



## **Using the Xilinx MicroBlaze Processor**

© 2012 ... Impulse Accelerated Technologies, Inc.

# Table of Contents

<b>Part I Quick Start Tutorials</b>	<b>3</b>
<b>1 Tutorial 1: Hello World on the MicroBlaze platform</b> .....	<b>5</b>
Loading the Hello World Application .....	5
Compiling the Application for Simulation .....	7
Hello World Source Code .....	9
Building the Application for the Target Platform .....	11
Exporting Files from CoDeveloper .....	13
Creating a Platform Using Xilinx System Builder .....	14
Importing the Generated Hardware to the Xilinx Tools .....	22
Importing the Application Software to the Xilinx Tools .....	26
Building and Downloading the Application .....	32
<b>2 Tutorial 2: Complex FIR Filter for Xilinx Design Suite 13.4 (or higher)</b> .....	<b>34</b>
Loading the Complex FIR Application .....	35
Understanding the Complex FIR Application .....	36
Compiling the Application for Simulation .....	37
Building the Application for the Target Platform .....	39
Exporting Files from CoDeveloper .....	41
Creating a Platform Using Xilinx Tools .....	42
Configuring the New Platform .....	50
Importing the Generated Hardware .....	55
Generating the FPGA Bitmap .....	57
Importing the Application Software .....	58
Running the Application .....	64

# 1 Quick Start Tutorials



## Overview

Impulse C can be used to generate hardware modules that are directly connected to an embedded processor (such as the Xilinx MicroBlaze) or to other hardware elements that may have been described using other design tools or techniques. As you have seen in earlier tutorials (found in the CoDeveloper primary Help information), the Impulse C programming model emphasizes the use of data streams, signals, and shared memories for process-to-process communication. These interfaces can be used to connect Impulse C processes to a wide variety of hardware devices and processors.

For embedded processors such as MicroBlaze, there are multiple possible ways to provide communication between a software application running on the processor, and a hardware accelerator running in the FPGA fabric. These include (among others):

- Using the PLB (Processor Local Bus) to create an Impulse C peripheral on a shared bus
- Using the FSL (Fast Simplex Link) to create a high-speed data stream
- Using shared memory

The following tutorials focus on data streaming applications, using both PLB and FSL. Shared memory examples are provided in your CoDeveloper product installation.

## Using the Xilinx MicroBlaze Processor

This following tutorials will lead you step-by-step through the compilation, execution and RTL generation of your first Impulse C applications on the Xilinx MicroBlaze platform.

The tutorials that follow assume that you have previously gone through at least one of the tutorials included in your standard CoDeveloper installation. It is also assumed that you are somewhat familiar with the Xilinx ISE and Platform Studio (EDK) tools.

*Note: These Tutorials uses Xilinx Design Suite version 13.4 or later.*

[Tutorial 1: Hello World on the MicroBlaze platform](#)

## Using the "Generic" Xilinx FPGA Platforms

If you are not using an embedded MicroBlaze processor or the Xilinx Platform Studio (EDK) tools, then you will use CoDeveloper to generate logic (in the form of HDL output files) that is connected to other hardware modules or to external device pins, typically through the use of named ports. (See Tutorial 4 in the CoDeveloper primary Help file for more information about **co\_port** types.)

The following section describes how HDL files generated by CoDeveloper can be easily imported into a Xilinx ISE project for synthesis. (These steps are similar for other synthesis and simulation tools, including the Synplicity Synplify Pro tools, Mentor's Modelsim, and others.)

[Exporting Files to Xilinx ISE](#)

**See Also**

## 1.1 Tutorial 1: Hello World on the MicroBlaze platform



### Overview

This tutorial will demonstrate how to generate hardware and related software interfaces in the form of RTL (Register Transfer Logic) HDL descriptions appropriate for use with the Xilinx Platform Studio™ (Embedded Development Kit) software. This tutorial assumes Platform Studio version 13.4; other versions of the Xilinx software may require changes to some of the steps presented.

The sample project will implement a trivial "Hello World" application. This sample project includes a software process (running on the MicroBlaze processor) and a hardware process (running in the FPGA) that communicate via a single stream over the PLB bus.

*Note: The FSL bus may also be used for increased I/O performance, but if FSL is used, some steps will differ from those described in this tutorial. [Tutorial 2](#) describes the use of FSL.*

The purpose of this tutorial is to take you through the entire process of generating hardware and software interfaces and importing the relevant files to the Xilinx environment. The tutorial will also describe how to create the platform and downloadable FPGA bitmap using the Xilinx tools.

This tutorial will require approximately 90 minutes to complete (including software run times).

### Steps

- [Loading the Hello World Application](#)
- [Compiling the Application for Simulation](#)
- [Building the Application for the Target Platform](#)
- [Hello World Source Code](#)
- [Exporting Files from CoDeveloper](#)
- [Importing the Generated Hardware to the Xilinx Tools](#)
- [Importing the Application Software to the Xilinx Tools](#)

### Note

This tutorial assumes you have purchased or are evaluating the CoDeveloper Platform Support Package for Xilinx Microblaze, and that you have installed and have valid licenses for the Xilinx ISE and Platform Studio (EDK) products.

### See Also

- [Hello World Source Code](#)

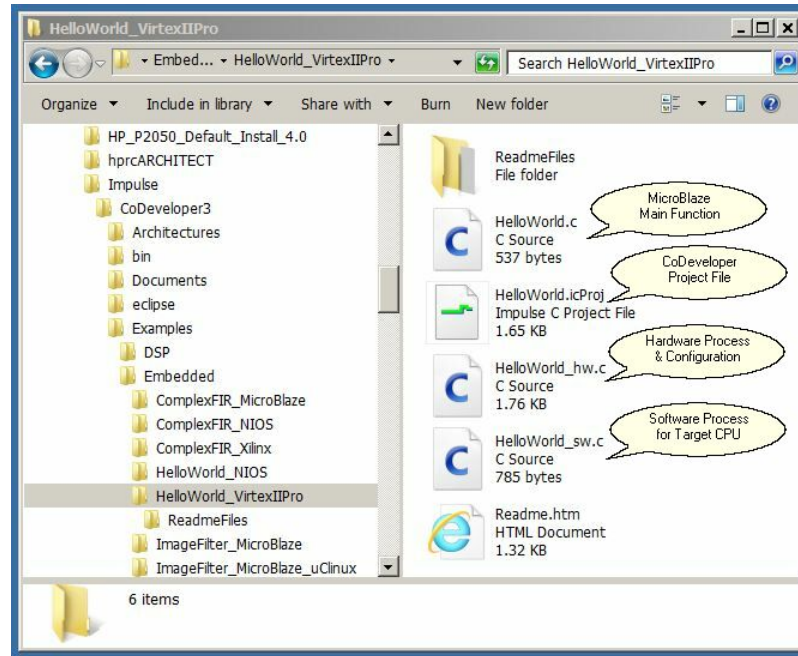
#### 1.1.1 Loading the Hello World Application

##### Hello World Tutorial for MicroBlaze, Step 1

To begin, start the CoDeveloper Application Manager by selecting Application Manager from the Start -> Programs -> Impulse Accelerated Technologies -> CoDeveloper program group.

*Note: this tutorial assumes that you have already read and understand the basic "Hello World" tutorial presented in the CoDeveloper User's Guide.*

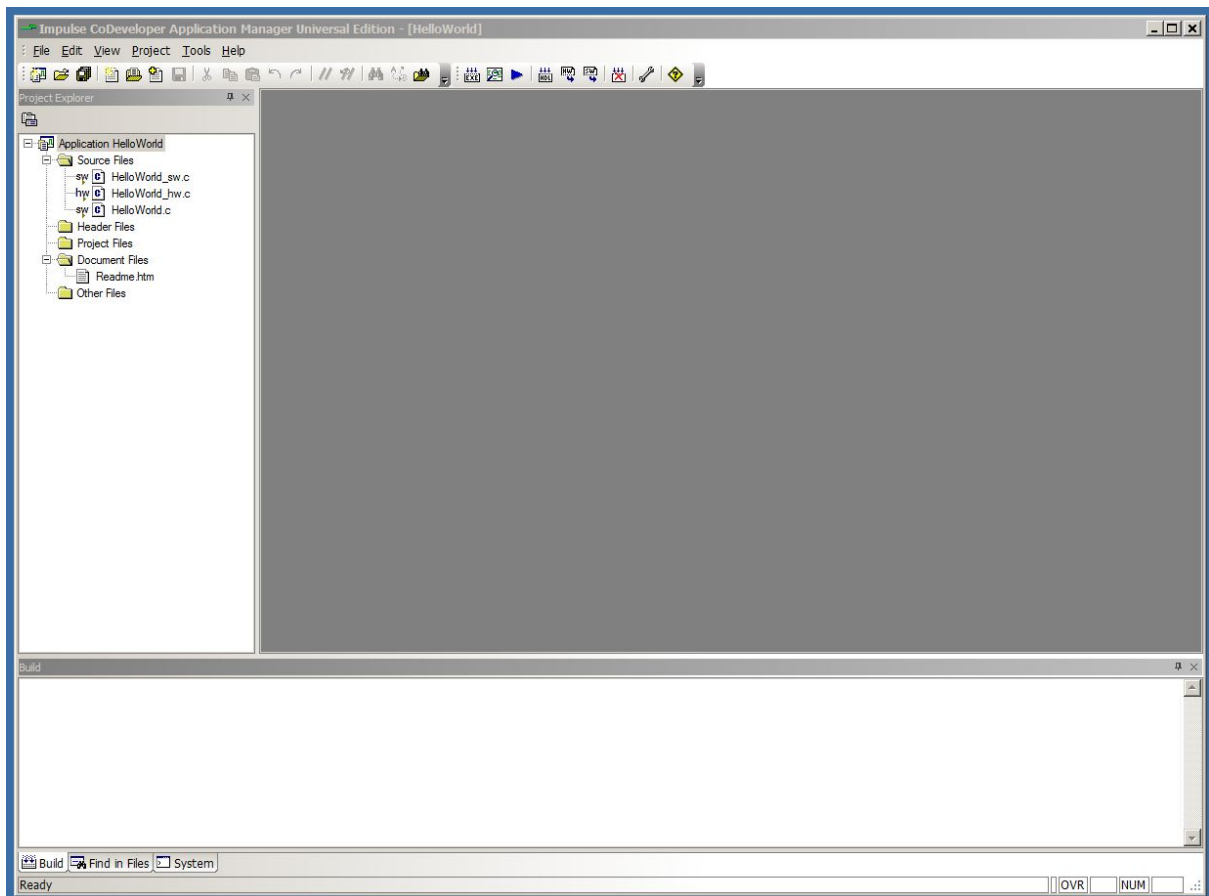
The first step is to copy the Impulse HelloWorld example to a working directory. Locate the HelloWorld example in the Impulse tree, typically on your C:\ disk drive:



Copy these files to a working directory so that if a mistake is made the original files are not altered, in this case a working directory is made in the C:\sandbox\HelloWord folder. Be certain to copy all of the files.

Launch the Impulse CoDeveloper software by double-clicking the HelloWorld.icProj file or from the Start menu with:

Start -> Impulse Accelerated Technologies -> CoDeveloper X.xx -> CoDeveloper Application Manager.  
You should have a display very similar to this one:



The HelloWorld project is now loaded and it is time to compile it for simulation.

## See Also

[Compiling the Application for Simulation](#)

### 1.1.2 Compiling the Application for Simulation

#### Hello World Tutorial for MicroBlaze, Step 2

Before compiling the HelloWorld application to the target MicroBlaze platform, let's first take a moment to understand its basic operation and the contents of its primary source files.

The specific process that we will be compiling to hardware is represented in HelloWorld\_hw.c by the following function:

```
void say_hello(co_stream hello_out, co_parameter arg)
```

This hardware process simply outputs data to an output stream (**hello\_out**), based on the value of the compile-time parameter **arg**.

This hardware process is accompanied in the HelloWorld\_hw.c file by a configuration function

(**config\_hello**) that specifies the connection between the **hello\_out** output stream and the corresponding input stream for the software process **hear\_hello**.

The process **hear\_hello** (which is declared as an **extern** function at the start of HelloWorld\_hw.c) is defined in the source file HelloWorld\_sw.c. This process will, when compiled, be loaded onto the MicroBlaze processor and will communicate with the FPGA hardware via an automatically-generated hardware/software interface. In this case the software process simply reads data from the stream associated with the hardware process. (In a more typical Impulse C application there are multiple input/output streams associated with a given hardware process.)

