Nios® V UART Project Using Visual Studio Code on the MAX® 10 FPGA 10M50 Evaluation Kit

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

January 2026

The previous two FGPA articles covered building Nios V applications using Ashling RiscFree™ IDE. The application development workflow using this tool is not as smooth as the mature Nios II workflow. A recent update includes a Ashling RiscFree plug-in for Visual Studio Code, which turns out to create a much better workflow. This article will demonstrate using VS Code to create a Nios V application that sends and receives characters over a UART connection. The article is based on the previous article: *Nios® II + UART Project on Intel® MAX® 10-10M08 Evaluation Kit*. Based on user feedback from that article a USB to UART/TTL Serial cable will be used for the UART connection.

Please, see the article *Altera™ Quartus® Prime Lite v25.1 and Nios® V Install Instructions* on Annabooks.com for information on how to install the software needed for this hands-on exercise.

The Project Requirements:

- Quartus Prime Lite Edition V25.1, Ashling RiscFree™ IDE, and Nios® V license are already installed.
- Visual Studio Code installed with the Ashling VS Code Extension for Altera FPGAs installed.
- Intel® MAX® 10 - 10M50 Evaluation Kit and the schematic for the evaluation board are required. The schematic PDF file can be downloaded from the Intel FPGA website.
    - J14 GPIO header with 2x7 0.1-inch headers installed
- USB-to-TTL Serial Convert Cable -3.3V.
    - DTECH USB to TTL Serial Cable 3.3V 3 Pin or equivalent.
- A terminal program such as ABCOMTerm (www.annabooks.com), HyperTerminal, or Tera Term.

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.

The Nios V workflow is different than the Nios II projects. This article will show how the Nios V application workflow is better integrated using VS Code:

- The first step is still the same: creating the hardware design in Quartus Prime and Platform Builder.
- The second step: Visual Studio Code will be used to create the BSP and application

Note: As of this writing, Altera is still migrating information and licensing details from Intel. You will see a mix of both company names until the transfer is complete.

1.1 Part 1: Basic Nios V Design

For this design, an application will run on the Nios V processor to send communications back and forth over a serial cable. A UART IP will be added to a Nios V

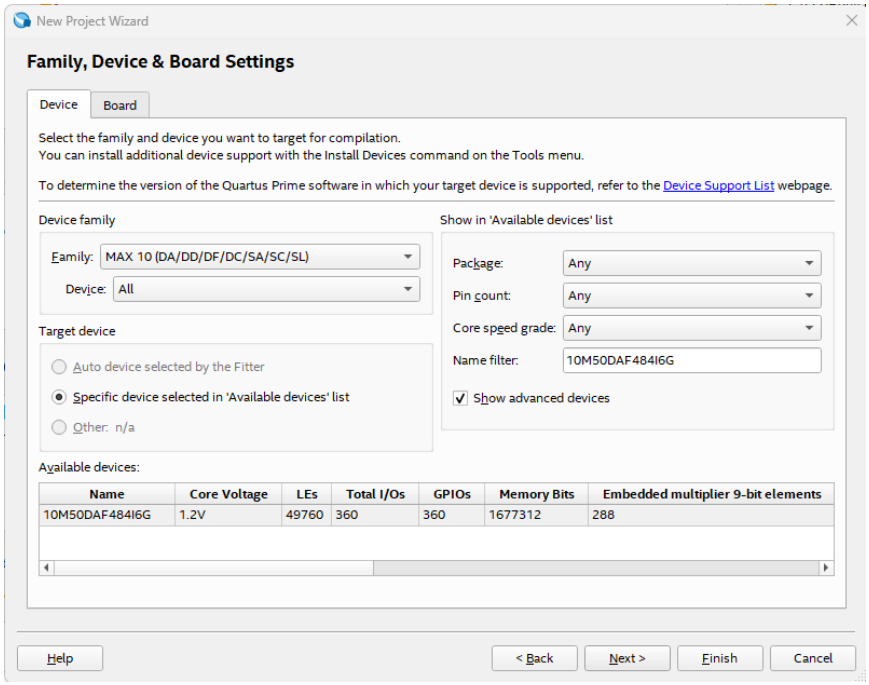
1.1.1 Create the Project

The first step is to create a design project.

1. Open Quartus.
2. Click on the New Project Wizard.
3. Select or create a project directory \niosv_usert (Do not use the Quartus installation directory) and name the project: “niosvuart”. 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.

4. Project Type: Empty project, click Next.
5. Add File: no files to add, click Next.
6. Family, Device & Board Settings:
    a. Change the Family to “MAX 10(DA/DD/DF/DC/SA/SC/SL)”
    b. Put 10M50DAF484I6G in the Name filter text box. This should narrow down the list to a single device.



7. Select the single available device and click Next.
8. For the EDA tools, clear everything to <<none>>, and click Next.
9. Summary: click Finish.

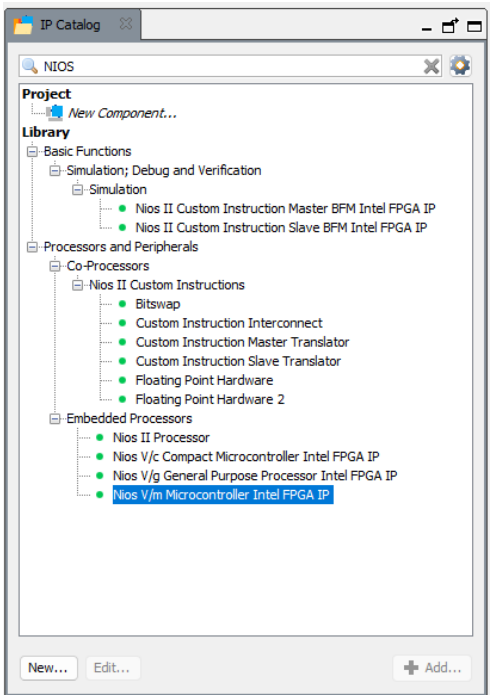
1.1.2 Create the Design Step 1: Platform Designer

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

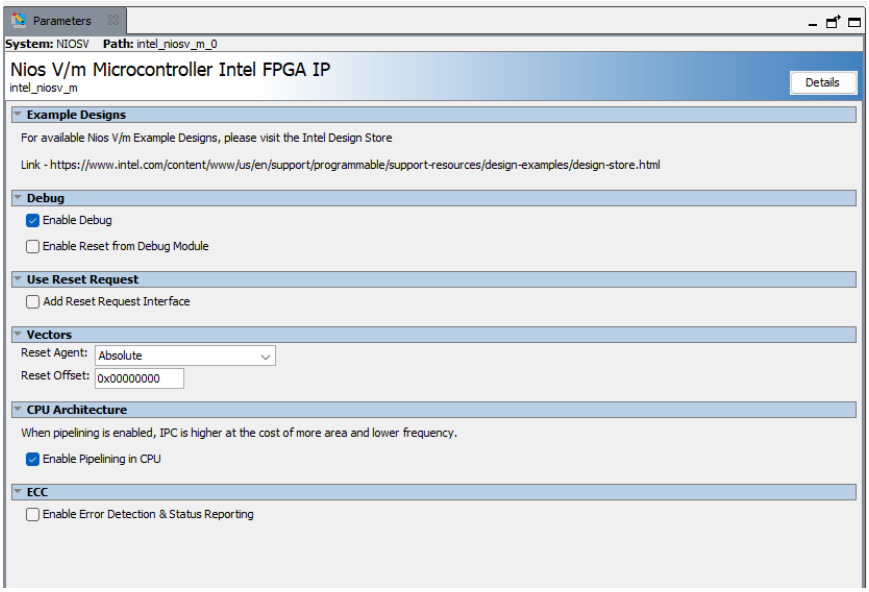
1. From the menu, select Tools->Platform Designer, or the Platform Designer icon  from the toolbar.

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

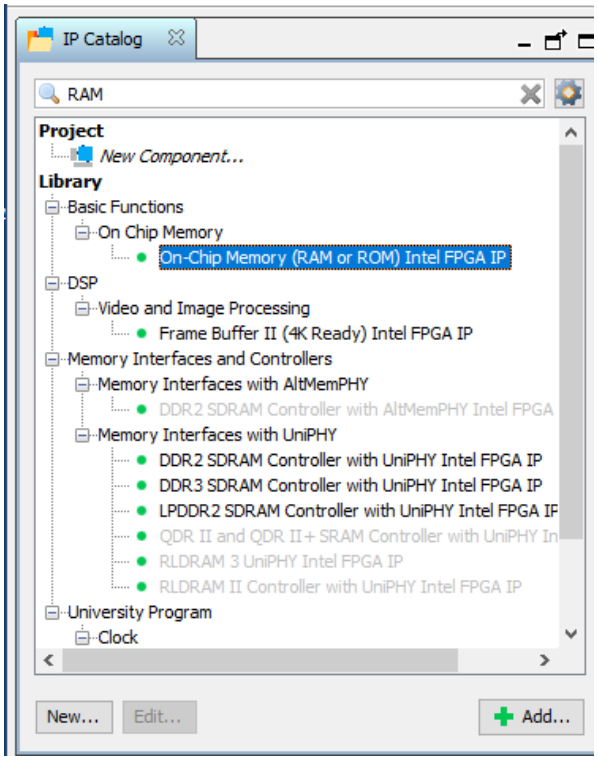
2. The top left pane contains the IP Catalog with all the available IP blocks that come with Quartus Prime. In the search box, type NIOS.



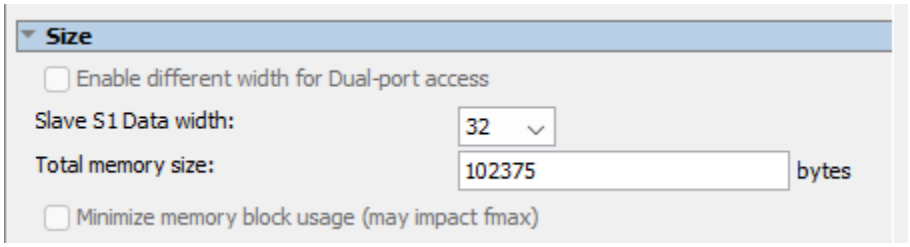
3. Expand the Processors and Peripherals and the Embedded Processors branches. Under Embedded Processors double-click on the Nios V/m Processor Intel FPGA IP.
4. This will open the Nios V/m Configuration page. We will keep the defaults for now. Click Finish.



