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

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

November 2022

As one digs into all the features of the Intel MAX 10, the Analog to Digital Converter, ADC, provides a nice multi-channel solution for audio applications. Add the ADC to a Nios II processor design and you can write applications that can process analog data to perform other functions. The only catch is the lack of examples on how to use the Nios II HAL API to access the ADC. Internet search results provide limited examples and those examples shared are based on older versions of the development tools and software implementations. This paper's hands-on exercises look to provide solutions based on the latest Quartus release.

The Intel Max 10 10M08 Evaluation Kit will be used as the target for this project. The evaluation kit is a bare-bones platform that provides the basics for learning FPGA development. The smaller number of logic elements and RAM blocks in the Intel Max 10 means that a design is going to be tight and the small C library has to be used for application development. The platform makes it ideal for learning how to design and program with limited resources.

The design will take advantage of what the board provides. The system will read the voltage from the 10KΩ trimmer pot and turn on red LEDs based on the voltage level. As the trimmer pot is adjusted from 0v to 3.3V the LEDs will be turned on or off based on the voltage level. A signal generator can be used if the 10KΩ trimmer pot is not populated on the board. The JTAG will act as a UART for standard output so you can see the ADC values. Two different applications will be developed. One will be a single shot reading of the ADC, and the other will be a continuous-reading, interrupt-driven application.

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.

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 is required. The schematic PDF file can be downloaded from the Intel FPGA website.
  - A populated 10KΩTrimmer pot for R94 on the schematic, part number 3362P-1-103TLF.
  - Alternative: Signal generator or other small analog signal source.
- 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 onboard.
- *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.

1.1 Nios II ADC Project

The custom MCU will comprise the following IP blocks:

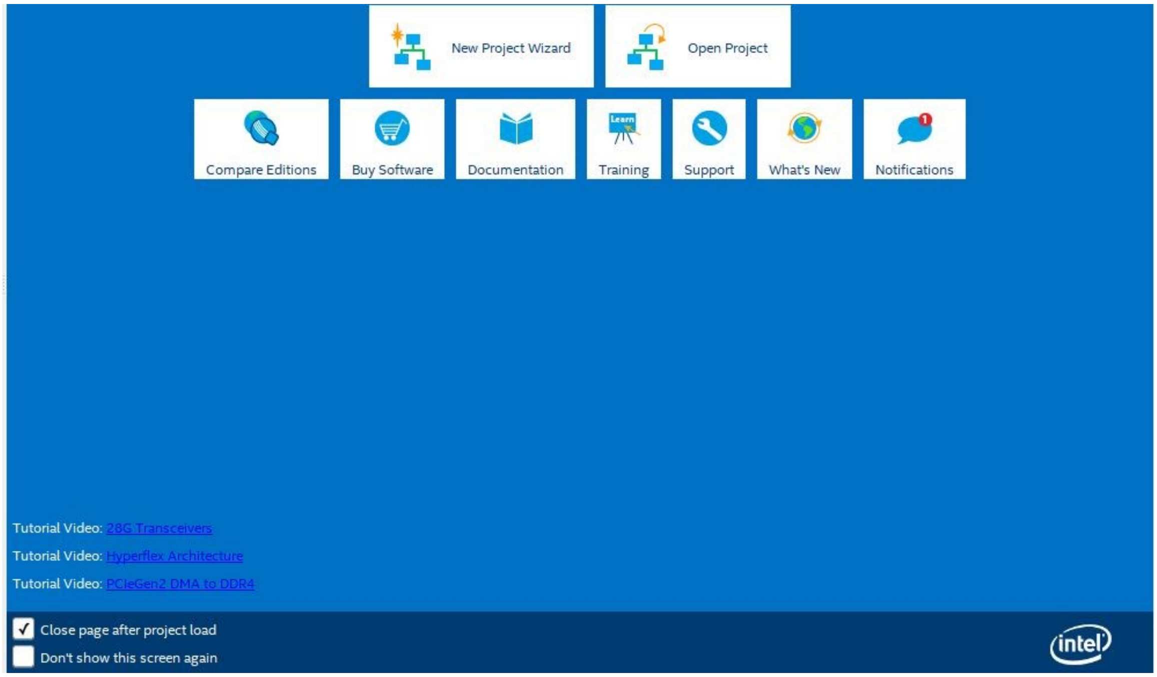
- Nios II processor

- Onchip RAM
- ADC
- Phase Lock Loop (PLL)
- Timer
- Sys ID
- JTAG UART

1.1.1 Create the Project

The first step is to create the design project.

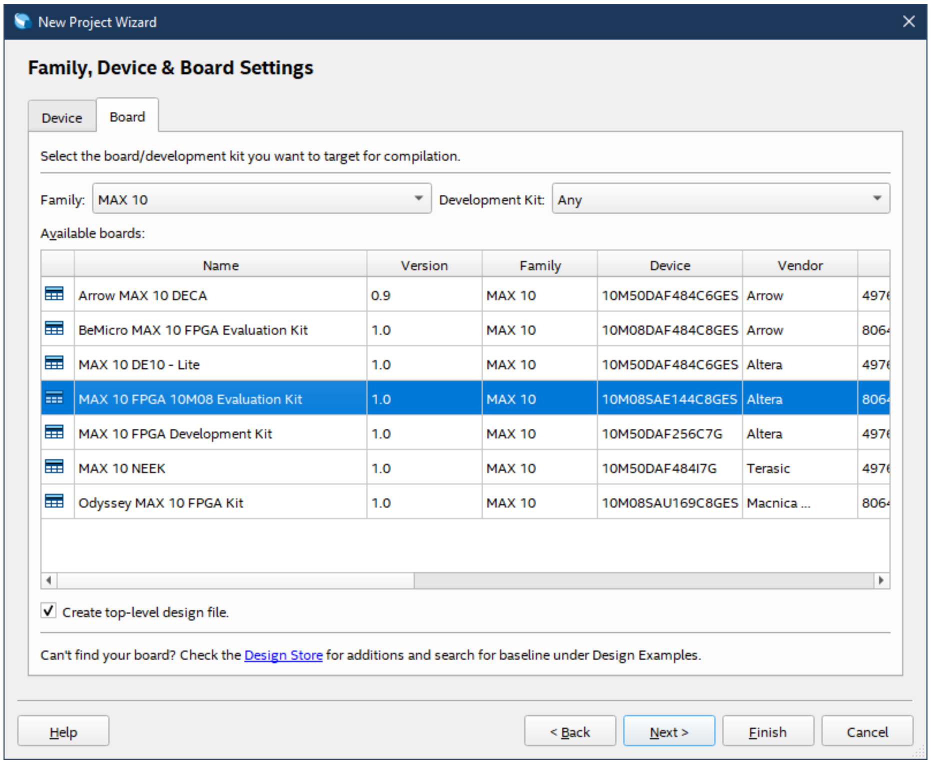
1. Open Quartus.
2. Click on the New Project Wizard.


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 \NIOS2_ADC (Do not use the Quartus installation directory) and name the project: “NIOS2adc”. 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 and 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

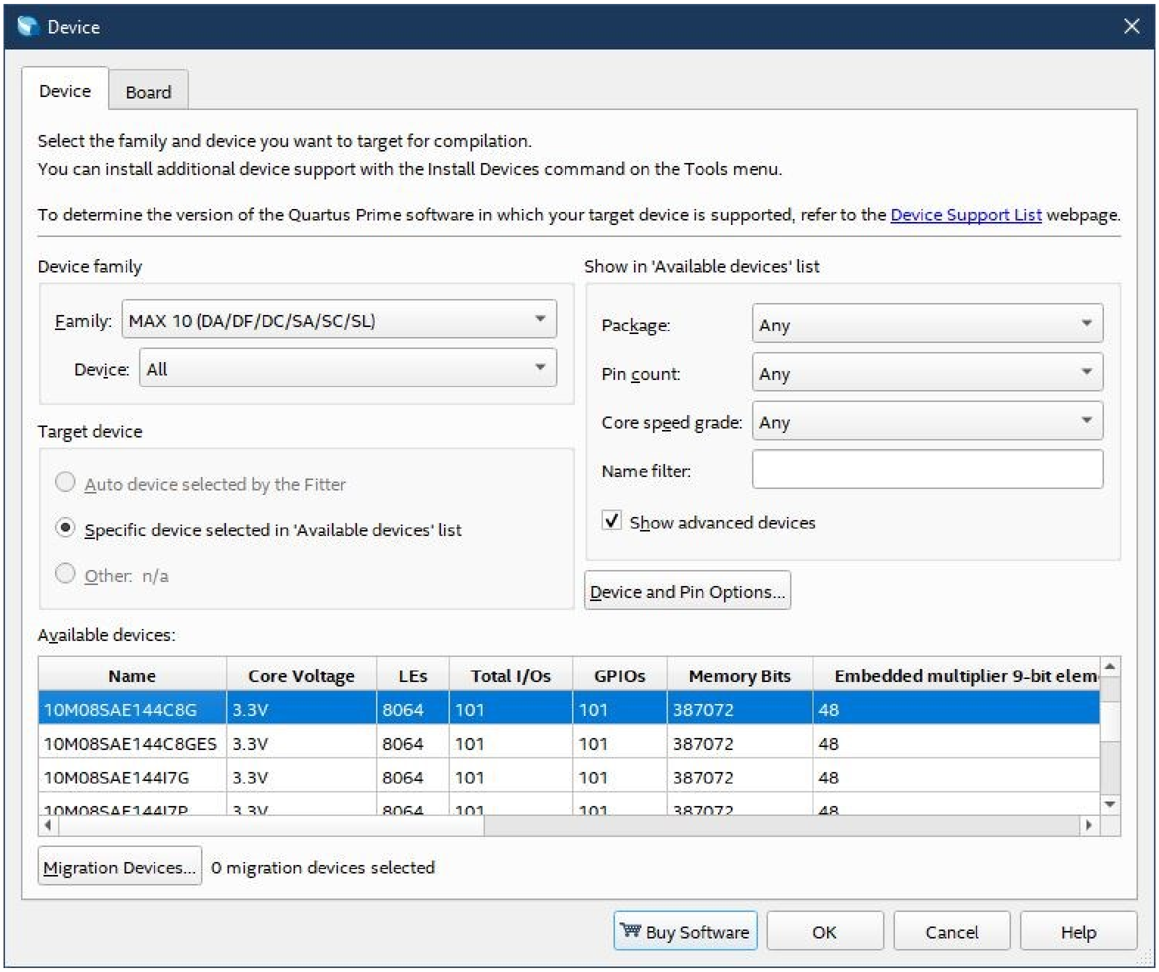
☑ Create top-level design file.

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. Depending on the hardware that you use, 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.



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

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

0 migration devices selected

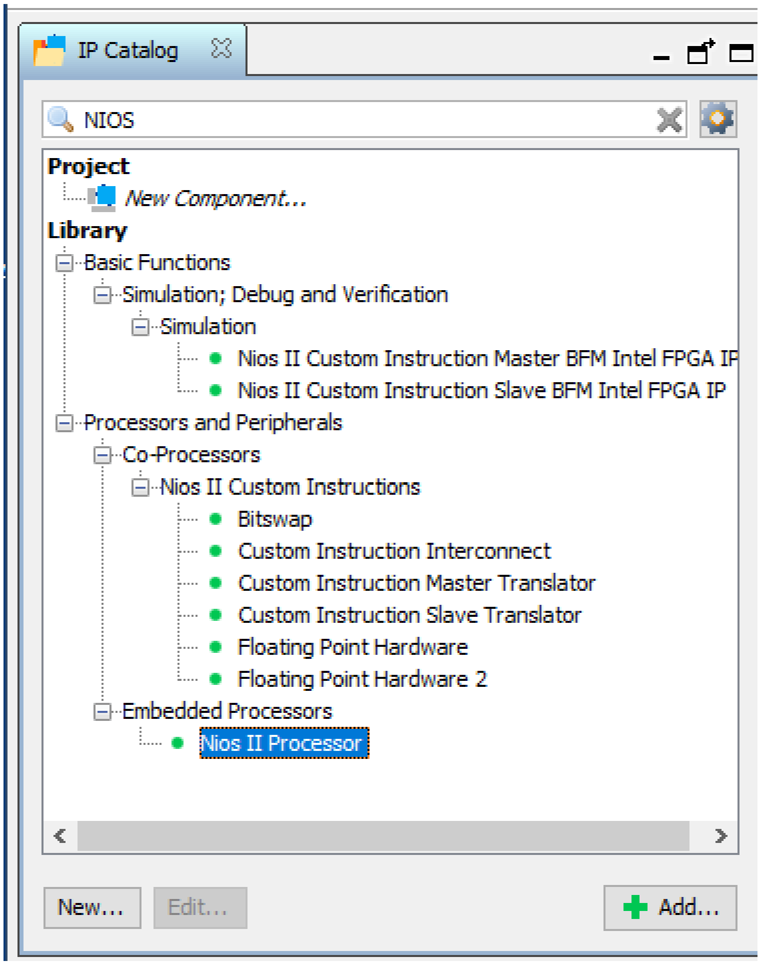
1.1.2 Create the Design in Platform Designer

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

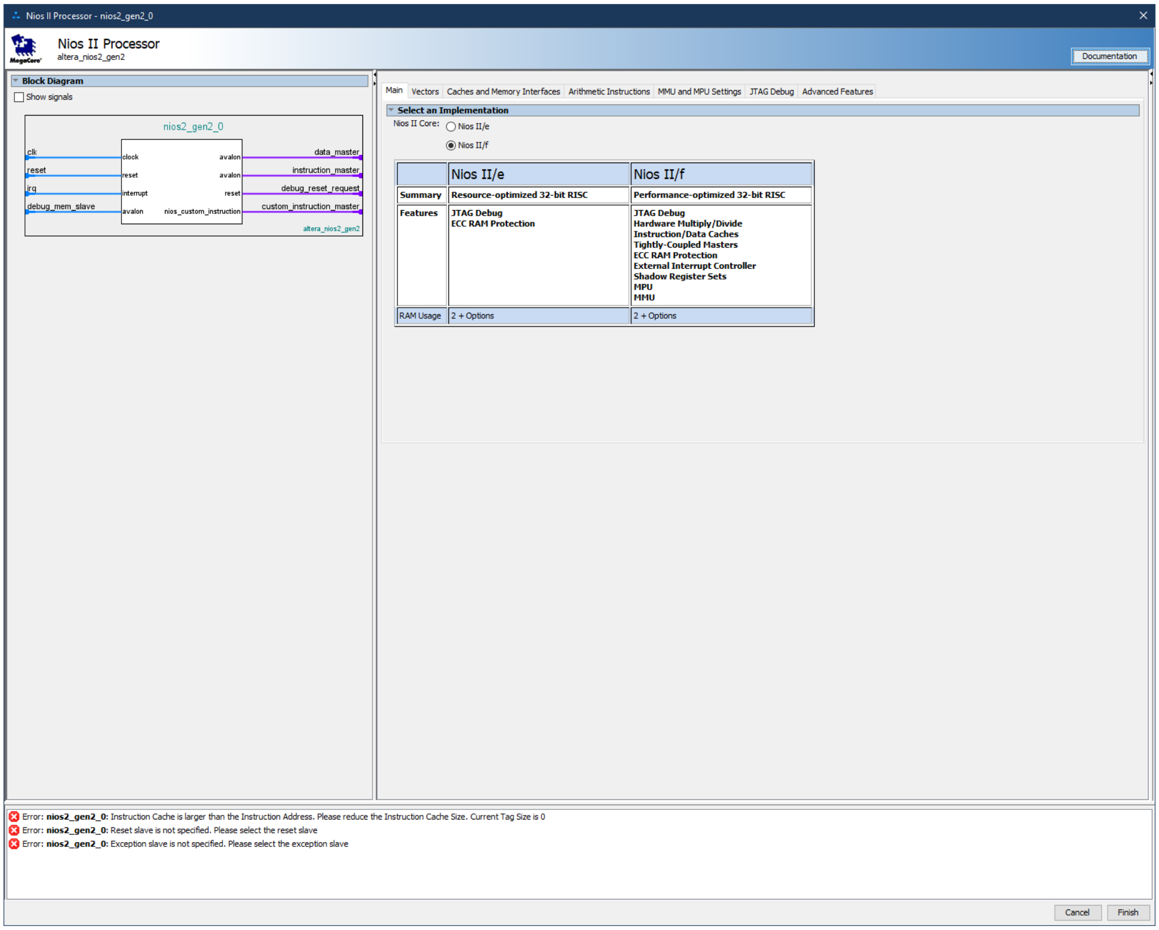
1. From the menu, select Tools->Platform Designer, or the Platform Designer icon from the toolbar.

