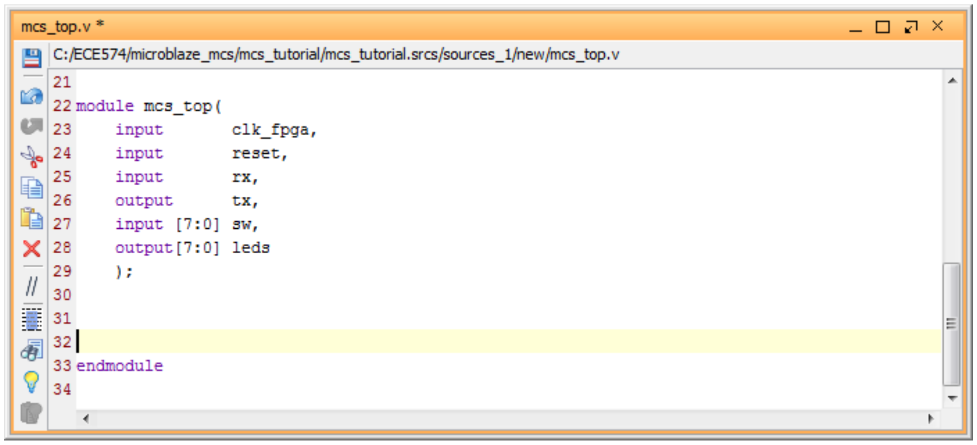
Microblaze MCS Tutorial for Xilinx Vivado 2015.1

This tutorial shows how to add a Microblaze Microcontroller System (MCS) embedded processor to a project including adding a simple C program. The design was targeted to an Artix 7 FPGA (on a Nexys4DDR board) but the steps should be general enough to work on other platforms.

Create a new project and create a top level module for the inputs and outputs we will use with the microblaze MCS. Create a constraints file to connect the ports to the appropriate FPGA pins to match your board.

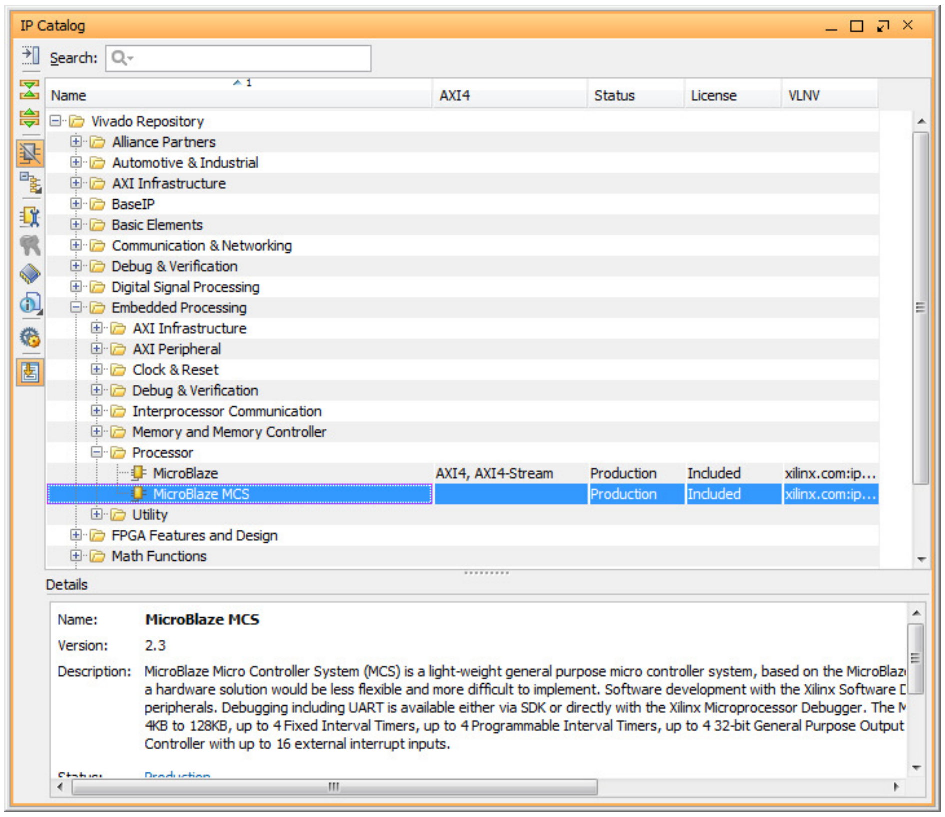


mcs_top.v *

C:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/sources_1/new/mcs_top.v

```
module mcs_top(
    input       clk_fpga,
    input       reset,
    input       rx,
    output      tx,
    input [7:0] sw,
    output[7:0] leds
    );



endmodule
```

Select IP Catalog from the Project Manager and then select ‘Microblaze MCS’:



IP Catalog

Search:

Name	AXI4	Status	License	VLNV
Vivado Repository				
Alliance Partners				
Automotive & Industrial				
AXI Infrastructure				
BaseIP				
Basic Elements				
Communication & Networking				
Debug & Verification				
Digital Signal Processing				
Embedded Processing				
AXI Infrastructure				
AXI Peripheral				
Clock & Reset				
Debug & Verification				
Interprocessor Communication				
Memory and Memory Controller				
Processor				
MicroBlaze	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip...
MicroBlaze MCS		Production	Included	xilinx.com:ip...
Utility				
FPGA Features and Design				
Math Functions				

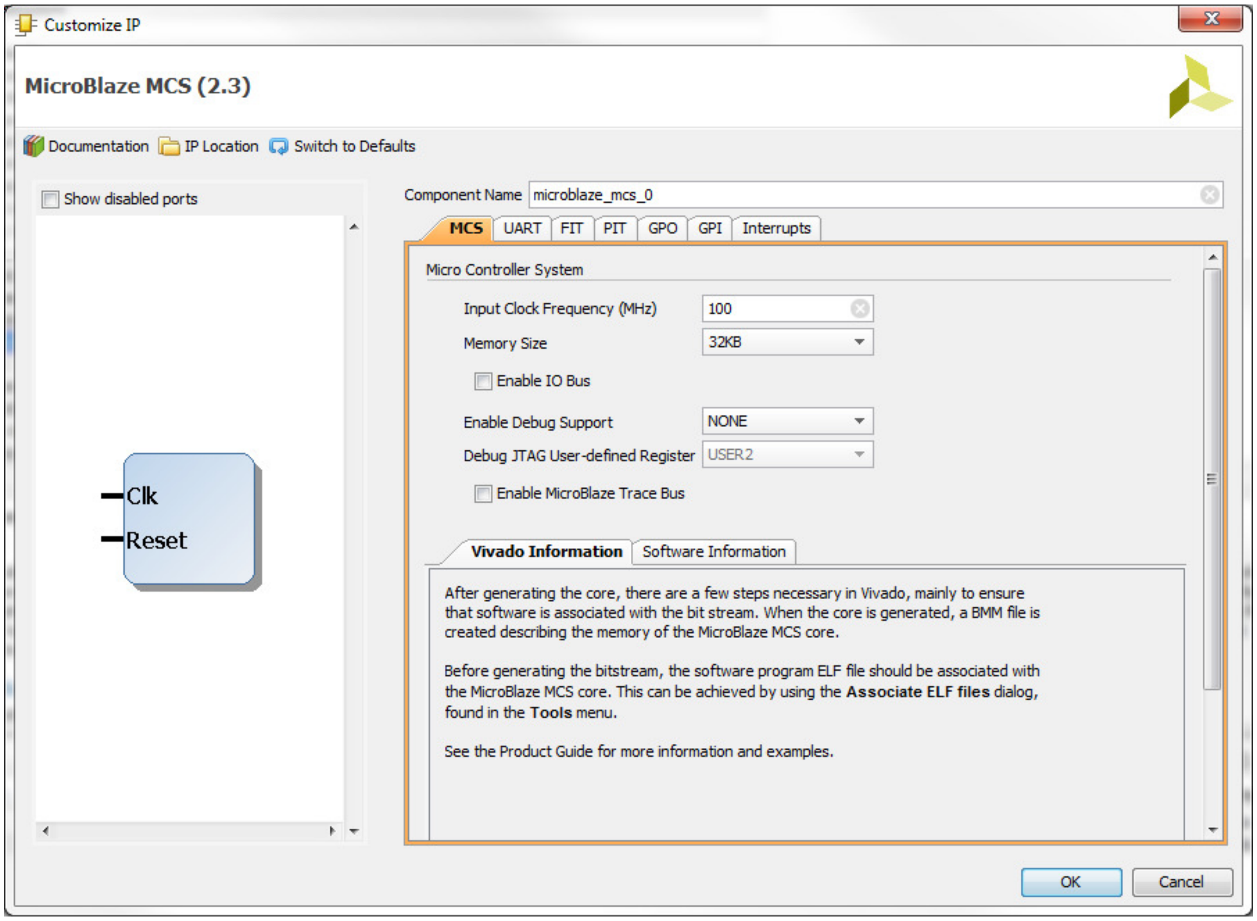
Details

Name: **MicroBlaze MCS**

Version: 2.3

Description: MicroBlaze Micro Controller System (MCS) is a light-weight general purpose micro controller system, based on the MicroBlaz
a hardware solution would be less flexible and more difficult to implement. Software development with the Xilinx Software D
peripherals. Debugging including UART is available either via SDK or directly with the Xilinx Microprocessor Debugger. The M
4KB to 128KB, up to 4 Fixed Interval Timers, up to 4 Programmable Interval Timers, up to 4 32-bit General Purpose Output
Controller with up to 16 external interrupt inputs.

The Customize IP for the Microblaze MCS opens:



Component Name: microblaze_mcs_0

Micro Controller System

Input Clock Frequency (MHz): 100

Memory Size: 32KB

Enable IO Bus: ☐

Enable Debug Support: NONE

Debug JTAG User-defined Register: USER2

Enable MicroBlaze Trace Bus: ☐

After generating the core, there are a few steps necessary in Vivado, mainly to ensure that software is associated with the bit stream. When the core is generated, a BMM file is created describing the memory of the MicroBlaze MCS core.

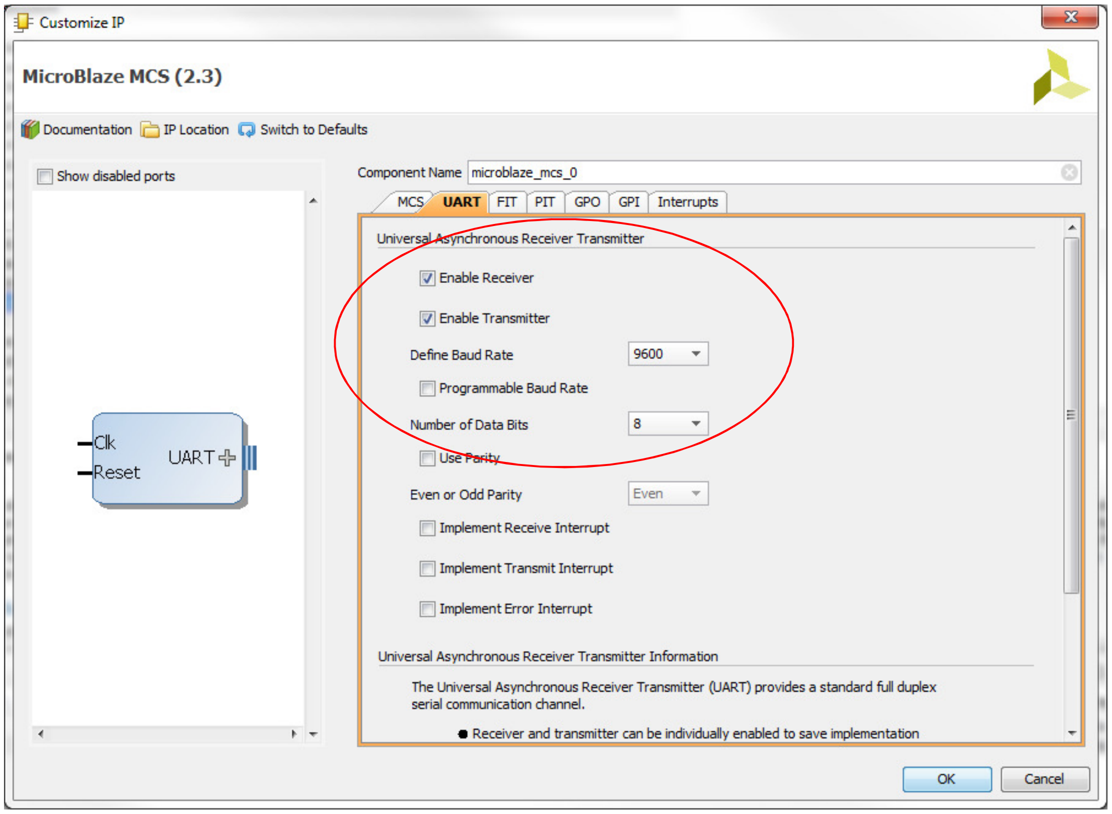
Before generating the bitstream, the software program ELF file should be associated with the MicroBlaze MCS core. This can be achieved by using the **Associate ELF files** dialog, found in the **Tools** menu.