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

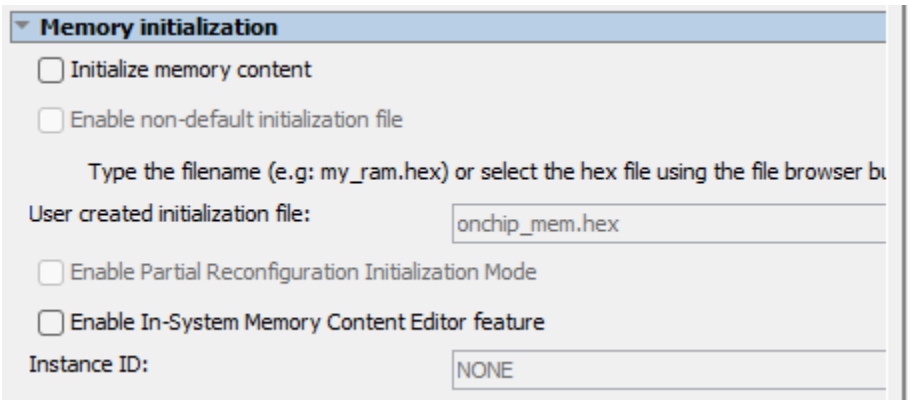


7. The configuration page will appear. Change the Total memory size to 102375. This is 819,000 bits which is about half the available memory.

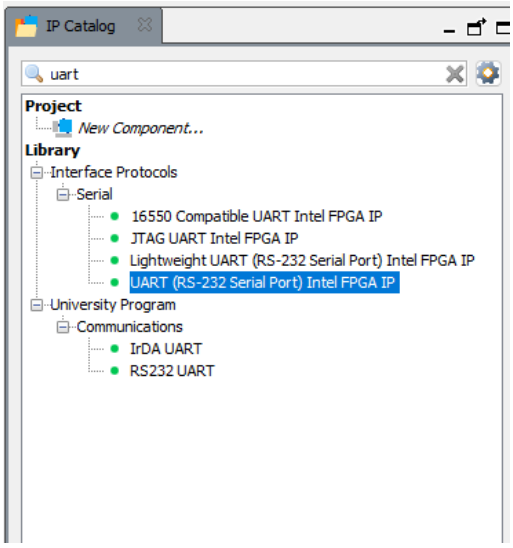


Intel Max 10 Part Number	Logic Elements	Maximum Embedded Memory	Maximum User I/O Count
10M02	2000	108 Kbits	246
10M04	4000	189 Kbits	246
10M08	8000	378 Kbits	250
10M16	16000	549 Kbits	320
10M25	25000	675 Kbits	360
10M40	40000	1.26 Mbits	500
10M50	**50000**	**1.638 Mbits**	**500**

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

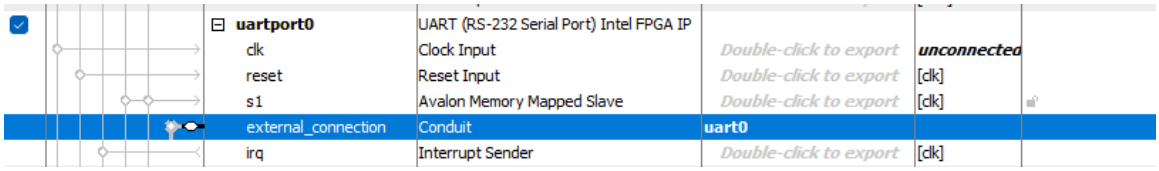


9. In the IP Catalog search, enter uart.
10. Double-click on the UART (RS-232 Serial Port) Intel FPGA IP16550

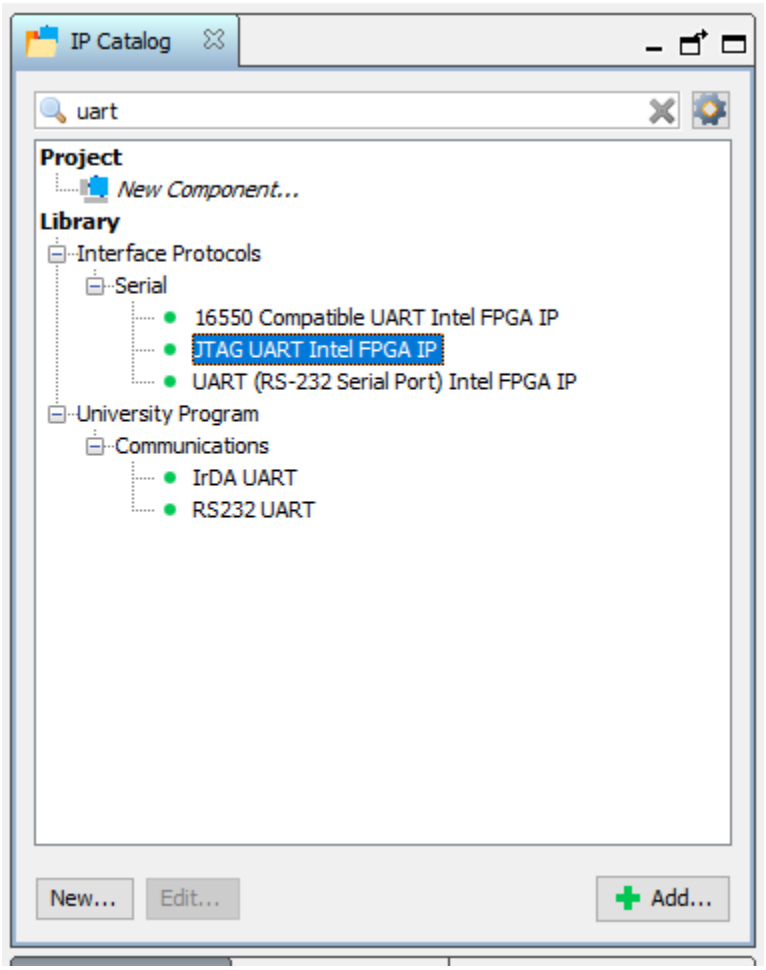


You will notice that there are two Serial port IPs.16550 Compatible UART Intel FPGA IP and UART (RS-232 Serial Port) Intel FPGA IP. The former contains support for a full FIFO and DMA transfer. The latter that we chose for this design doesn't have a buffer and is a very simplistic character reader. The UART (RS-232 Serial Port) Intel FPGA IP makes practical sense for a simple project as this one. 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.

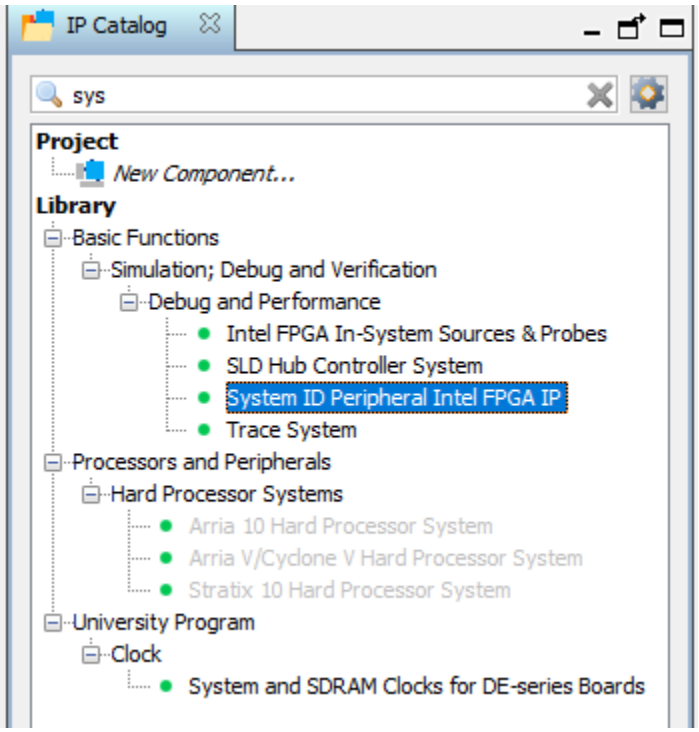
11. A configuration page will appear. The default settings of 115200-8-N-1 are fine for this project. No changes need to be made. Click Finish.
12. In the System Contents tab, change the name of the uart_0 to uartport0.
13. In the Export column, double click on the line for "external_connection".
14. Enter uart0.



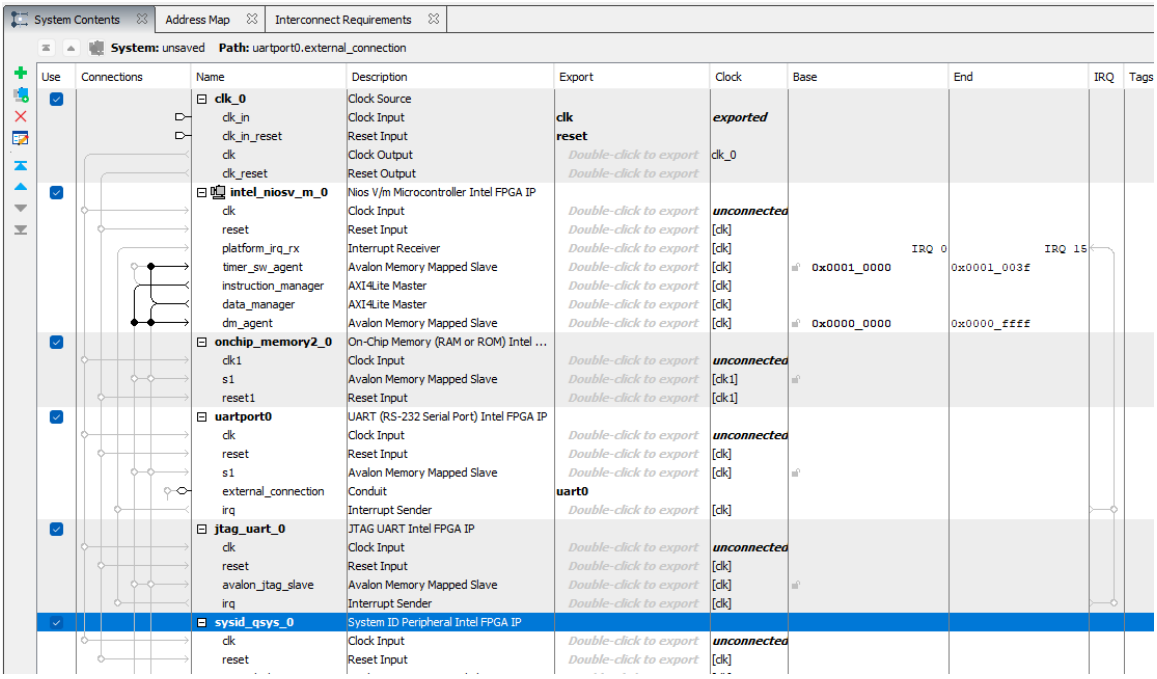
15. Double-click on the JTAG UART Intel FPGA IP.



