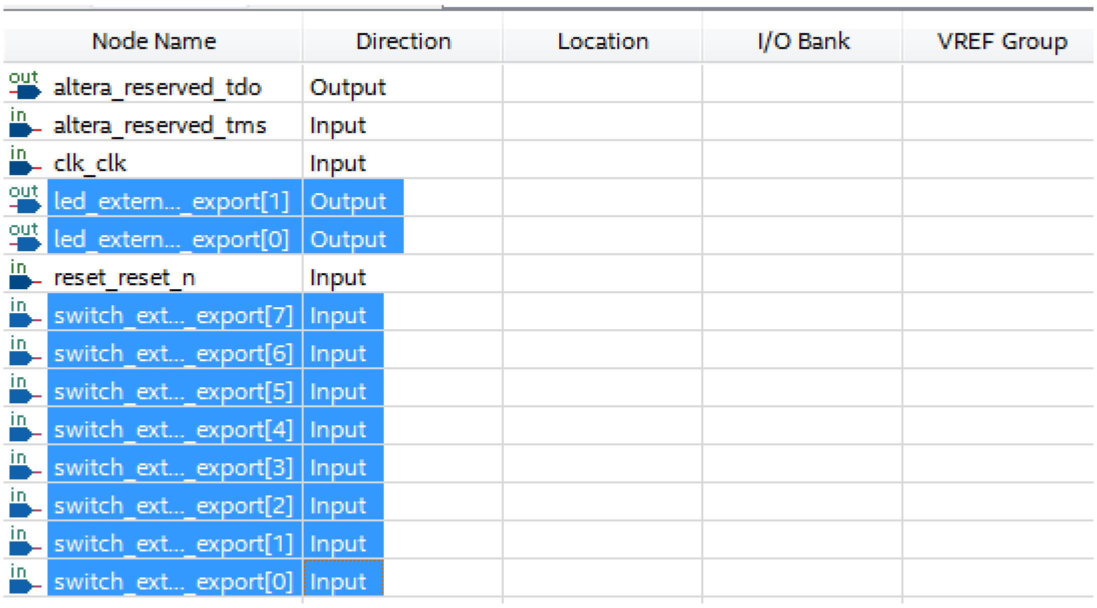
0% 00:00:00



f) We have to add each location to each node. Check the DE10-Lite board manual to know the locations.

g) Since we configured 8 inputs (switches) and 2 outputs (LEDs) in the Platform Designer, we will have the following in the Pin Planner:

- LEDs (Output)
- Switches (Input)



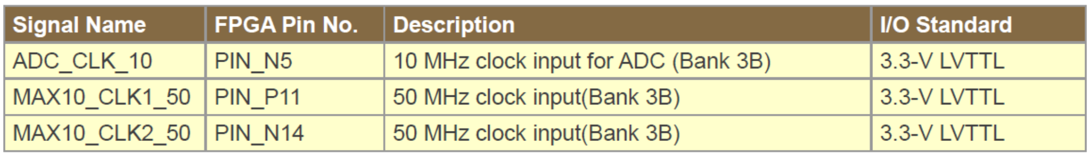
Node Name	Direction	Location	I/O Bank	VREF Group
altera_reserved_tdo	Output			
altera_reserved_tms	Input			
clk_clk	Input			
led_extern..._export[1]	Output			
led_extern..._export[0]	Output			
reset_reset_n	Input			
switch_ext..._export[7]	Input			
switch_ext..._export[6]	Input			
switch_ext..._export[5]	Input			
switch_ext..._export[4]	Input			
switch_ext..._export[3]	Input			
switch_ext..._export[2]	Input			
switch_ext..._export[1]	Input			
switch_ext..._export[0]	Input			

h) Click on the **Location** field beside each **Node Name**. A drop down menu with the pins will appear. Select the pins according to the following guide:

Signal Name	FPGA Pin No.	Description	I/O Standard
SW0	PIN_C10	Slide Switch[0]	3.3-V LVTTL
SW1	PIN_C11	Slide Switch[1]	3.3-V LVTTL
SW2	PIN_D12	Slide Switch[2]	3.3-V LVTTL
SW3	PIN_C12	Slide Switch[3]	3.3-V LVTTL
SW4	PIN_A12	Slide Switch[4]	3.3-V LVTTL
SW5	PIN_B12	Slide Switch[5]	3.3-V LVTTL
SW6	PIN_A13	Slide Switch[6]	3.3-V LVTTL
SW7	PIN_A14	Slide Switch[7]	3.3-V LVTTL
SW8	PIN_B14	Slide Switch[8]	3.3-V LVTTL
SW9	PIN_F15	Slide Switch[9]	3.3-V LVTTL

