Nios® V with GPIOs Example on MAX® 10 FPGA 10M50 Evaluation Kit

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

January 2026

The last article on the Nios V processor covered a basic hello world application running on the MAX 10-10M08 Evaluation Kit. Since Nios V HAL is very big and MAX 10-10M08 Evaluation Kit has limited memory capacity, the 10M50 Evauation kit will be used moving forward. In this specific article, we will explore GPIOs and interrupts with the Nios V.

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 Quartus 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.
- 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.

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.

- The first step is still the same: creating the hardware design in Quartus Prime and Platform Builder.
- The second step is to use the Nios V command line to create the BSP and the cmake project.
- The final step is to write and debug the application with RiscFree IDE.

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.

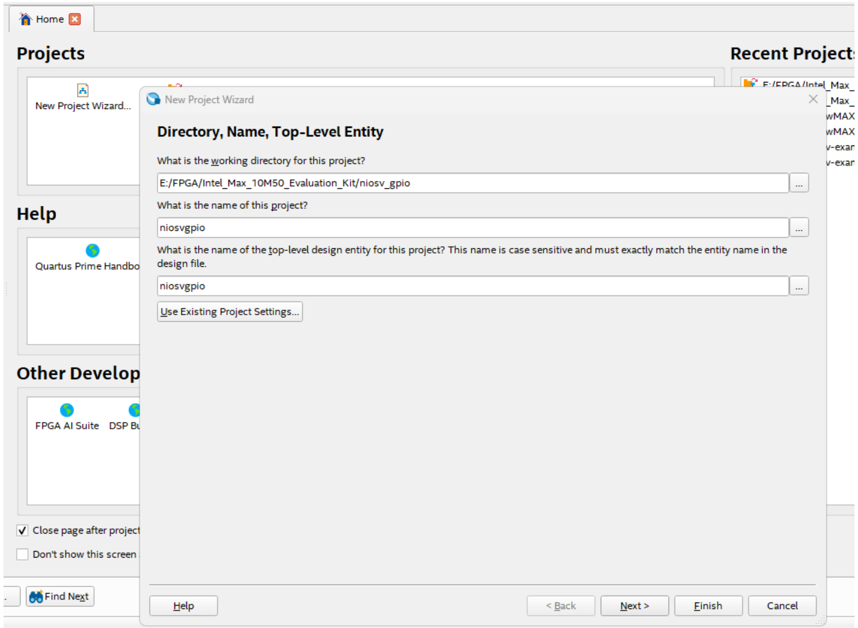
1.1 Part 1: Basic Nios V Design

For this design, we will have a Nios V/m processor IP block, along with an onboard RAM IP block, two GPIO IP blocks, and a JTAG UART IP block. The application will have a button that triggers an intererupt to toggle the 5 leds on the board. The JTAG interface will be used to send messages to a console application.

1.1.1 Create the Project

The first step is to create a design project.

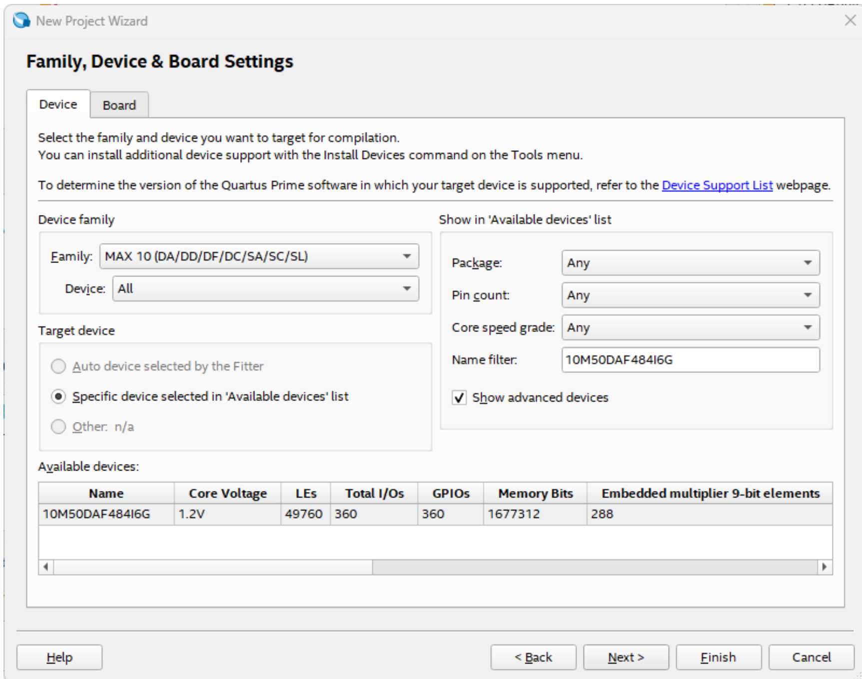
1. Open Quartus.
2. Click on the New Project Wizard.



3. Select or create a project directory \niosv_gpio (Do not use the Quartus installation directory) and name the project: “niosvgpio”. 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.

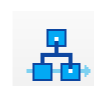
4. Project Type: Empty project, click Next.
5. Add File: no files to add, click Next.
6. Family, Device & Board Settings:
    a. Change the Family to “MAX 10(DA/DD/DF/DC/SA/SC/SL)”
    b. Put 10M50DAF484I6G in the Name filter text box. This should narrow down the list to a single device.



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

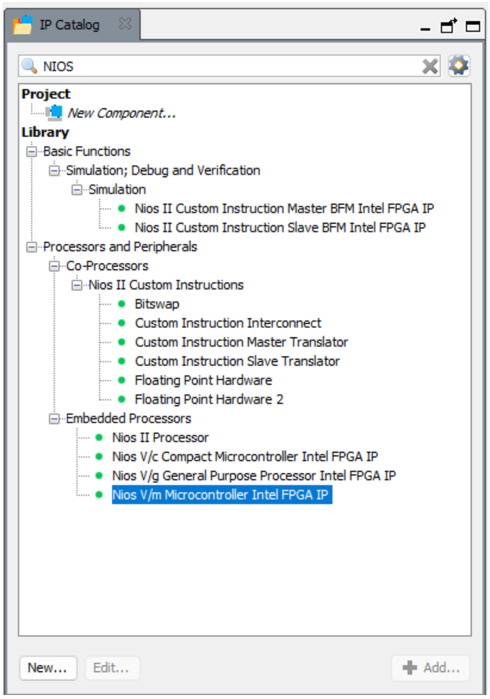
Quartus supports many design types to create an FPGA design. The Platform Designer tool will be used for this hands-on exercise. Platform Builder makes it easy to add already-built IP blocks and interconnect them.



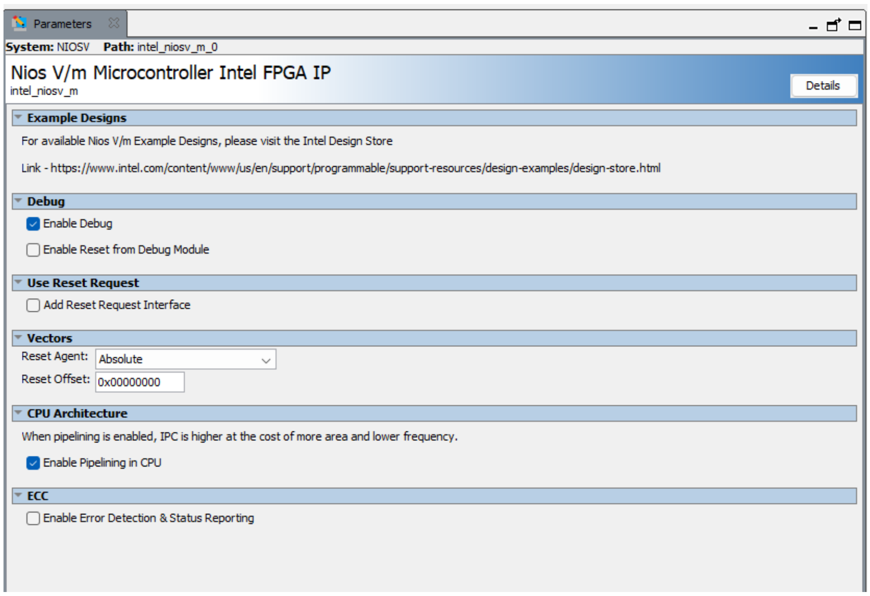
1. From the menu, select Tools->Platform Designer, or the Platform Designer icon from the toolbar.

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.

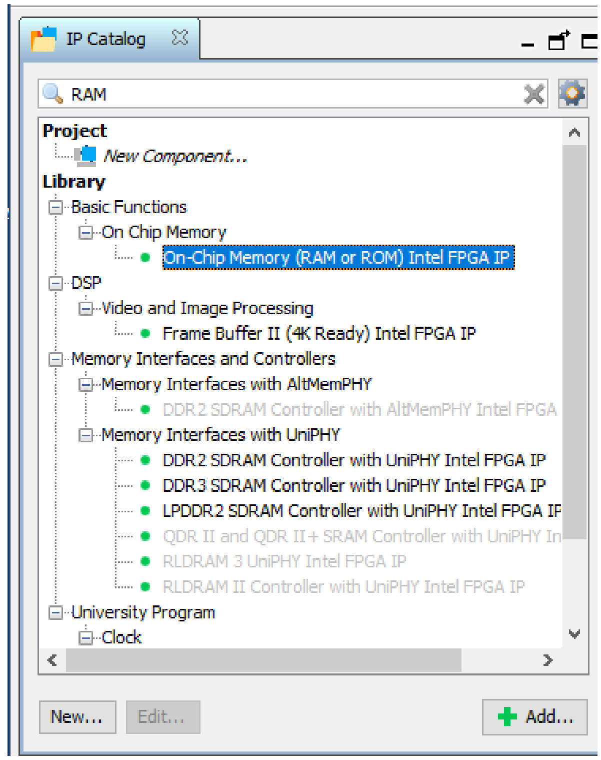
2. The top left pane contains the IP Catalog with all the available IP blocks that come with Quartus Prime. In the search box, type nios.



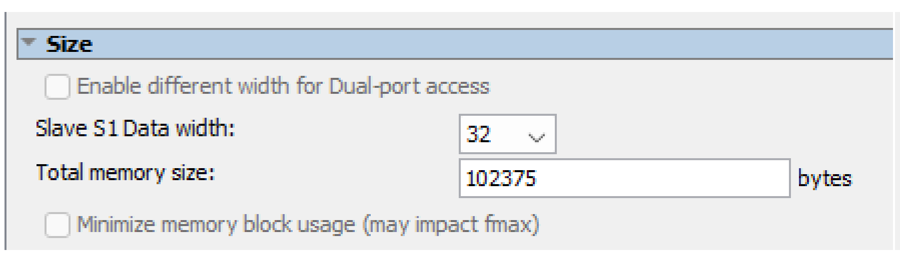
3. Expand the Processors and Peripherals and the Embedded Processors branches. Under Embedded Processors double-click on the Nios V/m Processor Intel FPGA IP.
4. This will open the Nios V/m Configuration page. We will keep the defaults for now. Click Finish.



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.