See the Product Guide for more information and examples.

We can use the default component name of microblaze_mcs_0.

- Set the Input Clock Frequency to match your Nexys4DDR board (100MHz)
- Increase the memory size from 8KB to 32KB (allows for slightly larger C program)

Select the UART Tab and enable the receiver and transmitter and select your baud rate:



Component Name microblaze_mcs_0

Universal Asynchronous Receiver Transmitter

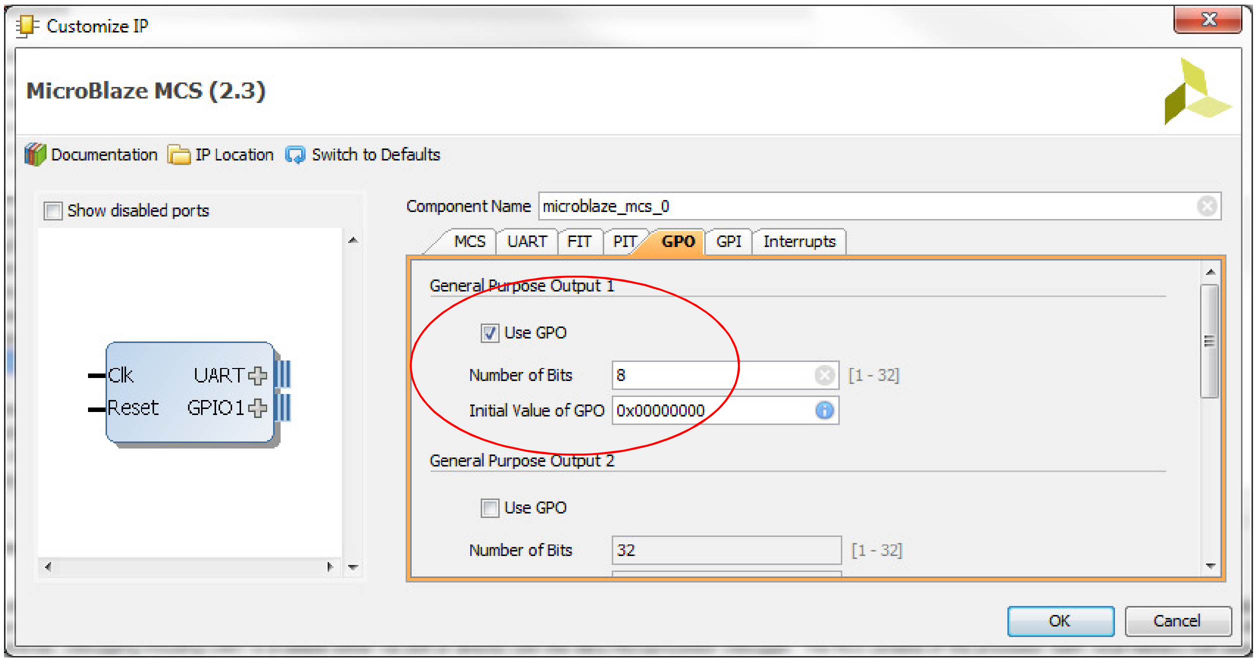
- [✓] Enable Receiver
- [✓] Enable Transmitter
- Define Baud Rate: 9600
- [ ] Programmable Baud Rate
- Number of Data Bits: 8
- [ ] Use Parity
- Even or Odd Parity: Even
- [ ] Implement Receive Interrupt
- [ ] Implement Transmit Interrupt
- [ ] Implement Error Interrupt

Universal Asynchronous Receiver Transmitter Information

The Universal Asynchronous Receiver Transmitter (UART) provides a standard full duplex serial communication channel.

- Receiver and transmitter can be individually enabled to save implementation

Add an 8-bit GPO (we will connect to LEDs later):



Component Name microblaze_mcs_0

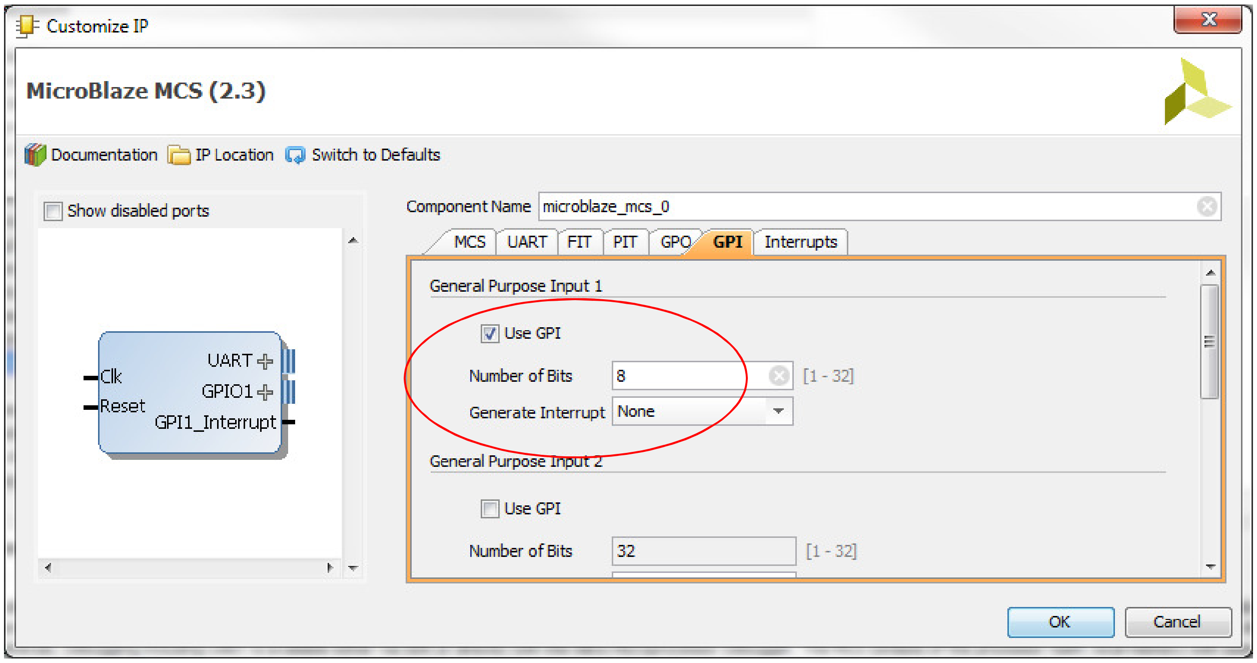
General Purpose Output 1

- [✓] Use GPO
- Number of Bits: 8 [1 - 32]
- Initial Value of GPO: 0x00000000

General Purpose Output 2

- [ ] Use GPO
- Number of Bits: 32 [1 - 32]

Add an 8-bit GPI (we will connect to the slider switches later):



Component Name: microblaze_mcs_0

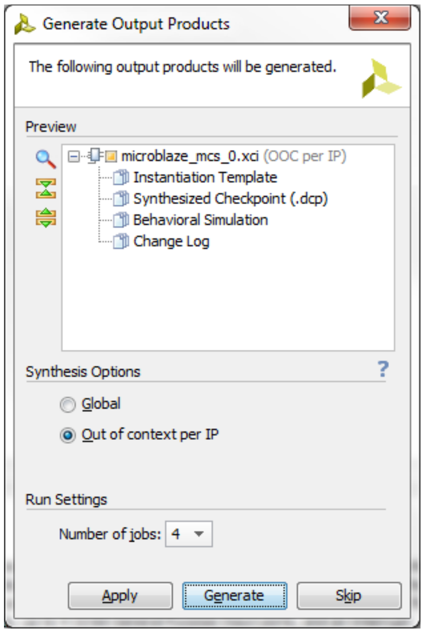
General Purpose Input 1

- [✓] Use GPI
- Number of Bits: 8 [1 - 32]
- Generate Interrupt: None

General Purpose Input 2

- [ ] Use GPI
- Number of Bits: 32 [1 - 32]

Click on **OK**



The following output products will be generated.

- microblaze_mcs_0.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Behavioral Simulation
  - Change Log

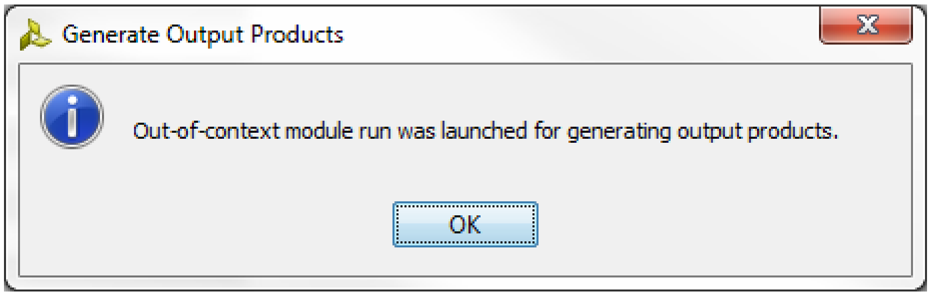
Synthesis Options

- ( ) Global
- (●) Out of context per IP

Run Settings

Number of jobs: 4

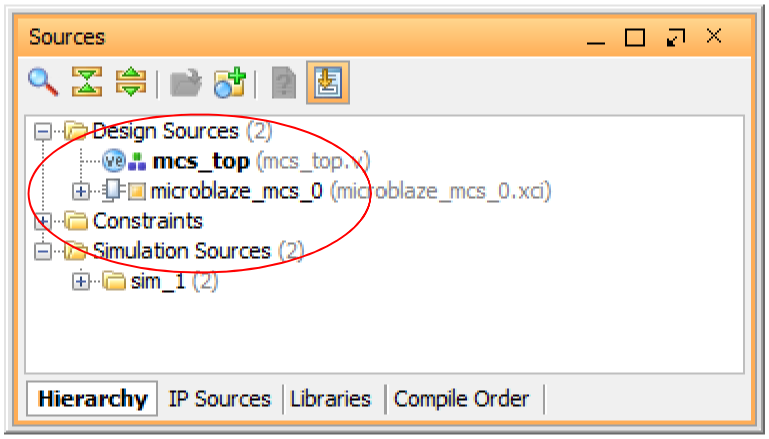
Click on **Generate**



Out-of-context module run was launched for generating output products.