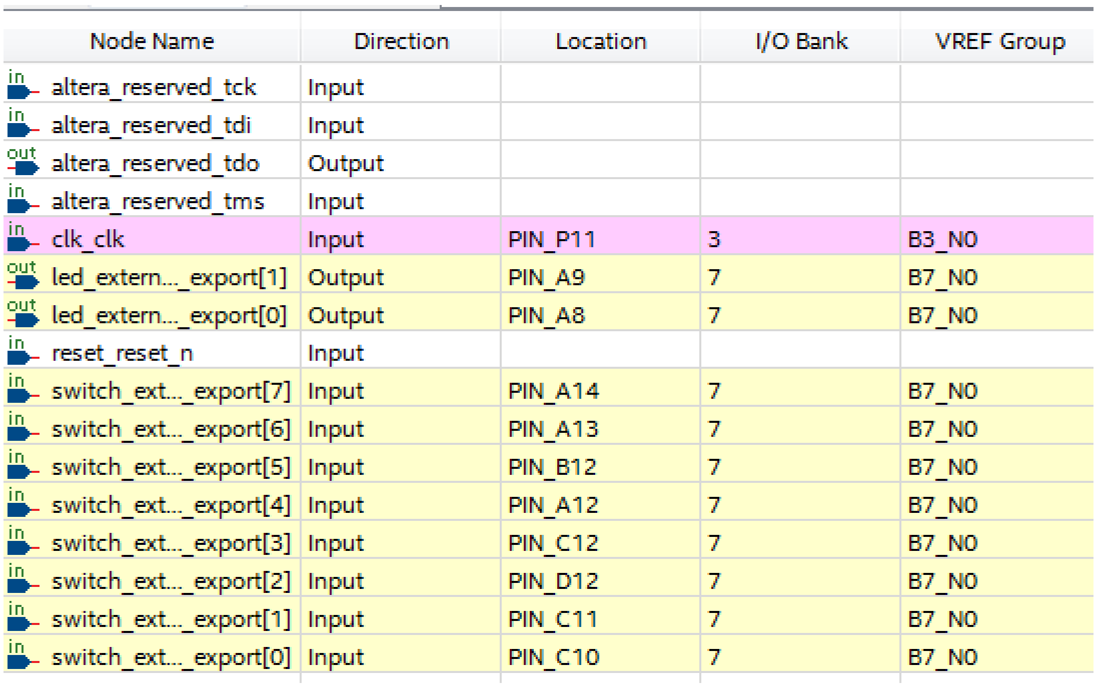
Signal Name	FPGA Pin No.	Description	I/O Standard
LEDR0	PIN_A8	LED [0]	3.3-V LVTTL
LEDR1	PIN_A9	LED [1]	3.3-V LVTTL
LEDR2	PIN_A10	LED [2]	3.3-V LVTTL
LEDR3	PIN_B10	LED [3]	3.3-V LVTTL
LEDR4	PIN_D13	LED [4]	3.3-V LVTTL
LEDR5	PIN_C13	LED [5]	3.3-V LVTTL
LEDR6	PIN_E14	LED [6]	3.3-V LVTTL
LEDR7	PIN_D14	LED [7]	3.3-V LVTTL
LEDR8	PIN_A11	LED [8]	3.3-V LVTTL
LEDR9	PIN_B11	LED [9]	3.3-V LVTTL

i) We need to configure the location of our **clk_clk:**



Signal Name	FPGA Pin No.	Description	I/O Standard
ADC_CLK_10	PIN_N5	10 MHz clock input for ADC (Bank 3B)	3.3-V LVTTL
MAX10_CLK1_50	PIN_P11	50 MHz clock input(Bank 3B)	3.3-V LVTTL
MAX10_CLK2_50	PIN_N14	50 MHz clock input(Bank 3B)	3.3-V LVTTL

j) After selecting the locations for our input (switches), output (LEDs) and clock pins:

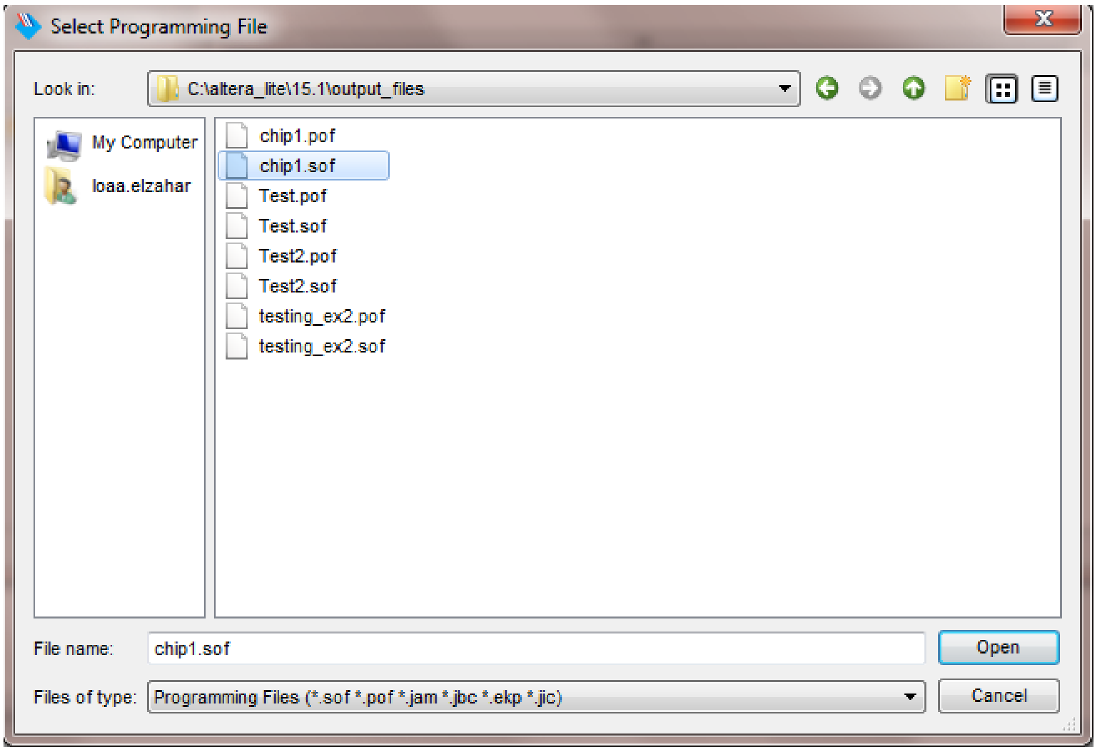


Node Name	Direction	Location	I/O Bank	VREF Group
altera_reserved_tck	Input			
altera_reserved_tdi	Input			
altera_reserved_tdo	Output			
altera_reserved_tms	Input			
clk_clk	Input	PIN_P11	3	B3_N0
led_extern..._export[1]	Output	PIN_A9	7	B7_N0
led_extern..._export[0]	Output	PIN_A8	7	B7_N0
reset_reset_n	Input			
switch_ext..._export[7]	Input	PIN_A14	7	B7_N0
switch_ext..._export[6]	Input	PIN_A13	7	B7_N0
switch_ext..._export[5]	Input	PIN_B12	7	B7_N0
switch_ext..._export[4]	Input	PIN_A12	7	B7_N0
switch_ext..._export[3]	Input	PIN_C12	7	B7_N0
switch_ext..._export[2]	Input	PIN_D12	7	B7_N0
switch_ext..._export[1]	Input	PIN_C11	7	B7_N0
switch_ext..._export[0]	Input	PIN_C10	7	B7_N0

k) Recompile the project. **Quartus** → **Processing** → **Start Compilation**.
- *This second compilation is really important, don't skip it.*

Step 12: Sending the Design to the Board:

a) At this point, you have to send the file **chip1.sof**, generated by the second compilation, into the board.



Select Programming File

Look in: C:\altera_lite\15.1\output_files

My Computer
loaa.elzahar

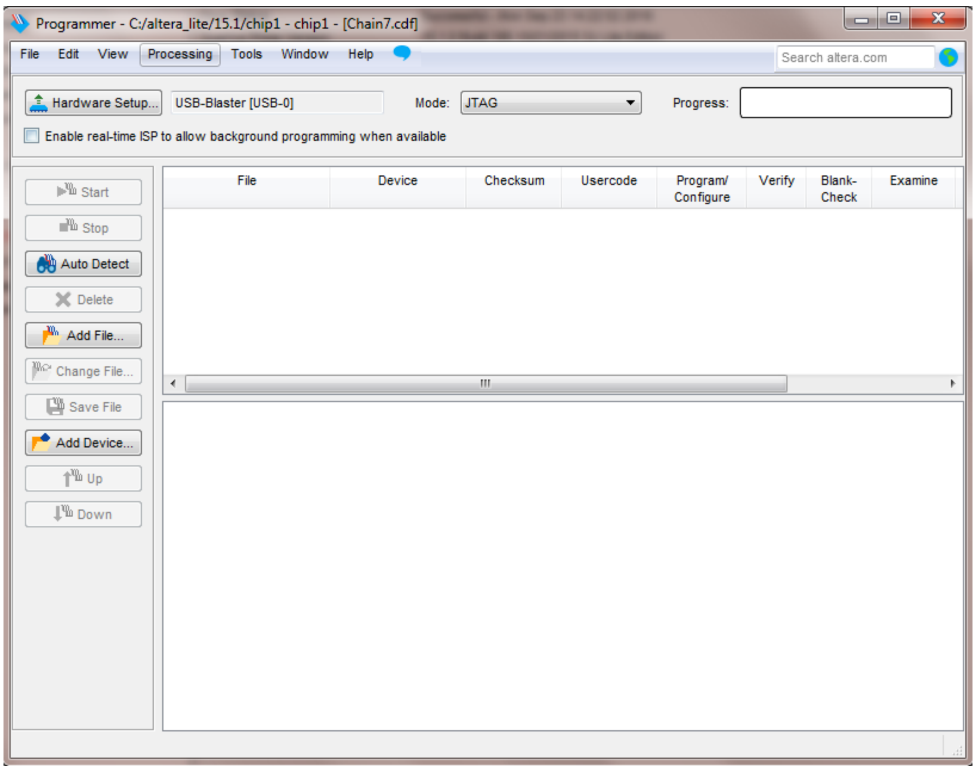
chip1.pof
chip1.sof
Test.pof
Test.sof
Test2.pof
Test2.sof
testing_ex2.pof
testing_ex2.sof