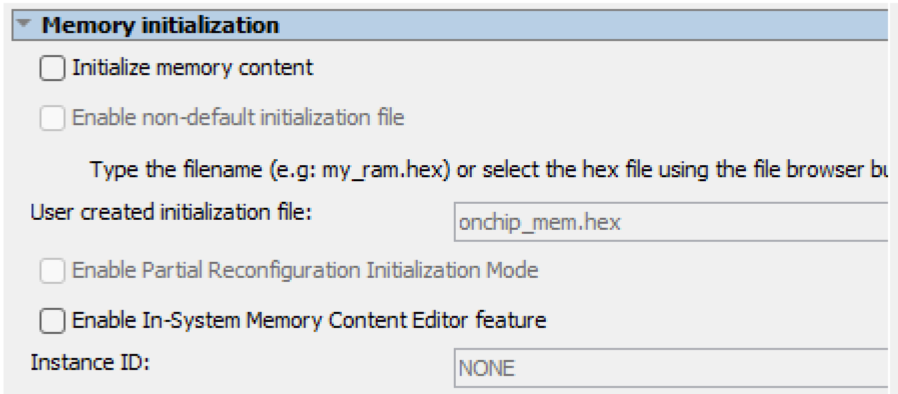


7. The configuration page will appear. Change the Total memory size to 102375. This is 819,000 bits which is about have the available memory.

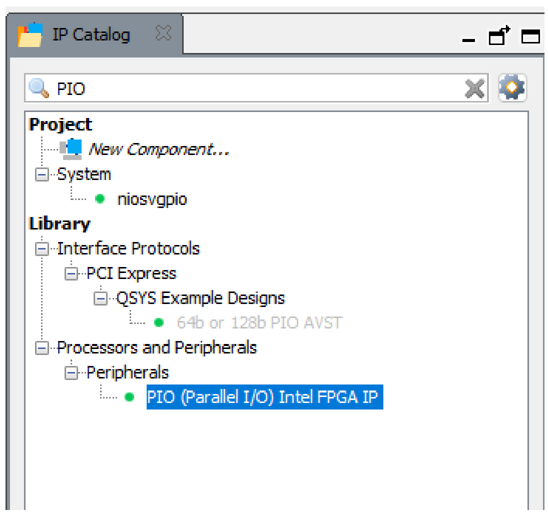


Intel Max 10 Part Number	Logic Elements	Maximum Embedded Memory	Maximum User I/O Count
10M02	2000	108 Kbits	246
10M04	4000	189 Kbits	246
10M08	8000	378 Kbits	250
10M16	16000	549 Kbits	320
10M25	25000	675 Kbits	360
10M40	40000	1.26 Mbits	500
10M50	**50000**	**1.638 Mbits**	**500**

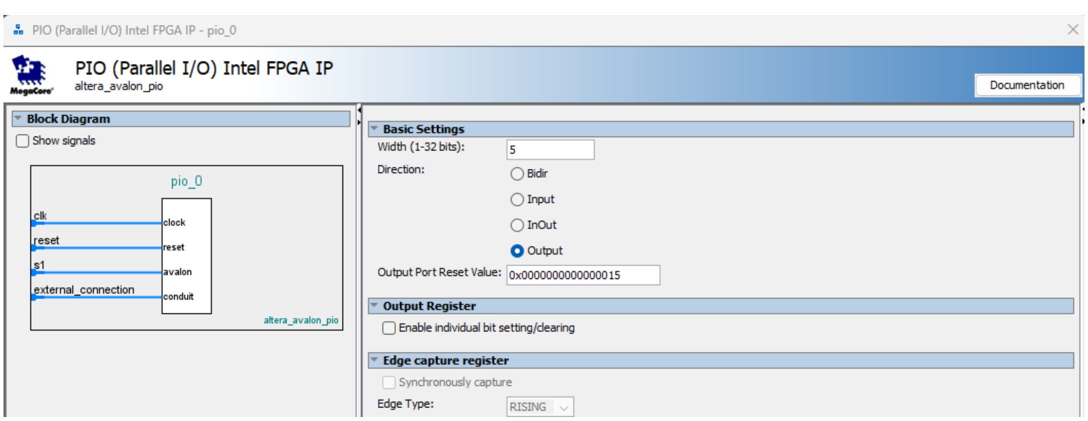
8. Uncheck the box for “Initialize memory content”, and click Finish.



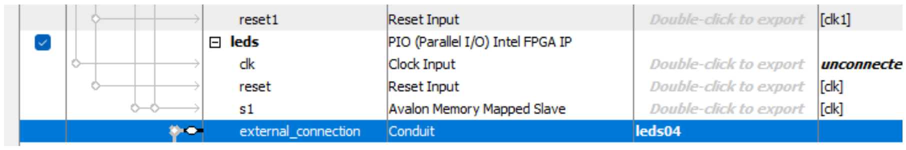
9. In the IP Catalog search, enter pio.
10. Double-click on the “PIO (Parallel I/O) Intel FPGA IP”.



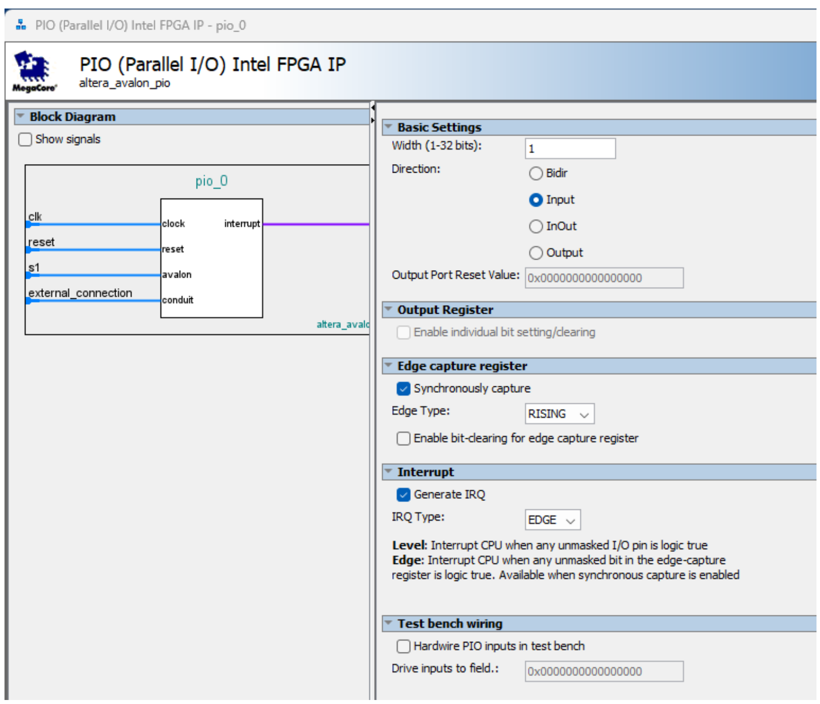
11. This GPIO block will be used for the LEDs.
    a. Set the Width to 5
    b. Set the Direction to Output
    c. Set the Output port Reset Value to 0x0000000000000015.



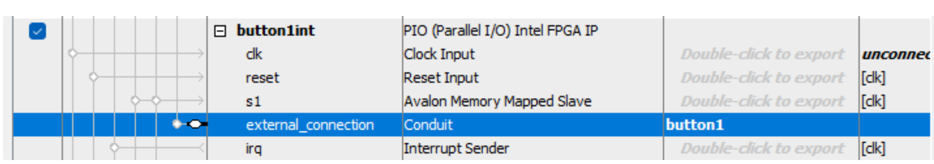
12. Click Finish.
13. In the System Contents tab, change the name of the pio_0 to leds.
14. In the Export column, double click on the line for “external_conenction.
15. Enter leds04.



16. Double-click on the “PIO (Parallel I/O) Intel FPGA IP”.
17. This GPIO block will be used for the single button interrtupt.
    a. Set the Width to 1
    b. Set the Direction to Input
    c. Check the box next to “Synchornously capture”
    d. Set the Edge Type to RISING
    e. Check the box next to Generate IRQ
    f. Set the IRQ type to EDGE.

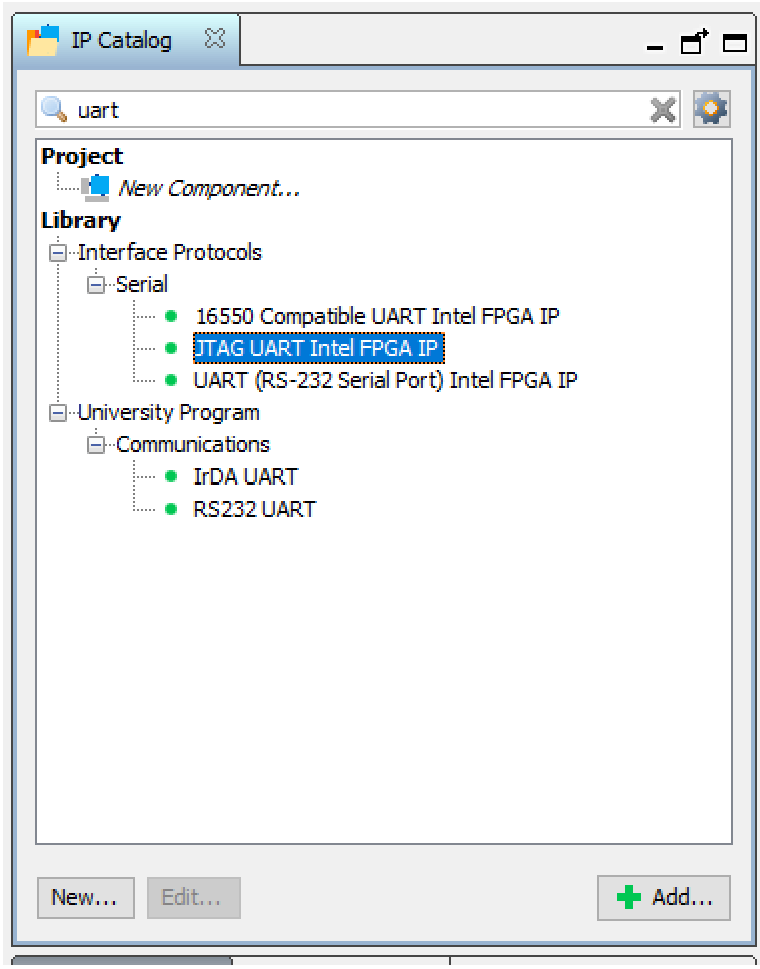


18. Click Finish.
19. In the System Contents tab, change the name of the pio_0 to button1int.
20. In the Export column, double click on the line for “external_conenction”.
21. Enter button1.