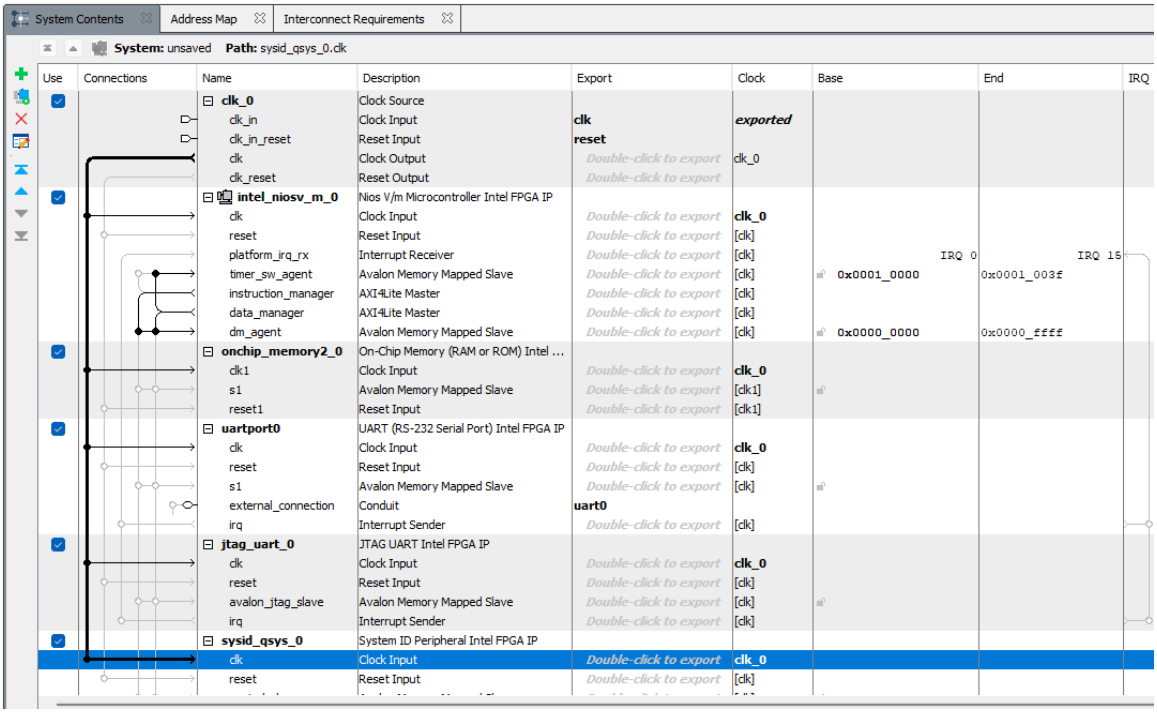
16. A configuration page will appear. There are no changes to be made. Click Finish.
17. In the IP Catalog search, enter the system ID.
18. Double-click on the System ID Peripheral Intel FPGA IP.



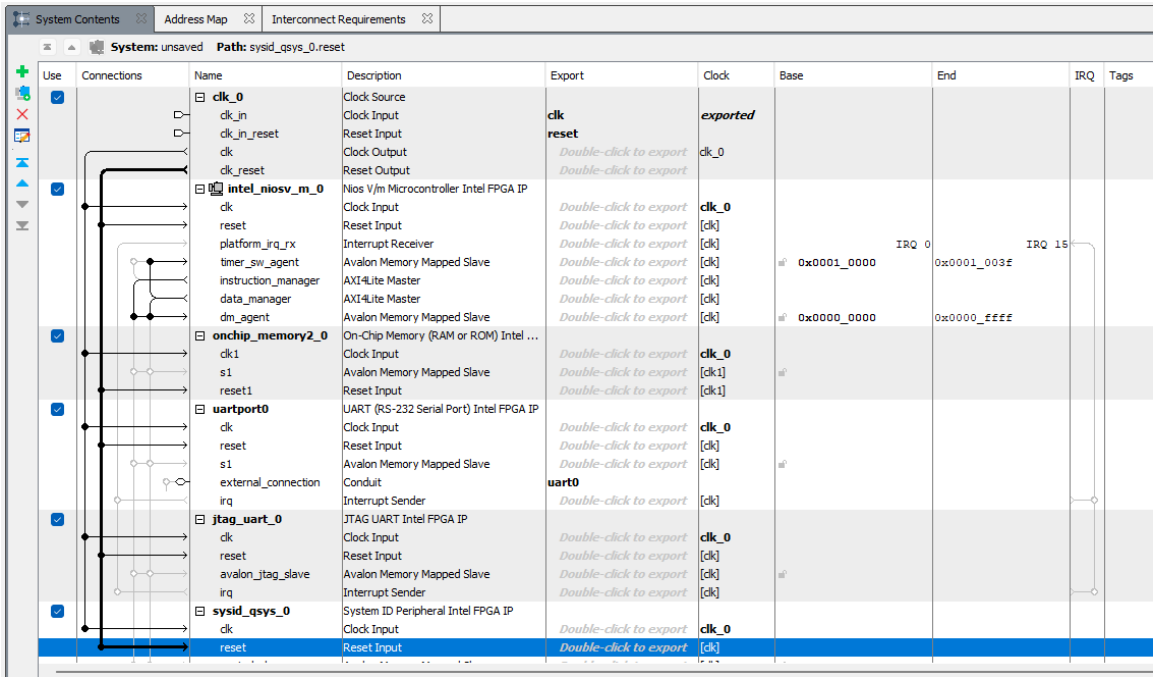
19. A configuration page will appear. There are no changes to be made. Click Finish.



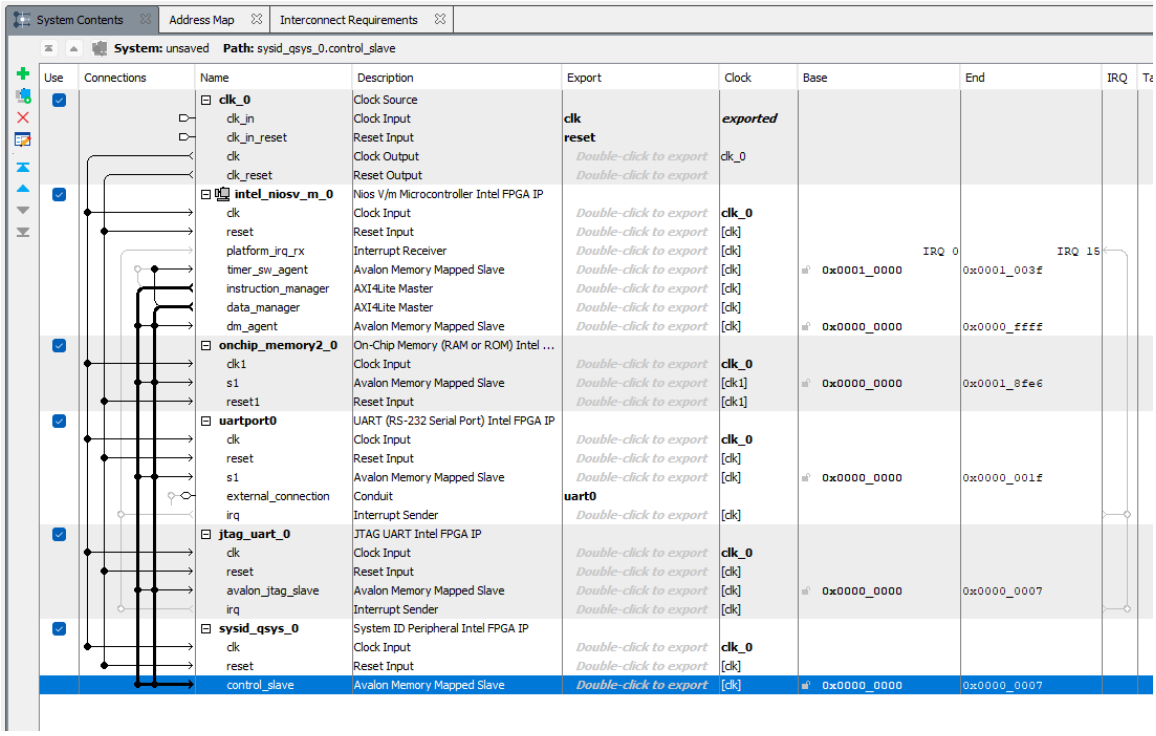
20. Now. we need to wire the IP blocks together. First, wire all the clk lines together by clicking on the dots for all IP blocks.



