General Nios ii IP Implementation

Table of Contents

Introduction

▪ Purpose/Use

The General Nios ii IP gives the user ability to code C code to a Nios ii softcore processor synthesized onto the FPGA.

This document will go through the process of setting up the Nios II Eclipse IDE Environment to run, debug, and create code to communicate with the rest of the FPGA.

Initialization and Setup

▪ Installing Eclipse IDE Environment

It is possible the Eclipse IDE environment is already installed in the Quartus directory. To check if it is installed or not, click the link below and follow step 5.

https://www.terasic.com.tw/wiki/Getting_Start_Install_Eclipse_IDE_into_Nios_EDS#:~:text=The%20procedure%20for%20installing%20Eclipse%20IDE%3A%201%20Download,%3C%20Intel%C2%AE%20Quartus%C2%AE%20Prime%20installation%20directory%3E%2Fnios2eds%2Fbin%2Feclipse_nios2.%20More%20items

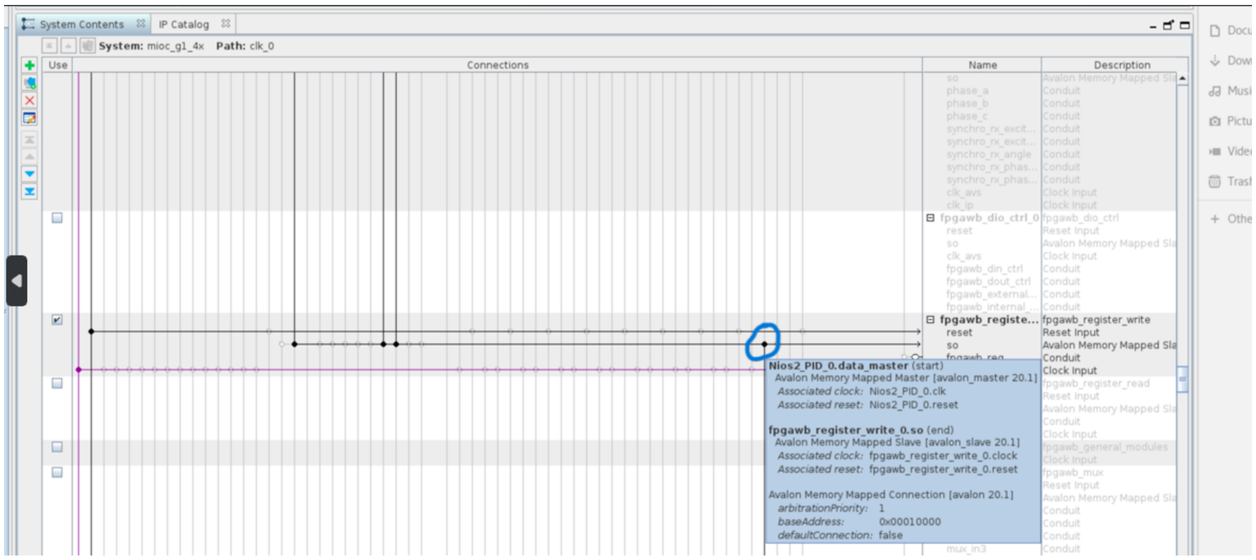
If the folder or file is not there, then follow steps 1-3 to install the Eclipse IDE Environment. Confirm again that is installed correctly with step 5.

Now the Nios II Eclipse IDE Environment should be installed and ready to use.

▪ Setting up Firmware

Generall, if there is a device the Nios is to communicate to, the Data Master Avalon Memory Mapped Master bus line from the Nios has to connect to that device. This can be done by opening the “mioc_g1_4x.qsys” file inside of Quartus Platform Designer.

The photo below shows an example of the Nios Data Master bus being connected to the FPGAWB IP Register Write. In this case there is only one IP the Nios is communicating to, if there are more in your project, the Nios Data Master must be individually connected to each of the IPs.



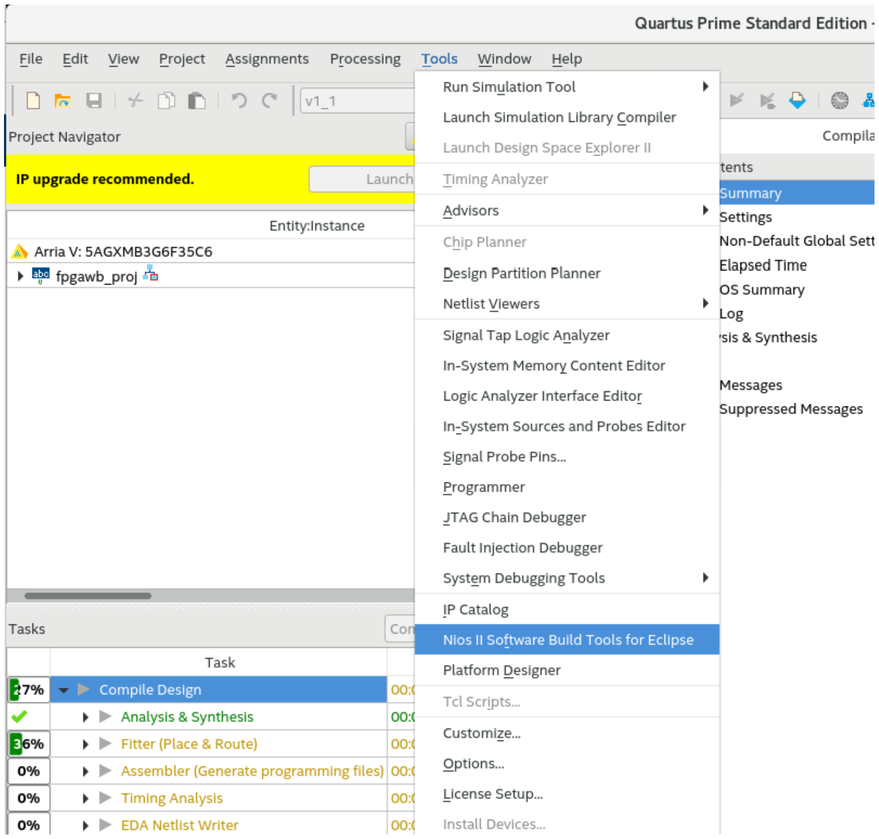
```
Nios2_PID_0.data_master (start)
 Avalon Memory Mapped Master [avalon_master 20.1]
  Associated clock:  Nios2_PID_0.clk
  Associated reset:  Nios2_PID_0.reset

fpgawb_register_write_0.so (end)
 Avalon Memory Mapped Slave [avalon_slave 20.1]
  Associated clock:  fpgawb_register_write_0.clock
  Associated reset:  fpgawb_register_write_0.reset

Avalon Memory Mapped Connection [avalon 20.1]
  arbitrationPriority:   1
  baseAddress:           0x00010000
  defaultConnection:     false
```

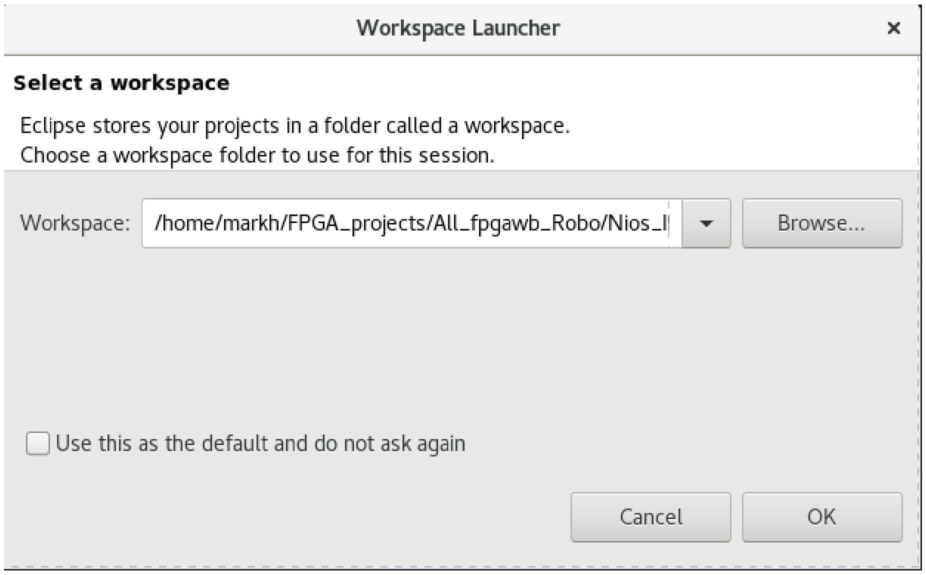
Once this task has been completed, save the file. **If prompted if you would like to generate, select no**.

Now the Quartus project is ready for a full compile.

▪ Creating Nios II Eclipse Project

After adding data bus connections and a full Quartus compile the Nios II Eclipse Project can now be created.

1. In the Quartus Project window select tools at the top of the window and select Nios II Software Build Tools for Eclipse.
   a. 

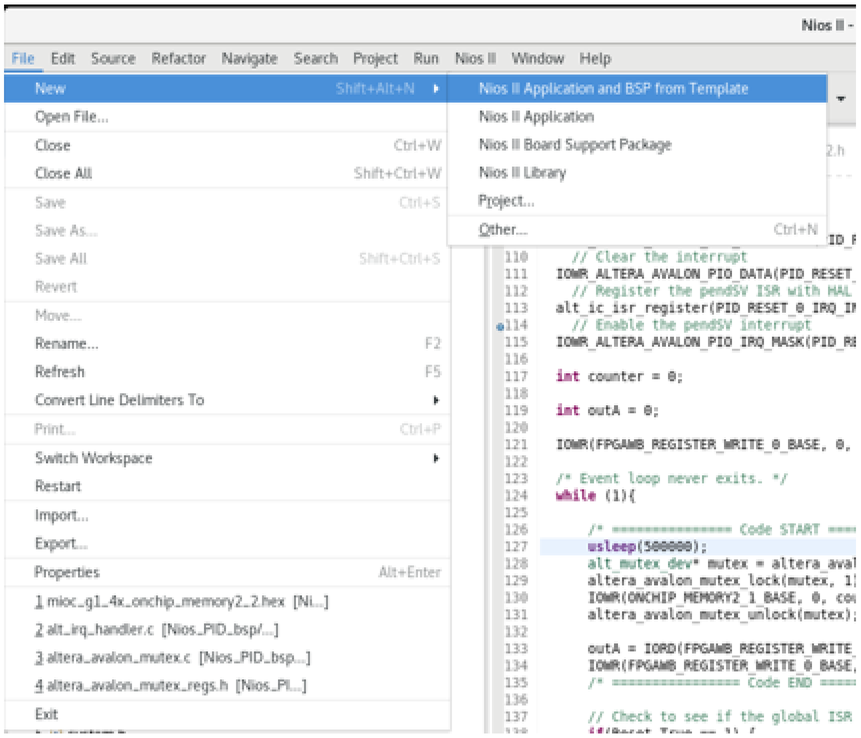
2. Select a directory to create the Eclipse project (location doesn't matter)
   a. 