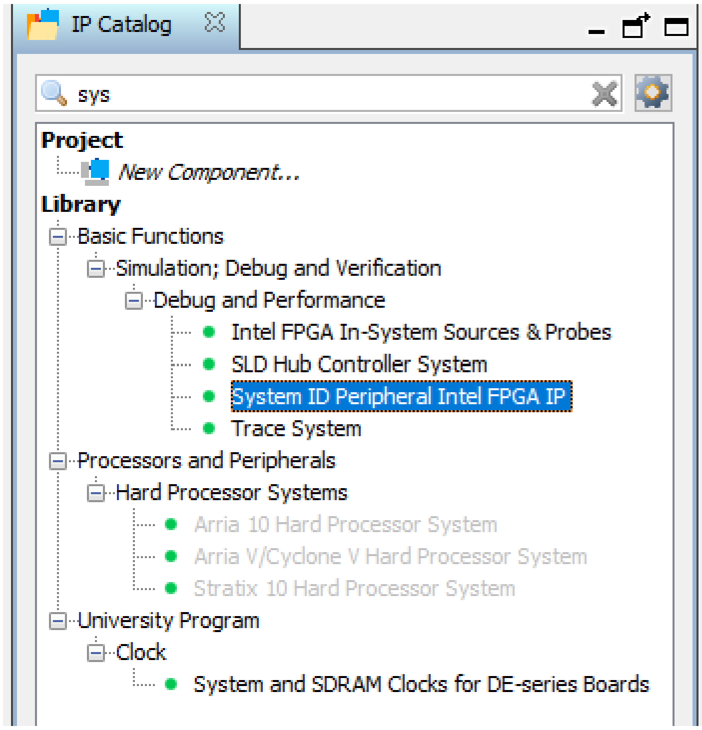


22. In the IP Catalog search, enter uart.

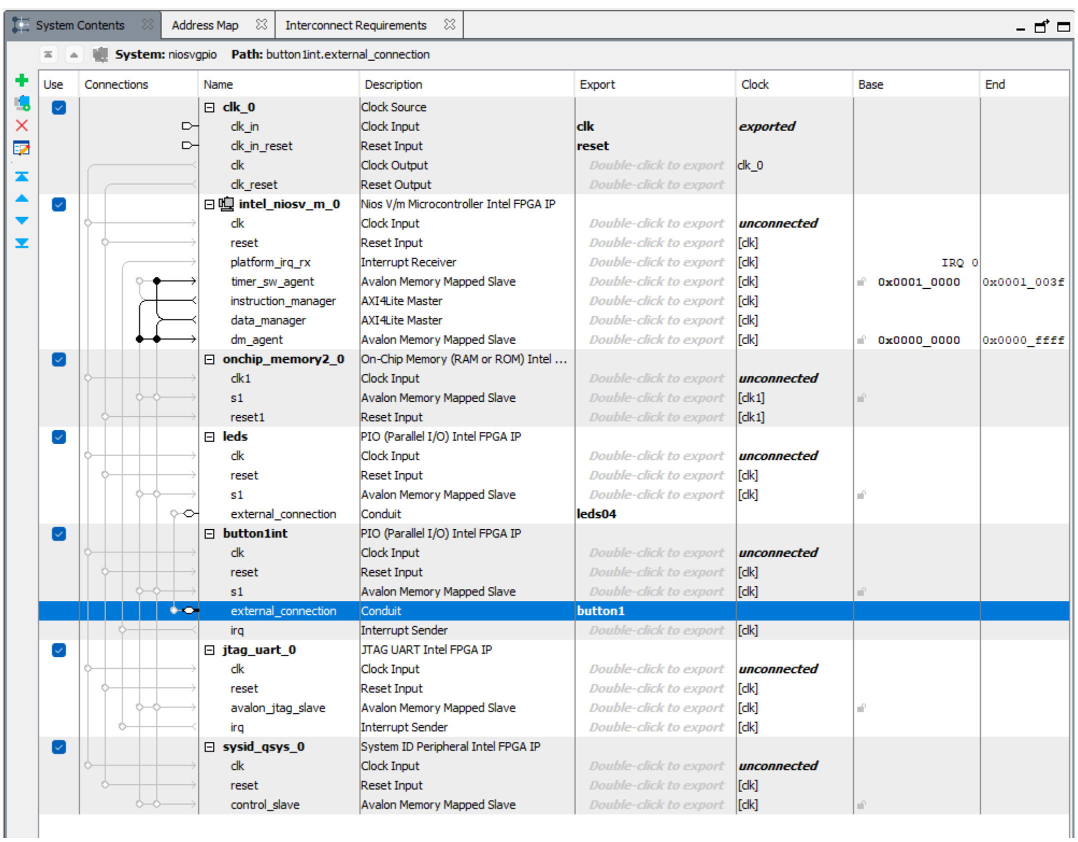
23. Double-click on the JTAG UART Intel FPGA IP.



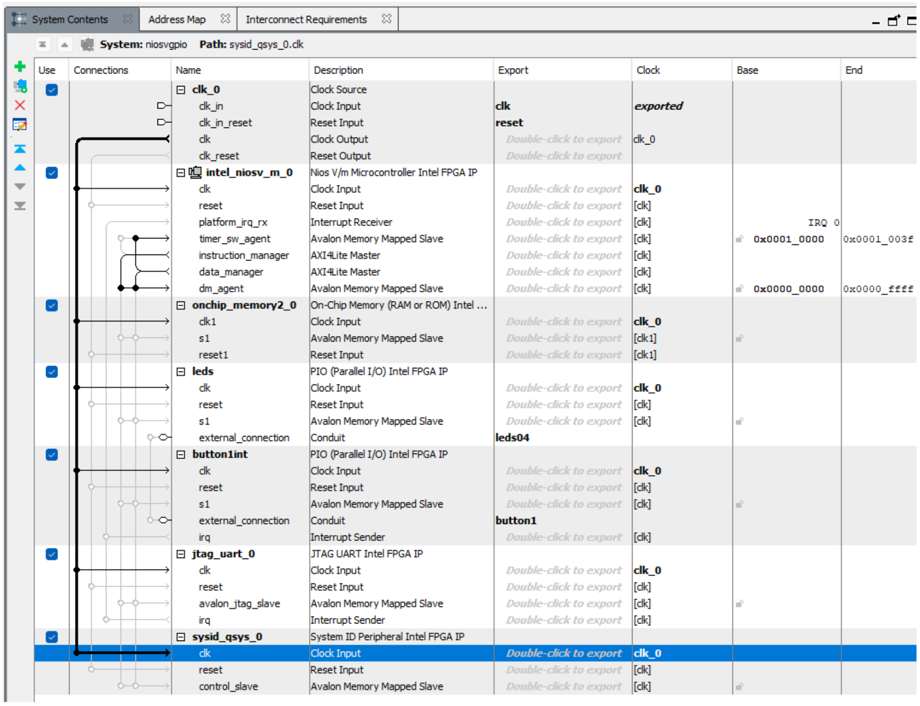
24. A configuration page will appear. There are no changes to be made. Click Finish.
25. In the IP Catalog search, enter the system ID.
26. Double-click on the System ID Peripheral Intel FPGA IP.



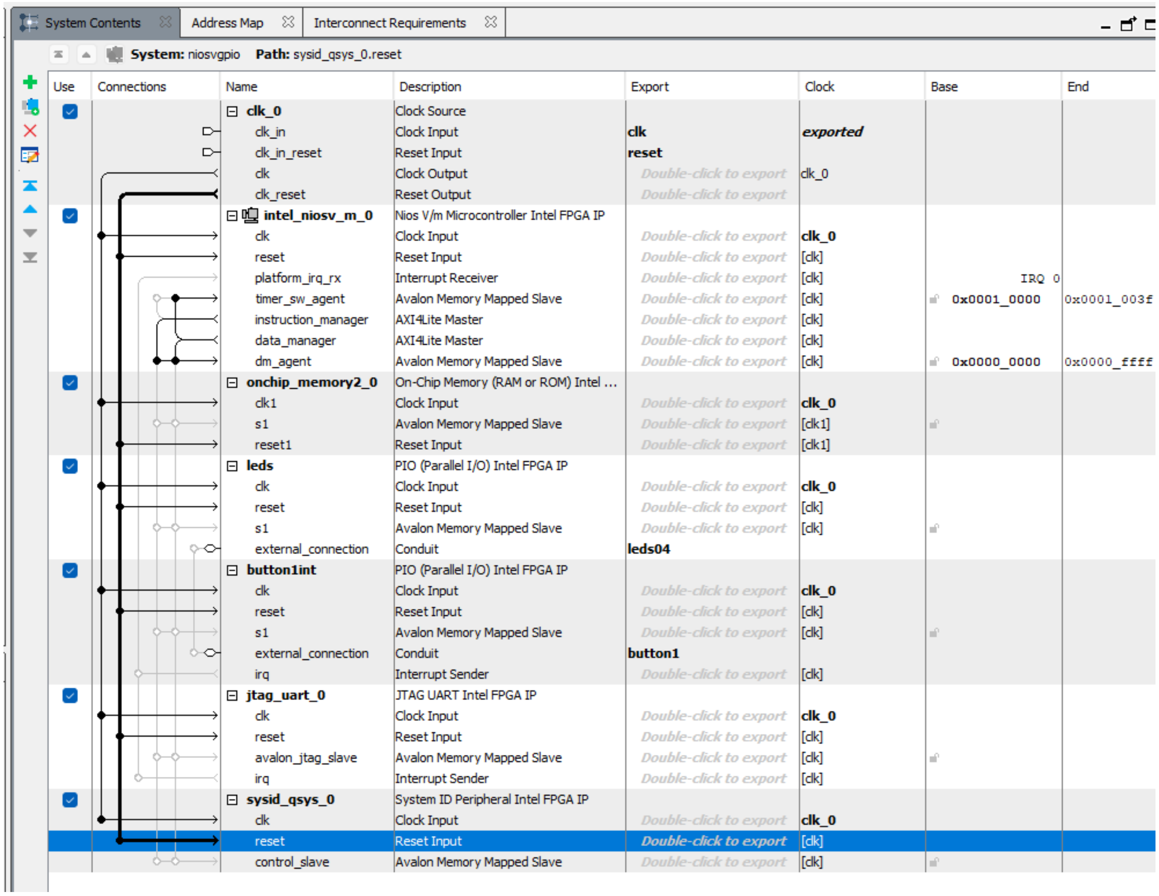
27. A configuration page will appear. There are no changes to be made. Click Finish.