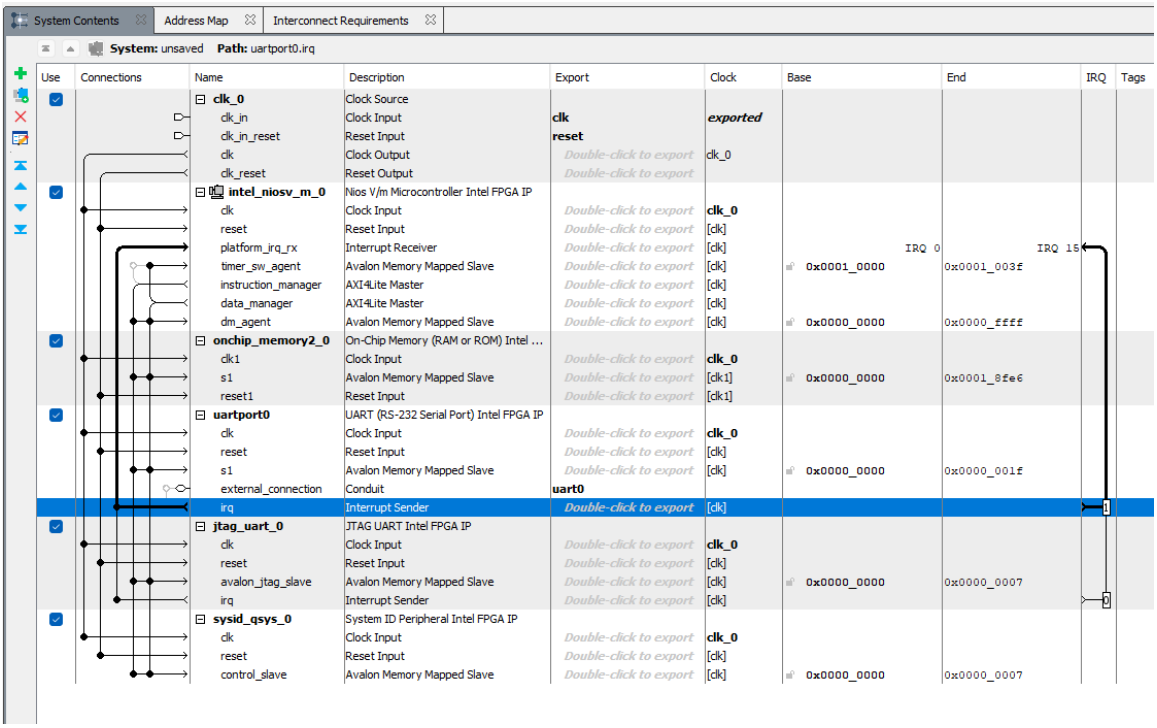
21. Next, connect all the reset lines together by clicking on the dots for all IP blocks.



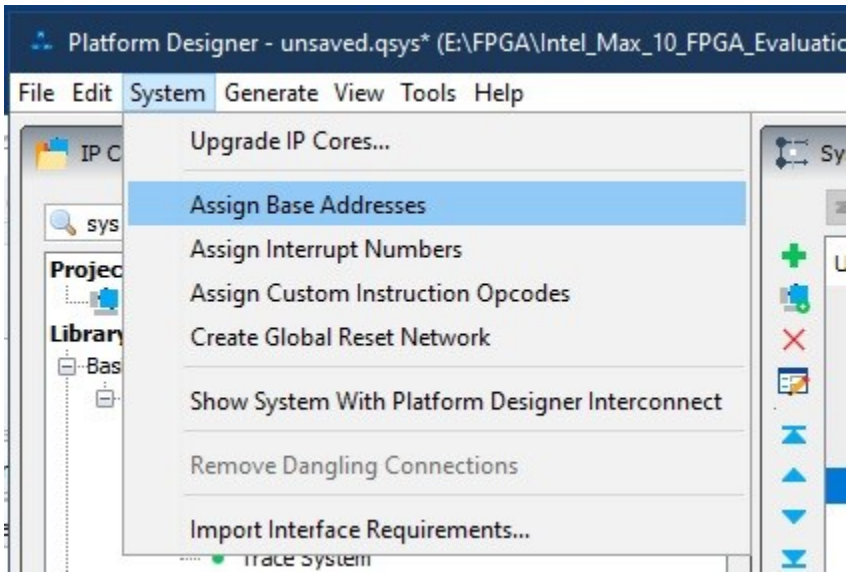
22. The memory lines have to be connected together. Connected the Instruction_manager and data_manager to all other items in the design.



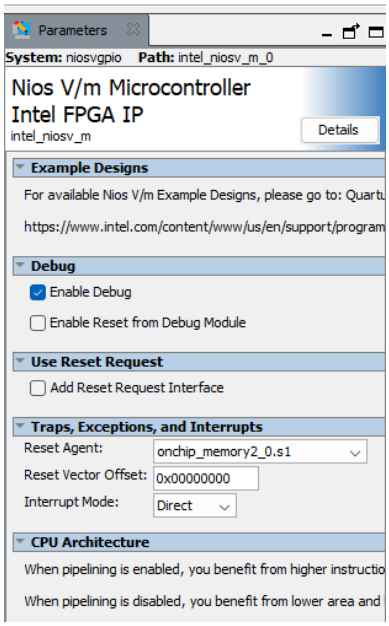
23. Finally, connect the jtaq_uart and uartport0 irq lines to the intel_niosv_m_0.
24. If you scroll to the right, the irqs are given a default value.



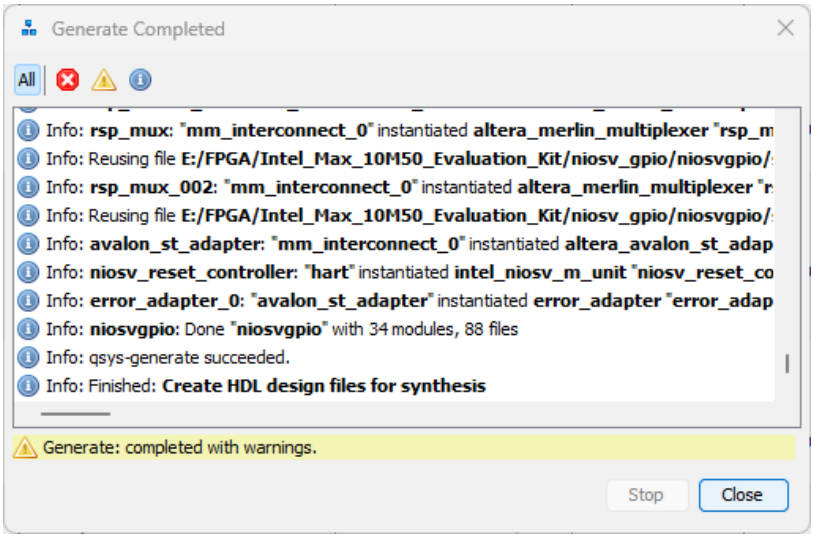
25. Let's assign a base address. From the menu, select System->Assign Base Address. This will remove a number of errors from the message box.



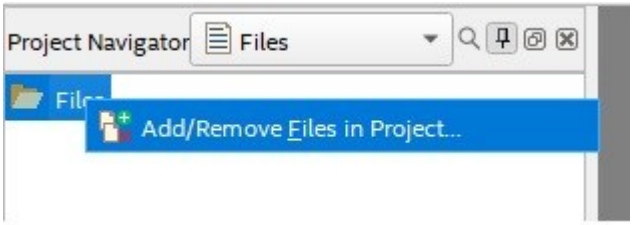
26. Finally, let's set the reset and exception vector addresses. Double-click on the intel_niosv_m_0 to open the configuration page.
27. In the Vectors section, change the Reset Agent to onchip_memory2_0.s1



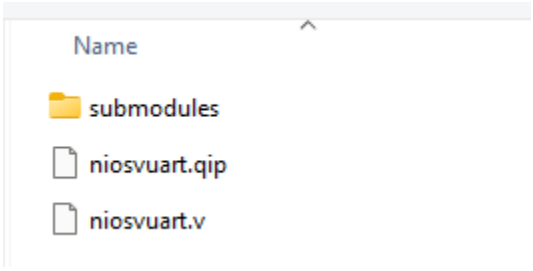
28. Save the design as niosvuart.qsys.
29. Once the save has been completed, click Close.
30. Click on Generate HDL…
31. A dialog appears, keep the defaults and click the Generate button.
32. The generate process kicks off. The processes should succeed with warnings, click Close.



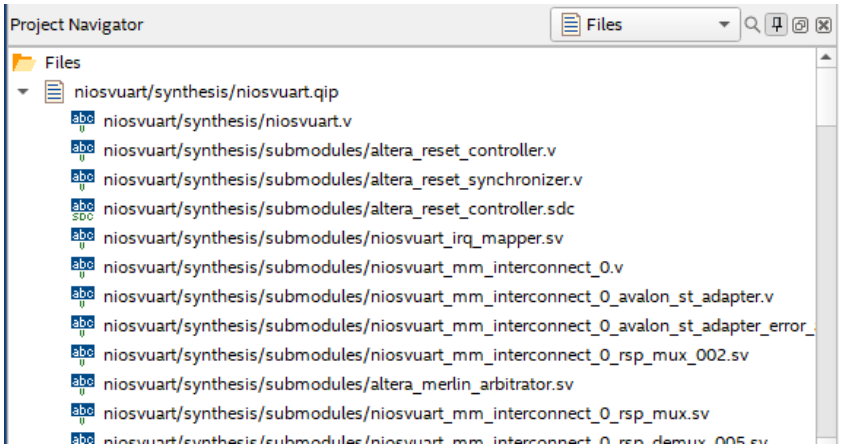
33. Click Finish to close the design.
34. Quartus then reminds you to add the new design to the project. Click Ok.
35. In the Project Navigator, click on the drop-down and select Files.
36. Right-Click on Files and select Add/Remove Files in Project.



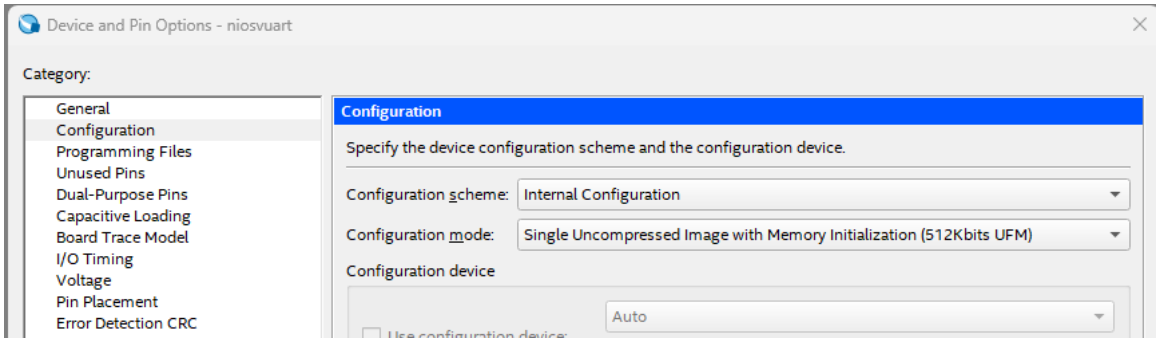
37. A Settings – niosvuart page appears with Files on the left highlighted. Click the three dots, browse button for File name, and navigate to \niosv_uart\niosvuart\synthesis folder.
38. Click on niosvuart.qip file and click open



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

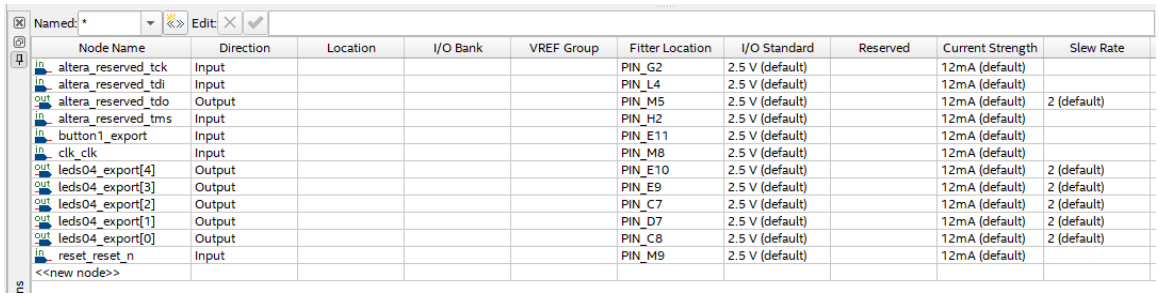


40. Save the project.
41. Click on Assignments->Device
42. Click the “Device and Pin Options…” button.
43. Change the Configuration mode to “Single Uncompressed Image with Memory Initialization (512kbits UFM)”.