File name: chip1.sof

Files of type: Programming Files (*.sof *.pof *.jam *.jbc *.ekp *.jic)

Open
Cancel

b) **Quartus** → **Tools** → **Programmer**.



Programmer - C:/altera_lite/15.1/chip1 - chip1 - [Chain7.cdf]

Hardware Setup... USB-Blaster [USB-0]

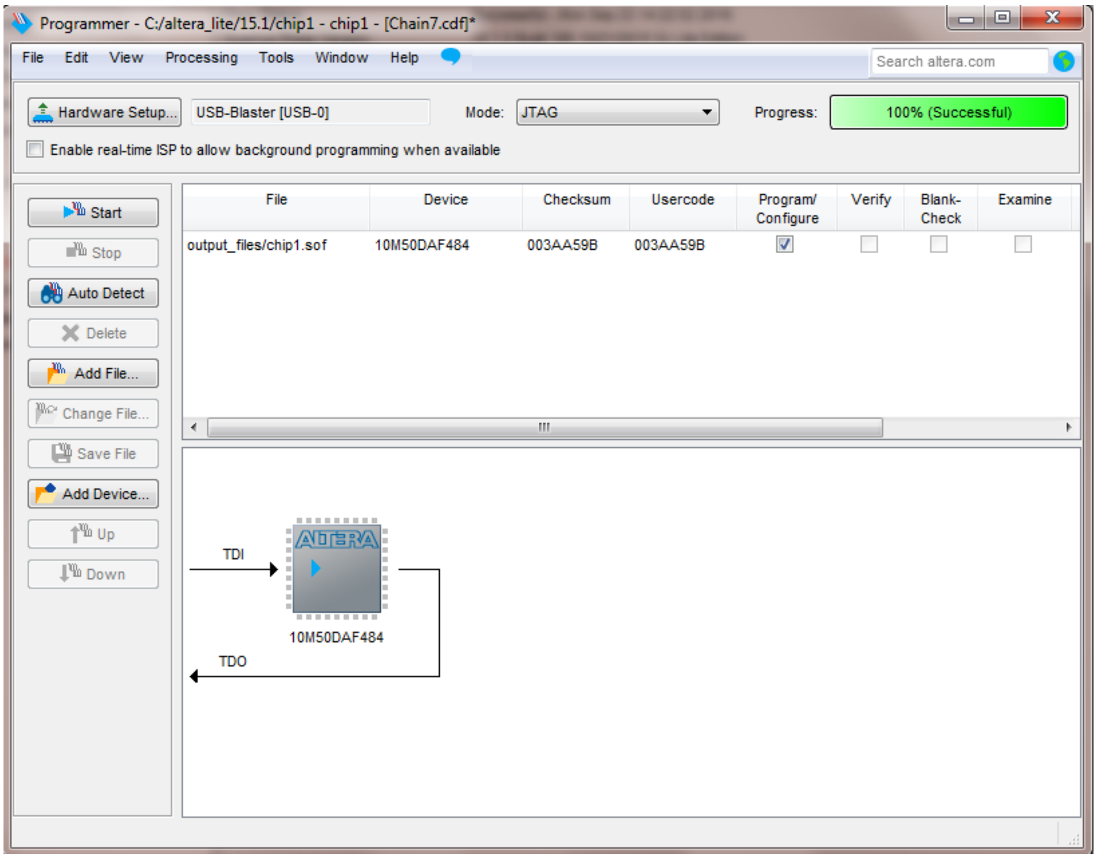
Mode: JTAG

Progress:

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine

c) You just have to click **Start** to upload the program into the board.