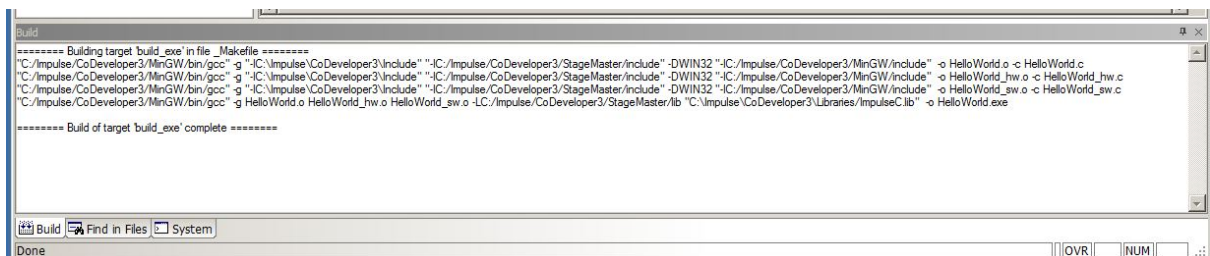
The remaining source file, HelloWorld.c, includes a main function for the application. This main function makes reference to the lower-level configuration file through the Impulse C standard function **co\_initialize()**, which has been declared as an **extern** function and may be found in HelloWorld\_hw.c.

*Note: the organization of source files shown in this example is not required, but is recommended in order to simplify the compilation process. It is a good idea to keep the **main()** function of your application in one file (as shown), and to place hardware and software processes in different files. Doing so will avoid potential problems with cross-compiler incompatibilities between hardware and software processes as well as simplifying the compilation and linking of complete applications for desktop simulation.*

## Simulating the Hello World Application

To compile and simulate the application for the purpose of functional verification:

1. Select Project -> Build Software Simulation Executable (or click the Build Software Simulation Executable button) to build the HelloWorld.exe executable. A command window will open, displaying the compile and link messages as shown below:

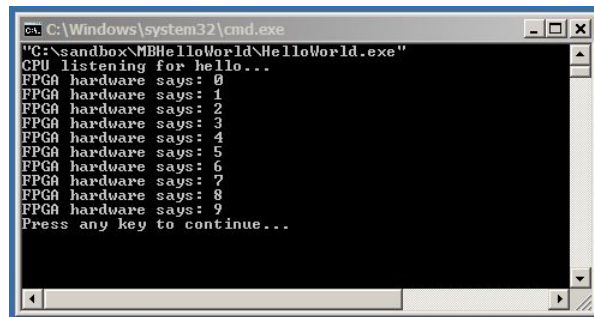


```
Build
***** Building target 'build_exe' in file '_Makefile' *****
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g -IC:/Impulse/CoDeveloper3/Include" -IC:/Impulse/CoDeveloper3/StageMaster/Include" -DWIN32 -IC:/Impulse/CoDeveloper3/MinGW/Include" -o HelloWorld.o -c HelloWorld.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g -IC:/Impulse/CoDeveloper3/Include" -IC:/Impulse/CoDeveloper3/StageMaster/Include" -DWIN32 -IC:/Impulse/CoDeveloper3/MinGW/Include" -o HelloWorld_hw.o -c HelloWorld_hw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g -IC:/Impulse/CoDeveloper3/Include" -IC:/Impulse/CoDeveloper3/StageMaster/Include" -DWIN32 -IC:/Impulse/CoDeveloper3/MinGW/Include" -o HelloWorld_sw.o -c HelloWorld_sw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g HelloWorld.o HelloWorld_hw.o HelloWorld_sw.o -L.C:/Impulse/CoDeveloper3/StageMaster/lib -L"C:/Impulse/CoDeveloper3/Libraries/ImpulseC.lib" -o HelloWorld.exe

***** Build of target 'build_exe' complete *****

Build Find in Files System
Done
```

2. You now have a Windows executable representing the HelloWorld application implemented as a desktop (console) software application. Run this executable by selecting Project -> Launch Simulation Executable. A command window will open and the simulation executable will run as shown below:



```
C:\Windows\system32\cmd.exe
"C:\sandbox\MBHelloWorld\HelloWorld.exe"
CPU listening for hello...
FPGA hardware says: 0
FPGA hardware says: 1
FPGA hardware says: 2
FPGA hardware says: 3
FPGA hardware says: 4
FPGA hardware says: 5
FPGA hardware says: 6
FPGA hardware says: 7
FPGA hardware says: 8
FPGA hardware says: 9
Press any key to continue...
```



Verify that the simulation produces the following output:

```
CPU listening for hello...
FPGA hardware says: 0
FPGA hardware says: 1
FPGA hardware says: 2
FPGA hardware says: 3
FPGA hardware says: 4
FPGA hardware says: 5
FPGA hardware says: 6
FPGA hardware says: 7
FPGA hardware says: 8
FPGA hardware says: 9
```

Note that, although the application display messages indicating that the FPGA hardware has generated the indicated values, in fact the values are being generated by a hardware process which, for simulation purposes, has been compiled as a software process on your host development system (Windows). The next steps will show how to take this same Impulse C application and compile to actual hardware.

## See Also

[Building the Application for the Target Platform](#)  
[HelloWorld Source Code](#)

### 1.1.3 Hello World Source Code

#### HelloWorld\_sw.c

```
////////////////////////////////////
// Copyright(c) 2003-2010 Impulse Accelerated Technologies, Inc.
// All Rights Reserved.
//
// HelloWorld_sw.c: Process to be executed on the target CPU.
//

#include "co.h"
#include "cosim_log.h"
#include <stdio.h>

#ifdef IMPULSE_C_TARGET
#define printf xil_printf
#endif

void hear_hello(co_stream hello_in) // NOTE: The only interface objects you
can use for arguments // are ImpulseC objects, such as
{ // only functions can communicate with
co_streams. However, software
int i;
global objects.
int hi = 0;

IF_SIM(cosim_logwindow log = cosim_logwindow_create("hear_hello");)

co_stream_open(hello_in, O_RDONLY, INT_TYPE(32));
```

```

printf("CPU listening for hello...\n\r");
for ( i = 0; i < 10; i++ ) {
    co_stream_read(hello_in, &hi, sizeof(int));
    printf("FPGA hardware says: %d\n\r", hi);
}

co_stream_close(hello_in);
}

```

## HelloWorld\_hw.c

```

////////////////////////////////////
// Copyright(c) 2003-2010 Impulse Accelerated Technologies, Inc.
// All Rights Reserved.
//
// HelloWorld_hw.c: Hardware processes and configuration code.
//

#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include "co.h"

#define MONITOR

#ifdef MONITOR
#include "cosim_log.h"
#endif

// Software process
extern void hear_hello(co_stream hello_in); // Software processes that communicate with the
hardware must be // forward declared as extern.

void say_hello(co_stream hello_out, co_parameter arg)
{
#ifdef MONITOR
    IF_SIM(cosim_logwindow log_say_hello;)
#endif
    co_int32 count = 0;
    co_int32 iterations = (co_int32) arg;

#ifdef MONITOR
    IF_SIM(log_say_hello = cosim_logwindow_create("say_hello");)
#endif

    co_stream_open(hello_out, O_WRONLY, INT_TYPE(32));

    for ( count = 0; count < iterations; count++ ) {
        co_stream_write(hello_out, &count, sizeof(co_int32));
    }

    co_stream_close(hello_out);
}

```

```
}

void config_hello(void * arg)
{
    co_process say_hello_proc, hear_hello_proc;
    co_stream hello_stream = co_stream_create("hello_stream", INT_TYPE(32), 8);

#ifdef MONITOR
    IF_SIM(cosim_logwindow_init());
#endif

    say_hello_proc = co_process_create("say_hello", (co_function)say_hello,
                                     2, // Number of arguments declared in the function interface.
                                     hello_stream, // Each argument must be listed individually and in the correct
order.
                                     10); // This is how to pass a COMPILE-TIME value to a function.

    hear_hello_proc = co_process_create("hear_hello", (co_function)hear_hello,
                                       1,
                                       hello_stream);

    co_process_config(say_hello_proc, co_loc, "PE0"); // The "PE0" must be defined in the Platform
Support Package.
}

co_architecture co_initialize(int arg)
{
    // The "xilinx_mb_opb" must be defined in the Platform Support Package.
    return co_architecture_create("hello", "xilinx_mb_opb", config_hello, (void *) arg);
}
```

### See Also

[Building the Application for the Target Platform](#)

## 1.1.4 Building the Application for the Target Platform

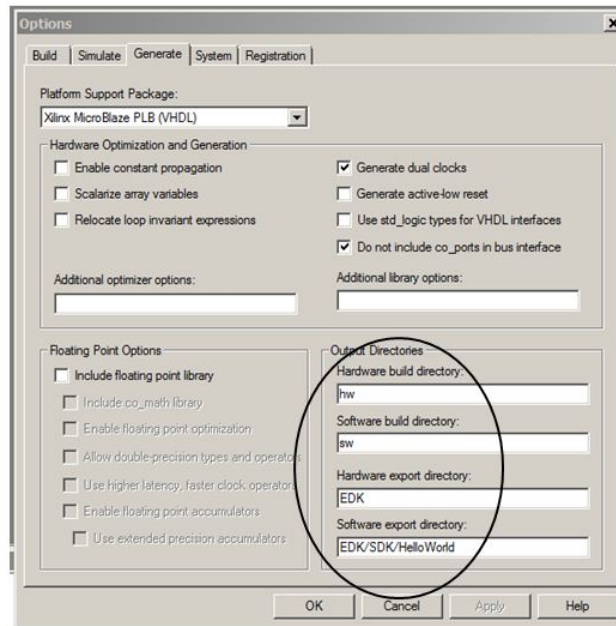
### Hello World Tutorial for MicroBlaze, Step 3

#### Specifying the Platform Support Package

The next step, prior to compiling and generating the HDL and related output files, is to select a platform target. Which target you select has a number of implications, including:

- The output file format (e.g. VHDL, Verilog or other intermediate language)
- The primitive components that will be referenced in the output (e.g. memory cores or buffer types)
- The types of optimizations performed during compilation
- The software library components that will be generated

To specify a platform target, open the Generate Options dialog as shown below using Project -> Options -> Generate.

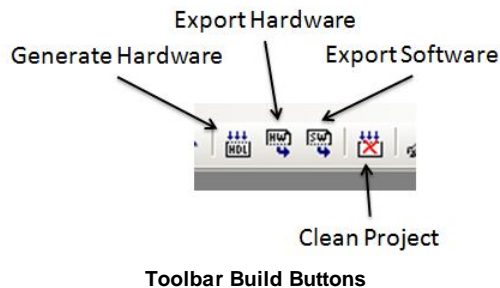


Specify *Xilinx MicroBlaze PLB (VHDL)* as shown. Also specify "hw" and "sw" for the hardware and software directories as shown, and specify "EDK" for the hardware export and EDK/SDK/HelloWorld for the software export directories. Click OK to save the options and exit the dialog.

*Note: The Xilinx MicroBlaze Platform Support Package includes three options for the on-chip bus: FSL, PLB and OPB. Which bus you choose is a system-level decision; in general the FSL bus will provide higher hardware/software data throughput but will be less able to handle external peripherals. Consult your Xilinx MicroBlaze documentation for additional details. Note that OPB is not used in the latest Xilinx products.*

### Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces and library files, select Generate HDL from the Project menu, or click on the Generate HDL button. A series of processing steps will run in a command window at the bottom of the main window.



When processing is complete you will have a number of resulting files created in the hw and sw sub-directories of your project directory. Take a moment to review these generated files. They include:

#### Hardware directory ("hw")

- Generated VHDL source files (HelloWorld\_comp.vhd and HelloWorld\_top.vhd) representing the hardware process and the generated hardware stream interface.

- A **lib** subdirectory containing required VHDL library elements.
- A **pcore** subdirectory containing generated files required by the Xilinx EDK tools.

#### Software directory ("sw")

- C source files extracted from the project that are required for compilation to the embedded processor (in this case HelloWorld\_sw.c and HelloWorld.c).
- A generated C file (co\_init.c) representing the hardware initialization function. This file will also be compiled to the embedded processor.
- A **driver** subdirectory containing additional software libraries to be compiled as part of the embedded software application. These libraries implement the software side of the hardware/software interface.