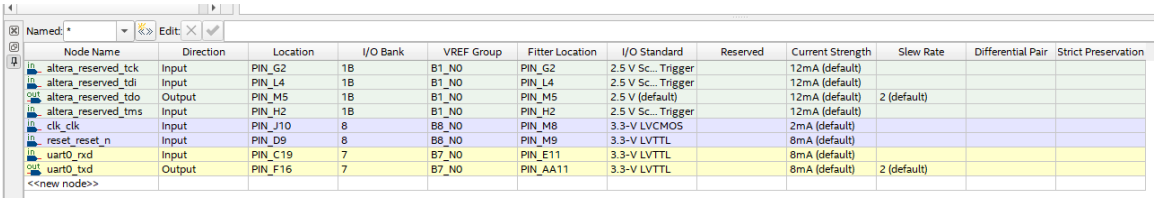
44. Click Ok.
45. Click Ok again.
46. In the project navigator, right click on NIOSV.qip and select “Set as Top-Level Entity”.
47. In the Task pane on the left, double-click on Fitter (Place & Route) to start the task. The analysis will take some time, and it should succeed in the end. This step helps to diagnose any errors and finds the Node Names for the pin assignments in the next step.

48. 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.



49. Using the board schematic, locate the pins for the 50Mhz clock, JTAG, J14 pin USER_IO3 for uart0_rxd, J14 pin USER_IO4 for uart0_txd, and reset. Set the Location values and I/O Standard for the MAX 10 – 10M50 Evaluation Board as follows:

Node Name	Location	I/O Standard
altera_reserved_tck	PIN_G2	2.5 V Schmitt Trigger
altera_reserved_tdi	PIN_L4	2.5 V Schmitt Trigger
altera_reserved_tdo	PIN_M5	2.5 V
altera_reserved_tms	PIN_H2	2.5 V Schmitt Trigger
clk_clk	PIN_J10	3.3-V LVCMOS
reset_reset_n	PIN_D9	3.3-V LVTTL
uart0_rxd	PIN_C19	3.3-V LVTTL
uart0_txd	PIN_F16	3.3-V LVTTL



50. Close the Pin Planner when finished.
51. Save the project.

1.1.3 Add Design Constraints File

1. From the menu in Quartus, select File->New-> Synopsys Design Constraints File
2. Click Ok
3. Add the following:

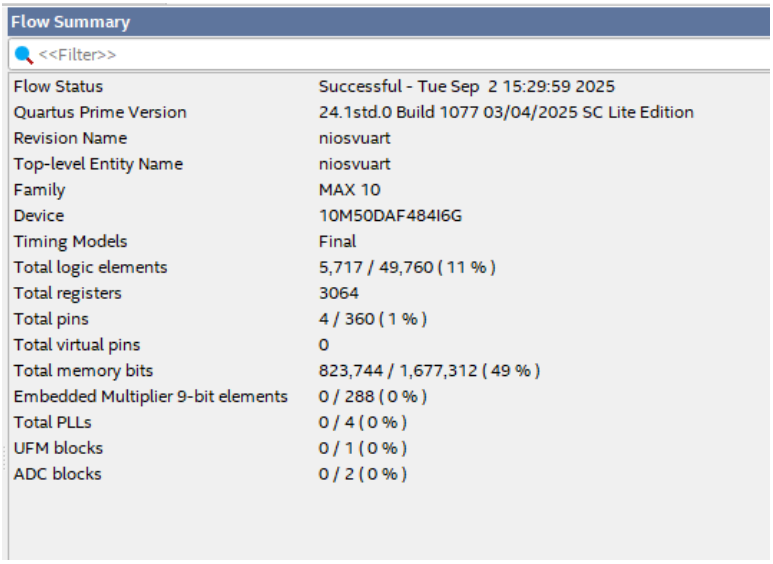
```
create_clock -period "50Mhz" -name clk_50MHz clk_clk

derive_pll_clocks -create_base_clocks
derive_clock_uncertainty

set_false_path -from [get_ports reset_reset_n]
set_false_path -from [get_ports altera_reserved*]
set_false_path -from * -to [get_ports altera_reserved_tdo]

set_false_path -from * -to [get_ports uart0_txd]
set_false_path -from [get_ports uart0_rxd]
```

4. Save the file as niosvuart.sdc

1.1.4 Compile the Project

The final step is to compile the design

1. 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 compile should complete successfully.



Flow Summary

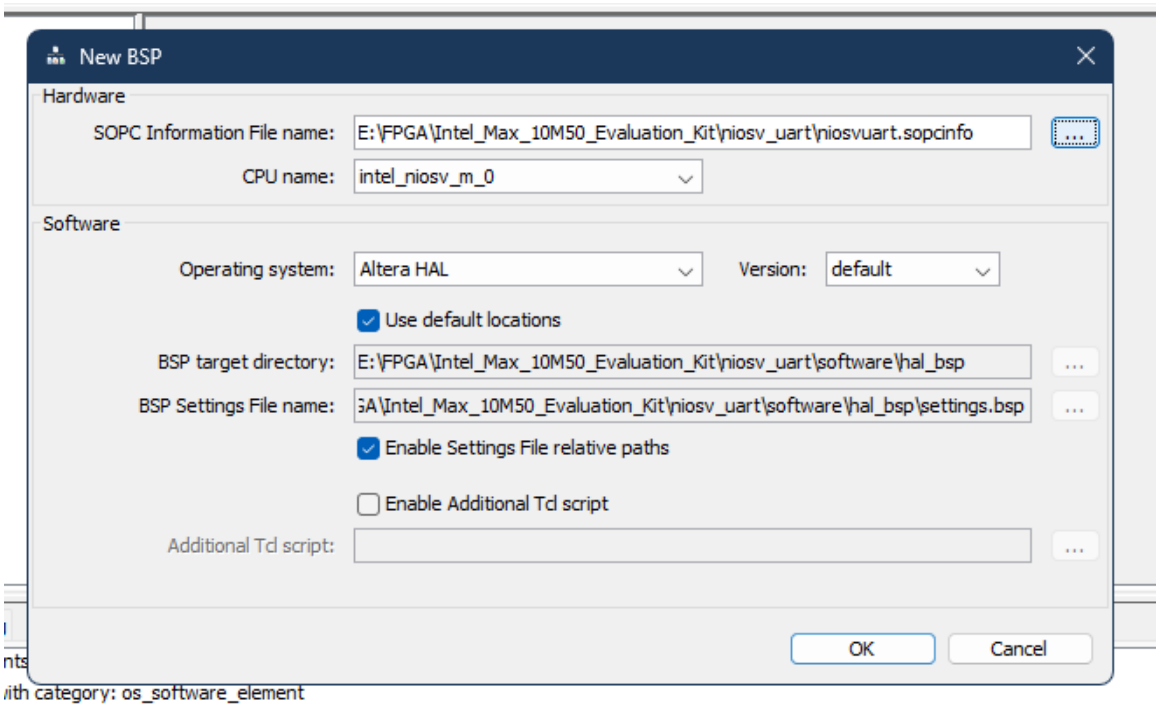
<<Filter>>

Flow Status	Successful - Tue Sep 2 15:29:59 2025
Quartus Prime Version	24.1std.0 Build 1077 03/04/2025 SC Lite Edition
Revision Name	niosvuart
Top-level Entity Name	niosvuart
Family	MAX 10
Device	10M50DAF484I6G
Timing Models	Final
Total logic elements	5,717 / 49,760 (11 %)
Total registers	3064
Total pins	4 / 360 (1 %)
Total virtual pins	0
Total memory bits	823,744 / 1,677,312 (49 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

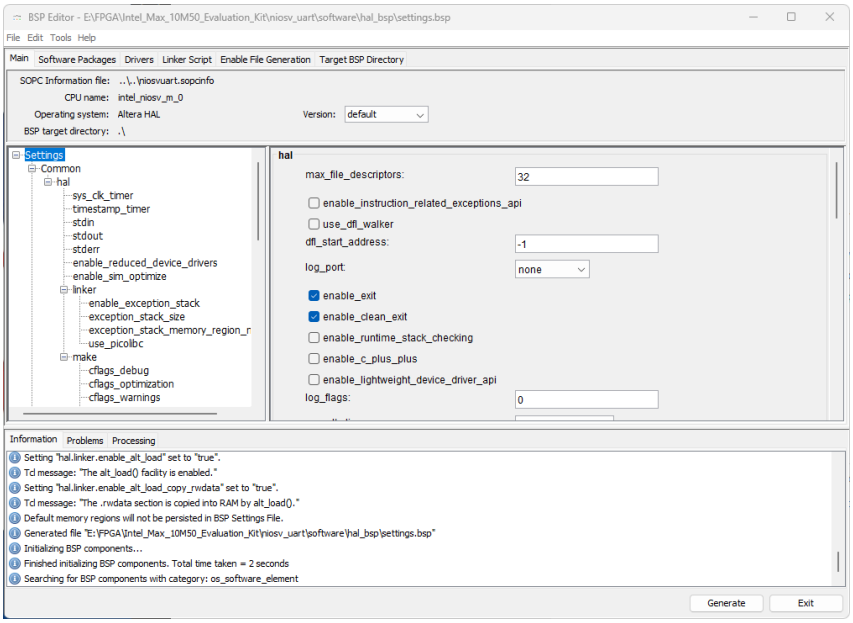
1.2 Part 2 Create BSP and Cmake Application using Visual Studio Code

The BSP has to be created first before the application can be created. Make sure that you have added the “Ashling VS Code Extension for Altera FPGAs” to Visual Studio Code. If you have already worked through the previous two articles using Ashling RiscFree IDE, you will notice some of the details have been woven into the extensions workflow.

