A Simple Nios® II Implementation on Intel® MAX® 10-10M08 Evaluation Kit

By Sean D. Liming and John R. Malin
Annabooks, LLC. – www.annabooks.com

November 2022

The Nios II is a 32-bit soft processor core for Intel FPGAs. The Nios II softcore processor is an IP block that comes with the Quartus Prime tools. The Nios II comes in two flavors Nios II/f (fast) and Nios II/e (economy). The Nios II/f is larger and takes up a few more Logical Elements in the chip, but it has MMU support for real-time operating system applications. Nios II/e is smaller with no MMU support. Nios II/f requires a license and the Nios II/e doesn't. In this hands-on exercise, we will create a simple Nios II/f design and run a hello world application on the new processor.

The Project Requirements:

- Intel Quartus Prime Lite Edition V21.0 and Nios® II SBT for Eclipse already installed.
- Intel® MAX® 10 - 10M08 Evaluation Kit and the schematic for the evaluation board are required. The schematic PDF file can be downloaded from the Intel FPGA website.
- Intel FPGA Programming cable – USB Blaster II or EthernetBlaster II. The Intel® MAX® 10 - 10M08 Evaluation Kit doesn't have a built-in USB Blaster II on board.
- *Intel® Quartus® Prime Lite and Nios® II SBT for Eclipse Installation Instructions* on Annabooks.com

Note: There are equivalent MAX 10 development and evaluation boards available. These boards can also be used as the target, but you will have to adjust to the available features on the board. Please make sure that you have the board's schematic files as these will be needed to identify pins.

Windows was used as the host OS during the development of this paper, and there were problems running the development tools on Windows. Eclipse is the application development environment, which by itself is not a problem. The issue is that Linux commands are required to set up and build application projects. The Windows Subsystem for Linux has to be installed along with a Linux distribution such as Ubuntu. There were many little steps to getting all these items set up. Even after all the little steps to get Eclipse running were taken, applications would fail to build because "make" was not found. After a few searches and comparison tests with an Ubuntu setup were made, all the little steps necessary to get the software to build and run were worked out. Please see the article *Intel® Quartus® Prime Lite and Nios® II SBT for Eclipse Installation Instructions* on Annabooks.com to install the software needed for this hands-on exercise.

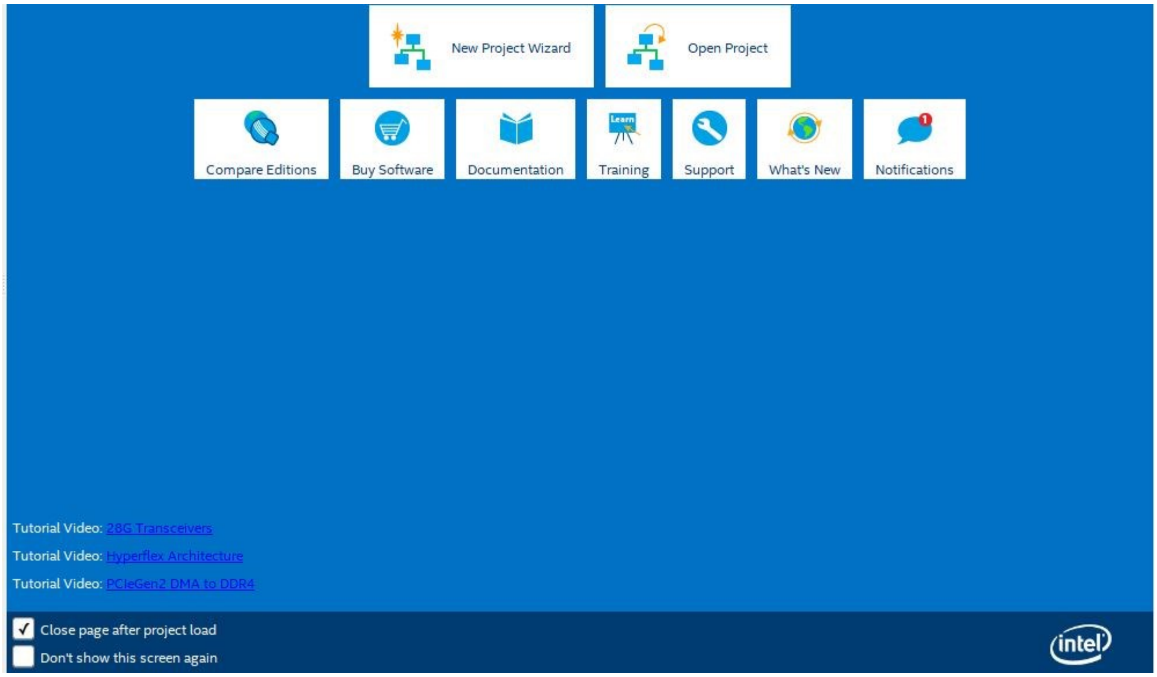
1.1 Basic Nios II Project

For this design, we will have a Nios II processor IP block, along with an onboard RAM IP Block, and a JTAG UART IP Block. The application will simply send messages out the JTAG interface to a console application running in Eclipse. There are two parts to the design process. The first involves creating the hardware design in Quartus Prime and Platform Builder. The second step is to create the application with Eclipse.

1.1.1 Create the Project

The first step is to create a design project.

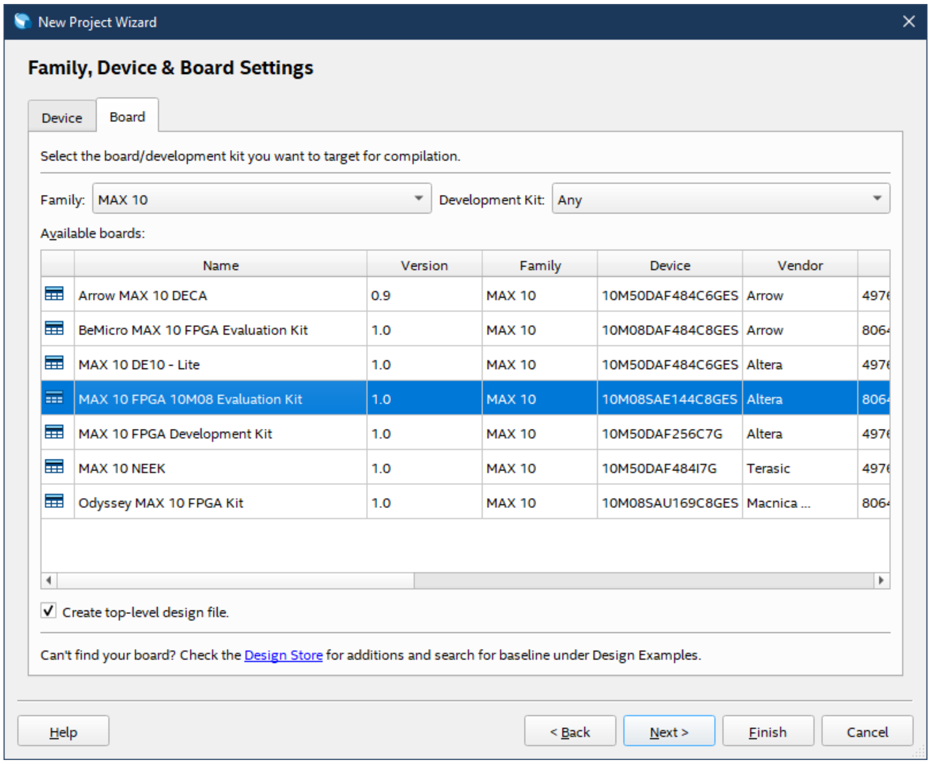
1. Open Quartus.
2. Click on the New Project Wizard.


New Project Wizard
Open Project
Compare Editions
Buy Software
Documentation
Training
Support
What's New
Notifications
Tutorial Video: 28G Transceivers
Tutorial Video: Hyperflex Architecture
Tutorial Video: PCIeGen2 DMA to DDR4
☑ Close page after project load
□ Don't show this screen again
intel

3. Click Next to the Introduction dialog.
4. Select or create a project directory \NIOSii_Example (Do not use the Quartus installation directory) and name the project: "NIOS2CPU". Click Next.

Note: By default, the root directory is the Quartus installation directory. Make sure the root project directory is a separate path from the Quartus installation files. Also, there can be no spaces in the name of the folders or projects.

5. Project Type: Empty project, click Next.
6. Add File: no files to add, click Next.
7. Family, Device & Board Settings: click the Board tab and select: MAX 10 FPGA 10M08 Evaluation Kit. Click Next.



Family, Device & Board Settings

Select the board/development kit you want to target for compilation.

Family: MAX 10

Development Kit: Any

Available boards:

	Name	Version	Family	Device	Vendor	
▦	Arrow MAX 10 DECA	0.9	MAX 10	10M50DAF484C6GES	Arrow	497
▦	BeMicro MAX 10 FPGA Evaluation Kit	1.0	MAX 10	10M08DAF484C8GES	Arrow	806
▦	MAX 10 DE10 - Lite	1.0	MAX 10	10M50DAF484C6GES	Altera	497
▦	MAX 10 FPGA 10M08 Evaluation Kit	1.0	MAX 10	10M08SAE144C8GES	Altera	806
▦	MAX 10 FPGA Development Kit	1.0	MAX 10	10M50DAF256C7G	Altera	497
▦	MAX 10 NEEK	1.0	MAX 10	10M50DAF484I7G	Terasic	497
▦	Odyssey MAX 10 FPGA Kit	1.0	MAX 10	10M08SAU169C8GES	Macnica ...	806

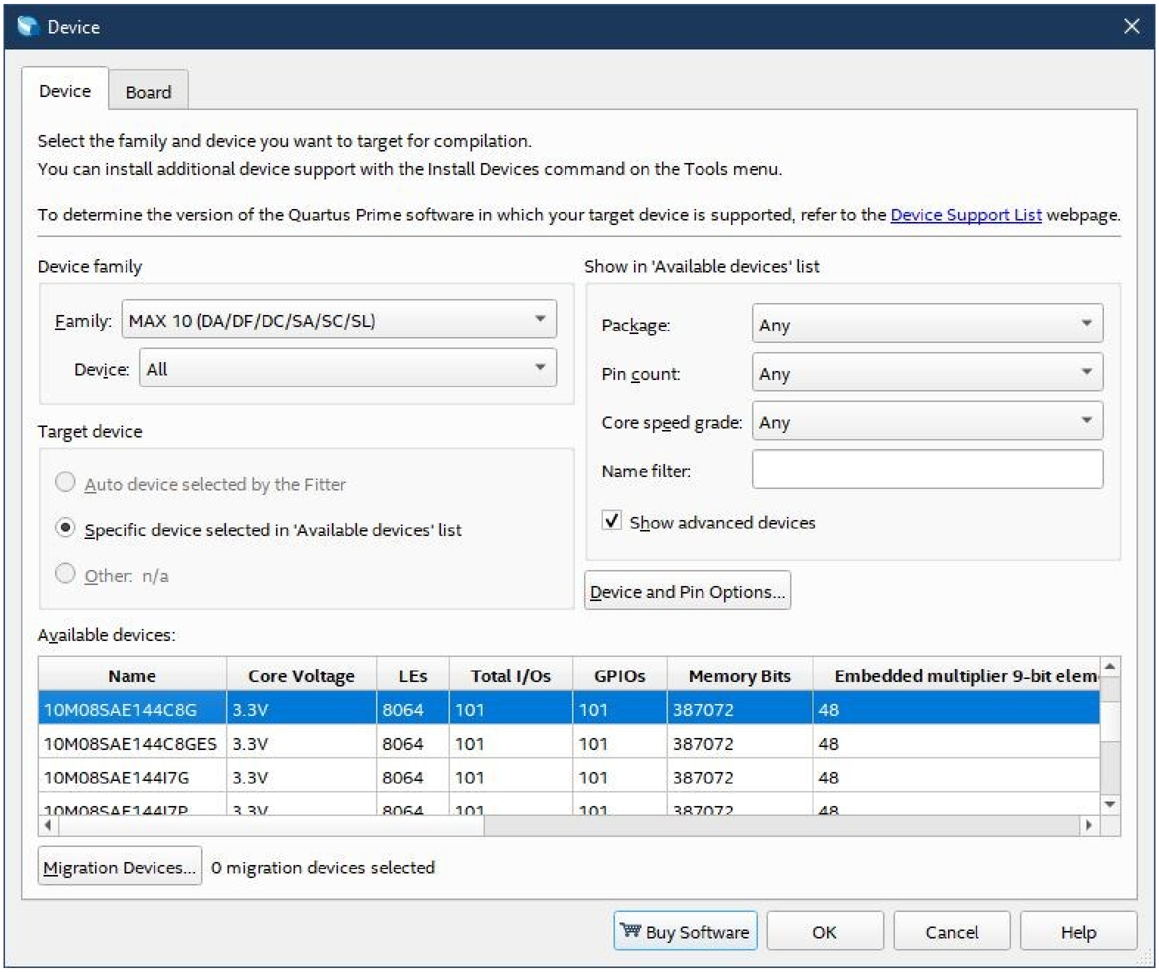
☑ Create top-level design file.

Can't find your board? Check the Design Store for additions and search for baseline under Design Examples.

8. EDA Tools: click Next.
9. Summary: click Finish

Note: The actual MAX 10 on our board is the 10M08SAE144C8G, thus it is not an Engineering Sample (ES). The next two steps change the device to the production device. Your experience might be different. These next two optional steps change the device.

10. In the project navigation pane on the left, right-click on 10: 10M08SAE144C8GE, and select Device from the context menu.
11. In the Available devices, scroll down and select the 10M08SAE144C8G. Click OK.



Device

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/SL)

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

Device and Pin Options...

