

# Nios® II + UART Project on Intel® MAX® 10-10M08 Evaluation Kit

By Sean D. Liming and John R. Malin  
Annabooks, LLC. – [www.annabooks.com](http://www.annabooks.com)

December 2022

This hands-on article will walk through the creation of a couple of Nios II + UART designs. We will explore some tool features along the way.

Please, see the [article Intel® Quartus® Prime Lite and Nios® II SBT for Eclipse Installation Instructions](#) on Annabooks.com for information on how 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 are 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 onboard.
- SchmartBoard RS-232 to UART board: [RS-232 I/O Module | Schmartboard](#). An RS-232 transceiver chip is needed to convert the RX and TX digital signals to the expected RS-232 signals.
- Serial null modem cable.
- RS-232-to-USB adapter.
- A terminal program such as ABCOMTerm ([www.annabooks.com](http://www.annabooks.com)), HyperTerminal, or Tera Term.
- Optional: SparkFun 16 x 2 matrix Serial LCD with Serial Interface. SparkFun has a few of these LCD displays that support Serial, I2C, and SPI interfaces: [SparkFun 16x2 SerLCD – RGB Text \(Qwiic\) – LCD-16397 – SparkFun Electronics](#)
- *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.

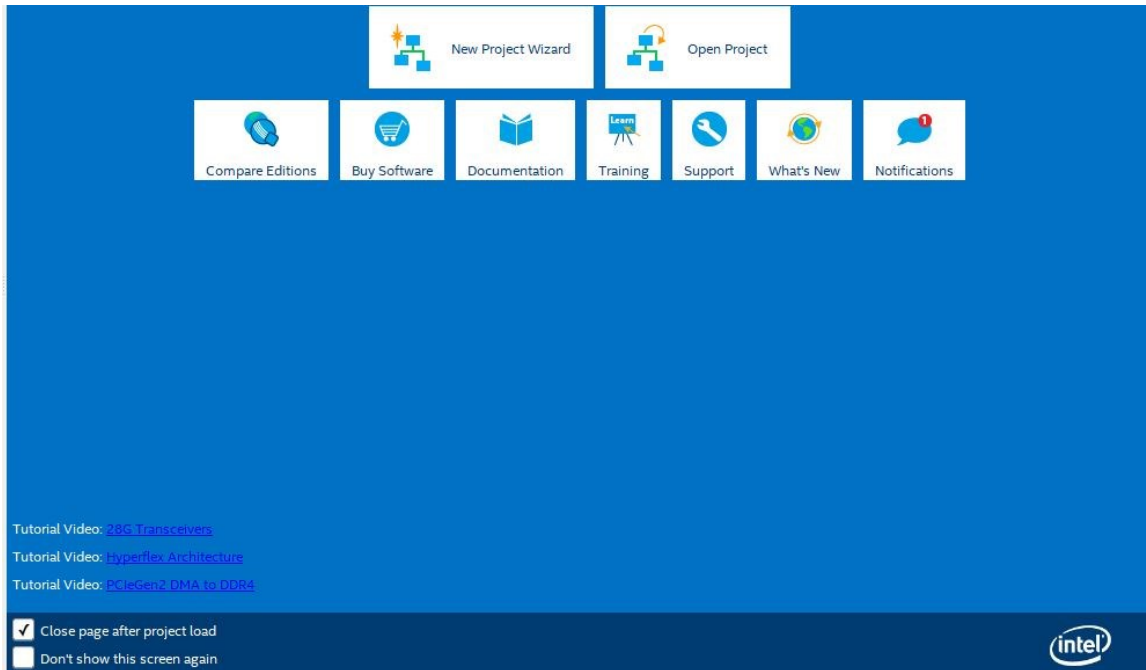
## 1.1 Simple Nios II UART Project

For this design, an application will run on the Nios II processor to send communications back and forth over a serial cable. There are two parts to the design process: the first involves creating the hardware design in Quartus Prime and Platform Builder, and the second step is to create the application with Eclipse.

### 1.1.1 Create the Project

The first step is to create the design project.

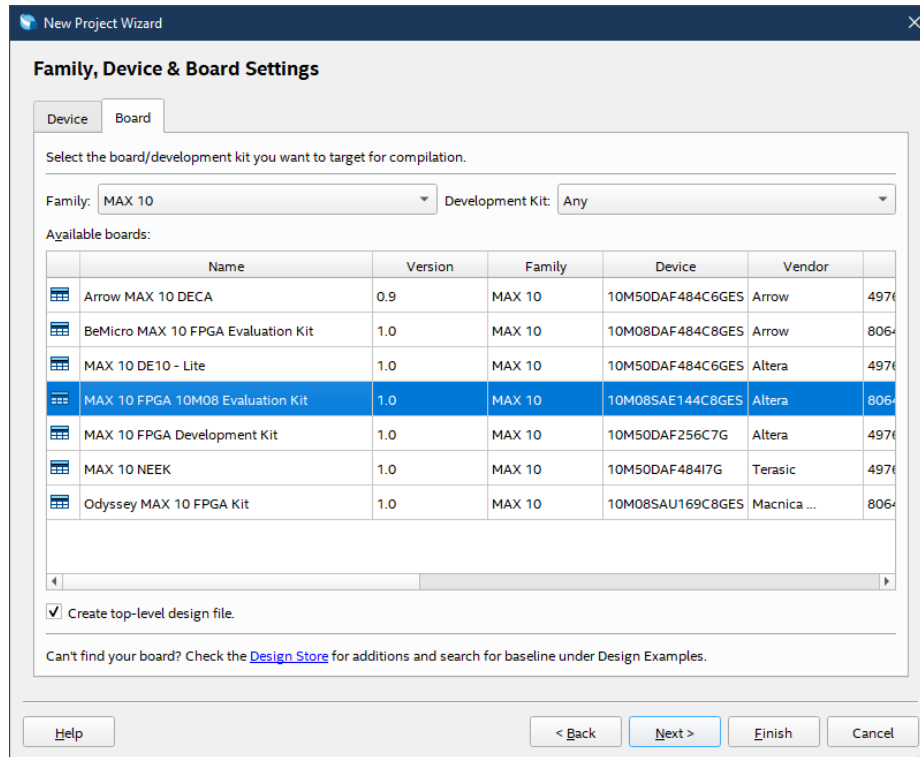
1. Open Quartus.
2. Click on the New Project Wizard.



3. Click Next to the Introduction dialog.
4. Select or create a project directory \NIOS2\_UART (Do not use the Quartus installation directory) and name of the project: "NIOS2uart". 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 names of the folders or projects.

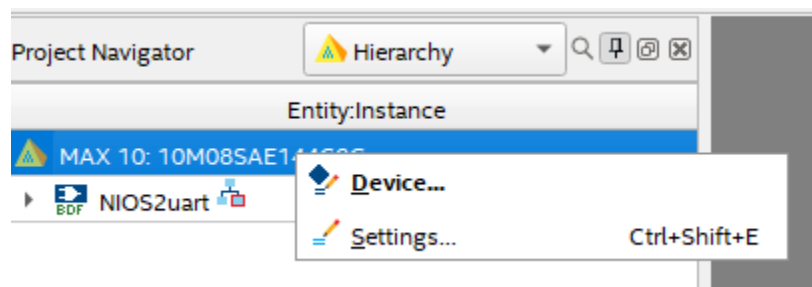
5. Project Type: Empty project, click Next.
6. Add File: no files to add, click Next.
7. Family, Device & Board Settings: click the Board tab and select: MAX 10 FPGA 10M08 Evaluation Kit. Click Next.



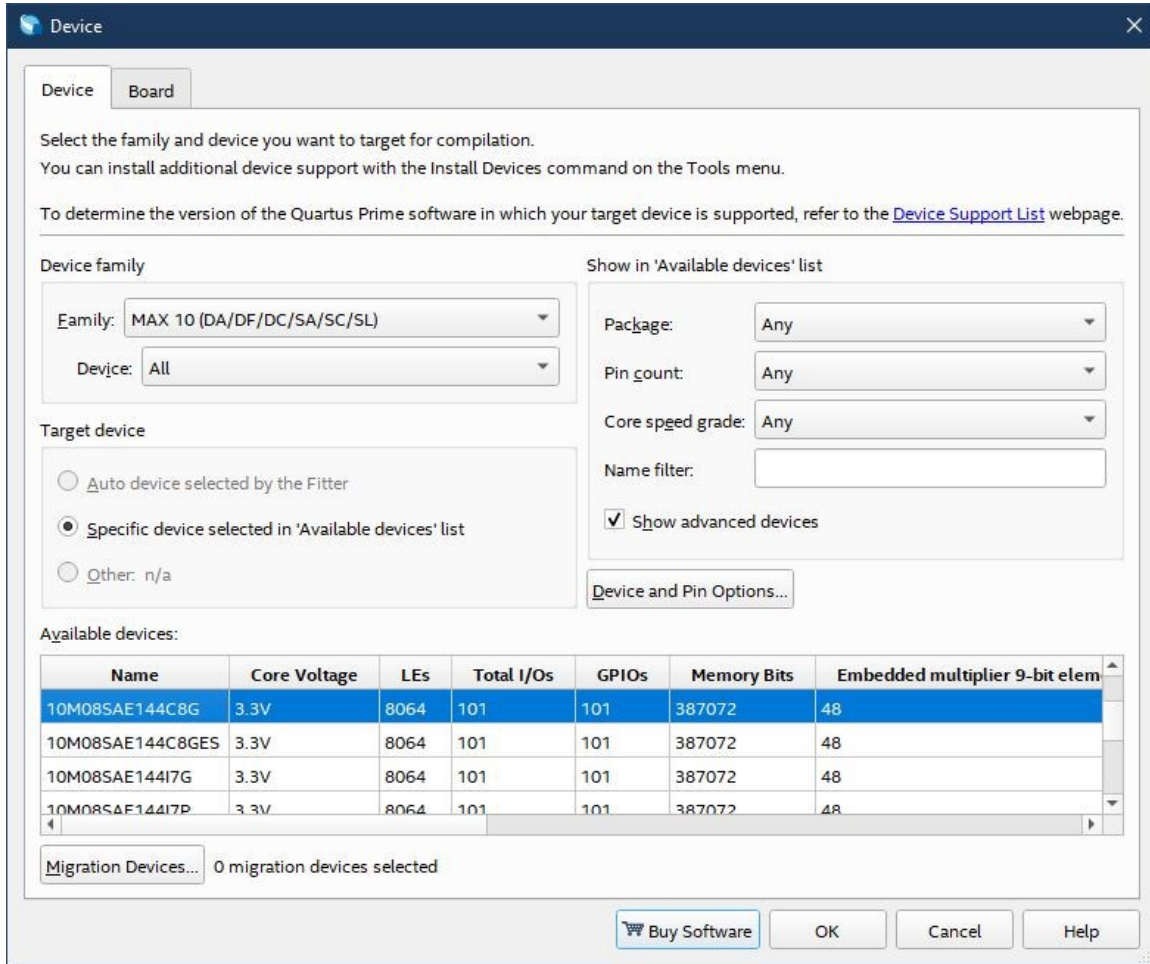
8. EDA Tools: click Next.
9. Summary: click Finish.

**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 Navigator 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.



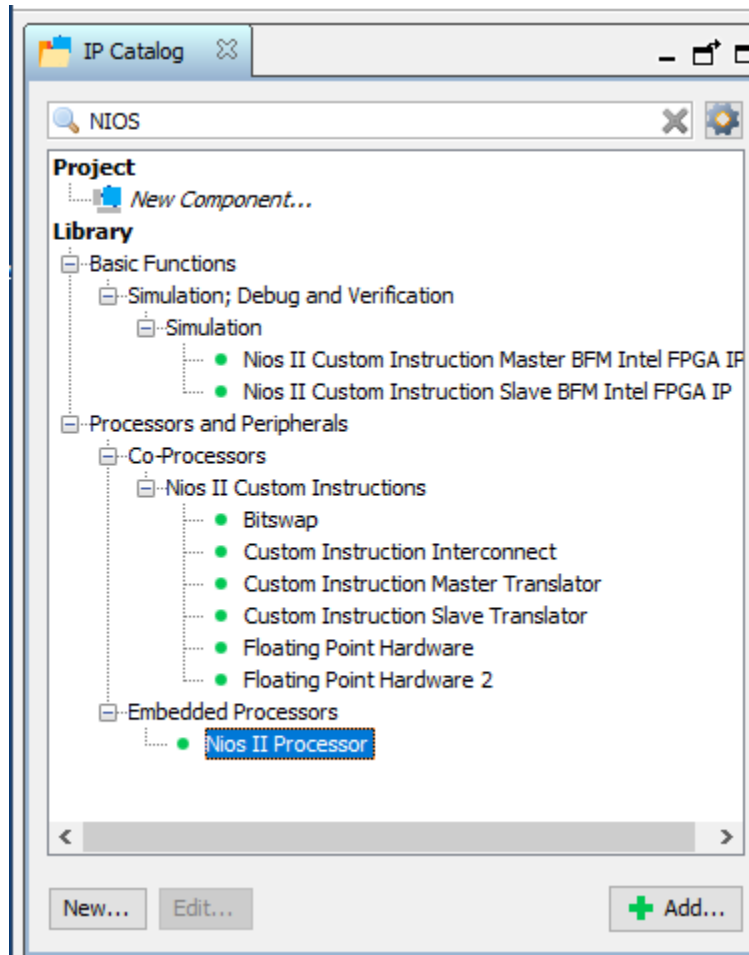
### 1.1.2 Create the Design Step 1: Platform Designer

Now, we will create the platform in Platform Designer.

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.



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 used to select the type of core: Nios II/e or Nios II/f. We will keep the defaults for now. Click Finish.

The screenshot shows the Nios II Processor configuration tool. The main window is titled "Nios II Processor" and "altera\_nios2\_gen2". The "Block Diagram" tab is active, showing a block diagram of the "nios2\_gen2\_0" processor. The diagram includes several input and output signals: "clk", "reset", "jtag", "debug\_mem\_slave", "data\_master", "instruction\_master", "debug\_reset\_request", "reset", "nios\_custom\_instruction", and "custom\_instruction\_master".

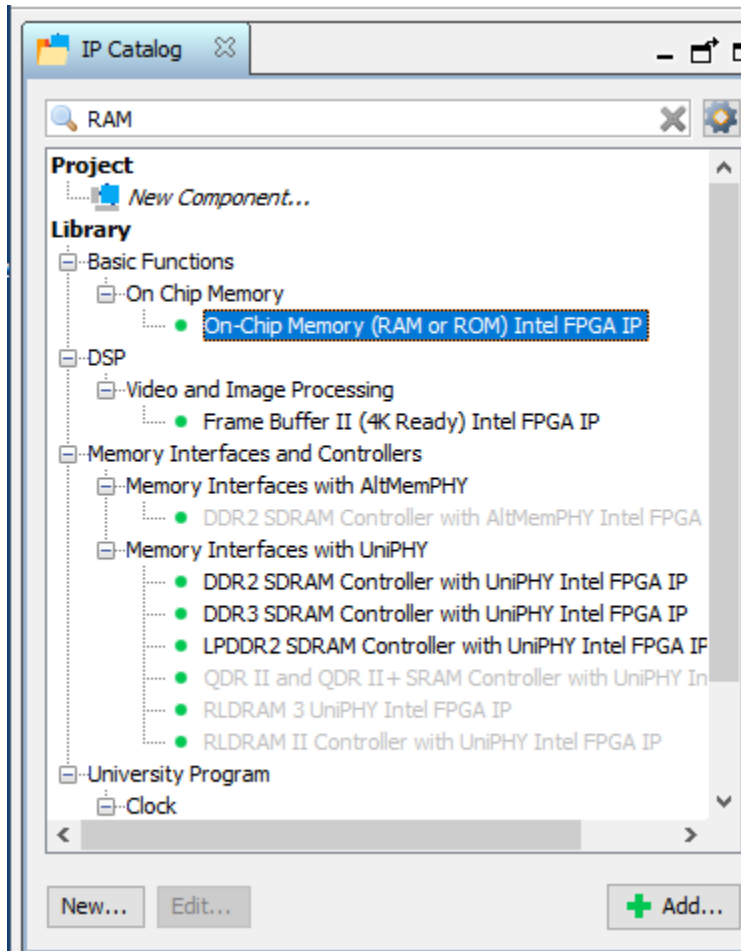
The "Select an Implementation" tab is also visible, showing two options for the Nios II Core: "Nios II/e" and "Nios II/f". The "Nios II/f" option is selected. Below the selection, there is a table comparing the two implementations:

	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 HPU HPU
RAM Usage	2 + Options	2 + Options

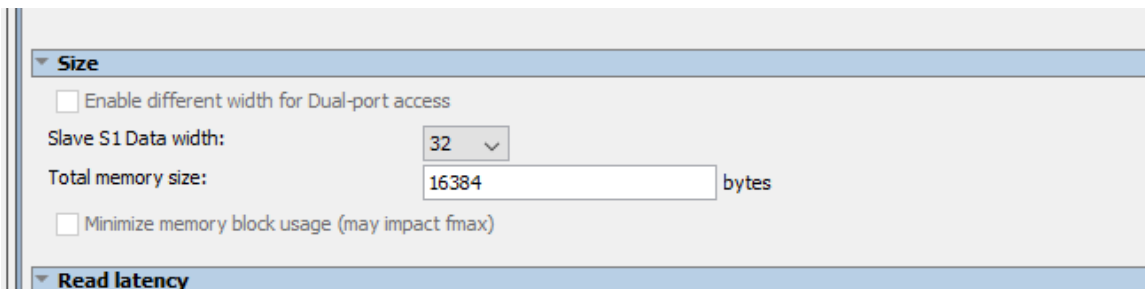
At the bottom of the window, there are three error messages:

- 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

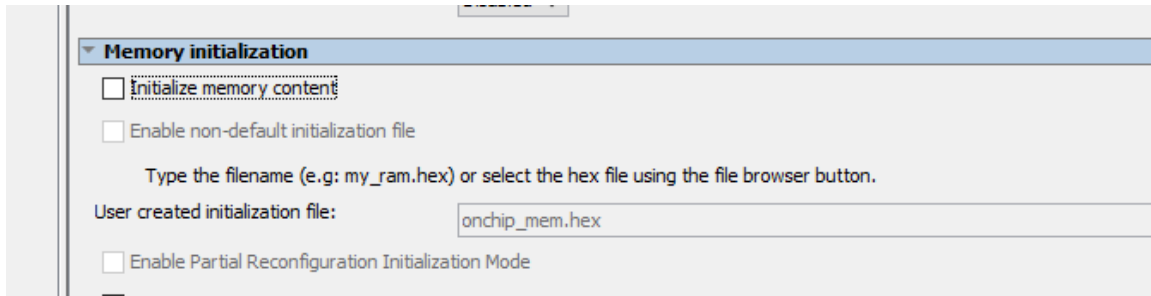
- Now let's add the RAM IP block. In the IP Catalog enter RAM in the search box.
- Double-click on On-chip Memory (RAM or ROM) in the Intel FPGA IP.