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

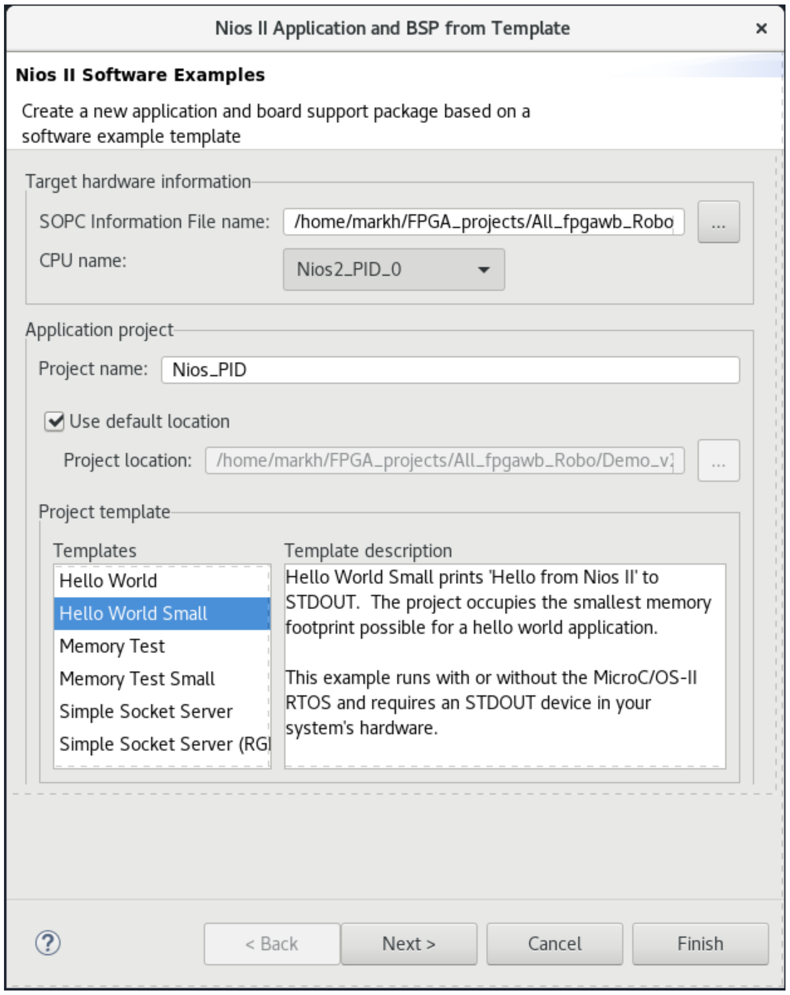
Workspace: /home/markh/FPGA_projects/All_fpgawb_Robo/Nios_I

3. The Nios II Eclipse window should now open select file to create a project from a template.

   a. 

```
  // Clear the interrupt
IOWR_ALTERA_AVALON_PIO_DATA(PID_RESET_
  // Register the pendSV ISR with HAL
alt_ic_isr_register(PID_RESET_0_IRQ_IN
  // Enable the pendSV interrupt
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PID_RE

int counter = 0;

int outA = 0;

IOWR(FPGAWB_REGISTER_WRITE_0_BASE, 0,

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

    /* ================ Code START ====
    usleep(500000);
    alt_mutex_dev* mutex = altera_aval
    altera_avalon_mutex_lock(mutex, 1)
    IOWR(ONCHIP_MEMORY2_1_BASE, 0, cou
    altera_avalon_mutex_unlock(mutex);

    outA = IORD(FPGAWB_REGISTER_WRITE_
    IOWR(FPGAWB_REGISTER_WRITE_0_BASE,
    /* ================= Code END =====

    // Check to see if the global ISR
```

4. After this you will have to select the “mioc_g1_4x.sopcinfo” file and the corresponding Nios you will be programming to. Give the Nios Project a name. **Select “Hello World” as the template NOT “Hello World Small”**

a.

Nios II Application and BSP from Template

Nios II Software Examples

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

Target hardware information

SOPC Information File name: /home/markh/FPGA_projects/All_fpgawb_Robo

CPU name: Nios2_PID_0

Application project

Project name: Nios_PID

[✓] Use default location

Project location: /home/markh/FPGA_projects/All_fpgawb_Robo/Demo_v]

Project template

Templates: Hello World, **Hello World Small**, Memory Test, Memory Test Small, Simple Socket Server, Simple Socket Server (RGI

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

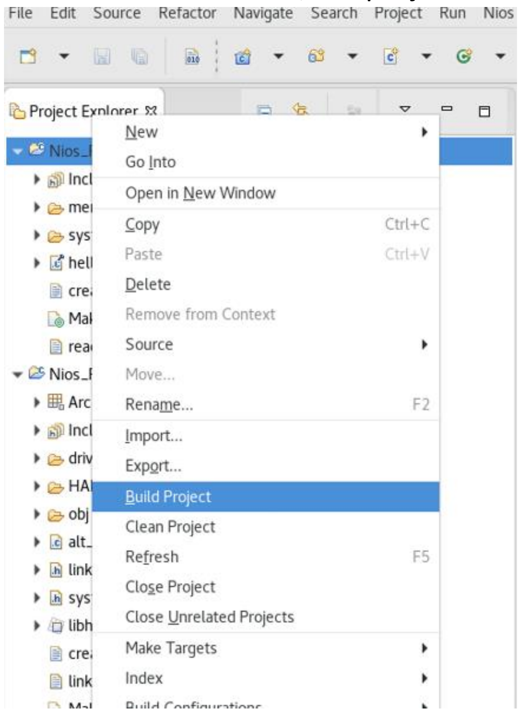
This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

i. Once completing the above, click finish and the project will be created.

Running Software on Nios

The project generated should have a simple file called hello_world.c that uses the printf() function to print to the console.

1. Step one will be to run this file to make sure everything is working before moving forward and implementing any more code.
2. Whenever the project is opened for the first time or changes have been made to the source code and saved, the project needs to be built.



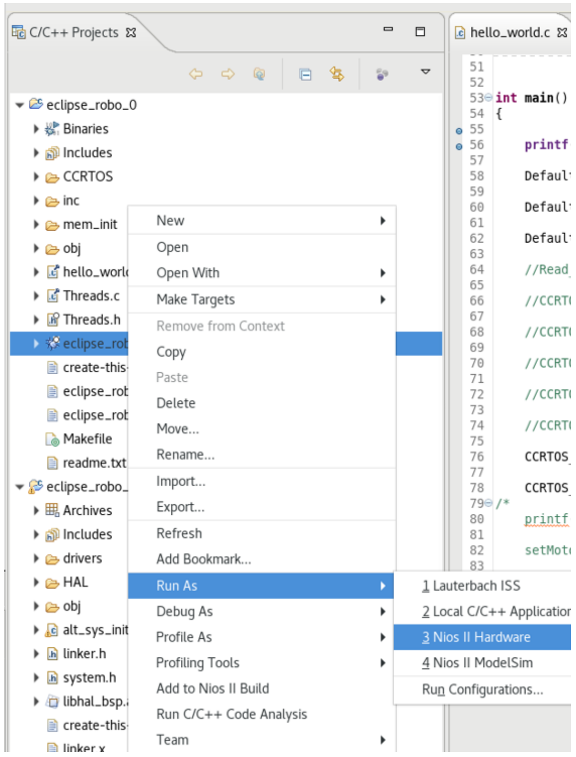
   a. Build the project to check if there are any errors in the code (which there shouldn't be at this point since the template hello_world.c file is the only thing being built).
      i. **Refer to Makefile error section under Misc. Problems/Errors if there is a build error.**
   b. Whenever the project is built a .elf file is created in the project directory. The .elf file is what is flashed to the On-Chip memory to run the Nios II C code.

▪ .elf Running as Hardware

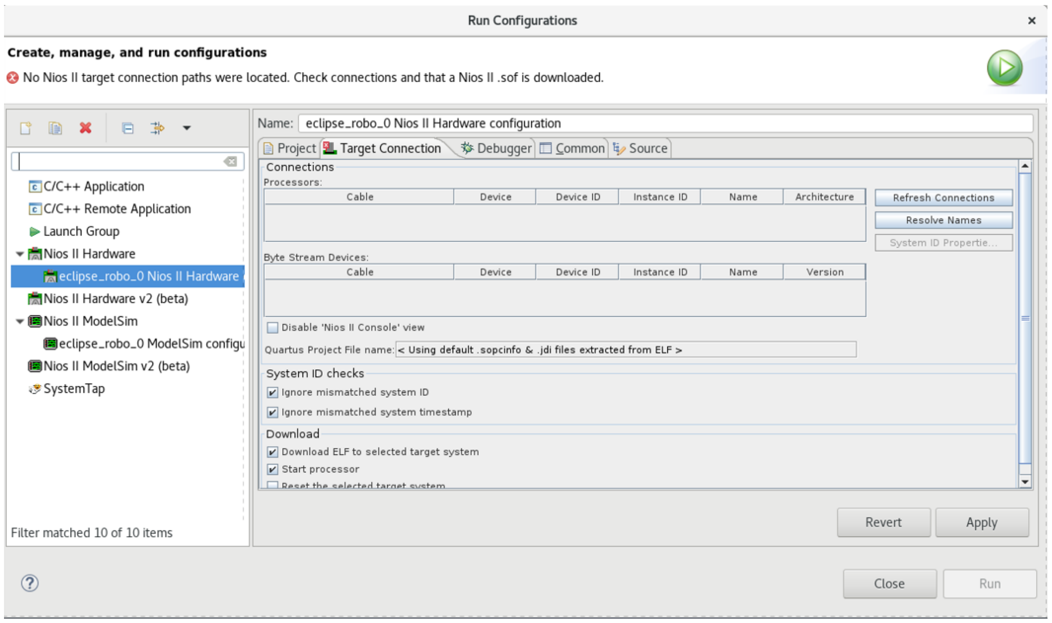
Whenever wanting to run C code on the Nios II processor synthesized to the FPGA the following will have to be done.