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit elem
10M08SAE144C8G	3.3V	8064	101	101	387072	48
10M08SAE144C8GES	3.3V	8064	101	101	387072	48
10M08SAE144I7G	3.3V	8064	101	101	387072	48
10M08SAE144I7P	3.3V	8064	101	101	387072	48

Migration Devices... 0 migration devices selected

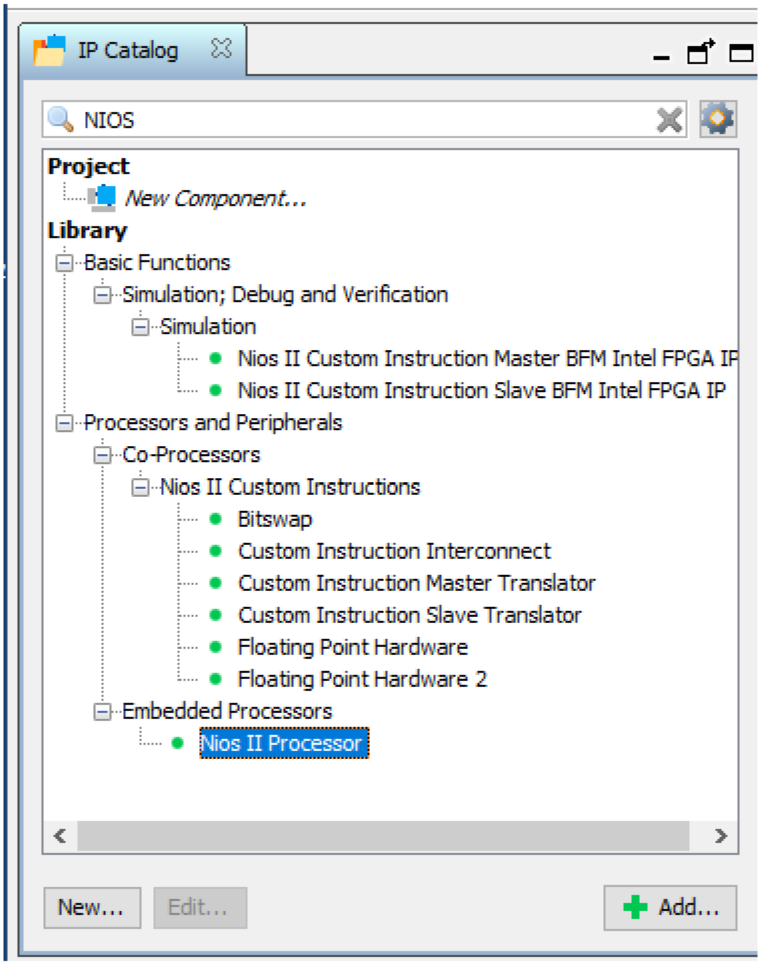
1.1.2 Create the Design Step 1: Platform Designer

Quartus supports many design types to create an FPGA design. The Platform Designer tool will be used for this hands-on exercise. Platform Builder makes it easy to add already-built IP blocks and interconnect them.

1. From the menu, select Tools->Platform Designer, or the Platform Designer icon from the toolbar.

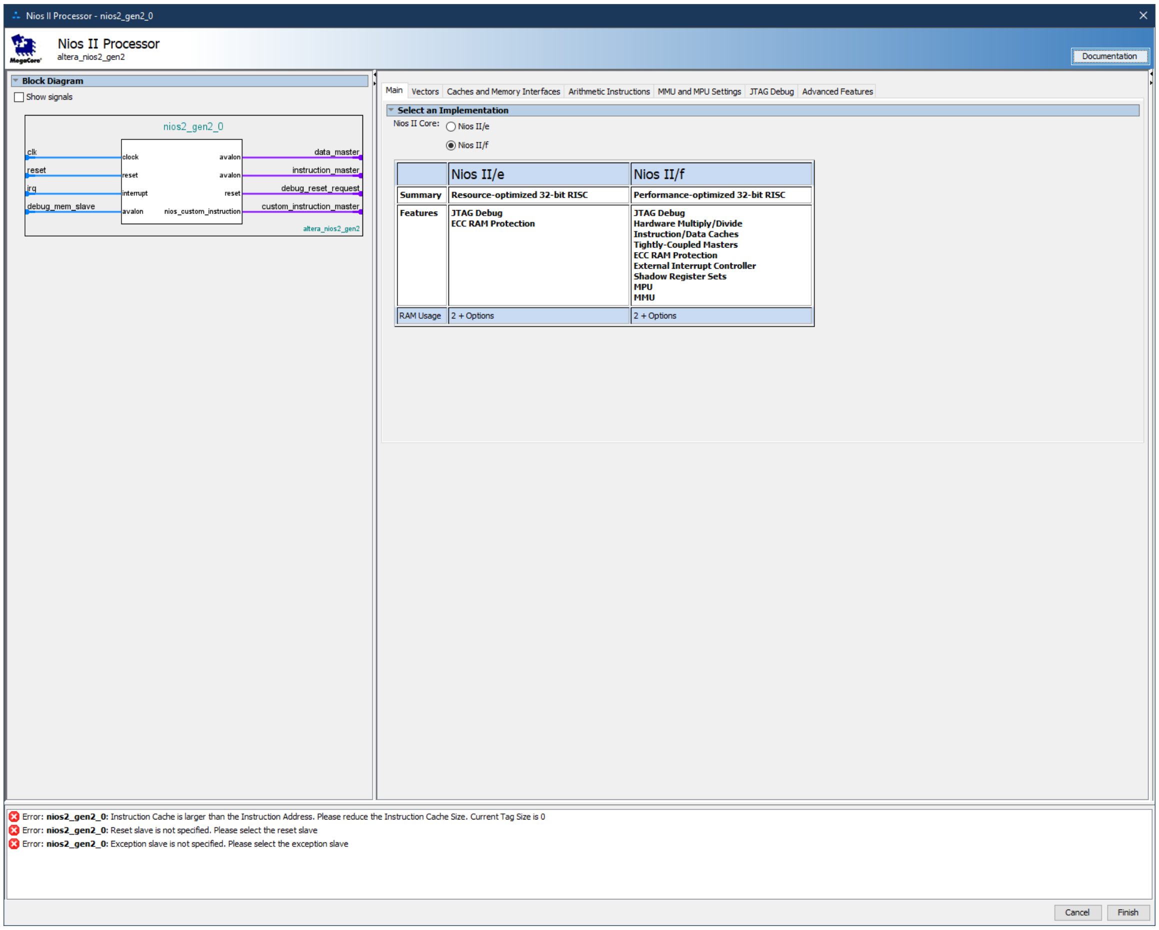
The Platform Designer tool is launched. By default, a clock (clk_0) is added to the design. Platform Designer makes it easy to add IP blocks and make interconnections between the blocks.

2. The top left pane contains the IP Catalog with all the available IP blocks that come with Quartus Prime. In the search box, type NIOS.



```
Project
  New Component...
Library
  Basic Functions
    Simulation; Debug and Verification
      Simulation
        ● Nios II Custom Instruction Master BFM Intel FPGA IP
        ● Nios II Custom Instruction Slave BFM Intel FPGA IP
  Processors and Peripherals
    Co-Processors
      Nios II Custom Instructions
        ● Bitswap
        ● Custom Instruction Interconnect
        ● Custom Instruction Master Translator
        ● Custom Instruction Slave Translator
        ● Floating Point Hardware
        ● Floating Point Hardware 2
    Embedded Processors
      ● Nios II Processor
```

3. Expand the Processors and Peripherals and the Embedded Processors branches. Under Embedded Processors double-click on the Nios II Processor.
4. This will open the Nios II Configuration page. The first tab is used to select the type of core Nios II/e or Nios II/f. We will keep the Nios II/f default for this exercise. Click Finish.



□ Show signals

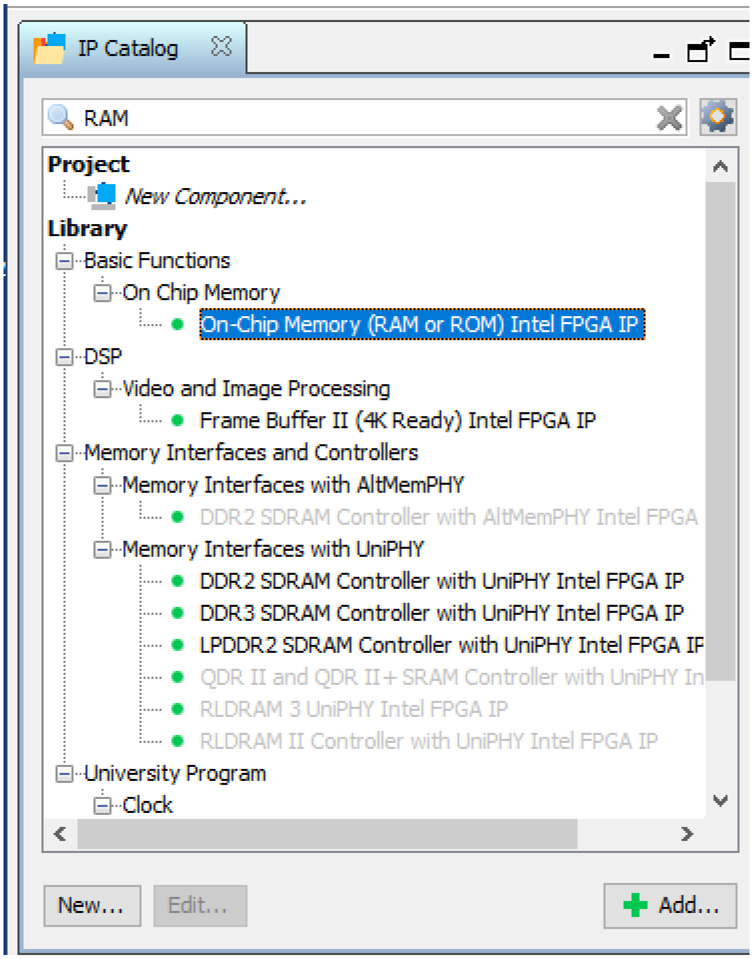
Select an Implementation

Nios II Core: ○ Nios II/e
● Nios II/f

	Nios II/e	Nios II/f
Summary	Resource-optimized 32-bit RISC	Performance-optimized 32-bit RISC
Features	JTAG Debug ECC RAM Protection	JTAG Debug Hardware Multiply/Divide Instruction/Data Caches Tightly-Coupled Masters ECC RAM Protection External Interrupt Controller Shadow Register Sets MPU MMU
RAM Usage	2 + Options	2 + Options

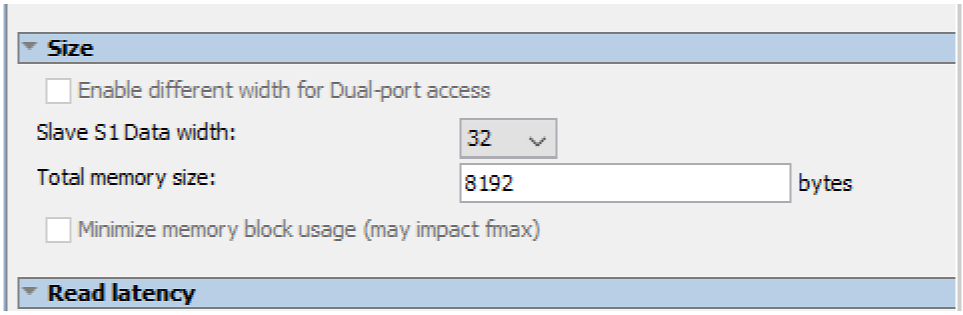
```
Error: nios2_gen2_0: Instruction Cache is larger than the Instruction Address. Please reduce the Instruction Cache Size. Current Tag Size is 0
Error: nios2_gen2_0: Reset slave is not specified. Please select the reset slave
Error: nios2_gen2_0: Exception slave is not specified. Please select the exception slave
```

5. Now let’s add the RAM IP block. In the IP Catalog enter RAM in the search box.
6. Double-click on On-chip Memory (RAM or ROM) in the Intel FPGA IP.


IP Catalog
Project
New Component...
Library
Basic Functions
On Chip Memory
On-Chip Memory (RAM or ROM) Intel FPGA IP
DSP
Video and Image Processing
Frame Buffer II (4K Ready) Intel FPGA IP
Memory Interfaces and Controllers
Memory Interfaces with AltMemPHY
DDR2 SDRAM Controller with AltMemPHY Intel FPGA
Memory Interfaces with UniPHY
DDR2 SDRAM Controller with UniPHY Intel FPGA IP
DDR3 SDRAM Controller with UniPHY Intel FPGA IP
LPDDR2 SDRAM Controller with UniPHY Intel FPGA IP
QDR II and QDR II+ SRAM Controller with UniPHY In
RLDRAM 3 UniPHY Intel FPGA IP
RLDRAM II Controller with UniPHY Intel FPGA IP
University Program
Clock
New...
Edit...
Add...

RAM

7. The configuration page will appear. Change the Total memory size to 8192 and click Finish.



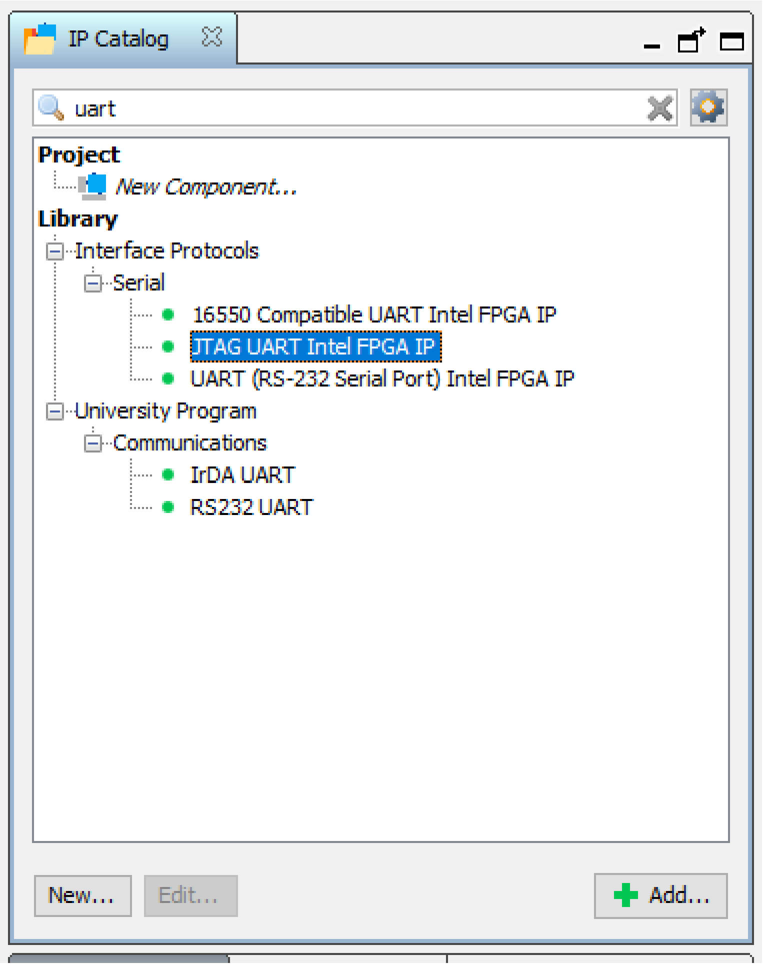
□ Enable different width for Dual-port access