USB-Blaster [USB-0]   Mode: JTAG   Progress: 100% (Successful)

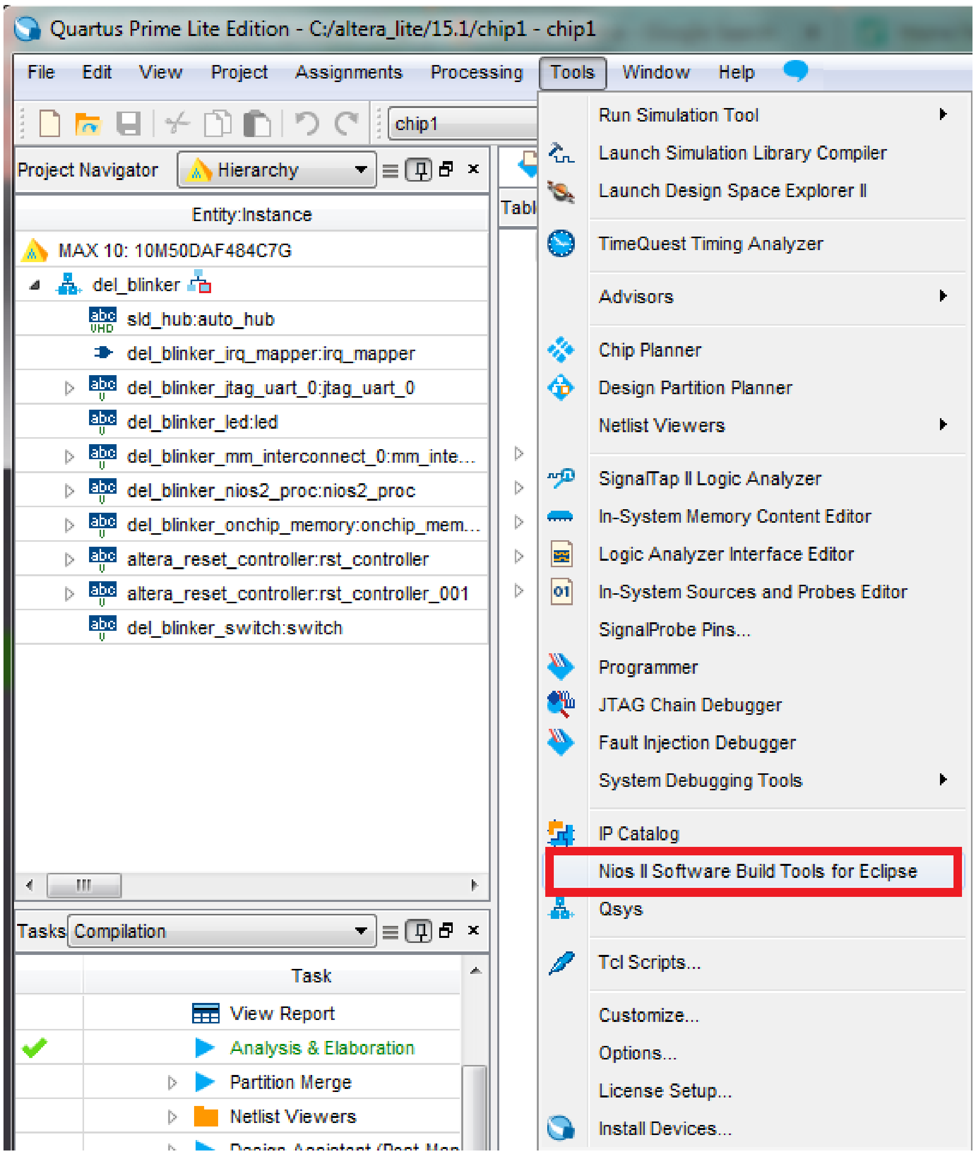
Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
output_files/chip1.sof	10M50DAF484	003AA59B	003AA59B	✓			