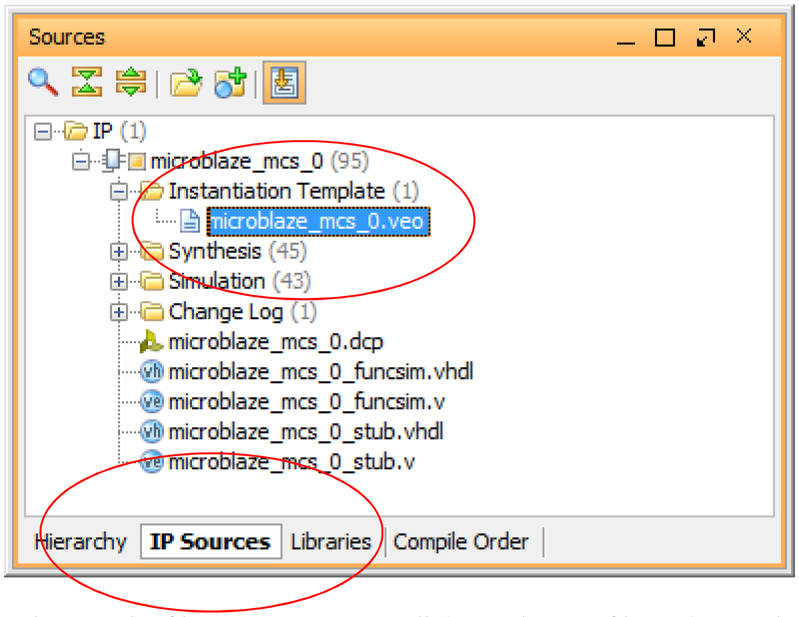
Click **OK**

You will now see the microblaze_mcs_0 core in the Design Sources window

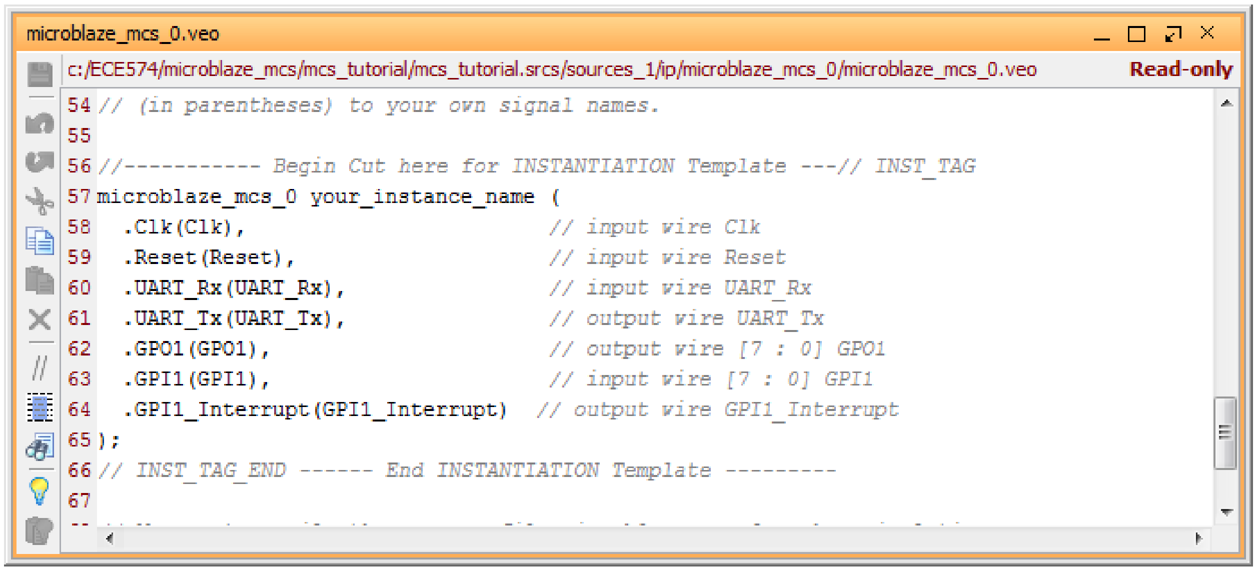


We now need to access the Instantiation Template so we can add the mcs to our top level module.

Select the IP sources tab then select the template:



Selecting the file you can now scroll down the veo file and copy the template portion:

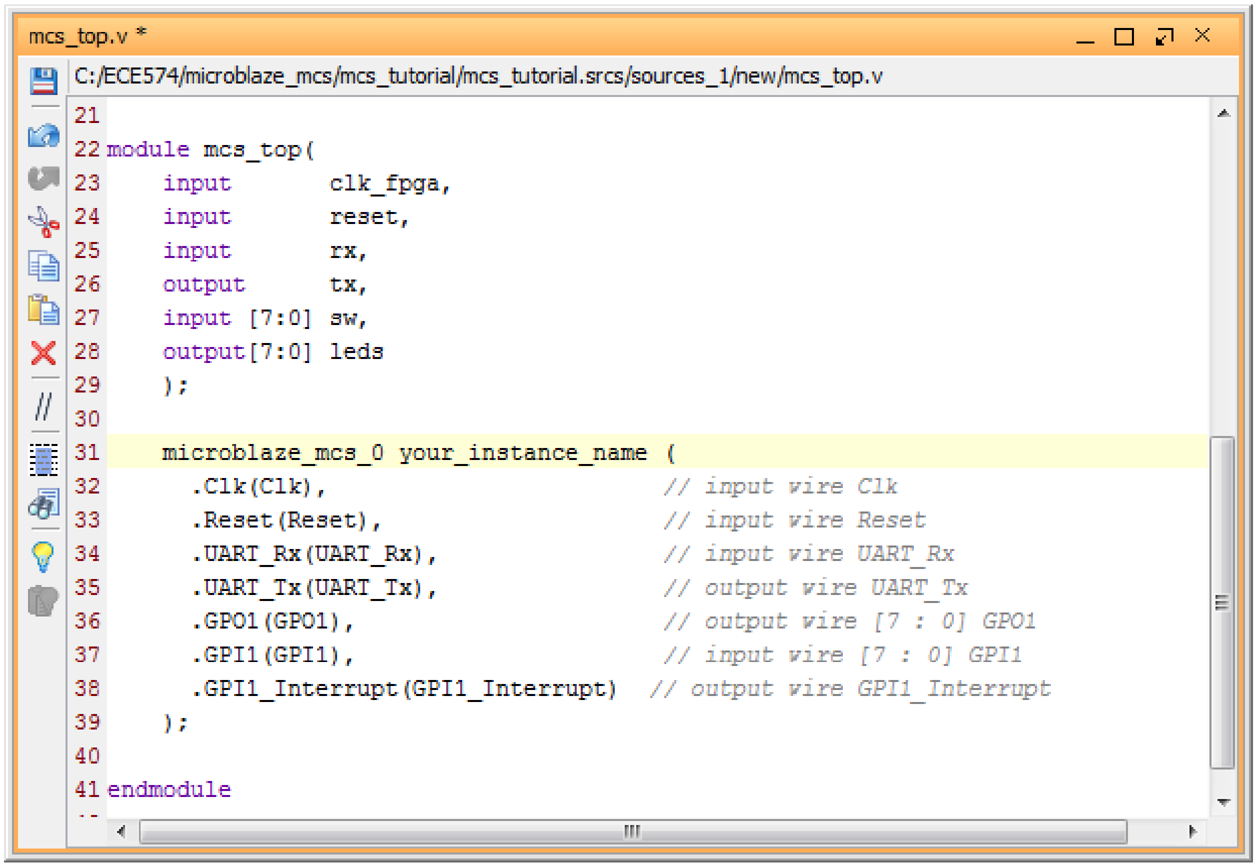


microblaze_mcs_0.veo

c:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/sources_1/ip/microblaze_mcs_0/microblaze_mcs_0.veo **Read-only**

```
// (in parentheses) to your own signal names.

//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
microblaze_mcs_0 your_instance_name (
  .Clk(Clk),                      // input wire Clk
  .Reset(Reset),                  // input wire Reset
  .UART_Rx(UART_Rx),              // input wire UART_Rx
  .UART_Tx(UART_Tx),              // output wire UART_Tx
  .GPO1(GPO1),                    // output wire [7 : 0] GPO1
  .GPI1(GPI1),                    // input wire [7 : 0] GPI1
  .GPI1_Interrupt(GPI1_Interrupt)  // output wire GPI1_Interrupt
);
// INST_TAG_END ------ End INSTANTIATION Template ---------

```

Copy the template (with control C) and then paste (control V) into your top level file:



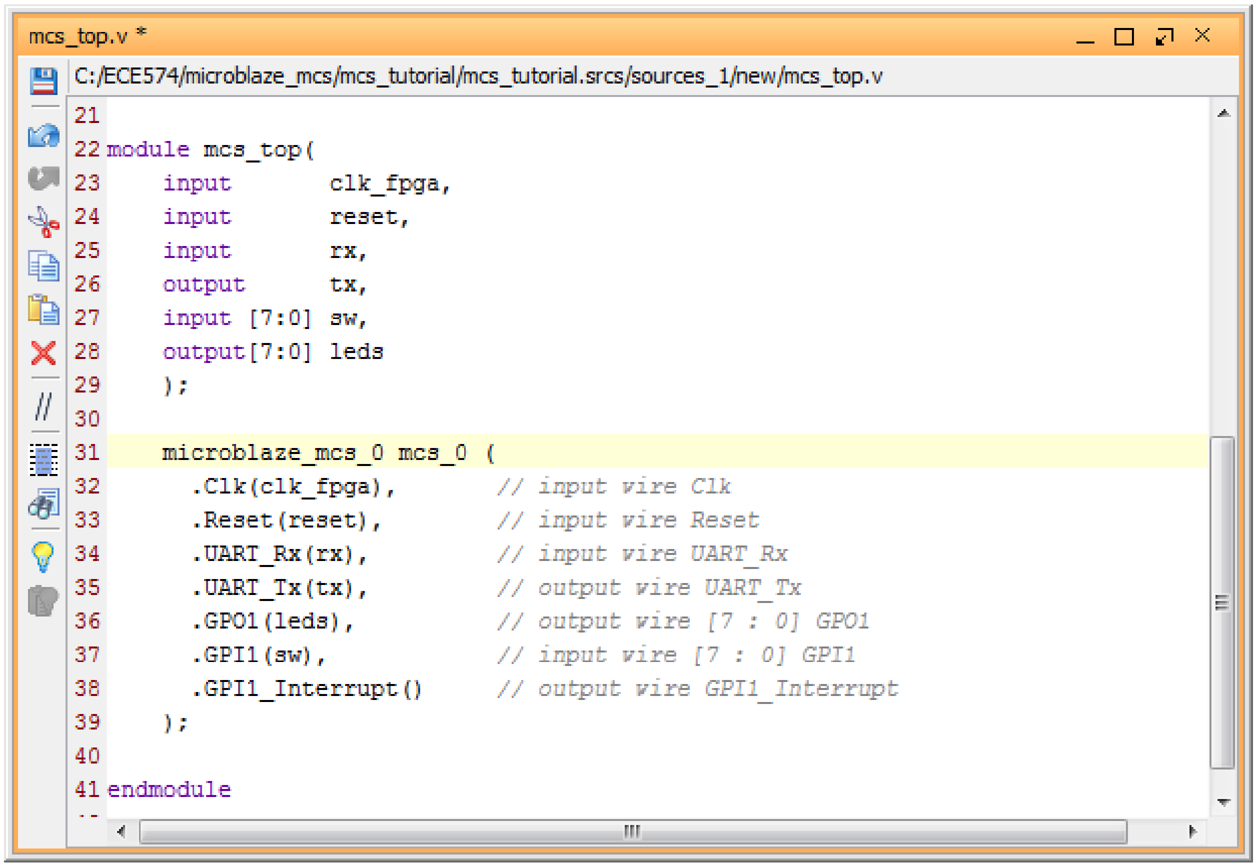
mcs_top.v *

C:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/sources_1/new/mcs_top.v

```

module mcs_top(
    input        clk_fpga,
    input        reset,
    input        rx,
    output       tx,
    input  [7:0] sw,
    output[7:0]  leds
    );

    microblaze_mcs_0 your_instance_name (
      .Clk(Clk),                      // input wire Clk
      .Reset(Reset),                  // input wire Reset
      .UART_Rx(UART_Rx),              // input wire UART_Rx
      .UART_Tx(UART_Tx),              // output wire UART_Tx
      .GPO1(GPO1),                    // output wire [7 : 0] GPO1
      .GPI1(GPI1),                    // input wire [7 : 0] GPI1
      .GPI1_Interrupt(GPI1_Interrupt)  // output wire GPI1_Interrupt
    );

endmodule
```

Change the instance_name to mcs_0 and modify the signal names in the instantiation to match your top level port names as shown below:

Note: you may see a GPI1_Interrupt signal (you can ignore this port – just leave it open)


mcs_top.v *

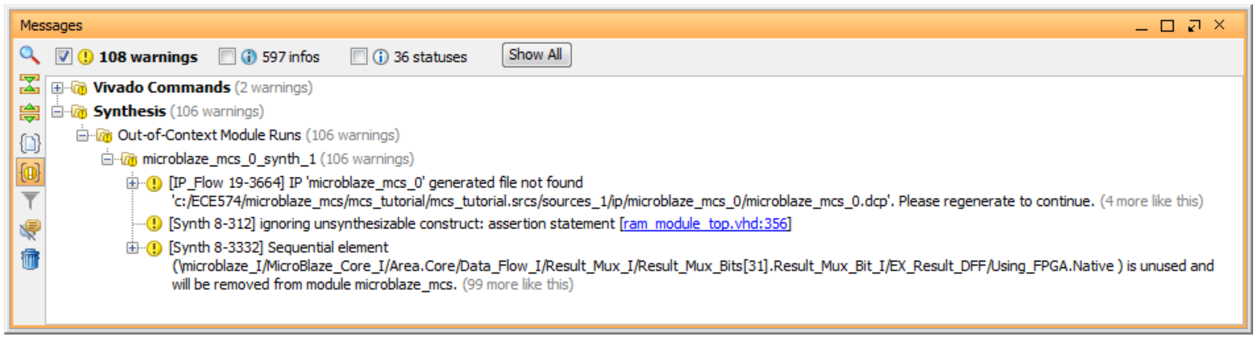
C:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/sources_1/new/mcs_top.v

```

module mcs_top(
    input        clk_fpga,
    input        reset,
    input        rx,
    output       tx,
    input  [7:0] sw,
    output[7:0]  leds
    );

    microblaze_mcs_0 mcs_0 (
      .Clk(clk_fpga),         // input wire Clk
      .Reset(reset),          // input wire Reset
      .UART_Rx(rx),           // input wire UART_Rx
      .UART_Tx(tx),           // output wire UART_Tx
      .GPO1(leds),            // output wire [7 : 0] GPO1
      .GPI1(sw),              // input wire [7 : 0] GPI1
      .GPI1_Interrupt()       // output wire GPI1_Interrupt
    );

endmodule
```

In this simple project we just have the microblaze_mcs module but of course you could add extra components or modules or other logic as required.