Slave S1 Data width: 32

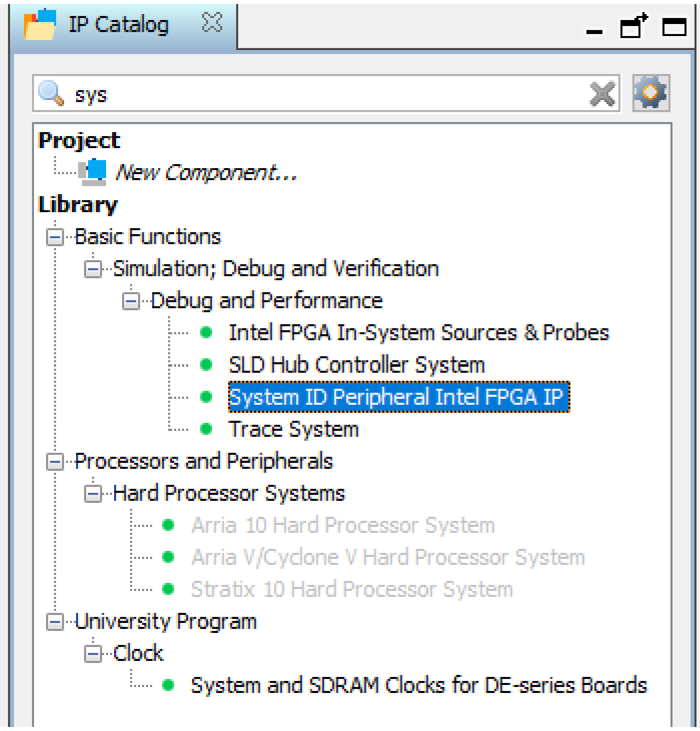
Total memory size: 8192 bytes

□ Minimize memory block usage (may impact fmax)

8. In the IP Catalog search, enter uart.
9. Double-click on the JTAG UART Intel FPGA IP.



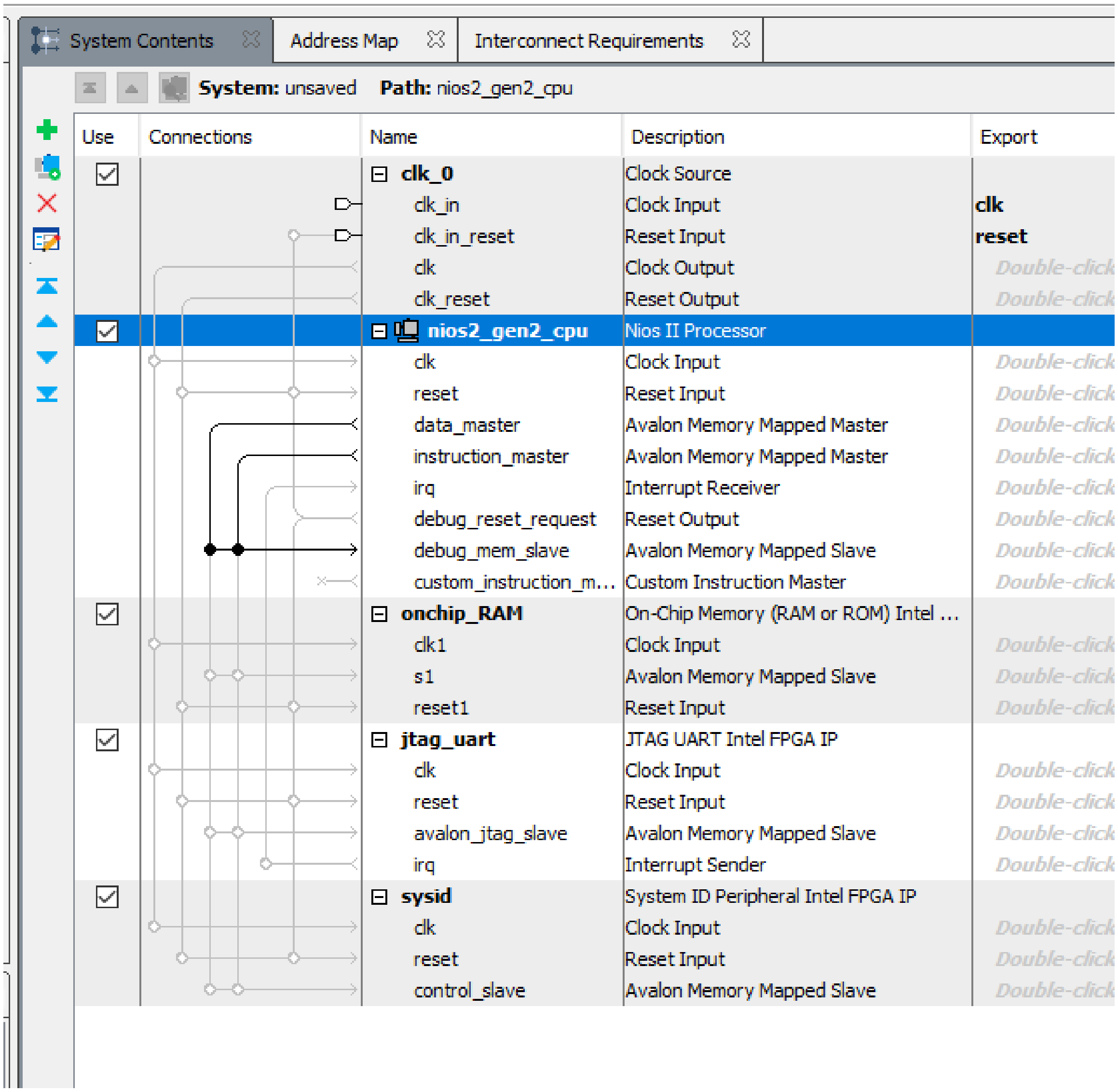
10. A configuration page will appear. There are no changes to be made. Click Finish.
11. In the IP Catalog search, enter the system ID.
12. Double-click on the System ID Peripheral Intel FPGA IP.



IP Catalog
sys
Project
New Component...
Library
Basic Functions
Simulation; Debug and Verification
Debug and Performance
Intel FPGA In-System Sources & Probes
SLD Hub Controller System
System ID Peripheral Intel FPGA IP
Trace System
Processors and Peripherals
Hard Processor Systems
Arria 10 Hard Processor System
Arria V/Cyclone V Hard Processor System
Stratix 10 Hard Processor System
University Program
Clock
System and SDRAM Clocks for DE-series Boards

13. A configuration page will appear. There are no changes to be made. Click Finish.
14. Rename the IP blocks as follows:

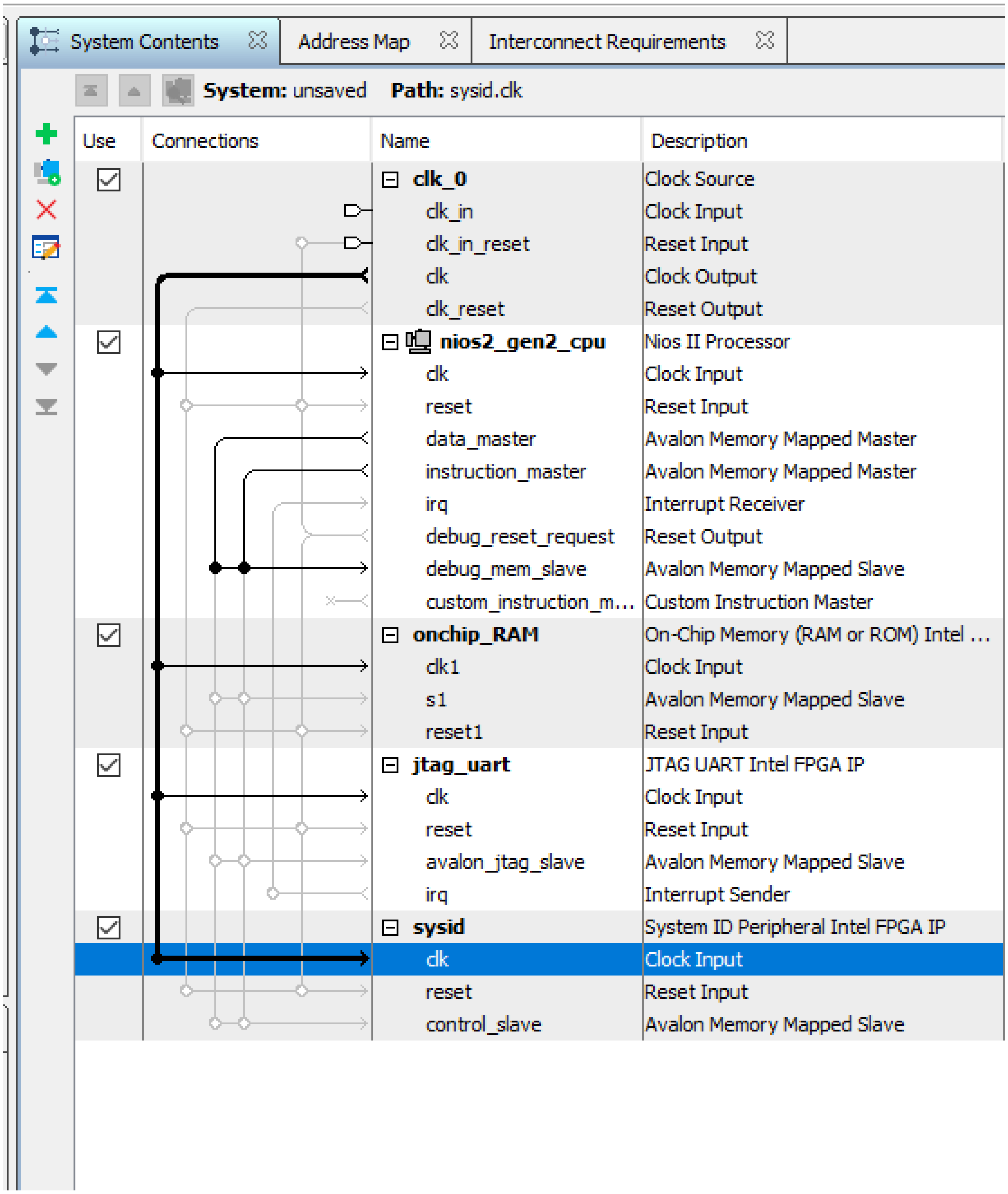
- nios2_gen2_cpu
- onchip_RAM
- jtag_uart
- sysid



System: unsaved **Path:** nios2_gen2_cpu

Use	Connections	Name	Description	Export
☑		⊟ **clk_0**	Clock Source	
		clk_in	Clock Input	**clk**
		clk_in_reset	Reset Input	**reset**
		clk	Clock Output	*Double-click*
		clk_reset	Reset Output	*Double-click*
☑		⊟ **nios2_gen2_cpu**	Nios II Processor	
		clk	Clock Input	*Double-click*
		reset	Reset Input	*Double-click*
		data_master	Avalon Memory Mapped Master	*Double-click*
		instruction_master	Avalon Memory Mapped Master	*Double-click*
		irq	Interrupt Receiver	*Double-click*
		debug_reset_request	Reset Output	*Double-click*
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click*
		custom_instruction_m...	Custom Instruction Master	*Double-click*
☑		⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel ...	
		clk1	Clock Input	*Double-click*
		s1	Avalon Memory Mapped Slave	*Double-click*
		reset1	Reset Input	*Double-click*
☑		⊟ **jtag_uart**	JTAG UART Intel FPGA IP	
		clk	Clock Input	*Double-click*
		reset	Reset Input	*Double-click*
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click*
		irq	Interrupt Sender	*Double-click*
☑		⊟ **sysid**	System ID Peripheral Intel FPGA IP	
		clk	Clock Input	*Double-click*
		reset	Reset Input	*Double-click*
		control_slave	Avalon Memory Mapped Slave	*Double-click*

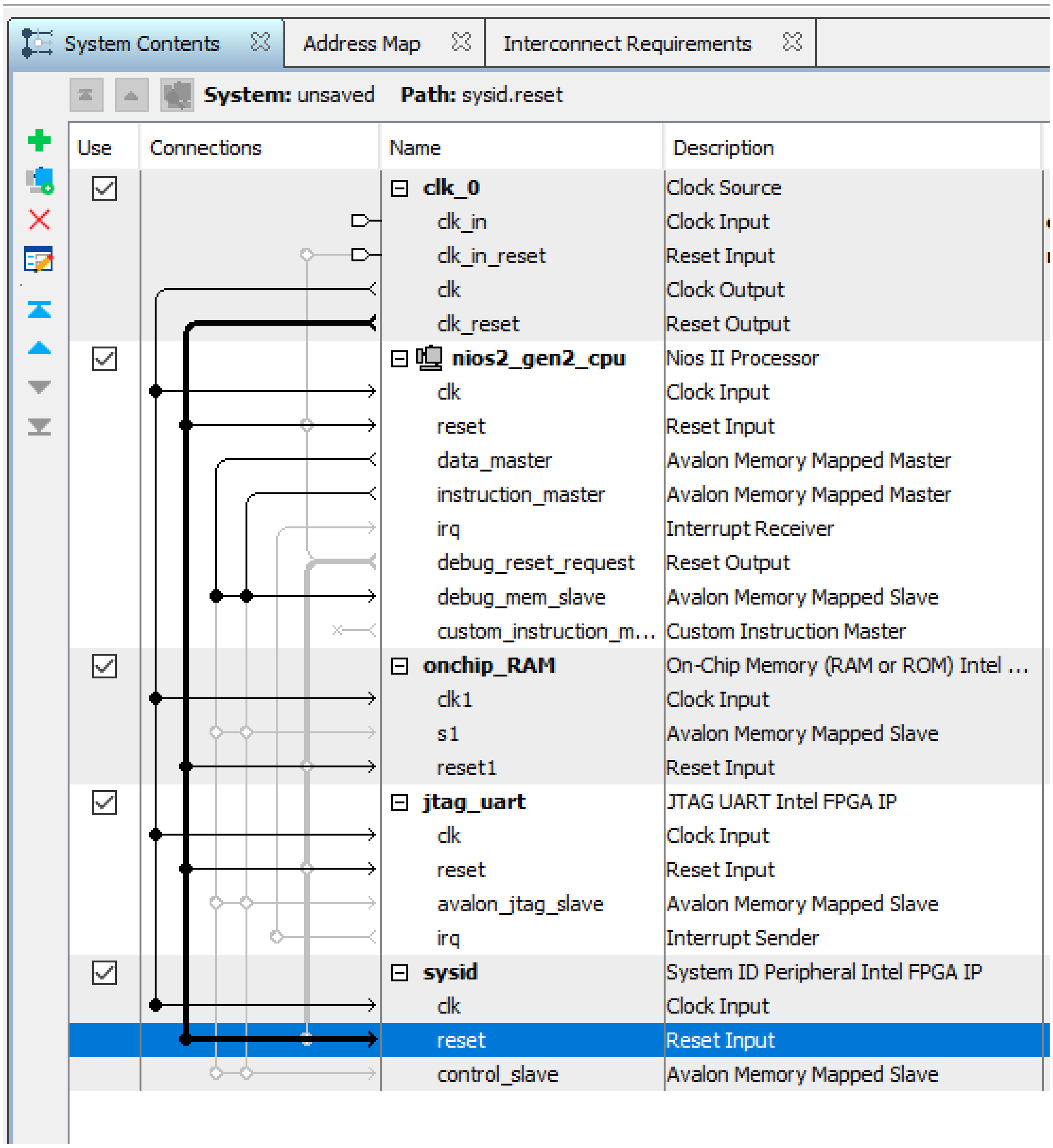
15. Now we need to wire the IP blocks together. First, wire all the clk lines together by clicking on the dots for all four IP blocks.