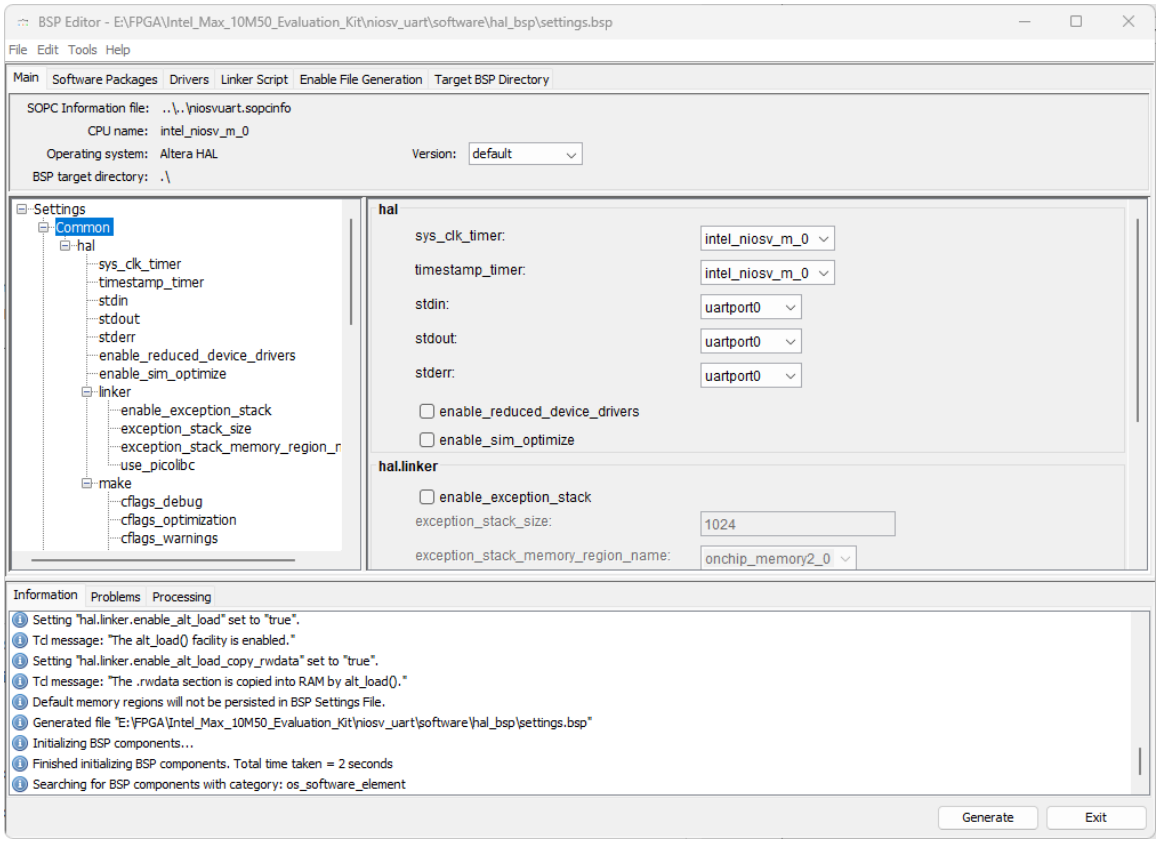
1. Open VS Code
2. In the Activity bar on the right, click on the Ashling icon  .
3. Primary Side bar shows the Ashling Project View with two tools: “Nios V BSP Generator” and “Nios V App Generator”. Click on the “Nios V BSP Generator”.
4. Click Yes to the prompt to open Nios V BSP Editor.
5. The Nios V BSP Editor launches. From the menu, select File-New BSP…
6. Click on the button with 3 dots for “SOPC Information File name:”
7. Open the niosvuart.sopcinfo. As the file opens, the file information is read and fills the rest of the dialog.



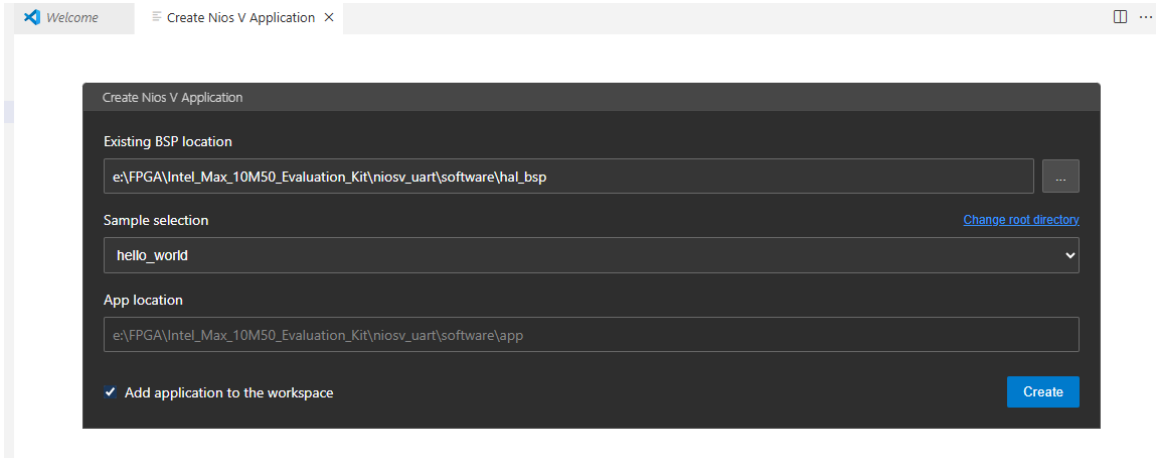
8. Keep the default settings. Click Ok.
9. Under Settings root, uncheck “enable_c_plus_plus”



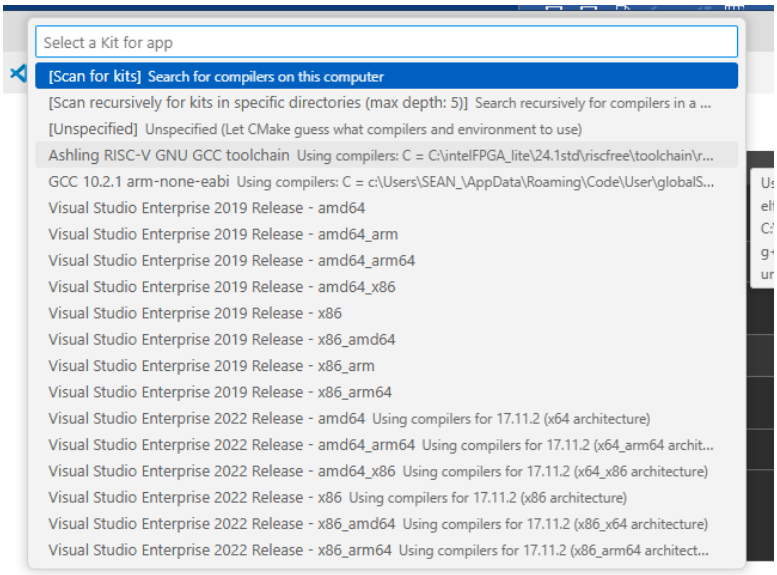
10. Under Common, change the stdin, stdout, and stderr to uartport0.



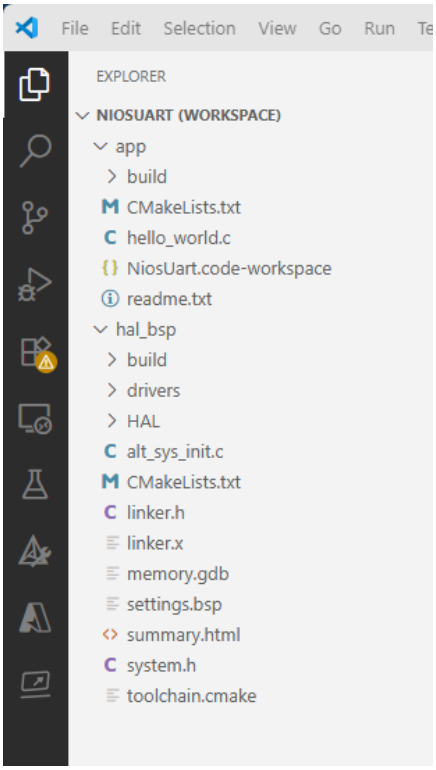
11. Click the Generate button.
12. Once the BSP has been generated, click on Exit to close the BSP Editor.
13. In VS Code, click on Nios V App Generator.
14. A Create Nios V Application appears in the Editor pane.
    a. Point the “Existing BSP location” to the previous generated BSP folder.
    b. Change the Sample selection to hello_world



15. Click Create.
16. Click Yes to the trust diablo.
17. A select Kit for the app appears. Select “Ashling RISC-V GNC GCC Toolchain…”



18. Close the Create Nios V Application tab.
19. Switch to Explorer view and you will see the application and Cmakelist.txt file are created automatically. Open the Cmakelist.txt, and you will see the links to the hal_bsp have been created.
20. Open the hello_world.c file. The application prints a hello message 1000 times.
21. Save the workspace. From the menu, select File->Save Workspace As…
22. Make the name NIOSUART and click save.
23. Let's add the bsp to the workspace. From the menu select File->Add Folder to Workspace.
24. Open the folder to the BSP \niosv_uart\software\hal_bsp. You should see both app and hall projects as part of the workspace.



1.3 Part 3: Write the UART Application in VS Code

Now, we are ready to write the applications in the VS Code.