Select **Run Synthesis** to synthesize your project.

You will notice there are 106 Synthesis Warnings but you can ignore these!

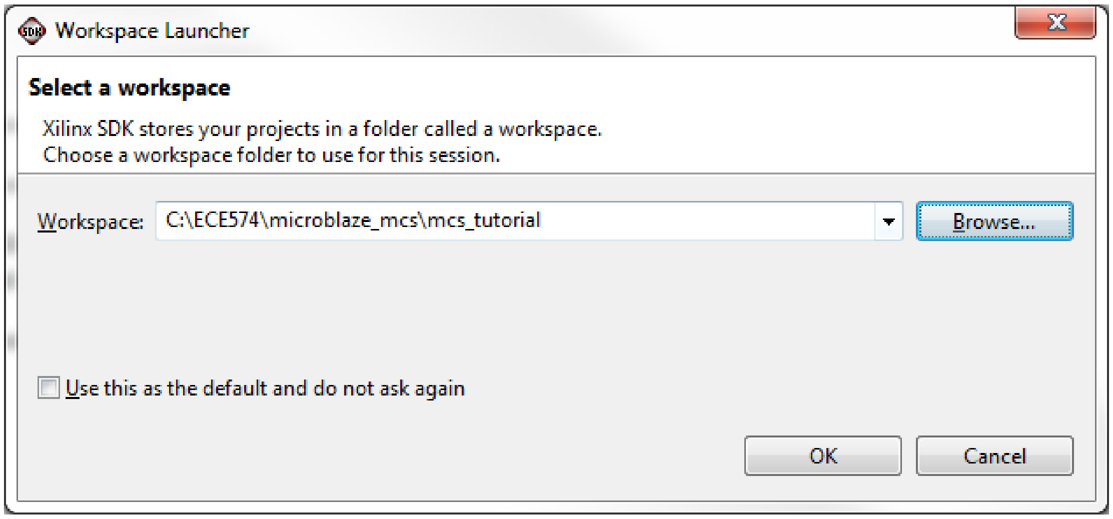


```
Vivado Commands (2 warnings)
Synthesis (106 warnings)
  Out-of-Context Module Runs (106 warnings)
    microblaze_mcs_0_synth_1 (106 warnings)
      [IP_Flow 19-3664] IP 'microblaze_mcs_0' generated file not found
       'c:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/sources_1/ip/microblaze_mcs_0/microblaze_mcs_0.dcp'. Please regenerate to continue. (4 more like this)
      [Synth 8-312] ignoring unsynthesizable construct: assertion statement [ram_module_top.vhd:356]
      [Synth 8-3332] Sequential element
       (\microblaze_I/MicroBlaze_Core_I/Area.Core/Data_Flow_I/Result_Mux_I/Result_Mux_Bits[31].Result_Mux_Bit_I/EX_Result_DFF/Using_FPGA.Native ) is unused and
       will be removed from module microblaze_mcs. (99 more like this)
```

Software Development

The next steps are related to the software development using SDK (Software Development Kit). You can download this from the Xilinx website if you do not already have this installed.

Start SDK and select the *Workspace* to match where your design is stored (for example the project is located in this example at C:\ece574\microblaze_mcs\mcs_tutorial):


Workspace Launcher

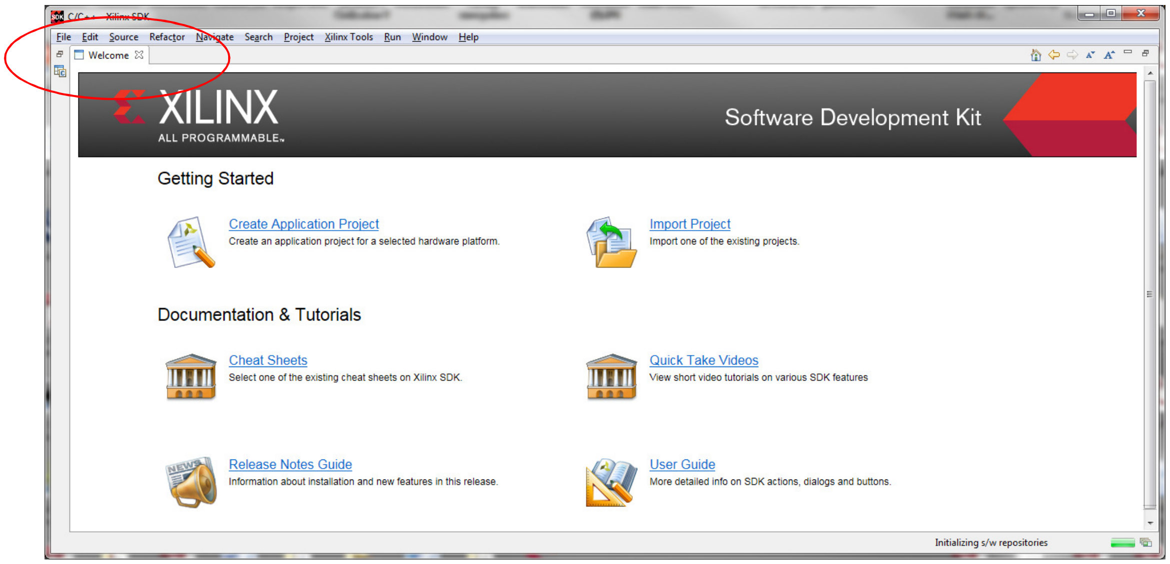
Select a workspace

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

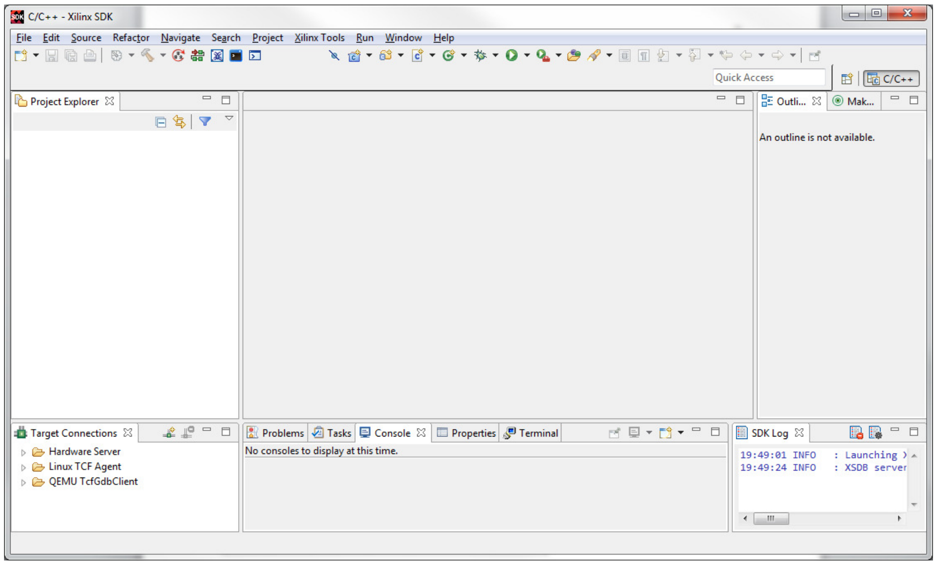
Workspace: C:\ECE574\microblaze_mcs\mcs_tutorial

Click **OK**

SDK Starts:


File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help
Welcome
XILINX
ALL PROGRAMMABLE.
Software Development Kit
Getting Started
Create Application Project
Create an application project for a selected hardware platform.
Import Project
Import one of the existing projects.
Documentation & Tutorials
Cheat Sheets
Select one of the existing cheat sheets on Xilinx SDK.
Quick Take Videos
View short video tutorials on various SDK features
Release Notes Guide
Information about installation and new features in this release.
User Guide
More detailed info on SDK actions, dialogs and buttons.
Initializing s/w repositories

Close the Welcome screen and the Project Explorer Window will open:



An outline is not available.

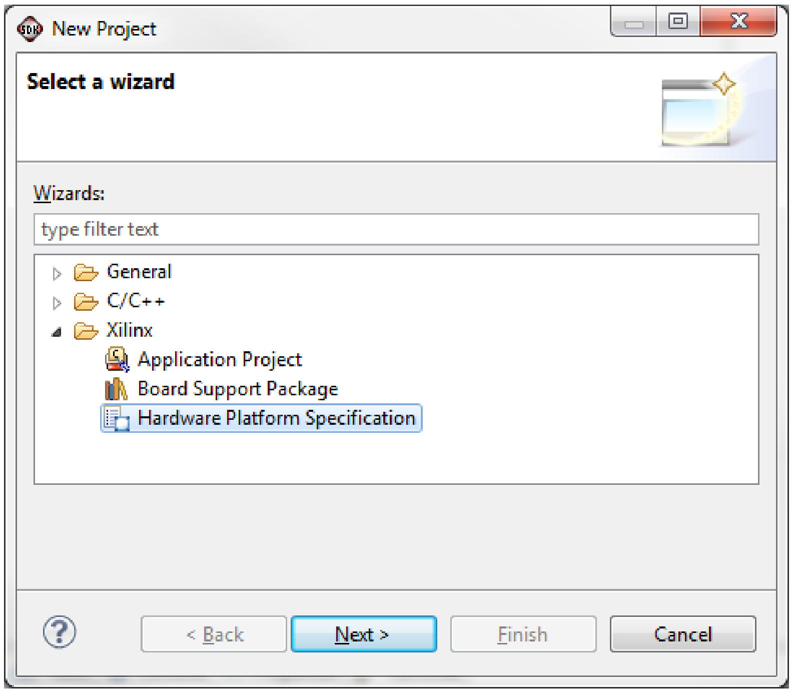
No consoles to display at this time.

```
19:49:01 INFO  : Launching X
19:49:24 INFO  : XSDB server
```

We need to import the Microblaze Hardware Description.