System: unsaved **Path:** sysid.clk

Use	Connections	Name	Description
☑		⊟ **clk_0**	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
☑		⊟ **nios2_gen2_cpu**	Nios II Processor
		clk	Clock Input
		reset	Reset Input
		data_master	Avalon Memory Mapped Master
		instruction_master	Avalon Memory Mapped Master
		irq	Interrupt Receiver
		debug_reset_request	Reset Output
		debug_mem_slave	Avalon Memory Mapped Slave
		custom_instruction_m...	Custom Instruction Master
☑		⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel ...
		clk1	Clock Input
		s1	Avalon Memory Mapped Slave
		reset1	Reset Input
☑		⊟ **jtag_uart**	JTAG UART Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		avalon_jtag_slave	Avalon Memory Mapped Slave
		irq	Interrupt Sender
☑		⊟ **sysid**	System ID Peripheral Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

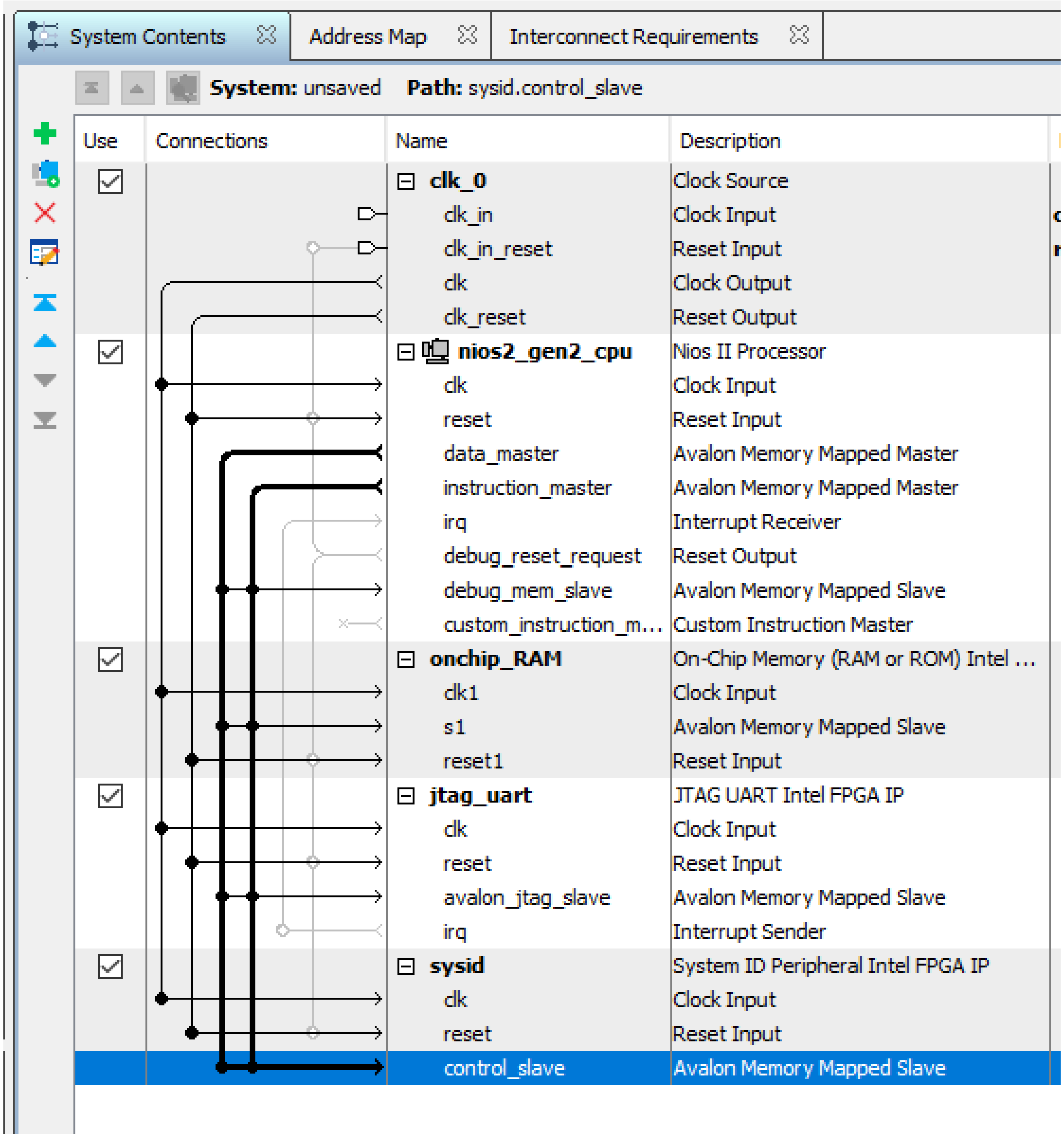
16. Next, connect all the reset lines together by clicking on the dots for all four IP blocks.



System: unsaved **Path:** sysid.reset

Use	Connections	Name	Description
☑		⊟ **clk_0**	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
☑		⊟ **nios2_gen2_cpu**	Nios II Processor
		clk	Clock Input
		reset	Reset Input
		data_master	Avalon Memory Mapped Master
		instruction_master	Avalon Memory Mapped Master
		irq	Interrupt Receiver
		debug_reset_request	Reset Output
		debug_mem_slave	Avalon Memory Mapped Slave
		custom_instruction_m…	Custom Instruction Master
☑		⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel …
		clk1	Clock Input
		s1	Avalon Memory Mapped Slave
		reset1	Reset Input
☑		⊟ **jtag_uart**	JTAG UART Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		avalon_jtag_slave	Avalon Memory Mapped Slave
		irq	Interrupt Sender
☑		⊟ **sysid**	System ID Peripheral Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

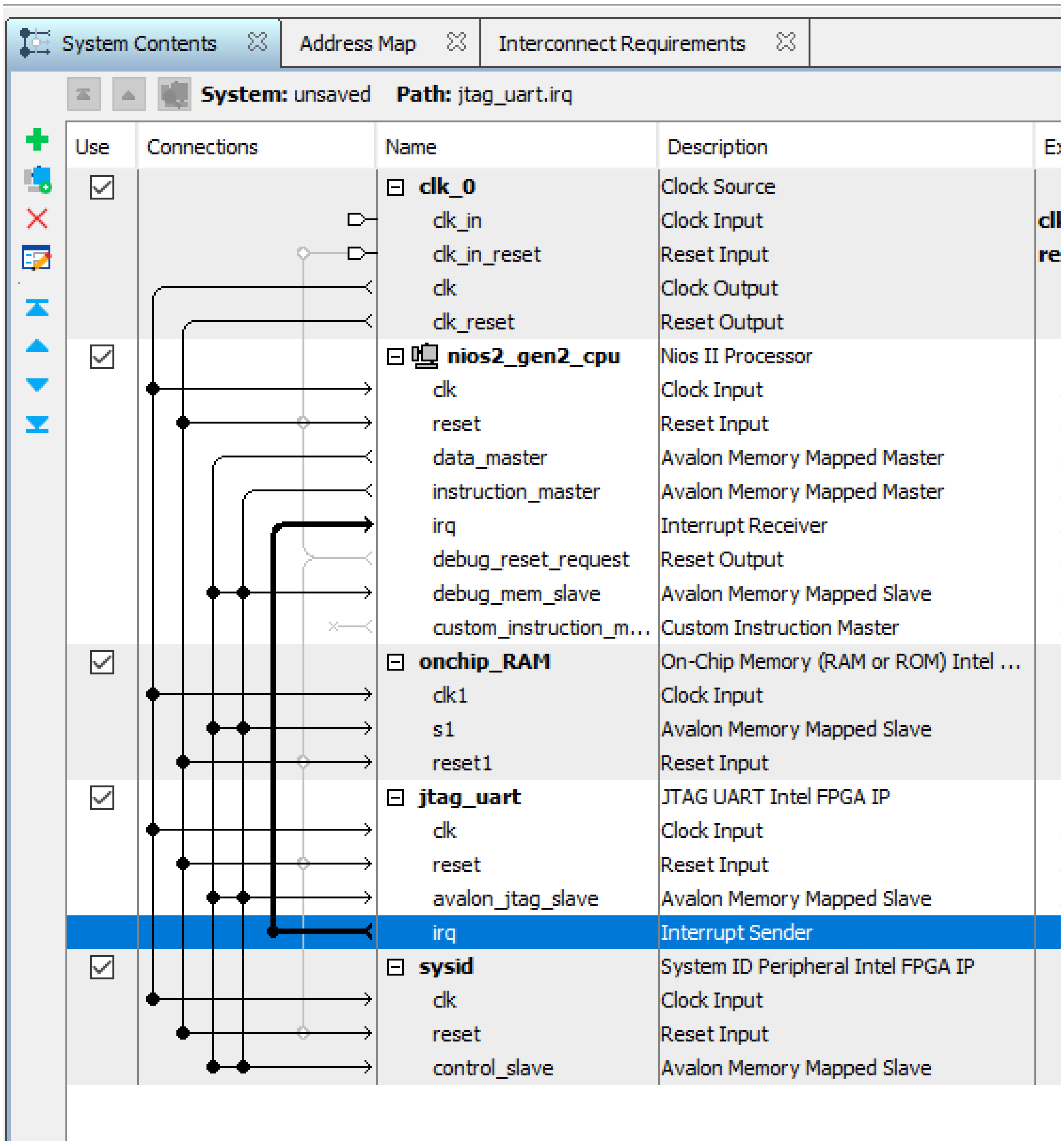
17. The memory lines have to be connected together. The s1 line in the RAM and Avalon_jtag_slave line in jtag_uart will be connected to the nios2_gen_cpu data_master and instruction master lines. Click on the dots for each IP block.
18. Connect the sysid control_slave to nios2_gen_cpu data_master.



System: unsaved **Path:** sysid.control_slave

Use	Connections	Name	Description
☑		⊟ **clk_0**	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
☑		⊟ **nios2_gen2_cpu**	Nios II Processor
		clk	Clock Input
		reset	Reset Input
		data_master	Avalon Memory Mapped Master
		instruction_master	Avalon Memory Mapped Master
		irq	Interrupt Receiver
		debug_reset_request	Reset Output
		debug_mem_slave	Avalon Memory Mapped Slave
		custom_instruction_m...	Custom Instruction Master
☑		⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel ...
		clk1	Clock Input
		s1	Avalon Memory Mapped Slave
		reset1	Reset Input
☑		⊟ **jtag_uart**	JTAG UART Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		avalon_jtag_slave	Avalon Memory Mapped Slave
		irq	Interrupt Sender
☑		⊟ **sysid**	System ID Peripheral Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

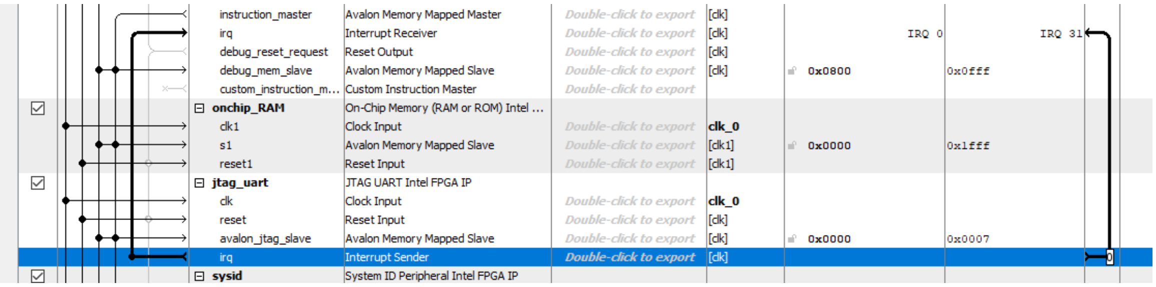
19. Finally, connect the jtaq_uart irq line to the nios2_gen2_cpu.



System: unsaved **Path:** jtag_uart.irq

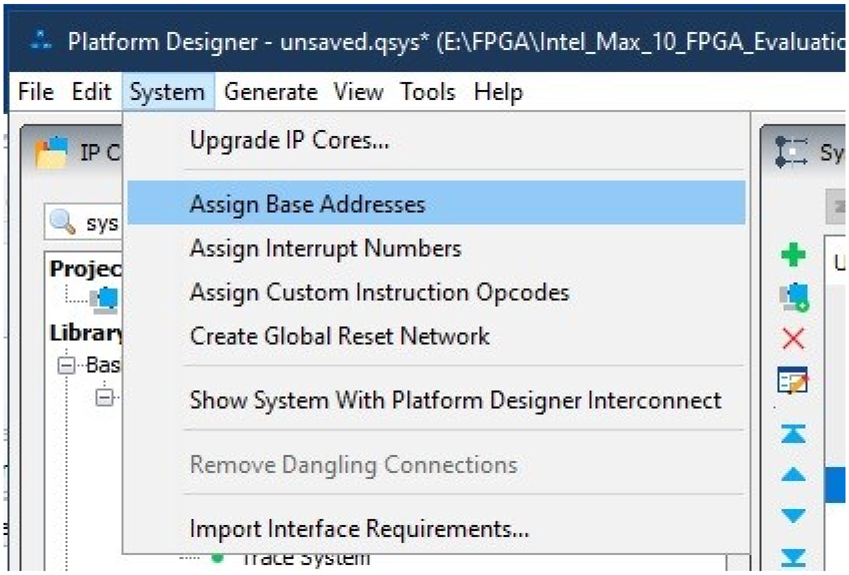
Use	Name	Description	E
☑	⊟ **clk_0**	Clock Source	
	clk_in	Clock Input	cll
	clk_in_reset	Reset Input	re
	clk	Clock Output	
	clk_reset	Reset Output	
☑	⊟ **nios2_gen2_cpu**	Nios II Processor	
	clk	Clock Input	
	reset	Reset Input	
	data_master	Avalon Memory Mapped Master	
	instruction_master	Avalon Memory Mapped Master	
	irq	Interrupt Receiver	
	debug_reset_request	Reset Output	
	debug_mem_slave	Avalon Memory Mapped Slave	
	custom_instruction_m…	Custom Instruction Master	
☑	⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel …	
	clk1	Clock Input	
	s1	Avalon Memory Mapped Slave	
	reset1	Reset Input	
☑	⊟ **jtag_uart**	JTAG UART Intel FPGA IP	
	clk	Clock Input	
	reset	Reset Input	
	avalon_jtag_slave	Avalon Memory Mapped Slave	
	irq	Interrupt Sender	
☑	⊟ **sysid**	System ID Peripheral Intel FPGA IP	
	clk	Clock Input	
	reset	Reset Input	
	control_slave	Avalon Memory Mapped Slave	