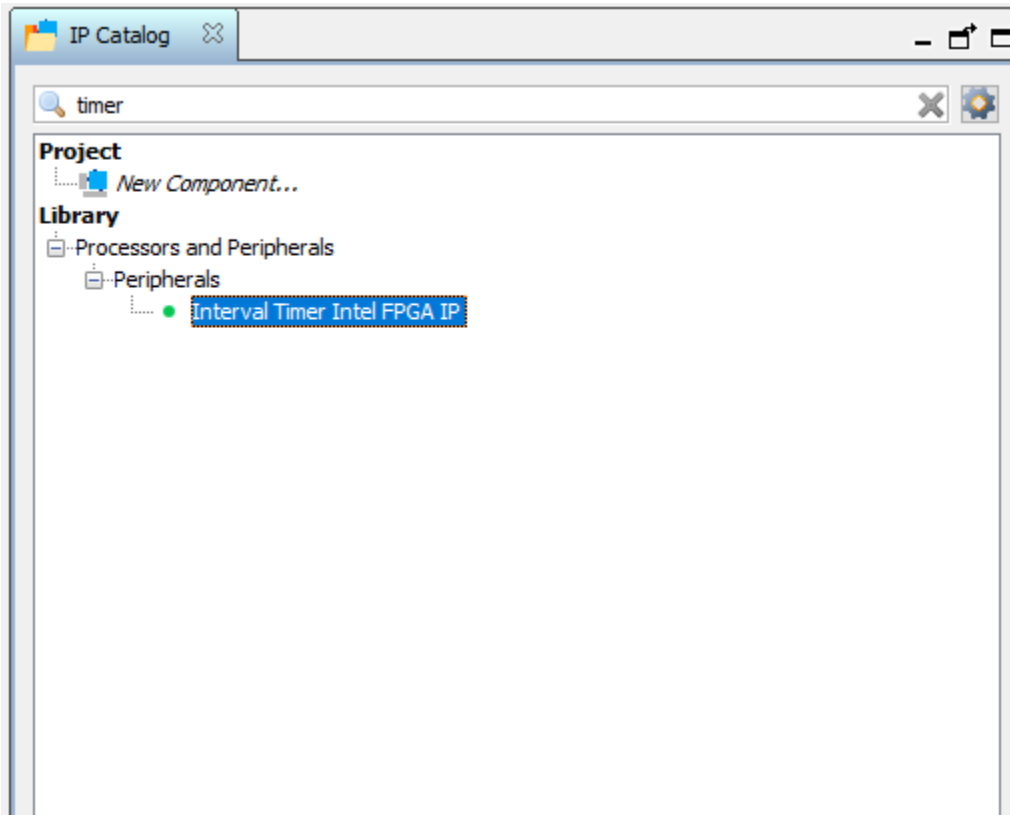
7. The configuration page will appear. Change the Total memory size to 16384. We need more memory to run the applications.



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

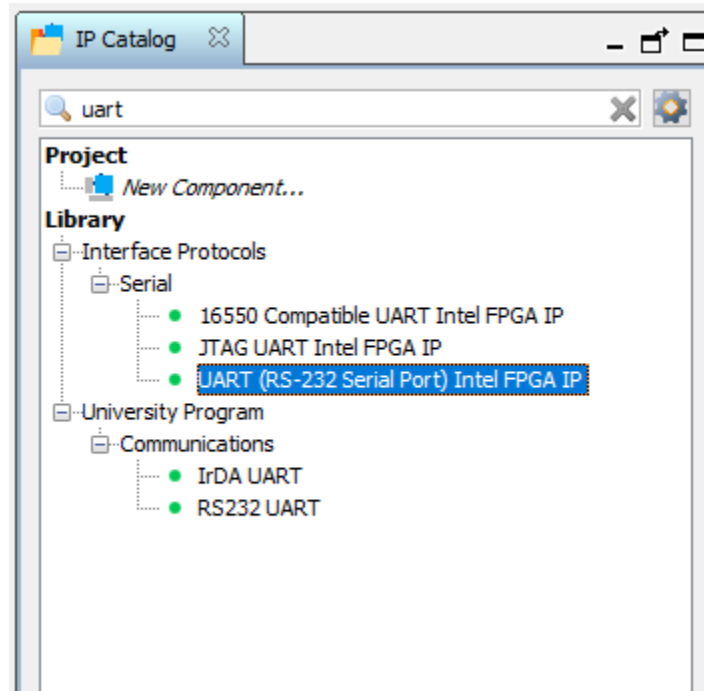


9. In the IP Catalog search, enter timer.
10. Double-click on the Interval Timer Intel FPGA IP.



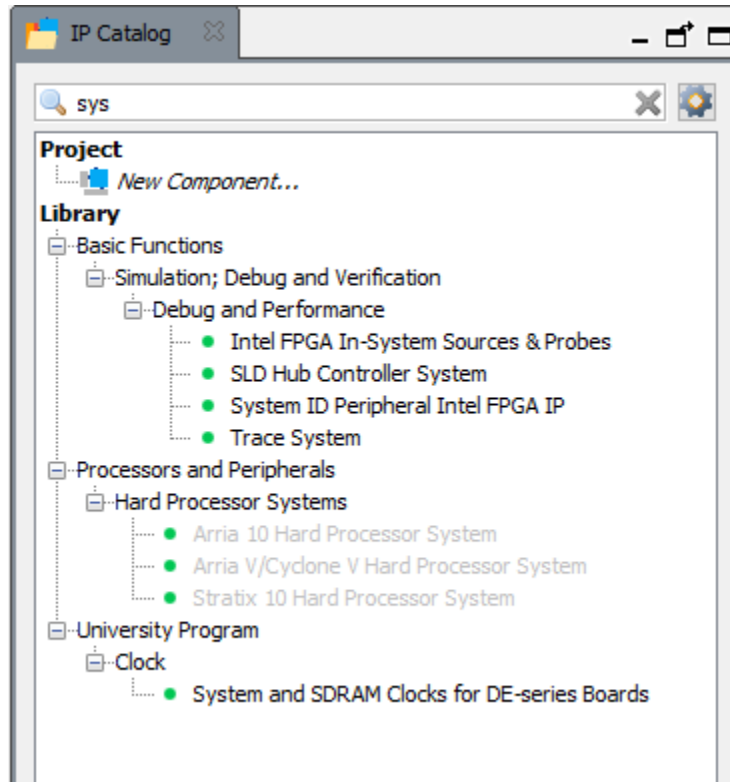
11. Keep the settings as they are and click Finish.
12. In the IP Catalog search, enter UART.
13. Double-click on the UART (RS-232 Serial Port) Intel FPGA IP16550 Compatible UART Intel FPGA IP.





You will notice that there are two Serial port IPs. 16550 Compatible UART Intel FPGA IP and UART (RS-232 Serial Port) Intel FPGA IP. The former contains support for a full FIFO and DMA transfer. The latter that we chose for this design doesn't have a buffer and is a very simplistic character reader. Since the MAX 10 10M08 Evaluation Kit board is limited to internal RAM blocks, the UART (RS-232 Serial Port) Intel FPGA IP makes practical sense to use. The challenge is with the documentation. If you check the details online for the UART (RS-232 Serial Port) Intel FPGA IP, you will see the name 16550 Compatible UART Intel FPGA IP, which is the other serial port IP, but all the information is for the UART (RS-232 Serial Port) Intel FPGA IP. There is an entire paper from Intel talking about the 16550 Compatible UART Intel FPGA IP – ID: 683130 *Embedded peripherals IP User Guide*. The dual information can get a little confusing.

14. A configuration page will appear. The default settings of 115200-8-N-1 are fine for this project. No changes need to be made. Click Finish.
15. In the IP Catalog search, enter the system ID.
16. Double-click on the System ID Peripheral Intel FPGA IP.



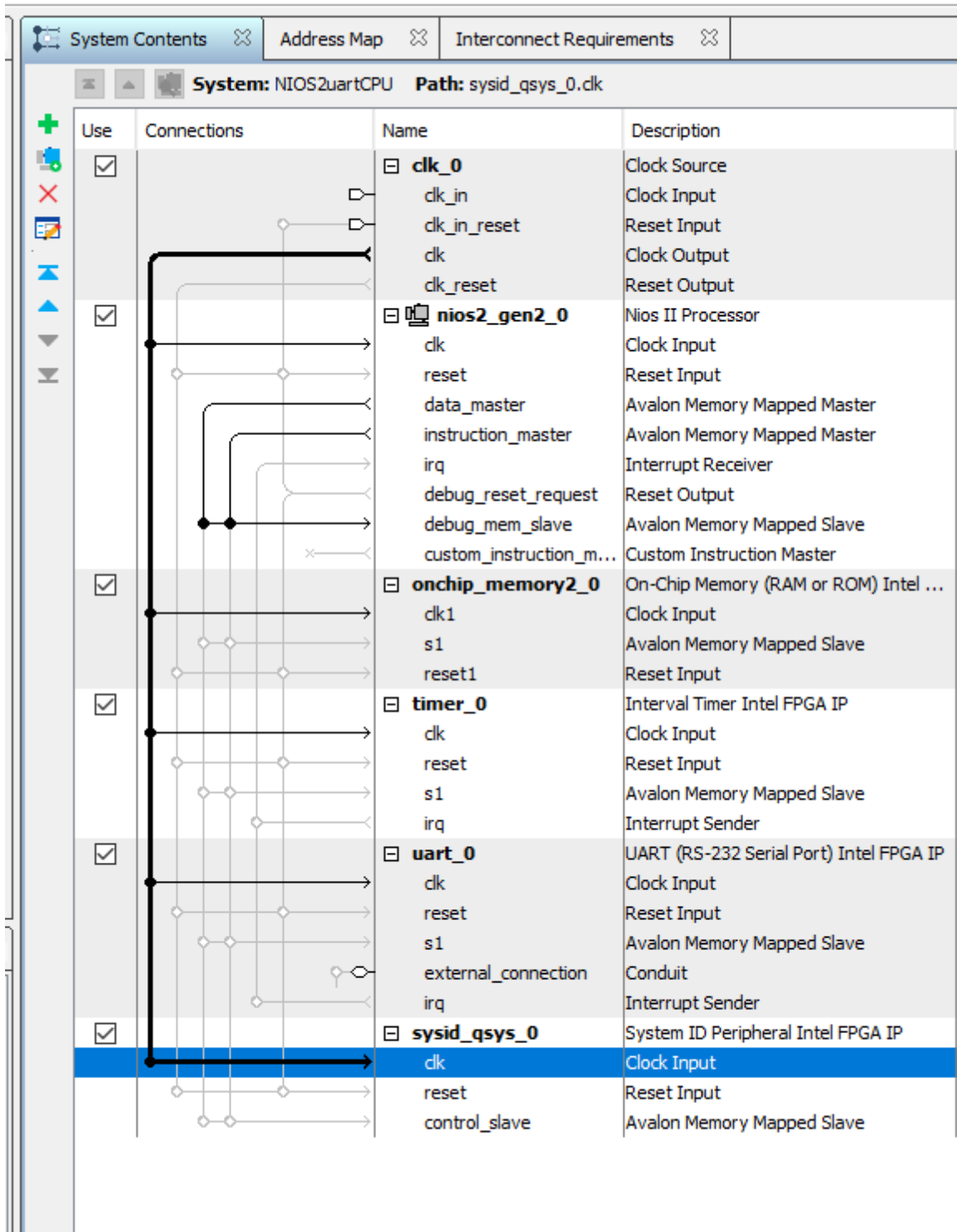
17. A configuration page will appear. No changes need to be made. Click Finish.

System Contents    Address Map    Interconnect Requirements

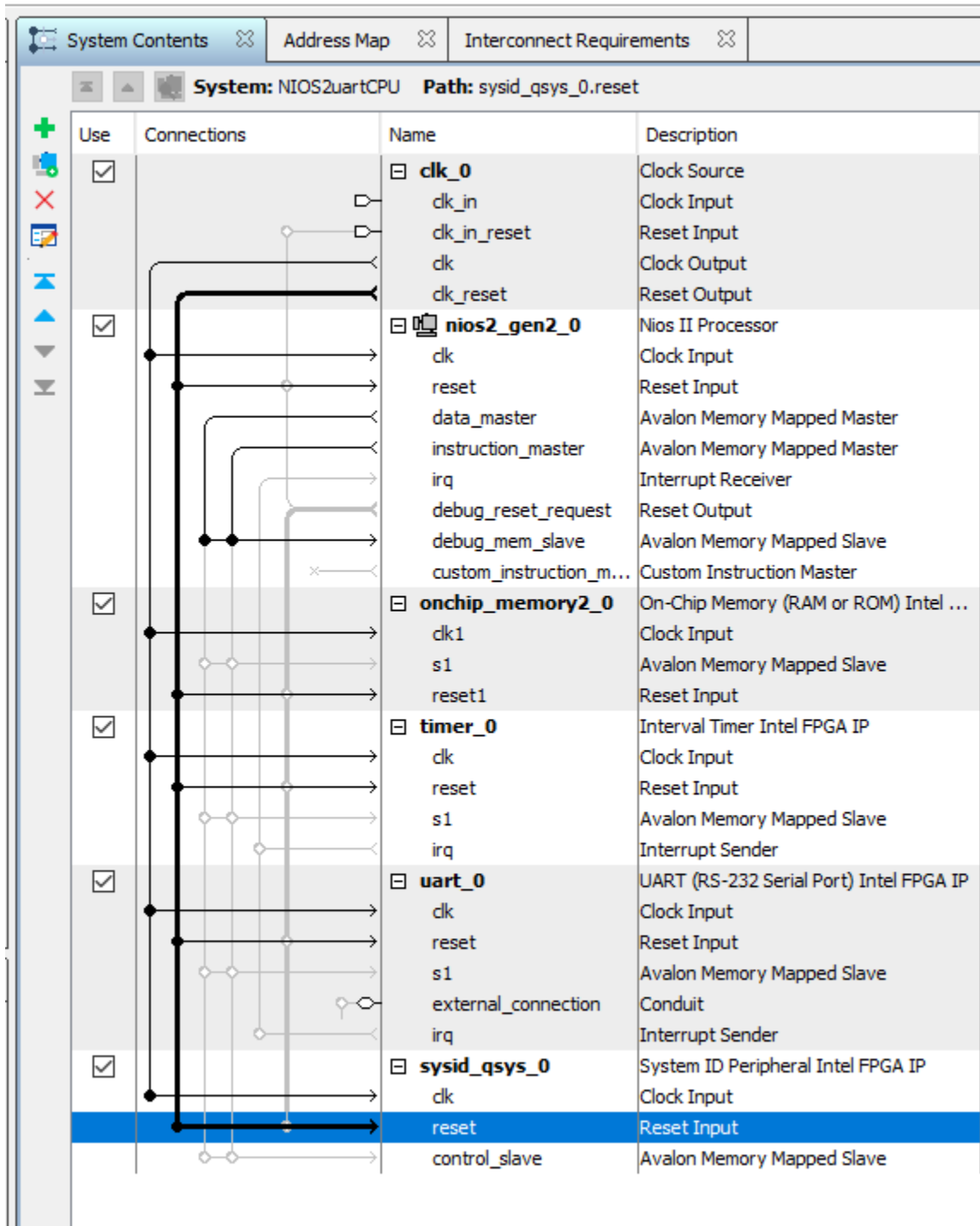
System: NIOS2uartCPU    Path: sysid\_qsys\_0

Use	Connections	Name	Description
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
<input checked="" type="checkbox"/>		<b>nios2_gen2_0</b>	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
<input checked="" type="checkbox"/>		<b>onchip_memory2_0</b>	On-Chip Memory (RAM or ROM) Intel ...
		clk1	Clock Input
		s1	Avalon Memory Mapped Slave
		reset1	Reset Input
<input checked="" type="checkbox"/>		<b>timer_0</b>	Interval Timer Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		s1	Avalon Memory Mapped Slave
		irq	Interrupt Sender
<input checked="" type="checkbox"/>		<b>uart_0</b>	UART (RS-232 Serial Port) Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		s1	Avalon Memory Mapped Slave
		external_connection	Conduit
		irq	Interrupt Sender
<input checked="" type="checkbox"/>		<b>sysid_qsys_0</b>	System ID Peripheral Intel FPGA IP
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