Select **File** -> **New -> Project** … in the menu

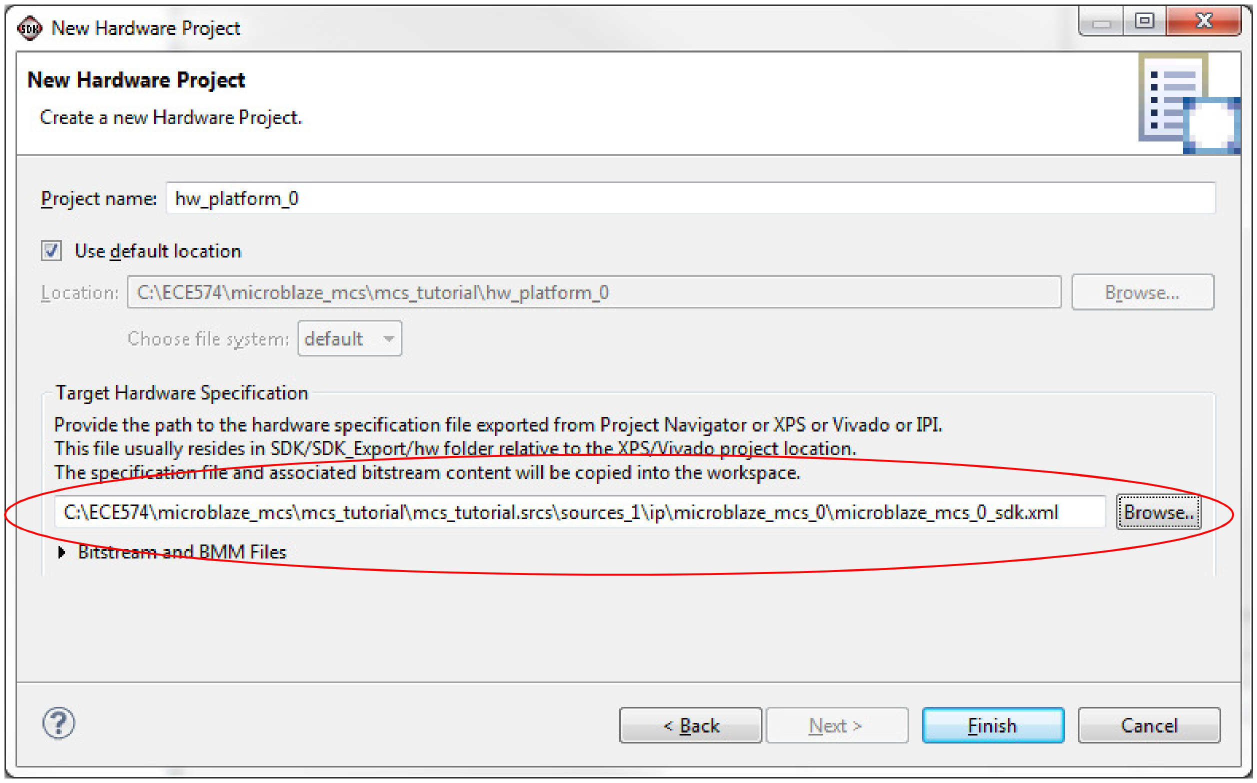
Expand Xilinx, and select *Hardware Platform Specification*



Click **Next**

Click **Browse** and navigate to the hardware description file which will be located at:

project-name.srcs/sources_1/ip/component_name/component-name_sdk.xml:



New Hardware Project

Create a new Hardware Project.

Project name: hw_platform_0

[✓] Use default location

Location: C:\ECE574\microblaze_mcs\mcs_tutorial\hw_platform_0 Browse...

Choose file system: default

Target Hardware Specification

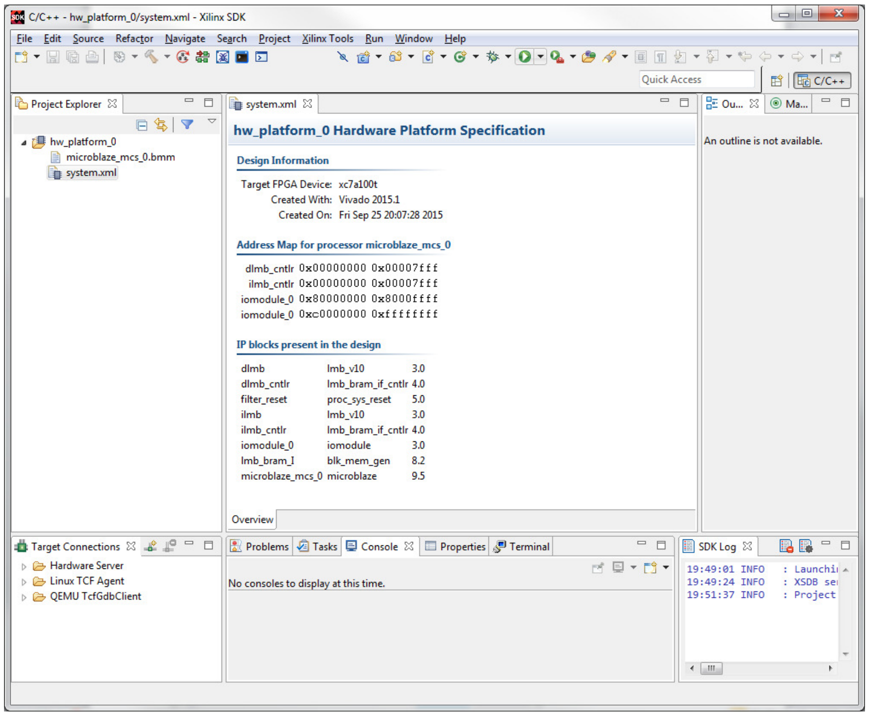
Provide the path to the hardware specification file exported from Project Navigator or XPS or Vivado or IPI.
This file usually resides in SDK/SDK_Export/hw folder relative to the XPS/Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

C:\ECE574\microblaze_mcs\mcs_tutorial\mcs_tutorial.srcs\sources_1\ip\microblaze_mcs_0\microblaze_mcs_0_sdk.xml Browse...

Bitstream and BMM Files

Note: At this point I modified the Project Name to just hw_platform_0 but you can leave with the default name provided when you provided the path to the hardware specification file.

Click **Finish** to perform the import



hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device: xc7a100t
Created With: Vivado 2015.1
Created On: Fri Sep 25 20:07:28 2015

Address Map for processor microblaze_mcs_0

dlmb_cntlr	0x00000000	0x00007fff
ilmb_cntlr	0x00000000	0x00007fff
iomodule_0	0x80000000	0x8000ffff
iomodule_0	0xc0000000	0xffffffff

IP blocks present in the design

dlmb	lmb_v10	3.0
dlmb_cntlr	lmb_bram_if_cntlr	4.0
filter_reset	proc_sys_reset	5.0
ilmb	lmb_v10	3.0
ilmb_cntlr	lmb_bram_if_cntlr	4.0
iomodule_0	iomodule	3.0
lmb_bram_I	blk_mem_gen	8.2
microblaze_mcs_0	microblaze	9.5

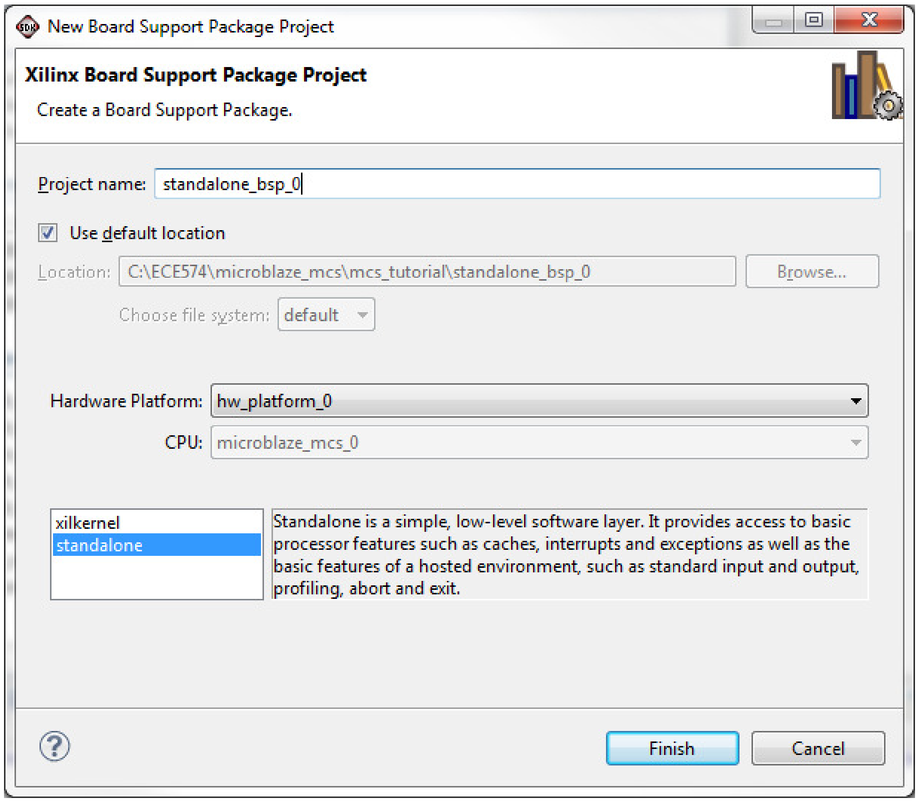
An outline is not available.

No consoles to display at this time.

```
19:49:01 INFO   : Launchi
19:49:24 INFO   : XSDB se
19:51:37 INFO   : Project
```

Now we have imported the hardware description, a standalone board support package can be created.

Select **File** -> **New** -> **Board Support Package**:



New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name: standalone_bsp_0

[✓] Use default location

Location: C:\ECE574\microblaze_mcs\mcs_tutorial\standalone_bsp_0

Choose file system: default

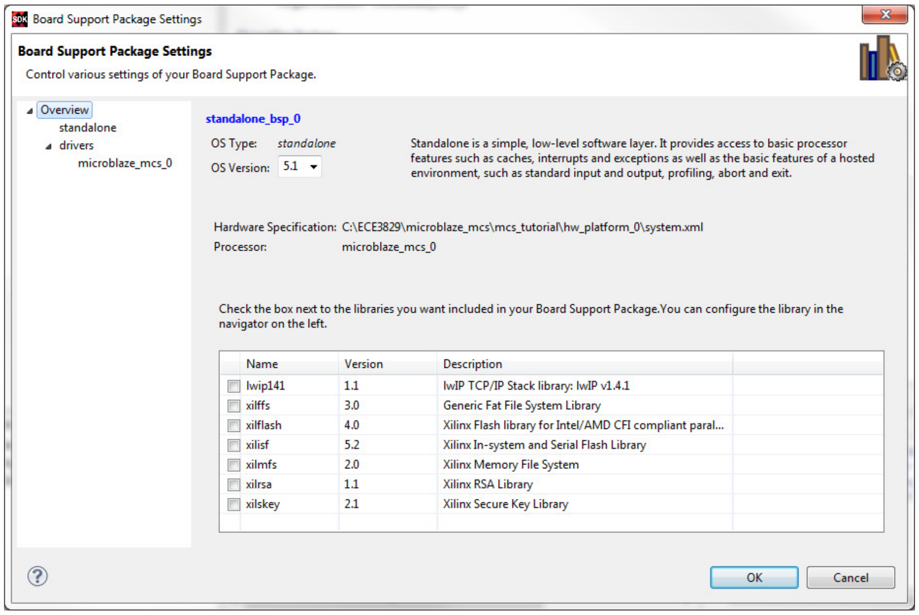
Hardware Platform: hw_platform_0

CPU: microblaze_mcs_0

xilkernel
[✓ standalone]

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Make sure 'standalone' for the Board Support Package OS is selected and then Click **Finish**



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

standalone_bsp_0

OS Type: *standalone*

OS Version: 5.1

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Hardware Specification: C:\ECE3829\microblaze_mcs\mcs_tutorial\hw_platform_0\system.xml

Processor: microblaze_mcs_0

Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

Name	Version	Description
☐ lwip141	1.1	lwIP TCP/IP Stack library: lwIP v1.4.1
☐ xilffs	3.0	Generic Fat File System Library
☐ xilflash	4.0	Xilinx Flash library for Intel/AMD CFI compliant paral...
☐ xilisf	5.2	Xilinx In-system and Serial Flash Library
☐ xilmfs	2.0	Xilinx Memory File System
☐ xilrsa	1.1	Xilinx RSA Library
☐ xilskey	2.1	Xilinx Secure Key Library