The .elf file is flash directly to the FPGA with a JTAG UART connection.

Make sure to follow the "Setting up JTAG USB Blaster" section before moving on.

1. First Right click the .elf file and Run as hardware
   a. 

```
int main()
{
    printf
    Defaul
    Defaul
    Defaul
    //Read_
    //CCRT
    //CCRT
    //CCRT
    //CCRT
    //CCRT
    CCRTOS_
    CCRTOS_
/*
    printf
    setMot
```
2. Upon initial run as hardware the user will have to select the hardware the .elf will be running off of and the following window will pop up.

a. 

Run Configurations

Create, manage, and run configurations

No Nios II target connection paths were located. Check connections and that a Nios II .sof is downloaded.

Name: eclipse_robo_0 Nios II Hardware configuration

Connections

Processors:

Cable	Device	Device ID	Instance ID	Name	Architecture

Byte Stream Devices:

Cable	Device	Device ID	Instance ID	Name	Version

☐ Disable 'Nios II Console' view

Quartus Project File name: < Using default .sopcinfo & .jdi files extracted from ELF >

System ID checks

☑ Ignore mismatched system ID

☑ Ignore mismatched system timestamp

Download

☑ Download ELF to selected target system

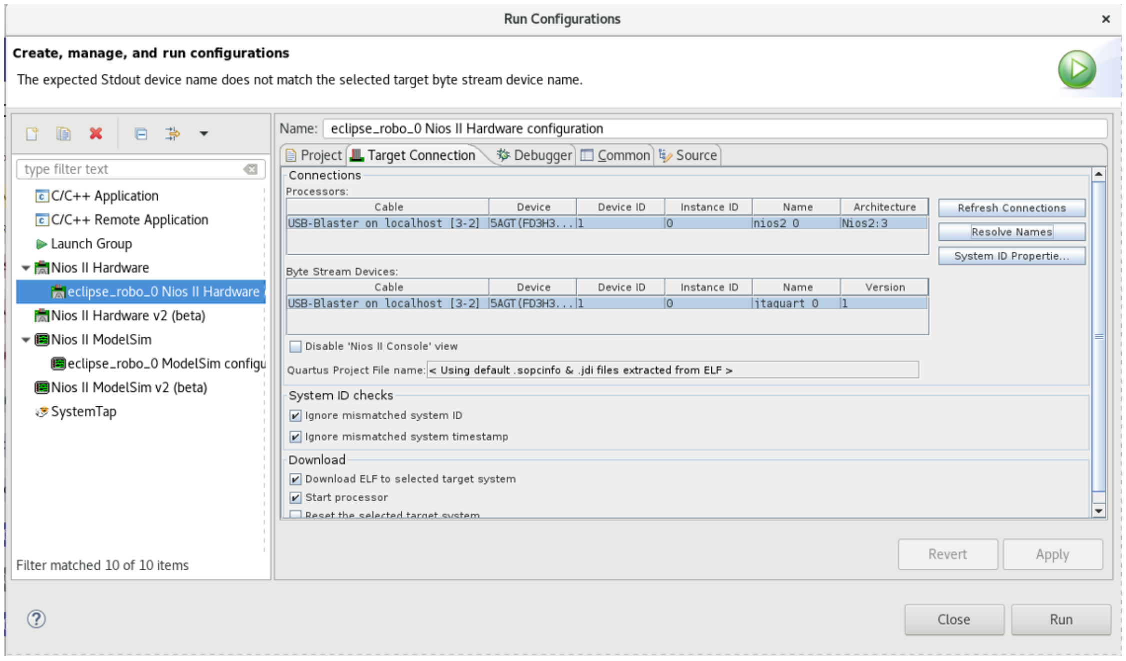
☑ Start processor

☐ Reset the selected target system

Filter matched 10 of 10 items

   i. Make sure to check both boxes under “System ID checks” after doing so click “Apply” then the “Refresh Connections” button.

3. After Refreshing the connections, the following should happen:

a. 

Run Configurations

Create, manage, and run configurations

The expected Stdout device name does not match the selected target byte stream device name.

Name: eclipse_robo_0 Nios II Hardware configuration

Connections

Processors:

Cable	Device	Device ID	Instance ID	Name	Architecture
USB-Blaster on localhost [3-2]	5AGT(FD3H3...	1	0	nios2_0	Nios2:3

Byte Stream Devices:

Cable	Device	Device ID	Instance ID	Name	Version
USB-Blaster on localhost [3-2]	5AGT(FD3H3...	1	0	jtaguart_0	1

☐ Disable 'Nios II Console' view

Quartus Project File name: < Using default .sopcinfo & .jdi files extracted from ELF >

System ID checks

☑ Ignore mismatched system ID

☑ Ignore mismatched system timestamp

Download

☑ Download ELF to selected target system

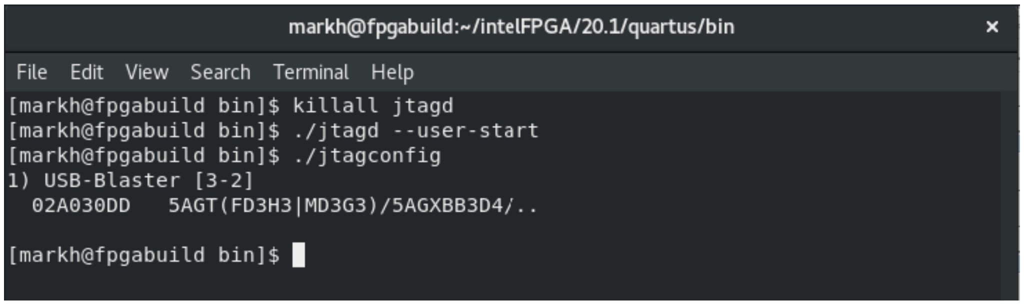
☑ Start processor

☐ Reset the selected target system

Filter matched 10 of 10 items

   i. This means that the Nios II soft-core processor has been detected, select it and click run.

4. If No connection is detected, try the following:
   a. Go to the directory:
      < Intel® Quartus® Prime installation directory>/quartus/bin>
   b. Once in the directory run the following commands in terminal:

c. 

markh@fpgabuild:~/intelFPGA/20.1/quartus/bin
File Edit View Search Terminal Help

```
[markh@fpgabuild bin]$ killall jtagd
[markh@fpgabuild bin]$ ./jtagd --user-start
[markh@fpgabuild bin]$ ./jtagconfig
1) USB-Blaster [3-2]
  02A030DD   5AGT(FD3H3|MD3G3)/5AGXBB3D4/..

[markh@fpgabuild bin]$ 
```

d. If the command has worked correctly something similar should pop up in terminal after the ./jtagconfig command.
e. Sometimes it can be necessary to run the “killall jtagd” multiple times.

5. Once this has been done refresh connections and select the Nios II processor to run the file.
6. Assuming the connection was found and the hello_world.c file has not changed, “Hello from Nios ii” should print to the Eclipse console.
7. This means that everything is working and new code can be added to the project file.

Anytime code is added, the files must be saved, built, then the .elf file must be re-flashed.

It is worth noting that even when the .elf file run as hardware is stopped the code will continue to run **UNLESS** there are any stdio library functions used in the code. Stdio library uses the JTAG connection and if the connection is broken the code will also break if, and only if, it is using the stdio library (such as printf()).

Using The Nios and Examples

From this point, coding in C works like any other case of Embedded C coding and the same can be done. The point of this part of the documentation is to highlight what use cases it has in terms of interacting with other FPGAWB IPs.

▪ Reading From and Writing to FPGAWB IPs

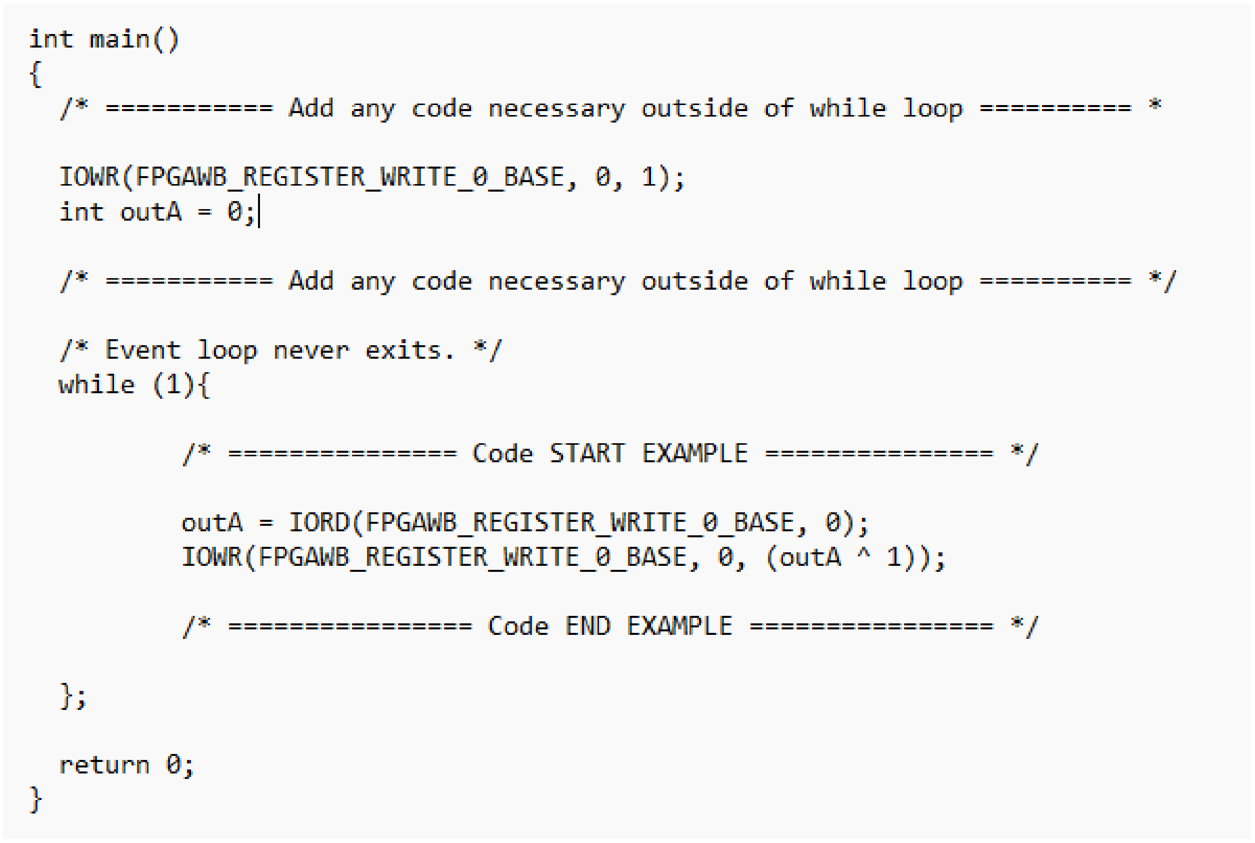
The following steps can only be done after the Data Master bus has been connected to the corresponding FPGAWB IP inside of Platform Designer.