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.


IP Catalog
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
New...
Edit...
Add...

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



Select an Implementation

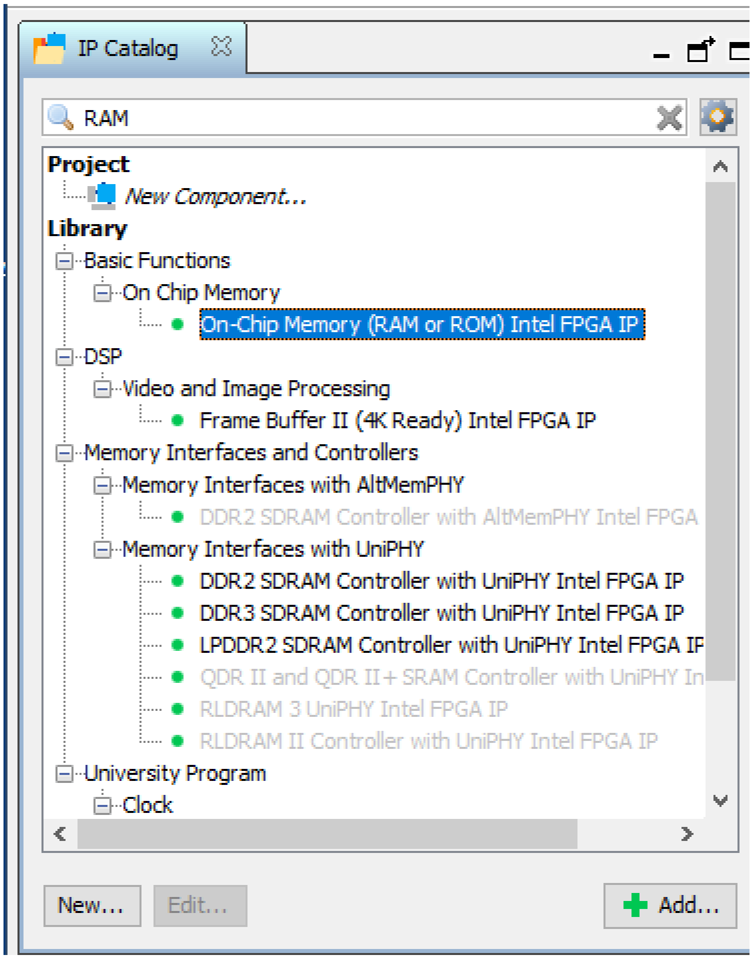
Nios II Core: ○ Nios II/e

● 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
```

1. The processor will be added to the design. Right-click on the name nios2_gen2_cpu, and rename it to nios2.
2. Now let's add the RAM IP block. In the IP Catalog enter RAM in the search box.
3. Double-click on On-chip Memory (RAM or ROM) in the Intel FPGA IP.



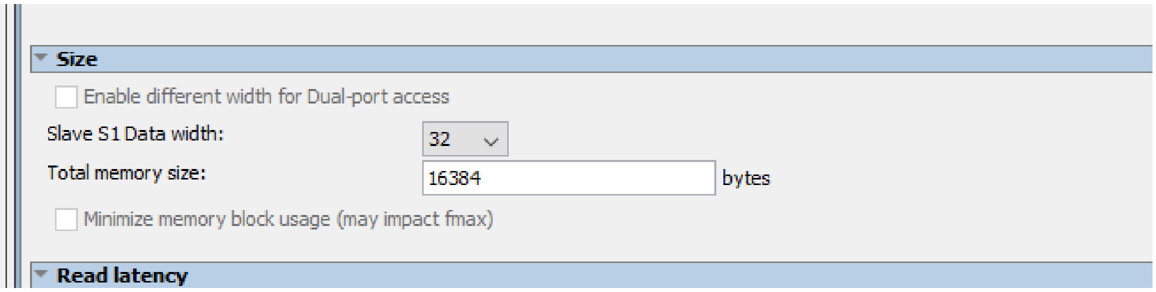
IP Catalog

RAM

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

4. The configuration page will appear. Change the Total memory size to 16384. We need more memory to run this application.



Size

□ Enable different width for Dual-port access

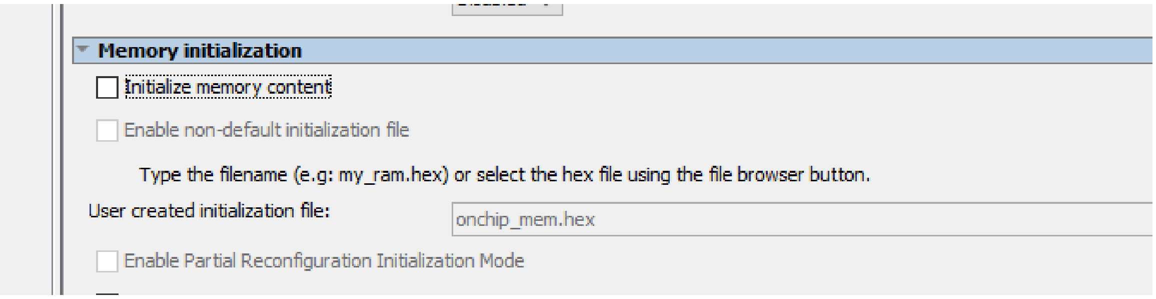
Slave S1 Data width: 32

Total memory size: 16384 bytes

□ Minimize memory block usage (may impact fmax)

Read latency

5. Uncheck the box for “Initialize memory content” and click Finish.



Memory initialization

□ Initialize memory content

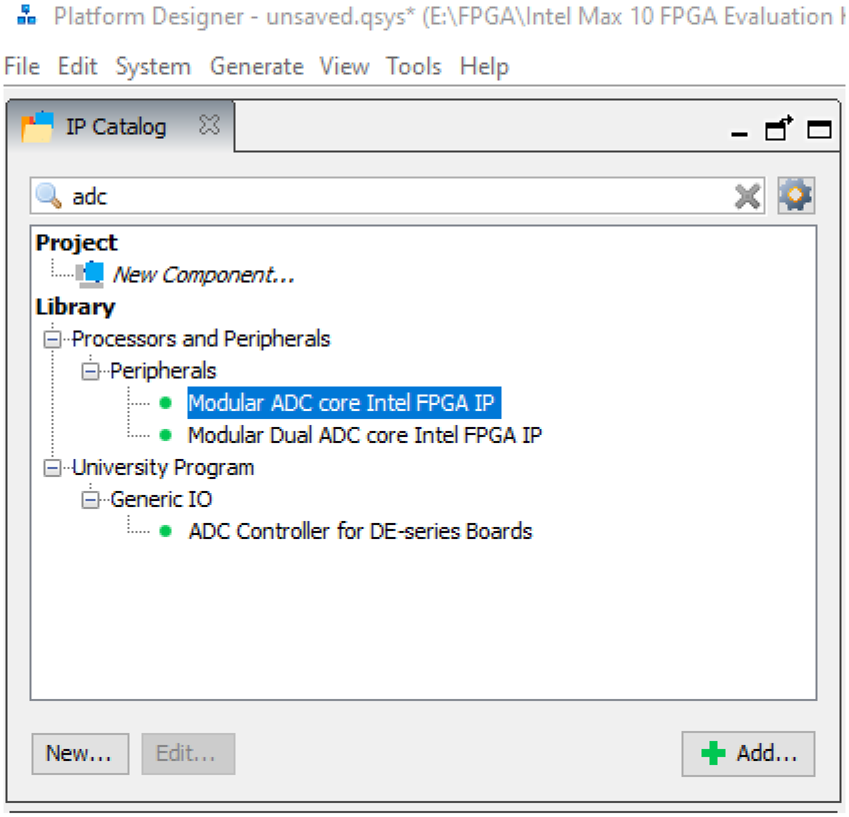
□ Enable non-default initialization file

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

□ Enable Partial Reconfiguration Initialization Mode

6. The On-chip Memory (RAM or ROM) in the Intel FPGA IP will be added to the design. Right-click on the name, and rename it to onchip_RAM
7. In the IP Catalog search box, type adc.



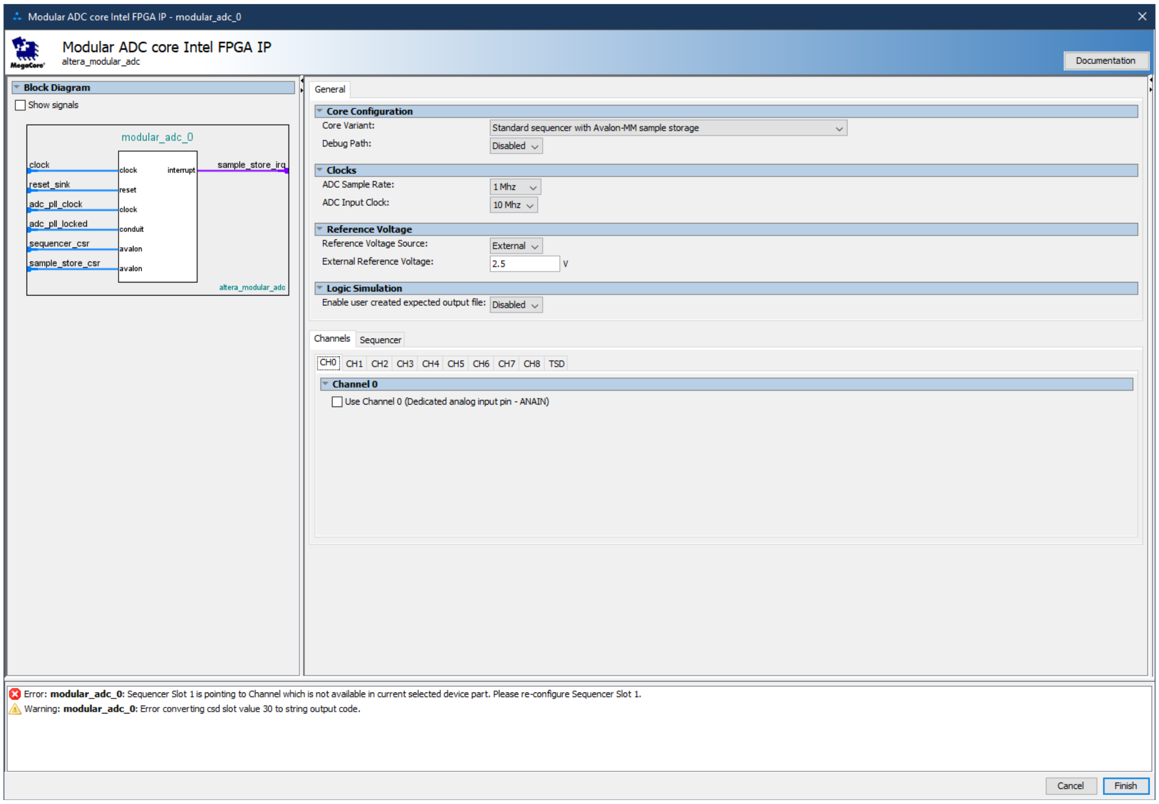
Platform Designer - unsaved.qsys* (E:\FPGA\Intel Max 10 FPGA Evaluation I

adc

Project
- *New Component...*

Library
- Processors and Peripherals
  - Peripherals
    - Modular ADC core Intel FPGA IP
    - Modular Dual ADC core Intel FPGA IP
- University Program
  - Generic IO
    - ADC Controller for DE-series Boards

8. Expanding the branches reveals the available IP. Double-click on Modular ADC core Intel FPGA IP. This will add the ADC IP to the design and open the Modular ADC core Intel FPGA IP configuration page.



Core Configuration

Core Variant: Standard sequencer with Avalon-MM sample storage

Debug Path: Disabled

Clocks

ADC Sample Rate: 1 Mhz

ADC Input Clock: 10 Mhz

Reference Voltage

Reference Voltage Source: External

External Reference Voltage: 2.5 V

Logic Simulation

Enable user created expected output file: Disabled

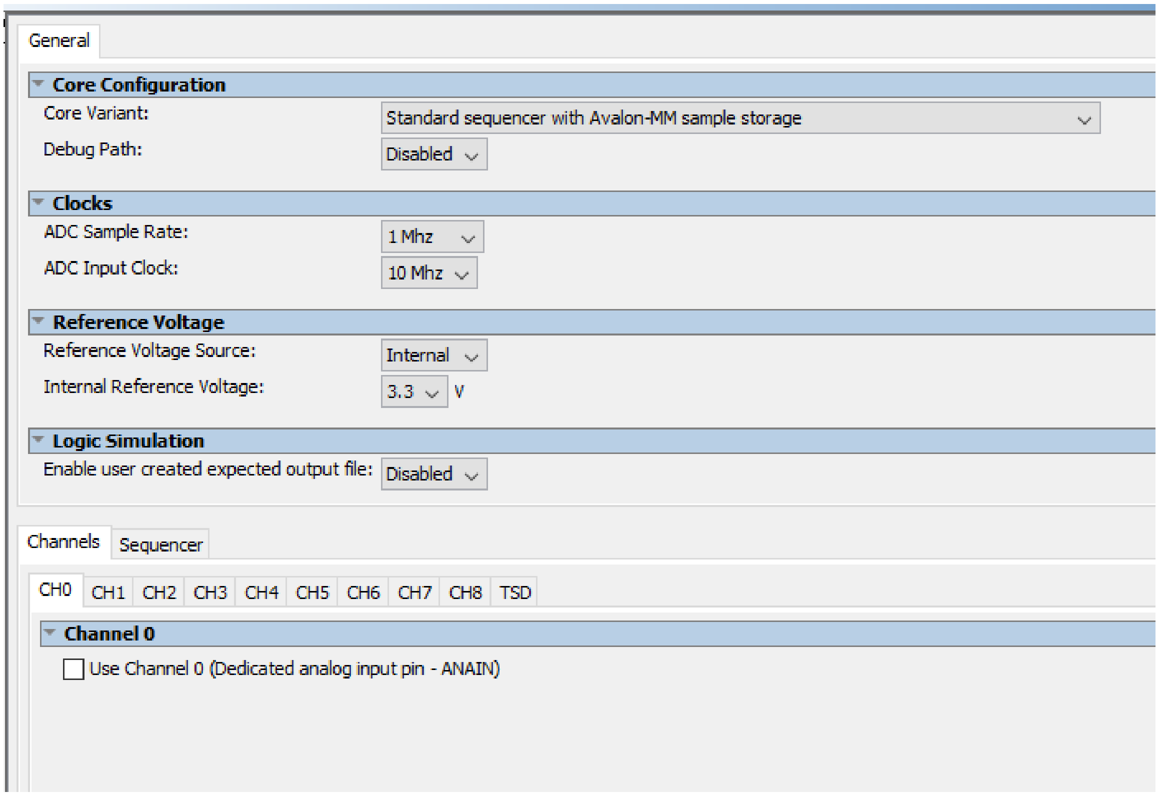
Channel 0

□ Use Channel 0 (Dedicated analog input pin - ANAIN)