If you are an experienced Xilinx EDK user you may copy these files manually to your EDK project area and, if needed, modify them to suit your needs. In the next step, however, we will show how to use the hardware and software export features of CoDeveloper to move these files into your Xilinx EDK project automatically.

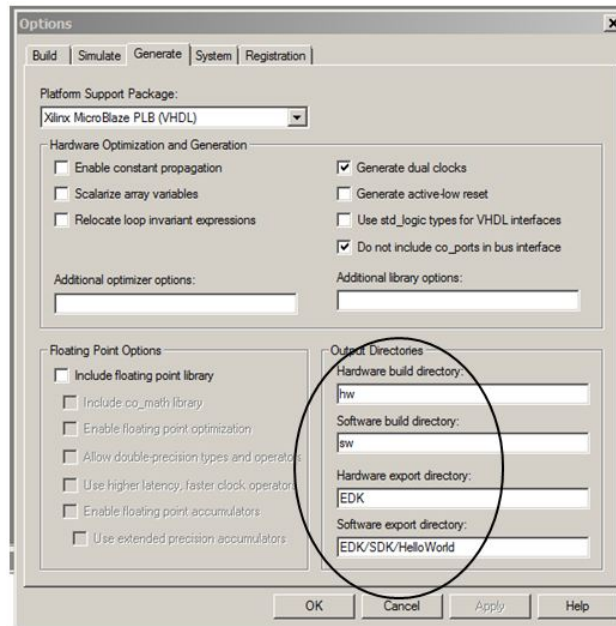
#### See Also

[Exporting Files from CoDeveloper](#)

### 1.1.5 Exporting Files from CoDeveloper

#### Hello World Tutorial for MicroBlaze, Step 4

From the CoDeveloper Application Manager. Recall that in Step 3 you specified the directory "EDK" as the export target for hardware and EDK/SDK/HelloWorld for software:



These export directories specify where the generated hardware and software processes are to be copied when the Export Software and Export Hardware features of CoDeveloper are invoked. Within these target directories (in this case "EDK"), the specific destination (which may be a subdirectory under "EDK") for each file previously generated is determined from the Platform Support Package architecture library files. It is therefore important that the correct Platform Support Package (in this case Xilinx MicroBlaze OPB) is selected prior to starting the export process.

To export the files from the build directories (in this case "hw" and "sw") to the export directories (in this case the "EDK" directory), select Project->Export Generated Hardware (HDL) and Project->Export Generated Software, or select the Export Generated Hardware and Export Generated Software buttons from the toolbar.



*Note: you must select BOTH Export Software and Export Hardware before going onto the next step.*

You have now exported all necessary files from CoDeveloper to the Xilinx tools environment.

**See Also**

[Creating a Platform Using Xilinx System Builder](#)

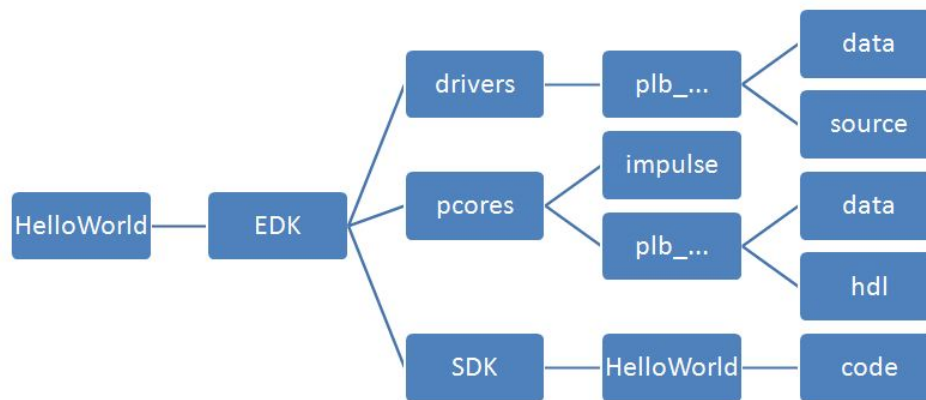
**1.1.6 Creating a Platform Using Xilinx System Builder  
Hello World Tutorial for MicroBlaze, Step 5**

As you saw in the previous step, CoDeveloper creates a number of hardware and software-related

output files that must all be used to create a complete hardware/software application on the target platform (in this case a Xilinx FPGA with an embedded MicroBlaze processor). You can, if you wish, copy these files manually and integrate them into your existing Xilinx projects. Usually, you will use the export features of CoDeveloper (prior step) to integrate the files into the Xilinx tools semi-automatically. This section will walk you through the process, using a new EDK System Builder (Platform Studio) project as an example.

*Note: you must have the Xilinx ISE and EDK software (version 13.4 or later) installed in order to proceed with this and subsequent steps. This tutorial demonstrates the steps need for EDK version 13.4. Other versions may require minor changes.*

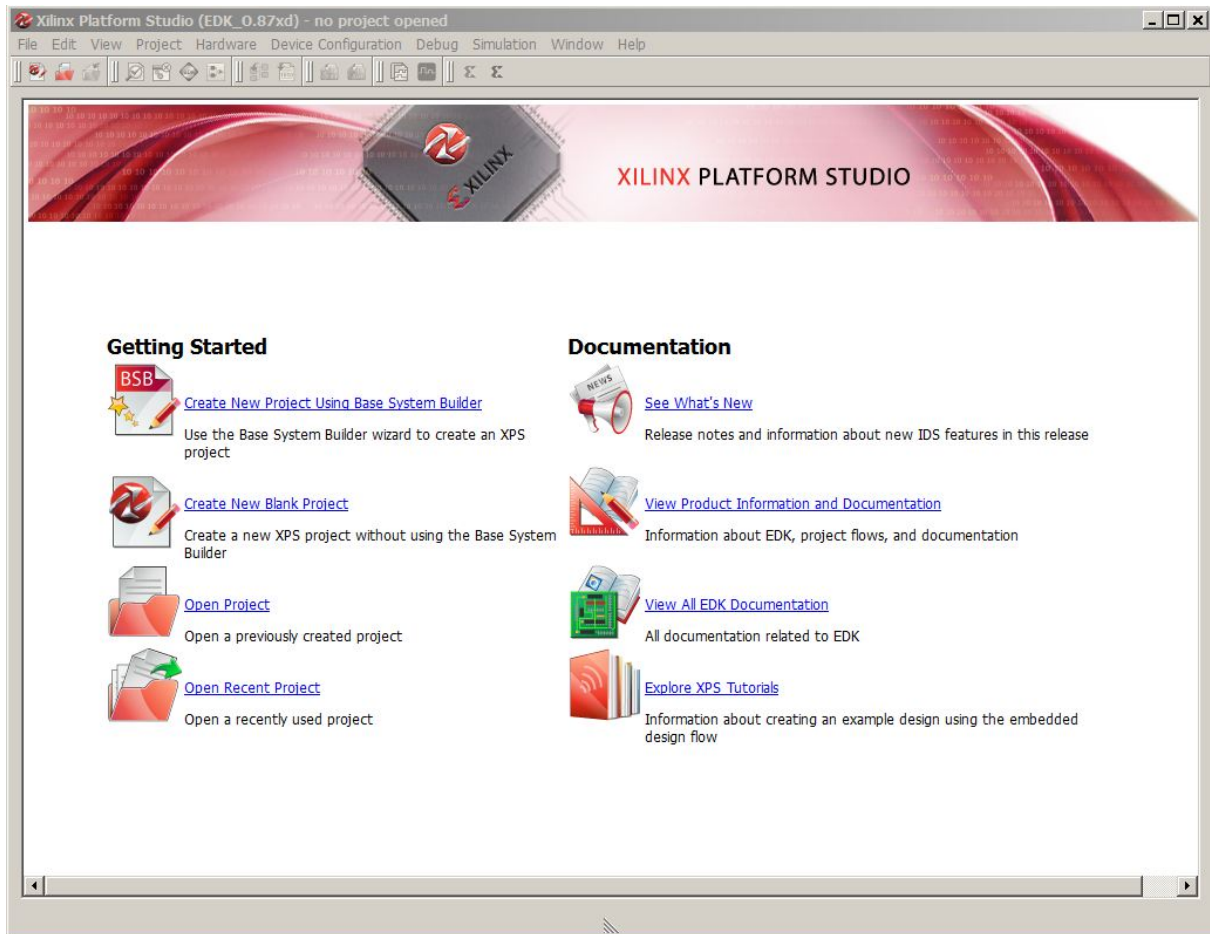
When exporting the project the directories shown below were created in the Impulse project directory. The organization of this directory tree is what the Xilinx tools expect to find and the Impulse generated files are already in the correct locations.



Directory structure for EDK/SDK projects

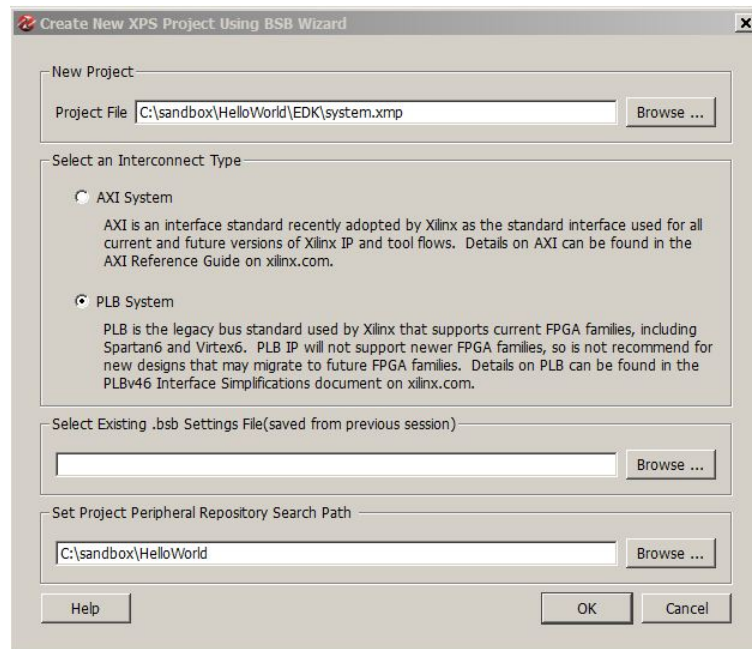
## Creating a New Xilinx Platform Studio Project

Now we'll move into the Xilinx tool environment. Begin by launching Xilinx Platform Studio (from the Windows Start->Xilinx ISE DesignSuite 13.4-EDK->Xilinx Platform Studio and creating a new project by selecting the Create New Project Using Base System Builder option.



After clicking the BSB option the Create New XPS Project Using BSB Wizard will appear.





Browse to the directory where you want the project file stored, this will be the EDK directory you chose above. Then choose if you will be using the newer AXI interface standard or the legacy PLB standard supported up to and including Xilinx Spartan and Virtex 6 families. In this example we will be using the legacy PLB flow. If you have an earlier .bsb (settings file) from a previous design to reuse then select it using the "SelectExisting" control. For the "Set Project Peripheral ..." control, navigate to your project directory that CONTAINS the EDK directory created above.