Reading and writing to address locations requires the source code to "#include "io.h"

IORD() returns the value at an address location.

IOWR() is used to write to an address location.

The photo below shows an example of reading the value from the FPGAWB IP Register WR and then inverting the value with an exclusive or operation, then writes the value back to the same address location.



```
int main()
{
  /* =========== Add any code necessary outside of while loop ========== *

  IOWR(FPGAWB_REGISTER_WRITE_0_BASE, 0, 1);
  int outA = 0;

  /* =========== Add any code necessary outside of while loop ========== */

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

          /* =============== Code START EXAMPLE =============== */

          outA = IORD(FPGAWB_REGISTER_WRITE_0_BASE, 0);
          IOWR(FPGAWB_REGISTER_WRITE_0_BASE, 0, (outA ^ 1));

          /* ================ Code END EXAMPLE ================ */

  };

  return 0;
}
```

The logic of the code above simply toggles the value inside at the address location "FPGAWB_REGISTER_WRITE_0_BASE".

The variable “FPGAWB_REGISTER_WRITE” is defined in the file system.h which is under the _bsp project folder and needs to be included when writing to any address location.

This is an important take away as it allows the user to not have to worry about what address location they are writing to but instead use the variables defined in system.h.

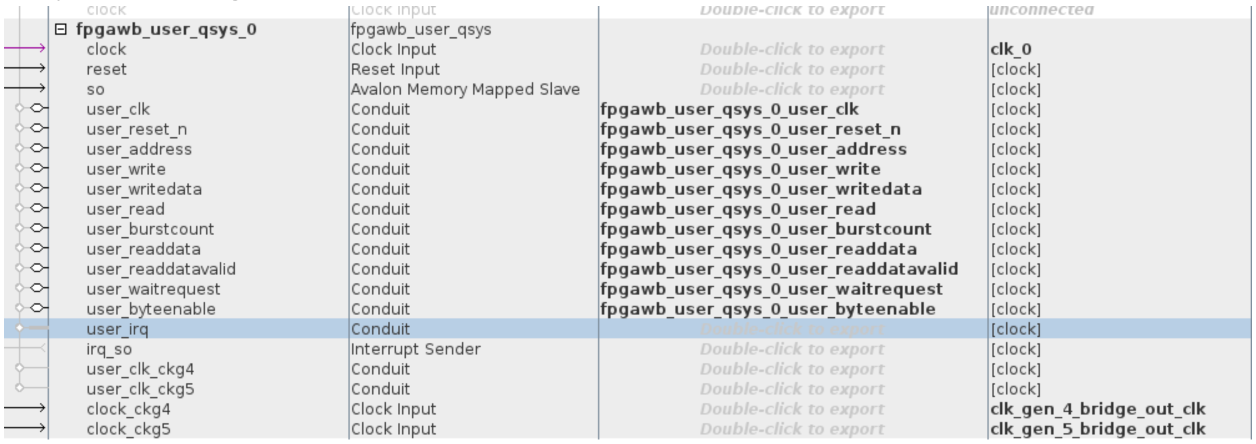
Setting up Hardware Interrupt IRQ from User Module IO

The FPGAWB IP User Module IO can send IRQs which can be received by the Nios II as a hardware interrupt.

Setting up Firmware

The User Module IO does not come with pre-built logic for a hardware interrupt, so any interrupt to be sent by this IP is expected to be rooted in custom logic. For the use-case of sending an IRQ to a Nios II processor, the pulse width for the interrupt to be set at as active high is recommended by the Nios documentation to be at minimum 50 clock cycles (Measured by the clock of the Nios, not the User Module IP_clk). This should be handled in the custom logic.

For the User Module to be able to send an IRQ, the Interrupt Sender instantiated in the IP's .qsys file must be utilized. The photo below shows how the user_qsys IP will look by default in the platform designer.

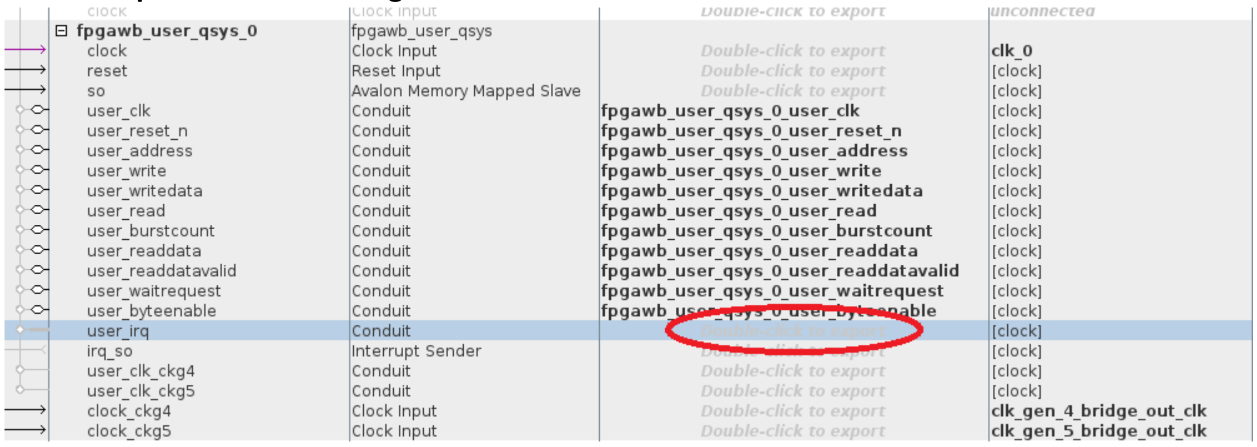


clock	Clock Input	Double-click to export	unconnected
⊟ **fpgawb_user_qsys_0**	fpgawb_user_qsys		
clock	Clock Input	Double-click to export	**clk_0**
reset	Reset Input	Double-click to export	[clock]
so	Avalon Memory Mapped Slave	Double-click to export	[clock]
user_clk	Conduit	**fpgawb_user_qsys_0_user_clk**	[clock]
user_reset_n	Conduit	**fpgawb_user_qsys_0_user_reset_n**	[clock]
user_address	Conduit	**fpgawb_user_qsys_0_user_address**	[clock]
user_write	Conduit	**fpgawb_user_qsys_0_user_write**	[clock]
user_writedata	Conduit	**fpgawb_user_qsys_0_user_writedata**	[clock]
user_read	Conduit	**fpgawb_user_qsys_0_user_read**	[clock]
user_burstcount	Conduit	**fpgawb_user_qsys_0_user_burstcount**	[clock]
user_readdata	Conduit	**fpgawb_user_qsys_0_user_readdata**	[clock]
user_readdatavalid	Conduit	**fpgawb_user_qsys_0_user_readdatavalid**	[clock]
user_waitrequest	Conduit	**fpgawb_user_qsys_0_user_waitrequest**	[clock]
user_byteenable	Conduit	**fpgawb_user_qsys_0_user_byteenable**	[clock]
user_irq	Conduit	Double-click to export	[clock]
irq_so	Interrupt Sender	Double-click to export	[clock]
user_clk_ckg4	Conduit	Double-click to export	[clock]
user_clk_ckg5	Conduit	Double-click to export	[clock]
clock_ckg4	Clock Input	Double-click to export	**clk_gen_4_bridge_out_clk**
clock_ckg5	Clock Input	Double-click to export	**clk_gen_5_bridge_out_clk**

The highlighted signal "user_irq" is a custom signal conduit created for this IP that is internally connected to the "irq_so" Interrupt Sender. Any logic driven to this conduit will control the output of irq_so.