20. If you scroll to the right, the irq is given a default value.

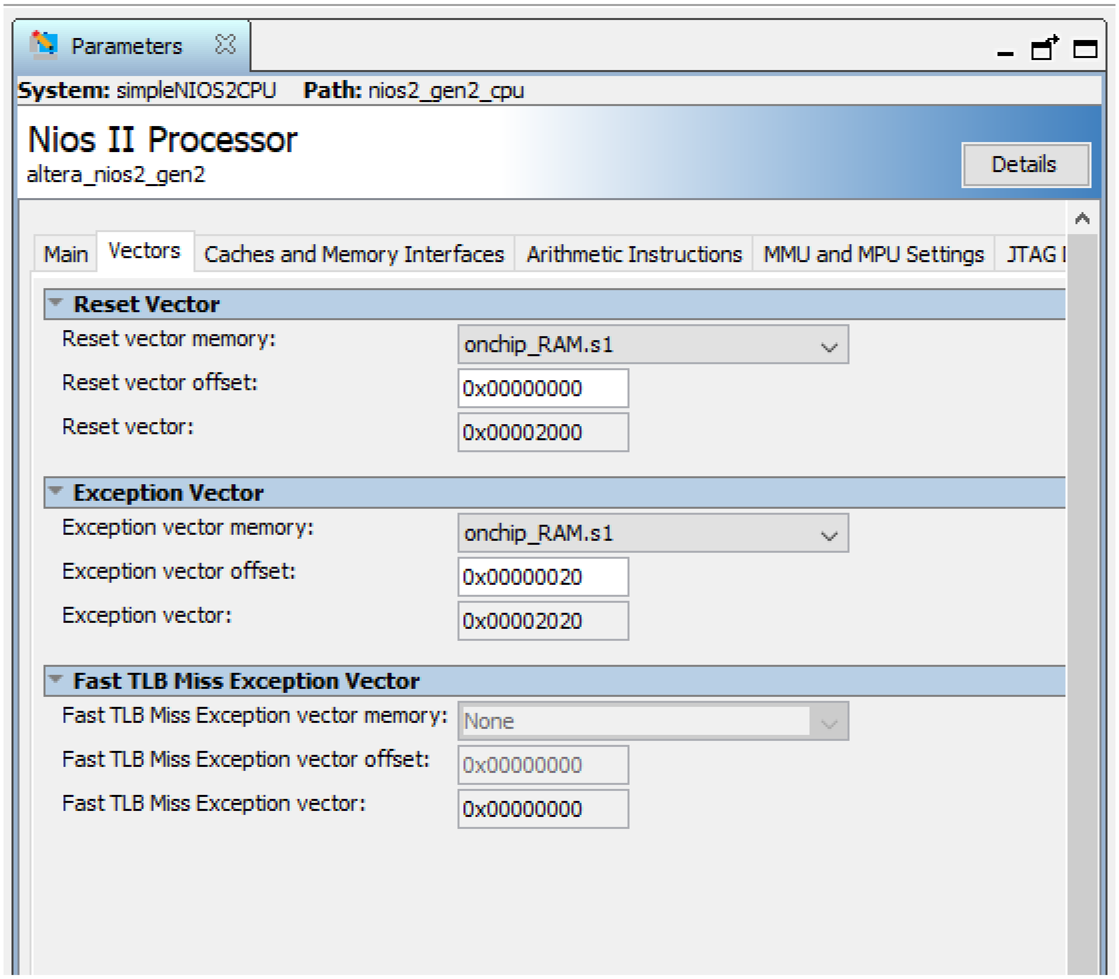


	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	irq	Interrupt Receiver	Double-click to export	[clk]		IRQ 0	IRQ 31
	debug_reset_request	Reset Output	Double-click to export	[clk]			
	debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0800	0x0fff	
	custom_instruction_m…	Custom Instruction Master	Double-click to export				
☑	⊟ **onchip_RAM**	On-Chip Memory (RAM or ROM) Intel …					
	clk1	Clock Input	Double-click to export	**clk_0**			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000	0x1fff	
	reset1	Reset Input	Double-click to export	[clk1]			
☑	⊟ **jtag_uart**	JTAG UART Intel FPGA IP					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000	0x0007	
	irq	Interrupt Sender	Double-click to export	[clk]			0
☑	⊟ **sysid**	System ID Peripheral Intel FPGA IP					

21. Let’s assign a base address. From the menu, select System->Assign Base Address. This will remove a number of errors from the message box.


Platform Designer - unsaved.qsys* (E:\FPGA\Intel_Max_10_FPGA_Evaluatic
File Edit System Generate View Tools Help
Upgrade IP Cores...
Assign Base Addresses
Assign Interrupt Numbers
Assign Custom Instruction Opcodes
Create Global Reset Network
Show System With Platform Designer Interconnect
Remove Dangling Connections
Import Interface Requirements...
IP C
sys
Projec
Library
Bas
Trace System
Sy
U

22. Finally, let's set the reset and exception vector addresses. Double-click on the nios2_gen2_cpu to open the configuration page.
23. Click on the Vectors tab.
24. Change the Reset vector memory drop-down to onchip_RAM.s1.
25. Change the Exception vector memory drop-down to onchip_RAM.s1.



Parameters

System: simpleNIOS2CPU **Path:** nios2_gen2_cpu

Nios II Processor
altera_nios2_gen2

Reset Vector

Reset vector memory: onchip_RAM.s1
Reset vector offset: 0x00000000
Reset vector: 0x00002000

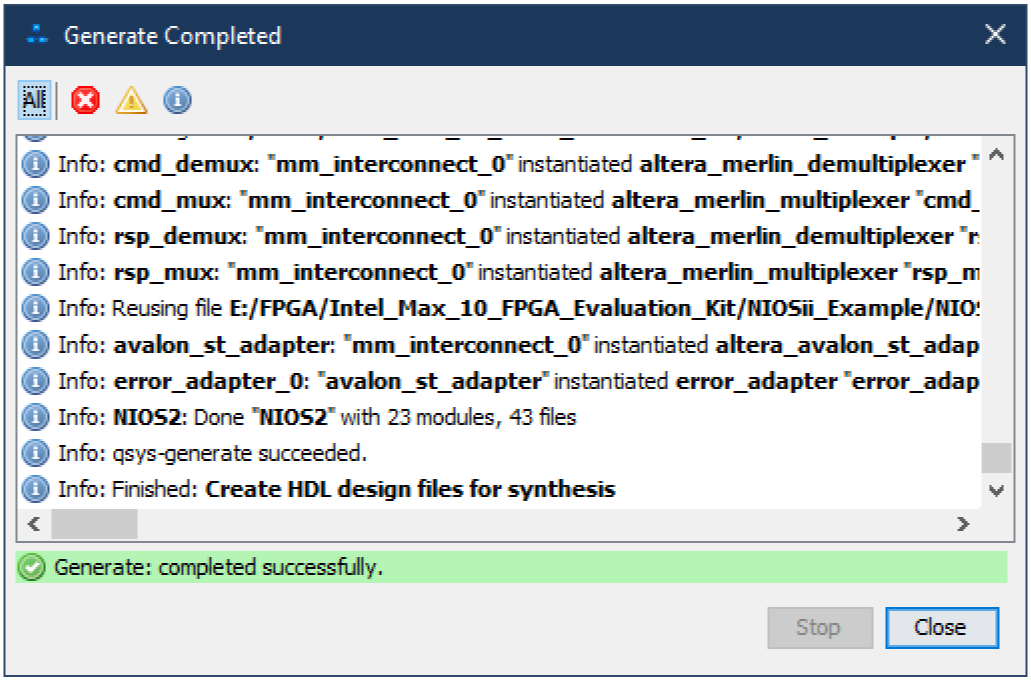
Exception Vector

Exception vector memory: onchip_RAM.s1
Exception vector offset: 0x00000020
Exception vector: 0x00002020

Fast TLB Miss Exception Vector

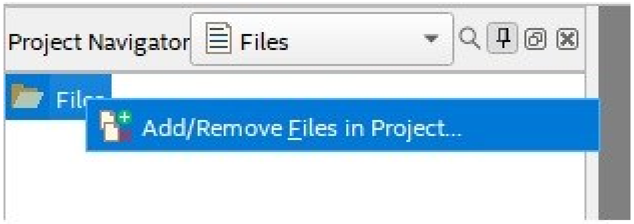
Fast TLB Miss Exception vector memory: None
Fast TLB Miss Exception vector offset: 0x00000000
Fast TLB Miss Exception vector: 0x00000000

26. Click on Generate HDL…
27. Keep the defaults and click the Generate button.
28. A dialog will appear asking you to save the design, click Save.
29. Give the name as NIOS2.qsys and click Save.
30. Once the save has been completed, click Close.
31. The generate process kicks off. The processes should succeed, click Close.

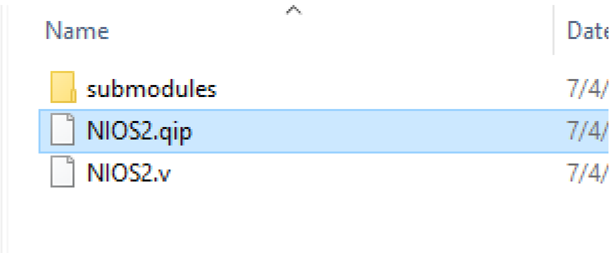


```
Info: cmd_demux: "mm_interconnect_0" instantiated altera_merlin_demultiplexer "
Info: cmd_mux: "mm_interconnect_0" instantiated altera_merlin_multiplexer "cmd_
Info: rsp_demux: "mm_interconnect_0" instantiated altera_merlin_demultiplexer "r
Info: rsp_mux: "mm_interconnect_0" instantiated altera_merlin_multiplexer "rsp_m
Info: Reusing file E:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOSii_Example/NIO
Info: avalon_st_adapter: "mm_interconnect_0" instantiated altera_avalon_st_adap
Info: error_adapter_0: "avalon_st_adapter" instantiated error_adapter "error_adap
Info: NIOS2: Done "NIOS2" with 23 modules, 43 files
Info: qsys-generate succeeded.
Info: Finished: Create HDL design files for synthesis
Generate: completed successfully.
```

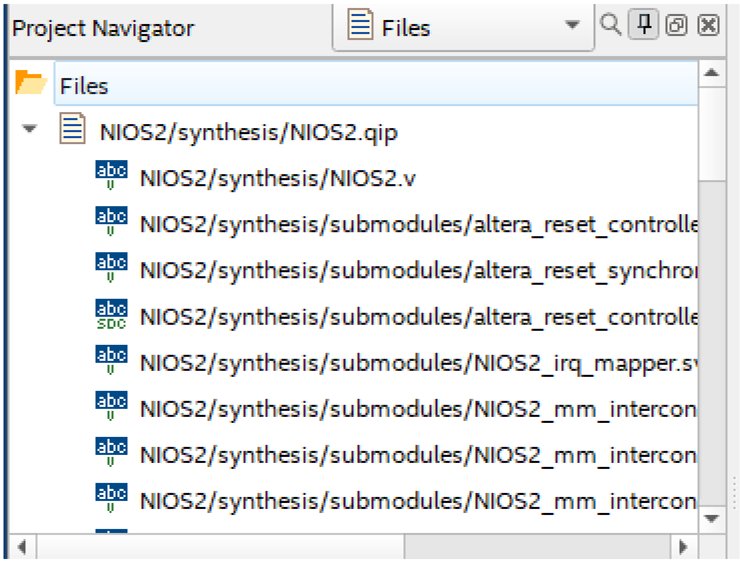
32. Click Finish to close the design.
33. Quartus then reminds you to add the new design to the project. Click Ok.
34. In the Project Navigator, click on the drop-down and select Files.
35. Right-Click on Files and select Add/Remote Files in Project.



36. A Settings – NIOS2CPU page appears with Files on the left highlighted. Click the three dots, browse button for File name, and navigate to \NIOSii_Example\NIOS2\synthesis folder.
37. Click on NIOS2.qip file and click open



38. Click OK to close the Settings - NIOS2CPU page. The qip file is added to the Project navigator list. Underneath are all the Verilog files that were generated by Platform Builder.



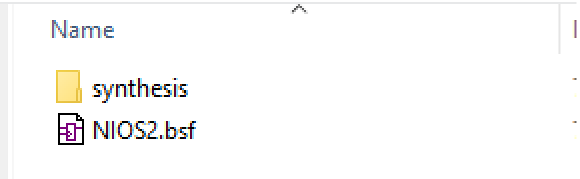
Project Navigator | Files

Files
- NIOS2/synthesis/NIOS2.qip
  - NIOS2/synthesis/NIOS2.v
  - NIOS2/synthesis/submodules/altera_reset_controlle
  - NIOS2/synthesis/submodules/altera_reset_synchro
  - NIOS2/synthesis/submodules/altera_reset_controlle
  - NIOS2/synthesis/submodules/NIOS2_irq_mapper.s
  - NIOS2/synthesis/submodules/NIOS2_mm_intercon
  - NIOS2/synthesis/submodules/NIOS2_mm_intercon
  - NIOS2/synthesis/submodules/NIOS2_mm_intercon

1.1.3 Create the Design Step 2: Block Diagram

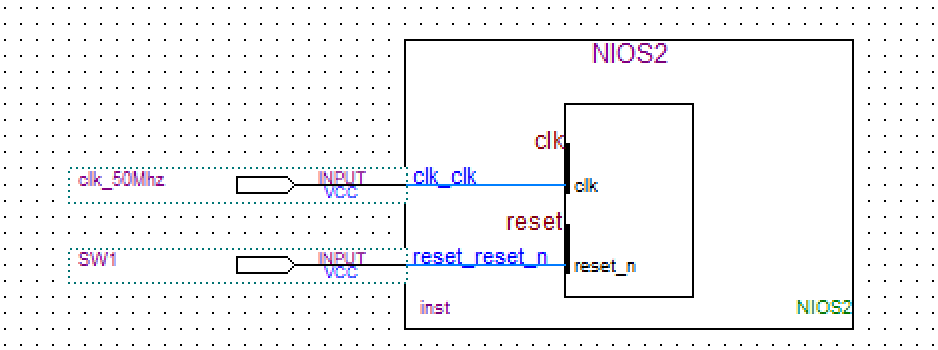
With the qip file and all the Verilog files added to the project, let's create the block diagram and complete the design.

1. From the menu, select New->Block Diagram/Schematic File or the icon from the toolbar. Click Ok.
2. The symbol window appears. Double-click on the symbol window and the symbol dialog appears.
3. Click on the 3 dots to open the file browser.
4. Browse to \NIOSii_Example\NIOS2 folder and open the NIOS2.bsf file.