NOTE: The Peripheral Repository Search Path must be the directory that is TWO levels above the pcores directory. In our example above, the full path to pcores is:

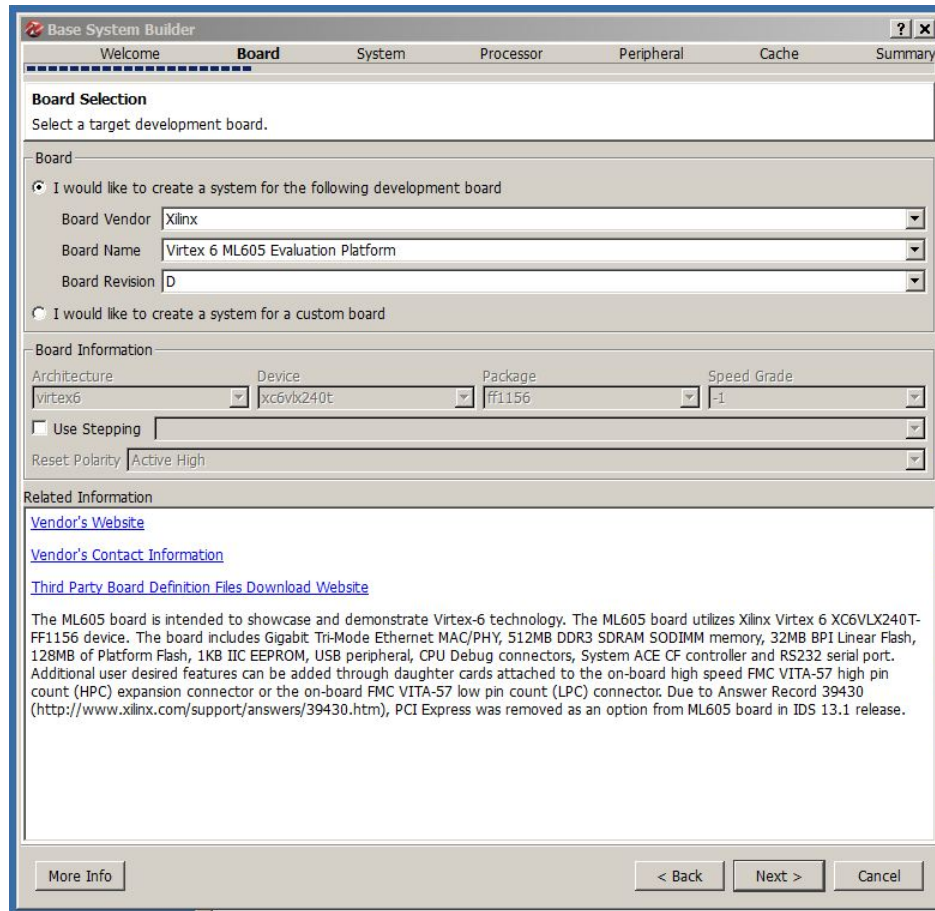
```
C:\sandbox\HelloWorld\EDK\pcores
```

Therefore, we selected C:\sandbox\HelloWorld for the repository search path.

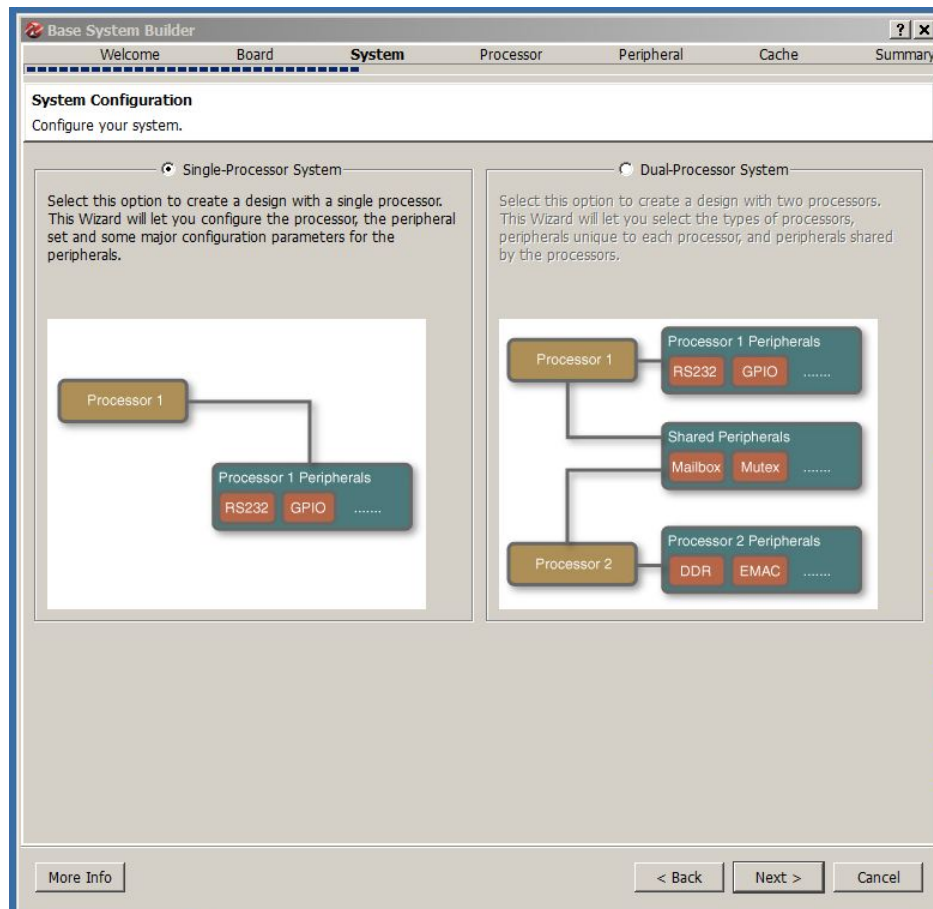
Then click "OK" and in the next wizard page select "I would like to create a new design" and click "Next".

Choose your development board from the dropdown boxes. This example will use the following board (you should choose the reference board you have available for this step):

```
Board Vendor: Xilinx  
Board Name: Virtex 6 ML605 Evaluation Platform  
Board Revision: D
```

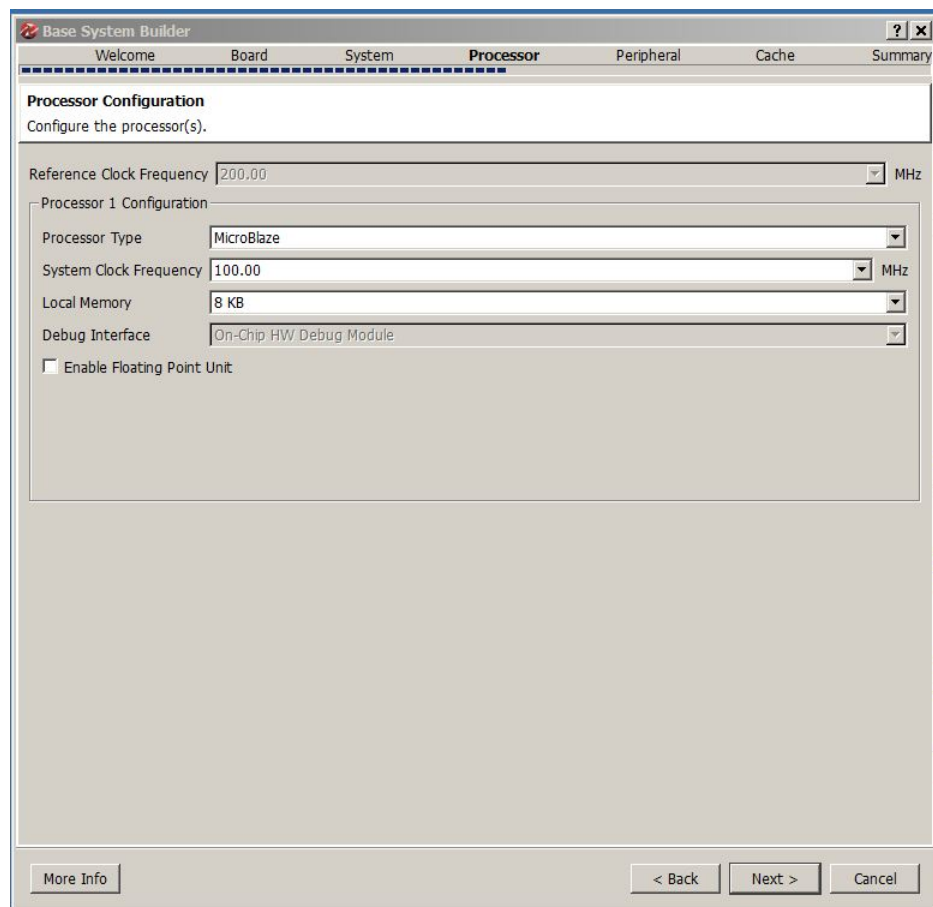


Click Next to continue with the System Builder Wizard. In the next wizard page, make sure that you select a single processor system.



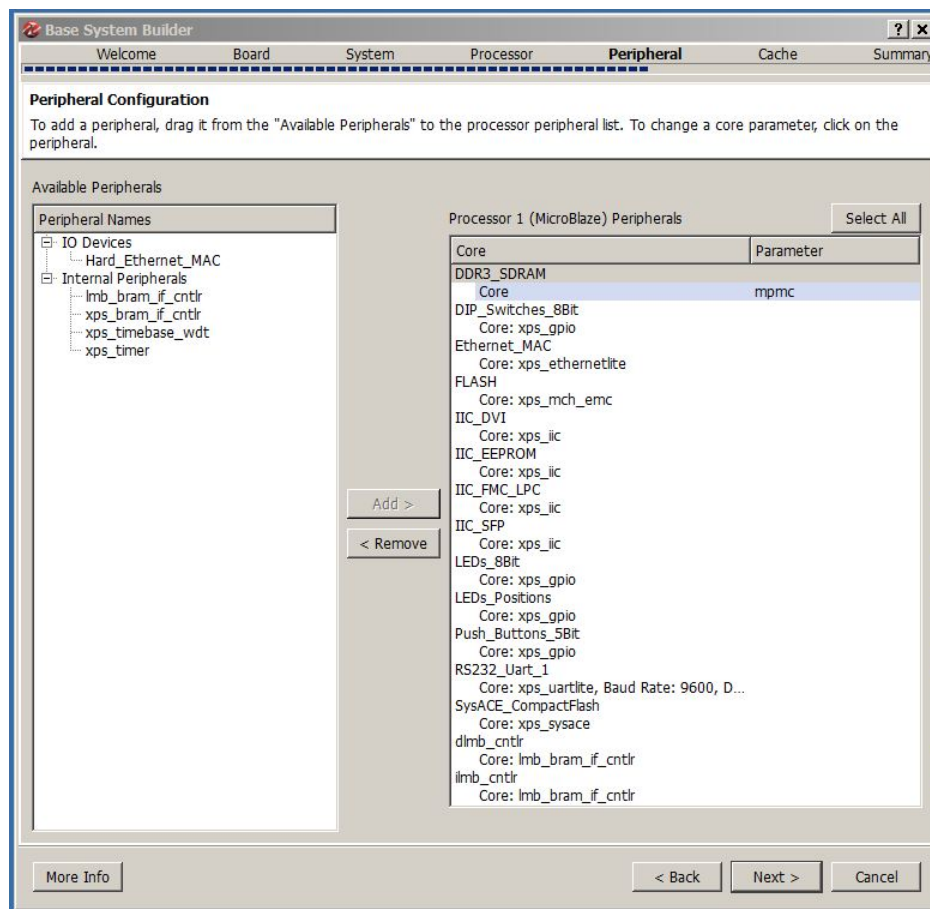
Click Next to continue with the System Builder Wizard.

With this Wizard page you select the Processor Type, Clock Frequency and Local Memory allocation. If you will be doing floating point operations in software then you may also enable the Floating Point Unit.



*Note: the Base System Builder options that follow may be different depending on the development board you are using.*

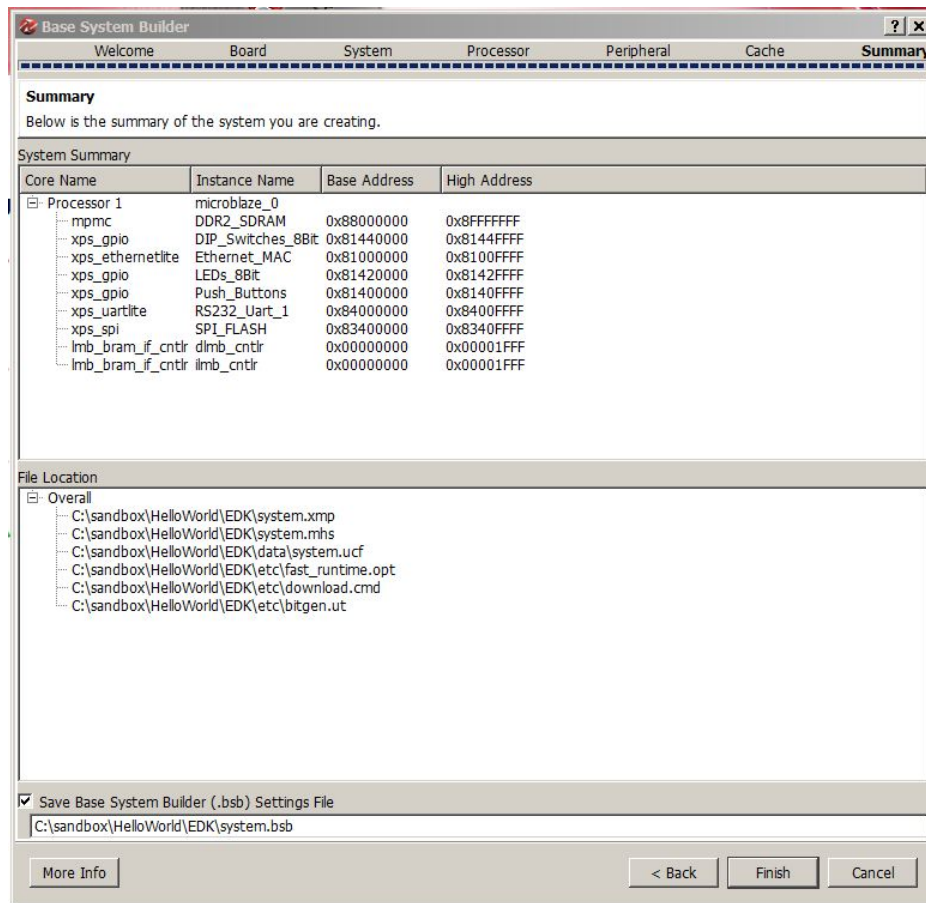
The wizard will present a peripheral selection page similar to this one:



### Instruction and Data Cache

The configuration of instruction and data cache is optional and configurable. We are not using cache in this example so you may skip the Wizard page. If you choose to use either or both caches the simply select the appropriate check box and choose the available displayed memory options, these come from the configuration of your evaluation board or your custom design.