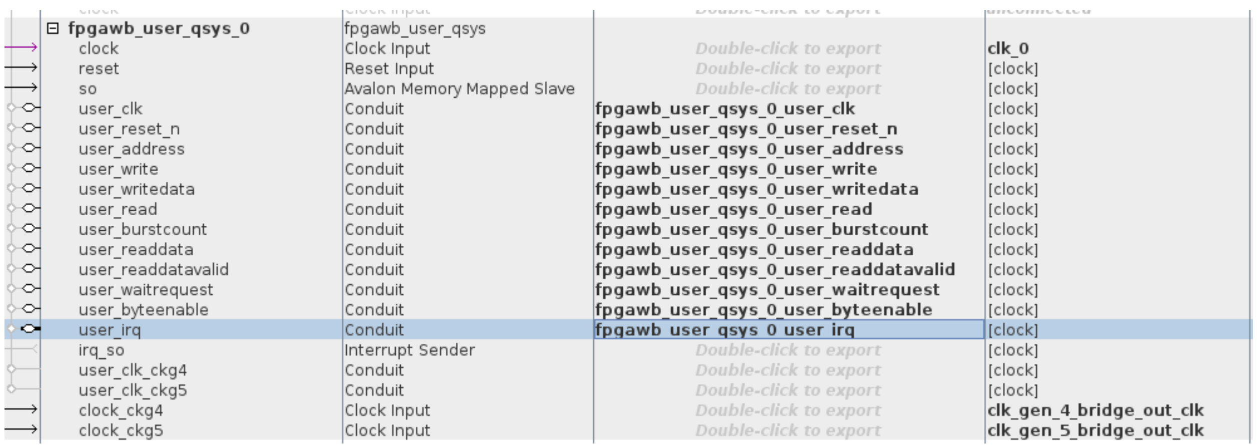
The following steps will allow for this user_irq signal to driven by custom logic:

1. **Export the user_irq signal in Platform Designer by double clicking the words “Double-click to export” next to the signal.**



clock	Clock Input	Double-click to export	unconnected
⊟ fpgawb_user_qsys_0	fpgawb_user_qsys		
clock	Clock Input	Double-click to export	clk_0
reset	Reset Input	Double-click to export	[clock]
so	Avalon Memory Mapped Slave	Double-click to export	[clock]
user_clk	Conduit	fpgawb_user_qsys_0_user_clk	[clock]
user_reset_n	Conduit	fpgawb_user_qsys_0_user_reset_n	[clock]
user_address	Conduit	fpgawb_user_qsys_0_user_address	[clock]
user_write	Conduit	fpgawb_user_qsys_0_user_write	[clock]
user_writedata	Conduit	fpgawb_user_qsys_0_user_writedata	[clock]
user_read	Conduit	fpgawb_user_qsys_0_user_read	[clock]
user_burstcount	Conduit	fpgawb_user_qsys_0_user_burstcount	[clock]
user_readdata	Conduit	fpgawb_user_qsys_0_user_readdata	[clock]
user_readdatavalid	Conduit	fpgawb_user_qsys_0_user_readdatavalid	[clock]
user_waitrequest	Conduit	fpgawb_user_qsys_0_user_waitrequest	[clock]
user_byteenable	Conduit	fpgawb_user_qsys_0_user_byteenable	[clock]
user_irq	Conduit	Double-click to export	[clock]
irq_so	Interrupt Sender	Double-click to export	[clock]
user_clk_ckg4	Conduit	Double-click to export	[clock]
user_clk_ckg5	Conduit	Double-click to export	[clock]
clock_ckg4	Clock Input	Double-click to export	clk_gen_4_bridge_out_clk
clock_ckg5	Clock Input	Double-click to export	clk_gen_5_bridge_out_clk

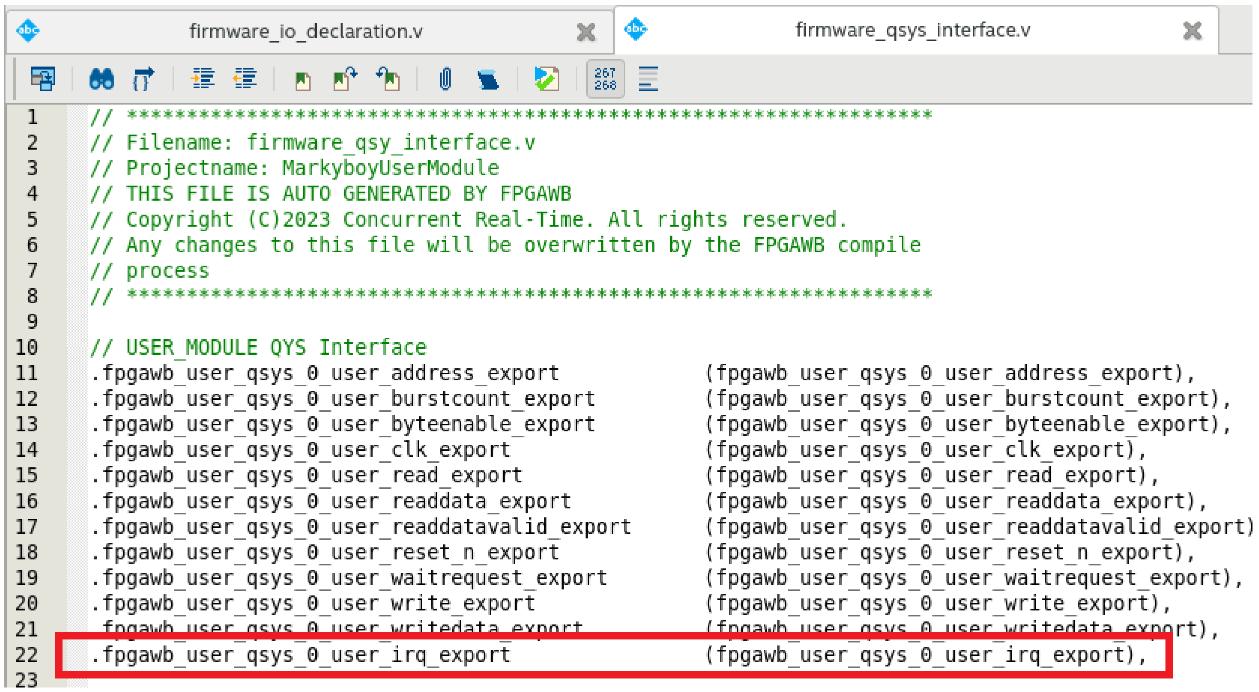
After it has been exported the signal should appear as so:



clock	Clock Input	Double-click to export	unconnected
⊟ fpgawb_user_qsys_0	fpgawb_user_qsys		
clock	Clock Input	Double-click to export	clk_0
reset	Reset Input	Double-click to export	[clock]
so	Avalon Memory Mapped Slave	Double-click to export	[clock]
user_clk	Conduit	fpgawb_user_qsys_0_user_clk	[clock]
user_reset_n	Conduit	fpgawb_user_qsys_0_user_reset_n	[clock]
user_address	Conduit	fpgawb_user_qsys_0_user_address	[clock]
user_write	Conduit	fpgawb_user_qsys_0_user_write	[clock]
user_writedata	Conduit	fpgawb_user_qsys_0_user_writedata	[clock]
user_read	Conduit	fpgawb_user_qsys_0_user_read	[clock]
user_burstcount	Conduit	fpgawb_user_qsys_0_user_burstcount	[clock]
user_readdata	Conduit	fpgawb_user_qsys_0_user_readdata	[clock]
user_readdatavalid	Conduit	fpgawb_user_qsys_0_user_readdatavalid	[clock]
user_waitrequest	Conduit	fpgawb_user_qsys_0_user_waitrequest	[clock]
user_byteenable	Conduit	fpgawb_user_qsys_0_user_byteenable	[clock]
user_irq	Conduit	fpgawb_user_qsys_0_user_irq	[clock]
irq_so	Interrupt Sender	Double-click to export	[clock]
user_clk_ckg4	Conduit	Double-click to export	[clock]
user_clk_ckg5	Conduit	Double-click to export	[clock]
clock_ckg4	Clock Input	Double-click to export	clk_gen_4_bridge_out_clk
clock_ckg5	Clock Input	Double-click to export	clk_gen_5_bridge_out_clk

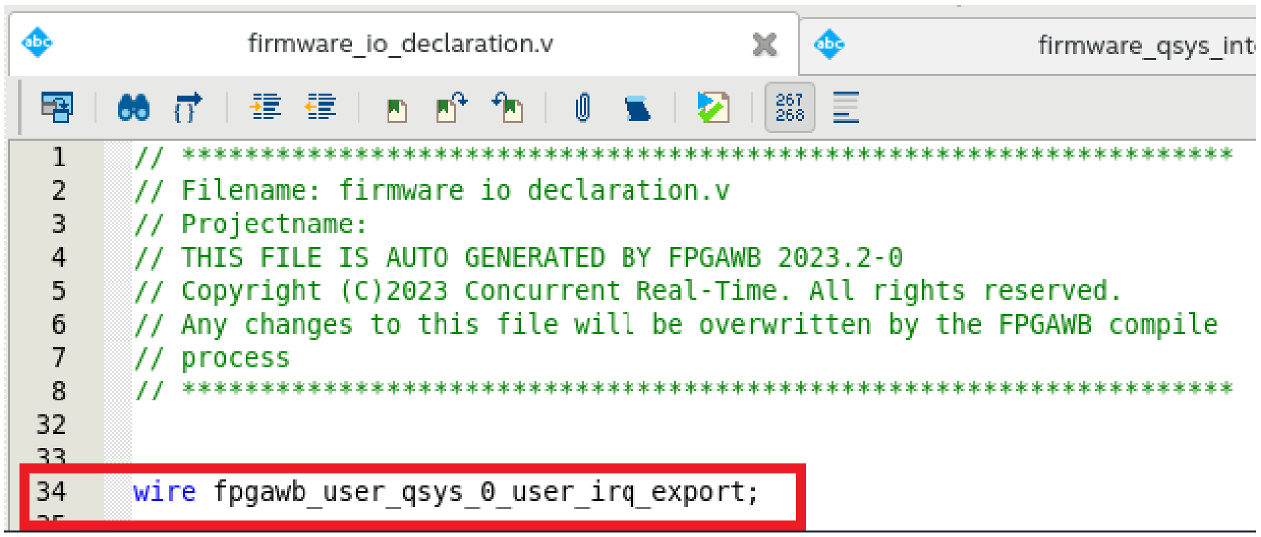
Do not change the exported name from “fpgawb_user_qsys_0_user_irq”.

2. **Connect the signal down multiple levels to meet the custom logic.**
   a. Add this line of Verilog code to the firmware_qsys_interface.v file that is generated in the working project directory of the FPGAWB project.