1. First step is to do a little house keeping. The Ashling extension is nice, but hello_world is a bit out of place. Right click on hello_world.c and Rename the file uart.c.
2. In VS Code, under the app project, open CMakeList.txt.
3. Change the target_sources, hello_world.c to uart.c.

```
:
:

add_executable(app.elf)

target_sources(app.elf
  PRIVATE
    uart.c
)

target_include_directories(app.elf
  PRIVATE
  PUBLIC
)
:
:
```

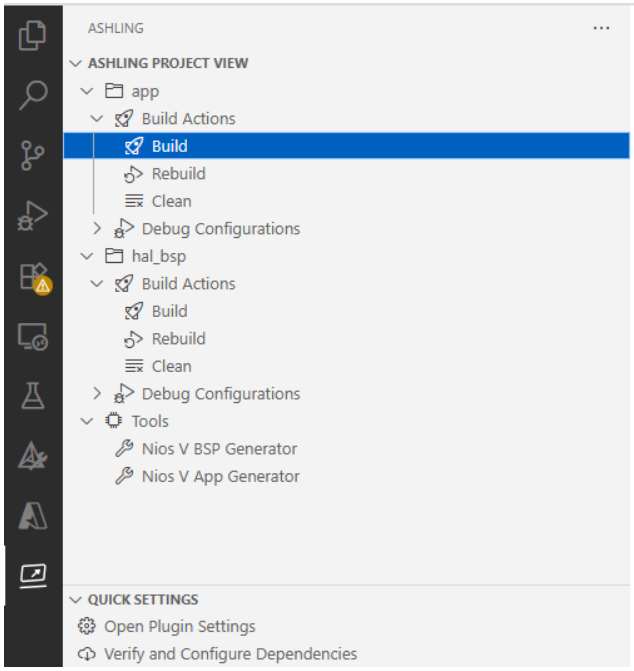
4. Save and close the file.

5. Open uart.c and enter the following code:

```
#include <stdio.h>
#include <alt_types.h>
#include <altera_avalon_uart_regs.h>
#include <altera_avalon_uart.h>
#include <system.h>

void uart_send_char(char c) {
  while (!(IORD_ALTERA_AVALON_UART_STATUS(UARTPORT0_BASE) & ALTERA_AVALON_UART_STATUS_TRDY_MSK));
  IOWR_ALTERA_AVALON_UART_TXDATA(UARTPORT0_BASE, c);
}

char uart_receive_char() {
  while (!(IORD_ALTERA_AVALON_UART_STATUS(UARTPORT0_BASE) & ALTERA_AVALON_UART_STATUS_RRDY_MSK));
  return IORD_ALTERA_AVALON_UART_RXDATA(UARTPORT0_BASE);
}

void uart_send_string(const char *str) {
 while (*str) {
   uart_send_char(*str++);
  }
}

int main() {

  IOWR_ALTERA_AVALON_UART_CONTROL(UARTPORT0_BASE, 0x00); // Disable interrupts

  uart_send_string("Hello from UART example!\n");
  uart_send_string("Enter a character or string:\n");

  while (1) {

    char received = uart_receive_char();
    uart_send_char(received);  // Echo received character
  }

  return 0;
}
```

The UART (RS-232 Serial Port) Intel FPGA IP doesn't allow for Baud rate changes. The only step in main is to disable interrupts. Also, this is a character serial port since there is no buffer like a 16550 UART. The uart_send_char() function sends a character when the transmit bit is ready. uart_receive_char() function when the receive bit is ready. The uart_send_string() function loops to send each character of a string passed to the function out the serial port. After the main() function disables interrupts, two messages are sent out the UART port, and the While-loop waits for characters to arrive and send the characters back out the serial port.