### Base System Summary Page



Click the "Finish" button to save the base system and exit the Wizard.

The next steps will demonstrate how to configure the MicroBlaze processor and create the necessary I/O interfaces for our sample application.

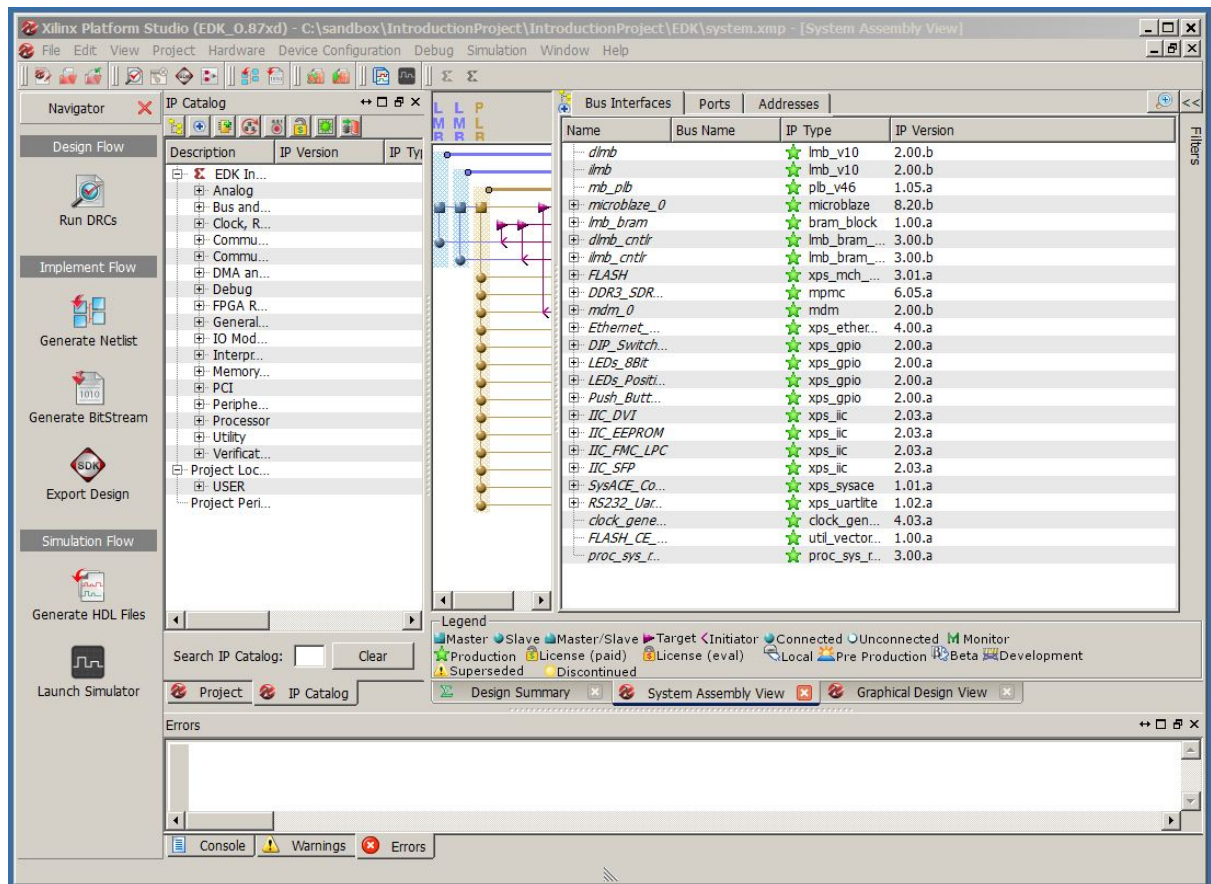
**See Also**

[Importing the Generated Hardware to the Xilinx Tools](#)

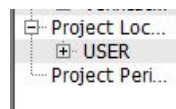
**1.1.7 Importing the Generated Hardware to the Xilinx Tools**

**Hello World Tutorial for MicroBlaze, Step 7**

After the Base System Builder Wizard finishes you will be taken to the Xilinx Platform Studio System Assembly View.



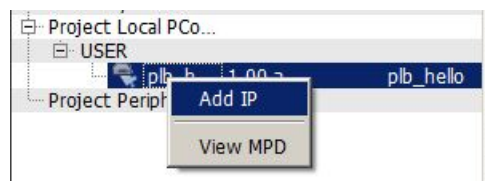
Next, add the module representing the HelloWorld hardware process to your development system. Select the IP Catalog tab. Notice the USER icon.



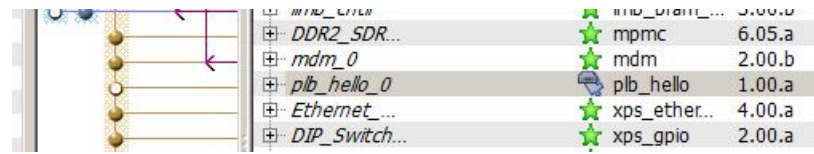
Select the + symbol to open the USER modules. Notice that the plb\_helloworld object.



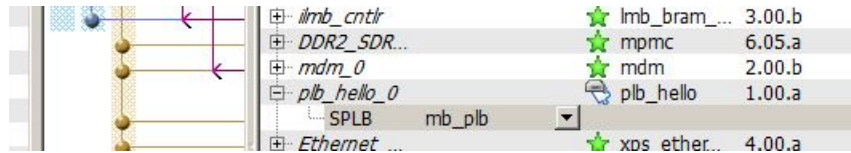
Right-click the plb\_helloworld IP to display its context menu and select Add IP (as shown):



Now find the plb\_helloworld... object in the system assembly view. Move the mouse pointer into the Bus Connections pane (to the left of the Bus Interfaces) and when the pointer is on the same line as the plb\_hello\_0 instance the open slave connection to the PLB will display (as shown);



Note that the circle (indicating a bus slave) for the plb\_hello\_0 instance is empty, click the empty circle to implement the connection to the PLB.



### Specify the Addresses

Now you will need to set the addresses for each of the Peripherals specified for the platform. This can be done simply by choosing the Addresses tab resulting in a window that is similar to this:

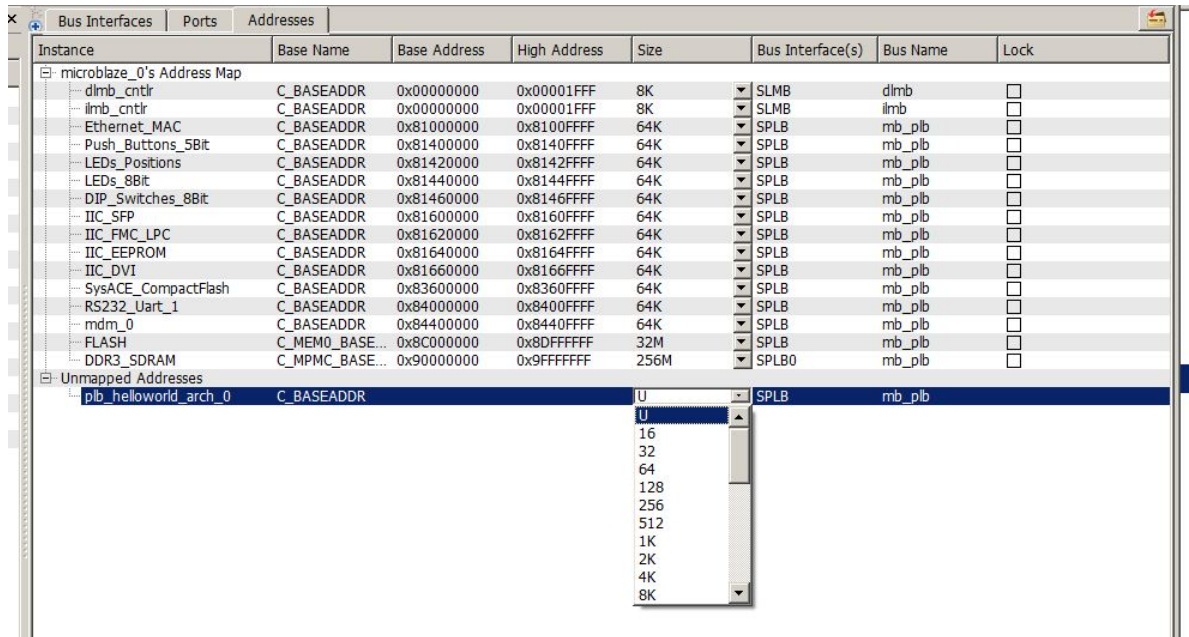
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	dmb	<input type="checkbox"/>
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	ilmb	<input type="checkbox"/>
Ethernet_MAC	C_BASEADDR	0x81000000	0x8100FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
Push_Buttons_5Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_Positions	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_8Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
DIP_Switches_8Bit	C_BASEADDR	0x81460000	0x8146FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_SFP	C_BASEADDR	0x81600000	0x8160FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_FMC_LPC	C_BASEADDR	0x81620000	0x8162FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_EEPROM	C_BASEADDR	0x81640000	0x8164FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_DVI	C_BASEADDR	0x81660000	0x8166FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
FLASH	C_MEM0_BASE...	0x8C000000	0x8DFFFFFF	32M	SPLB	mb_plb	<input type="checkbox"/>
DDR3_SDRAM	C_MPMC_BASE...	0x90000000	0x9FFFFFFF	256M	SPLB0	mb_plb	<input type="checkbox"/>
Unmapped Addresses							

Expand the "Unmapped Address" line by clicking the "+" icon to locate the plb\_helloworld\_arch\_0 object instance.

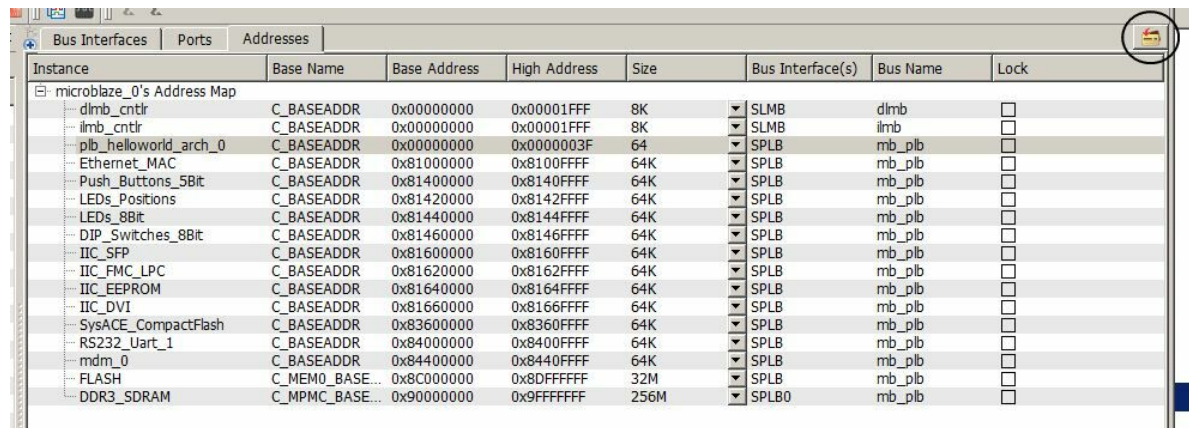
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	dmb	<input type="checkbox"/>
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	ilmb	<input type="checkbox"/>
Ethernet_MAC	C_BASEADDR	0x81000000	0x8100FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
Push_Buttons_5Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_Positions	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_8Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
DIP_Switches_8Bit	C_BASEADDR	0x81460000	0x8146FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_SFP	C_BASEADDR	0x81600000	0x8160FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_FMC_LPC	C_BASEADDR	0x81620000	0x8162FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_EEPROM	C_BASEADDR	0x81640000	0x8164FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_DVI	C_BASEADDR	0x81660000	0x8166FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
FLASH	C_MEM0_BASE...	0x8C000000	0x8DFFFFFF	32M	SPLB	mb_plb	<input type="checkbox"/>
DDR3_SDRAM	C_MPMC_BASE...	0x90000000	0x9FFFFFFF	256M	SPLB0	mb_plb	<input type="checkbox"/>
Unmapped Addresses							
plb_helloworld_arch_0	C_BASEADDR			U	SPLB	mb_plb	<input type="checkbox"/>



Now open the dropdown for the `plb_helloworld_arch_0` line to open the memory assignment popup.