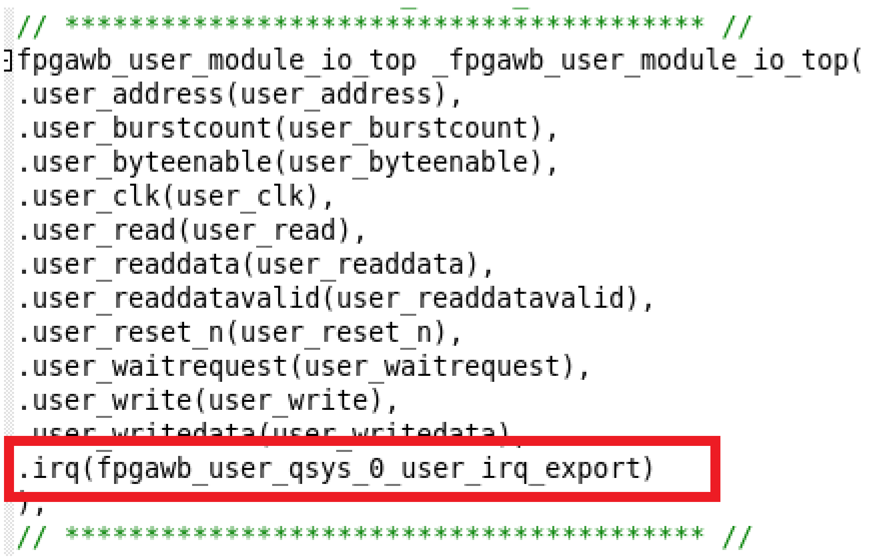

firmware_io_declaration.v | firmware_qsys_interface.v

```
// ********************************************************************
// Filename: firmware_qsy_interface.v
// Projectname: MarkyboyUserModule
// THIS FILE IS AUTO GENERATED BY FPGAWB
// Copyright (C)2023 Concurrent Real-Time. All rights reserved.
// Any changes to this file will be overwritten by the FPGAWB compile
// process
// ********************************************************************

// USER_MODULE QYS Interface
.fpgawb_user_qsys_0_user_address_export          (fpgawb_user_qsys_0_user_address_export),
.fpgawb_user_qsys_0_user_burstcount_export       (fpgawb_user_qsys_0_user_burstcount_export),
.fpgawb_user_qsys_0_user_byteenable_export       (fpgawb_user_qsys_0_user_byteenable_export),
.fpgawb_user_qsys_0_user_clk_export              (fpgawb_user_qsys_0_user_clk_export),
.fpgawb_user_qsys_0_user_read_export             (fpgawb_user_qsys_0_user_read_export),
.fpgawb_user_qsys_0_user_readdata_export         (fpgawb_user_qsys_0_user_readdata_export),
.fpgawb_user_qsys_0_user_readdatavalid_export    (fpgawb_user_qsys_0_user_readdatavalid_export)
.fpgawb_user_qsys_0_user_reset_n_export          (fpgawb_user_qsys_0_user_reset_n_export),
.fpgawb_user_qsys_0_user_waitrequest_export      (fpgawb_user_qsys_0_user_waitrequest_export),
.fpgawb_user_qsys_0_user_write_export            (fpgawb_user_qsys_0_user_write_export),
.fpgawb_user_qsys_0_user_writedata_export        (fpgawb_user_qsys_0_user_writedata_export),
.fpgawb_user_qsys_0_user_irq_export              (fpgawb_user_qsys_0_user_irq_export),
```

b. Add these two lines of Verilog code to the firmware_io_declaration.v file that is generated in the working project directory of the FPGAWB project.


firmware_io_declaration.v | firmware_qsys_int

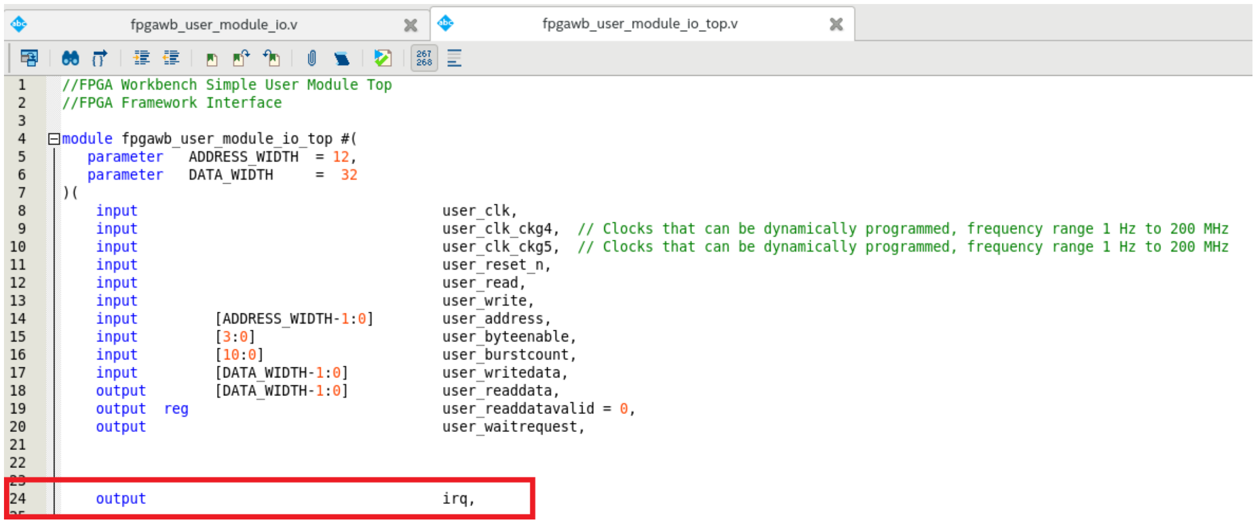
```
// ********************************************************************
// Filename: firmware io declaration.v
// Projectname:
// THIS FILE IS AUTO GENERATED BY FPGAWB 2023.2-0
// Copyright (C)2023 Concurrent Real-Time. All rights reserved.
// Any changes to this file will be overwritten by the FPGAWB compile
// process
// ********************************************************************


wire fpgawb_user_qsys_0_user_irq_export;
```



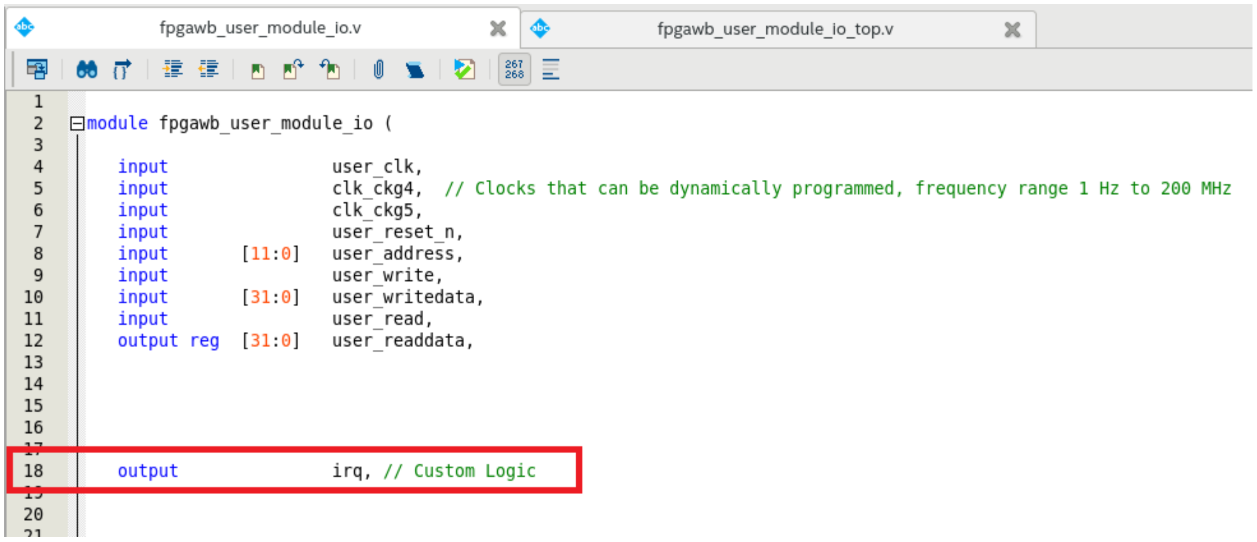
```
// ****************************************** //
fpgawb_user_module_io_top _fpgawb_user_module_io_top(
.user_address(user_address),
.user_burstcount(user_burstcount),
.user_byteenable(user_byteenable),
.user_clk(user_clk),
.user_read(user_read),
.user_readdata(user_readdata),
.user_readdatavalid(user_readdatavalid),
.user_reset_n(user_reset_n),
.user_waitrequest(user_waitrequest),
.user_write(user_write),
.user_writedata(user_writedata),
.irq(fpgawb_user_qsys_0_user_irq_export)
);
// ****************************************** //
```

In the image above the name of the signal in the fpgawb_user_module_io_top port list is shown to be ‘irq’. This was an arbitrary name for this example. The signal is simply a single bit from custom logic, and so in this image and the following two images ‘irq’ may be replaced with some other signal name.

c. Create a custom logic signal that ties through the fpgawb_user_module_io_top.v and fpgawb_user_module_io.v files found in the **fpgawb_user_module_io** directory that is auto-generated in the project.