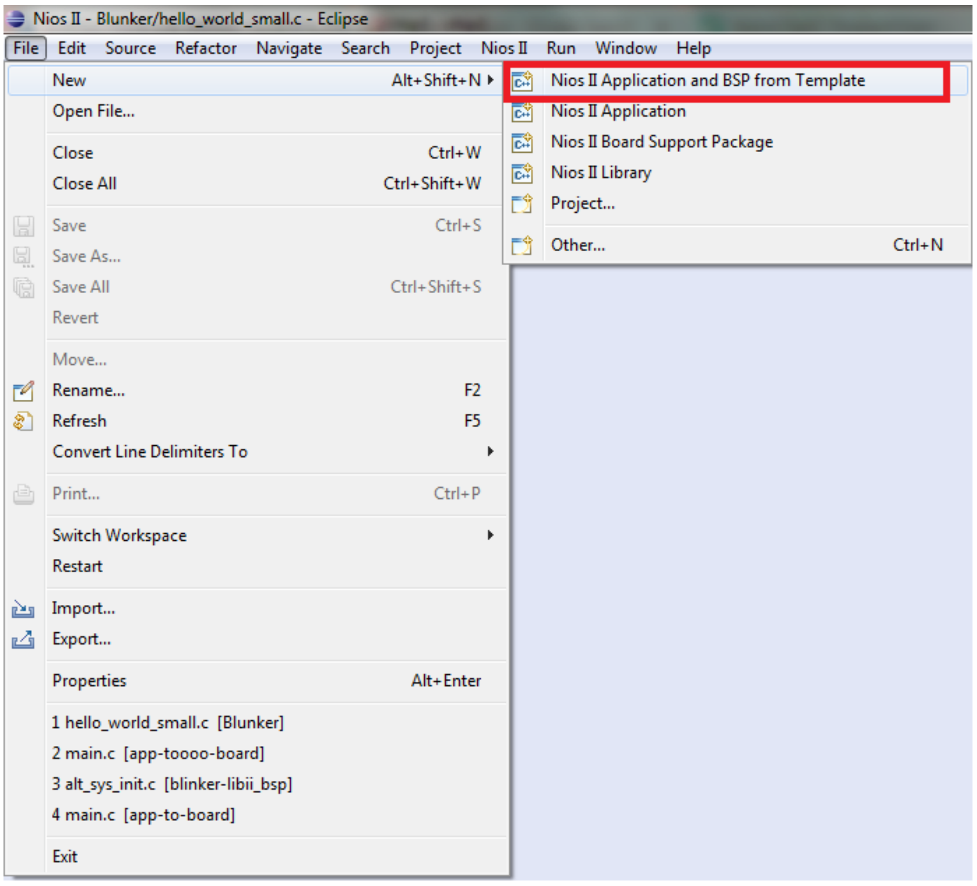
Step 13: Creating a BSP library with Nios II Software Tools:

Important Note:
It's necessary to have the board plugged and turned on.
Otherwise the BSP library won't find the corresponding ID nor the timestamp of the current program.
Indeed, when you create the BSP, the library will read the ID and timestamp of the board's program.
Once the design is sent to the board, you can create our BSP library.

a) **Quartus → Tools → Nios II Software Build Tools for Eclipse**


Quartus Prime Lite Edition - C:/altera_lite/15.1/chip1 - chip1
File
Edit
View
Project
Assignments
Processing
Tools
Window
Help
chip1
Project Navigator
Hierarchy
Entity:Instance
MAX 10: 10M50DAF484C7G
del_blinker
sld_hub:auto_hub
del_blinker_irq_mapper:irq_mapper
del_blinker_jtag_uart_0:jtag_uart_0
del_blinker_led:led
del_blinker_mm_interconnect_0:mm_inte...
del_blinker_nios2_proc:nios2_proc
del_blinker_onchip_memory:onchip_mem...
altera_reset_controller:rst_controller
altera_reset_controller:rst_controller_001
del_blinker_switch:switch
Tasks
Compilation
Task
View Report
Analysis & Elaboration
Partition Merge
Netlist Viewers
Run Simulation Tool
Launch Simulation Library Compiler
Launch Design Space Explorer II
TimeQuest Timing Analyzer
Advisors
Chip Planner
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Fault Injection Debugger
System Debugging Tools
IP Catalog
Nios II Software Build Tools for Eclipse
Qsys
Tcl Scripts...
Customize...
Options...
License Setup...
Install Devices...

b) **Eclipse → File → New→ Nios II Application and BSP from Template**


Nios II - Blunker/hello_world_small.c - Eclipse
File Edit Source Refactor Navigate Search Project Nios II Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 hello_world_small.c [Blunker]
2 main.c [app-toooo-board]
3 alt_sys_init.c [blinker-libii_bsp]
4 main.c [app-to-board]
Exit
Nios II Application and BSP from Template
Nios II Application
Nios II Board Support Package
Nios II Library
Project...
Other... Ctrl+N

c) In **SOPC Information File name**, choose **de1_blinker.sopcinfo**

d) Automatically in the CPU name part you can see **nios2_proc** appears in the drop-down menu.

e) Add a name in the **Project name** part, for example **blinker-lib**.

f) In the **Project template**, let **Hello Small World** as template.