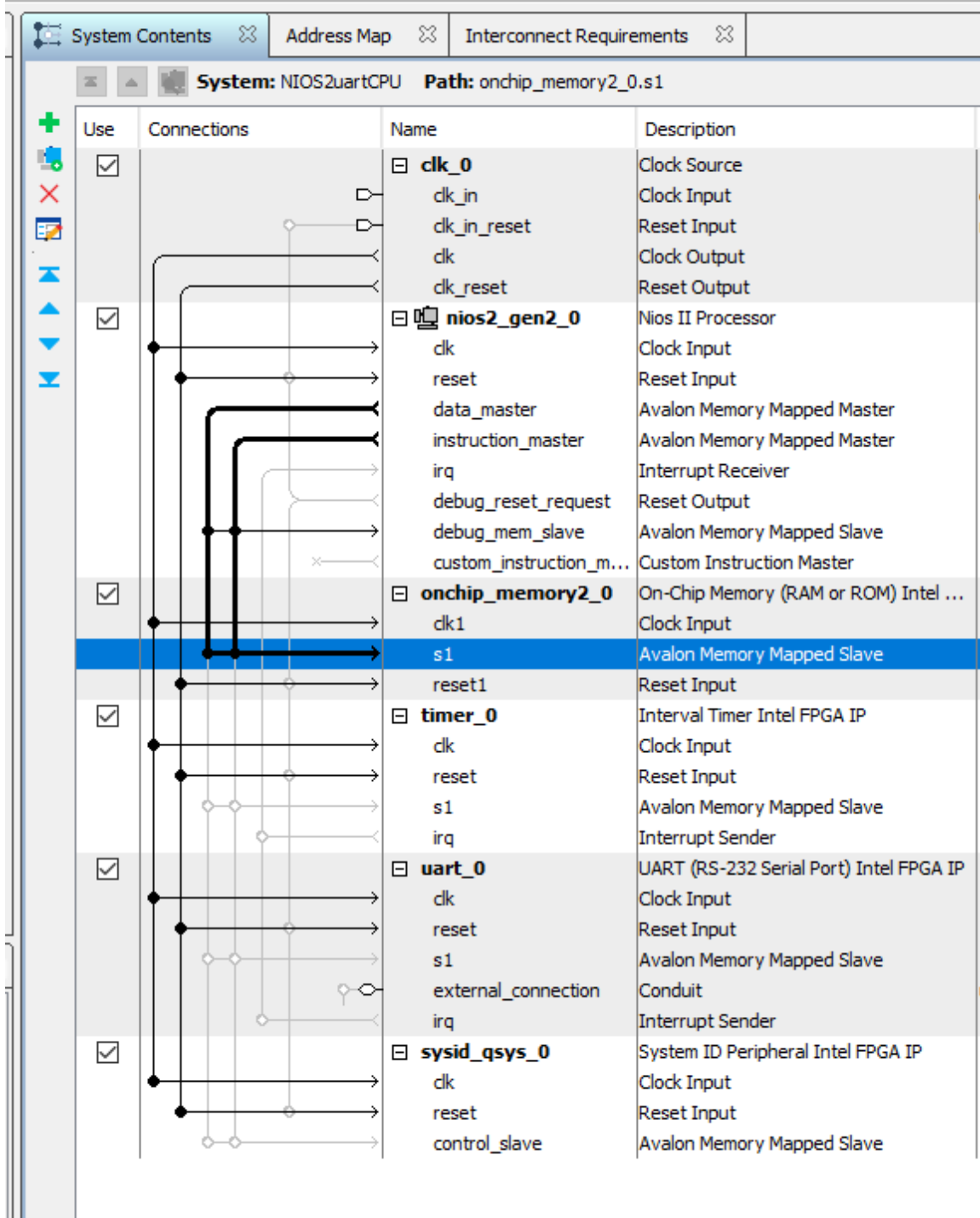
18. Now we need to wire the IP blocks together. First, wire all the clk lines together by clicking on the dots for all five IP blocks.



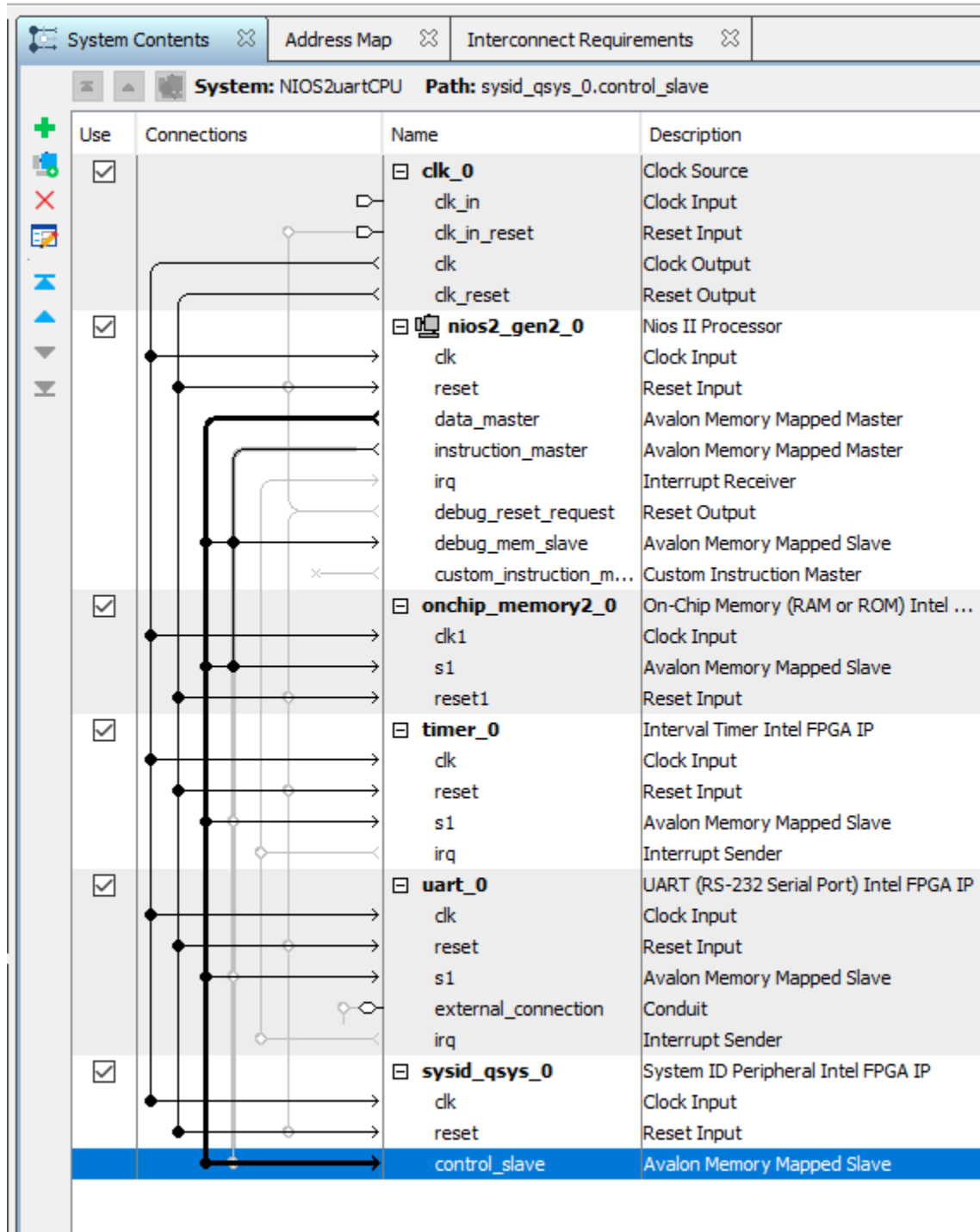
19. Next, connect all the reset lines together by clicking on the dots for all five IP blocks.



20. The memory lines have to be connected together. The s1 line in the RAM is to be connected to the nios2\_gen\_cpu data\_master and instruction master lines. Click on the dots for the RAM s1 line.



21. Connect Uart\_0's s1 line, time\_0 s1 line, and sysid\_qsys\_0's control\_slave to the nios2\_gen2\_cpu's data\_master.



22. Connect uart\_0's irq line and timer\_0's irq to nios2\_gen2\_0 irq.

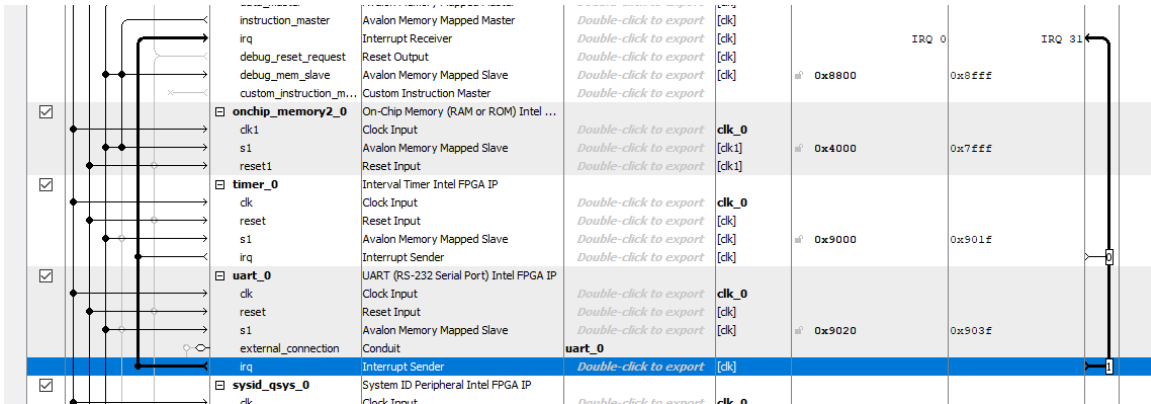
System Contents | Address Map | Interconnect Requirements

System: NIOS2uartCPU Path: uart\_0.irq

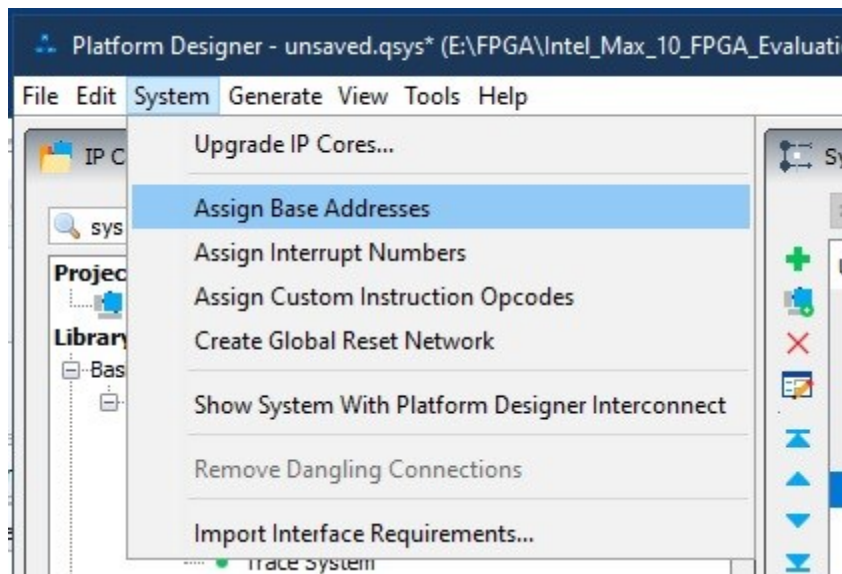
Use	Connections	Name	Description
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
<input checked="" type="checkbox"/>		<b>nios2_gen2_0</b>	Nios II Processor
		clk	Clock Input
		reset	Reset Input
		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	
<input checked="" type="checkbox"/>		<b>onchip_memory2_0</b>	On-Chip Memory (RAM or ROM) Intel ...
		clk1	Clock Input
		s1	Avalon Memory Mapped Slave
<input checked="" type="checkbox"/>		<b>timer_0</b>	Interval Timer Intel FPGA IP
		reset1	Reset Input
		clk	Clock Input
		s1	Avalon Memory Mapped Slave
<input checked="" type="checkbox"/>		<b>uart_0</b>	UART (RS-232 Serial Port) Intel FPGA IP
		irq	Interrupt Sender
		clk	Clock Input
		reset	Reset Input
		s1	Avalon Memory Mapped Slave
<input checked="" type="checkbox"/>		<b>sysid_qsys_0</b>	System ID Peripheral Intel FPGA IP
		external_connection	Conduit
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

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





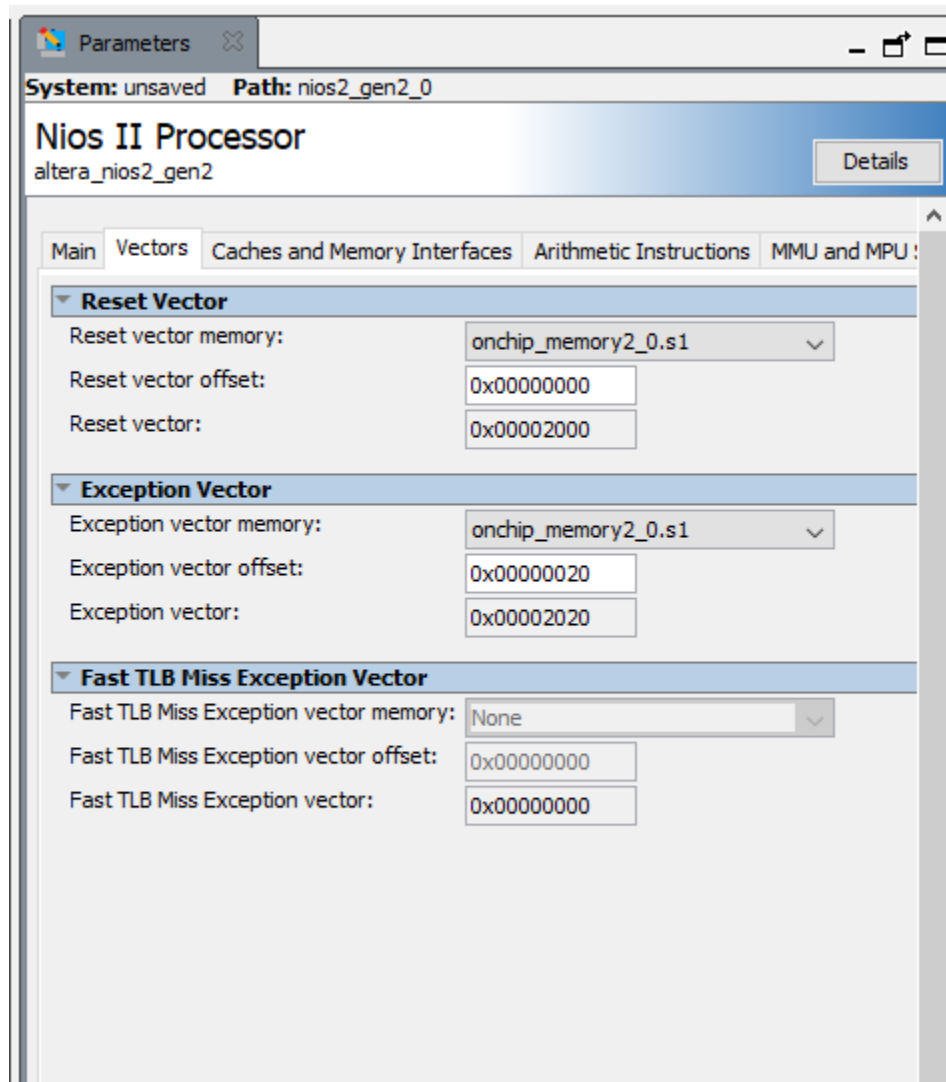
24. Let's assign a base address. From the menu, select System->Assign Base Address. This will remove a number of errors from the message box.



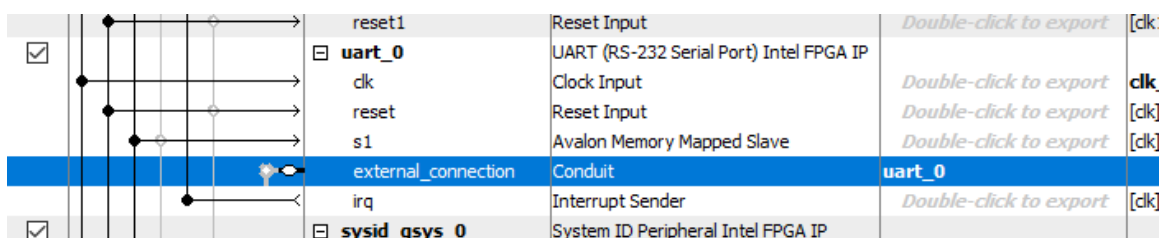
You will see the base and end address values change.