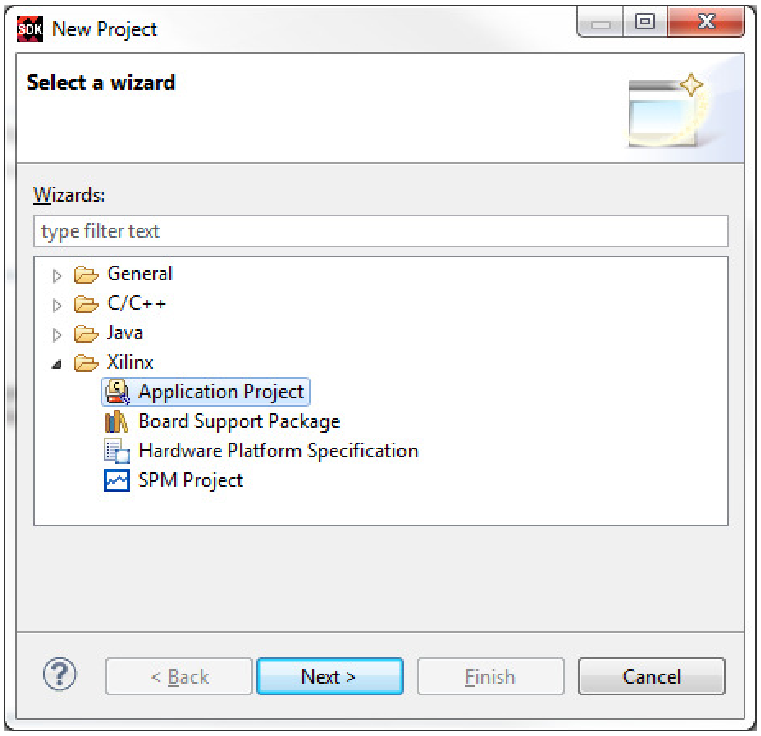
Click **OK**

You should eventually see in the SDK Console Window at the bottom of the window (select the Console Tab):

```
"Compiling iomodule"
"Running Make libs in microblaze_mcs_0/libsrc/standalone_v5_1/src"
make -C microblaze_mcs_0/libsrc/standalone_v5_1/src -s libs  "SHELL=CMD"
"COMPILER=mb-gcc" "ARCHIVER=mb-ar" "COMPILER_FLAGS= -O2 -c -mcpu=v9.5 -mlittle-endian
-mno-xl-reorder -mxl-soft-mul" "EXTRA_COMPILER_FLAGS=-g"
"Compiling standalone";
'Finished building libraries'
```

Now we will create a new C program:

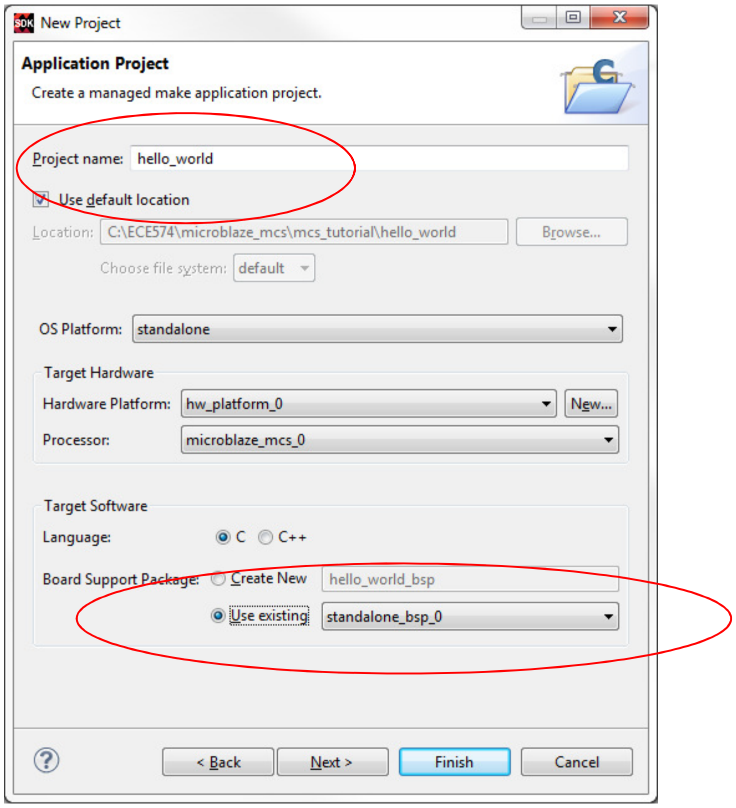
Select **File => New Project** and select *Application Project* under Xilinx



Click **Next**

Type 'hello_world' for the project name

Select the existing Board support Package:


New Project

Application Project

Create a managed make application project.

Project name: hello_world

Use default location

Location: C:\ECE574\microblaze_mcs\mcs_tutorial\hello_world

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: hw_platform_0

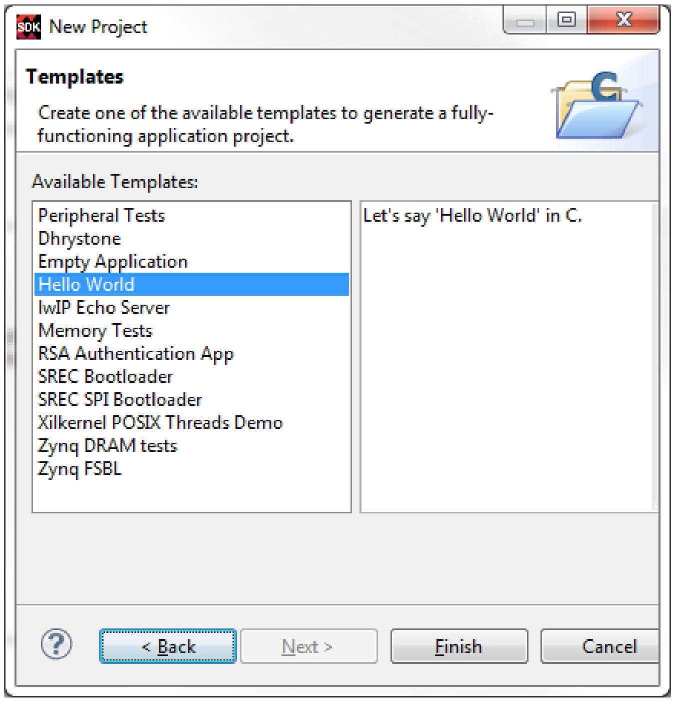
Processor: microblaze_mcs_0

Target Software

Language: [✓ C] C++

Board Support Package: Create New hello_world_bsp [✓ Use existing] standalone_bsp_0

Click **Next**


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Peripheral Tests
- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- RSA Authentication App
- SREC Bootloader
- SREC SPI Bootloader
- Xilkernel POSIX Threads Demo
- Zynq DRAM tests
- Zynq FSBL

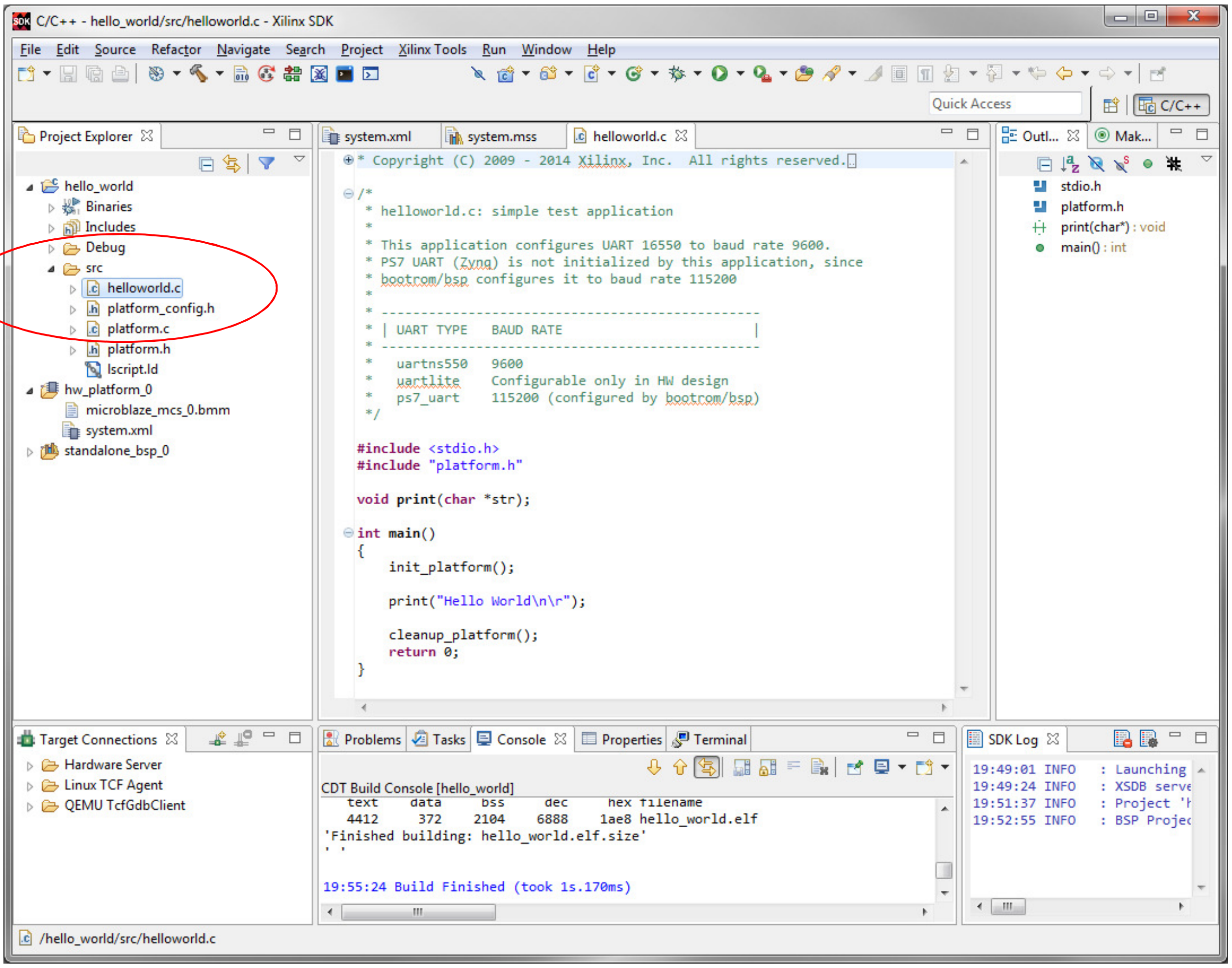
Let's say 'Hello World' in C.

Select **Hello World** and click Finish

Hello_world_0.elf is produced (ELF is Executable and Linkable Format):

```
mb-size hello_world.elf  |tee "hello_world.elf.size"
   text         data        bss         dec          hex       filename
   4412          372       2104        6888         1ae8       hello_world.elf
'Finished building: hello_world.elf.size'
' '
```

You can view the C program for Hello World by expanding the src folder under hello_world and selecting the helloworld.c file:



```
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    cleanup_platform();
    return 0;
}
```

CDT Build Console [hello_world]

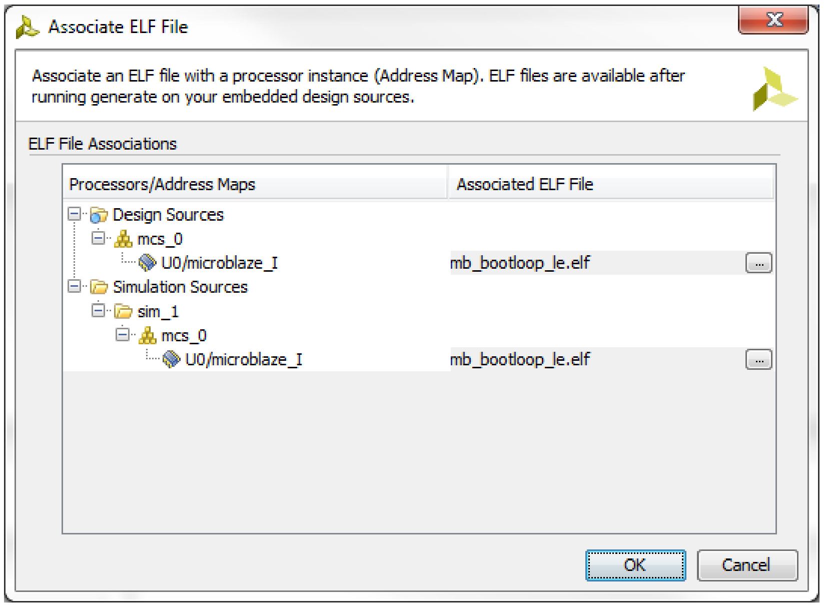
```
   text    data     bss     dec     hex filename
   4412     372    2104    6888    1ae8 hello_world.elf
'Finished building: hello_world.elf.size'
' '

19:55:24 Build Finished (took 1s.170ms)
```

```
19:49:01 INFO   : Launching
19:49:24 INFO   : XSDB serve
19:51:37 INFO   : Project 'h
19:52:55 INFO   : BSP Projec
```

We now need to associate the ELF file with our hardware.

Go back to Vivado.

Select **Tools -> Associate ELF file** … in the menu.



Associate ELF File

Associate an ELF file with a processor instance (Address Map). ELF files are available after running generate on your embedded design sources.