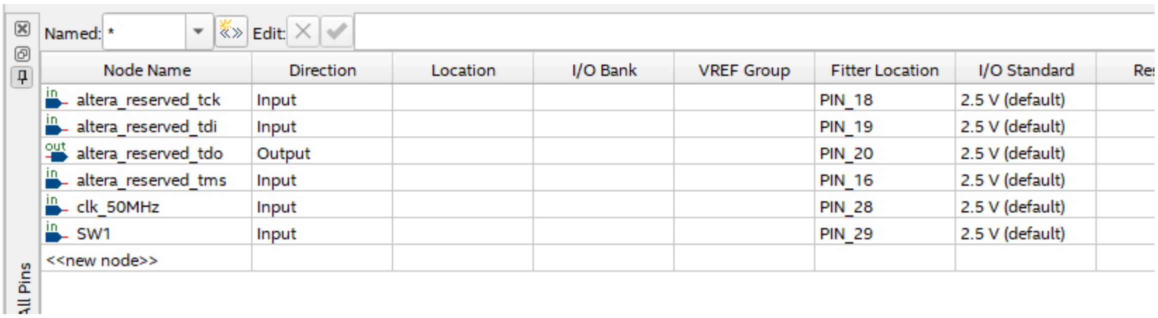
Name
- synthesis
- NIOS2.bsf

5. The symbol for the NIOS2 appears. Click OK to add the symbol to the schematic.
6. Drag the mouse with the NIOS2 symbol to a location on the diagram and then left-click to drop it in place.
7. Right-click on the NIOS2 symbol and select Generate Pins for Symbol Ports.
8. Change the name of the clk_clk pin to clk_50MHz.
9. Change the name of the reset_reset_n to SW1. The SW1 switch on the evaluation kit is connected to the FPGA DEV_CLRN pin. The circuit for SW1 is logic 1 on startup and logic0 when pressed.



10. Save the schematic as NIOS2CPU.bdf.
11. In the Task pane on the left, double-click on Fitter (Place & Route) to start the task. The analysis will take some time, and it should succeed in the end. This step helps to diagnose any errors and finds the Node Names for the pin assignments in the next step.

12. Once the process completes, the pin assignments need to be set. From the menu select Assignments->Pin Planner or click on the icon from the toolbar. The analysis that was just run populated the Node Name list at the bottom of the Pin Planner dialog.



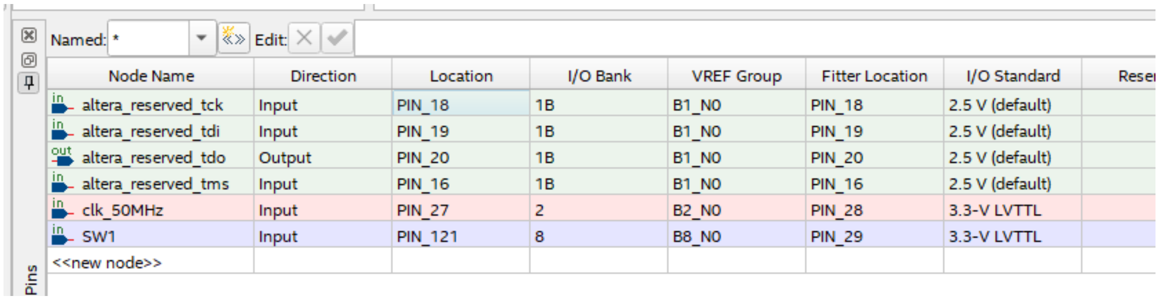
Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Re
altera_reserved_tck	Input				PIN_18	2.5 V (default)	
altera_reserved_tdi	Input				PIN_19	2.5 V (default)	
altera_reserved_tdo	Output				PIN_20	2.5 V (default)	
altera_reserved_tms	Input				PIN_16	2.5 V (default)	
clk_50MHz	Input				PIN_28	2.5 V (default)	
SW1	Input				PIN_29	2.5 V (default)	
<<new node>>							

13. Using the board schematic, locate the pins for the SW1 and the 50Mhz clock. Set the Location values for both node names. For the MAX 10 – 10M08 Evaluation Board, these values are as follows:

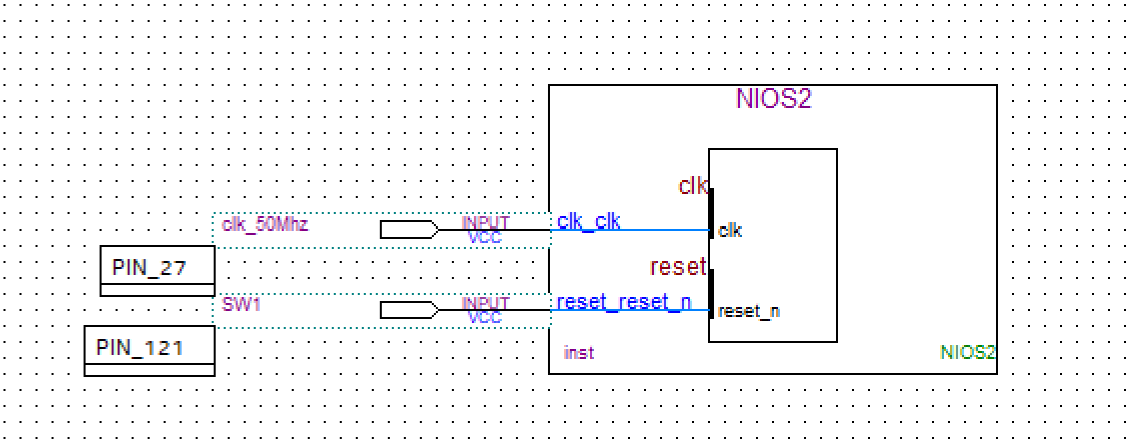
Node Name	Location
SW1	PIN_121
Clk_50MHz:	PIN_27
altera_reserved_tck	PIN_18
altera_reserved_tdi	PIN_19
altera_reserved_tdo	PIN_20
altera_reserved_tms	PIN_16

14. Set the I/O Standard to 3.3V-LVTTL for both pins. You can see from the schematic that all of the I/O are tied to 3.3V.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Rese
altera_reserved_tck	Input	PIN_18	1B	B1_N0	PIN_18	2.5 V (default)	
altera_reserved_tdi	Input	PIN_19	1B	B1_N0	PIN_19	2.5 V (default)	
altera_reserved_tdo	Output	PIN_20	1B	B1_N0	PIN_20	2.5 V (default)	
altera_reserved_tms	Input	PIN_16	1B	B1_N0	PIN_16	2.5 V (default)	
clk_50MHz	Input	PIN_27	2	B2_N0	PIN_28	3.3-V LVTTL	
SW1	Input	PIN_121	8	B8_N0	PIN_29	3.3-V LVTTL	
<<new node>>							

15. Close the Pin Planner when finished. The diagram gets updated with the pin numbers.

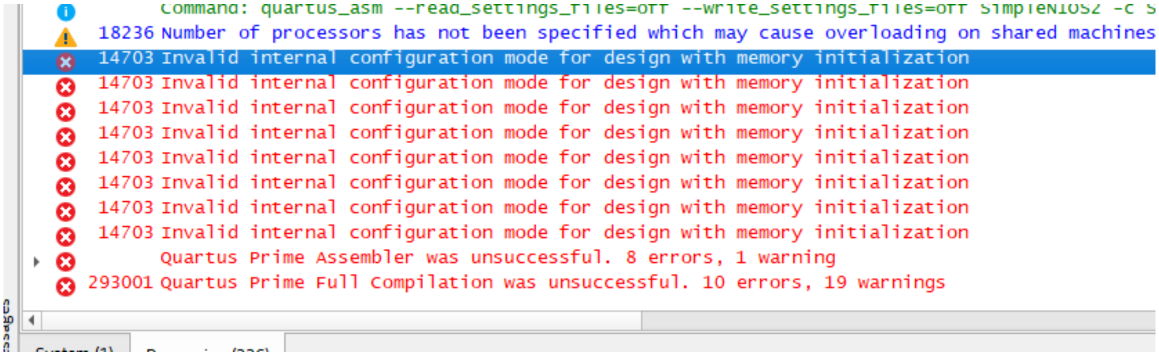


16. Save the project.

Note: A best practice at this point would be to make a backup of the project folder. Quartus can crash unexpectedly, since it appears to be written in Java.

17. Finally, compile the design. In the Task pane, right-click on Compile and Design and select Start from the context menu, or you can click on the ▶ symbol in the toolbar.

When the compilation hits the Assembler task, an error will appear.



```
        Command: quartus_asm --read_settings_files=off --write_settings_files=off SimpleNios2 -c S
  18236 Number of processors has not been specified which may cause overloading on shared machines
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
  14703 Invalid internal configuration mode for design with memory initialization
        Quartus Prime Assembler was unsuccessful. 8 errors, 1 warning
 293001 Quartus Prime Full Compilation was unsuccessful. 10 errors, 19 warnings
```

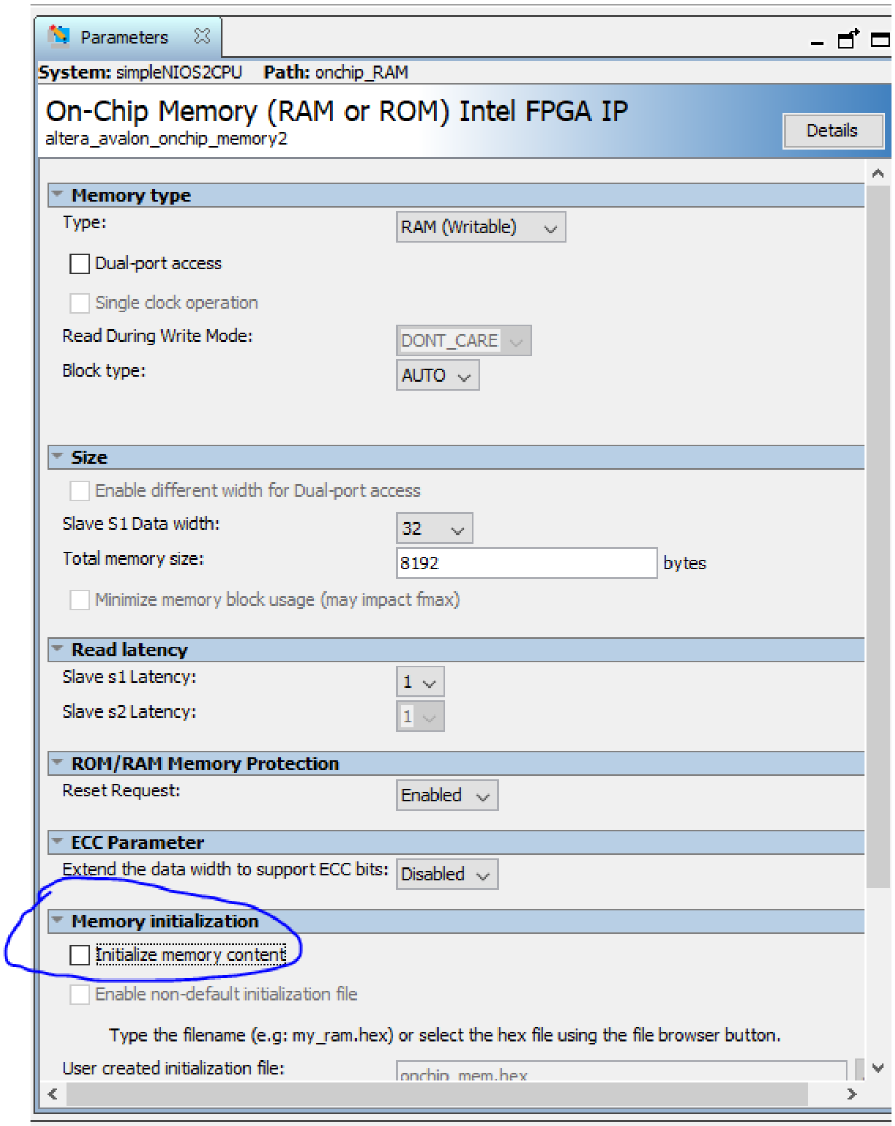
Performing an Internet search reveals the solution on intel.com. It is a known problem for the Max 10: Error (14703): Invalid internal configuration mode for design with... (intel.com)

From this page:

> ***Bug ID:*** *FB: 466229*;
>
> ***Description***
> *You may see this error while compiling a custom FIFO or a RAM block in Quartus® Prime software Standard / Lite version for a MAX® 10 device.*
>
> *This error is seen because MAX 10 Compact variants do not support memory initialization. If you have not provided any mif file for your custom design and still see this error in Quartus Prime software, it may be because a mif file is being inferred by the RTL coding style*
>
> ***Resolution***
> *Signal declaration for memory_type should be changed from*
>
> ```
>    signal mem : memory_type :=(others => (others => '0'));
> ```
>
> *to*
>
> ```
>    signal mem : memory_type;
> ```
>
> *This is to ensure that memory is not initialized and there is no compilation error in the Assembler stage.*

The solution is dealing with the code, but there is a simple solution back in the onchip_RAM confirmation page.

18. Open Platform Builder and open the NIOS2.qsys file.
19. After the design loads, double-click on the onchip_RAM to bring up the configuration page.
20. Uncheck the “Initialize memory content”.