Name	Description	Export	Clock	Base	End	IRQ
<ul style="list-style-type: none"> <li>clk_0                             <ul style="list-style-type: none"> <li>clk_in</li> <li>clk_in_reset</li> <li>clk</li> <li>clk_reset</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Clock Source</li> <li>Clock Input</li> <li>Reset Input</li> <li>Clock Output</li> <li>Reset Output</li> </ul>	<ul style="list-style-type: none"> <li>clk</li> <li>reset</li> </ul>	<ul style="list-style-type: none"> <li>exported</li> </ul>			
<ul style="list-style-type: none"> <li>nios2_gen2_0                             <ul style="list-style-type: none"> <li>clk</li> <li>reset</li> <li>data_master</li> <li>instruction_master</li> <li>irq</li> <li>debug_reset_request</li> <li>debug_mem_slave</li> <li>custom_instruction_m...</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Nios II Processor</li> <li>Clock Input</li> <li>Reset Input</li> <li>Avalon Memory Mapped Master</li> <li>Avalon Memory Mapped Master</li> <li>Interrupt Receiver</li> <li>Reset Output</li> <li>Avalon Memory Mapped Slave</li> <li>Custom Instruction Master</li> </ul>	<ul style="list-style-type: none"> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> </ul>	<ul style="list-style-type: none"> <li>clk_0</li> <li>[clk]</li> <li>[clk]</li> <li>[clk]</li> <li>[clk]</li> <li>[clk]</li> <li>[clk]</li> </ul>			<ul style="list-style-type: none"> <li>IRQ 0</li> <li>IRQ 31</li> </ul>
<ul style="list-style-type: none"> <li>onchip_memory2_0                             <ul style="list-style-type: none"> <li>clk1</li> <li>s1</li> <li>reset1</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>On-Chip Memory (RAM or ROM) Intel ...</li> <li>Clock Input</li> <li>Avalon Memory Mapped Slave</li> <li>Reset Input</li> </ul>	<ul style="list-style-type: none"> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> </ul>	<ul style="list-style-type: none"> <li>clk_0</li> <li>[clk1]</li> <li>[clk1]</li> </ul>	<ul style="list-style-type: none"> <li>0x4800</li> <li>0x2000</li> </ul>	<ul style="list-style-type: none"> <li>0x4fff</li> <li>0x3fff</li> </ul>	
<ul style="list-style-type: none"> <li>uart_0                             <ul style="list-style-type: none"> <li>clk</li> <li>reset</li> <li>s1</li> <li>external_connection</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>UART (RS-232 Serial Port) Intel FPGA IP</li> <li>Clock Input</li> <li>Reset Input</li> <li>Avalon Memory Mapped Slave</li> <li>Conduit</li> </ul>	<ul style="list-style-type: none"> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> </ul>	<ul style="list-style-type: none"> <li>clk_0</li> <li>[clk]</li> <li>[clk]</li> <li>[clk]</li> </ul>	<ul style="list-style-type: none"> <li>0x5000</li> </ul>	<ul style="list-style-type: none"> <li>0x501f</li> </ul>	
<ul style="list-style-type: none"> <li>irq</li> </ul>	<ul style="list-style-type: none"> <li>Interrupt Sender</li> </ul>	<ul style="list-style-type: none"> <li>Double-click to export</li> </ul>	<ul style="list-style-type: none"> <li>[clk]</li> </ul>			
<ul style="list-style-type: none"> <li>sysid_qsys_0                             <ul style="list-style-type: none"> <li>clk</li> <li>reset</li> <li>control_slave</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>System ID Peripheral Intel FPGA IP</li> <li>Clock Input</li> <li>Reset Input</li> <li>Avalon Memory Mapped Slave</li> </ul>	<ul style="list-style-type: none"> <li>Double-click to export</li> <li>Double-click to export</li> <li>Double-click to export</li> </ul>	<ul style="list-style-type: none"> <li>clk_0</li> <li>[clk]</li> <li>[clk]</li> </ul>	<ul style="list-style-type: none"> <li>0x5020</li> </ul>	<ul style="list-style-type: none"> <li>0x5027</li> </ul>	

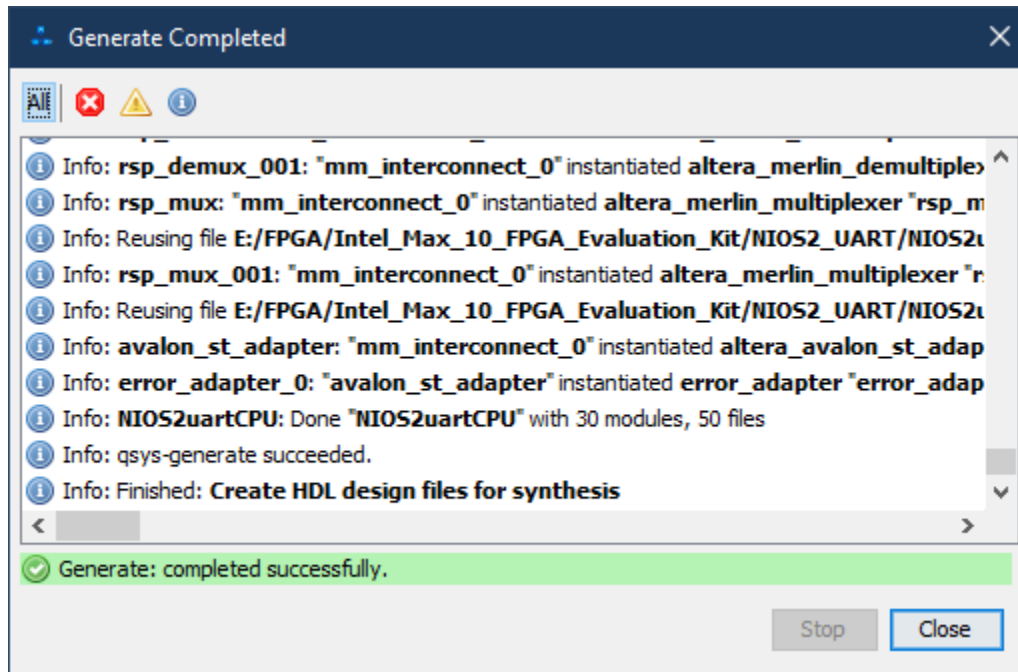
- Set the reset and exception vector addresses. Double-click on the nios2\_gen2\_cpu to open the configuration page.
- Click on the Vectors tab.
- Change the Reset vector memory drop-down to onchip\_memory2\_0.s1.
- Change the Exception vector ... memory drop-down to onchip\_memory2\_0.s1.



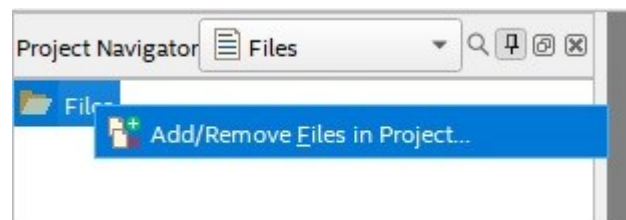
29. We need to export the RX and TX lines for uart\_0. On the external\_connection line, double-click in the Export column cell and type in uart\_0. This will show as pins on the part in the block diagram.



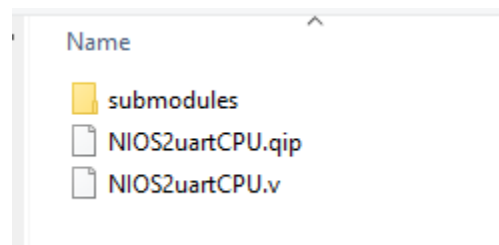
30. Click on Generate HDL...
31. Keep the defaults and click the Generate button.
32. A dialog will appear asking you to save the design, click Save.
33. Give the name as NIOS2uartCPU.qsys, and click Save.
34. Once the save has been completed, click Close.
35. The generate process kicks off. The process should succeed, click Close.



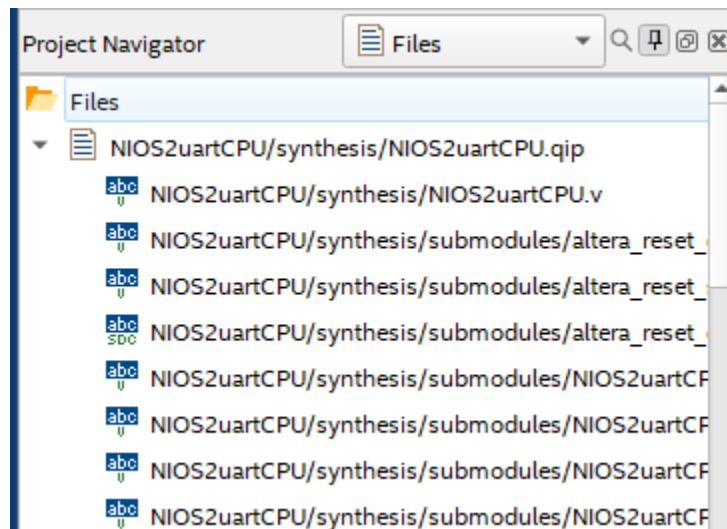
36. Click Finish to close the design.
37. Quartus then reminds you to add the new design to the project. Click Ok.
38. In the Project Navigator, click on the drop-down and select Files.
39. Right-click on Files and select Add/Remote Files in Project



40. A Settings – NIOS2uart page appears with Files on the left highlighted. Click the three dots browse button for File name, and navigate to \NIOS2\_UART\Nios2uartCPU\synthesis folder.
41. Click on NIOS2uartCPU.qip file and click open.




42. Click OK to close the Settings- NIOS2uart page. The qip file is added to the Project navigator list. Underneath are all the Verilog files that were generated by Platform Builder.

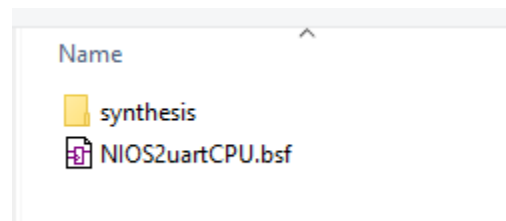


### 1.1.3 Create the Design Step 2: Block Diagram

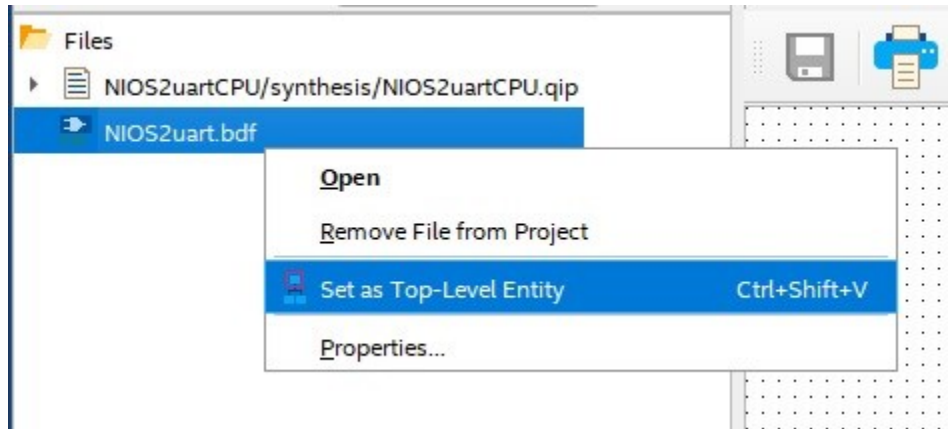
Technically, you don't have to create a Block Diagram, but we are going to add external logic to the CPU block in the next exercise. With the qip file and all the Verilog files added to the project, let's create the block diagram and complete the design.



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 \NIOS2\_UART\Niops2uartCPU folder and open the NIOS2uartCPU.bsf file.



5. The symbol for the nios2uart appears. Click OK to add the symbol to the schematic.
6. Drag the nios2uart symbol with the mouse to a location on the diagram and then left-click to drop it in place.
7. Right-click on the nios2uart 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 logic 0 when pressed. Leave the UART pin names as they are.
10. Save the schematic as NIOS2uart.bdf.
11. In the Project Navigator go to Files, right-click on NIOS2\_UART.bdf, and click on the Set as Top-Level Entity.
12. Save project.



13. 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 eventually succeed. This step helps to diagnose any errors and finds the Node Names for the pin assignments in the next step.

14. 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.

15. Using the board schematic, locate the pins for the SW1 and the 50MHz clock. For the uart\_0 lines, we want to go to one of the 2x20 header connector pins. For this example, set uart\_0\_rxd to PIN\_57, which is header J8-8, and set uart\_0\_txd to PIN\_58, which is header J8-6. Set the Location values for all 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
uart_0_rxd	PIN_57
uart_0_txd	PIN_58


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

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
altera_reserved_tck	Input	PIN_18	1B	B1_NO	PIN_18	2.5 V Sc... Trigger
altera_reserved_tdi	Input	PIN_19	1B	B1_NO	PIN_19	2.5 V Sc... Trigger
altera_reserved_tdo	Output	PIN_20	1B	B1_NO	PIN_20	2.5 V
altera_reserved_tms	Input	PIN_16	1B	B1_NO	PIN_16	2.5 V Sc... Trigger
clk_50MHz	Input	PIN_27	2	B2_NO	PIN_27	3.3-V LVTTTL
SW1	Input	PIN_121	8	B8_NO	PIN_121	3.3-V LVTTTL
uart_0_rxd	Input	PIN_57	3	B3_NO	PIN_57	3.3-V LVTTTL
uart_0_txd	Output	PIN_58	3	B3_NO	PIN_58	3.3-V LVTTTL
<<new node>>						

17. Close the Pin Planner when finished.
18. Save the project.

**Note:** Quartus can crash unexpectedly, which may be due to the fact that it was written in Java and is not a native Windows application based on .NET. Therefore, a best practice at this point is to make a backup of the project folder.

19. 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.

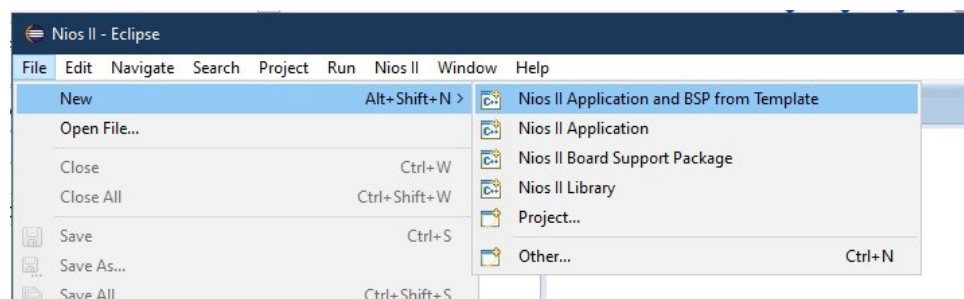
The compilation should complete successfully.

Flow Summary	
<input type="text" value="Filter"/>	
Flow Status	Successful - Fri Jul 8 21:27:36 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	NIOS2uart
Top-level Entity Name	NIOS2uart
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	3,327 / 8,064 ( 41 % )
Total registers	1988
Total pins	4 / 101 ( 4 % )
Total virtual pins	0
Total memory bits	193,216 / 387,072 ( 50 % )
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 Write the uartTerminal Application in Eclipse

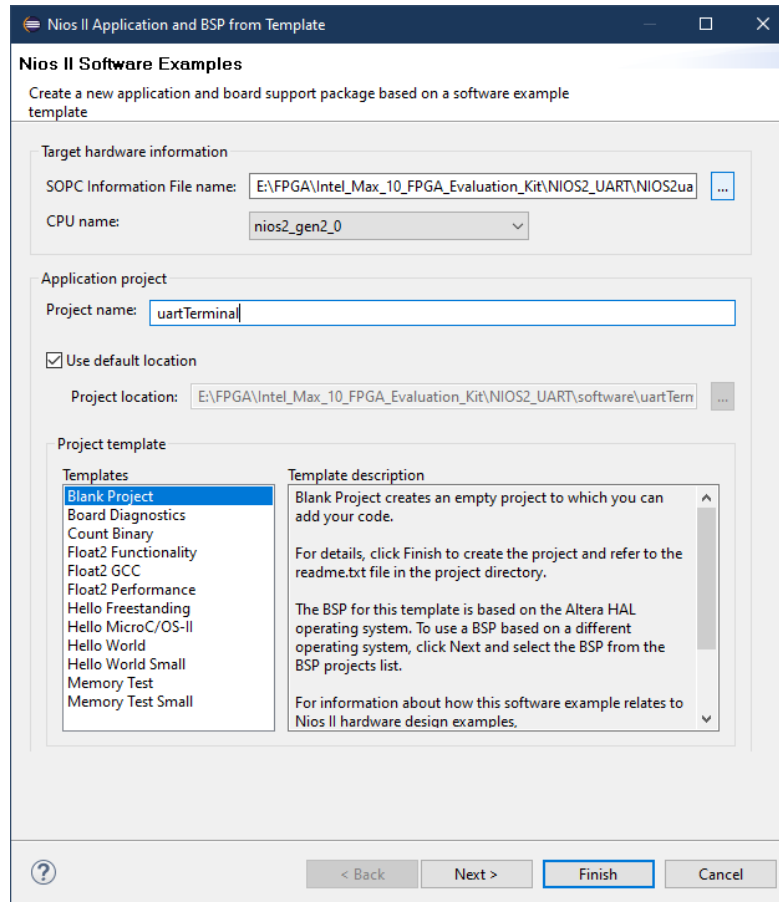
Now, we are ready to create an application to run on the Nios II processor. The application will read a string from a remote terminal and echo the string back to the remote terminal. The Nios II HAL APIs and BSP drivers will be used to interact with the Nios CPU and UART.

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 what the location of the workspace is, 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.