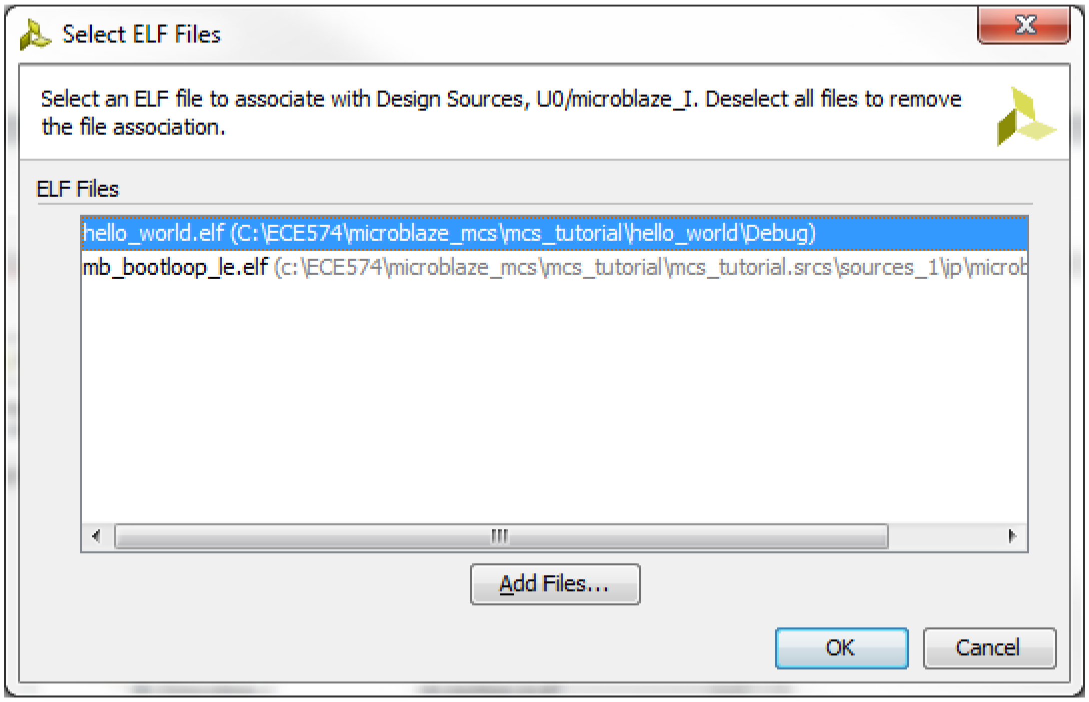
ELF File Associations

Processors/Address Maps	Associated ELF File
Design Sources	
mcs_0	
U0/microblaze_I	mb_bootloop_le.elf
Simulation Sources	
sim_1	
mcs_0	
U0/microblaze_I	mb_bootloop_le.elf

Initially the default infinite loop ELF file, mb_bootloop_le.elf is associated with the Microblaze MCS core.

Click on browse under Design Sources,

Select Add Files, and browse to the hello_world.elf file (in mcs_tutorial/hello_world/debug):



Select ELF Files

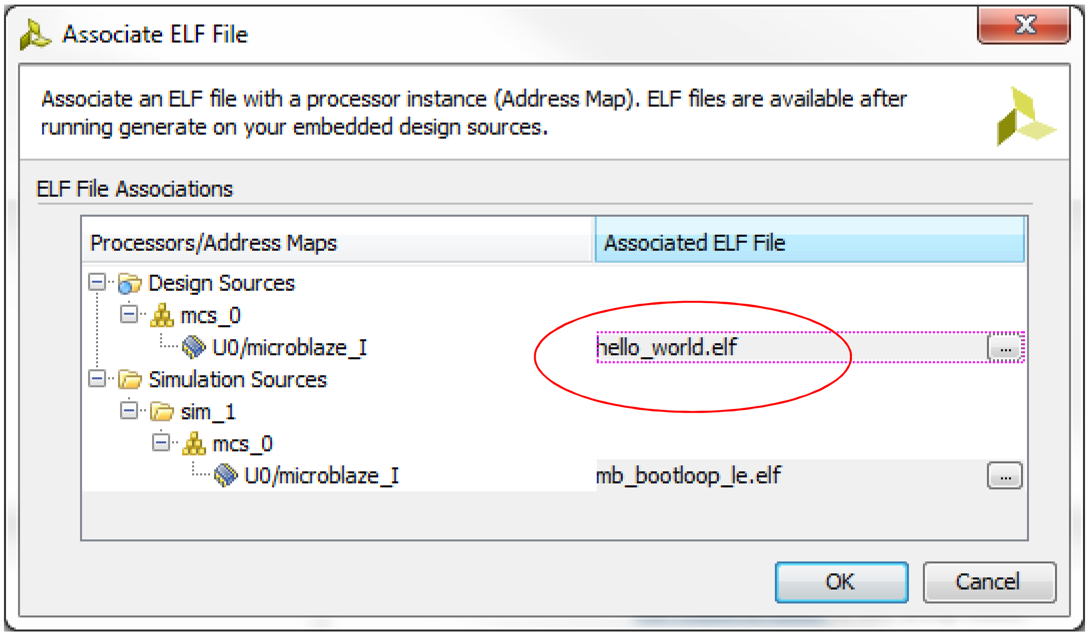
Select an ELF file to associate with Design Sources, U0/microblaze_I. Deselect all files to remove the file association.

ELF Files

hello_world.elf (C:\ECE574\microblaze_mcs\mcs_tutorial\hello_world\Debug)
mb_bootloop_le.elf (c:\ECE574\microblaze_mcs\mcs_tutorial\mcs_tutorial.srcs\sources_1\ip\microb

Select **OK**

We now have the hello_wrold.elf file containing our software progeam associated with the hardware design.



Associate an ELF file with a processor instance (Address Map). ELF files are available after running generate on your embedded design sources.

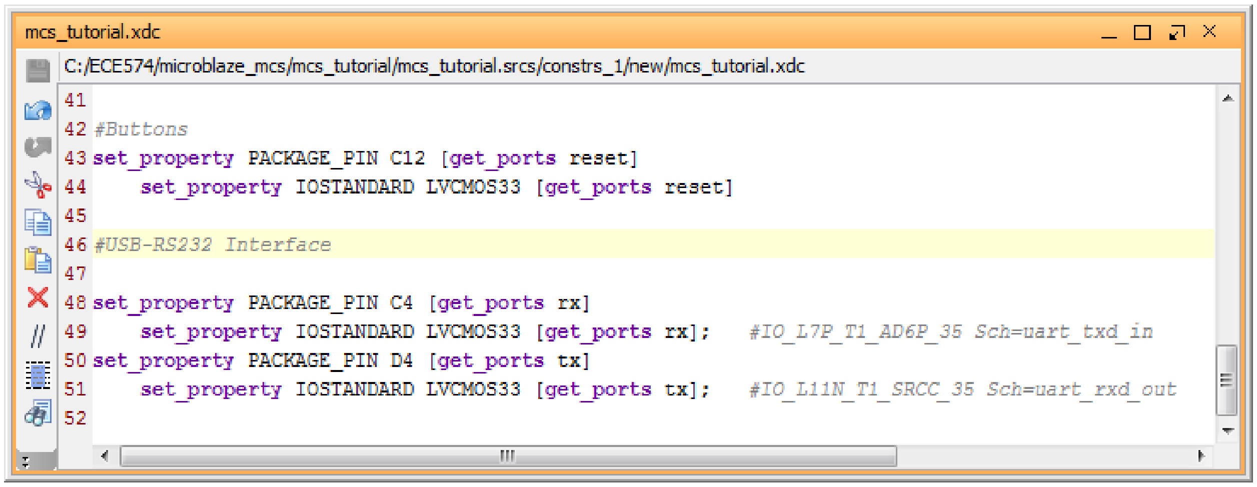
ELF File Associations

Processors/Address Maps	Associated ELF File
Design Sources	
mcs_0	
U0/microblaze_I	hello_world.elf
Simulation Sources	
sim_1	
mcs_0	
U0/microblaze_I	mb_bootloop_le.elf

Click **OK**

In Project Manager add a constraint source file to match your board for all the FPGA connections.

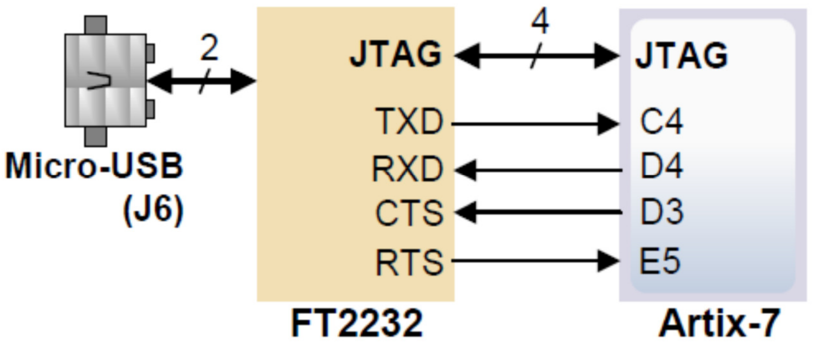
For example:


mcs_tutorial.xdc
C:/ECE574/microblaze_mcs/mcs_tutorial/mcs_tutorial.srcs/constrs_1/new/mcs_tutorial.xdc

```
#Buttons
set_property PACKAGE_PIN C12 [get_ports reset]
    set_property IOSTANDARD LVCMOS33 [get_ports reset]

#USB-RS232 Interface

set_property PACKAGE_PIN C4 [get_ports rx]
    set_property IOSTANDARD LVCMOS33 [get_ports rx];    #IO_L7P_T1_AD6P_35 Sch=uart_txd_in
set_property PACKAGE_PIN D4 [get_ports tx]
    set_property IOSTANDARD LVCMOS33 [get_ports tx];    #IO_L11N_T1_SRCC_35 Sch=uart_rxd_out
```

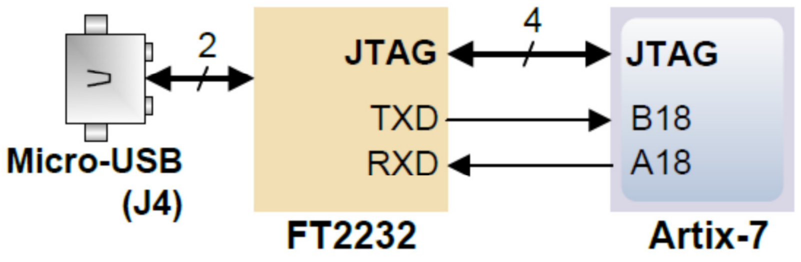
A comment regarding the UART connection:

In the Nexys4DDR board reference manual the UART TX and RX are shown as follows. This is showing the direction of transmission as seen by the UART.



This means that the FPGA transmits on D4 (port 'tx' in the XDC file) and receives on C4 (port 'rx').

For comparison, on the Basys3 board the FPGA transmits on A18 and receives on B18:



We can now Implement the Project and then create a bit file by running the **Generate Bitstream** step**.**

You may receive 8 warning messages after implementation.

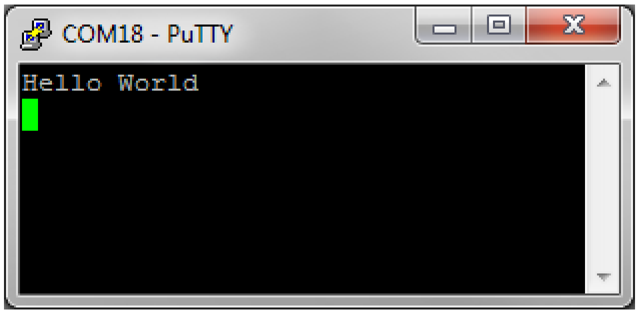
7 warnings are related to the Microblaze core that you can ignore.
1 warning is related to the clk_fpga (100MHz) being driven directly by an IO rather than a Clock Buffer. (In this simple example we did not use an MMCM for the clock but this would be recommended.)

Note: On the Nexys4DDR and Basys3 boards the USB-UART bridge (Serial Port) allows a PC application to communicate with the board using standard Windows COM port commands. The same USB is also used for the Digilent USB-JTAG circuitry but the functions behave independent of each other (read the Digilent user manual for more information).

Once a bitstream is created use the Hardware Manager to program the device.

Connect to the USB-UART using a serial communications link connected to the correct serial com port.