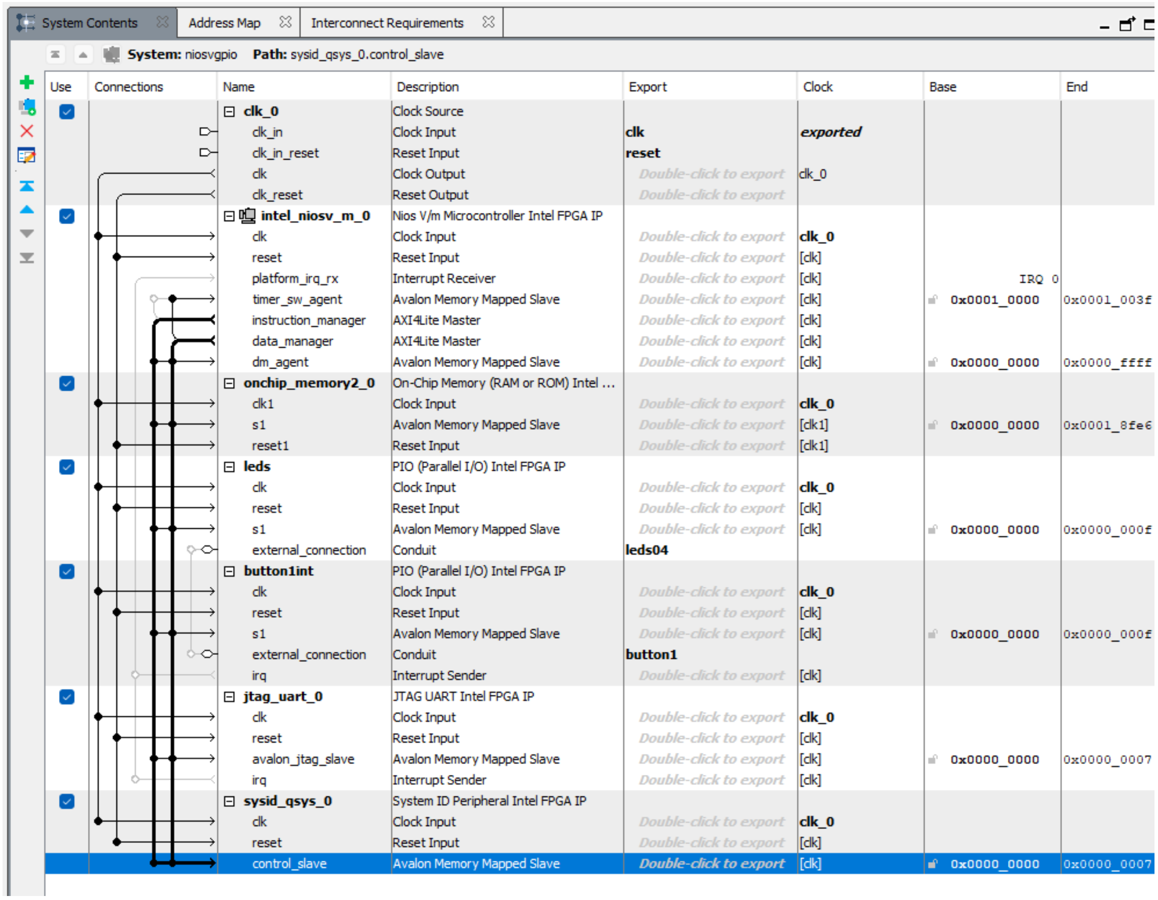
28. Now, we need to wire the IP blocks together. First, wire all the clk lines together by clicking on the dots for all IP blocks.



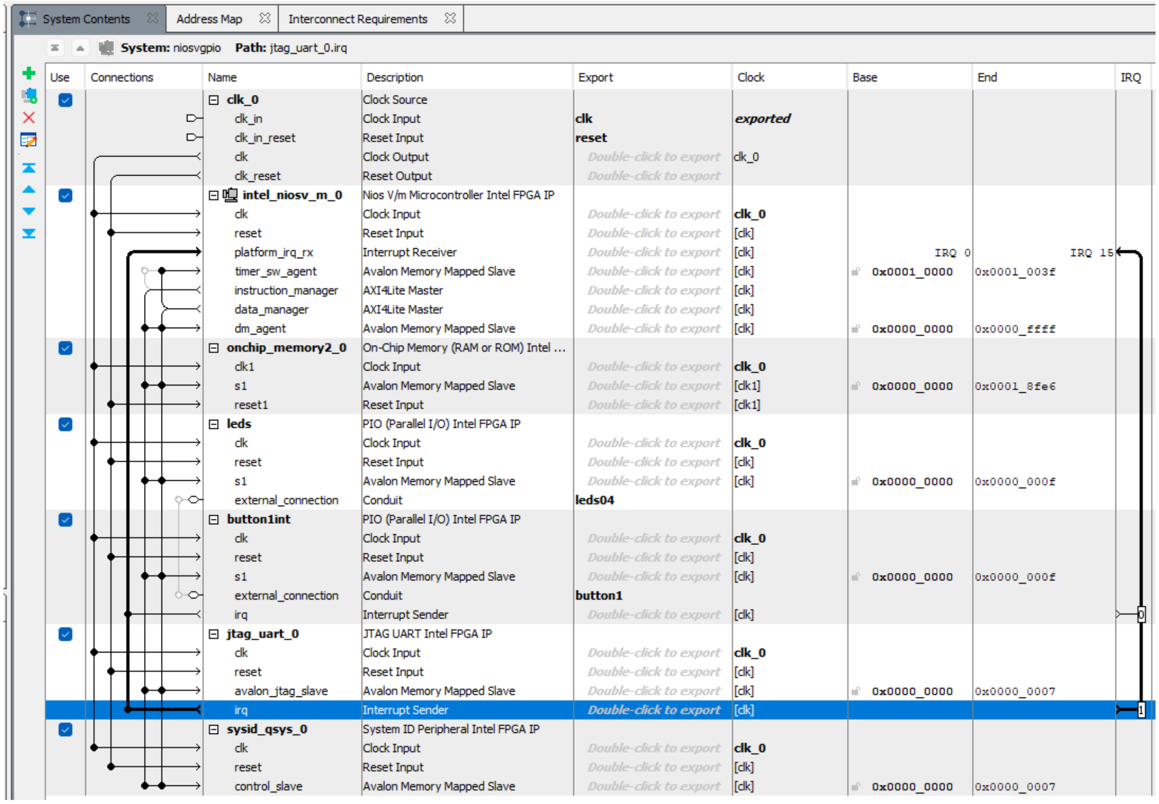
29. Next, connect all the reset lines together by clicking on the dots for all IP blocks.



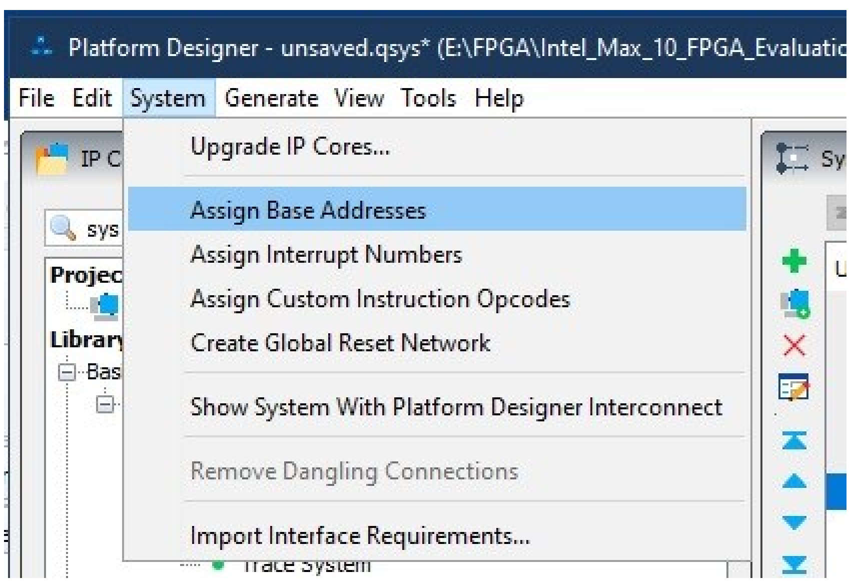
30. The memory lines have to be connected together. Connected the Instruction_manager and data_manager to all other items in the design.



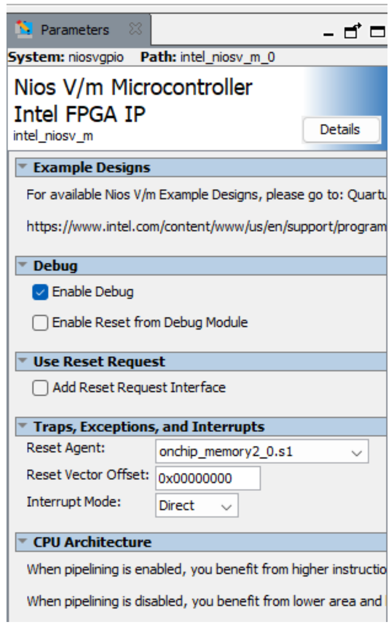
31. Finally, connect the jtaq_uart and button1int irq lines to the intel_niosv_m_0.
32. If you scroll to the right, the irqs are given a default value.



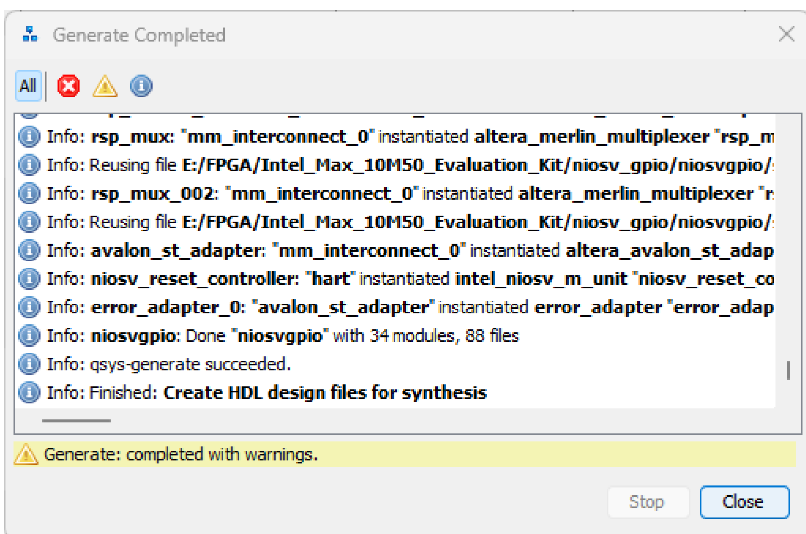
33. Let's assign a base address. From the menu, select System->Assign Base Address. This will remove a number of errors from the message box.



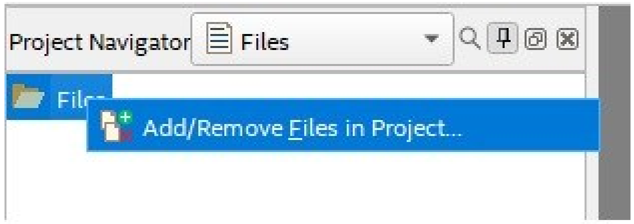
34. Finally, let's set the reset and exception vector addresses. Double-click on the intel_niosv_m_0 to open the configuration page.
35. In the Vectors section, change the Reset Agent to onchip_memory2_0.s1



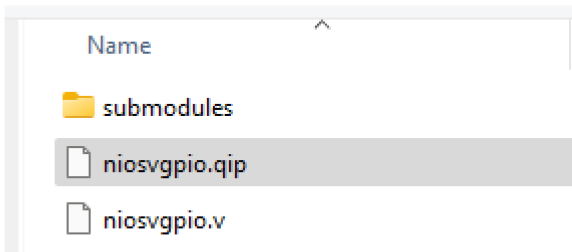
36. Save the design as niosvgpio.qsys.
37. Once the save has been completed, click Close.
38. Click on Generate HDL…
39. A dialog appears, keep the defaults and click the Generate button.
40. The generate process kicks off. The processes should succeed with warnings, click Close.



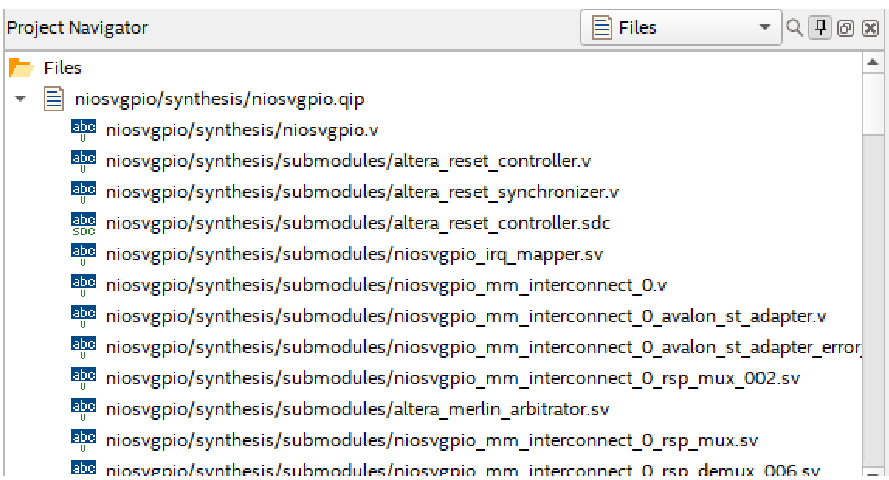
41. Click Finish to close the design.
42. Quartus then reminds you to add the new design to the project. Click Ok.
43. In the Project Navigator, click on the drop-down and select Files.
44. Right-Click on Files and select Add/Remove Files in Project.