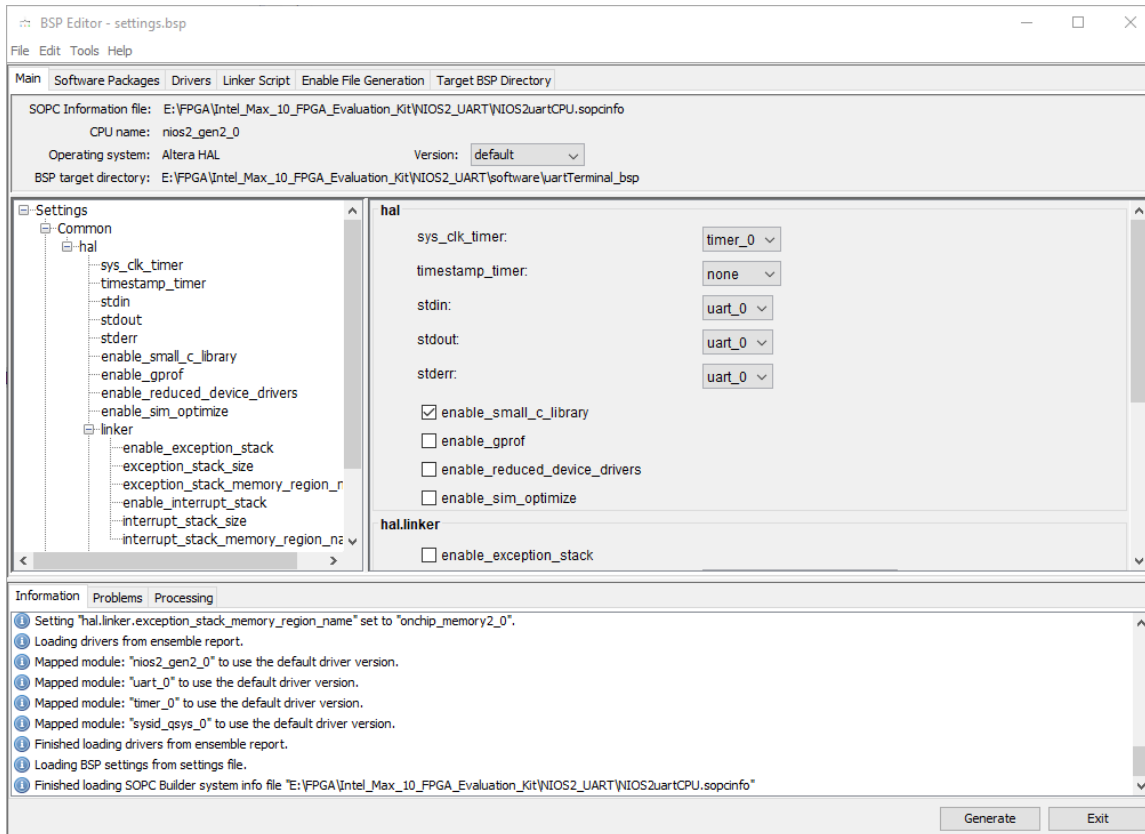


4. The first step is to open the SOPC file that was generated for the hardware design. Click on the three dots button.
5. Navigate to the \NIOS2\_UART folder and open the NIOS2uartCPU.sopcinfo file. The CPU name will reflect the name we gave the CPU in Platform Builder.
6. Enter the project name: uartTerminal.
7. In the Project Template, select Blank Project.
8. Click Finish.



Two projects will be generated. The `uartTerminal_bsp` is generated to give you the HAL drivers and API based on the hardware design. The `uartTerminal` 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 `uartTerminal_bsp` and select Nios II->BSP Editor from the context menu.
10. The BSP Editor opens and opens the `settings.bsp` file automatically. If you were to have started the BSP Editor from the main menu, you would have to manually navigate to open the file. You can see this is where the `small_c` library and reduced drivers are set. The standard input, output, and error ports to send messages to are already set to `uart_0`. Tick the box for `enable_small_c_library`, and click Generate to make the changes.



11. Click Exit when finished.

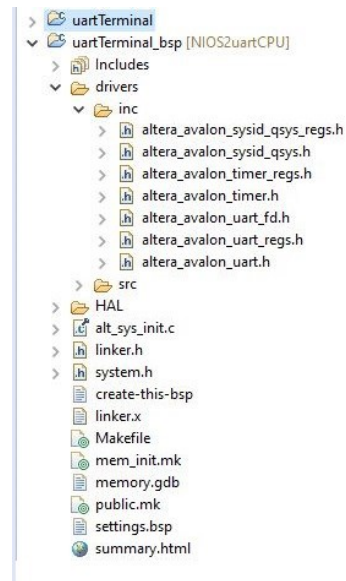
The `uartTerminal_bsp` contains the key files that will be helpful with filling in the code to access the serial port. `System.h` contains the definitions that were set up for the UART in Platform Designer. From the picture below, you can see the communication settings 115200-8-N-1, the memory base address of 0x5000, and the IRQ number.

```

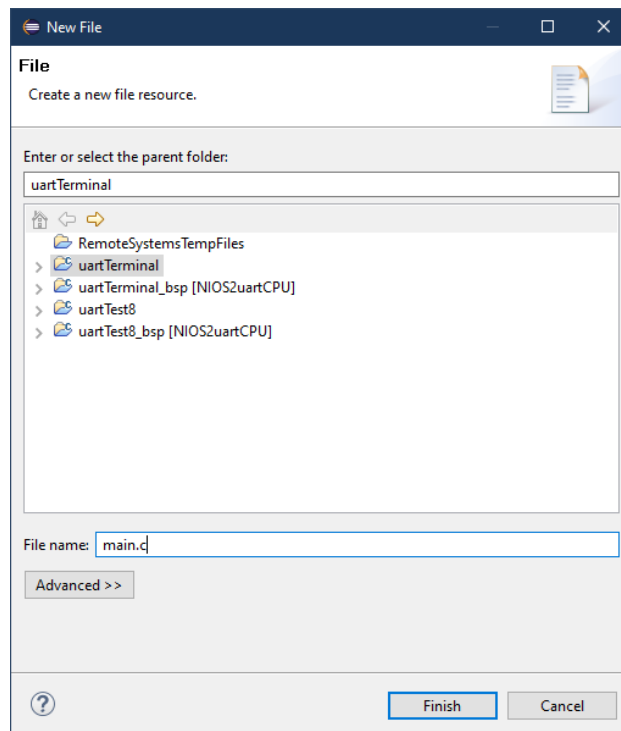
268
269 /*
270  * uart_0 configuration
271  *
272  */
273
274 #define ALT_MODULE_CLASS_uart_0 altera_avalon_uart
275 #define UART_0_BASE 0x9000
276 #define UART_0_BAUD 115200
277 #define UART_0_DATA_BITS 8
278 #define UART_0_FIXED_BAUD 1
279 #define UART_0_FREQ 50000000
280 #define UART_0_IRQ 1
281 #define UART_0_IRQ_INTERRUPT_CONTROLLER_ID 0
282 #define UART_0_NAME "/dev/uart_0"
283 #define UART_0_PARITY 'N'
284 #define UART_0_SIM_CHAR_STREAM ""
285 #define UART_0_SIM_TRUE_BAUD 0
286 #define UART_0_SPAN 32
287 #define UART_0_STOP_BITS 1
288 #define UART_0_SYNC_REG_DEPTH 2
289 #define UART_0_TYPE "altera_avalon_uart"
290 #define UART_0_USE_CTS_RTS 0
291 #define UART_0_USE_EOP_REGISTER 0
292
293 #endif /* __SYSTEM_H */
294

```

Since we are using the small\_C\_library to conserve storage space, the standard C io calls cannot be used. Instead, we will be using the Nios II HAL APIs to access the UART. The other header files are under the drivers\inc folder altera\_avalon\_uart\_\*.h. Each file contains the function prototypes of the commands that will be used in the application.



12. We need to add a main.c file to the project. Right-click on the uartTerminal project, and select New->File from the context menu.
13. Enter the file name main.c and click Finish.



14. Add the following code to the main.c file:

```

1.  #include "sys/alt_stdio.h"
2.  #include "system.h"
3.  #include "altera_avalon_uart.h"
4.  #include "altera_avalon_uart_regs.h"
5.  #include "altera_avalon_uart_fd.h"
6.  #include "priv/alt_busy_sleep.h"
7.  #include "sys/alt_sys_wrappers.h"
8.
9.  int main()
10. {
11.     alt_putstr("Hello from Nios II!\n");
12.
13.     char rx_buffer[1];
14.     char rx_bufferString[256];
15.     int i=0;
16.     int x = 0;
17.
18.     int fp_uart_0;
19.     fp_uart_0 = ALT_OPEN(UART_0_NAME,2);
20.
21.     alt_putstr("Enter a string and hit enter\n");
22.     alt_putstr("*****\n");
23.     alt_putstr("\n");
24.
25.     //Clear buffer
26.     for(x = 0; x <= 255; x++){
27.         rx_bufferString[x] = 0;
28.     }
29.
30.
31.     /* Event loop never exits. */
32.     while (1){
33.
34.         do{
35.             ALT_READ(fp_uart_0,rx_buffer, sizeof(rx_buffer));
36.             rx_bufferString[i] = rx_buffer[0];
37.             i++;
38.         }while(rx_buffer[0] != '\n' | (i<=255));
39.
40.         i = 0;
41.
42.         alt_putstr("You entered\n");
43.         ALT_WRITE(fp_uart_0,rx_bufferString, sizeof(rx_bufferString));
44.         alt_putstr("\n");
45.         alt_putstr("Enter a string and hit enter\n");
46.         alt_putstr("*****\n");
47.         alt_putstr("\n");
48.
49.         //Clear buffer
50.         for(x = 0; x <= 255; x++){
51.             rx_bufferString[x] = 0;
52.         }
53.     }
54.
55.     return 0;
56. }

```

Programming access to the HAL drivers is provided by the HAL API Wrappers. The various driver header files contain the wrapper APIs such as ALT\_OPEN, ALT\_WRITE, and ALT\_READ that are used to access the serial port.

## Application

## Nios II HAL API Wrappers

## Nios II HAL Drivers

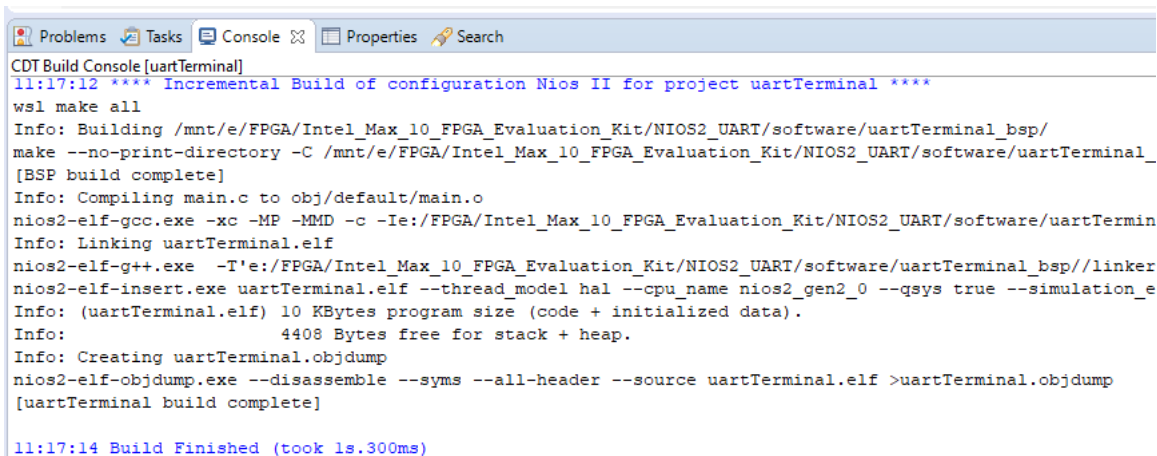
Lines 1-7 define the header files to use from the `uartTerminal_bsp` project. The make file defines the path access to the `uartTerminal_bsp` project. For some header files, you have to add a deeper path folder. The `alt_stdio.h` is the small C replacement for the `stdio.h` library, which is too big to fit in the memory the hardware provides. A bigger MAX 10 chip with more RAM blocks would make it feasible, but the MAX 10 10M08 Evaluation Kit is basic.

Line 13 makes a call to send a string out the standard I/O port, which will be `UART_0`. Lines 13-16 define the variables and the `rx_buffer` arrays to store data. Lines 18-19 open the `UART_0` for read/write access (flag = 2). The `ALT_OPEN` command returns an integer to the `UART_0` device structure.

Lines 21-28 send a message string to the standard I/O port and clear the big receive buffer. The main part of the program is in Lines 32-53. The HAL UART driver is a single-character input driver. The application stops at the `ALT_READ` command and waits to receive a character before continuing. There is no need to set up an interrupt handler, but you could to handle the receive interrupts differently. For this project, we will use the driver as is. Only 16 bits of data are read in at a time. The do-while loop will store each character received and store it in a larger buffer. When the line feed character (`\n`) is received or the buffer is full (`i >=255`) it ends the loop.

Once the line feed character has been reached, the do-while loop is exited, and the rest of the program runs to complete the while-loop. The message is then transmitted back out using the `ALT_WRITE` command, another message is sent to the user, and the `re_bufferString` is cleared.

15. Save the file.
16. Right-click on `uartTerminal` project again, and select Build Project. The build should complete successfully, and the `uartTerminal.elf` file has been created.



```

CDT Build Console [uartTerminal]
11:17:12 **** Incremental Build of configuration Nios II for project uartTerminal ****
wsl make all
Info: Building /mnt/e/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_UART/software/uartTerminal_bsp/
make --no-print-directory -C /mnt/e/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_UART/software/uartTerminal_
[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_UART/software/uartTermin
Info: Linking uartTerminal.elf
nios2-elf-g++.exe -T'e:/FPGA/Intel_Max_10_FPGA_Evaluation_Kit/NIOS2_UART/software/uartTerminal_bsp//linker
nios2-elf-insert.exe uartTerminal.elf --thread_model hal --cpu_name nios2_gen2_0 --qsys true --simulation_e
Info: (uartTerminal.elf) 10 KBytes program size (code + initialized data).
Info:          4408 Bytes free for stack + heap.
Info: Creating uartTerminal.objdump
nios2-elf-objdump.exe --disassemble --syms --all-header --source uartTerminal.elf >uartTerminal.objdump
[uartTerminal build complete]

11:17:14 Build Finished (took 1s.300ms)

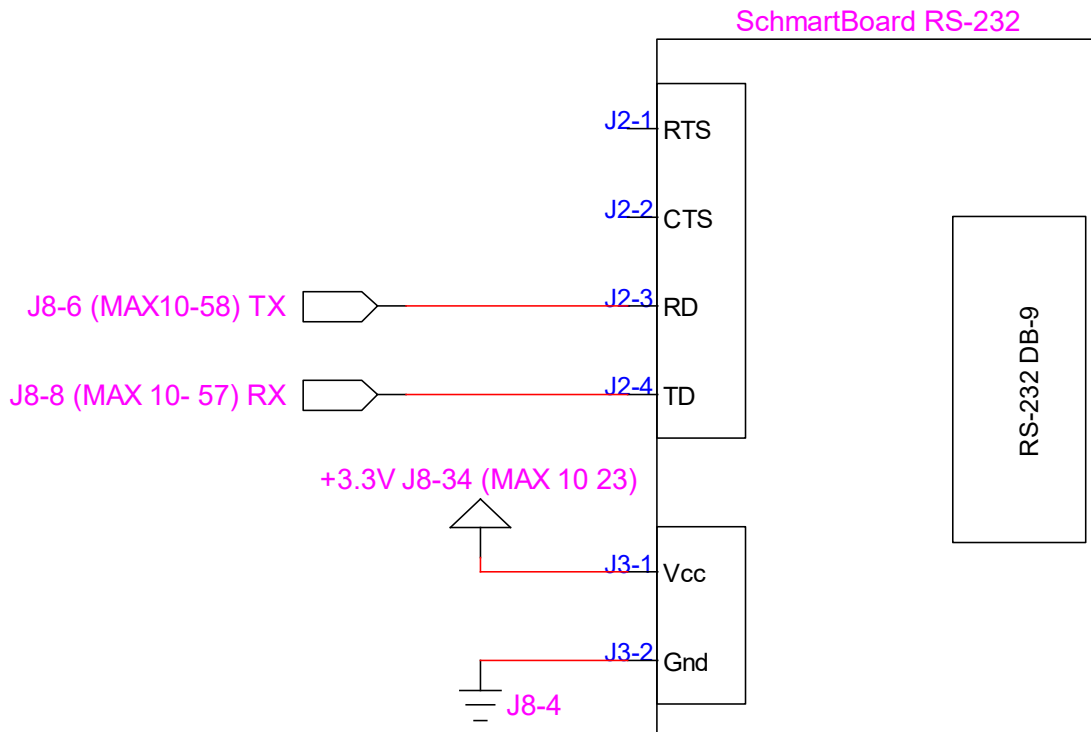
```

17. Close Eclipse

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

### 1.1.5 Circuit Setup

The TX and RX from the board need to be connected to a chip or converted from TTL or CMOS to RS-232 signals. The Schmartboard RS-232 is one solution. For this particular board, connect the RD pin to the designated TX pin (J8-8) on the evaluation kit and TD to the RX pin (J8-6) of the evaluation kit. Connect a null-modem cable to the RS-232 connection and the other end to a computer's RS-232 port or USB-to-RS-232 connector.