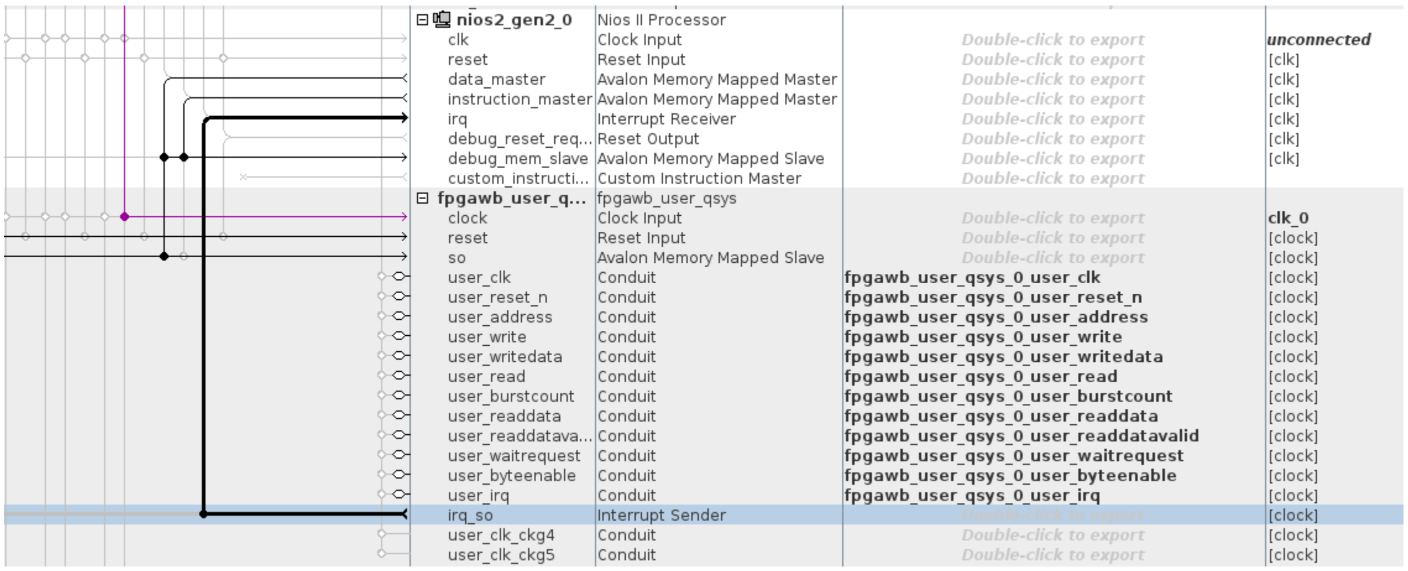


```
//FPGA Workbench Simple User Module Top
//FPGA Framework Interface

module fpgawb_user_module_io_top #(
   parameter   ADDRESS_WIDTH  = 12,
   parameter   DATA_WIDTH     =  32
)(
    input                                      user_clk,
    input                                      user_clk_ckg4,  // Clocks that can be dynamically programmed, frequency range 1 Hz to 200 MHz
    input                                      user_clk_ckg5,  // Clocks that can be dynamically programmed, frequency range 1 Hz to 200 MHz
    input                                      user_reset_n,
    input                                      user_read,
    input                                      user_write,
    input         [ADDRESS_WIDTH-1:0]          user_address,
    input         [3:0]                        user_byteenable,
    input         [10:0]                       user_burstcount,
    input         [DATA_WIDTH-1:0]             user_writedata,
    output        [DATA_WIDTH-1:0]             user_readdata,
    output  reg                                user_readdatavalid = 0,
    output                                     user_waitrequest,


    output                                     irq,
```

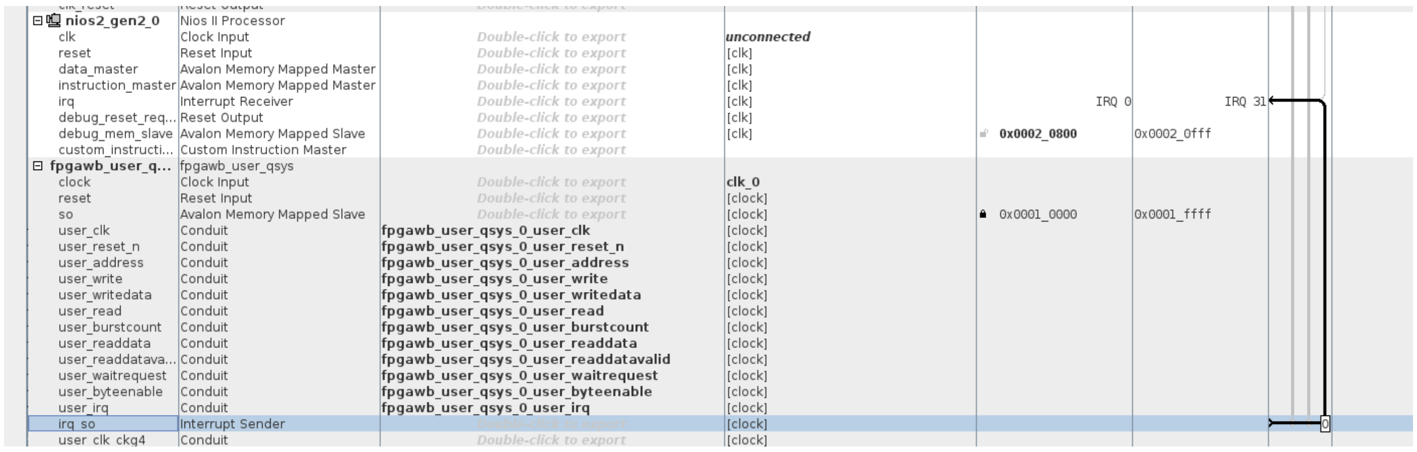


```
module fpgawb_user_module_io (

    input                user_clk,
    input                clk_ckg4,  // Clocks that can be dynamically programmed, frequency range 1 Hz to 200 MHz
    input                clk_ckg5,
    input                user_reset_n,
    input         [11:0] user_address,
    input                user_write,
    input         [31:0] user_writedata,
    input                user_read,
    output reg    [31:0] user_readdata,




    output               irq, // Custom Logic

```

3. Connect the Interrupt Sender to the desired Interrupt Receiver.



nios2_gen2_0	Nios II Processor		unconnected
clk	Clock Input	Double-click to export	[clk]
reset	Reset Input	Double-click to export	[clk]
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]
irq	Interrupt Receiver	Double-click to export	[clk]
debug_reset_req...	Reset Output	Double-click to export	[clk]
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]
custom_instructi...	Custom Instruction Master	Double-click to export	
fpgawb_user_q...	fpgawb_user_qsys		
clock	Clock Input	Double-click to export	clk_0
reset	Reset Input	Double-click to export	[clock]
so	Avalon Memory Mapped Slave	Double-click to export	[clock]
user_clk	Conduit	fpgawb_user_qsys_0_user_clk	[clock]
user_reset_n	Conduit	fpgawb_user_qsys_0_user_reset_n	[clock]
user_address	Conduit	fpgawb_user_qsys_0_user_address	[clock]
user_write	Conduit	fpgawb_user_qsys_0_user_write	[clock]
user_writedata	Conduit	fpgawb_user_qsys_0_user_writedata	[clock]
user_read	Conduit	fpgawb_user_qsys_0_user_read	[clock]
user_burstcount	Conduit	fpgawb_user_qsys_0_user_burstcount	[clock]
user_readdata	Conduit	fpgawb_user_qsys_0_user_readdata	[clock]
user_readdatava...	Conduit	fpgawb_user_qsys_0_user_readdatavalid	[clock]
user_waitrequest	Conduit	fpgawb_user_qsys_0_user_waitrequest	[clock]
user_byteenable	Conduit	fpgawb_user_qsys_0_user_byteenable	[clock]
user_irq	Conduit	fpgawb_user_qsys_0_user_irq	[clock]
irq_so	Interrupt Sender	Double-click to export	[clock]
user_clk_ckg4	Conduit	Double-click to export	[clock]
user_clk_ckg5	Conduit	Double-click to export	[clock]

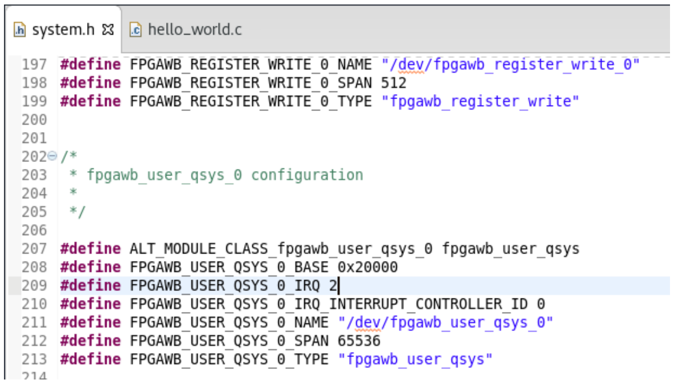


clk_reset	Reset Output	Double-click to export				
⊟ nios2_gen2_0	Nios II Processor					
clk	Clock Input	Double-click to export	unconnected			
reset	Reset Input	Double-click to export	[clk]			
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
irq	Interrupt Receiver	Double-click to export	[clk]		IRQ 0	IRQ 31
debug_reset_req...	Reset Output	Double-click to export	[clk]			
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0002_0800	0x0002_0fff	
custom_instructi...	Custom Instruction Master	Double-click to export				
⊟ fpgawb_user_q...	fpgawb_user_qsys					
clock	Clock Input	Double-click to export	clk_0			
reset	Reset Input	Double-click to export	[clock]			
so	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0001_0000	0x0001_ffff	
user_clk	Conduit	fpgawb_user_qsys_0_user_clk	[clock]			
user_reset_n	Conduit	fpgawb_user_qsys_0_user_reset_n	[clock]			
user_address	Conduit	fpgawb_user_qsys_0_user_address	[clock]			
user_write	Conduit	fpgawb_user_qsys_0_user_write	[clock]			
user_writedata	Conduit	fpgawb_user_qsys_0_user_writedata	[clock]			
user_read	Conduit	fpgawb_user_qsys_0_user_read	[clock]			
user_burstcount	Conduit	fpgawb_user_qsys_0_user_burstcount	[clock]			
user_readdata	Conduit	fpgawb_user_qsys_0_user_readdata	[clock]			
user_readdatava...	Conduit	fpgawb_user_qsys_0_user_readdatavalid	[clock]			
user_waitrequest	Conduit	fpgawb_user_qsys_0_user_waitrequest	[clock]			
user_byteenable	Conduit	fpgawb_user_qsys_0_user_byteenable	[clock]			
user_irq	Conduit	fpgawb_user_qsys_0_user_irq	[clock]			
irq_so	Interrupt Sender	Double-click to export	[clock]			0
user_clk_ckg4	Conduit	Double-click to export	[clock]			

Make sure to set the interrupt priority to the desired level based on the receiver's application.

• Setting up Software

The job of the software is to be able to handle whenever there is a hardware interrupt and have an ISR to enter. The first step is to change some variable definitions inside of the system.h file.