Address space is not expensive, so even though only a few addresses are needed for the two streams used in this tutorial project, we're selecting 64. We can now see that the `plb_helloworld_arch_0` instance is properly mapped into the MicroBlaze address space.

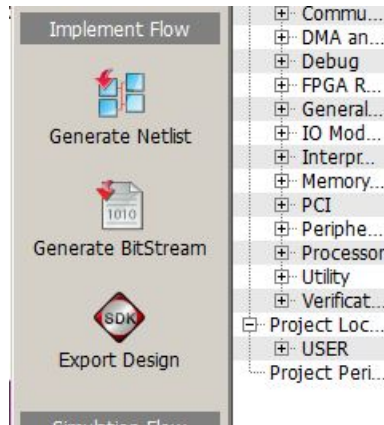


However, if you look carefully you will find that several addresses overlap. To correct this problem click the button (circled above) to remap the memory space. If your software will use fixed memory access based on this mapping it would be prudent to select the "Lock" box for those memories you want fixed. Subsequent remap activities will not change them.

**NOTE:** Please remember that the MicroBlaze is a Harvard Architecture processor. It has separate data and instruction address spaces. It is normal and acceptable for the `dlimb` and `ilmb` controllers to have overlapping addresses spaces, they are two different physical memories.

## Generate Netlist and BitStream

It is now time to generate a netlist, BitStream, and export the design to the Xilinx SDK. The easiest way to do this is to open the Navigator toolbar (View->Toolbars->Navigator) and then sequentially click Generate Netlist, Generate BitStream, and Export Design.



Generate Netlist and Generate BitStream are complex operations that take significant wall clock time. Depending on the size of the design (the example is very small) and your systems memory and processor, even trivial designs will take at least 30 minutes and large complex designs may take in excess of eight hours. Should you find errors, the most likely cause will be overlapping address spaces that have not been corrected.

When you click Export Design, the design will be pushed out to the correct locations and the SDK tool will be opened to allow building the software side of our project.

## See Also

[Importing the Application Software to the Xilinx Tools](#)

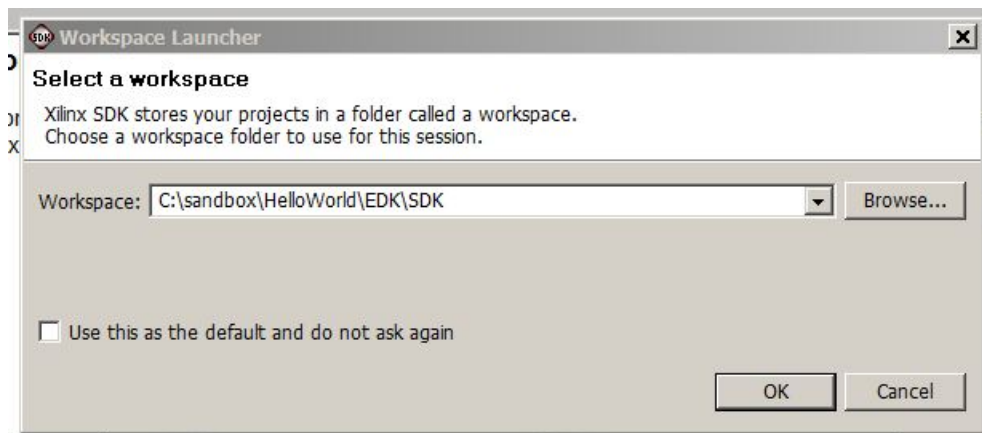
### 1.1.8 Importing the Application Software to the Xilinx Tools

#### Hello World Tutorial for MicroBlaze, Step 8

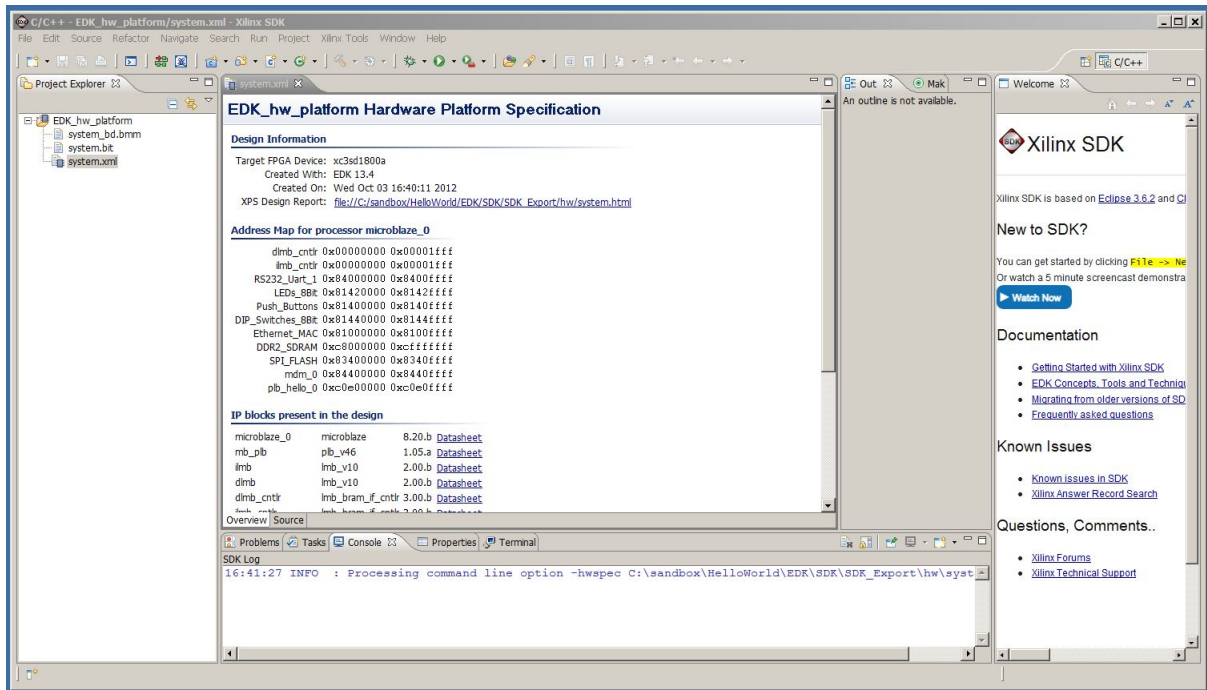
After clicking the Export Design button the popup (shown) is displayed giving you the option to perform an Export only or to Export files and start the Software Development Kit (SDK), for this tutorial please click "Export & Launch SDK".



After the SDK tool starts, the following popup will be displayed. The export tool will have correctly filled in the project workspace path data. We do not recommend setting this as the "default" since it is not likely to be the default for other projects.

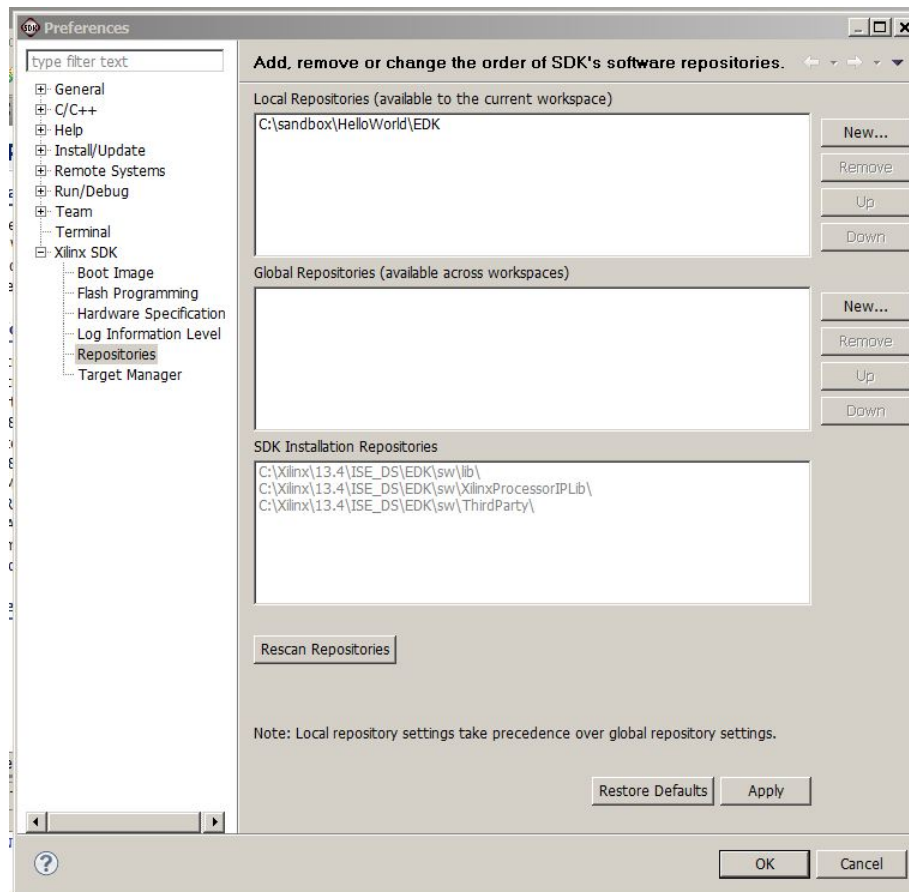


Once the SDK window is open you can close the EDK window as it will not be needed further for this tutorial. Your screen should be similar to the one shown:



## Configure SDK

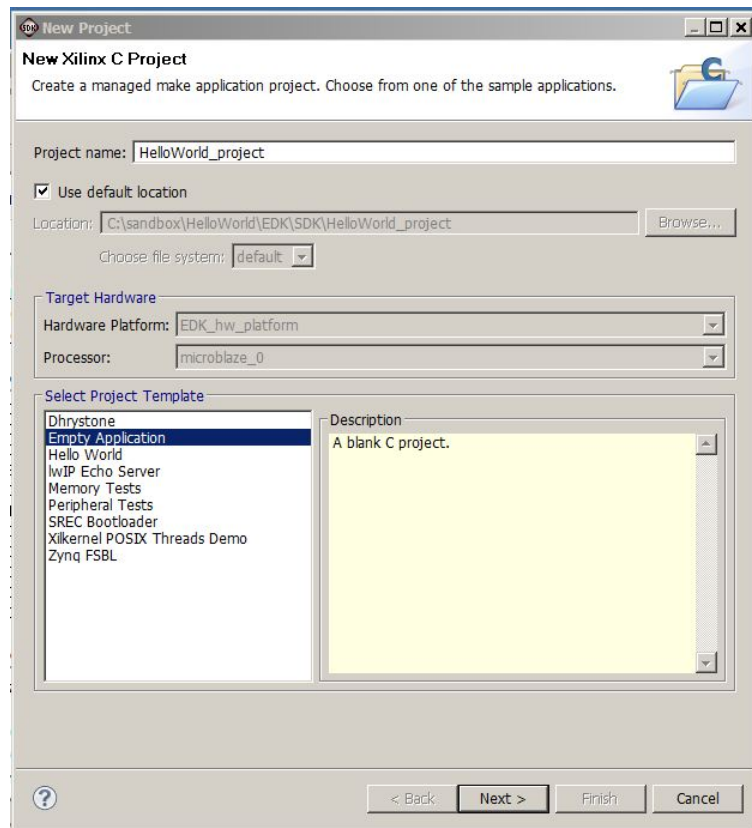
It is first necessary to configure the SDK environment to reference the EDK project directory as a "repository" in order to reference the FPGA logic driver by choosing the "Repositories" menu item from the "Xilinx Tools" menu. Click the "New" button next to the "Local Repositories" then navigate to the project's EDK directory and select it. The Preferences dialog should be similar to that shown here:



Click "Apply" and then "OK" to continue.