Parameters

System: simpleNIOS2CPU Path: onchip_RAM

On-Chip Memory (RAM or ROM) Intel FPGA IP
altera_avalon_onchip_memory2

Memory type

Type: RAM (Writable)

□ Dual-port access

□ Single clock operation

Read During Write Mode: DONT_CARE

Block type: AUTO

Size

□ Enable different width for Dual-port access

Slave S1 Data width: 32

Total memory size: 8192 bytes

□ Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1

Slave s2 Latency: 1

ROM/RAM Memory Protection

Reset Request: Enabled

ECC Parameter

Extend the data width to support ECC bits: Disabled

Memory initialization

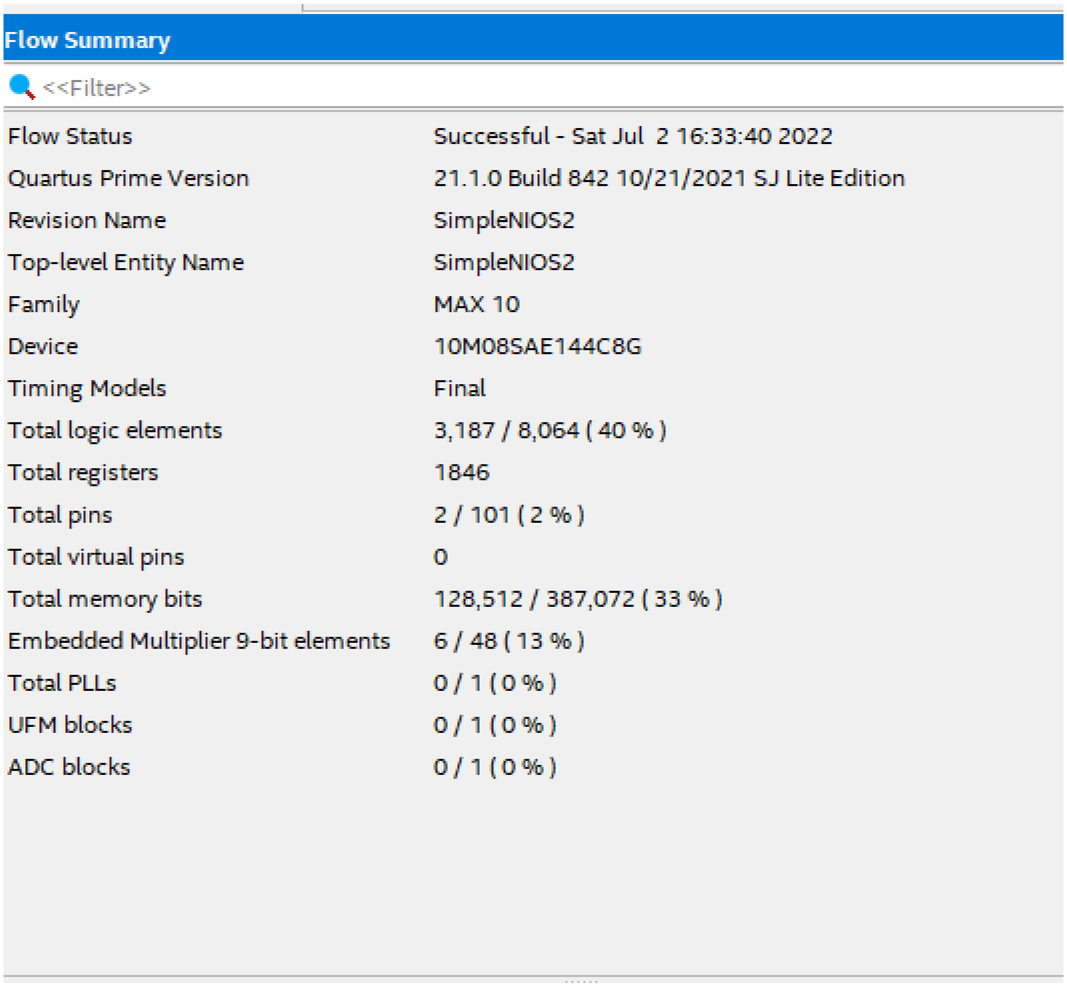
□ Initialize memory content

□ Enable non-default initialization file

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

User created initialization file: onchip_mem.hex

21. Click on Generate HDL…
22. Click Close after the save process completes.
23. Click Close after the generation completes.
24. Click Finish to close Platform Builder.
25. The Quartus reminder appears. Click Close.
26. Compile the design again, and this time it should be successful.



Flow Summary

<<Filter>>

Flow Status	Successful - Sat Jul 2 16:33:40 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	SimpleNIOS2
Top-level Entity Name	SimpleNIOS2
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	3,187 / 8,064 (40 %)
Total registers	1846
Total pins	2 / 101 (2 %)
Total virtual pins	0
Total memory bits	128,512 / 387,072 (33 %)
Embedded Multiplier 9-bit elements	6 / 48 (13 %)
Total PLLs	0 / 1 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

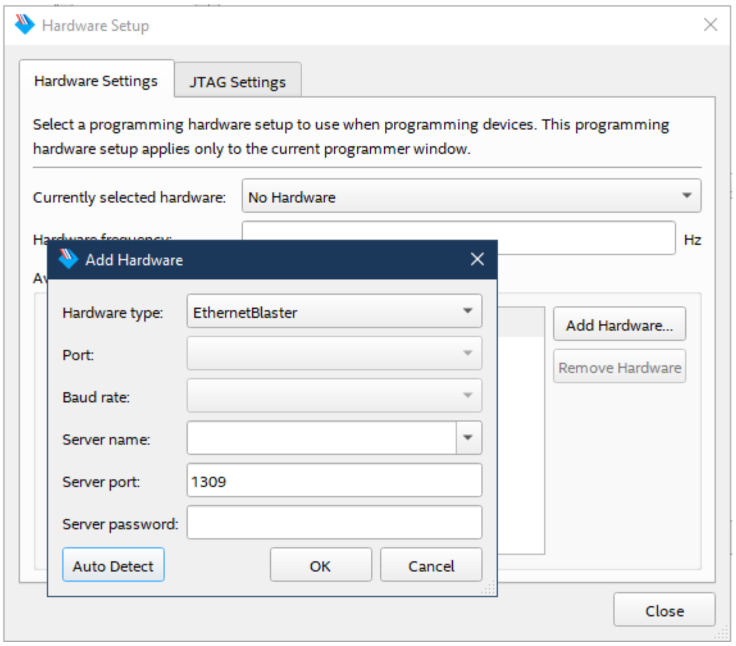
1.1.4 Program the Board

With the design compiled, we can now test the design on the board.

1. Connect the board and the programming cable together per the cable instructions.

Note: The MAX 10 – 10M08 Evaluation Kit doesn't come with a programming cable or built-in JTAG USB Blaster II. You will have to use either the USB Blaster II or EthernetBlaster II external cables. The EthernetBlaster II was used. DHCP setup was not working, so a direct Ethernet cable connection was made between a PC and the EthernetBlaster II. Set the static IP for the PC network card to 198.162.0.1. Access the EthernetBlaster II via a browser and then change the IP to a static IP that matches the network. The new IP address was used as the Server name. Your experience might be different.

2. Power on the board and the programming cable box.
3. In Quartus Prime, from the Task pane, right-click on Program Device (Open Programmer) and select Open from the context menu or click on the icon on the toolbar.
4. The Programmer dialog appears, click on the "Hardware Setup" button.
5. Click the Add hardware button, select the Hardware type, and fill in any remaining information, and click OK.



Hardware Setup

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No Hardware

Add Hardware

Hardware type: EthernetBlaster

Port:

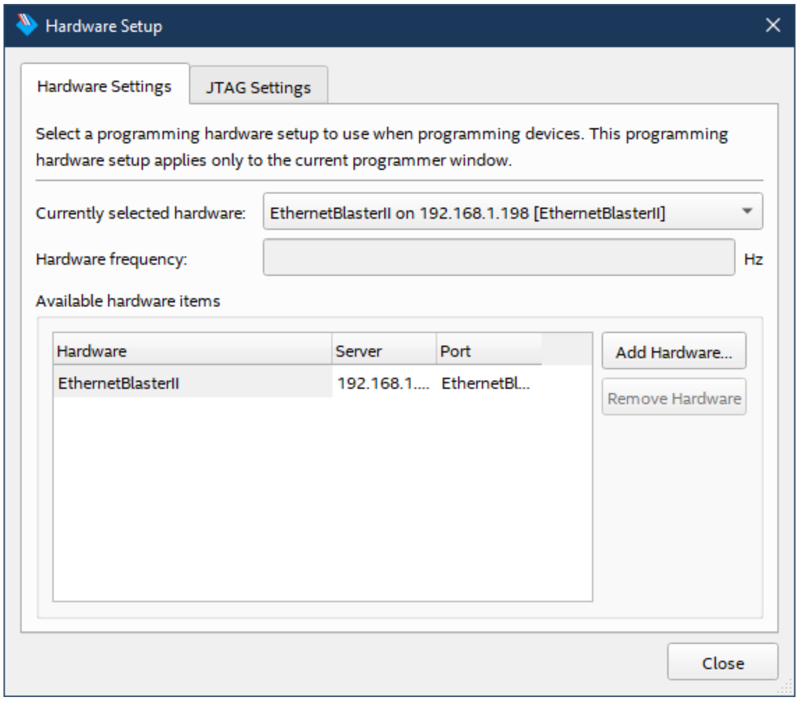
Baud rate:

Server name:

Server port: 1309

Server password:

6. The tool allows you to connect to a number of programming cables. We need to select the one for our board. In the “Currently selected hardware”, click the drop-down, select the hardware cable for the board, and click Close when finished



Hardware Setup

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: EthernetBlasterII on 192.168.1.198 [EthernetBlasterII]

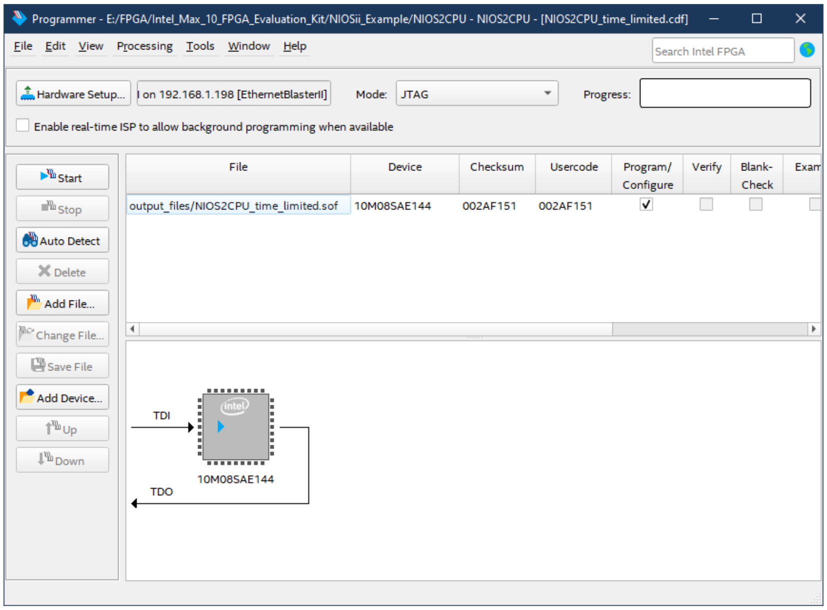
Hardware frequency: Hz

Available hardware items

Hardware	Server	Port
EthernetBlasterII	192.168.1....	EthernetBl...

7. An NIOS2CPU_time_limited.sof file gets created during the Compile Design flow. The file is automatically filled in. There is only one FPGA on the board and in the JTAG chain so the file already has the Program/Configure checkbox checked. Click the Start button to program the board. The process takes a few seconds and shows that the task was completed successfully.

Note: The reason for the “time_limited” in the name of the .sof file is that we chose a Nios II/f, which requires a license. The design must be connected to the JTAG cable or the system will shut off after an hour.



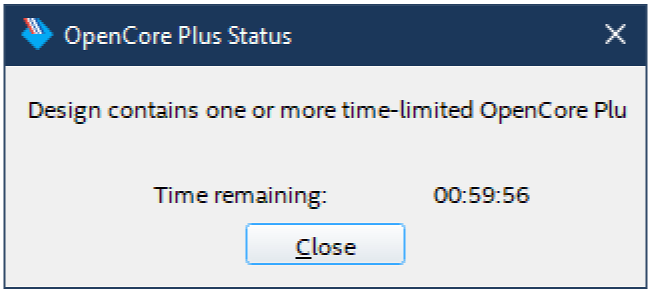
Programmer - E:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOSii_Example/NIOS2CPU - NIOS2CPU - [NIOS2CPU_time_limited.cdf]

Hardware Setup... | I on 192.168.1.198 [EthernetBlasterII] | Mode: JTAG | Progress:

□ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Exam
output_files/NIOS2CPU_time_limited.sof	10M08SAE144	002AF151	002AF151	☑	□	□	□

A dialog will appear that the design is time limited to one hour. The design can always be reloaded if the timeout occurs.



OpenCore Plus Status

Design contains one or more time-limited OpenCore Plu

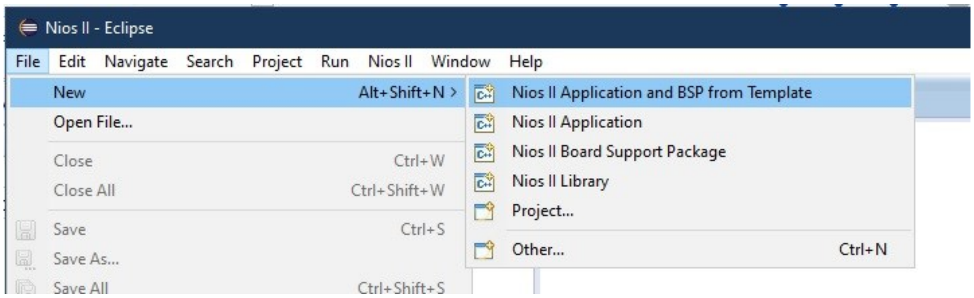
Time remaining: 00:59:56

Important: This dialog acts as a tether to the time-limited IP. You must leave this dialog running while you are running applications.

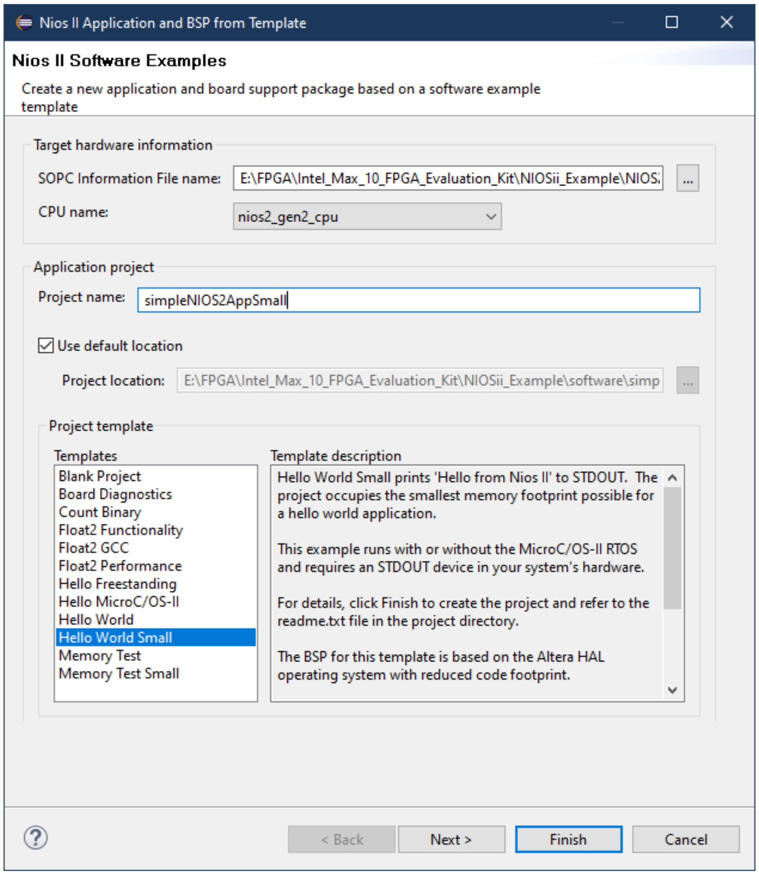
1.1.5 Write and Deploy an Application in Eclipse

Now, we are ready to create an application to run on the Nios II processor.

1. In Quartus Prime, from the menu, select Tools->NIOS II Software Build Tools for Eclipse.
2. Eclipse will open and ask for the root workspace directory. Set the workspace folder to something like \Documents\FPGA\Apps, and hit ok. It doesn’t matter the location of the workspace, since the actual applications for the project will exist within the \NIOSii_Example\software folder.
3. In Eclipse, from the menu, select File->New-> Nios II Application and BSP from Template.