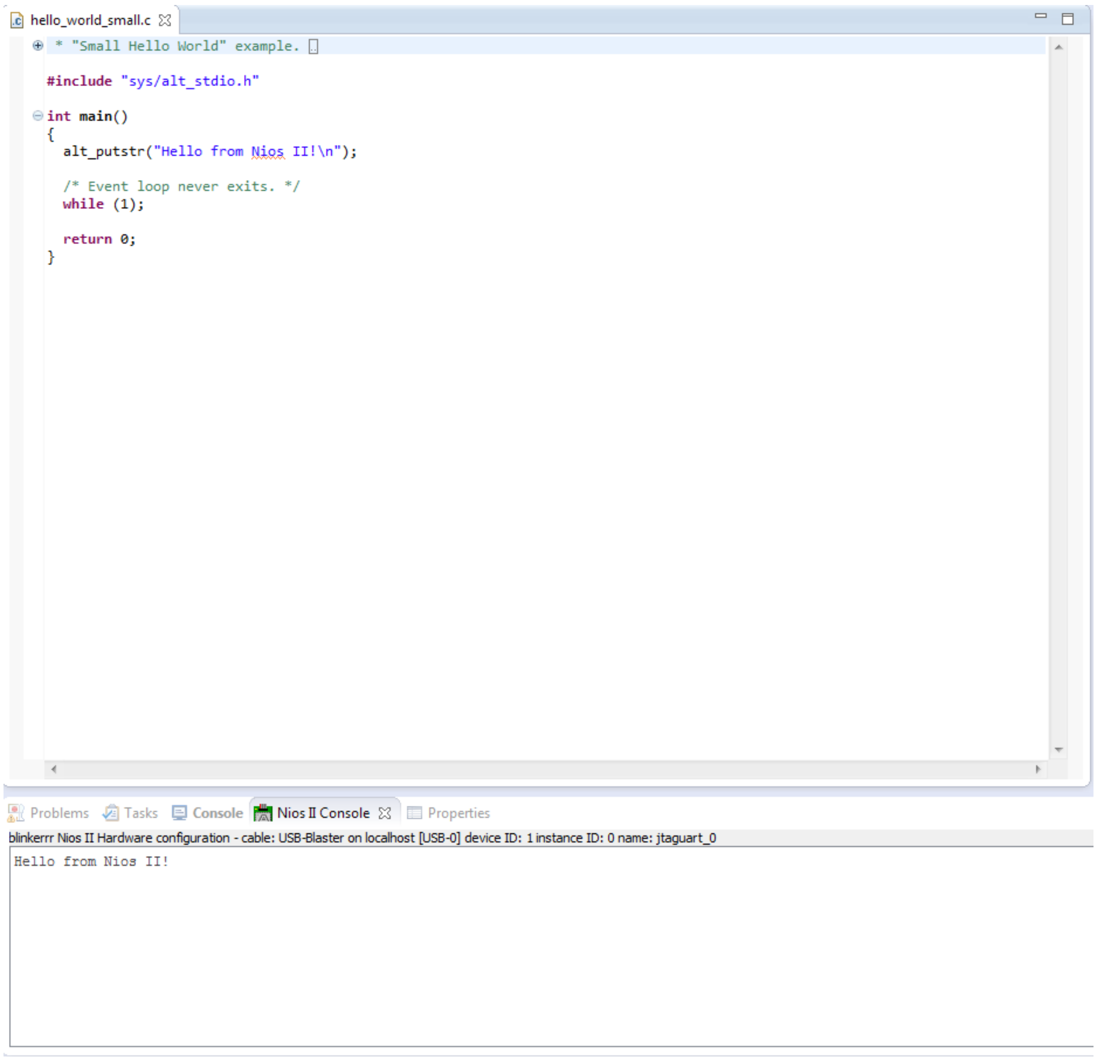
g) Click Next, you should see **blinker-lib_bsp** as **BSP Project name**. Click Finish.

h) The compilation of the BSP library starts and you can find two new projects. In the **Project Explorer**, there is two new projects:

- blinker-lib
- blinker-lib_bsp

i) **Project Explorer** → **Right click on blinker-lib** → **Run As** → **Nios II Hardware**

j) You should see a message on your console.



```
 * "Small Hello World" example.

#include "sys/alt_stdio.h"

int main()
{
  alt_putstr("Hello from Nios II!\n");

  /* Event loop never exits. */
  while (1);

  return 0;
}
```

```
blinkerrr Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtaguart_0
Hello from Nios II!
```