```
Error: modular_adc_0: Sequencer Slot 1 is pointing to Channel which is not available in current selected device part. Please re-configure Sequencer Slot 1.
Warning: modular_adc_0: Error converting csd slot value 30 to string output code.
```

9. In the General tab, set the following:
   a. Core Variant: Standard sequencer with Avalon-MM sample storage.
   b. Debug Path: Disabled.
   c. ADC Sample Rate: 1 MHz.
   d. ASC Input Clock: 10 MHz.
   e. Reference Voltage Source: External.
   f. Internal reference Voltage: 3.3V.
   g. Enable user-created expect output file: Disabled.

The ADC IP block supports several implementation variants. The one chosen will use the MAX 10's internal RAM to save the data. The evaluation kit has a 2.5 V reference voltage for the ADC, but the 10K trimmer pot can supply 3.3 to the channel so we will use the internal 3.3V.



General

Core Configuration

Core Variant: Standard sequencer with Avalon-MM sample storage

Debug Path: Disabled

Clocks

ADC Sample Rate: 1 Mhz

ADC Input Clock: 10 Mhz

Reference Voltage

Reference Voltage Source: Internal

Internal Reference Voltage: 3.3 V

Logic Simulation

Enable user created expected output file: Disabled

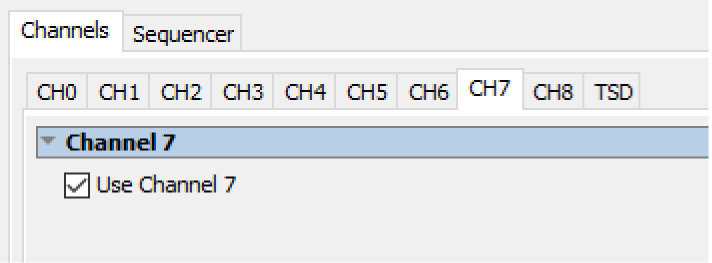
Channels | Sequencer

CH0 | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 | CH8 | TSD

Channel 0

☐ Use Channel 0 (Dedicated analog input pin - ANAIN)

10. In the Channels tab, click on CH7, and check the “Use Channel 7” box



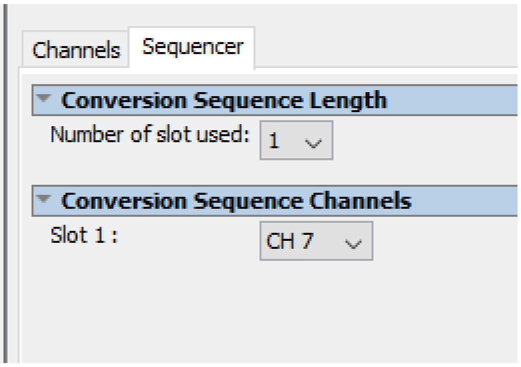
Channels | Sequencer

CH0 | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 | CH8 | TSD

Channel 7

☑ Use Channel 7

11. Click on the Sequencer tab.
12. Set the number of slots used to 1.
13. Set Slot 1: to CH7



Channels | Sequencer

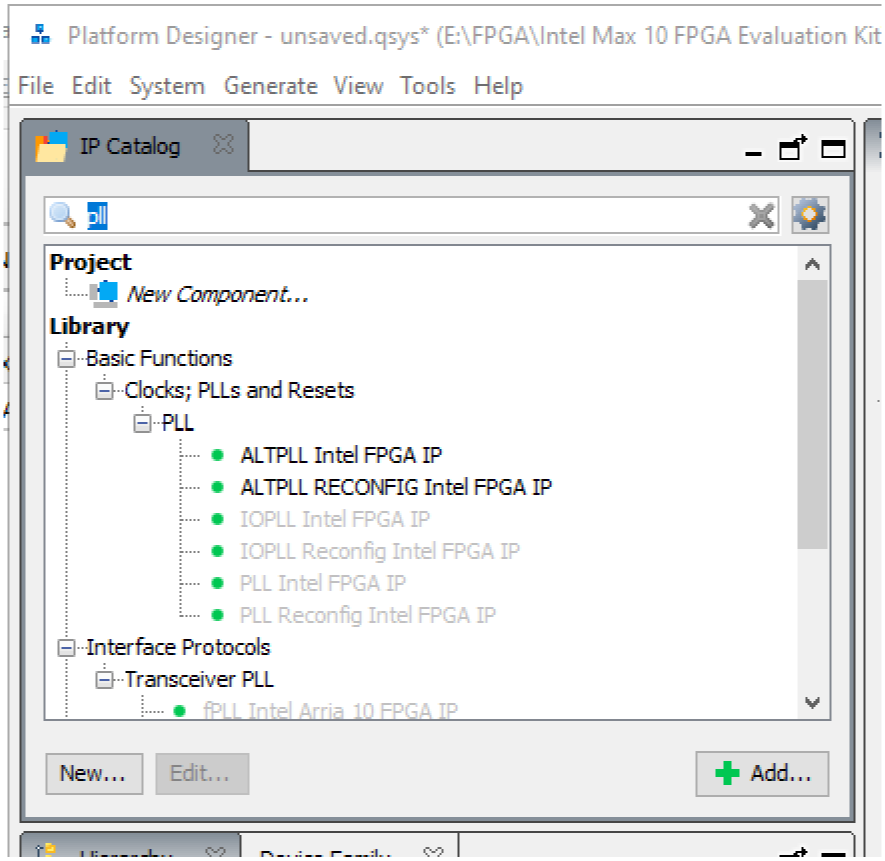
Conversion Sequence Length

Number of slot used: 1

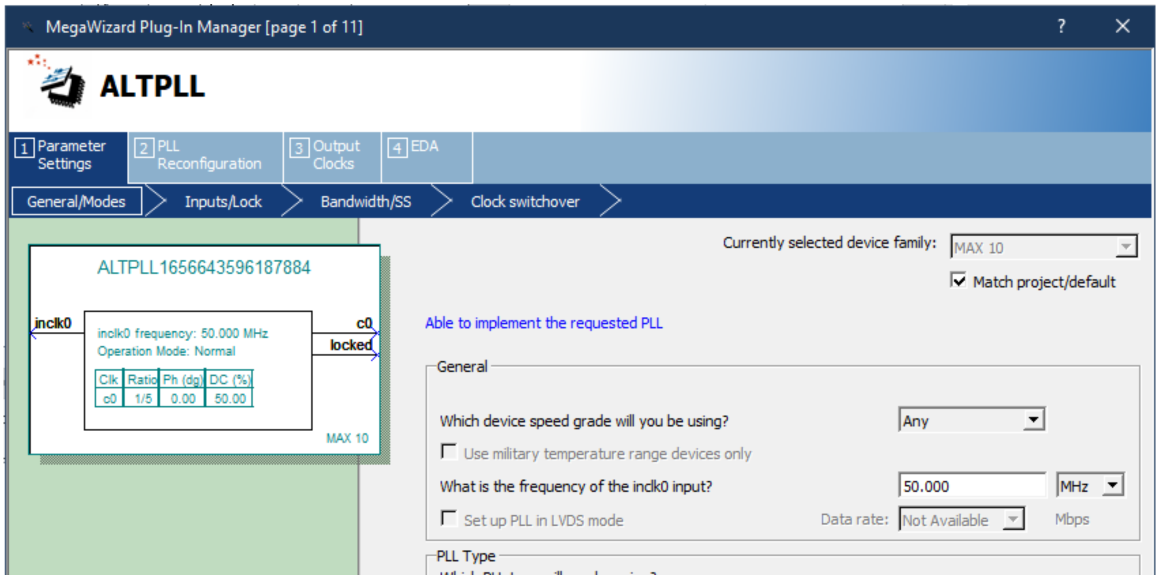
Conversion Sequence Channels

Slot 1 : CH 7

14. Click Finish.
15. The ADC will be added to the design. In the System Contents, you will see the ADC has been added to the list of devices to be interconnected. Right-click on the name and rename the device to ADC0.
16. Now we need to add the PPL. In the IP Catalog, type pll in the search.
17. A number of different PLLs appear in the branches, but only a few are available. Double-click on the ALTPLL Intel FPGA IP to add it to the design.


Platform Designer - unsaved.qsys* (E:\FPGA\Intel Max 10 FPGA Evaluation Kit
File Edit System Generate View Tools Help
IP Catalog
pll
Project
New Component...
Library
Basic Functions
Clocks; PLLs and Resets
PLL
ALTPLL Intel FPGA IP
ALTPLL RECONFIG Intel FPGA IP
IOPLL Intel FPGA IP
IOPLL Reconfig Intel FPGA IP
PLL Intel FPGA IP
PLL Reconfig Intel FPGA IP
Interface Protocols
Transceiver PLL
fPLL Intel Arria 10 FPGA IP
New...
Edit...
Add...

18. The PLL is added, and the ALTPLL Intel FPGA IP configuration page appears. The configuration page has a workflow-like presentation, 1 Parameter Setting contains the general settings for the PLL. For the “What is the frequency of the inclk0 input?” set the value to 50.000 MHz. The evaluation kit has a 50 MHz oscillator.



MegaWizard Plug-In Manager [page 1 of 11]

ALTPLL

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/5	0.00	50.00

Currently selected device family: MAX 10

☑ Match project/default

Able to implement the requested PLL

General

Which device speed grade will you be using? Any

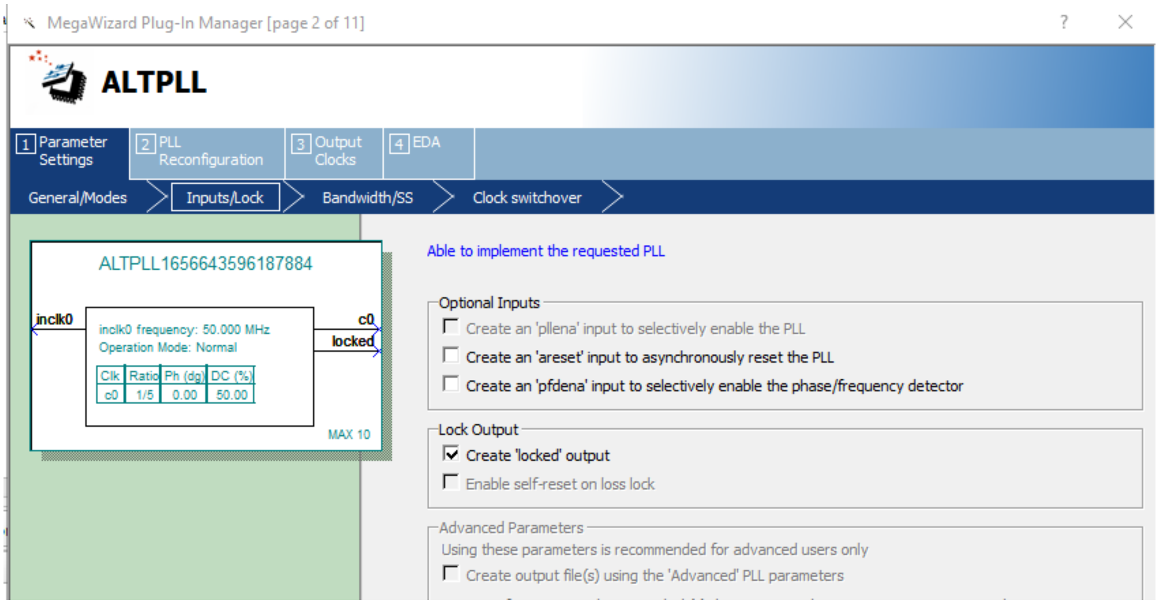
□ Use military temperature range devices only

What is the frequency of the inclk0 input? 50.000 MHz

□ Set up PLL in LVDS mode Data rate: Not Available Mbps

PLL Type

19. Click Next.
20. Uncheck the box next to “Create an ‘areset’ input to asynchronously reset the PLL”. This signal is not needed for this design, and this will remove one warning from the list. Leave Create ‘locked’ output checked.



MegaWizard Plug-In Manager [page 2 of 11]

ALTPLL

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/5	0.00	50.00

Able to implement the requested PLL

Optional Inputs

□ Create an 'pllena' input to selectively enable the PLL

□ Create an 'areset' input to asynchronously reset the PLL

□ Create an 'pfdena' input to selectively enable the phase/frequency detector

Lock Output

☑ Create 'locked' output

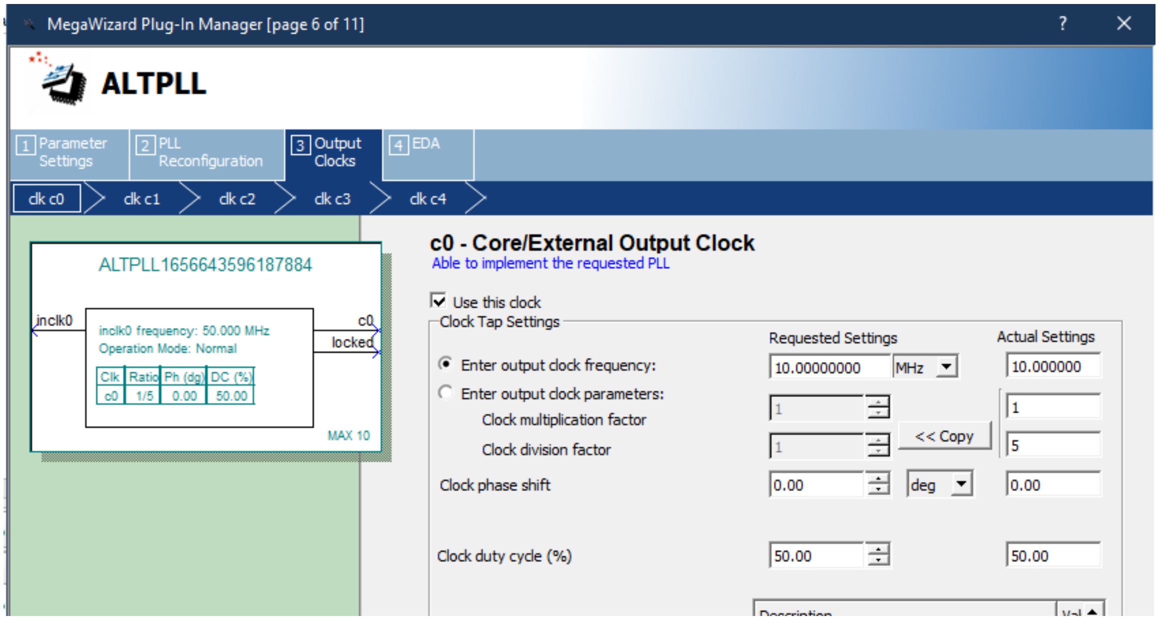
□ Enable self-reset on loss lock

Advanced Parameters

Using these parameters is recommended for advanced users only

□ Create output file(s) using the 'Advanced' PLL parameters

21. Click on 3, Output Clocks tab.
22. There are 5 output clock settings. All we need is clk c0. Under clk c0, click the radio button next to Enter output clock frequency.
23. Set the Requested Settings to 10.00 MHz. This is to match the input clock frequency of the ADC.



MegaWizard Plug-In Manager [page 6 of 11]

ALTPLL

1 Parameter Settings | 2 PLL Reconfiguration | 3 Output Clocks | 4 EDA

clk c0 | clk c1 | clk c2 | clk c3 | clk c4

ALTPLL1656643596187884

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/5	0.00	50.00

c0 - Core/External Output Clock

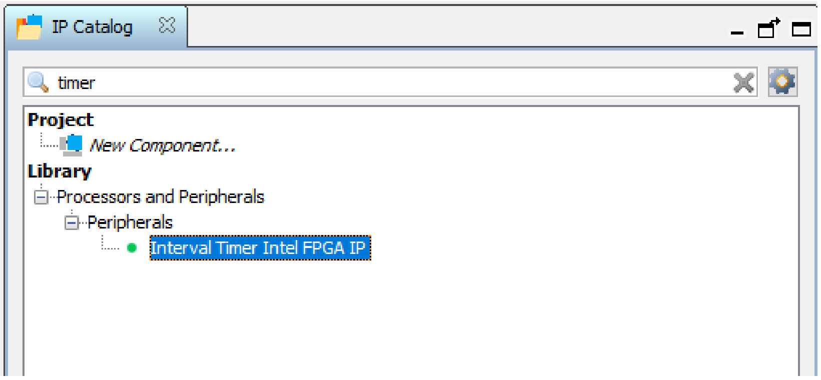
Able to implement the requested PLL

☑ Use this clock

Clock Tap Settings

	Requested Settings		Actual Settings
● Enter output clock frequency:	10.00000000	MHz	10.000000
○ Enter output clock parameters:			
Clock multiplication factor	1	<< Copy	1
Clock division factor	1		5
Clock phase shift	0.00	deg	0.00
Clock duty cycle (%)	50.00		50.00

24. Click Finish.
25. The PLL is added to the design. Rename the PLL as pll_0.
26. In the IP Catalog search, enter timer.
27. Double-click on the Interval Timer Intel FPGA IP.



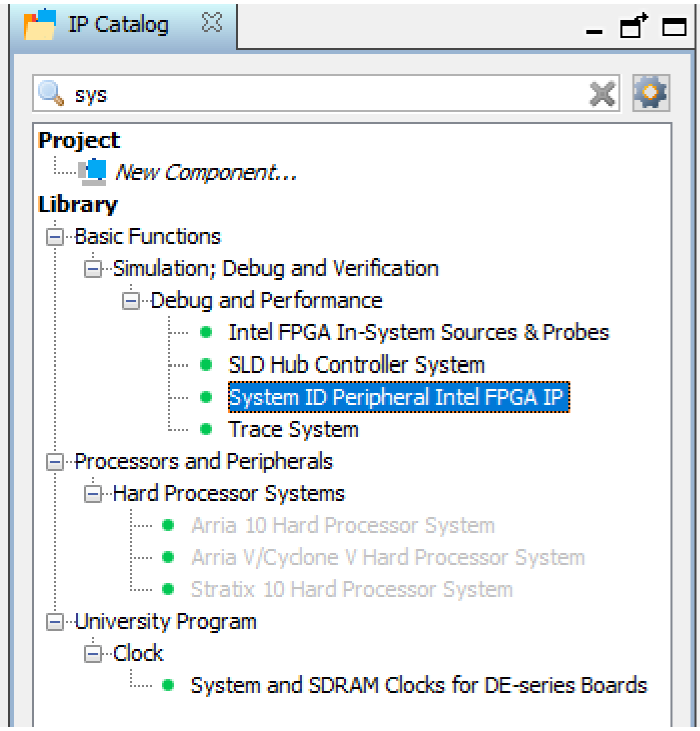
IP Catalog

timer

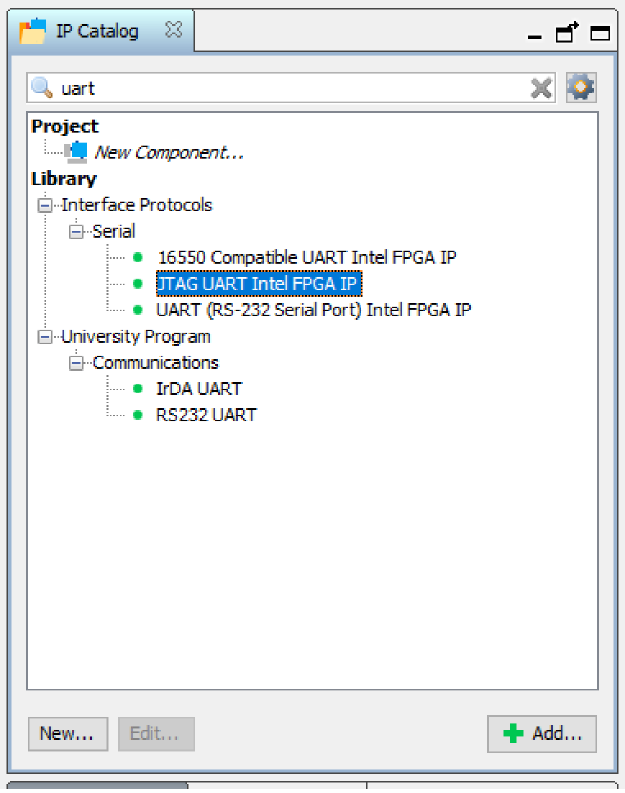
Project
- *New Component...*

Library
- Processors and Peripherals
  - Peripherals
    - Interval Timer Intel FPGA IP

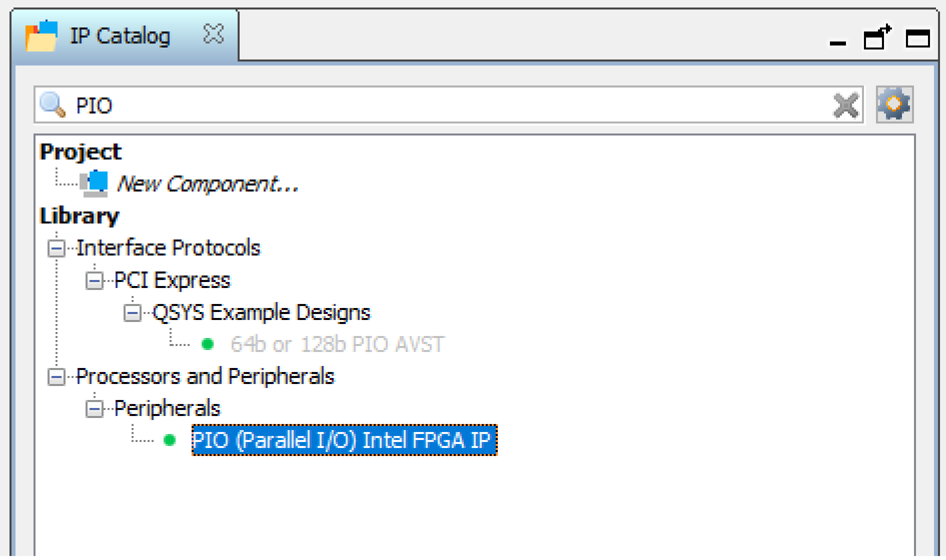
28. Keep the settings as they are and click Finish.
29. In the IP Catalog search, enter system ID.
30. Double-click on the System ID Peripheral Intel FPGA IP.


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

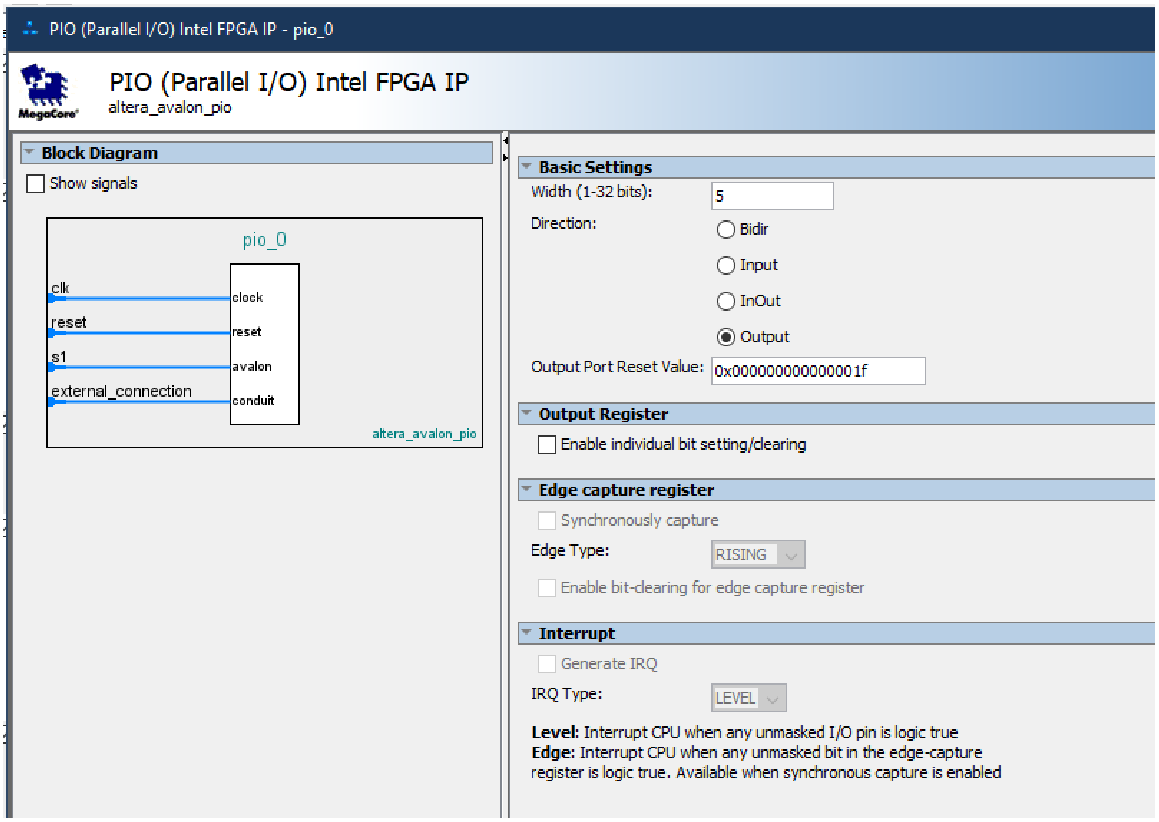
31. A configuration page will appear. There are no changes to be made. Click Finish.
32. In the IP Catalog search, enter uart.
33. Double-click on the JTAG UART Intel FPGA IP.



34. A configuration page will appear. There are no changes to be made. Click Finish.
35. In the IP Catalog, enter pio in the search box.
36. Add the PIO (Parallel I/O) Intel FPGA IP to the design.



37. In the configuration page, set Width to 5, leave the Direction as Output, and set the Output Port Reset Value to 0x1f. Since the LEDs are active low, the value turns all 5 LEDs off on startup.
38. Click Finish.



PIO (Parallel I/O) Intel FPGA IP - pio_0

PIO (Parallel I/O) Intel FPGA IP
altera_avalon_pio

Block Diagram

□ Show signals

Basic Settings

Width (1-32 bits): 5

Direction:
○ Bidir
○ Input
○ InOut
● Output

Output Port Reset Value: 0x000000000000001f

Output Register

□ Enable individual bit setting/clearing

Edge capture register

□ Synchronously capture

Edge Type: RISING

□ Enable bit-clearing for edge capture register

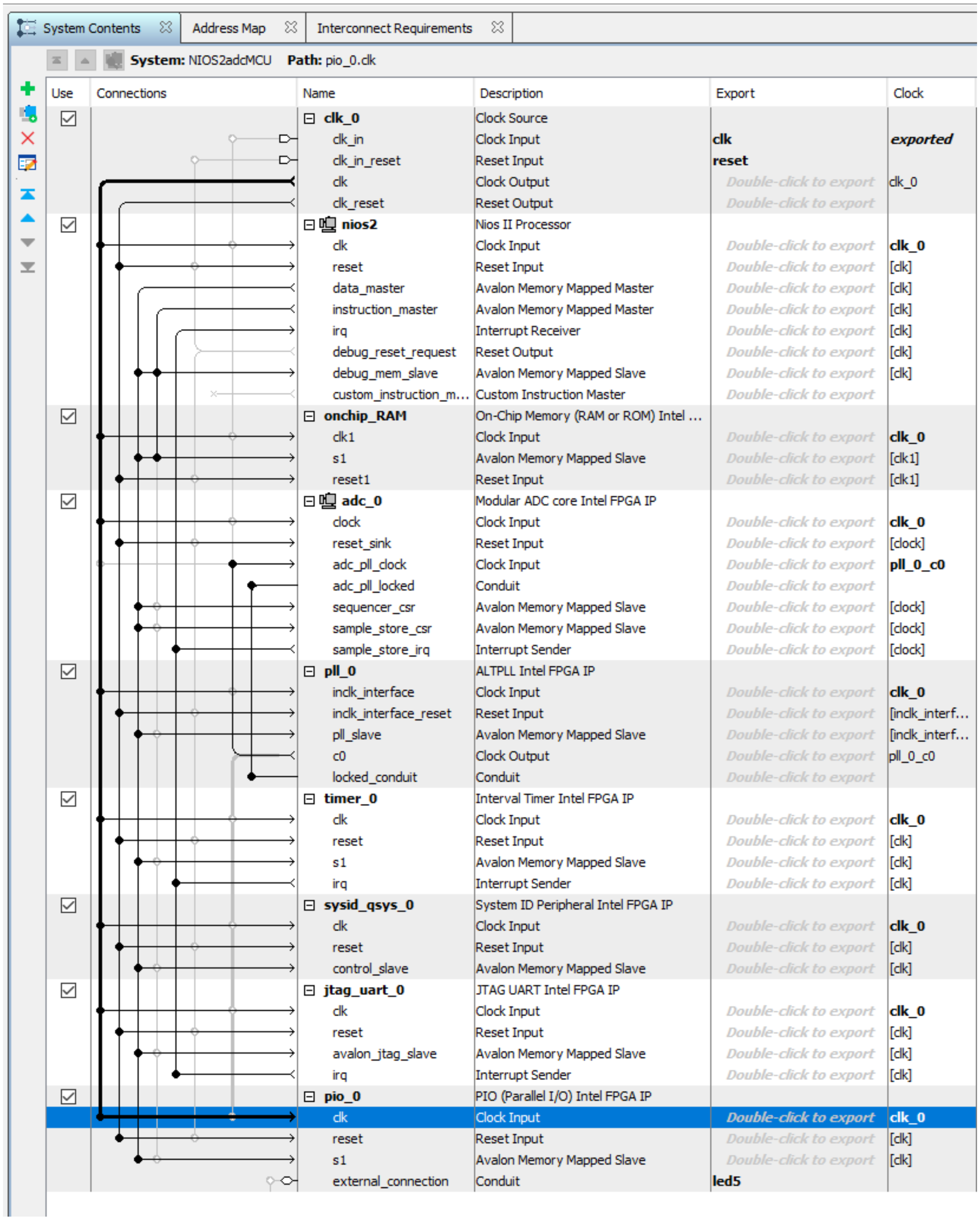
Interrupt

□ Generate IRQ

IRQ Type: LEVEL

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

39. The PIO will be added to the design. Rename the PIO to pio_0.
40. For the PIO exernal_connection, under pio_0, double-click on the "Double-click to export" in the exernal_connection row and Export column and set the value to led5. This will provide a base name for connecting the signals to the PINs on the chip. The connection will be made in PIN Planner.
41. Now we need to wire the IP blocks together. The picture below shows all the wiring connections for the design.

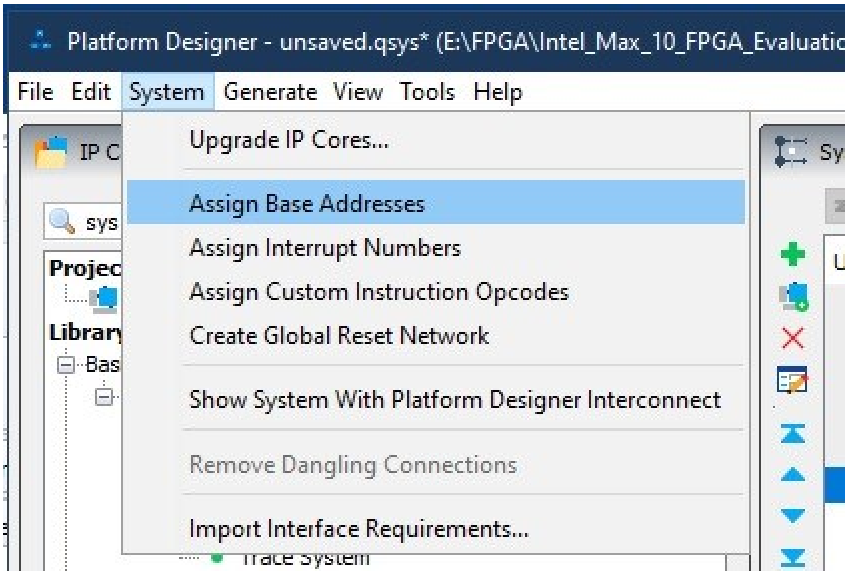


System Contents | Address Map | Interconnect Requirements

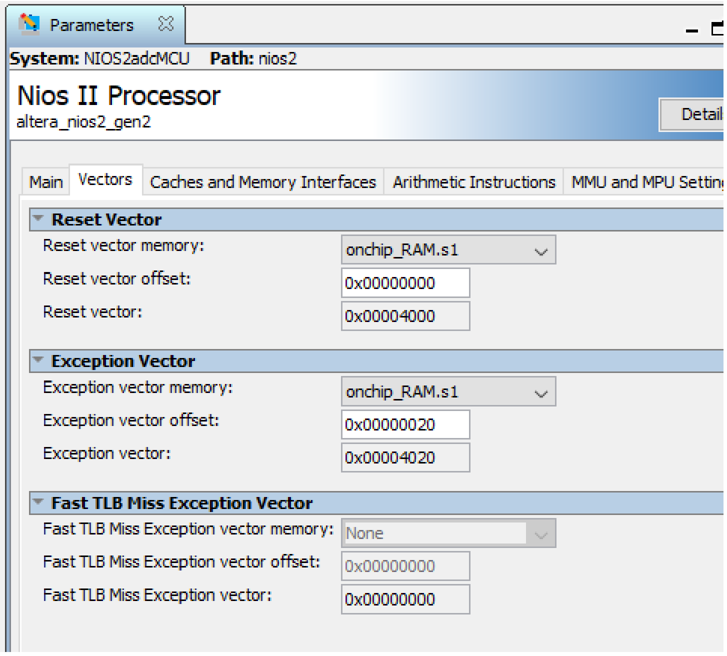
System: NIOS2adcMCU **Path:** pio_0.clk

Use	Connections	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*	
☑		⊟ **nios2**	Nios II Processor		
		clk	Clock Input	*Double-click to export*	**clk_0**
		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_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]
		reset1	Reset Input	*Double-click to export*	[clk1]
☑		⊟ **adc_0**	Modular ADC core Intel FPGA IP		
		clock	Clock Input	*Double-click to export*	**clk_0**
		reset_sink	Reset Input	*Double-click to export*	[clock]
		adc_pll_clock	Clock Input	*Double-click to export*	**pll_0_c0**
		adc_pll_locked	Conduit	*Double-click to export*	
		sequencer_csr	Avalon Memory Mapped Slave	*Double-click to export*	[clock]
		sample_store_csr	Avalon Memory Mapped Slave	*Double-click to export*	[clock]
		sample_store_irq	Interrupt Sender	*Double-click to export*	[clock]
☑		⊟ **pll_0**	ALTPLL Intel FPGA IP		
		inclk_interface	Clock Input	*Double-click to export*	**clk_0**
		inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interf...
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interf...
		c0	Clock Output	*Double-click to export*	pll_0_c0
		locked_conduit	Conduit	*Double-click to export*	
☑		⊟ **timer_0**	Interval Timer Intel FPGA IP		
		clk	Clock Input	*Double-click to export*	**clk_0**
		reset	Reset Input	*Double-click to export*	[clk]
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
		irq	Interrupt Sender	*Double-click to export*	[clk]
☑		⊟ **sysid_qsys_0**	System ID Peripheral Intel FPGA IP		
		clk	Clock Input	*Double-click to export*	**clk_0**
		reset	Reset Input	*Double-click to export*	[clk]
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
☑		⊟ **jtag_uart_0**	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]
		irq	Interrupt Sender	*Double-click to export*	[clk]
☑		⊟ **pio_0**	PIO (Parallel I/O) Intel FPGA IP		
		clk	Clock Input	*Double-click to export*	**clk_0**
		reset	Reset Input	*Double-click to export*	[clk]
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
		external_connection	Conduit	**led5**	

42. 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. You will see the base address values for each IP change in the System Contents tab.


Platform Designer - unsaved.qsys* (E:\FPGA\Intel_Max_10_FPGA_Evaluatic
File Edit System Generate View Tools Help
IP C
sys
Projec
Library
Bas
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...
Trace System
Sy

43. Finally, let’s set the reset and exception vector addresses. Double-click on the nios2 to open the configuration page.
44. Click on the Vectors tab.
45. Change the Reset vector memory drop-down to onchip_RAM.s1.
46. Change the Exception vector memory drop-down to onchip_RAM.s1.



System: NIOS2adcMCU **Path:** nios2

Nios II Processor
altera_nios2_gen2

Reset Vector

Reset vector memory: onchip_RAM.s1
Reset vector offset: 0x00000000
Reset vector: 0x00004000

Exception Vector

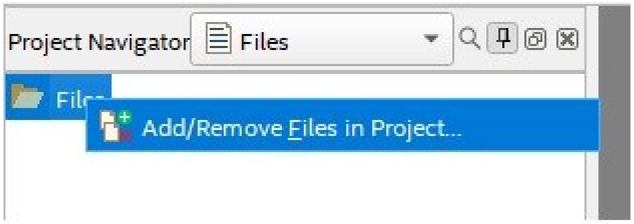
Exception vector memory: onchip_RAM.s1
Exception vector offset: 0x00000020
Exception vector: 0x00004020

Fast TLB Miss Exception Vector

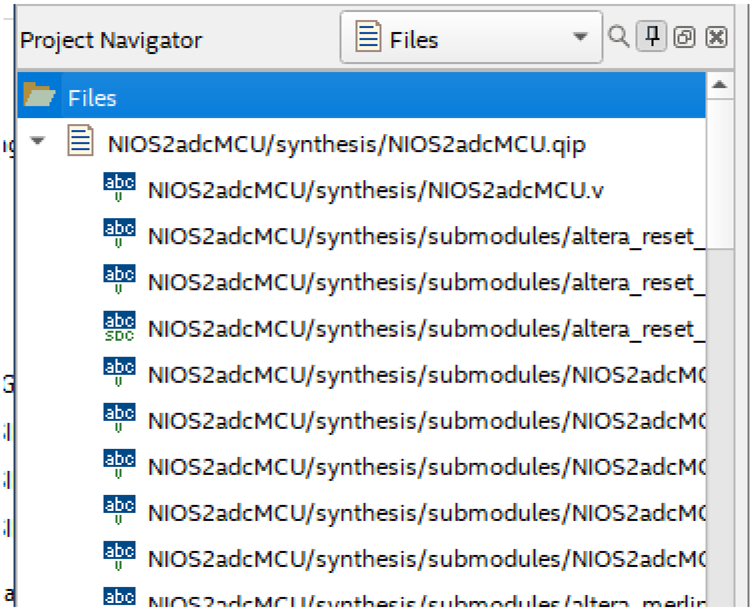
Fast TLB Miss Exception vector memory: None
Fast TLB Miss Exception vector offset: 0x00000000
Fast TLB Miss Exception vector: 0x00000000

47. Click on Generate HDL…
48. Keep the defaults and click the Generate button.
49. A dialog will appear asking you to save the design, click Save.
50. Give the name as NIOS2adcMCU.qsys, and click Save.
51. Once the save has completed, click Close.
52. The generate process kicks off. The processes should succeed, click Close.

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



57. A Settings – NIOS2adc page appears with Files on the left highlighted. Click the three dots browse button for File name, and navigate to the \NIOS2_ADC\NIOS2adcMCU\synthesis folder.
58. Click on the NIOS2adcMCU.qip file and click open.
59. Click OK to close the Settings- NIOS2adc page. The qip file is added to the Project navigator list. Underneath are all the Verilog files that were generated by Platform Designer.


Project Navigator
Files
Files
NIOS2adcMCU/synthesis/NIOS2adcMCU.qip
NIOS2adcMCU/synthesis/NIOS2adcMCU.v
NIOS2adcMCU/synthesis/submodules/altera_reset_
NIOS2adcMCU/synthesis/submodules/altera_reset_
NIOS2adcMCU/synthesis/submodules/altera_reset_
NIOS2adcMCU/synthesis/submodules/NIOS2adcM
NIOS2adcMCU/synthesis/submodules/NIOS2adcM
NIOS2adcMCU/synthesis/submodules/NIOS2adcM
NIOS2adcMCU/synthesis/submodules/NIOS2adcM
NIOS2adcMCU/synthesis/submodules/NIOS2adcM
NIOS2adcMCU/synthesis/submodules/altera_merlin

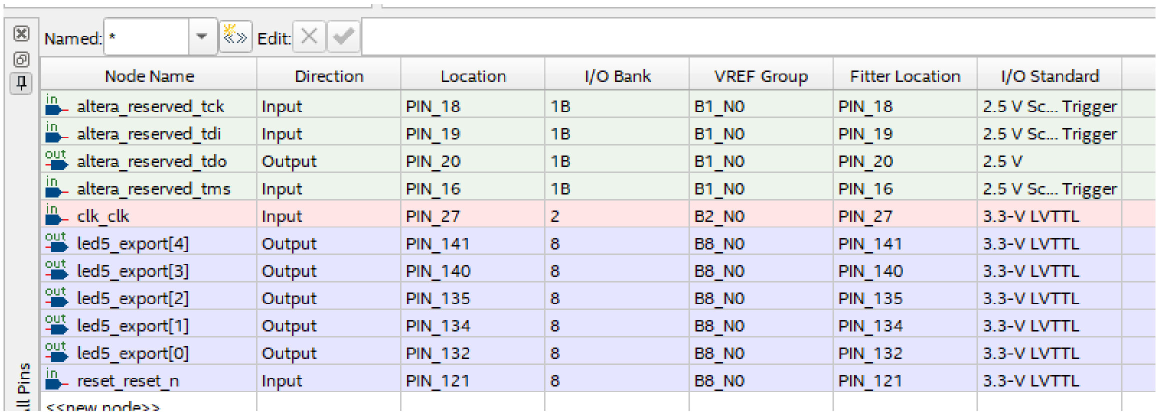
60. In the Project Navigator, Right-click on the NIOS2adcMCU/synthesis/NIOS2adcMCU.v file and select Set as Top-Level Entity from the context menu.
61. Save the project.
62. In the Task pane on the left, double-click on Fitter (Place & Route) to start the task. The analysis will take some time, and it 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.

63. 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 just run populated the Node Name list at the bottom of the Pin Planner dialog.

64. 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
Led5_export[4]	PIN_141
Led5_export[3]	PIN_140
Led5_export[2]	PIN_135
Led5_export[1]	PIN_134
Led5_export[0]	PIN_132

65. Set the I/O Standard to 3.3V-LVTTL for all pins except JTAG. You can see from the schematic that the I/O are all tied to 3.3V.



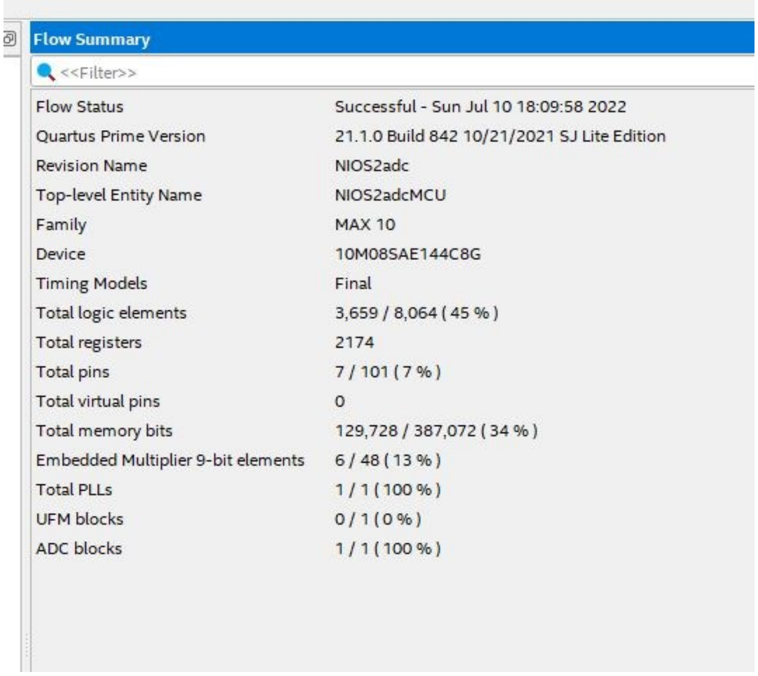
Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
altera_reserved_tck	Input	PIN_18	1B	B1_N0	PIN_18	2.5 V Sc... Trigger
altera_reserved_tdi	Input	PIN_19	1B	B1_N0	PIN_19	2.5 V Sc... Trigger
altera_reserved_tdo	Output	PIN_20	1B	B1_N0	PIN_20	2.5 V
altera_reserved_tms	Input	PIN_16	1B	B1_N0	PIN_16	2.5 V Sc... Trigger
clk_clk	Input	PIN_27	2	B2_N0	PIN_27	3.3-V LVTTL
led5_export[4]	Output	PIN_141	8	B8_N0	PIN_141	3.3-V LVTTL
led5_export[3]	Output	PIN_140	8	B8_N0	PIN_140	3.3-V LVTTL
led5_export[2]	Output	PIN_135	8	B8_N0	PIN_135	3.3-V LVTTL
led5_export[1]	Output	PIN_134	8	B8_N0	PIN_134	3.3-V LVTTL
led5_export[0]	Output	PIN_132	8	B8_N0	PIN_132	3.3-V LVTTL
reset_reset_n	Input	PIN_121	8	B8_N0	PIN_121	3.3-V LVTTL
<<new node>>						

66. Close the Pin Planner when finished. The diagram gets updated with the pin numbers.
67. 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. Archiving is simple. From the menu, Project->Archive Project.

68. 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. The design should compile successfully.



Flow Summary

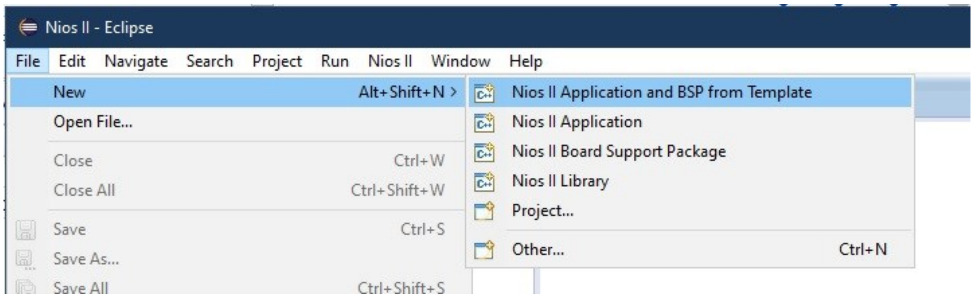
<<Filter>>

Flow Status	Successful - Sun Jul 10 18:09:58 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	NIOS2adc
Top-level Entity Name	NIOS2adcMCU
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	3,659 / 8,064 (45 %)
Total registers	2174
Total pins	7 / 101 (7 %)
Total virtual pins	0
Total memory bits	129,728 / 387,072 (34 %)
Embedded Multiplier 9-bit elements	6 / 48 (13 %)
Total PLLs	1 / 1 (100 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	1 / 1 (100 %)

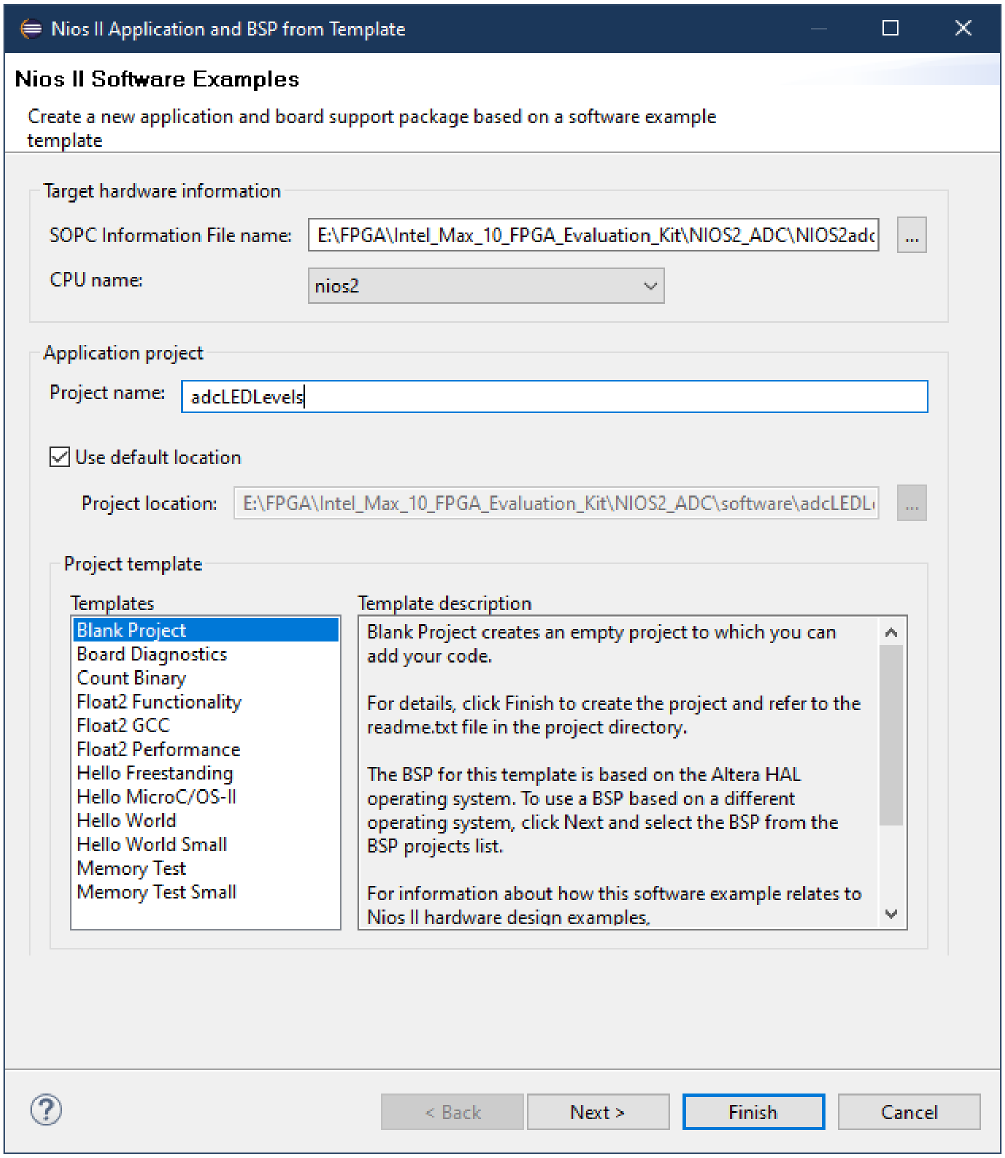
1.1.3 Eclipse Application 1: adcLEDLevels One Shot Read

Now, we are ready to create an application to run on the Nios II processor. The application will configure the ADC for a single read of the input analog signal and then light the LEDs based on the resulting voltage value. The one-shot solution is good when you only need to take a reading once in a while. For example, reading the TSD to get the FPGA temperature.

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 \NIOS2_UART\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 \NIOS2_ADC folder and open the NIOS2adcMCU.sopcinfo file. The CPU name will reflect the name we gave the CPU in Platform Builder.
6. Enter the project name: adcLEDLevels.
7. In the Project Template, select Blank Project.
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\NIOS2_ADC\NIOS2adc

CPU name: nios2

Application project

Project name: adcLEDLevels

☑ Use default location

Project location: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\software\adcLEDL

Project template

Templates: Blank Project (selected)

Template description

Blank Project creates an empty project to which you can add your code.

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. To use a BSP based on a different operating system, click Next and select the BSP from the projects list.

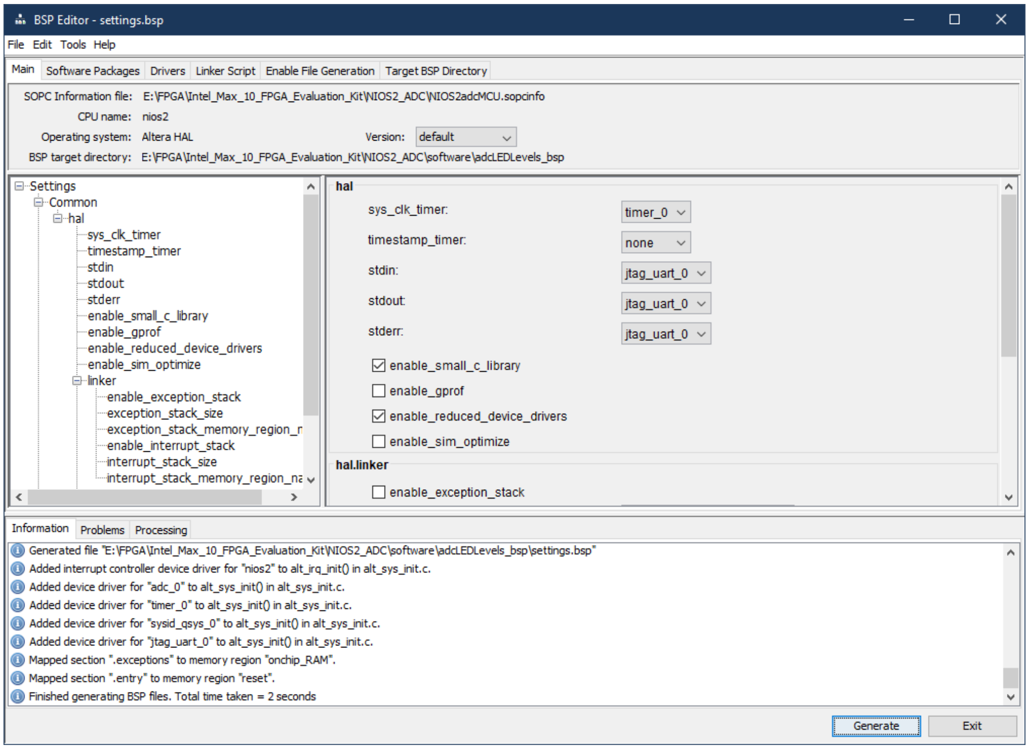
For information about how this software example relates to Nios II hardware design examples,

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

9. We need to edit the BSP to use the small C library. The BSP Editor tool allows you to edit the settings.bsp file to make specific changes for the target. Right-click on adcLEDLevels_bsp and select Nios II->BSP Editor from the context menu.

The BSP Editor opens and opens the settings.bsp file automatically. If you started the BSP Editor from the main menu, you would have to manually navigate to open the file. In the BSP Editor, you can see this is where the selection of the small_c library and reduced drivers are set. The standard input, output, and error ports to handle messages are already set to jtag_uart_0.

10. Tick the box for enable_small_c_library and enable_reduced_device_drivers, and click Generate to make the changes.



SOPC Information file: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\NIOS2adcMCU.sopcinfo
CPU name: nios2
Operating system: Altera HAL Version: default
BSP target directory: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\software\adcLEDLevels_bsp

hal

sys_clk_timer: timer_0
timestamp_timer: none
stdin: jtag_uart_0
stdout: jtag_uart_0
stderr: jtag_uart_0

☑ enable_small_c_library
□ enable_gprof
☑ enable_reduced_device_drivers
□ enable_sim_optimize

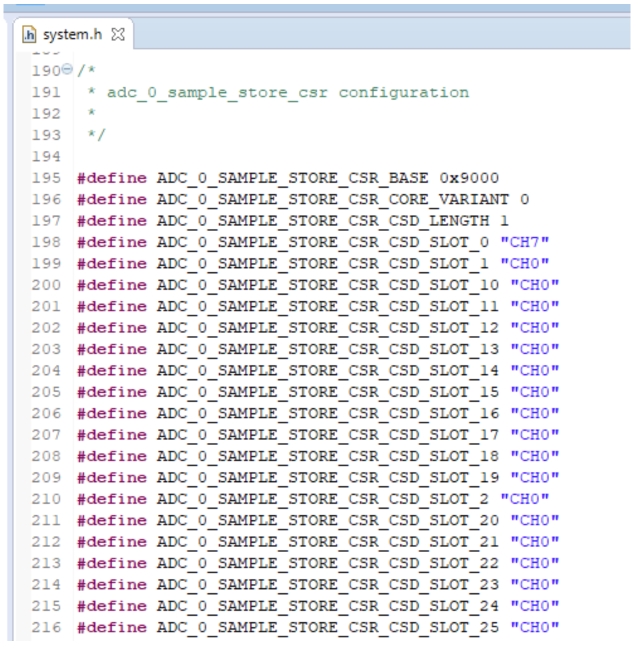
hal.linker

□ enable_exception_stack