### 1.1.6 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 together per the cable instructions.

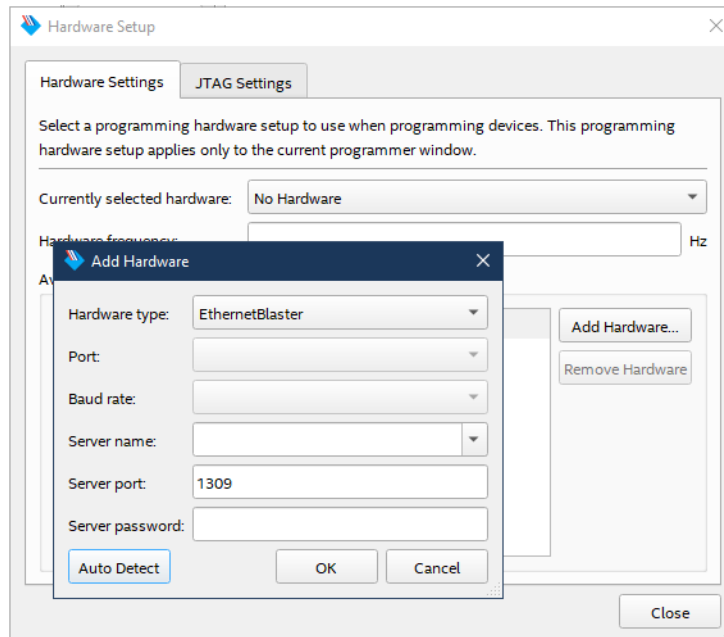
**Note:** The MAX 10 – 10M08 Evaluation Kit doesn't come with a programming cable or built-in JTAG USB Blaster II. You will have to use either the USB Blaster II or EthernetBlaster II external cables. The EthernetBlaster II was used for this example. DHCP setup was not working so a direct Ethernet cable connection was made between a PC and the EthernetBlaster II. The static IP was set for the PC network card to 198.162.0.1. The EthernetBlaster II was accessed via a browser and then the IP address was changed to a static IP that matched the network. The new IP address was used as the Server name.

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)

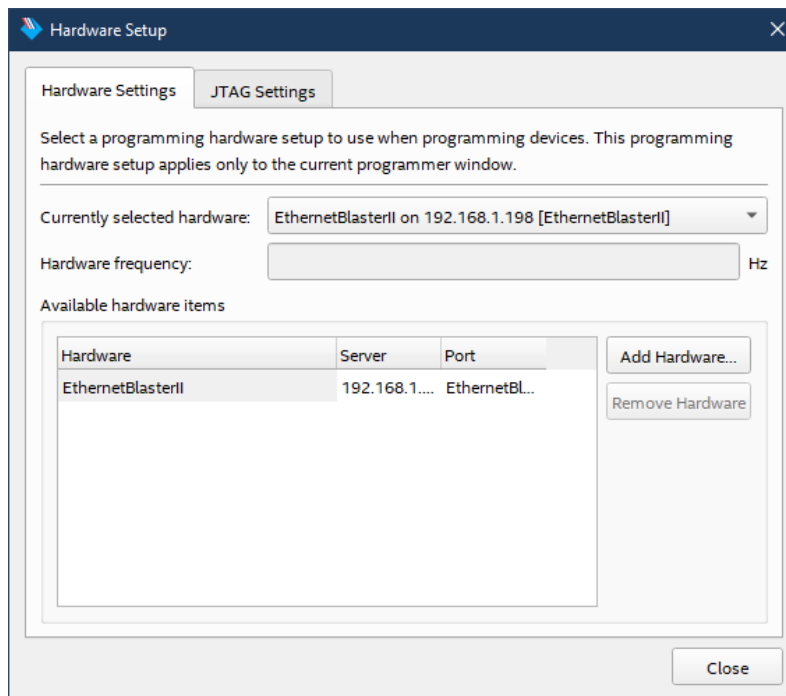


and select Open from the context menu or click on the icon on the toolbar.

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.



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



7. A NIOS2uart\_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

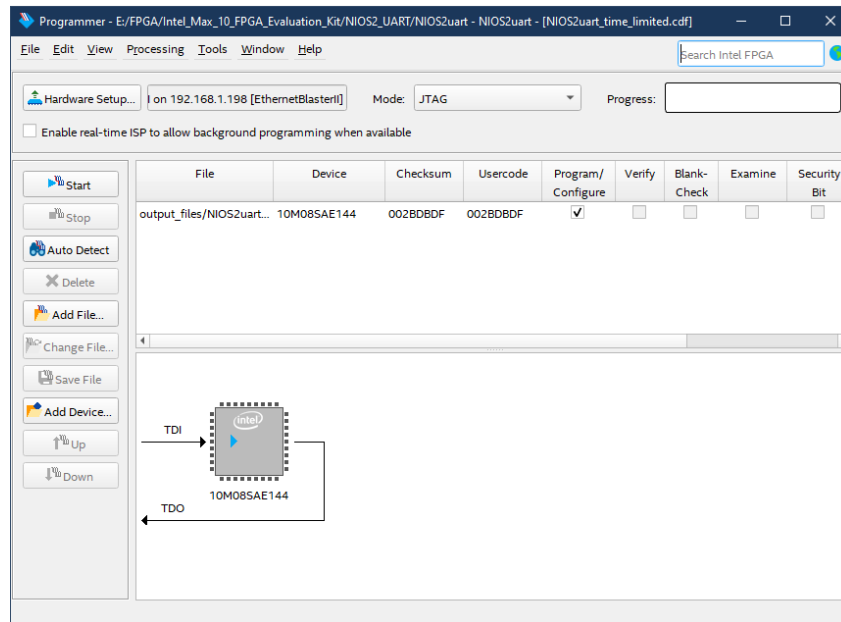
Copyright © 2022 Annabooks, LLC. All rights reserved

Intel, Quartus, Nios II, and MAX 10 are registered trademarks of Intel Corporation

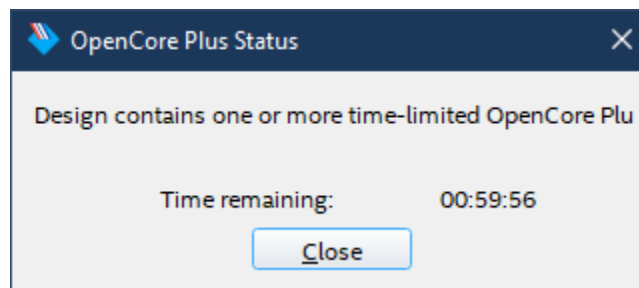
All other copyrighted, registered, and trademarked material remains the property of the respective owners.

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 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.



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



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

### 1.1.7 Deploy the Application in Eclipse

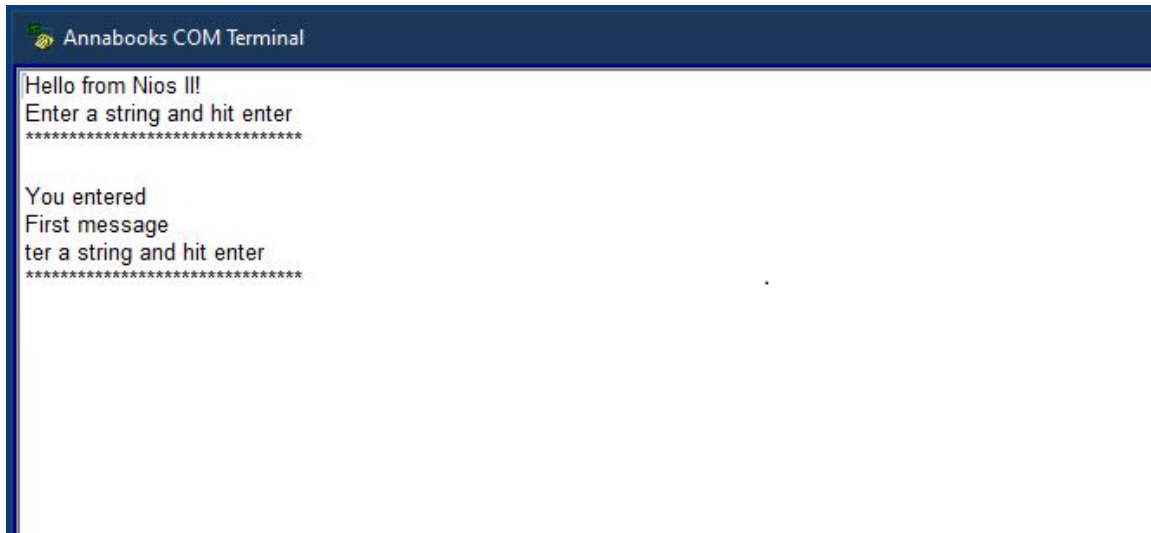
With the design loaded and the connection to JTAG up and running, we can test the application.

1. Open a serial terminal application.
2. Set the serial connection to 115200-8-N-1 and do a connect or call that enables the serial port communication.
3. From the Quartus menu, select Tools-> Nios II Software Build Tools for Eclipse.
4. Open Main.c, and set a breakpoint at line 11 just inside main().
5. Right-click on uartTerminal and select Debug As->Nios II Hardware.

Copyright © 2022 Annabooks, LLC. All rights reserved  
Intel, Quartus, Nios II, and MAX 10 are registered trademarks of Intel Corporation  
All other copyrighted, registered, and trademarked material remains the property of the respective owners.



6. The program will load and start running.
7. Click F6 or hit the step-over button from the toolbar, and then click F8 or the resume button.
8. Enter a string in the terminal and hit enter. You should see the string echoed back. You can always go back and re-run the debugger to watch the local variables.



```
Annabooks COM Terminal
Hello from Nios II!
Enter a string and hit enter
*****

You entered
First message
ter a string and hit enter
*****
```

9. Send a few more messages, and then stop debugging.
10. Run the application as a release build. Right-click on uartTerminal and select Run As->Nios II Hardware.
11. When finished close Eclipse, the OpenCore Plus dialog, and JTAG programming application.

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

## 1.2 Nios II with DUAL UART and LEDs Project

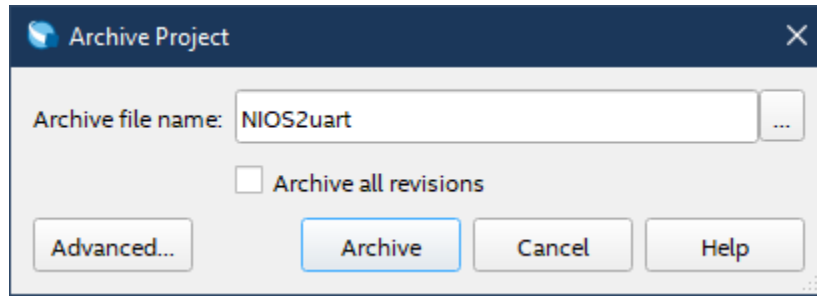
We will add to the last project to demonstrate how to updating the design requires an update to the BSP. Another UART port will be added to display messages from the application to a SparkFun 16 x 2 matrix Serial LCD, and PIO will be added for accessing the 5 LEDs on the board. The application will be modified to accept a numeric value (0-31) sent by the user from the terminal to light up the LEDs. To make things more interesting, we will implement some external logic on the block diagram to make the LEDs light up differently.

**Note:** The following exercise is based on an older Serial LCD model (4/15/2015) that has a different 3.3v powered controller, so the command set will be different from the latest SparkFun LCD products.

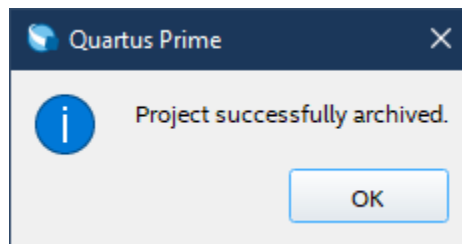
### 1.2.1 Archive the Current Design

It is always a good idea to archive or back up a project.

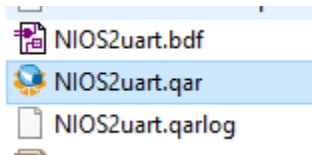
1. Open Quartus.
2. Open the NIOS2uart project.
3. From the menu, select Project->Archive.



4. A dialog appears that allows you to change the name and location. Advanced options let you select what is to be archived. Click Archive.



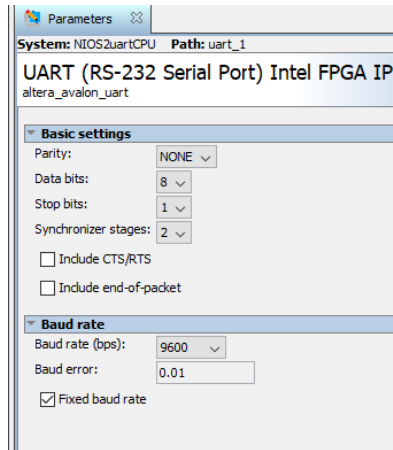
The archive will be saved as a .qar file.



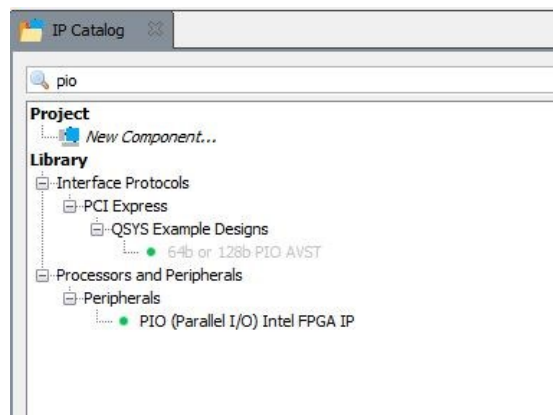
### 1.2.2 Modify the Hardware Design

Now, let's modify the design.

1. Open Quartus.
2. Open the NIOS2uart project.
3. Open Platform Designer.
4. Open the NIOS2uartCPU.qsys file.
5. In the IP Catalog, enter UART in the search box.
6. Add the UART (RS-232 Serial Port) Intel FPGA IP to the design.
7. Set the baud rate to 9600, which is the speed of the Sparkfun LCD.



8. Click Finish. The new UART will be `uart_1`.
9. In the IP Catalog enter PIO in the search box.
10. Add the PIO (Parallel I/O) Intel FPGA IP to the design.



11. In the configuration page, set Width to 5, leave the Direction as Output, and set the Output Port Reset Value to 0x3.
12. Click Finish.

The screenshot shows the configuration window for the PIO (Parallel I/O) Intel FPGA IP. The title bar indicates the project is "PIO (Parallel I/O) Intel FPGA IP - pio\_0". The main window is titled "PIO (Parallel I/O) Intel FPGA IP" and "altera\_avalon\_pio".

**Block Diagram:** A block diagram shows the "pio\_0" module. On the left, there are four input signals: "clk", "reset", "s1", and "external\_connection". On the right, there are four output signals: "clock", "reset", "avalon", and "conduit". The module is labeled "altera\_avalon\_pio". A checkbox "Show signals" is present.

**Basic Settings:**

- Width (1-32 bits): 5
- Direction:  Bidir,  Input,  InOut,  Output
- Output Port Reset Value: 0x0000000000000003

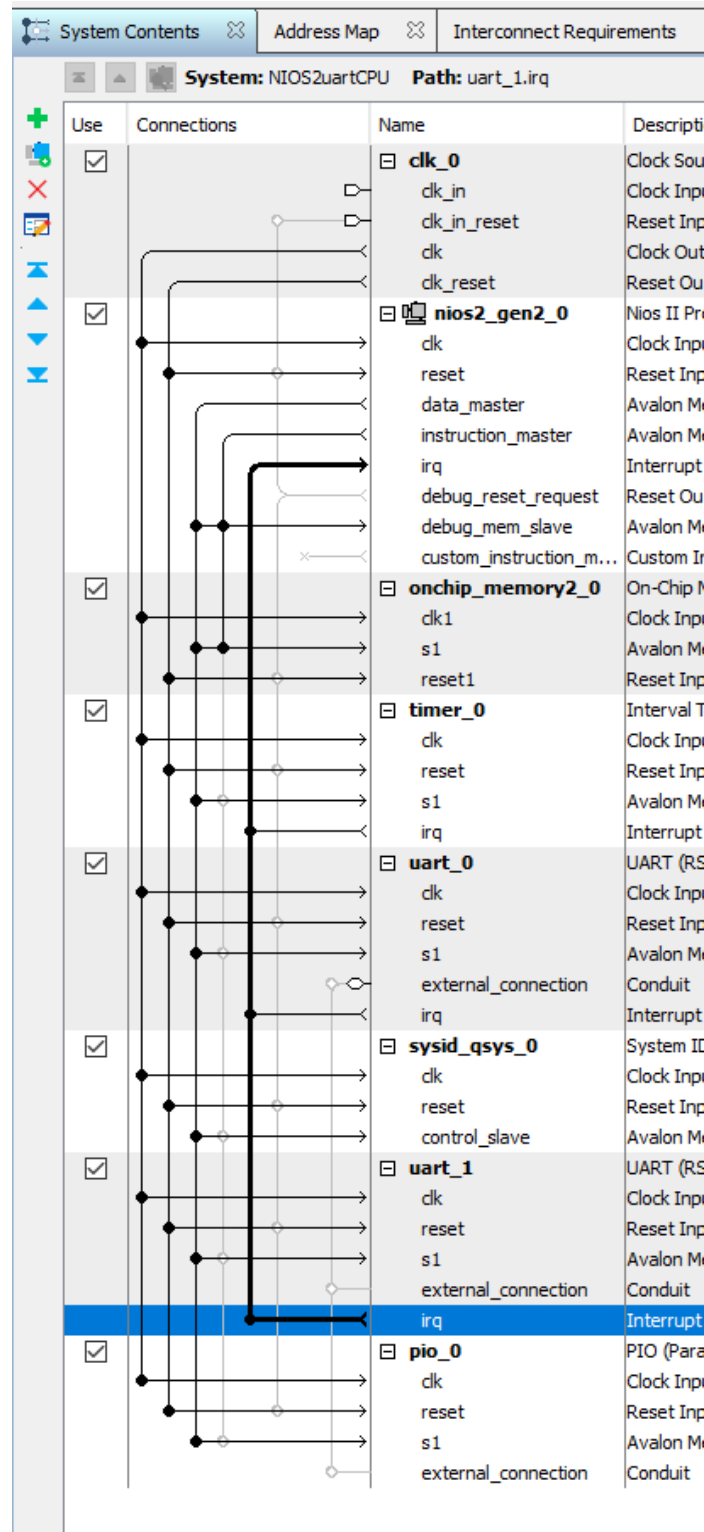
**Output Register:**

- Enable individual bit setting/clearing

**Edge capture register:**

- Synchronously capture
- Edge Type: RISING
- Enable bit-clearing for edge capture register

13. Connect the wires for the new modules: clk, reset, s1 to data\_master of nios2\_gen2\_0. Connect uart\_1's irq to nios\_gen2\_0's irq.