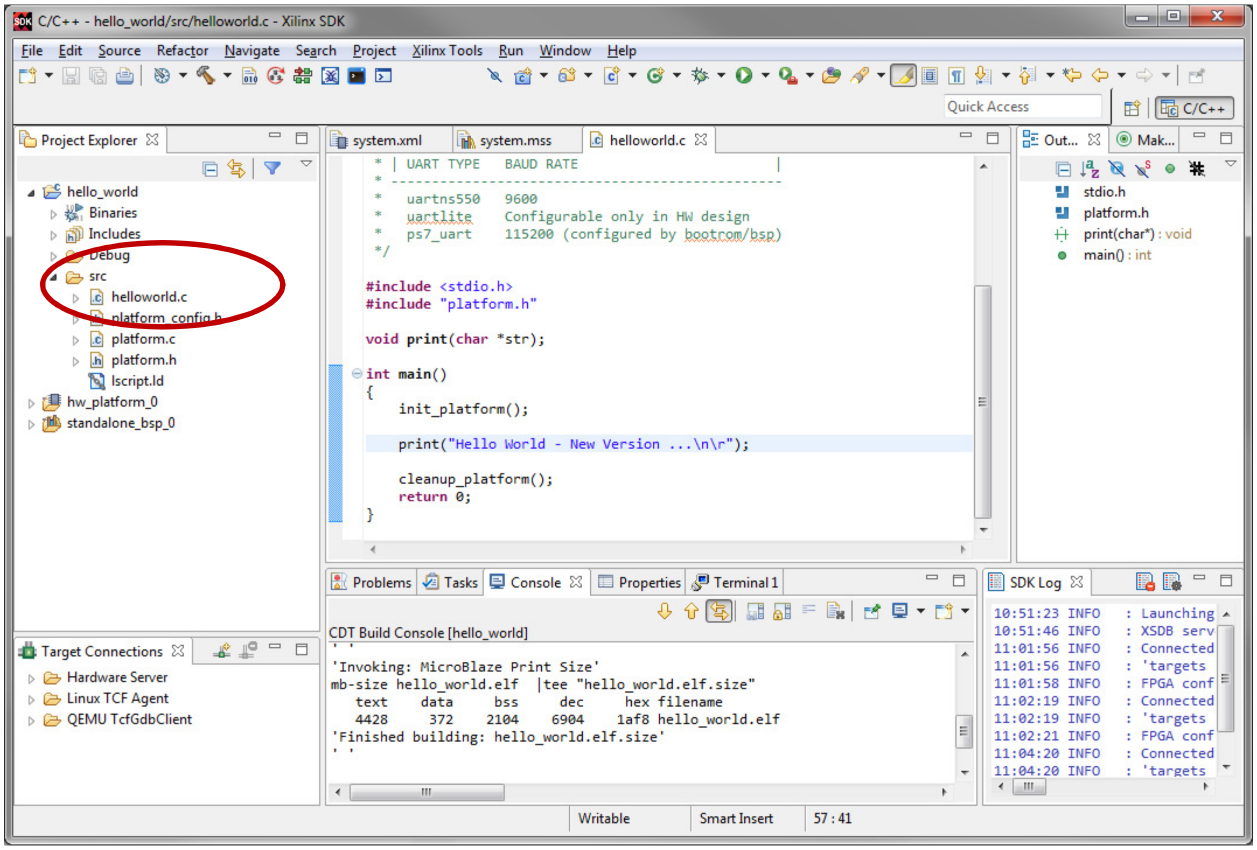
Press the reset button on the board and you should see "Hello World" appear on a serial communications link such as Putty or a Hyperterminal window:



```
Hello World
```

Extra: Modifying the C Program.

In the Xilinx XDK program, expand the src folder from the C project ,and double-click on the hello_world.c file. You can see the C statements:



```
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World - New Version ...\n\r");

    cleanup_platform();
    return 0;
}
```

CDT Build Console [hello_world]

```
' '
'Invoking: MicroBlaze Print Size'
mb-size hello_world.elf  |tee "hello_world.elf.size"
   text    data     bss     dec     hex filename
   4428     372    2104    6904    1af8 hello_world.elf
'Finished building: hello_world.elf.size'
' '
```

```
10:51:23 INFO   : Launching
10:51:46 INFO   : XSDB serv
11:01:56 INFO   : Connected
11:01:56 INFO   : 'targets
11:01:58 INFO   : FPGA conf
11:02:19 INFO   : Connected
11:02:19 INFO   : 'targets
11:02:21 INFO   : FPGA conf
11:04:20 INFO   : Connected
11:04:20 INFO   : 'targets
```

Modify the statements as required (for example change the “Hello World” to add your name) and then press **save**. A new ELF file is automatically generated.

Back in Vivado we will now see a message that says ‘write_bitsream Out-of-date’. This is due to hello_world.elf changing.

Rerun the **Generate Bitstream** process to create an updated bit file with the new C program added (you do not need to redo any of the previous synthesis or implementation steps unless you also change the hardware design).

Download the new bit file to the board and verify the new changes.

Extra: Accessing the GPIO.

To access GPI/GPO use XIOModule_DiscreteRead and XIOModule_DiscreteWrite with channel 1-4 for GPI1-4 and GPO1-4. For example:

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"      // add
#include "xiomodule.h"        // add

void print(char *str);

int main()
{
    init_platform();

    u32 data;
    XIOModule gpi;
    XIOModule gpo;

    print("Reading switches and writing to LED port\n\r");

    data = XIOModule_Initialize(&gpi, XPAR_IOMODULE_0_DEVICE_ID);
    data = XIOModule_Start(&gpi);

    data = XIOModule_Initialize(&gpo, XPAR_IOMODULE_0_DEVICE_ID);
    data = XIOModule_Start(&gpo);

    while (1)
    {
        data = XIOModule_DiscreteRead(&gpi, 1); // read switches (channel 1)
        XIOModule_DiscreteWrite(&gpo, 1, data); // turn on LEDs (channel 1)
    }

    cleanup_platform();

    return 0;
}
```

You can find the API documentation in the SDK Project Explorer, under <BSP Name>/BSP Documentation/iomodule_v1_00_a. Click on "Files", "xiomodule.h" for a list of functions.

Extra: Modifying the C Program to use xil_printf

The usual printf function is too large to fit into the small memory of the Microblaze but you can use the Xilinx light-weight version of printf called xil_printf.

Here is an example of its use in my C program:

```
counter = 1234;
xil_printf("The counter value is %d in decimal and %x in hex.", counter, counter);
```

And this is what is displayed in hyperterminal:

```
The counter value is 1234 in decimal and 4D2 in hex.
```

xil_printf is defined in 'stdio.h'.

Note: However I found out that in Xilinx version 14.1 the declaration was missing in this header file and you will see an 'implicit function declaration' warning. It did seem to link without errors and run OK. (This seems to be corrected in Version 14.2 and later so you can probably ignore this step)

But if you see the warning and want to fix it on your own system, right click on the stdio.h at the top of your C program (#include <stdio.h>) and select 'Open Declaration'
Add this to line 230

```
void   _EXFUN(xil_printf, (const char*, ...));
```

so the nearby lines look like:

```
int     _EXFUN(remove, (const char *));
int     _EXFUN(rename, (const char *, const char *));
void   _EXFUN(xil_printf, (const char*, ...));
#endif
```

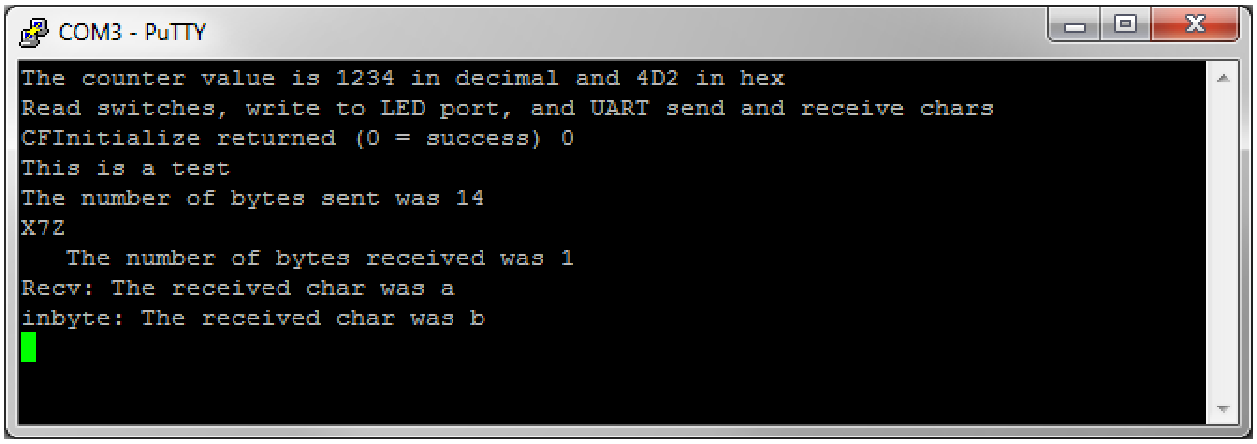
Assembler instructions:

If you want to see the assembler instructions that are created from your C program look in the hello_world => Debug => Src folder (top left pane in the Xilinx SDK application) and double-click on the hello_world_0.elf file.

If you scroll down this file until you find 'int main()' you will see your C instructions and the corresponding assembler and machine code values. Interesting stuff!

Extra: Accessing the GPIO, using xil_printf, and using the UART.

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"   // add
#include "xiomodule.h"     // add

void print(char *str);

int main()
{
    init_platform();

    u32 data;
    XIOModule iomodule;    // iomodule variable for gpi, gpo, and uart

    u8 msg[15] = "This is a test";// buffer for sending message using XIOModule_Send
    u8 rx_buf[10];                // receive buffer using XIOModule_Recv

    u32 counter;

    // example using xil_printf
    counter = 1234;
    xil_printf("The counter value is %d in decimal and %x in hex\n\r", counter,
counter);

    print("Read switches, write to LED port, and UART send and receive chars\n\r");

    // Initialize module to obtain base address
    data = XIOModule_Initialize(&iomodule, XPAR_IOMODULE_0_DEVICE_ID);
    data = XIOModule_Start(&iomodule);

    // Need to call CfgInitialize to use UART Send and Recv functions
    // int XIOModule_CfgInitialize(XIOModule *InstancePtr, XIOModule_Config *Config,
u32 EffectiveAddr);
    // note config and effective address arguments are not used
    data = XIOModule_CfgInitialize(&iomodule, NULL, 1);
    xil_printf("CFInitialize returned (0 = success) %d\n\r", data);

    // Send 12 characters using Send
    // Send is non-blocking so must be called in a loop, may return without sending a
character
    // unsigned int XIOModule_Send(XIOModule *InstancePtr, u8 *DataBufferPtr, unsigned
int NumBytes);
    const int count = 14;
    int index = 0;
    while (index < count) {
       data = XIOModule_Send(&iomodule, &msg[index],  count - index);
        index += data;
    }
    xil_printf("\n\rThe number of bytes sent was %d\n\r", index);

    // Another way to send individual characters
    outbyte('X');
    outbyte(0x37);  // number '7'
    outbyte('Z');
    outbyte('\n');  // line feed

    // Receive a character and store in rx_buf
    // unsigned int XIOModule_Recv(XIOModule *InstancePtr, u8 *DataBufferPtr, unsigned
int NumBytes);
    while
       ((data = XIOModule_Recv(&iomodule, rx_buf, 1)) == 0);
    xil_printf("The number of bytes received was %d\n\r", data);
    xil_printf("Recv: The received char was %c\n\r", rx_buf[0]);

    // Another way to receive a single character
    rx_buf[0] = inbyte();
    xil_printf("inbyte: The received char was %c\n\r", rx_buf[0]);

    while (1)
    {
        //data = XIOModule_DiscreteRead(&iomodule, 1);      // read switches (channel
1)
        data = XIOModule_DiscreteRead(&iomodule, 2); // read push (channel 2)
        XIOModule_DiscreteWrite(&iomodule, 1, data); // turn on LEDs (channel 1)
    }

    cleanup_platform();

    return 0;

}
```



```
The counter value is 1234 in decimal and 4D2 in hex
Read switches, write to LED port, and UART send and receive chars
CFInitialize returned (0 = success) 0
This is a test
The number of bytes sent was 14
X7Z
   The number of bytes received was 1
Recv: The received char was a
inbyte: The received char was b
```