```
Generated file "E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\software\adcLEDLevels_bsp\settings.bsp"
Added interrupt controller device driver for "nios2" to alt_irq_init() in alt_sys_init.c.
Added device driver for "adc_0" to alt_sys_init() in alt_sys_init.c.
Added device driver for "timer_0" to alt_sys_init() in alt_sys_init.c.
Added device driver for "sysid_qsys_0" to alt_sys_init() in alt_sys_init.c.
Added device driver for "jtag_uart_0" to alt_sys_init() in alt_sys_init.c.
Mapped section ".exceptions" to memory region "onchip_RAM".
Mapped section ".entry" to memory region "reset".
Finished generating BSP files. Total time taken = 2 seconds
```

11. Click Exit when finished.

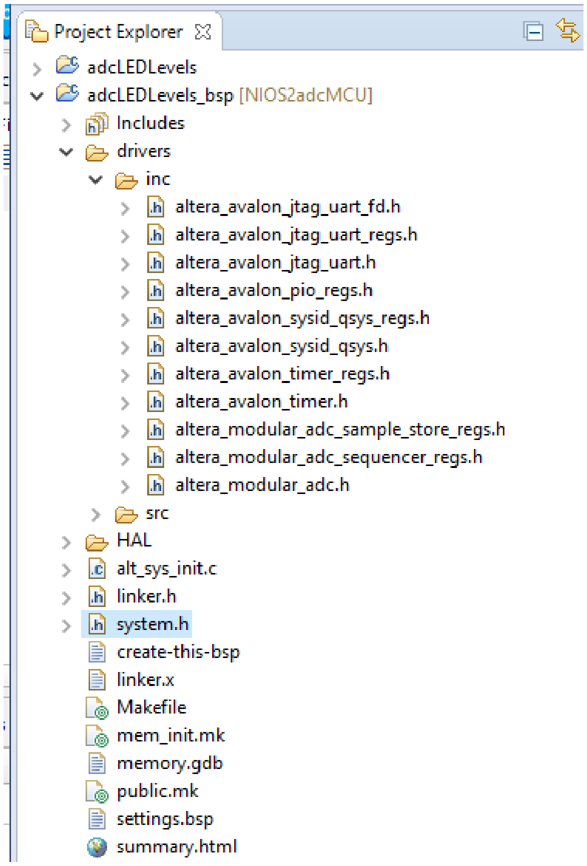
The adcLEDLevels_bsp contains the key files that will help with filling in the code to access the ADC and PIO ports. System.h contains the definitions that can be used in Platform Designer to set up the ADC and PIO. The ADC list is very long as the SampleStore and Sequencer define multiple values.