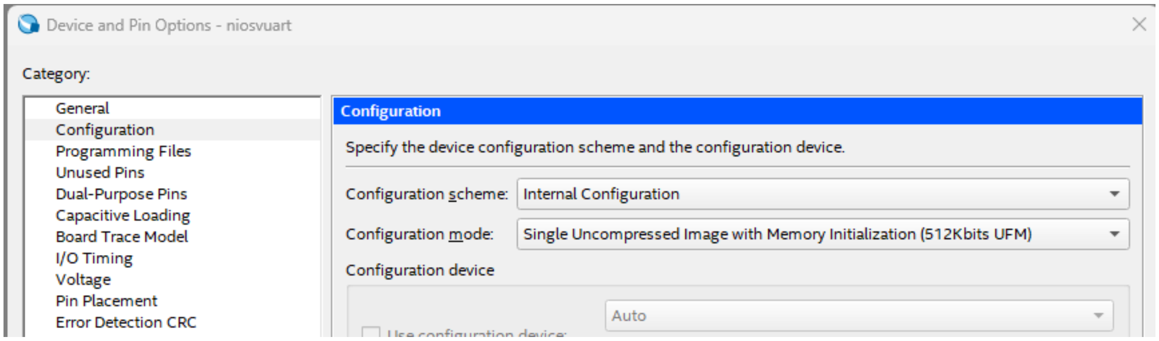
45. A Settings – NIOSVCPU page appears with Files on the left highlighted. Click the three dots, browse button for File name, and navigate to \niosv_gpio\niosvgpio\synthesis\NIOSV_Example\NIOSV\synthesis folder.
46. Click on niosvgpio.qip file and click open



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

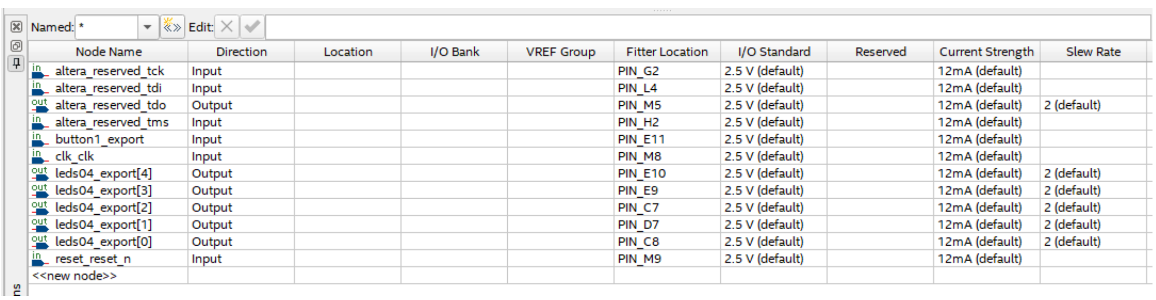


48. Save the project.
49. Click on Assignments->Device
50. Click the “Device and Pin Options…” button.
51. Change the Configuration mode to “Single Uncompressed Image with Memory Initialization (512kbits UFM)”.



52. Click Ok.
53. Click Ok again.
54. In the project navigator, right click on NIOSV.qip and select “Set as Top-Level Entity”.
55. 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.

56. 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.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
altera_reserved_tck	Input				PIN_G2	2.5 V (default)		12mA (default)	
altera_reserved_tdi	Input				PIN_L4	2.5 V (default)		12mA (default)	
altera_reserved_tdo	Output				PIN_M5	2.5 V (default)		12mA (default)	2 (default)
altera_reserved_tms	Input				PIN_H2	2.5 V (default)		12mA (default)	
button1_export	Input				PIN_E11	2.5 V (default)		12mA (default)	
clk_clk	Input				PIN_M8	2.5 V (default)		12mA (default)	
leds04_export[4]	Output				PIN_E10	2.5 V (default)		12mA (default)	2 (default)
leds04_export[3]	Output				PIN_E9	2.5 V (default)		12mA (default)	2 (default)
leds04_export[2]	Output				PIN_C7	2.5 V (default)		12mA (default)	2 (default)
leds04_export[1]	Output				PIN_D7	2.5 V (default)		12mA (default)	2 (default)
leds04_export[0]	Output				PIN_C8	2.5 V (default)		12mA (default)	2 (default)
reset_reset_n	Input				PIN_M9	2.5 V (default)		12mA (default)	
<<new node>>									

57. Using the board schematic, locate the pins for the button S1, 50Mhz clock, User LEDS, JTAG, 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
button1_export	PIN_R20	1.2V
clk_clk	PIN_J10	3.3-V LVCMOS
leds04_export[4]	PIN_C7	3.3-V LVTTL
leds04_export[3]	PIN_D5	3.3-V LVTTL
leds04_export[2]	PIN_C5	3.3-V LVTTL
leds04_export[1]	PIN_C4	3.3-V LVTTL
leds04_export[0]	PIN_C3	3.3-V LVTTL
reset_reset_n	PIN_D9	3.3-V LVTTL

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservatio
altera_reserved_tck	Input	PIN_G2	1B	B1_N0	PIN_G2	2.5 V Sc... Trigger		12mA (default)			
altera_reserved_tdi	Input	PIN_L4	1B	B1_N0	PIN_L4	2.5 V Sc... Trigger		12mA (default)			
altera_reserved_tdo	Output	PIN_M5	1B	B1_N0	PIN_M5	2.5 V (default)		12mA (default)	2 (default)		
altera_reserved_tms	Input	PIN_H2	1B	B1_N0	PIN_H2	2.5 V Sc... Trigger		12mA (default)			
button1_export	Input	PIN_R20	5	B5_N0	PIN_E11	1.2 V		12mA (default)			
clk_clk	Input	PIN_J10	8	B8_N0	PIN_M8	3.3-V LVCMOS		2mA (default)			
leds04_export[4]	Output	PIN_C7	8	B8_N0	PIN_E10	3.3-V LVTTL		8mA (default)	2 (default)		
leds04_export[3]	Output	PIN_D5	8	B8_N0	PIN_E9	3.3-V LVTTL		8mA (default)	2 (default)		
leds04_export[2]	Output	PIN_C5	8	B8_N0	PIN_C7	3.3-V LVTTL		8mA (default)	2 (default)		
leds04_export[1]	Output	PIN_C4	8	B8_N0	PIN_D7	3.3-V LVTTL		8mA (default)	2 (default)		
leds04_export[0]	Output	PIN_C3	8	B8_N0	PIN_C8	3.3-V LVTTL		8mA (default)	2 (default)		
reset_reset_n	Input	PIN_D9	8	B8_N0	PIN_M9	3.3-V LVTTL		8mA (default)			
<<new node>>											

58. Close the Pin Planner when finished.
59. Save the project.

1.1.3 Add Design Constraints File

1. From the menu in Quartus, select File->New-> Synopsys Design Constraings 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 leds04*]
set_false_path -from [get_ports button1*]
```

4. Save the file as niosvgpio.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 - Mon Sep 1 21:00:24 2025
Quartus Prime Version	24.1std.0 Build 1077 03/04/2025 SC Lite Edition
Revision Name	niosvgpio
Top-level Entity Name	niosvgpio
Family	MAX 10
Device	10M50DAF484I6G
Timing Models	Final
Total logic elements	5,582 / 49,760 (11 %)
Total registers	2995
Total pins	8 / 360 (2 %)
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: Nios V Shell Create BSP and Cmake Application

This section will create the BSP and basic application project, and the following section will be used to create the application in RiscFree IDE.

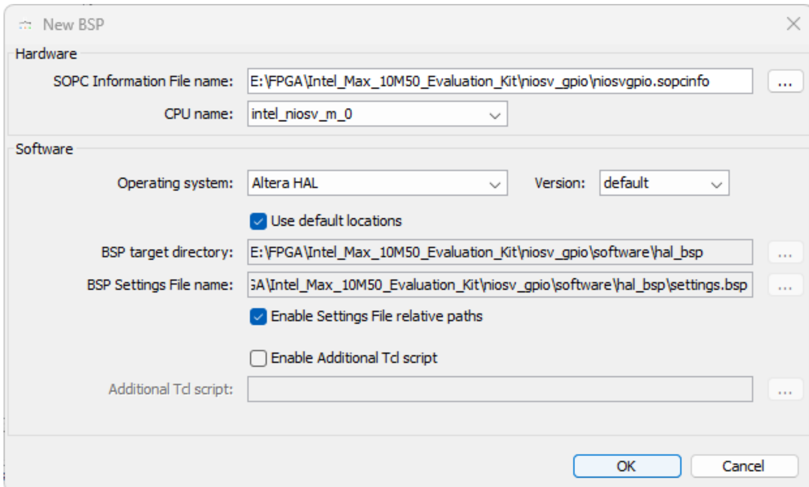
1. From the Start menu, open the Nios V Command Shell.
2. A command window appears. Change directory to C:\altera_lite\25.1std\niosv\bin if the window doesn't open to the folder.
3. Enter DIR and hit enter. You will see a number of applications available.

Nios V Command Shell (Quart

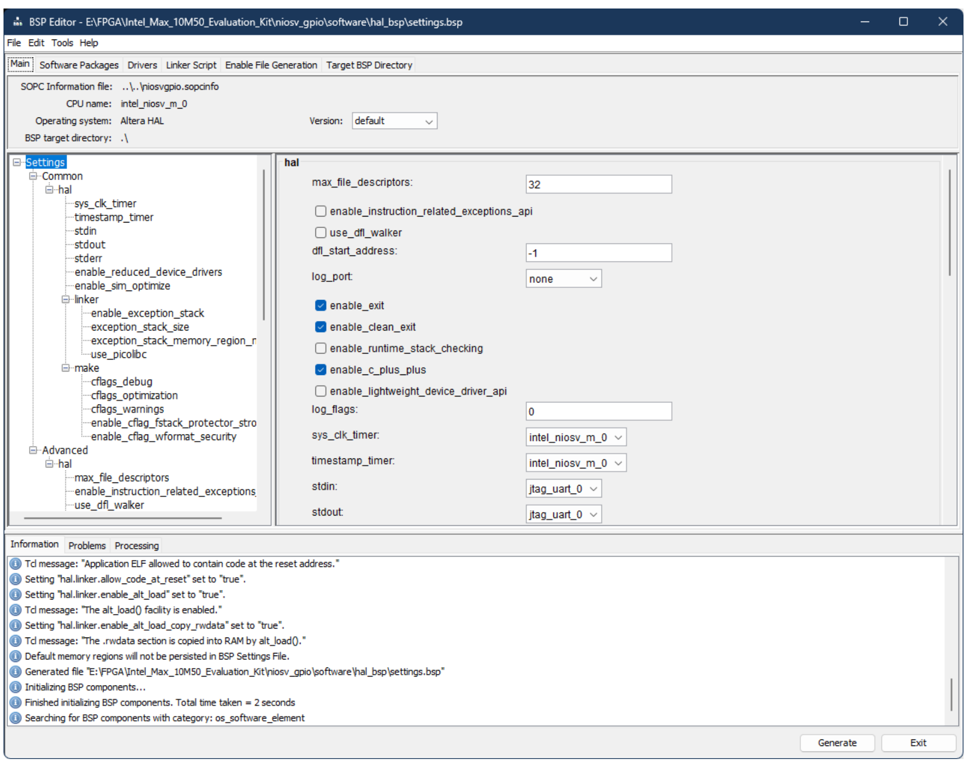
```
[niosv-shell] C:\Users\SEAN_\AppData\Local\quartus> cd \altera_lite\25.1std\niosv\bin