6. Click on the Ashling menu icon to view the Ashling Project View sub-menu.



7. Click on Build. The output will show that the target app.elf file has been built successfully. If there are any errors, make the appropriate corrections and rebuild.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  MEMORY  XRTOS  AZURE  Filter
[build] [100%] Built target niosv-stack-report
[driver] Build completed: 00:00:04.438
[build] Build finished with exit code 0
[proc] Executing command: C:/intelFPGA_lite/24.1std/riscfree/toolchain/riscv32-unknown-elf/bin/riscv32-unknown-elf-gdb.exe --vers
[main] Building folder: e:/FPGA/Intel_Max_10M50_Evaluation_Kit/niosv_uart/software/app/build
[build] Starting build
[proc] Executing command: C:\intelFPGA_lite\24.1std\riscfree\build_tools\cmake\bin\cmake.exe --build e:/FPGA/Intel_Max_10M50_Eval
--config Debug --target all -j 16 --
[build] [ 96%] Built target hal2_bsp
[build] [ 96%] Building C object CMakeFiles/app.elf.dir/uart.c.obj
[build] [ 97%] Linking C executable app.elf
[build] [ 97%] Built target app.elf
[build] [ 98%] Creating app.elf.objdump.
[build] [ 99%] Reporting memory available for stack + heap in app.elf.
[build] [100%] Creating onchip_memory2_0.hex.
[build] TBBmalloc: skip allocation functions replacement in ucrtbase.dll: unknown prologue for function _msize
[build] [100%] Built target create-objdump
[build] app.elf
[build]  * 32.70 KB - Program size (code + initialized data).
[build]  * 137.04 KB - Free for stack + heap.
[build] [100%] Built target create-hex
[build] [100%] Built target niosv-stack-report
[driver] Build completed: 00:00:05.169
[build] Build finished with exit code 0
```

1.4 Part 4: Test the Design and the Application

With the FPGA design and UART application compiled, we can now test the design and application on the board.

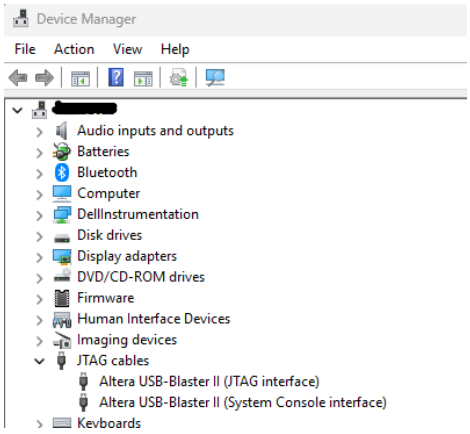
1.4.1 Hardware Setup

It is assumed that 0.1 inch headers have been soldered to J14. The FPGA design has USER_IO4 as UART TXD and USER_IO3 (PIN 9) as UART_RXD (PIN13).

1. Connect the USB to UART Green (RXD) to USER_IO4.
2. Connect the USB to UART White (TXD) to USER_IO3.
3. Connect the USB to UART Black (GND) to PIN 11 GND.

The 10M50 Evaluation kit comes with a USB cable with red and black USB connectors. The black connector is for the JTAG. The red connector is only for power. JTAG can be over USB or through the 10-PIN JTAG connector. JTAG over USB will be used for this example.

4. Switch 4 on switch block 2 (SW2.4) enables/ disables the USB JTAG. The default mode is off. Switch SW2.4 to On.
5. Since this will be powered by USB make sure jumper 11 (J11) has pins 2 and 3 connected.
6. Connect the USB cable to the 10M50 Evaluation kit and the red and black connectors to your development system that has Quartus installed.
7. Open Device Manager, and you should see the Altera USB Blaster II in the device list. If the device doesn't show up makes sure you installed the driver that comes with the kit.

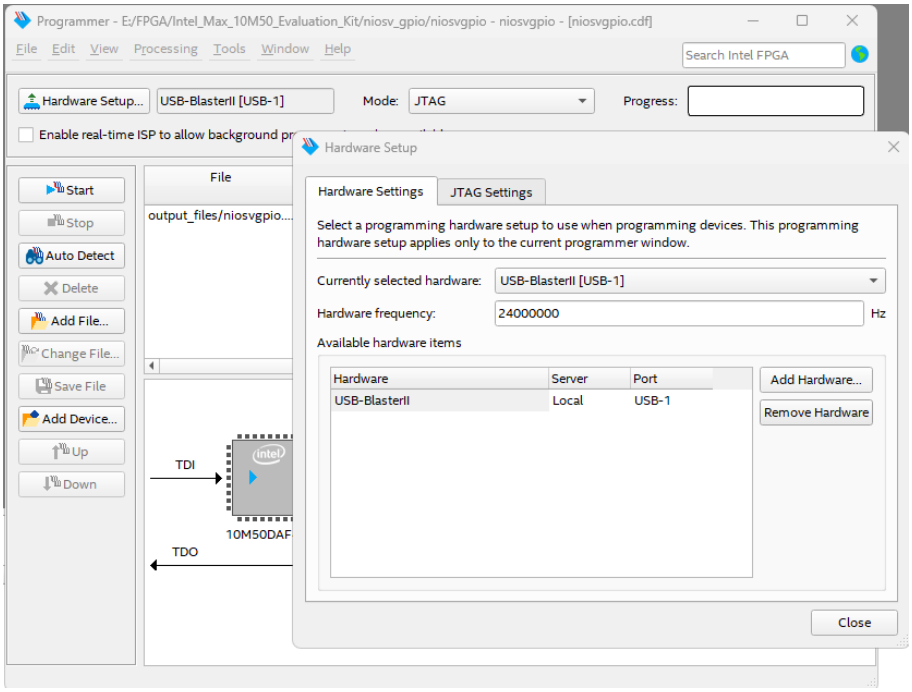


8. Plug the USB to UART USB connector into the development computer.
9. In device manager, the USB to UART should show up as a COM port.

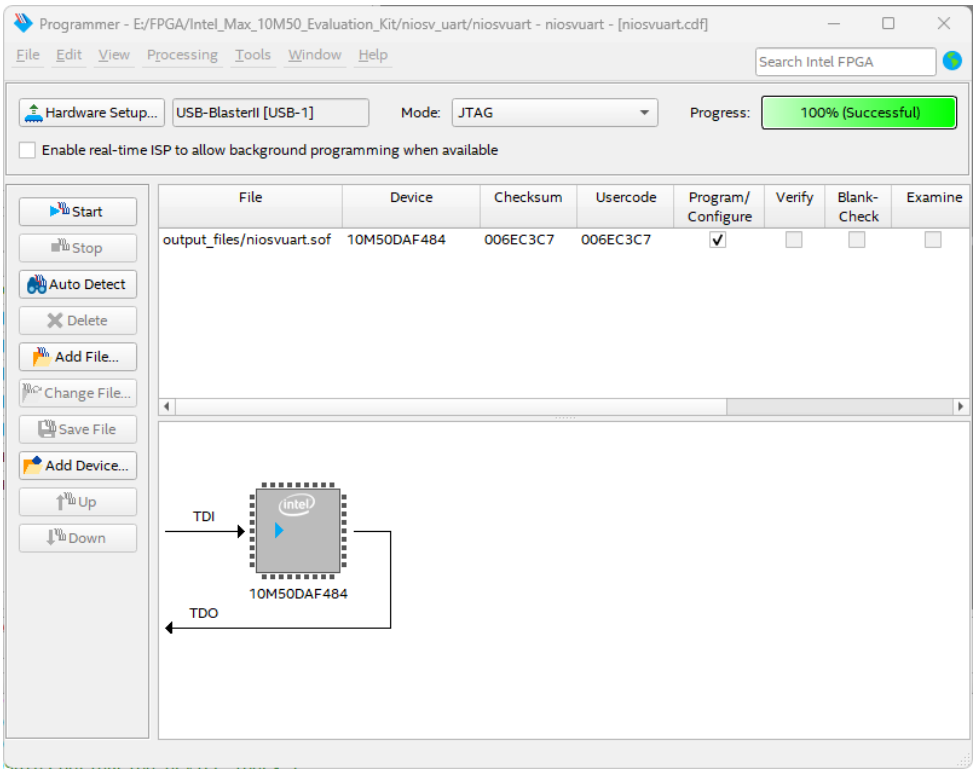
1.4.2 Program the Board

We will go back to Quartus and program the board with the design.

1. 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.
2. The Programmer dialog appears, click on the "Hardware Setup" button.
3. Click the Add hardware button, select the Hardware type, and fill in any remaining information, and click OK.



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

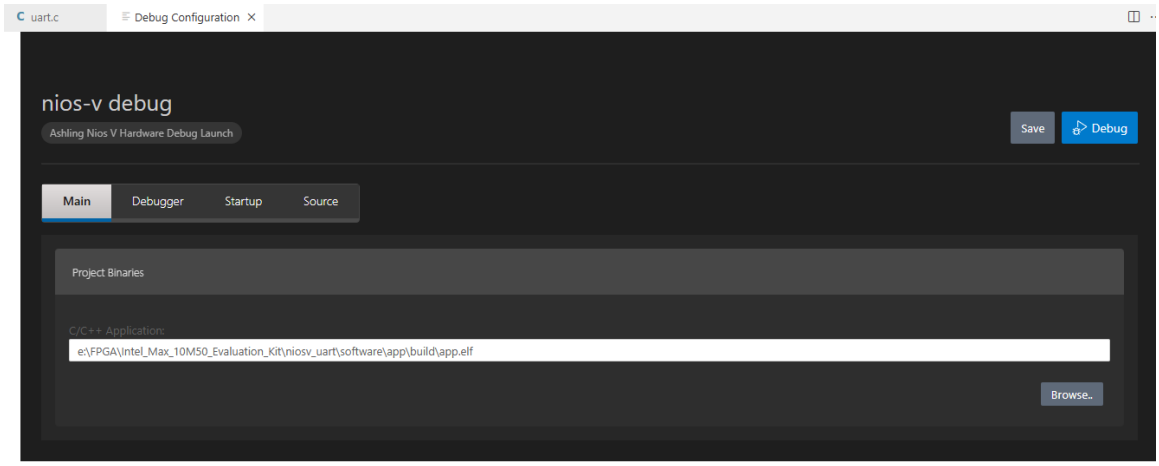


Notice that there is no time constraint dialog box popping up. The Nios V is free and doesn't required a paid license.

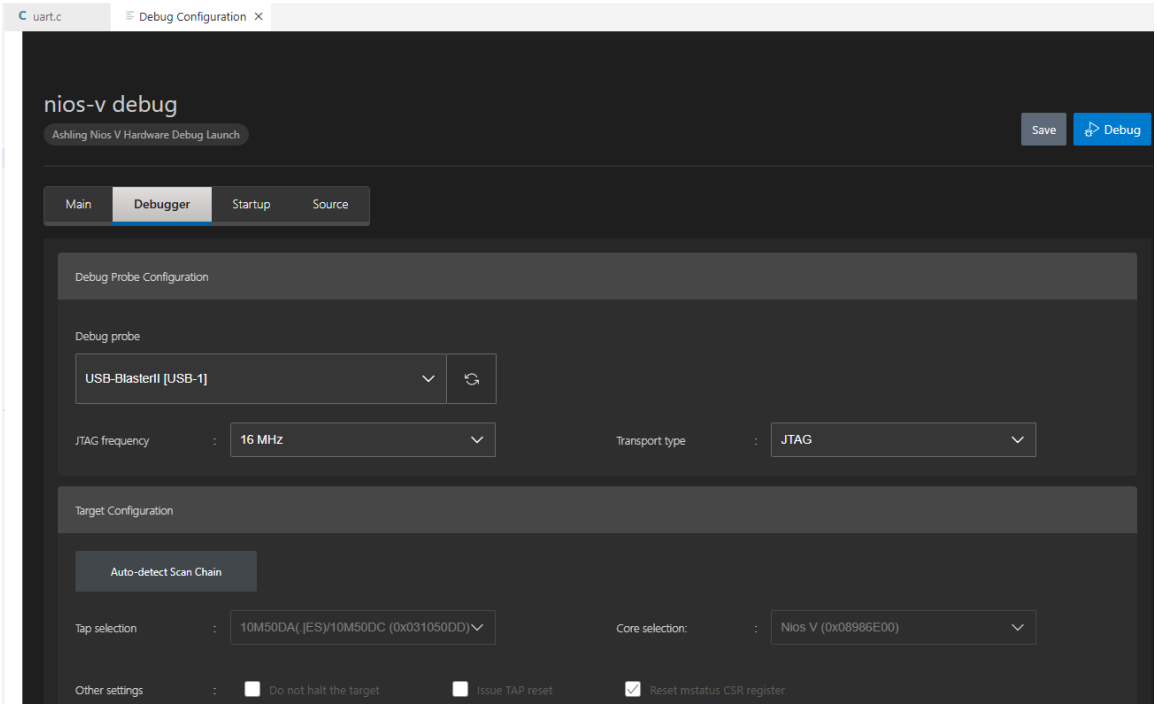
1.4.3 Debug the Application

Now, we go back to the VS Code to download and debug the application

1. With the JTAG cable connected, in VS Code, click on the Ashling menu icon to bring up the Ashling Project View submenu
2. Under the app folder, click on Debug Configurations.
3. The system will ask for what type of debugging, select Nios V Hardware Debugging.
4. Give the debug configuration a name and hit enter
5. One the main page, set the path to the app.elf file.



6. Click on the Debugger button,
7. Set the Debug probe to USB-Blaster II and leave the default JTAG frequency at 16 MHz.
8. Click the Auto-detect Scan Chain so the correct Tap selection and core selection get filled in.

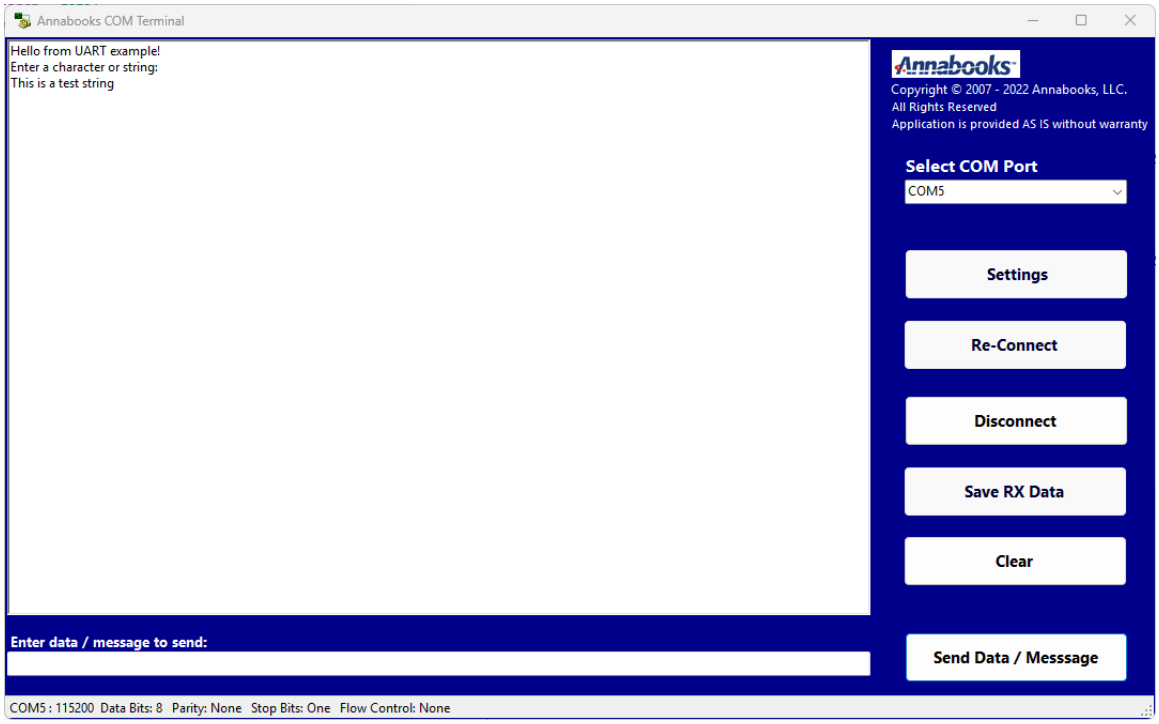


9. Save the configuration and click on the Debug to start the debug.

Note: With VS Code, the debugger can fail to connected to the JTAG debugger. You will have to re-try debugging if the connection is not made.

10. The debugger will stop at the first line in main() and the debug controls  appear at the top of the screen.

11. Open a terminal program like Annabooks COM Terminal and connect the USB to UART COM port.
12. Step through the code and you will see the messages out the terminal.
13. Click continue and send a string from the terminal program. The program should echo the string back out the UART port.



14. Stop debugging when finished.

1.5 Summary: Better Application Development Experience with VS Code

If you worked through the previous two articles, you can see the steps to create a project, create the BSP, and debug the project have been integrated into the extension. The VS Code extension provides a shell to the Ashling RiscFree IDE that removes the need to drop down to the command line to create the project and BSP. Compared to the Ashling RISC Free IDE based on Eclipse CDT, VS Code provides a better NIOS V applications development workflow. Debugging has a little glitch connecting sometimes, but once the debugger is running, debugging is a breeze. Most important, there is no need to change a perspective.