```
/*
 * adc_0_sample_store_csr configuration
 *
 */

#define ADC_0_SAMPLE_STORE_CSR_BASE 0x9000
#define ADC_0_SAMPLE_STORE_CSR_CORE_VARIANT 0
#define ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH 1
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_0 "CH7"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_1 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_10 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_11 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_12 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_13 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_14 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_15 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_16 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_17 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_18 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_19 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_2 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_20 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_21 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_22 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_23 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_24 "CH0"
#define ADC_0_SAMPLE_STORE_CSR_CSD_SLOT_25 "CH0"
```

Since we are using the small_C_library for space reasons, the standard C io calls cannot be used. Instead, we will be using the Nios II HAL API to access the ADC, PIO, and standard output via the

JTAG UART. The other header files are under the drivers\inc folder altera_avalon_adc_*.h. Each file contains the function prototypes of the commands that will be used in the application.


Project Explorer
adcLEDLevels
adcLEDLevels_bsp [NIOS2adcMCU]
Includes
drivers
inc
altera_avalon_jtag_uart_fd.h
altera_avalon_jtag_uart_regs.h
altera_avalon_jtag_uart.h
altera_avalon_pio_regs.h
altera_avalon_sysid_qsys_regs.h
altera_avalon_sysid_qsys.h
altera_avalon_timer_regs.h
altera_avalon_timer.h
altera_modular_adc_sample_store_regs.h
altera_modular_adc_sequencer_regs.h
altera_modular_adc.h
src
HAL
alt_sys_init.c
linker.h
system.h
create-this-bsp
linker.x
Makefile
mem_init.mk
memory.gdb
public.mk
settings.bsp
summary.html

Before we move on to writing the application, we need to fix bugs that are in the generation of the BSP. Intel has done a great job of taking the heavy HDL coding out of the design, but they forgot a few things.

12. In the adcLEDLevels _bsp project, expand Drivers\inc, and open the altera_modular_adc.h file. At about line 92 you will see the following:

```
#define ALTERA_MODULAR_ADC_INSTANCE(name, dev) \
static alt_modular_adc_dev dev =        \
{                                       \
  {                                     \
    ALT_LLIST_ENTRY,                    \
    name##_NAME,                        \
    NULL,                               \
    NULL,                               \
    NULL,                               \
    NULL,                               \
    NULL,                               \
    NULL,                               \
    NULL,                               \
  },                                    \
   NULL,                                \
   NULL,                                \
   0,                                   \
   0,                                  \
   name##_DUAL_ADC_MODE                \
}


/*
 * The macro ALTERA_MODULAR_ADC_INIT is called by the auto-generated
function
 * alt_sys_init() to initialize a given device instance.
 */
#define ALTERA_MODULAR_ADC_INIT(name, dev) \
 altera_modular_adc_init(&dev,     name##_IRQ_INTERRUPT_CONTROLLER_ID,
name##_IRQ);
```

This autogenerated file is not correct. The variable names are not set up correctly. For example, name##_NAME should actually be name##_SEQUENCER_CSR_NAME. This matches the system.h defines. Name## resolves to adc_0; the name we gave the ADC in Platform Designer.

13. Change the code to the following:

```
#define ALTERA_MODULAR_ADC_INSTANCE(name, dev) \
static alt_modular_adc_dev dev =       \
{                                      \
  {                                    \
    ALT_LLIST_ENTRY,                   \
    name##_SEQUENCER_CSR_NAME,          \
    NULL,                              \
    NULL,                              \
    NULL,                              \
    NULL,                              \
    NULL,                              \
    NULL,                              \
    NULL,                              \
  },                                   \
   NULL,                               \
   NULL,                               \
   0,                                  \
   0,                                  \
   name##_SEQUENCER_CSR_DUAL_ADC_MODE               \
}


/*
 * The macro ALTERA_MODULAR_ADC_INIT is called by the auto-generated
function
 * alt_sys_init() to initialize a given device instance.
 */
#define ALTERA_MODULAR_ADC_INIT(name, dev) \
            altera_modular_adc_init(&dev,
name##_SAMPLE_STORE_CSR_IRQ_INTERRUPT_CONTROLLER_ID,
name##_SAMPLE_STORE_CSR_IRQ);
```

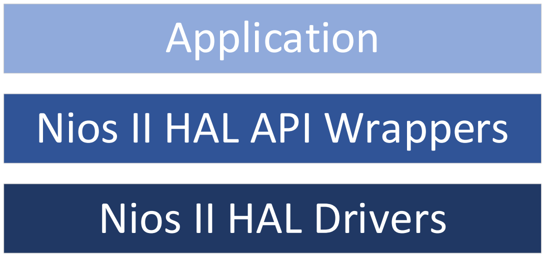
14. Save and close the file.

Note: Any time you have to update the adcLEDLevels_bsp project by generating a new BSP because of a hardware design change, you also have to fix this file again.

15. We need to add a main.c file to the project. Right-click on the adcLEDLevels project, and select New->File from the context menu.
16. Enter the file name main.c and click Finish.
17. Add the following code to the main.c file.