4. The first step is to open the SOPC file that was generated for the hardware design. Click on the three dots button.
5. Navigate to the \NIOSii_Example folder and open the NIOS2.sopcinfo file. The CPU name will reflect the name we gave the CPU in Platform Builder.
6. Enter the project name: simpleNIOS2AppSmall.
7. In the Project Template, select Hello World Small. The Hello World Small template creates a small footprint application that uses the newlib C library and other footprint-reduced features. This is the ideal template to use when memory is tight. The calls made in the project are to the HAL-exposed APIs.
8. Click Finish.


Nios II Application and BSP from Template

Nios II Software Examples

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOSii_Example\NIOS2

CPU name: nios2_gen2_cpu

Application project

Project name: simpleNIOS2AppSmall

☑ Use default location

Project location: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOSii_Example\software\simp

Project template

Templates: Blank Project, Board Diagnostics, Count Binary, Float2 Functionality, Float2 GCC, Float2 Performance, Hello Freestanding, Hello MicroC/OS-II, Hello World, Hello World Small (selected), Memory Test, Memory Test Small

Template description:

Hello World Small prints 'Hello from Nios II' to STDOUT. The project occupies the smallest memory footprint possible for a hello world application.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

The BSP for this template is based on the Altera HAL operating system with reduced code footprint.

Two projects will be generated. The simpleNIOS2AppSmall_bsp is generated to give you the HAL drivers and API based on the hardware design. The simpleNIOS2AppSmall is the application that will run on the hardware.

9. Expand the simpleNIOS2AppSmall project.

10. Open the hello_world_small.c file. The code will look similar to a C application, but instead of a printf method, an alt_putstr method is called.

```
#include "sys/alt_stdio.h"

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

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

  return 0;
}
```

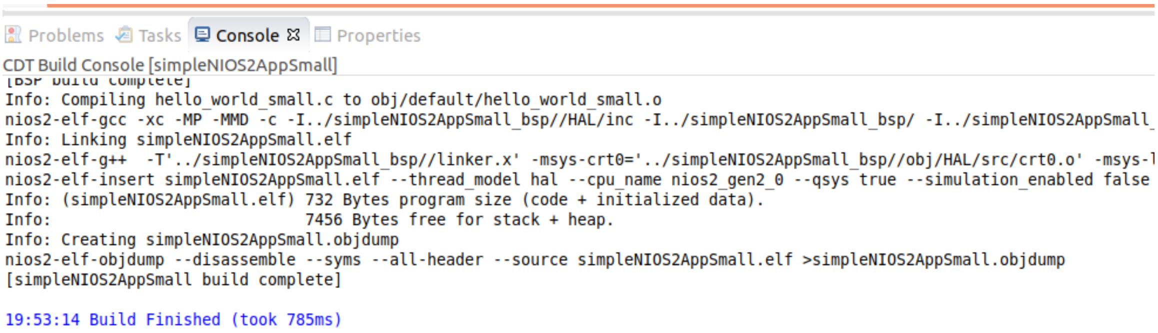
The #include “sys/alt_stdio.h” is and include-file from the BSP project that contains the prototypes for standard IO calls.

```
#include <stdarg.h>


#ifdef __cplusplus
extern “C” {
#endif

int alt_getchar();
int alt_putchar(int c);
int alt_putstr(const char* str);
void alt_printf(const char *fmt, ...);
#ifdef ALT_SEMIHOSTING
int alt_putcharbuf(int c);
int alt_putstrbuf(const char* str);
int alt_putbufflush();
#endif
#ifdef __cplusplus
}
#endif
```

11. Add a newline character to output an additional string to the standard output.

```
#include "sys/alt_stdio.h"

int main()
{
  alt_putstr("Hello from Nios II!\n");
   alt_putstr("Programming is fun!");
  /* Event loop never exits. */
  while (1);

  return 0;
}
```

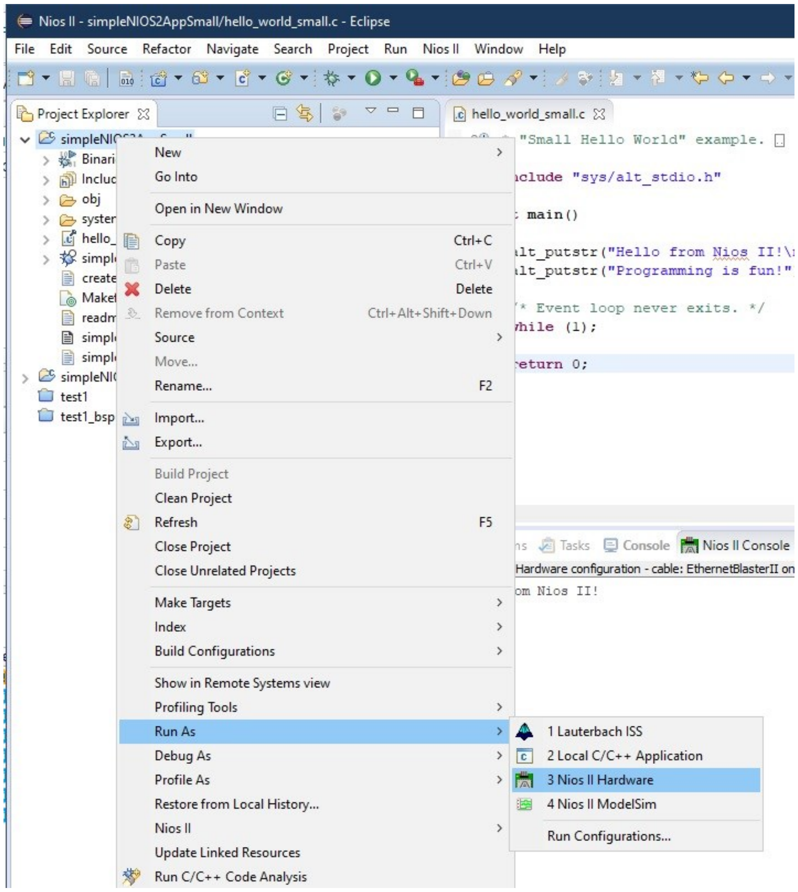
12. Right-click on simpleNIOS2AppSmall project and select Build Project from the context menu. The build should complete successfully with a simpleNIOS2AppSmall.elf file being created.



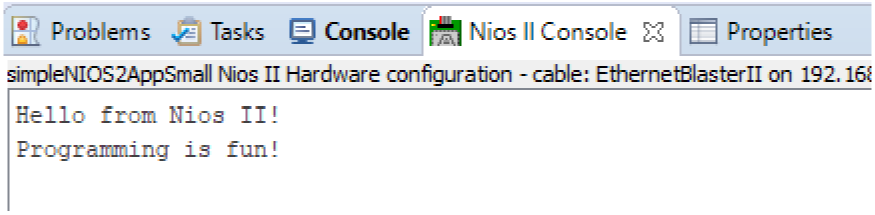
CDT Build Console [simpleNIOS2AppSmall]

```
[BSP build complete]
Info: Compiling hello_world_small.c to obj/default/hello_world_small.o
nios2-elf-gcc -xc -MP -MMD -c -I../simpleNIOS2AppSmall_bsp//HAL/inc -I../simpleNIOS2AppSmall_bsp/ -I../simpleNIOS2AppSmall_
Info: Linking simpleNIOS2AppSmall.elf
nios2-elf-g++  -T'../simpleNIOS2AppSmall_bsp//linker.x' -msys-crt0='../simpleNIOS2AppSmall_bsp//obj/HAL/src/crt0.o' -msys-l
nios2-elf-insert simpleNIOS2AppSmall.elf --thread_model hal --cpu_name nios2_gen2_0 --qsys true --simulation_enabled false
Info: (simpleNIOS2AppSmall.elf) 732 Bytes program size (code + initialized data).
Info:                           7456 Bytes free for stack + heap.
Info: Creating simpleNIOS2AppSmall.objdump
nios2-elf-objdump --disassemble --syms --all-header --source simpleNIOS2AppSmall.elf >simpleNIOS2AppSmall.objdump
[simpleNIOS2AppSmall build complete]

19:53:14 Build Finished (took 785ms)
```

13. With the JTAG cable connected and the hard design already programmed into the chip, right-click on simpleNIOS2AppSmall project and Run AS->NIOS II Hardware.



The application will download and run. Since the JTAG is the standard output for the system, the Nios II Console will show the output from the JTAG UART.



simpleNIOS2AppSmall Nios II Hardware configuration - cable: EthernetBlasterII on 192.168

```
Hello from Nios II!
Programming is fun!
```

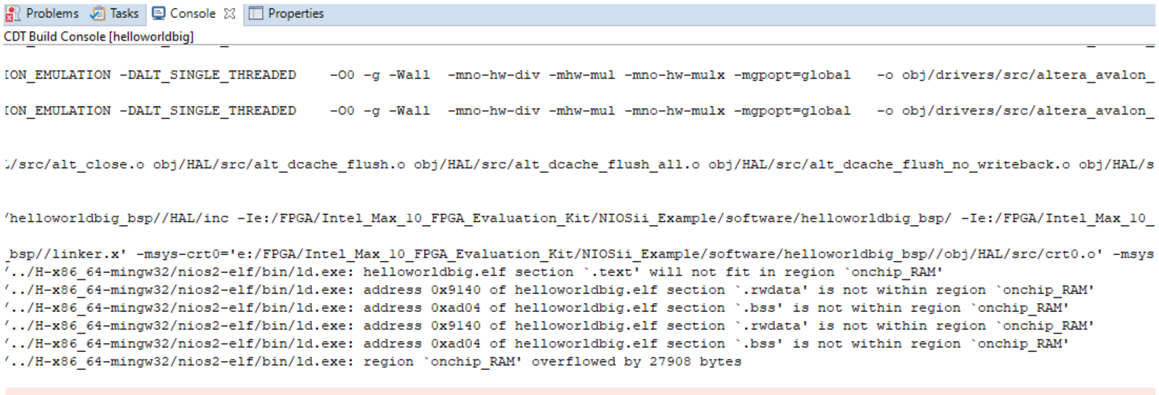
Warning: The tools make it easy to download and run applications but multiple application download attempts can cause the tools to crash with a java runtime pop-up error.

1.1.6 What if Hello World Template Is Used Instead?

If the Hello World template was chosen instead of the Hello World small, the code for the application would look like the regular C hello world application that we all know.

```
#include <stdio.h>

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

  return 0;
}
```

When building the project, the compilation will result in an error that you have overflowed the RAM allocated for the hardware. The small template is better to use when memory is this limited.



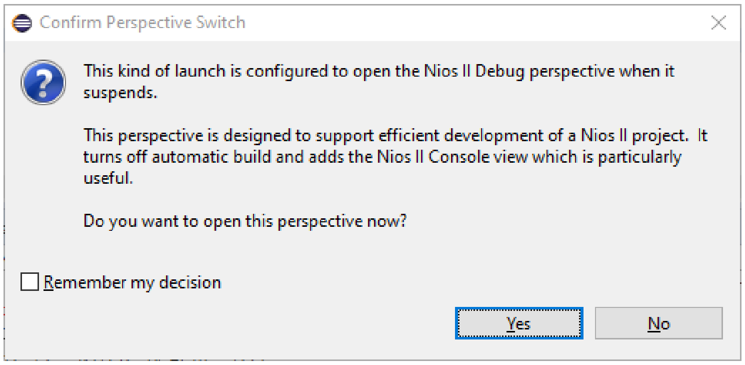
```
ION_EMULATION -DALT_SINGLE_THREADED    -O0 -g -Wall  -mno-hw-div -mhw-mul -mno-hw-mulx -mgpopt=global   -o obj/drivers/src/altera_avalon_

ION_EMULATION -DALT_SINGLE_THREADED    -O0 -g -Wall  -mno-hw-div -mhw-mul -mno-hw-mulx -mgpopt=global   -o obj/drivers/src/altera_avalon_

./src/alt_close.o obj/HAL/src/alt_dcache_flush.o obj/HAL/src/alt_dcache_flush_all.o obj/HAL/src/alt_dcache_flush_no_writeback.o obj/HAL/s

'helloworldbig_bsp//HAL/inc -Ie:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOSii_Example/software/helloworldbig_bsp/ -Ie:/FPGA/Intel_Max_10_

_bsp//linker.x' -msys-crt0='e:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOSii_Example/software/helloworldbig_bsp//obj/HAL/src/crt0.o' -msys
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: helloworldbig.elf section `.text' will not fit in region `onchip_RAM'
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: address 0x9140 of helloworldbig.elf section `.rwdata' is not within region `onchip_RAM'
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: address 0xad04 of helloworldbig.elf section `.bss' is not within region `onchip_RAM'
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: address 0x9140 of helloworldbig.elf section `.rwdata' is not within region `onchip_RAM'
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: address 0xad04 of helloworldbig.elf section `.bss' is not within region `onchip_RAM'
'../H-x86_64-mingw32/nios2-elf/bin/ld.exe: region `onchip_RAM' overflowed by 27908 bytes
```

Keep in mind the 10M08 is a middle-range of the MAX 10 family. A part in the upper end of the family with more logic elements could handle the bigger libraries like stdio. The Intel® MAX® 10 - 10M08 Evaluation Kit is a simple board that helps you get started with the basics of designing FPGAs.

1.1.7 Debug the SimpleNIOS2AppSmall Application

Debugging an application is simple.

1. In the hello_world_small.c file, set a breakpoint at the first alt_putstr.
2. With the JTAG cable connected and the hard design already programmed to the chip, right-click on simpleNIOS2AppSmall project, Debug As->NIOS II Hardware.
3. In Windows, a dialog will ask to unblock the application from running. Click Yes.
4. Eclipse will pop a message asking to change to debug perspective.



Confirm Perspective Switch

This kind of launch is configured to open the Nios II Debug perspective when it suspends.

This perspective is designed to support efficient development of a Nios II project. It turns off automatic build and adds the Nios II Console view which is particularly useful.

Do you want to open this perspective now?

☐ Remember my decision

5. The application will be downloaded and started running, and the breakpoint will be reached. You can now step through the application.

1.2 Summary: Lots of Options to Choose From

There are many design choices to consider. For example, the Nios II/e might have been a better solution for this example than the Nios II/f. A little experimentation is required to get a feel for the features. Hopefully, this walk-through of a Nios II design from the start to running a C application on the hardware was beneficial.

1.3 References

The following references were used for this article:

1.3.1 Intel FPGA Training Videos:

The Nios® II Processor: Introduction to Developing Software

Designing with NIOS® II Processor Part 1

Designing with NIOS® II Processor Part 2

1.3.2 Online Videos

My First Nios II Tutorial (1) by terasIC

My First Nios II Tutorial (2) by terasIC

Qsys Tutorial 1 - Adder using NIOS II processor

1.3.3 Other Resources

Nios II Software Developer Handbook - 3.1. Installing Eclipse IDE into Nios® II EDS (intel.com)

Intel® MAX® 10-10M08 Evaluation Kit schematic file.
Altera_10M08S_E144_eval_schematic_REV_1_0.pdf.