[niosv-shell] C:\altera_lite\25.1std\niosv\bin> dir
 Volume in drive C is OS
 Volume Serial Number is 8407-76ED

 Directory of C:\altera_lite\25.1std\niosv\bin

12/15/2025  06:08 PM    <DIR>          .
12/15/2025  06:08 PM    <DIR>          ..
10/22/2025  05:16 PM           248,832 elf2flash.exe
10/22/2025  05:16 PM            91,867 elf2flash.jar
10/22/2025  05:16 PM           248,832 elf2hex.exe
10/22/2025  05:16 PM            71,571 elf2hex.jar
10/22/2025  06:25 PM           249,344 niosv-app.exe
10/22/2025  06:25 PM           249,344 niosv-bsp-editor.exe
10/22/2025  06:24 PM           249,344 niosv-bsp.exe
12/15/2025  06:08 PM    <DIR>          niosv-download-files
10/22/2025  06:29 PM           218,112 niosv-download.exe
12/15/2025  06:08 PM    <DIR>          niosv-shell-files
10/22/2025  06:29 PM           218,112 niosv-shell.exe
12/15/2025  06:08 PM    <DIR>          niosv-stack-report-files
10/22/2025  06:29 PM           218,112 niosv-stack-report.exe
              10 File(s)      2,063,470 bytes
               5 Dir(s)  1,392,243,388,416 bytes free

[niosv-shell] C:\altera_lite\25.1std\niosv\bin>
```

4. Run niosv-bsp-editor.exe
5. The BSP Editor opens up.

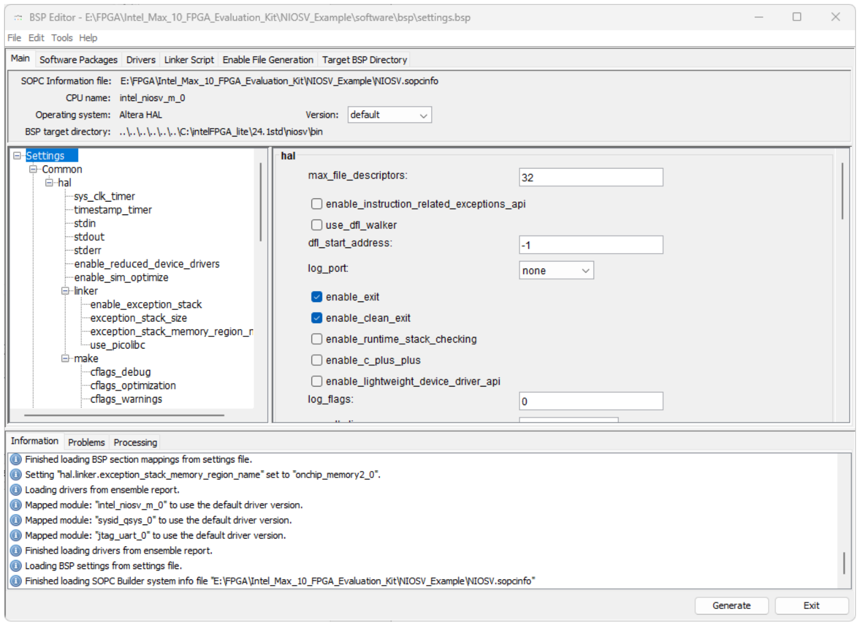
6. From the menu, select File-New BSP…
7. Click on the button with 3 dots for “SOPC Information File name:”
8. Open the niosvgpio.sopcinfo. As the file opens, the file information is read and fills the rest of the dialog.



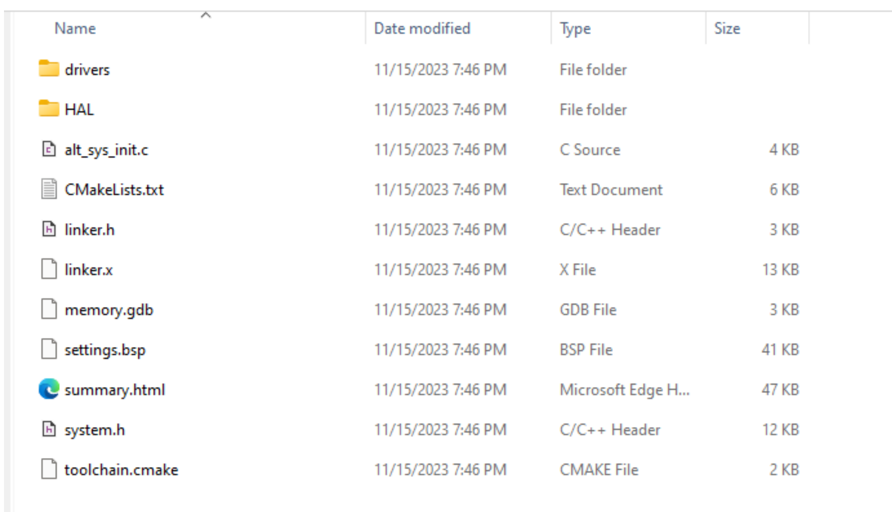
9. Keep the default settings. Click Ok.



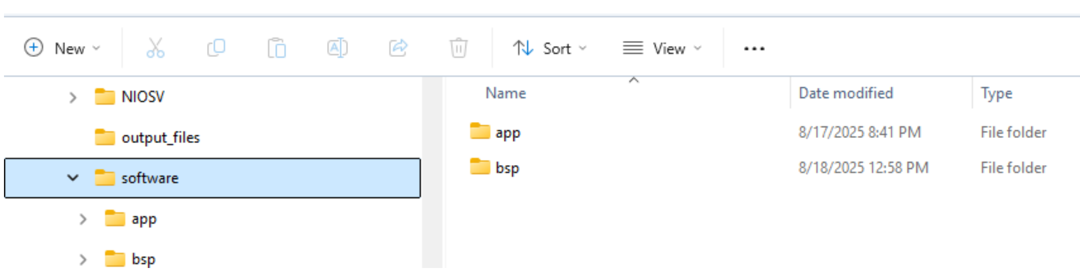
10. Under Settings root, uncheck “enable_c_plus_plus”



11. Click the Generate button.
12. Once the BSP has been generated, click on Exit to close the BSP Editor.
13. Open File Explorer, and navigate to \NIOSV_Example\software folder. You will see what BSP files were generated.



14. Under \NIOSV_Example\software folder, create a subfolder called “app.”



15. Create a text file called main.c in the \niosv_gpio\software\app folder.
16. In the niosv-shell, change directory to the \niosv_gpio\software\app app folder.
17. Run the following command:

```
niosv-app -b=../hal_bsp -a=. -s=main.c
```

18. A CmakeList.txt file gets generated, open the text file in an editor.

```
cmake_minimum_required(VERSION 3.14)

add_subdirectory(../hal_bsp hal_bsp)

include(../hal_bsp/toolchain.cmake)

project(app)

enable_language(ASM)
enable_language(C)
enable_language(CXX)

add_executable(app.elf)

target_sources(app.elf
    PRIVATE
        main.c
)

target_include_directories(app.elf
    PRIVATE
    PUBLIC
)

target_link_libraries(app.elf
    PRIVATE
        -T "${BspLinkerScript}" -nostdlib
        "${ExtraArchiveLibraries}"
        -Wl,--start-group "${BspLibraryName}" -lc -lstdc++ -lgcc -lm -Wl,--end-group
)

# Create objdump from ELF.
set(objdump app.elf.objdump)
add_custom_command(
    OUTPUT "${objdump}"
    DEPENDS app.elf
    COMMAND "${ToolchainObjdump}" "${ToolchainObjdumpFlags}" app.elf >
            "${objdump}"
    COMMENT "Creating ${objdump}."
    VERBATIM
)
add_custom_target(create-objdump ALL DEPENDS "${objdump}")

# Report space free for stack + heap. Note that the file below is never created
# so the report is always output on build.
```

```
set(stack_report_file app.elf.stack_report)
add_custom_command(
   OUTPUT "${stack_report_file}"
   DEPENDS app.elf
   COMMAND niosv-stack-report -p "${ToolchainPrefix}" app.elf
   COMMENT "Reporting memory available for stack + heap in app.elf."
   VERBATIM
)
add_custom_target(niosv-stack-report ALL DEPENDS "${stack_report_file}")

# Generate HEX file(s) from app.elf using elf2hex tool.
# Note : If ECC Full is enabled, width of 39 is set for NiosV TCM. Otherwise, 32.
add_custom_command(
   OUTPUT "onchip_memory2_0.hex"
   DEPENDS app.elf
   COMMAND elf2hex app.elf -o onchip_memory2_0.hex -b 0x00000000 -w 32 -e 0x00018FE6 -r 4
   COMMENT "Creating onchip_memory2_0.hex."
   VERBATIM
)
add_custom_target(create-hex ALL DEPENDS "onchip_memory2_0.hex")
```

1.3 Part 3: Write the GPIO-Interrupt Application in RiscFree IDE for Intel FPGAs

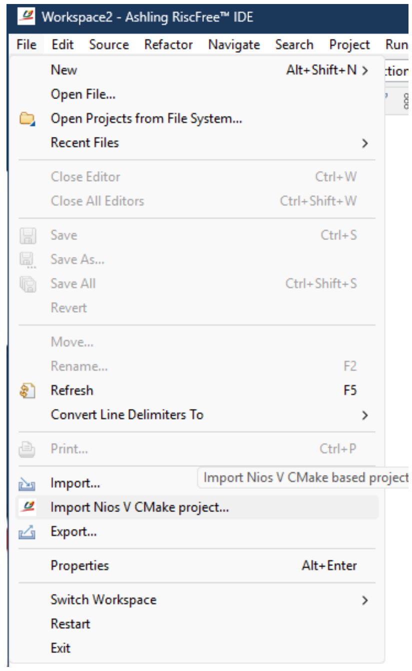
Now, we are ready to write the applications in the RiscFree IDE.

1. Open niosv-shell if not already open.
2. Type the following and hit enter to start RiscFree IDE:

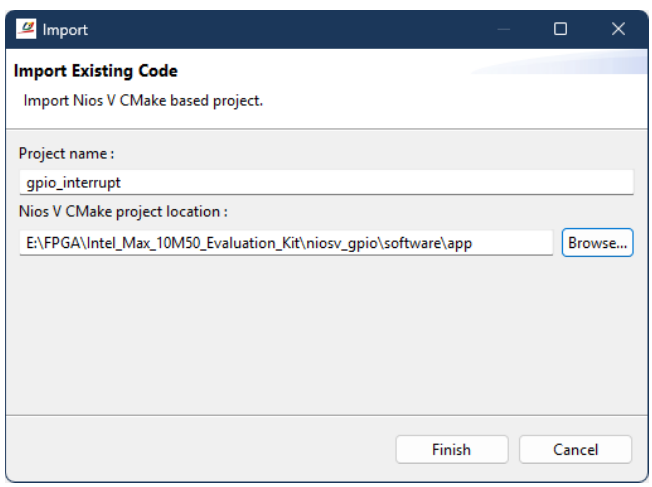
riscfree.exe

Warning: The reason to open RiscFree.exe via niosv-shell is to take advantage of the niosv shell path settings so the build tools can run the HEX generation convertions. If RiscFree.exe is run outside the niosv-shell environment, the application will be built, but fail on the HEX conversion.

3. Keep the default work space and click Launch.
4. Click on Create Project or from the menu select File->"Import Nios V CMake project…"



5. In the import dialog, click on browser and open the app folder location.
6. Change the project name to gpio_interrupt.



7. Click Finish.

The system.h file in the \hal_bsp folder contains the defines for the GPIO such as BUTTON1INT_BASE and LEDS_BASE. These hardware defines are used in the main application.

8. Open the main.c and fill in the following code:

```
#include "stdio.h"
#include "system.h"
#include "unistd.h"
#include "alt_types.h"
#include <sys/alt_sys_init.h>
#include "sys/alt_stdio.h"
```

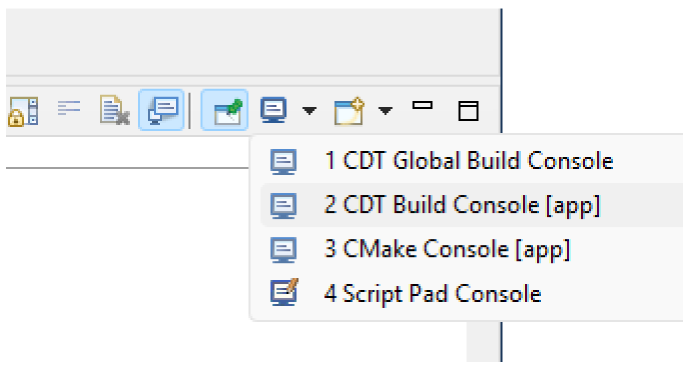
```
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"

// Global variable to track interrupt events
volatile int gpio_interrupt_flag = 0;

// Interrupt Service Routine (ISR) for GPIO
void gpio_isr(void* context) {

        volatile int* edge_capture_ptr = (volatile int*)context;

    // Clear the interrupt edge capture register
        *edge_capture_ptr =
IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1INT_BASE);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1INT_BASE, 0);

    // Set the interrupt flag
    gpio_interrupt_flag = 1;

    printf("GPIO Interrupt Triggered!\n"); //For example only, should don't this normally
}

int main() {

    // Variable to store edge capture value
    volatile int edge_capture = 0;
    int in, out;

    // Clear any pending interrupts
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1INT_BASE, 0);

    // Enable interrupts for the GPIO
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON1INT_BASE, 0x1); // Enable
interrupt for pin 0

    // Register the ISR
    alt_ic_isr_register(BUTTON1INT_IRQ_INTERRUPT_CONTROLLER_ID,
BUTTON1INT_IRQ, gpio_isr, (void*)&edge_capture, 0);

    printf("GPIO Interrupt Example Started.\n");

    while (1) {
        if (gpio_interrupt_flag) {
            // Handle the interrupt event
            gpio_interrupt_flag = 0;
            printf("Handling GPIO Interrupt...\n");
             in = IORD_ALTERA_AVALON_PIO_DATA(LEDS_BASE);
             out = in ^ 0x1F;
             IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, out);
        }