```
#include "sys/alt_stdio.h"
#include "system.h"
#include "priv/alt_busy_sleep.h"
#include "sys/alt_sys_wrappers.h"
#include "altera_modular_adc.h"
#include "altera_modular_adc_sequencer_regs.h"
#include "altera_modular_adc_sample_store_regs.h"
#include "altera_avalon_pio_regs.h"


int main()
{
  alt_putstr("ADC and LED test!\n");

  alt_u32 adc_slot_data[64]; //There are 64 slots available
  alt_u32 slot_value_data;
  int x = 0;
  //Set all the slots to be zero
  for(x = 0; x < 64; x++){
        adc_slot_data[x]=0;
  }

  adc_stop(ADC_0_SEQUENCER_CSR_BASE);
  adc_set_mode_run_once(ADC_0_SEQUENCER_CSR_BASE);
  adc_interrupt_disable(ADC_0_SAMPLE_STORE_CSR_BASE);

  adc_start(ADC_0_SEQUENCER_CSR_BASE);

  alt_adc_word_read(ADC_0_SAMPLE_STORE_CSR_BASE, adc_slot_data, ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH); //fill in all the slots
  slot_value_data = adc_slot_data[0]; //CH7 is set for slot 1 (the values are off set by 1, thus 0 for the array).
  alt_printf("ADC Value (HEX) from wrapper: %x\n", slot_value_data);

  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1F);

  //LEDs act like a level each one turns on the higher the voltage.
  //Since the LEDs are active low, the 0s turn them on. Saves on having to add NOT gates for each line in the design.
  if(slot_value_data > 700){
       IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1E);
  }
  if(slot_value_data > 1500){
       IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1C);
  }
  if(slot_value_data > 2300){
       IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x18);
  }
46.     if(slot_value_data > 3000){
47.          IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x10);
48.     }
49.     if(slot_value_data > 3600){
50.          IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x0);
51.     }
52.
53.     return 0;
54.   }
```

The basic concept for programming on top of the provided HAL drivers is through the use of the HAL API Wrappers. The various driver header files contain the wrapper APIs that are used to access the ADC and PIO.

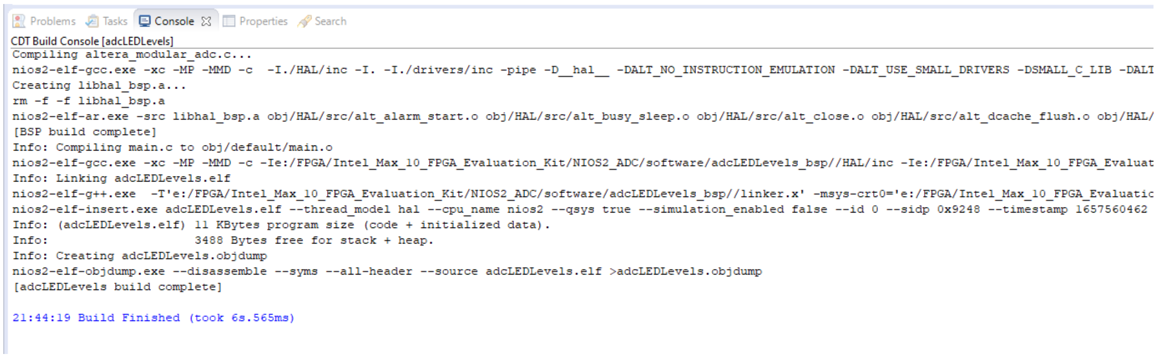

Application
Nios II HAL API Wrappers
Nios II HAL Drivers

Lines 15-19 set up the slot buffer. There are 64 possible slots available, but only slot 1 is being used for CH7. For completeness, the buffer has room for all 64.

Lines 23-25 configure the ADC for single-reading. The sequencer is stopped, the sequencer mode is set to run once, and interrupts are disabled.

Once the ADC has been set up, Line 27 starts the sequencer to take one reading. The slot buffer is filled with the results and the CH7/Slot1 result is then sent to the standard I/O. The value is never converted to an actual voltage. The values can range from 0 to 4095 (0xfff). The data results are then used to turn on the corresponding LEDs. The tolerance of the resistor pots can yield different results so you can adjust the values and the LEDs that get turned on accordingly.

18. Save the file.
19. Right-click on adcLEDLevels project again, and select Build Project. The build should complete successfully, and the adcLEDLevels.elf file should have been created.



CDT Build Console [adcLEDLevels]

```
Compiling altera_modular_adc.c...
nios2-elf-gcc.exe -xc -MP -MMD -c  -I./HAL/inc -I. -I./drivers/inc -pipe -D__hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_USE_SMALL_DRIVERS -DSMALL_C_LIB -DALT
Creating libhal_bsp.a...
rm -f -f libhal_bsp.a
nios2-elf-ar.exe -src libhal_bsp.a obj/HAL/src/alt_alarm_start.o obj/HAL/src/alt_busy_sleep.o obj/HAL/src/alt_close.o obj/HAL/src/alt_dcache_flush.o obj/HAL/
[BSP build complete]
Info: Compiling main.c to obj/default/main.o
nios2-elf-gcc.exe -xc -MP -MMD -c -Ie:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_ADC/software/adcLEDLevels_bsp//HAL/inc -Ie:/FPGA/Intel_Max_10_FPGA_Evaluat
Info: Linking adcLEDLevels.elf
nios2-elf-g++.exe  -T'e:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_ADC/software/adcLEDLevels_bsp//linker.x' -msys-crt0='e:/FPGA/Intel_Max_10_FPGA_Evaluatic
nios2-elf-insert.exe adcLEDLevels.elf --thread_model hal --cpu_name nios2 --qsys true --simulation_enabled false --id 0 --sidp 0x9248 --timestamp 1657560462
Info: (adcLEDLevels.elf) 11 KBytes program size (code + initialized data).
Info:                    3488 Bytes free for stack + heap.
Info: Creating adcLEDLevels.objdump
nios2-elf-objdump.exe --disassemble --syms --all-header --source adcLEDLevels.elf >adcLEDLevels.objdump
[adcLEDLevels build complete]

21:44:19 Build Finished (took 6s.565ms)
```

20. Close Eclipse

Now, we are ready to program the board with the design and debug the application.

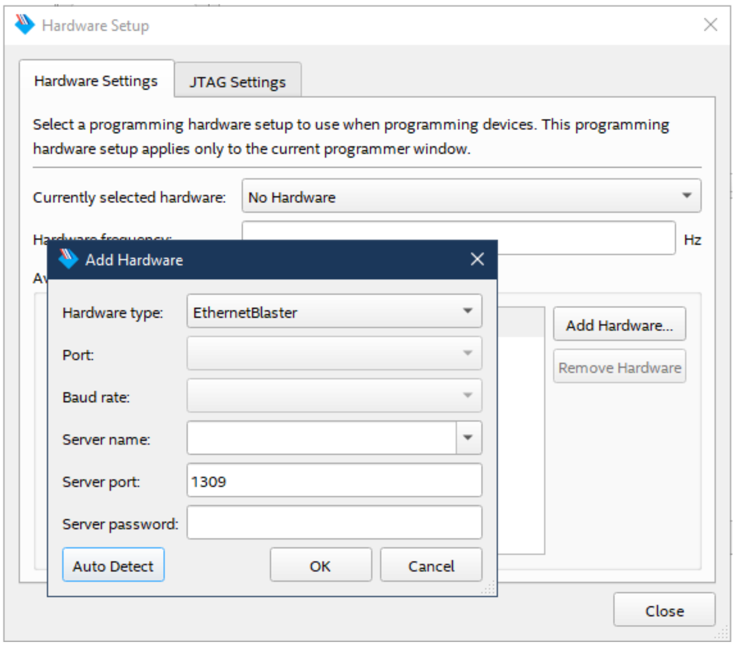
1.1.4 Program the Board

With the design compiled, application ready, and circuit connected, we can now test the design on the board.

1. Connect the board and the programming cable 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 for this example. 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:

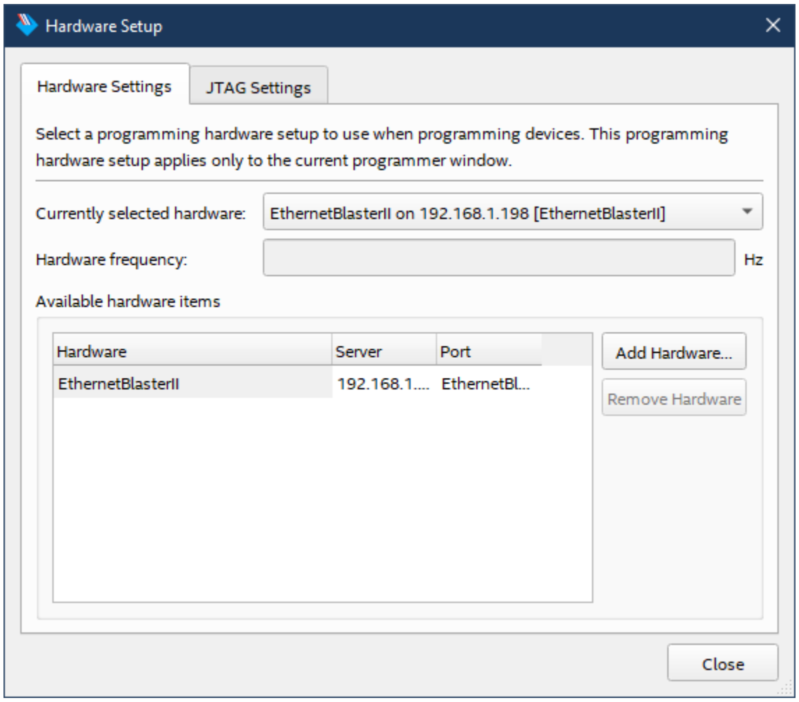
Baud rate:

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 and select the hardware cable for the board, and click Close when finished



Hardware Setup

Hardware Settings | JTAG Settings

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]

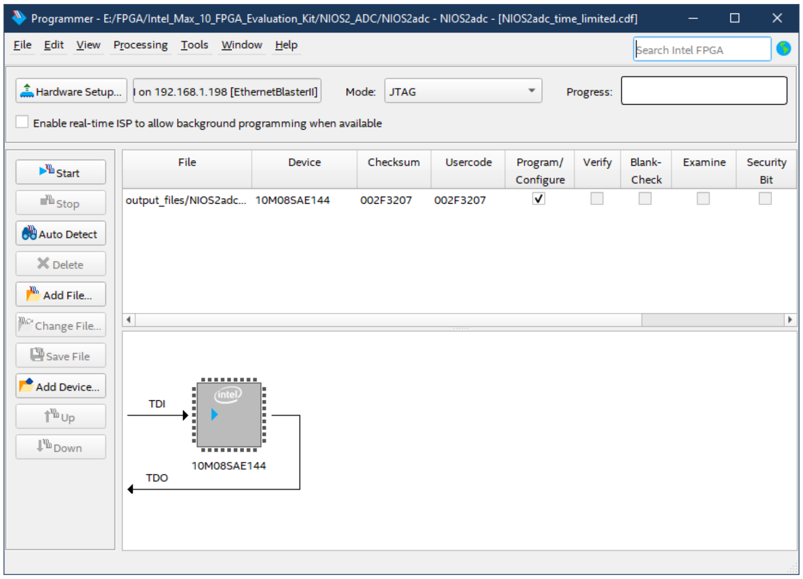
Hardware frequency: Hz

Available hardware items

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

7. A NIOS2adc_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 completed successfully.

Note: The reason for the “time_limited” in the name of the .sof file is that we chose an 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.



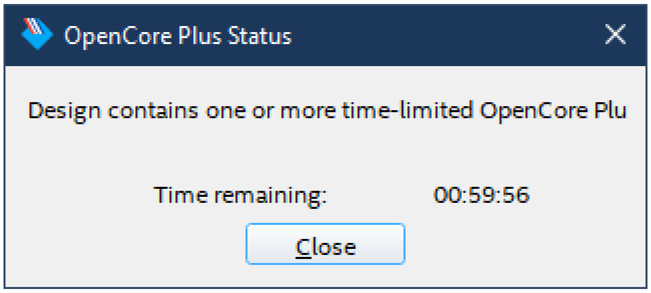
Programmer - E:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_ADC/NIOS2adc - NIOS2adc - [NIOS2adc_time_limited.cdf]

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	Examine	Security Bit
output_files/NIOS2adc...	10M08SAE144	002F3207	002F3207	☑	□	□	□	□

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



OpenCore Plus Status

Design contains one or more time-limited OpenCore Plu

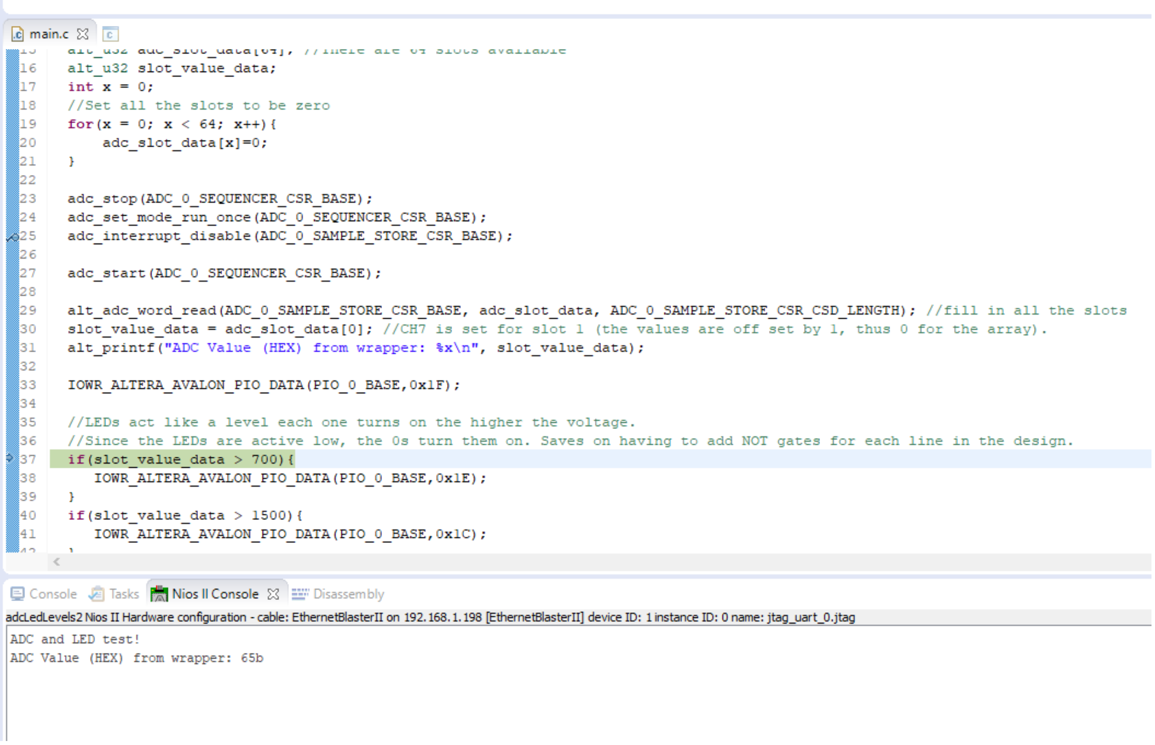
Time remaining: 00:59:56

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 Deploy the Application in Eclipse

With the design loaded and the connection to JTAG up and running, we can test the application.

1. From the Quartus menu, select Tools-> Nios II Software Build Tools for Eclipse.
2. Open the main.c application.
3. Toggle a breakpoint at line 25, when the ADC interrupt is disabled.
4. Right-click on adcLEDLevels and select Debug As->Nios II Hardware.
5. The program will load and start running.
6. Click F8 or the resume button to jump to the breakpoint.
7. Step through the program to start the ADC sequencer and read the data into the slot buffers. In the watch list, the slot buffer will only have one value in the first slot. Continue to step through the program and LEDs are turned on based on the value.