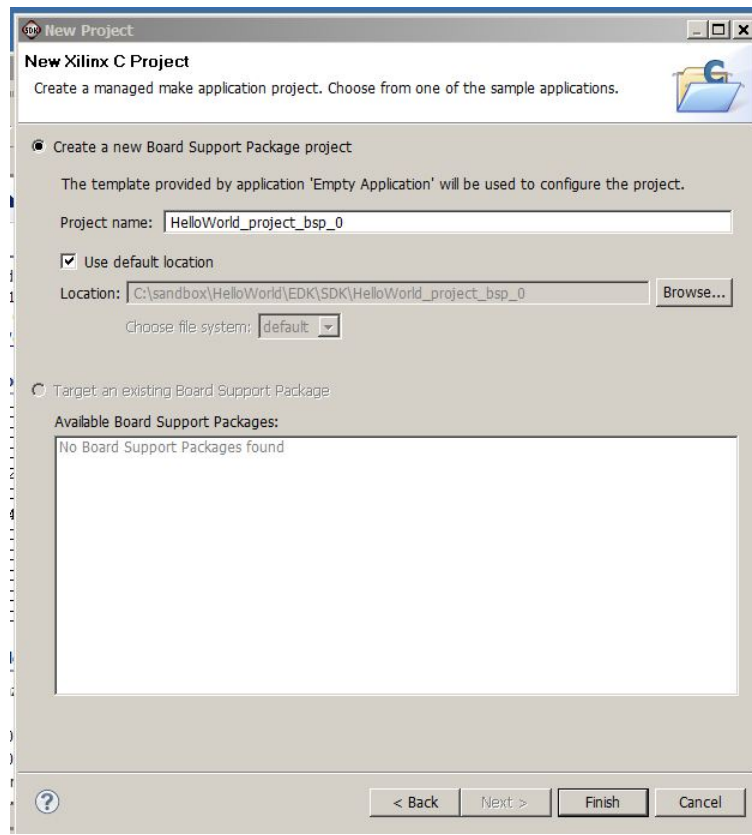
## Create the HelloWorld Project

Choose **File -> New -> Xilinx C Project** from the File menu. Then select **Empty Application** and set the **Project name:** to HelloWorld\_project. The window will appear similar to what is shown here:



Click **Next** to continue.

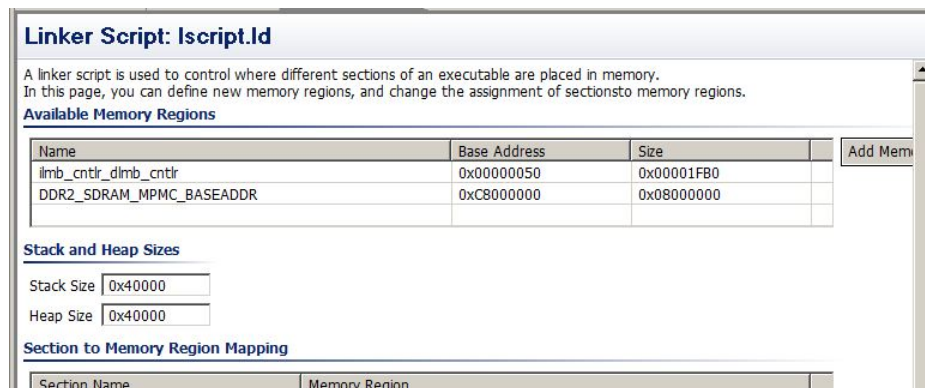
On the next dialog window set the Project name: to HelloWorld\_project\_bsp\_0; the window will now appear similar to what is shown here:



Click **Finish** to continue.

The **Project Explorer** will now include the two new projects, HelloWorld\_project and HelloWorld\_project\_bsp\_0. Notice that as soon as the files were read into the project that the entire project was built. This is the default behavior of SDK. If you followed this tutorial accurately there should have been no errors requiring edits.

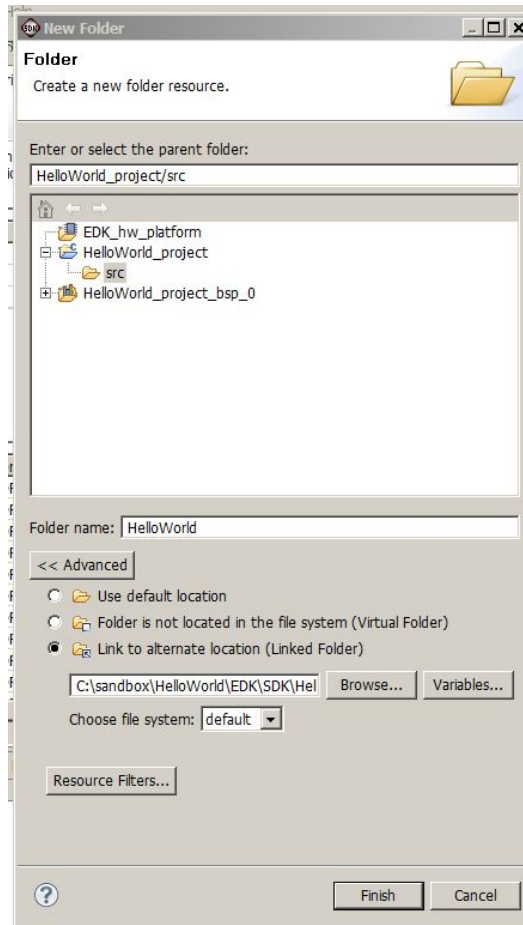
It is now necessary to increase the Stack and Heap memory sizes using the linker script. From the Project Explorer open the lscript.ld file, you will find it in the HelloWorld\_project src directory. Change the Stack and Heap sizes to 0x4000 as shown here:



Either click the file save icon or enter <ctrl>s to save the modified linker script.

Now the software files exported from CoDeveloper must be added to the project. We will do this by linking to their location as follows:

Right click the **src** directory in the HelloWorld\_project and select **New -> Folder**. Click the **Advanced** button and select **Link to alternate location (Linked Folder)**. Use the browse button to locate the HelloWorld\EDK\SDK\HelloWorld directory. The New Folder dialog should look similar to that shown here:



When you click Finish to continue the SDK will automatically rebuild the entire project and prepare the HelloWorld\_project.elf file. In the next step, the BitStream and ELF files will be downloaded into the FPGA.

## See Also

[Building and Downloading the Application](#)

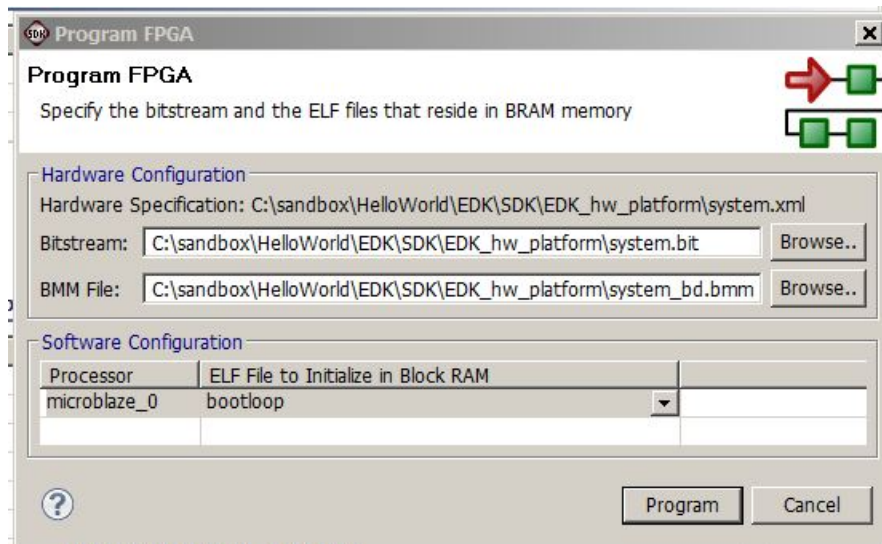
### 1.1.9 Building and Downloading the Application Setting up the Hardware



Configure your hardware for downloading a BitStream by following the detailed instructions provided by the target board vendor.

## Downloading the FPGA BitStream

From the menu bar, select **Xilinx Tools -> Program FPGA**. The Program FPGA dialog will pop-up (as shown), click the **Program** button to complete the process.



## Downloading and Running the Software Binary

To download and run the software binary, in the **Project Explorer** view, right-click the **HelloWorld\_project** project and select **Run As -> Launch on Hardware**. This will cause **SDK** to download the software binary and start the application

This tutorial is completed.

## 1.2 Tutorial 2: Complex FIR Filter for Xilinx Design Suite 13.4 (or higher)



---

### Overview

This detailed tutorial will demonstrate how to use **Impulse C** to create, compile and optimize a digital signal processing (**DSP**) example for the **MicroBlaze** platform. We will also show how to make use of the **Fast Simplex Link (FSL)** bus provided in the **MicroBlaze** platform.

The goal of this application will be to compile the algorithm (a **Complex FIR Filter** function) on hardware on the FPGA. The **MicroBlaze** will be used to run test code (producer and consumer processes) that will pass text data into the algorithm and accept the results.

This example makes use of the **Xilinx ML605 Evaluation Board**. The board features a **Virtex-6 FPGA** with a **MicroBlaze** soft processor. This tutorial also assumes you are using the **Xilinx Design Suite 13.4** (or later) development tools.

This tutorial will require approximately two hours to complete, including software run times.

*Note: this tutorial is based on a sample DSP application developed by Bruce Karsten of Xilinx, Inc. A more complete description of the algorithm can be found in the **Impulse C User Guide**. This tutorial assumes that you are familiar with the basic steps involved in using the **Xilinx EDK** tools. For brevity this tutorial will omit some EDK details that are covered in introductory EDK and Impulse C tutorials.*

*Note also that most of the detailed steps in this tutorial only need to be performed once, during the initial creation of your **MicroBlaze** application. Subsequent changes to the application do not require repeating these steps.*

### Steps

- [Loading the Complex FIR Application](#)
- [Understanding the Complex FIR Application](#)
- [Compiling the Application for Simulation](#)
- [Building the Application for the Target Platform](#)
- [Creating the Platform Using the Xilinx Tools](#)
- [Configuring the New Platform](#)
- [Exporting Files from CoDeveloper](#)
- [Importing the Generated Hardware](#)
- [Generating the FPGA Bitmap](#)
- [Importing the Application Software](#)
- [Running the Application](#)

### See Also

- [Tutorial 1: Hello World on the MicroBlaze platform](#)