    }
```

```
    return 0;
}
```

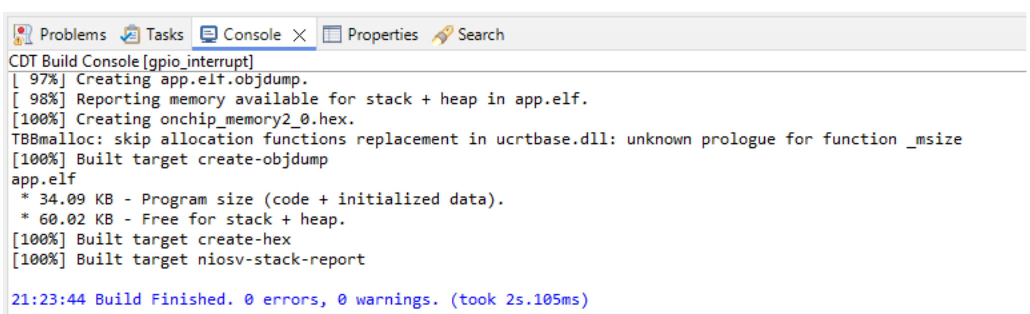
The application toggles the user LEDs on each press of the button. A global variable is setup as an interrupt flag. The interrupt flag will be used by the main while-loop to check when the interrupt has been triggerd. The gpio_isr() handles the interrupt. The isr checks the edge register and clears the register so other interrupts can be proceed. The interrupt flag is then set and message is printed to the console.

The main() application creates a variable for the edge capture value, and two other variables that will be use to read/write to the User LEDs. All intererupts are cleared, the button interrupt is enabled, the gpio_isr is registered to the button interrupt. After a message is sent that the application has started, the main while-loop is reached. If the interrupt flag is set, the flag is cleared, a message is sent to the console, and the LEDs are XOR to toggle between off-on-off-on-off and on-off-on-off-on.

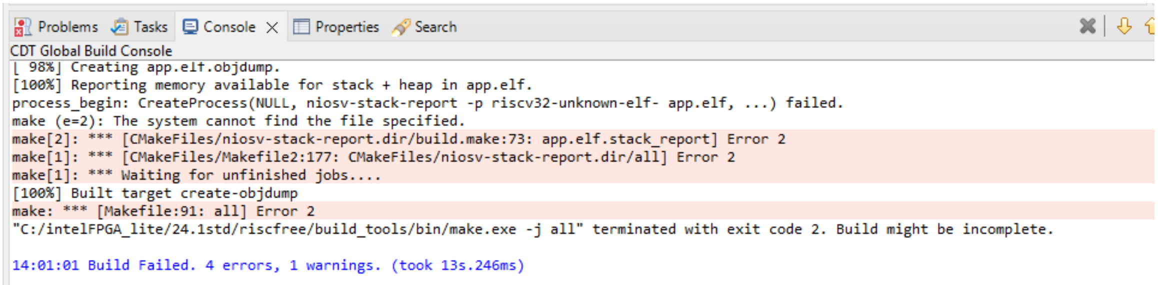
9. Right-click on app project and select Build Project from the context menu.
10. Near the bottom right, set the console output to CDT Build Console [app].



The build should complete successfully with a app.elf file being created.



Warning: If you open RiscFree IDE independently of the niosv-shell, an error will appear at the end. The file was built, but the paths to run the stack report and create HEX files were invalid. You can always run the niosv-stack-report.exe and elf2hex.exe seperatly.



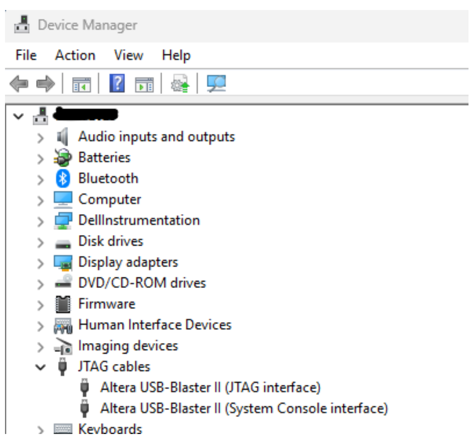
1.4 Part 4: Test the Design and the Application

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

1.4.1 Hardware Setup

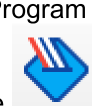
The 10M50 Evaluation kit comes with a USB cable with red and black USB connectors. The black connector is for the JTAG. The red connector is only for power. JTAG communcation can be over USB or through the 10-PIN JTAG connector. JTAG over USB will be used for this example.

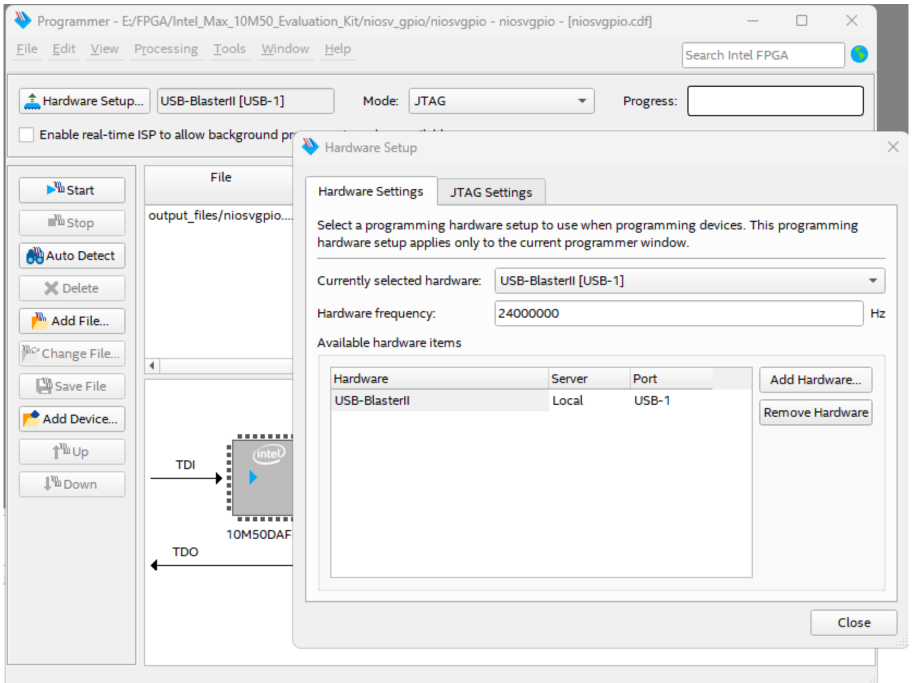
1. Switch 4 on switch block 2 (SW2.4) enables/ diables the USB JTAG. The default mode is off. Switch SW2.4 to On for USB JTAG.
2. Since this will be powered by USB make sure jumper 11 (J11) has pins 2 and 3 connected.
3. Connect the USB cable to the 10M50 Eveluation kit and the red and black connectors to your Quartus development system.
4. 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.



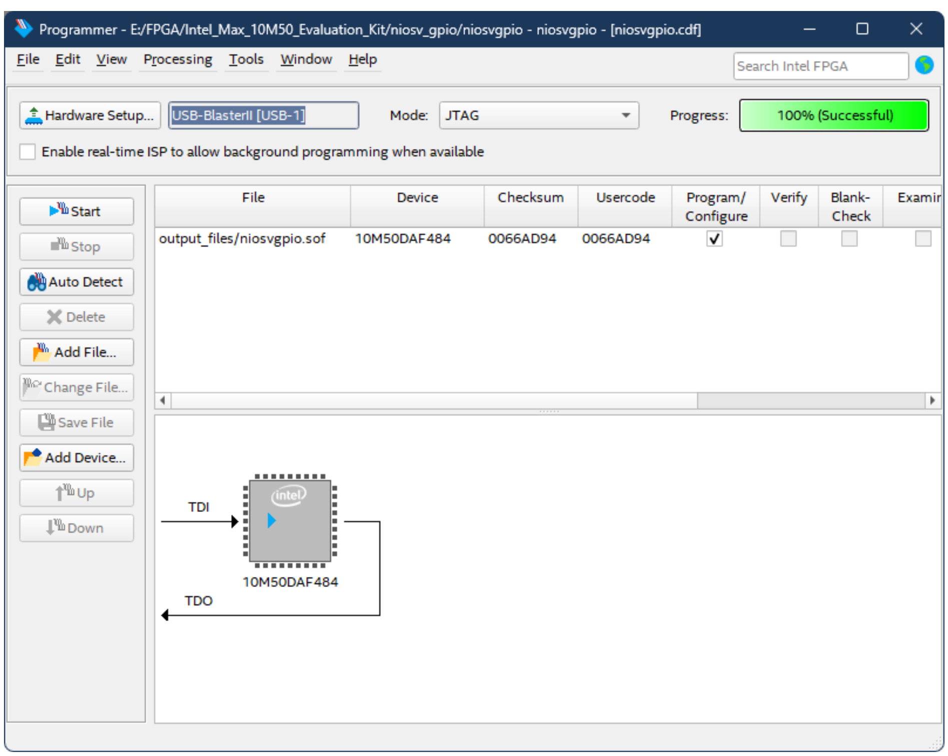
1.4.2 Program the Board

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 niosvgpio.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.

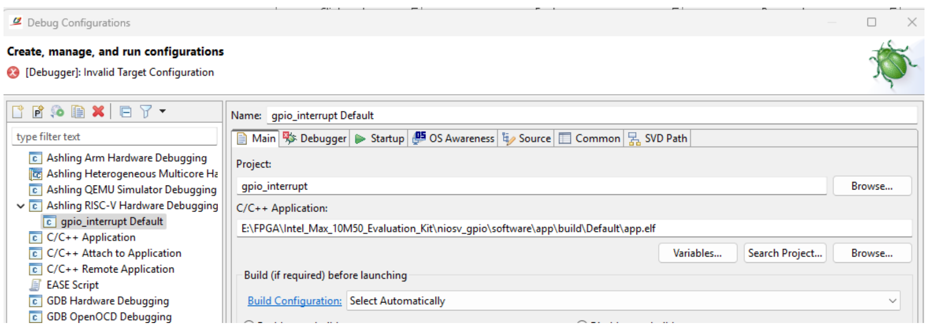


The 5 user leds get set to off-on-off-on-off per the preconfiguration performed in Platform Designer. Notice that there is no time constraint dialog box popping up. The Nios V is free and doesn't required a paid license.

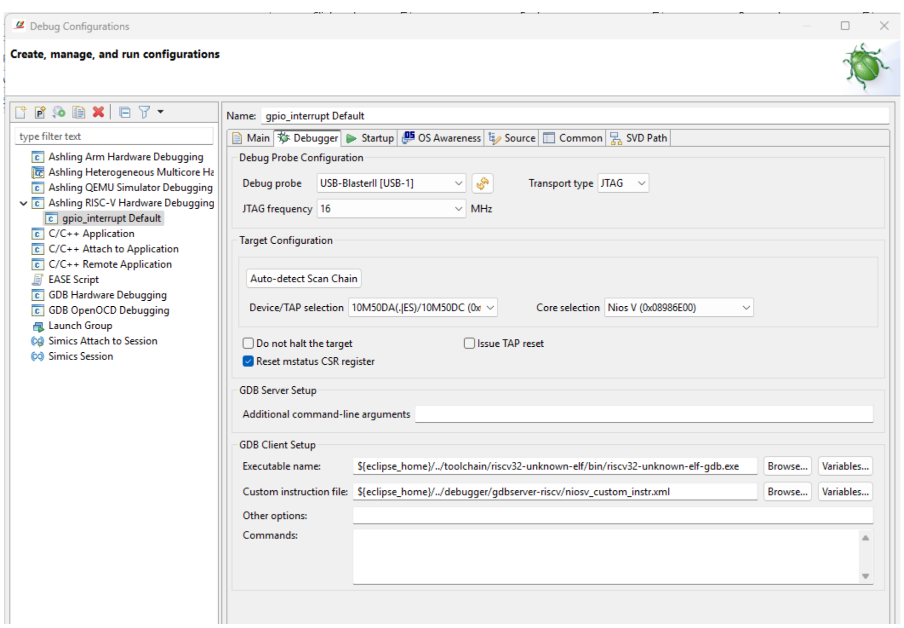
1.4.3 Debug the Application

Now, we go back to the RiscFree IDE to download and debug the applicaiton

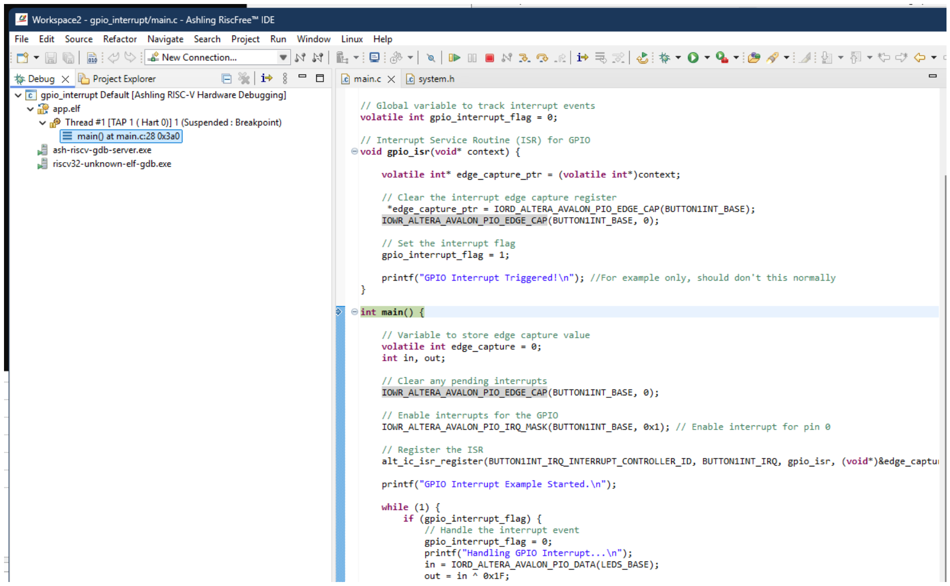
1. With the JTAG cable connected, in RiscFree IDE, right click on the gpio_interrupt project in Project Explorer.
2. Select Debug AS->Debug Configurations…
3. Click on Ashling RISC-V hardware Debugging
4. In the Main tab, you should see the project pointing to the app.elf file.



5. Click on Debugger tab.
6. Set the Debug probe to the debug probe connected to the board.
7. Set the JTAG frequency to 16 MHz
8. Click on the "Auto-detect Scan Chain". The Devcie/TAP selection should be filed in with the 10M08 FPGA and the Core selection should be the Nios V.

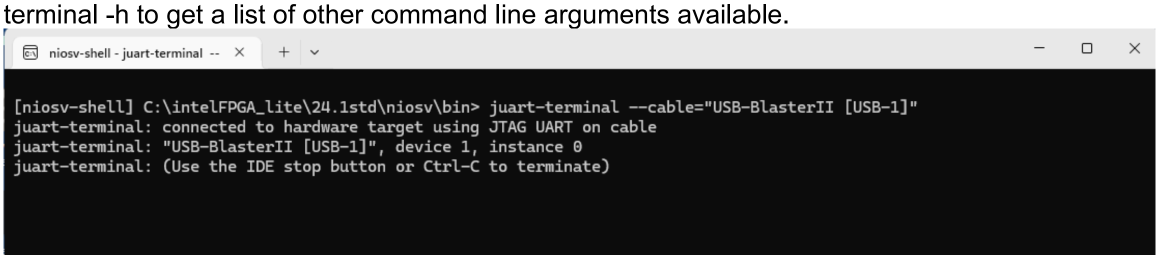


9. Click Apply
10. Click Debug. The application will download and stop at the first line in Main().

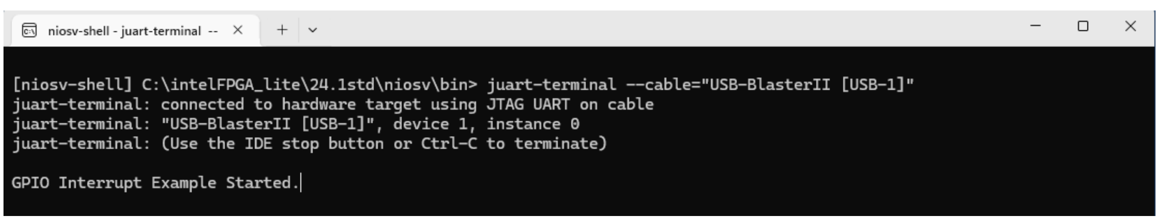


11. Open niosv-shell.l
12. Run the following command to start the terminal application: `juart-terminal --cable="USB-BlasterII [USB-1]"`

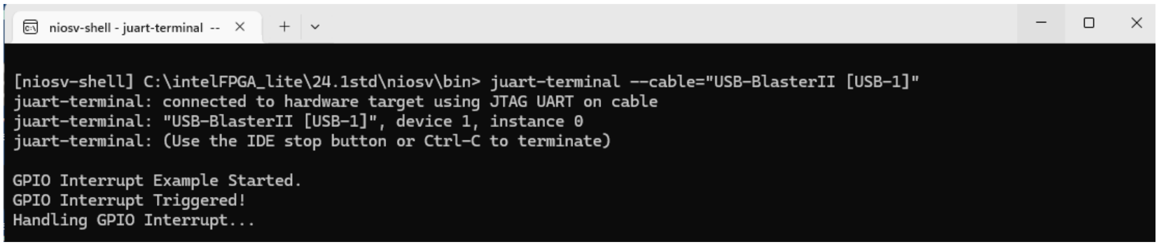
Note: The arguments specify which cable to use as the JTAG console connection. Run juart-terminal -h to get a list of other command line arguments available.



13. In RiscFree IDE, step throught the code. You will see the messages sent to the juart-terminal.



14. Click Continue to let the program run.
15. Press switch button 1 on the board. The LEDs should toggle to on-off-on-off-on, and messages aboth the interrupt being serviced should appear on the juart terminal.



16. Stop debugging when finished.

1.5 Summary: Building Up a Custom Microcontroller

The 10M50 Evaluaiton Kit offers more on chip RAM to support Nios V applications. The exercise demonstrated GPIO read, write, and interrupt handling with Nios V. From here, all the rest of the microcontroller such as UART, SPI, I2C can be added to create a custom micro controller.

1.6 References

The following references were used for this article:

RISKFree IDE Documentation:
https://www.intel.com/content/www/us/en/docs/programmable/730783/24-2/about-the-ide.html

Video: Debugging the Nios® V Processor Using the RiscFree* IDE for Intel® FPGAs
Video: Build A Soft Core CPU - Part Three - NIOS II in Intel FPGA