```
15     alt_u32 adc_slot_data[64]; //There are 64 slots available
16     alt_u32 slot_value_data;
17     int x = 0;
18     //Set all the slots to be zero
19     for(x = 0; x < 64; x++){
20         adc_slot_data[x]=0;
21     }
22
23     adc_stop(ADC_0_SEQUENCER_CSR_BASE);
24     adc_set_mode_run_once(ADC_0_SEQUENCER_CSR_BASE);
25     adc_interrupt_disable(ADC_0_SAMPLE_STORE_CSR_BASE);
26
27     adc_start(ADC_0_SEQUENCER_CSR_BASE);
28
29     alt_adc_word_read(ADC_0_SAMPLE_STORE_CSR_BASE, adc_slot_data, ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH); //fill in all the slots
30     slot_value_data = adc_slot_data[0]; //CH7 is set for slot 1 (the values are off set by 1, thus 0 for the array).
31     alt_printf("ADC Value (HEX) from wrapper: %x\n", slot_value_data);
32
33     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1F);
34
35     //LEDs act like a level each one turns on the higher the voltage.
36     //Since the LEDs are active low, the 0s turn them on. Saves on having to add NOT gates for each line in the design.
37     if(slot_value_data > 700){
38        IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1E);
39     }
40     if(slot_value_data > 1500){
41        IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1C);
```

adcLedLevels2 Nios II Hardware configuration - cable: EthernetBlasterII on 192.168.1.198 [EthernetBlasterII] device ID: 1 instance ID: 0 name: jtag_uart_0.jtag

```
ADC and LED test!
ADC Value (HEX) from wrapper: 65b
```

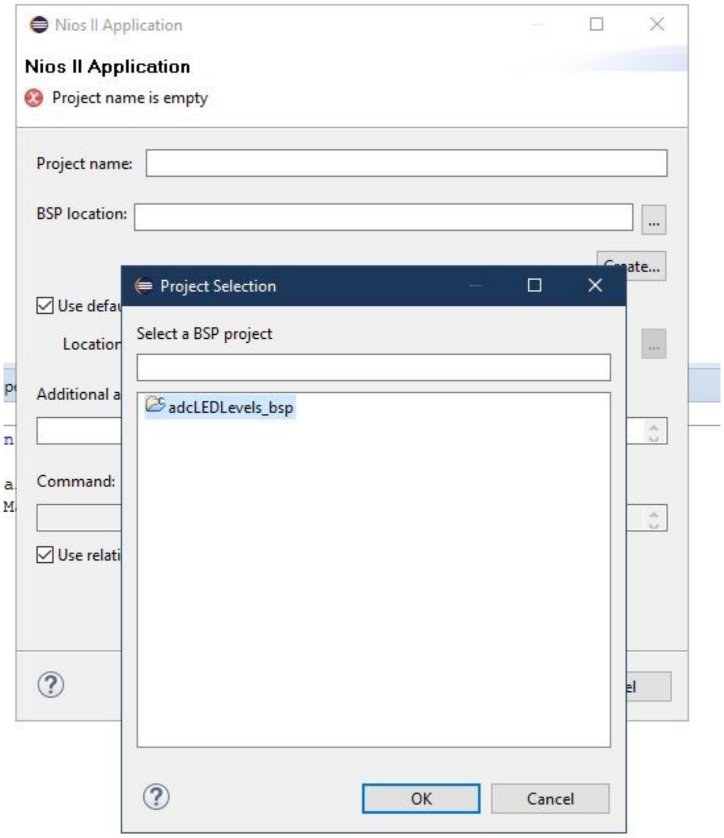
8. When finished close Eclipse, the OpenCore Plus dialog, and the JTAG programming application.

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

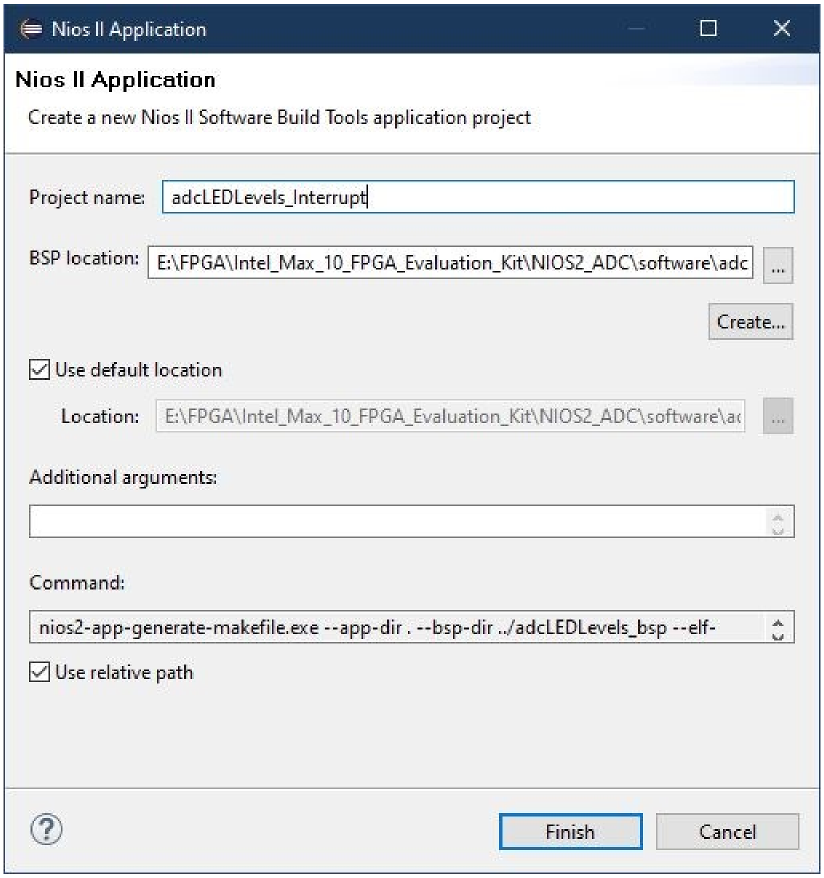
1.1.6 Eclipse Application 2: adcLEDLevels_Interrupt Continuous Read

The ADC in this application will be set to run continuously. An interrupt will be set to signal when data is in the buffer ready to be read. There will be a while-loop that waits for the interrupt, reads the data, and turns on the LEDs like the first application.

1. We can take advantage of the already create adcLEDLevels_bsp to create this new application. From the menu in Eclipse, select File->New->Nios II Application.
2. The dialog that appears asks for the name and the BSP project in the workspace.
3. Click on the 3 dots button, select the adcLEDLevels_bsp project, and click OK.



4. Enter the project name adcLEDLevels_Interrupt.



Nios II Application

Create a new Nios II Software Build Tools application project

Project name: adcLEDLevels_Interrupt

BSP location: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\software\adc

☑ Use default location

Location: E:\FPGA\Intel_Max_10_FPGA_Evaluation_Kit\NIOS2_ADC\software\ac

Additional arguments:

Command:

```
nios2-app-generate-makefile.exe --app-dir . --bsp-dir ../adcLEDLevels_bsp --elf-
```

☑ Use relative path

5. Click Finish.
6. Right-click on adcLEDLevels_Interrupt and select New-> File from the context menu.
7. Enter the name main.c and click Finish.
8. In the main.c file, enter the following:

```c
#include "sys/alt_stdio.h"
#include "system.h"
#include "priv/alt_busy_sleep.h"
#include "sys/alt_sys_wrappers.h"
#include "altera_modular_adc.h"
#include "altera_modular_adc_sequencer_regs.h"
#include "altera_modular_adc_sample_store_regs.h"
#include "altera_avalon_pio_regs.h"


void adc0_handler (void *context)
{
       adc_interrupt_disable(ADC_0_SAMPLE_STORE_CSR_BASE);

}


int main()
{
       alt_putstr("ADC and LED test!\n");
       IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1F);


  alt_u32 adc_slot_data[64]; //There are 64 slots available
25.      alt_u32 slot_value_data;
26.      int x = 0;
27.      //Set all the slots to be zero
28.      for(x = 0; x < 64; x++){
29.            adc_slot_data[x]=0;
30.      }
31.
32.      alt_modular_adc_dev adc0_dev, *p_adc0;
33.      p_adc0 = &adc0_dev;
34.
35.      adc_stop(ADC_0_SEQUENCER_CSR_BASE);
36.      p_adc0 = altera_modular_adc_open(ADC_0_SEQUENCER_CSR_NAME);
37.      alt_adc_register_callback (p_adc0, adc0_handler, NULL,
       ADC_0_SAMPLE_STORE_CSR_BASE);
38.      adc_set_mode_run_continuously(ADC_0_SEQUENCER_CSR_BASE);
39.      adc_clear_interrupt_status(ADC_0_SAMPLE_STORE_CSR_BASE);
40.      adc_interrupt_enable(ADC_0_SAMPLE_STORE_CSR_BASE);
41.      adc_start(ADC_0_SEQUENCER_CSR_BASE);
42.
43.
44.     while(1){
45.
46.           adc_wait_for_interrupt(ADC_0_SAMPLE_STORE_CSR_BASE);
47.           alt_adc_word_read(ADC_0_SAMPLE_STORE_CSR_BASE, adc_slot_data,
       ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH); //fill in all the slots
48.           slot_value_data = adc_slot_data[0]; //CH7 is set for slot 1
       (the values are off set by 1, thus 0 for the array).
49.           alt_printf("ADC Value (HEX) from wrapper: %x\n",
       slot_value_data);
50.
51.           IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1F);
52.
53.           //LEDs act like a level each one turns on the higher the
       voltage.
54.           //Since the LEDs are active low, the 0s turn them on. Saves on
       having to add NOT gates for each line in the design.
55.           if(slot_value_data > 700){
56.                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1E);
57.           }
58.           if(slot_value_data > 1500){
59.                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x1C);
60.           }
61.           if(slot_value_data > 2300){
62.                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x18);
63.           }
64.           if(slot_value_data > 3000){
65.                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x10);
66.           }
67.           if(slot_value_data > 3600){
68.                  IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,0x0);
69.           }
70.
71.         alt_busy_sleep(10000);
72.     }
73.
74.      return 0;
75.    }
```

For a continuously running ADC sequencer, an interrupt handler is needed to stop interrupts so the slot buffers can be read. Lines 11-15 define the handler. The handler simply disables the interrupts. Lines 24-30 set up the slot buffer as the first application. Lines 32-41 set up an instance of the alt_modular_adc_dev structure and a pointer based on the structure. The pointer is set to point to the address of the instance. The pointer is then used to open the adc0 based on the sequencer name. The handler is then registered as the callback of the adc0 instance. The sequencer is then set to run continuously, the interrupt status is cleared and the interrupt is then enabled. The sequencer is then started.

The while-loop contains the main running application. The program will wait for the adc0 interrupt status register (ISR) to be set to 1. With the handler stopping the interrupts, the ISR can be set and the read of the buffer can take place. If the interrupt is not stopped, the wait will go on forever. Once the buffer has been read, the interrupt is then turned back on by the driver. The program then writes out the value to the standard output and turns on the LEDs accordingly. The program will continue to loop.

9. Save the file.
10. Build the application.
11. Set a breakpoint at line 41, starting the sequencer.
12. Make sure that you have deployed the design to the FPGA and the OpenCore Status dialog is running.
13. Right-click on adcLEDLevels_Interrupt and select Debug As->Nios II Hardware.
14. The program will load and start running.
15. Click F8 or the resume button to jump to the breakpoint.
16. Step through the program to start the ADC sequencer. As you step through the while-loop and get to the wait-on-interrupt, you will notice a slight pause in the processing before you can continue stepping through the code. If you want to set a breakpoint in the adc0_handler, you can see when the handler has been triggered. The program runs like the first program but in a continuous loop to repeatedly read the data from the ADC, stopping the interrupt each time to get and display the data.
17. Hit F8 and the application will continue to run at full speed and you can adjust the trimmer pot and watch the LEDs turn on and off like a bar graph as the voltage level changes.
18. When finished, close Eclipse, the OpenCore Plus dialog, and JTAG programming application.

1.2 Summary: Limited Documentation and Examples

The documentation on the API wrappers for the ADC is limited to the source code. Any clue on how to use the API in an application is left up to guesswork, experience, and searching the Intel community platforms for answers. Internet searches resulted in several half-baked examples based on older versions of Nios II HAL code. For someone new to this development, these examples can be confusing and misleading. Some examples confused the sequencer and sample store defines when calling the APIs, which is a mistake that can crash the application. Hopefully, the walk-through of this design and the two types of ADC applications will help you with your project.

1.3 References

The following reference were used for this article:

- Intel® MAX® 10 Analog to Digital Converter User Guide - https://www.intel.com/content/www/us/en/docs/programmable/683596/20-1/analog-to-digital-converter-overview.html
- Introduction to Analog to Digital Conversion in Intel® MAX® 10 Devices Parts 1 and 2 - Intel FPGA training site Intel® FPGA Technical Training
- Using the ADC Toolkit in Intel® MAX® 10 Devices - Intel FPGA training site Intel® FPGA Technical Training

- How to Create ADC Design in MAX 10 Device Using Qsys Tool - https://cdrdv2.intel.com/v1/dl/getContent/649255?explicitVersion=true / https://www.youtube.com/watch?v=0oO1RFa-4Xk
- Intel® MAX® 10-10M08 Evaluation Kit schematic file. Altera_10M08S_E144_eval_schematic_REV_1_0.pdf.

The following are a few Nios II ADC applications reference found with an Internet search:

- http://leliuria.blog.jp/archives/49026265.html
- https://community.intel.com/t5/FPGA-Intellectual-Property/Modular-ADC-MAX10/td-p/182084
- https://faculty-web.msoe.edu/johnsontimoj/EE3921/files3921/max10_adc_nios_example.pdf