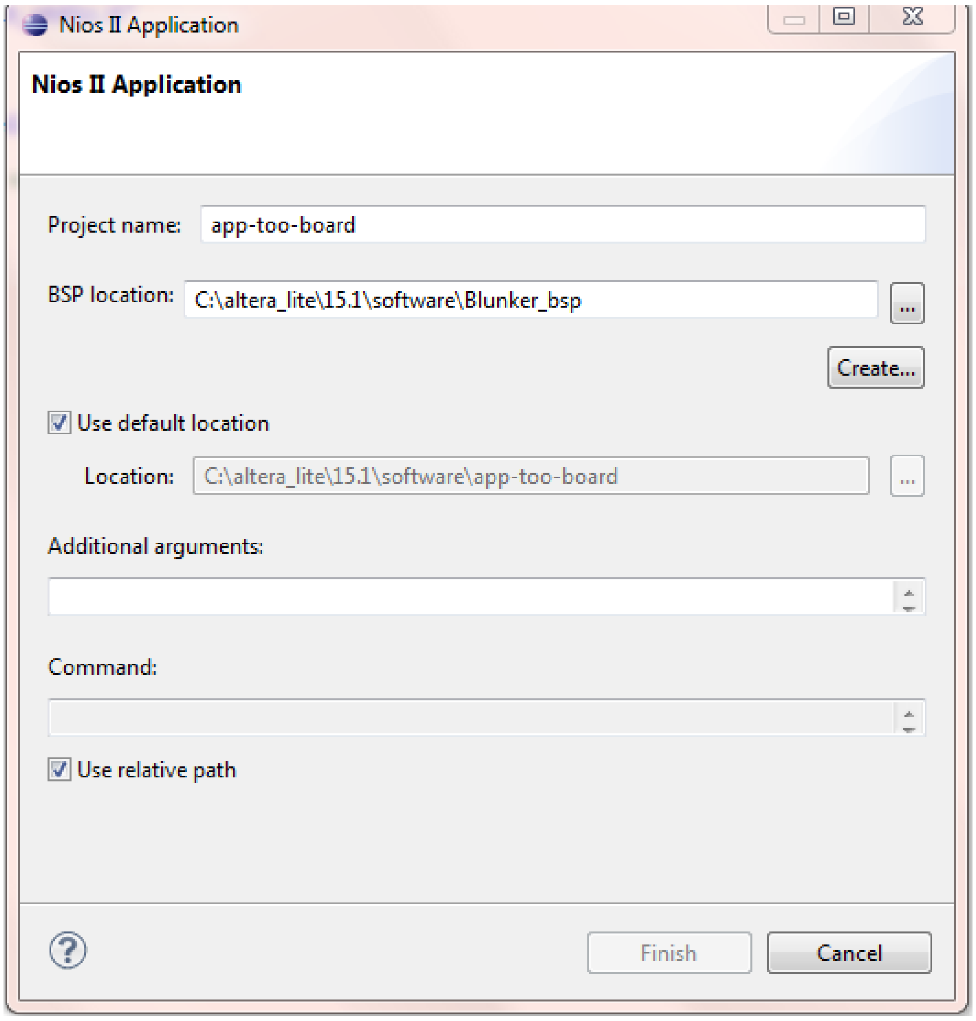
Step 14: Creating a User Application with Nios II Software Tools:

a) It's time now to create a user application.

b) **Eclipse** → **File** → **New** → **Nios II Application**

c) Add a name in the **Project name**, for example **app-to-board**.

d) In the **BSP location**, choose **blinker-lib_bsp**. Click Finish.



Nios II Application

Project name: app-too-board

BSP location: C:\altera_lite\15.1\software\Blunker_bsp

☑ Use default location

Location: C:\altera_lite\15.1\software\app-too-board

Additional arguments:

Command:

☑ Use relative path

e) Let's create a C file.

f) **Project Explorer** → **Right-click app-to-board** → **New** → **Source File**

g) As **Source file**, type **main.c**.

h) As **Template**, choose **Default C source template**. Click Finish.

i) In the new file created, write the following code that were are going to explain soon:

```
#include "io.h"
#include "system.h"

// Gets led_pattern as parameter in order to write to the register.

void set_led(unsigned char led_pattern) {
    IOWR(LED_BASE, 0, led_pattern); // writes register
}

// Main, what else?
// Gets LEDs pattern from switchers.
// Sets LEDs register according to the pattern.

void main() {
    unsigned char led_pattern = 0x00;

    while (1) { // infinite loop
        led_pattern = IORD(SWITCH_BASE, 0); // gets LEDs
        set_led(led_pattern); // sets LEDs
    }
}
```

j) Then from the **Project Explorer** → **Right-click app-to-board** → **Run As** → **3 Nios II Hardware.**

Note:

- The main program start compiling then it is linked. It's possible that the Run Configuration window opens.
- Often there is a lack of information, so in the Target Connection tab, you have to click Refresh Connections on the right.
- If you click the System ID Properties, you should see expected ID base, system and timestamp.
- Sometimes, there are errors because Eclipse or the board cannot retrieve correct information (for example because the board wasn't turned on during the BSP creation).
- In this case, check Ignore mismatched system ID and/or Ignore mismatched system timestamp in the Target Connection tab at the bottom. Then click Apply and Run.
- Even sometimes if you close and reopen Eclipse it works. Or if you relanch a compilation from your design from Quartus and reload the chip1.sof into the board. Or just try to recompile the project.
- It's not always easy to understand, I agree, but electronics is a delicate thing.

- At the end of the process, you should see in the console, something like that:

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: not responding.
Resetting and trying again: OK
Reading System ID at address 0x00009020: verified
Initializing CPU cache (if present)
OK

Downloading 00004000 ( 0%)
Downloading 00004F0C (61%)
Downloaded 4KB in 0.0s

Verifying 00004000 ( 0%)
Verifying 00004F0C (61%)
Verified OK
Starting processor at address 0x00004020
```

- If yes, then the program is into the board.
- Try to push switchers SW0 and SW1 to high, you should see LEDR0 and LEDR1 light respectively.