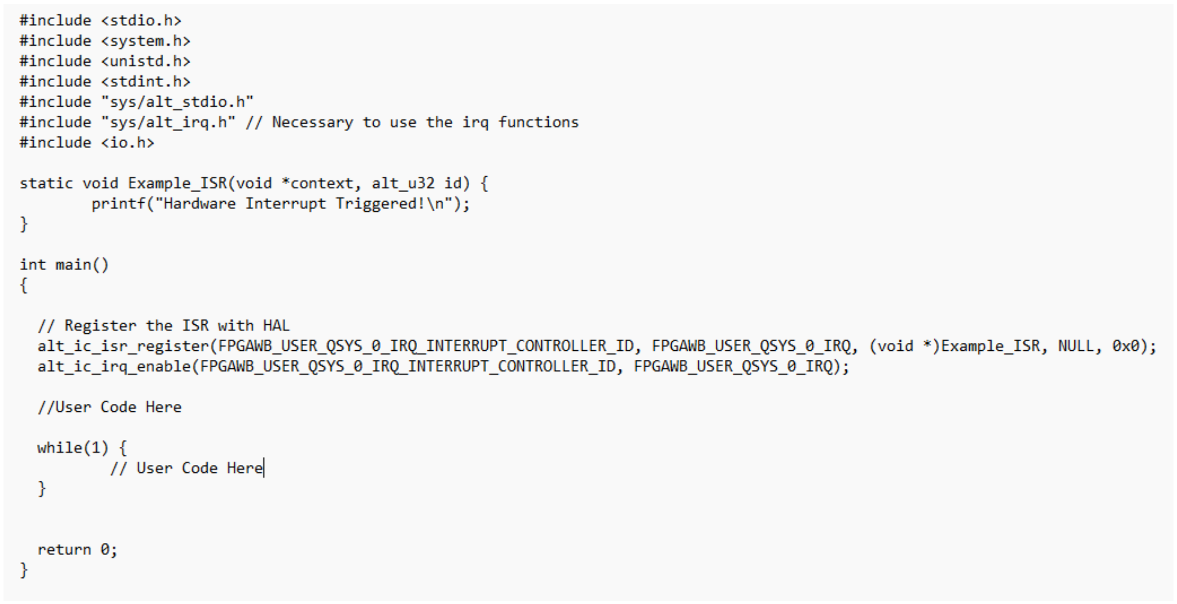
1. Change system.h file
   a. Go to the system.h file and find the definition of FPGAWB_USER_QSYS_0_IRQ
   b. By default FPGAWB_USER_QSYS_0_IRQ is defined as -1, it should reflect the priority value that was given to it inside of Platform Designer when the Interrupt sender was connected to the Nios.
   c. Along with this, FPGAWB_USER_QSYS_0_IRQ_INTERRUPT_CONTROLLER_ID should be changed to 0.
   d. **NOTE: Anytime the BSP is generated parts a-c must be done.**
   e. The photo below shows how it should look.
   f. 

system.h ⊠ | hello_world.c

```
197 #define FPGAWB_REGISTER_WRITE_0_NAME "/dev/fpgawb_register_write_0"
198 #define FPGAWB_REGISTER_WRITE_0_SPAN 512
199 #define FPGAWB_REGISTER_WRITE_0_TYPE "fpgawb_register_write"
200
201
202 /*
203  * fpgawb_user_qsys_0 configuration
204  *
205  */
206
207 #define ALT_MODULE_CLASS_fpgawb_user_qsys_0 fpgawb_user_qsys
208 #define FPGAWB_USER_QSYS_0_BASE 0x20000
209 #define FPGAWB_USER_QSYS_0_IRQ 2
210 #define FPGAWB_USER_QSYS_0_IRQ_INTERRUPT_CONTROLLER_ID 0
211 #define FPGAWB_USER_QSYS_0_NAME "/dev/fpgawb_user_qsys_0"
212 #define FPGAWB_USER_QSYS_0_SPAN 65536
213 #define FPGAWB_USER_QSYS_0_TYPE "fpgawb_user_qsys"
```

   g. Lines 209 and 210 are the lines that need to be edited. The value '2' on line 209 of the example photo is dependent on the value set to the interrupt sender priority in Platform Designer.

2. Register the ISR with HAL

   a. 

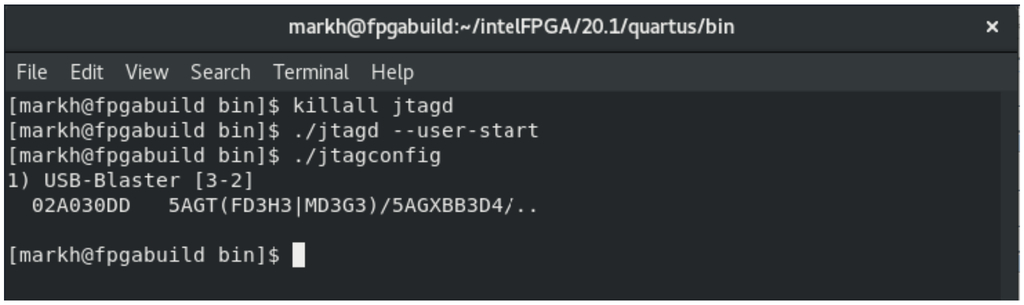
```
#include <stdio.h>
#include <system.h>
#include <unistd.h>
#include <stdint.h>
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h" // Necessary to use the irq functions
#include <io.h>

static void Example_ISR(void *context, alt_u32 id) {
        printf("Hardware Interrupt Triggered!\n");
}

int main()
{

  // Register the ISR with HAL
  alt_ic_isr_register(FPGAWB_USER_QSYS_0_IRQ_INTERRUPT_CONTROLLER_ID, FPGAWB_USER_QSYS_0_IRQ, (void *)Example_ISR, NULL, 0x0);
  alt_ic_irq_enable(FPGAWB_USER_QSYS_0_IRQ_INTERRUPT_CONTROLLER_ID, FPGAWB_USER_QSYS_0_IRQ);

  //User Code Here

  while(1) {
          // User Code Here
  }


  return 0;
}
```

   b. The photo above shows all the code necessary to enable the hardware interrupt.

At this point the C code and software should have everything necessary to catch the hardware interrupt and enter the ISR when triggered.

Setting Up JTAG USB Blaster

This step is necessary to use JTAG USB Blaster to flash .elf files to the Nios.

1. Create a file called "51-usbblaster.rules" in the directory "/etc/udev/rules.d/51-usbblaster.rules"
   a. If the folder rules.d doesn't exist, create it in the udev directory.
2. Add the following line of code to 51-usbblaster.rules
   a. ATTR{idVendor}=="09fb", ATTR{idProduct}=="6001", MODE="0666"
3. The parameters types in here depend on the type of USB Blaster being used. Type the following terminal command to determine these values:
   a. lsusb | grep Altera
   b. In this case Altera is the company brand used for the USB Blaster, which is why Altera is used in the grep command, if a different company created the USB Blaster in question type that name instead.
   c. The command should print to console the values needed for the .rules file.
4. Now the setup should be finished, after this there are a set of commands useful when using JTAG and can all be found in the < Intel® Quartus® Prime installation directory>/quartus/bin> directory.
5. The following procedure can be useful when wanting to do a sort of "Reset" to the JTAG connection.
   a. 
markh@fpgabuild:~/intelFPGA/20.1/quartus/bin
File Edit View Search Terminal Help

```
[markh@fpgabuild bin]$ killall jtagd
[markh@fpgabuild bin]$ ./jtagd --user-start
[markh@fpgabuild bin]$ ./jtagconfig
1) USB-Blaster [3-2]
  02A030DD   5AGT(FD3H3|MD3G3)/5AGXBB3D4/..

[markh@fpgabuild bin]$
```

6. If ./jtagconfig command still doesn't print out something similar to what is in the image above, try the following.
   a. Sudo killall -9 jtagd (multiple times until)

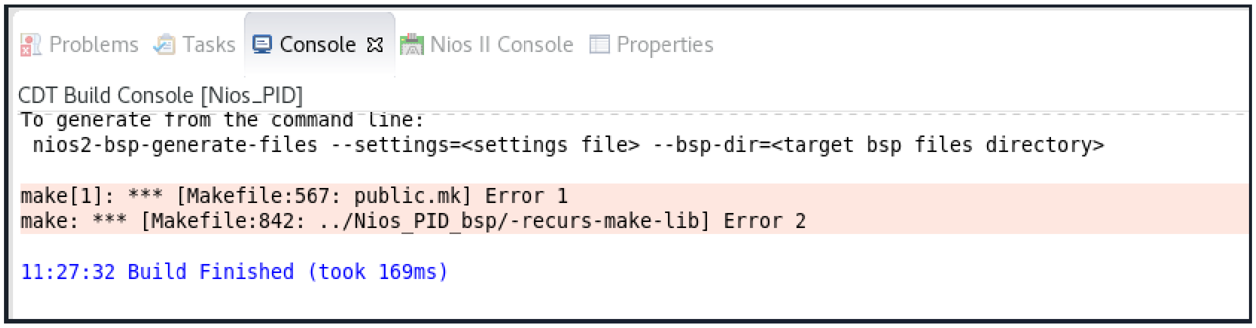
The Reset process of the JTAG may be necessary every time wanting to use JTAG on initial startup.

The ./jtagconfig command is the best way to determine if the connection to the FPGA is made correctly.

Misc. Problems/Errors

▪ Makefile Error

If you build your project and get an error similar to the following error:



```
To generate from the command line:
 nios2-bsp-generate-files --settings=<settings file> --bsp-dir=<target bsp files directory>

make[1]: *** [Makefile:567: public.mk] Error 1
make: *** [Makefile:842: ../Nios_PID_bsp/-recurs-make-lib] Error 2

11:27:32 Build Finished (took 169ms)
```

This occurs because the firmware has changed or it is an initial startup the solution is simple:

1. Right click the C project folder in the left panel of Nios II Eclipse
2. Scroll to the bottom of the options and hover over “Nios II”
3. Select “Generate BSP”
4. Now build the project and the error should be gone.