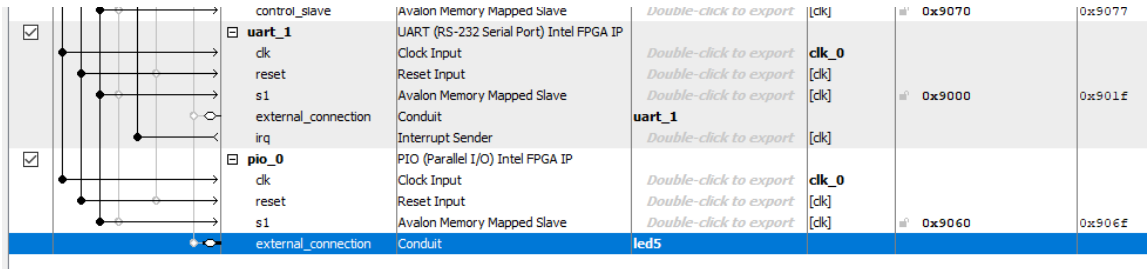


14. The external\_connection for uart\_1 needs to be set. Double-click on the external\_connection line under Export.

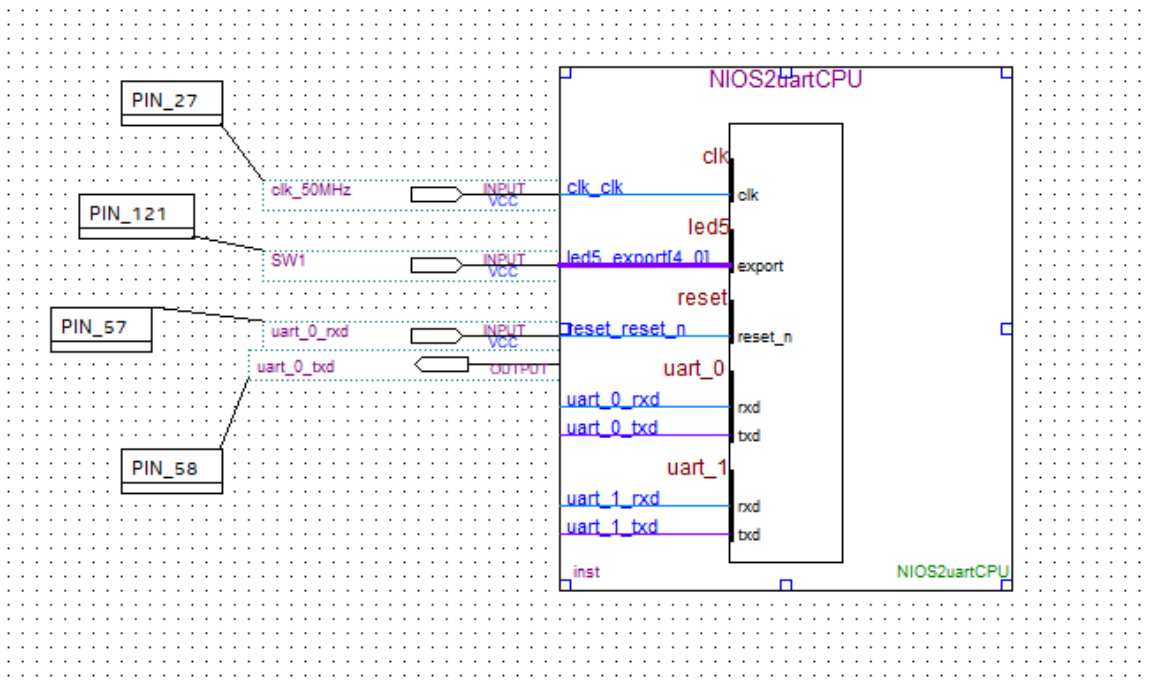
15. Set the name to uart\_1.

16. The external\_connection for pio\_0 needs to be set. Double-click on the external\_connection line under Export.

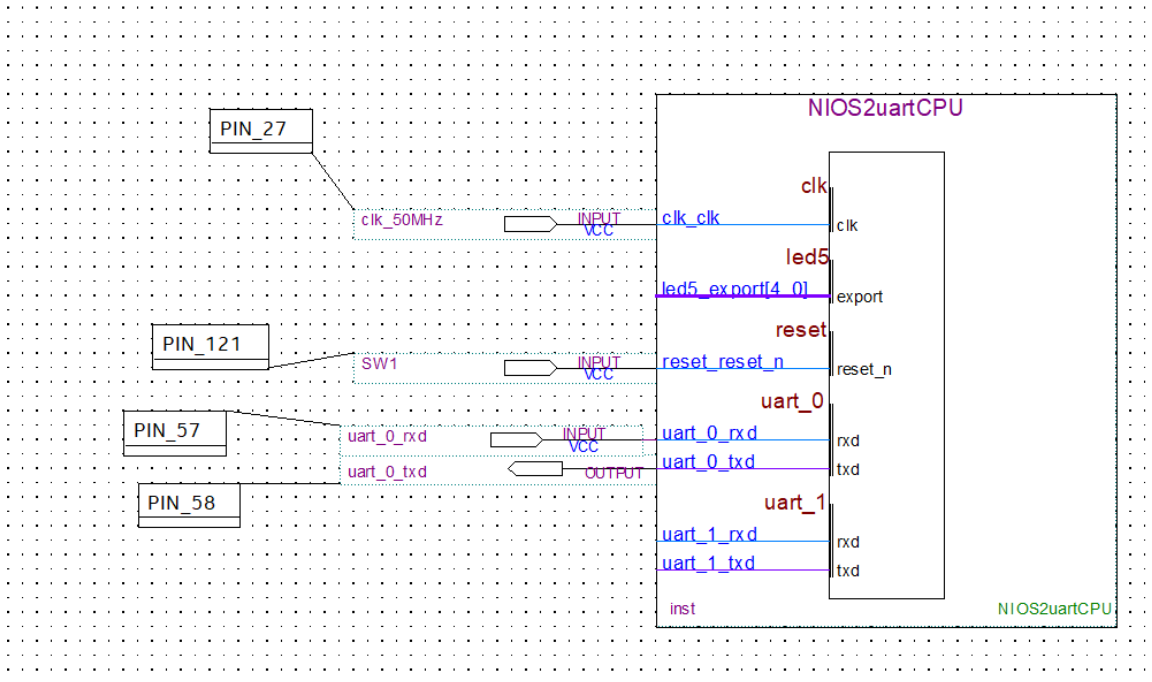
17. Set the name to led5.
18. From the menu, select System->Assign Base Addresses. This will give the new hardware address location in the memory map.



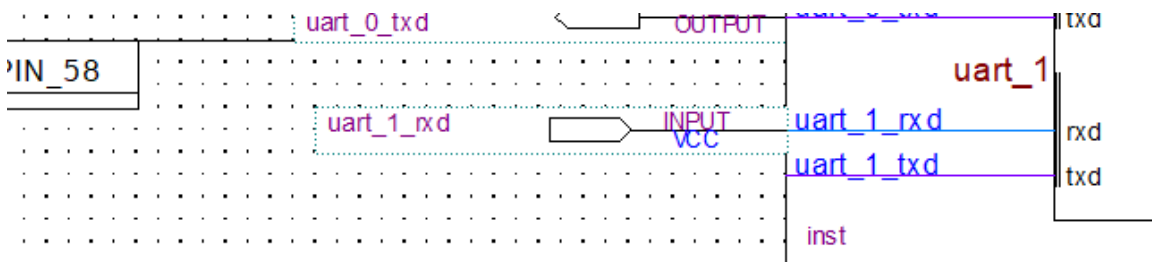
19. Click Generate HDL...
20. Click Close, when the design has been saved.
21. After the Generate process completes, click Close.
22. Click Finish to close Platform Designer.
23. A dialog appears reminding you that the design has changed. Click OK.
24. Open the block diagram.
25. Right-click on the NIOS2uartCPU and select Update Symbol or Block from the context menu. The new connection points for the NIOS2uartCPU will appear.



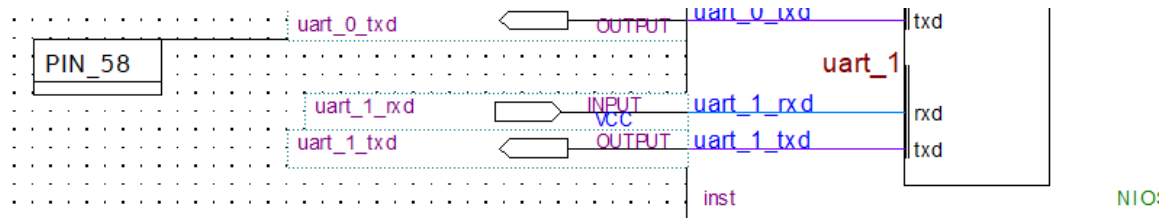
26. Adjust the connections to their original states.



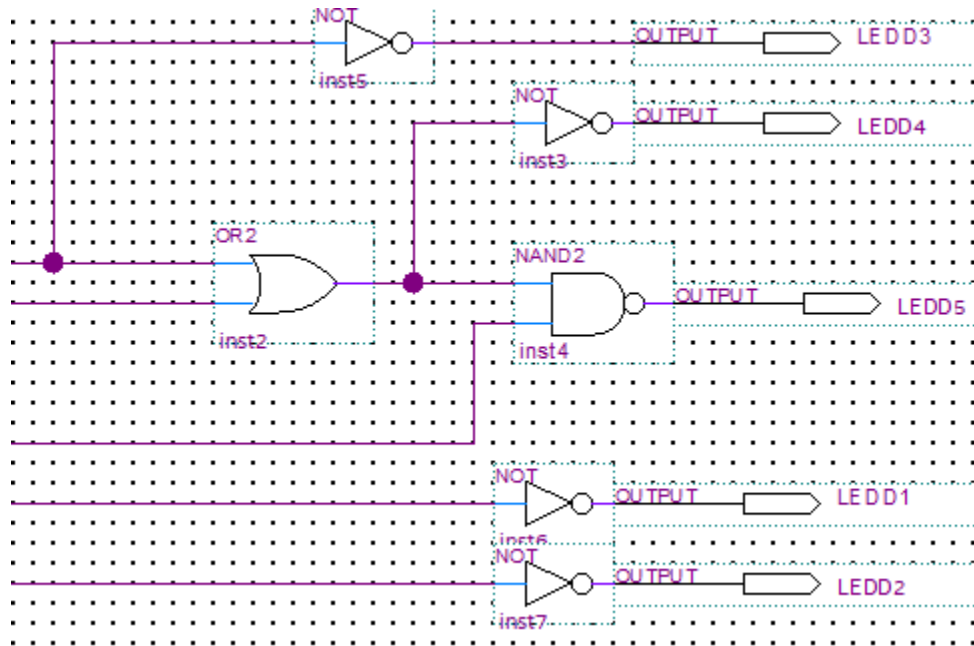
27. From the toolbar, click on the pin tool and select input.
28. Drop and connect the new pin to uart\_1\_rxd.
29. Rename the pin connection to uart\_1\_rxd.



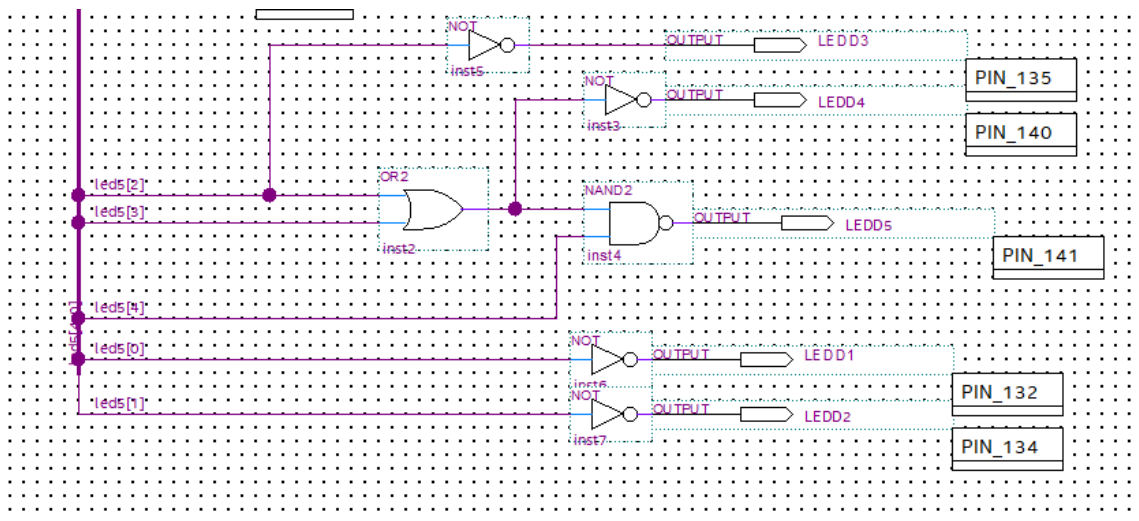
30. From the toolbar, click on the pin tool and select output.
31. Drop the symbol on the design.
32. Right-click on the symbol and select Flip Horizontal from the context menu.
33. Move the pin to connect to uart\_1\_txd.
34. Rename the pin connection to uart\_1\_txd.



35. Double-click on the design to bring up the Symbol tool. We are going to add the logic gates that were created in the second project of the *Getting Started with Intel® Quartus® Prime v21.0 and the Intel® MAX® 10-10M08 Evaluation Kit* article.
36. Expand the libraries by going to primitives->logic.
37. Add an OR2, NAND2, and four NOT gates below the NIOS2uart CPU.
38. Add five output pins.
39. Connect the gates and pins and rename the input and output pins as shown.



40. Draw a bus line from led5\_export[4..0] down to the logic gates.
41. Right-click on the BUS line, and select Properties.
42. Name the bus line led5[4..0], and click OK.
43. Connect the lines to the bus as shown and name each line for one bit of the bus line. Each line is one of the bus line's bits. Be sure to connect the output LEDD3 to the led5[2] line.



Led5-1 and led5-2 will be directly connected to the LEDD1 and LEDD2 respectively. Led5-3 will be connected to LEDD3, but also to our little logic circuit. The pin assignments will connect everything

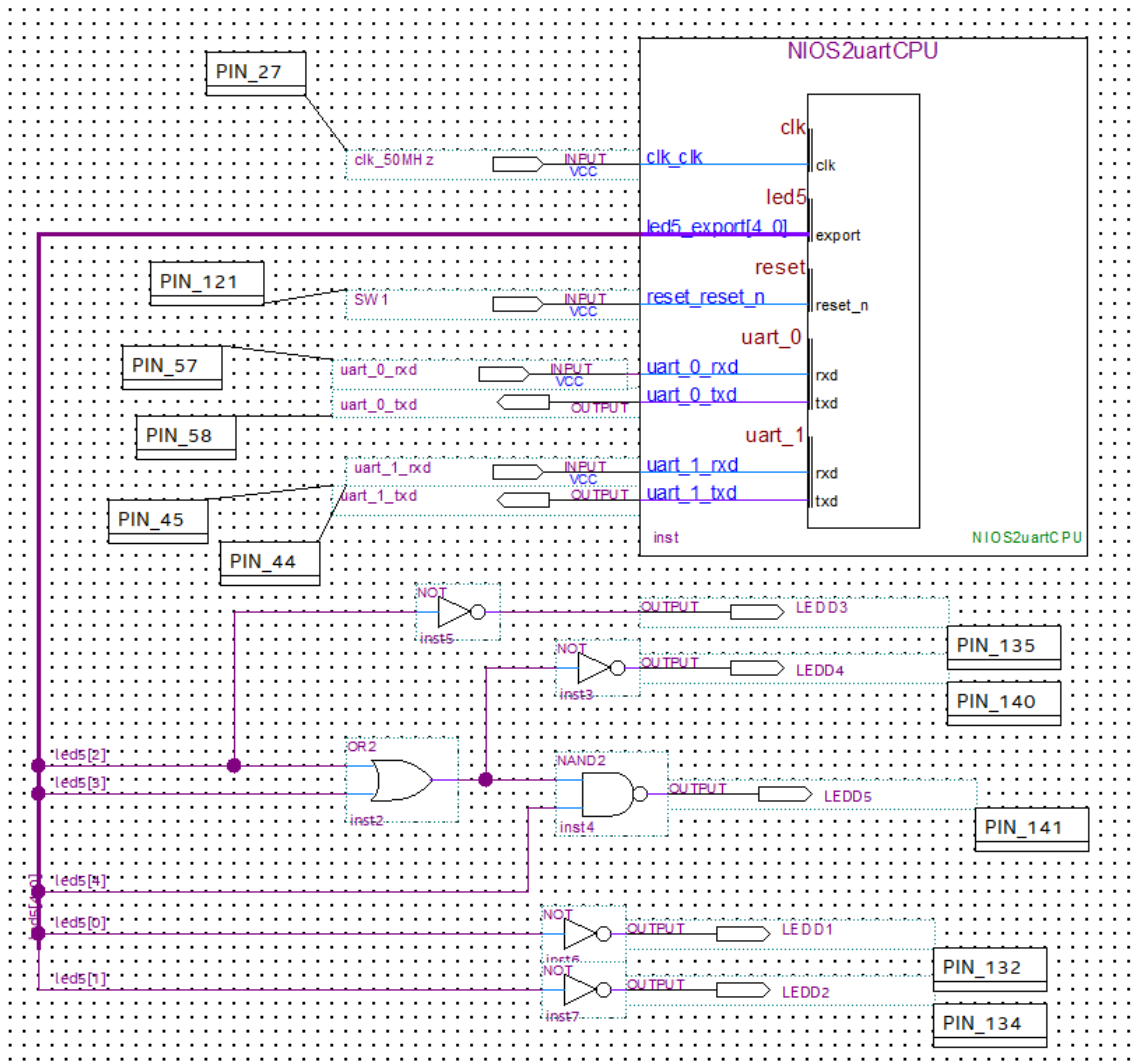


together. The lesson here is that the FPGA can act as an MCU processor, but with room left over. More logic can be added to help reduce component count and board space.

44. Make sure all of the part symbols don't have the same 'inst' name or the compile will fail.
45. Save the design.
46. In the Task pane on the left, double-click on Fitter (Place & Route) to start the task. If there is a failure, like having two components with the same 'inst' name like we have here, rename the 'inst' to a different value like inst4. Save the design and try again.
47. Open Pin Planner. You will need the board schematic to assist with the pin placement. The original pin assignments should already be set. We just need to add the new pin assignments.
48. For uart\_1, let's set the RX line to pin 44, which is J8-16 on the header. Set the TX line to 45, which is J8-14 on the header.
49. The LED will be connected in the following numerical order: 132 (LED1), 134 (LED2), 135 (LED3), 140 (LED4), and 141 (LED5).
50. Set all the voltages to 3.3-V LVTTTL

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
altera_reserved_tck	Input	PIN_18	1B	B1_NO	PIN_18	2.5 V Sc... Trigger
altera_reserved_tdi	Input	PIN_19	1B	B1_NO	PIN_19	2.5 V Sc... Trigger
altera_reserved_tdo	Output	PIN_20	1B	B1_NO	PIN_20	2.5 V
altera_reserved_tms	Input	PIN_16	1B	B1_NO	PIN_16	2.5 V Sc... Trigger
clk_50MHz	Input	PIN_27	2	B2_NO	PIN_27	3.3-V LVTTTL
LEDD1	Output	PIN_132	8	B8_NO	PIN_17	3.3-V LVTTTL
LEDD2	Output	PIN_134	8	B8_NO	PIN_13	3.3-V LVTTTL
LEDD3	Output	PIN_135	8	B8_NO	PIN_113	3.3-V LVTTTL
LEDD4	Output	PIN_140	8	B8_NO	PIN_15	3.3-V LVTTTL
LEDD5	Output	PIN_141	8	B8_NO	PIN_11	3.3-V LVTTTL
SW1	Input	PIN_121	8	B8_NO	PIN_121	3.3-V LVTTTL
uart_0_rxd	Input	PIN_57	3	B3_NO	PIN_57	3.3-V LVTTTL
uart_0_txd	Output	PIN_58	3	B3_NO	PIN_58	3.3-V LVTTTL
uart_1_rxd	Input	PIN_44	3	B3_NO	PIN_79	3.3-V LVTTTL
uart_1_txd	Output	PIN_45	3	B3_NO	PIN_120	3.3-V LVTTTL
<<new node>>						

51. Close the Pin planner and save the project. The block diagram will be filled with the pin assignments,



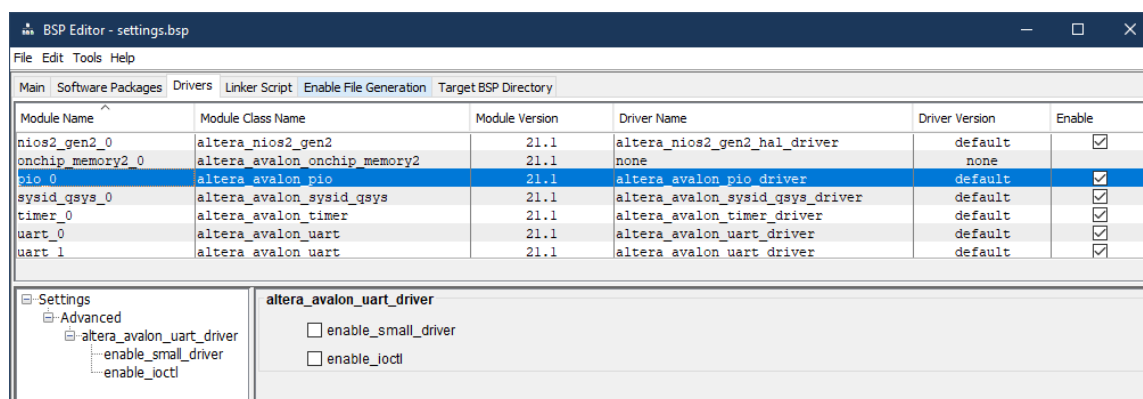
52. Compile the design. There should be no errors. You can see from the summary that we are only using 44% of the logic elements in the MAX 10 10M08. There is more that can be added if needed.

Flow Summary	
Flow Status	Successful - Sat Jul 9 21:24:14 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	NIOS2uart
Top-level Entity Name	NIOS2uart
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	3,538 / 8,064 ( 44 % )
Total registers	2122
Total pins	11 / 101 ( 11 % )
Total virtual pins	0
Total memory bits	193,216 / 387,072 ( 50 % )
Embedded Multiplier 9-bit elements	6 / 48 ( 13 % )
Total PLLs	0 / 1 ( 0 % )
UFM blocks	0 / 1 ( 0 % )
ADC blocks	0 / 1 ( 0 % )

### 1.2.3 Update the BSP and Change the application

With the new design successfully compiled, we can move on to modifying the application.

1. In Quartus Prime, from the menu, select Tools->NIOS II Software Build Tools for Eclipse.
2. Right-click on the uartTerminal\_bsp and select Clean Project.
3. Right-click on the uartTerminal\_bsp and select Refresh.
4. Right-click on the uartTerminal\_bsp and select Nios II->Generate BSP. The updated information will be pulled from the design, but it will not be obvious.
5. You will see that a new altera\_avalon\_pio\_regs.h has been added, and system.h now has the constants for uart\_1 and pio\_0.
6. Another way to see that the BSP has been updated is to right-click on the uartTerminal\_bsp and select Nios II->BESP Editor.
7. When the editor opens, click on the Drivers tab. You will see that pio\_0 and uart\_1 have been added. The one change to use small\_C library stays the same. Note that anytime you change something in the design, you have to re-generate the project's BSP.



8. Click Exit.
9. Open the main.c file in uartTerminal.

10. Let's back up the code in its current state to a different file. From the menu, select File->Save As, and save the main.c file as main.c.org.
11. Close main.c.org.
12. Open main.c
13. Before we go further, we also need to update the project with the new #include's. Right-click on uartTerminal, select index->Re-resolve Unresolved includes. The reason for this is that files like system.h, in the bsp project, have been updated and there is a new altera\_avalon\_pio\_regs.h file. The application needs to be refreshed, or we will get errors with function calls and defines not being found.

Change the main.c file to the following:

```

1.  #include "sys/alt_stdio.h"
2.  #include "system.h"
3.  #include "altera_avalon_uart.h"
4.  #include "altera_avalon_uart_regs.h"
5.  #include "altera_avalon_uart_fd.h"
6.  #include "altera_avalon_pio_regs.h"
7.  #include "priv/alt_busy_sleep.h"
8.  #include "sys/alt_sys_wrappers.h"
9.
10. int main()
11. {
12.     alt_putstr("Fun with NIOS II and LED Logic!\n");
13.
14.     char rx_buffer[1];
15.     char rx_bufferString[4]; //covers two characters, the carriage return
    \r and line feed \n
16.     char tx_buffer[] = "Number Received: ";
17.
18.     //SparkFun SerLCD commands
19.     char LCDCLEAR[2]={ 0xFE, 0x01 };
20.     char LCDLINE1[2]={ 0xFE, 0x80 };
21.     char LCDLINE2[2]={ 0xFE, 0xC0 };
22.
23.     int i=0;
24.     int x = 0;
25.     int y = 0;
26.
27.     int fp_uart_0;
28.     fp_uart_0 = ALT_OPEN(UART_0_NAME,2);
29.
30.     int fp_uart_1;
31.     fp_uart_1 = ALT_OPEN(UART_1_NAME,2);
32.
33.     alt_putstr("Enter a number 0 through 31 and hit enter\n");
34.     alt_putstr("*****\n");
35.     alt_putstr("\n");
36.
37.     //Clear buffer
38.     for(y = 0; y <= 4; y++){
39.         rx_bufferString[y] = 0;
40.     }
41.     //Clear line display
42.     ALT_WRITE(fp_uart_1,LDCLEAR, sizeof(LDCLEAR));
43.     //Clear LEDs
44.     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0x0);
45.
46.
47.     /* Event loop never exits. */
48.     while (1){
49.

```

```

50.         do{
51.             ALT_READ(fp_uart_0,rx_buffer, sizeof(rx_buffer));
52.             rx_bufferString[i] = rx_buffer[0];
53.             i++;
54.         }while((rx_buffer[0] != '\n'));
55.
56.         //convert to integer
57.         //If the second value is the carriage return '\r' then only
convert
58.         //the ones to an integer
59.         if(rx_bufferString[1] == '\r'){
60.             x = rx_bufferString[0]-'0';
61.         }
62.         else{
63.             x = rx_bufferString[0]-'0';
64.             x *= 10;
65.             x += rx_bufferString[1]-'0';
66.         }
67.
68.         i=0;
69.
70.         //write to the LEDs
71.         if((x >= 0) && (x <= 31)){
72.
73.             IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, x);
74.         }
75.         //Send message to Line display Line 1
76.         ALT_WRITE(fp_uart_1,LCDLINE1, sizeof(LCDLINE1));
77.         ALT_WRITE(fp_uart_1,tx_buffer, sizeof(tx_buffer));
78.
79.         alt_busy_sleep(1000000);
80.         //Send message to Line display Line 2
81.         ALT_WRITE(fp_uart_1,LCDLINE2, sizeof(LCDLINE2));
82.         ALT_WRITE(fp_uart_1,rx_bufferString, sizeof(rx_bufferString));
83.
84.         alt_putstr("\n");
85.         alt_putstr("Enter a number 0 through 31 and hit enter\n");
86.         alt_putstr("*****\n");
87.         alt_putstr("\n");
88.
89.         alt_busy_sleep(4000000);
90.
91.         //Clear buffer
92.         for(y = 0; y <= 4; y++){
93.             rx_bufferString[y] = 0;
94.         }
95.     }
96.
97.     return 0;
98. }

```

Since only a number from 0 to 31 can change the LEDs, the `rx_bufferString` has space for only 4 characters. When data is received from the serial port, it is not just the message but also the carriage return (`\r`) and the linefeed (`\n`) characters as well. The `rx_bufferString` makes room only for those possibilities. A message and commands for the 16x2 serial LCD are set up. Both UARTs are opened. The buffer, LEDs, and display are all cleared. A message to the user is sent over the standard I/O port (`uart_0`) to the terminal application.

The main program is similar to the original. The application waits for a message; and when the message arrives, the `rx_bufferString` value is converted to a number. The number is then sent to the PIO to control the LEDs. A message is sent to the 16x2 serial LCD indicating the value that

was received. A user message is sent asking to enter a number, and the serial buffer is cleared for the next message.

14. Save main.c
15. Build the application. Fix any errors and rebuild.
16. Close Eclipse.

### 1.2.4 Circuit Setup

The Schmartboard RS-232 module stays connected as before. The 16x2 serial LCD can be connected with VCC connected to J8-37, GND connected to J8-39, and the RX pin connected to J8-14.

### 1.2.5 Program the Board and Test the Application

First, program the board with the hardware design and then download and run the application. Make sure the JTAG is connected to the MAX 10 10M08 Evaluation Kit board, and power on the board. Also, you should connect a null modem cable from the host PC, with running a terminal application, to the board.

1. Open Quartus and the NIOS2uart project that you have previously compiled.
2. Open the JTAG programming tool and program the board with the NIOS2uart\_time\_limited.sof file. The countdown dialog will appear.
3. Open a serial terminal program and configure the serial communications to 115200-8-N-1.
4. Open Eclipse.
5. Right-click on the uartTerminal application and select Run As->Nios II Hardware.

Once the application has been downloaded and started running, LEDD1 and LEDD2 will be on, since we set the PIO initial value to 3 in the Platform Designer. The message to enter a number should appear on the terminal. Enter a number between 0 and 31 and hit enter. The LEDs will change accordingly and the 16x2 serial LCD will have a message and the number you entered. You will see some strange characters after the number. These are the `\r` and `\n`. You could add some code to remove these characters from appearing.

The LED results are a little different because of the logic gates used to control LEDD4 and LEDD5. The LEDs are active low, thus the use of NOT gates for all the LEDs.

Decimal	Binary	LED5	LED4	LED3	LED2	LED1
0	0 0000	OFF	OFF	OFF	OFF	OFF
1	0 0001	OFF	OFF	OFF	OFF	ON
2	0 0010	OFF	OFF	OFF	ON	OFF
3	0 0011	OFF	OFF	OFF	ON	ON
4	0 0100	OFF	ON	ON	OFF	OFF
5	0 0101	OFF	ON	ON	OFF	ON
6	0 0110	OFF	ON	ON	ON	OFF
7	0 0111	OFF	ON	ON	ON	ON
8	0 1000	OFF	ON	OFF	OFF	OFF
9	0 1001	OFF	ON	OFF	OFF	ON
10	0 1010	OFF	ON	OFF	ON	OFF
11	0 1011	OFF	ON	OFF	ON	ON
12	0 1100	OFF	ON	ON	OFF	OFF
13	0 1101	OFF	ON	ON	OFF	ON
14	0 1110	OFF	ON	ON	ON	OFF
15	0 1111	OFF	ON	ON	ON	ON
16	1 0000	OFF	OFF	OFF	OFF	OFF

17	1 0001	OFF	OFF	OFF	OFF	
18	1 0010	OFF	OFF	OFF	ON	OFF
19	1 0011	OFF	OFF	OFF	ON	ON
20	1 0100	ON	ON	ON	OFF	OFF
21	1 0101	ON	ON	ON	OFF	ON
22	1 0110	ON	ON	ON	ON	OFF
23	1 0111	ON	ON	ON	ON	ON
24	1 1000	ON	ON	OFF	OFF	OFF
25	1 1001	ON	ON	OFF	OFF	ON
26	1 1010	ON	ON	OFF	ON	OFF
27	1 1011	ON	ON	OFF	ON	ON
28	1 1100	ON	ON	ON	OFF	OFF
29	1 1101	ON	ON	ON	OFF	ON
30	1 1110	ON	ON	ON	ON	OFF
31	1 1111	ON	ON	ON	ON	ON

- When you are finished testing the application, close Eclipse, click Ok on the OpenCore Status dialog, and close Quartus.

### 1.3 Summary: Build Your Own MCU

With Nios II and the Intel FPGAs, you can build your own custom MCU with the peripherals needed for the system. Serial ports, SPI host, ADC, I2C master, and PIO are some of the items that can be added to a design to build a custom MCU. The last project had two UART ports and PIO ports.

There were a number of other items covered and hinted at in the projects:

- The BSP Editor is used to modify C libraries and HAL drivers.
- Remembering to update the Eclipse project BSP after there is a hardware design change is important.
- Note the differences between the available UART IP blocks.
- Opening Eclipse after programming the board with the design and then running the Eclipse application helps to avoid any issues with the JTAG. If you didn't see this, it is because we instructed you to close Eclipse beforehand.
- The `\r` and `\n` in the message sent from a terminal have to be addressed in the program.
- Wiring and mapping the external hardware to the board requires the help of the schematic diagram.
- External logic can be added to the Nios II MCU within the FPGA design.

Finally, because memory was limited, the small C Library was used, thus there are no C library stdio calls nor the usual open, close, etc. for accessing the serial ports. Instead, the Nios wrapper APIs in the BSP were used for accessing the serial ports.

### 1.4 References

The following references were used for this article:

#### 1.4.1 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](#)

Copyright © 2022 Annabooks, LLC. All rights reserved  
Intel, Quartus, Nios II, and MAX 10 are registered trademarks of Intel Corporation  
All other copyrighted, registered, and trademarked material remains the property of the respective owners.

### 1.4.2 Online Videos

[My First Nios II Tutorial \(1\)](#) by Terasic, Inc

[My First Nios II Tutorial \(2\)](#) by Terasic, Inc

[Qsys Tutorial 1 - Adder using NIOS II processor](#)

[37 FPGA NIOS II QSYS 07 uart \(or serial port\) - YouTube](#)

### 1.4.3 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.