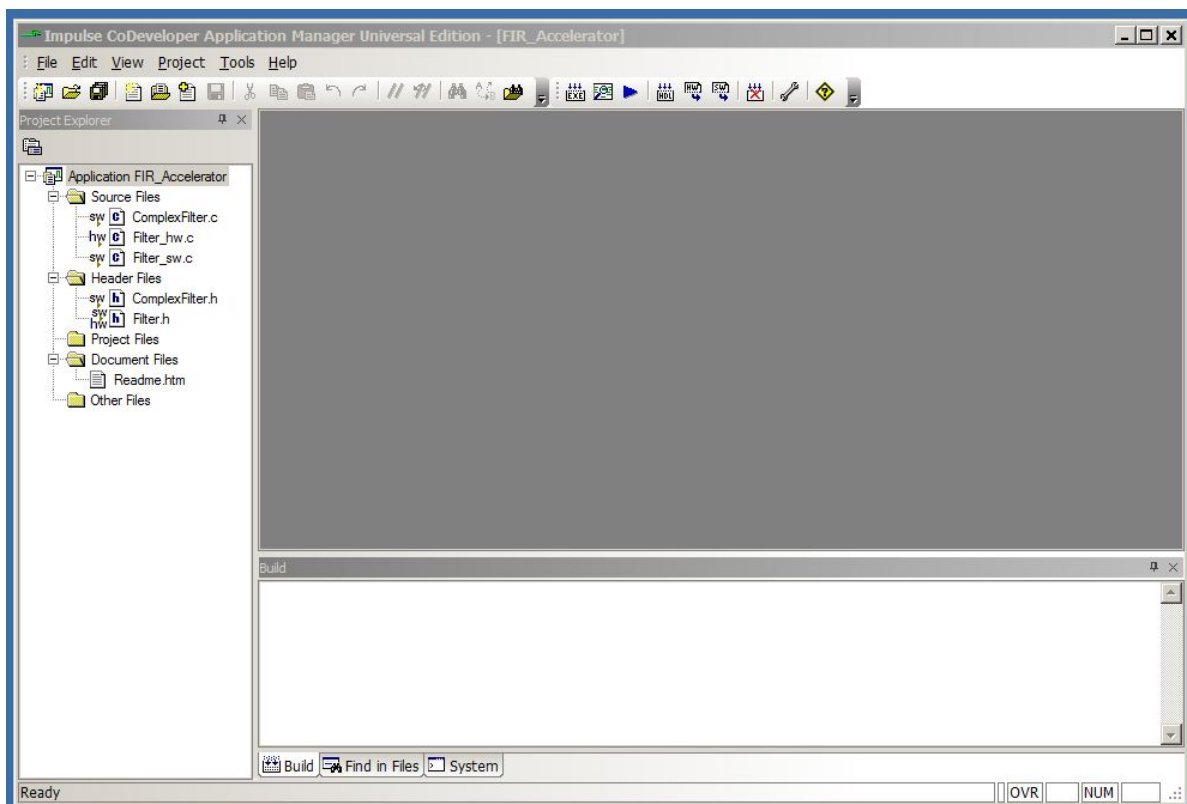
## 1.2.1 Loading the Complex FIR Application

### Complex FIR Filter Tutorial for MicroBlaze, Step 1

To begin, start the **CoDeveloper Application Manager** by selecting from the **Windows Start -> Programs -> Impulse Accelerated Technologies -> CoDeveloper Application Manager** program group.

*Note: this tutorial assumes that you have read and understand the **Complex FIR Filter** example and tutorial presented in the main **CoDeveloper** help file.*

Open the **Xilinx MicroBlaze ComplexFIR** sample project by selecting **Open Project** from the **File** menu, or by clicking the **Open Project** toolbar button. Navigate to the `.\Examples\Embedded\ComplexFIR_MicroBlaze\` directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) The project file is also available online at <http://impulsec.com/ReadyToRun/>. The screen shots in this tutorial show the working directory being `c:\sandbox\CompFIRmb`. Opening the project will result in the display of a window similar to the following:



Files included in the **Complex FIR Filter** project include:

Source files **ComplexFilter.c**, **Filter\_hw.c** and **Filter\_sw.c** - These source files represent the complete application, including the **main()** function, consumer and producer software processes and a single hardware process.

Header files **ComplexFilter.h** and **Filter.h** - function prototypes and definitions.

## See Also

[Understanding the Complex FIR Application](#)

## 1.2.2 Understanding the Complex FIR Application

### Complex FIR Filter Tutorial for MicroBlaze, Step 2

Before compiling the **Complex FIR Filter** application to hardware, let's first take a moment to understand its basic operation and the contents of its primary source files, and in particular **Filter\_hw.c**.

The specific process that we will be compiling to hardware is represented by the following function (located in **Filter\_hw.c**):

```
void complex_fir(co_stream filter_in, co_stream filter_out)
```

This function reads two types of data:

- Filter coefficients used in the **Complex FIR Filter** convolution algorithm.
- An incoming data stream

The results of the convolution are written by the process to the stream **filter\_out**.

The **complex\_fir** function begins by reading the coefficients from the **filter\_in** stream and storing the resulting data into a local array (**coef\_mem**). The function then reads and begins processing the data, one at a time. Results are written to the output stream **filter\_out**.

The repetitive operations described in the **complex\_fir** function are complex convolution algorithm.

The complete test application includes test routines (including **main**) that run on the **MicroBlaze** processor, generating test data and verifying the results against the legacy C algorithm from which **complex\_fir** was adapted.

The configuration that ties these modules together appears toward the end of the **Filter\_hw.c** file, and reads as follows:

```
void config_filt (void *arg) {
    int i;

    co_stream to_filt, from_filt;
    co_process cpu_proc, filter_proc;

    to_filt = co_stream_create ("to_filt", INT_TYPE(32), 4);
    from_filt = co_stream_create ("from_filt", INT_TYPE(32), 4);

    cpu_proc = co_process_create ("cpu_proc", (co_function)
    call_accelerator, 2, to_filt, from_filt);
```

```
    filter_proc = co_process_create ("filter_proc", (co_function)
complex_fir,      2, to_filt, from_filt);

    co_process_config (filter_proc, co_loc, "PE0");
}
```

As in the **Hello World** example (described in the main **CoDeveloper** help file), this configuration function describes the connectivity between instances of each previously defined process.

Only one process in this example (**filter\_proc**) will be mapped onto hardware and compiled by the Impulse C compiler. This process (**filter\_proc**) is flagged as a hardware process through the use of the **co\_process\_config** function, which appears here at the last statement in the configuration function. **Co\_process\_config** instructs the compiler to generate hardware for **complex\_fir** (or more accurately, the instance of **complex\_fir** that has been declared here as **filter\_proc**).

NOTE: In many cases the name of the hardware process and the name of the instance of the hardware process are the same. This tutorial points out that the instance name and the definition name need not be the same. This ability makes it possible to replicate a hardware module multiple times in a single application.

The **ComplexFilter.c** generates a set of **Complex FIR Filter** coefficients and also a group of input data being processed.

The **Filter\_sw.c** will run in the **MicroBlaze** embedded processor, controlling the stream flow and printing results.

## See Also

[Compiling the Application for Simulation](#)

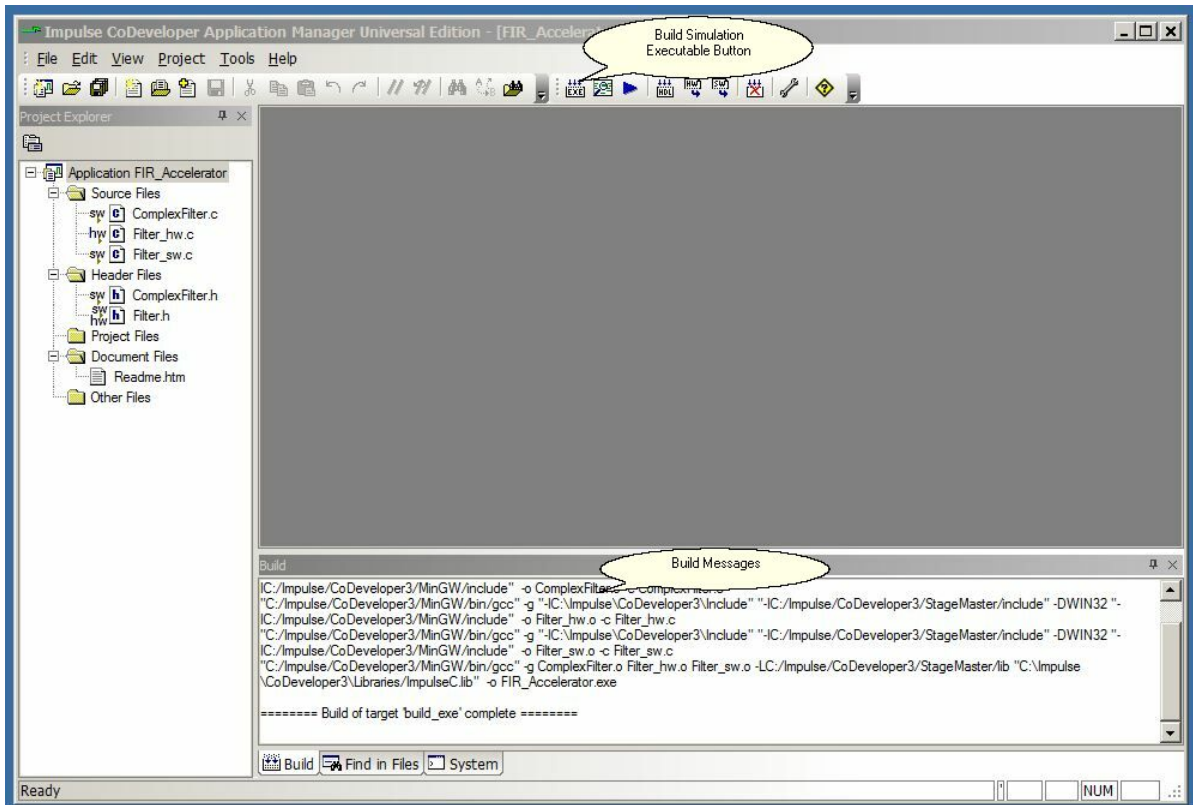
### 1.2.3 Compiling the Application for Simulation

#### Complex FIR Filter Tutorial for MicroBlaze, Step 3

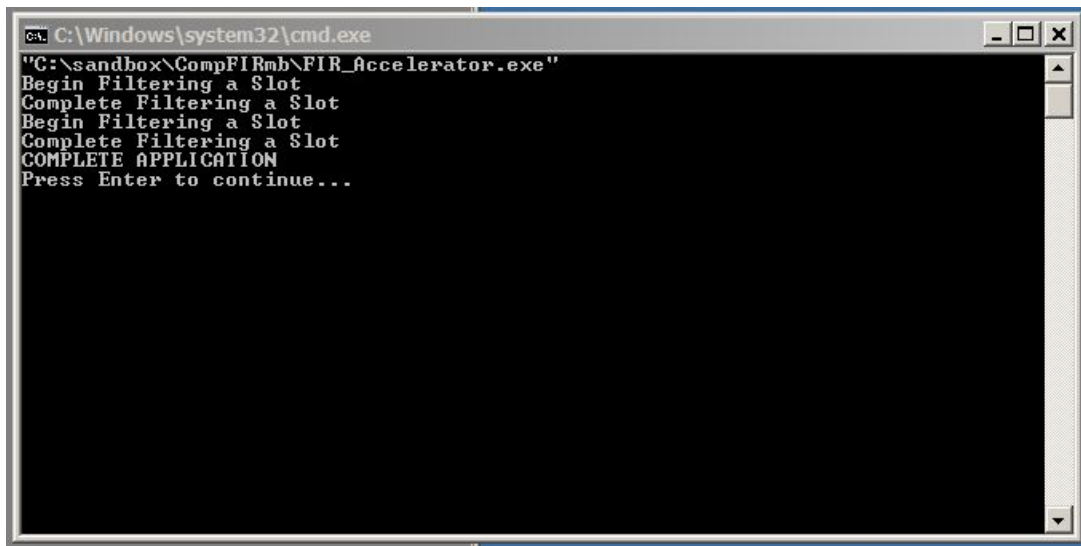
Simulation allows you to verify the correct operation and functional behavior of your algorithm before attempting to generate hardware for the FPGA. When using Impulse C, simulation simply refers to the process of compiling your C code to the desktop (host) development system using a standard C compiler, in this case the **GCC** compiler included with the Impulse **CoDeveloper** tools.

To compile and simulate the application for the purpose of functional verification:

1. Select **Project -> Build Software Simulation Executable** (or click the **Build Software Simulation Executable** button) to build the **FIR\_Accelerator.exe** executable. A command window will open, displaying the compile and link messages as shown below:



2. You now have a Windows executable representing the **Complex FIR Filter** application implemented as a desktop (console) software application. Run this executable by selecting **Project -> Launch Simulation Executable**. A command window will open and the simulation executable will run as shown below:



Verify that the simulation produces the output shown. Note that although the messages indicate that the **ComplexFIR** algorithm is running on the FPGA, the application (represented by hardware and software processes) is actually running entirely in software as a compiled, native Windows executable. The

messages you will see have been generated as a result of instrumenting the application with simple `printf` statements such as the following:

```
#if defined(MICROBLAZE)
    xil_printf ("COMPLETE APPLICATION\r\n");
    return 0;
#else
    printf ("COMPLETE APPLICATION\r\n");
    printf ("Press Enter to continue...\r\n");
    c = getc(stdin);
#endif
```

Notice in the above C source code that **#ifdef** statements have been used to allow the software side of the application to be compiled either for the embedded **MicroBlaze** processor, or to the host development system for simulation purposes.

### See Also

[Building the Application for the Target Platform](#)

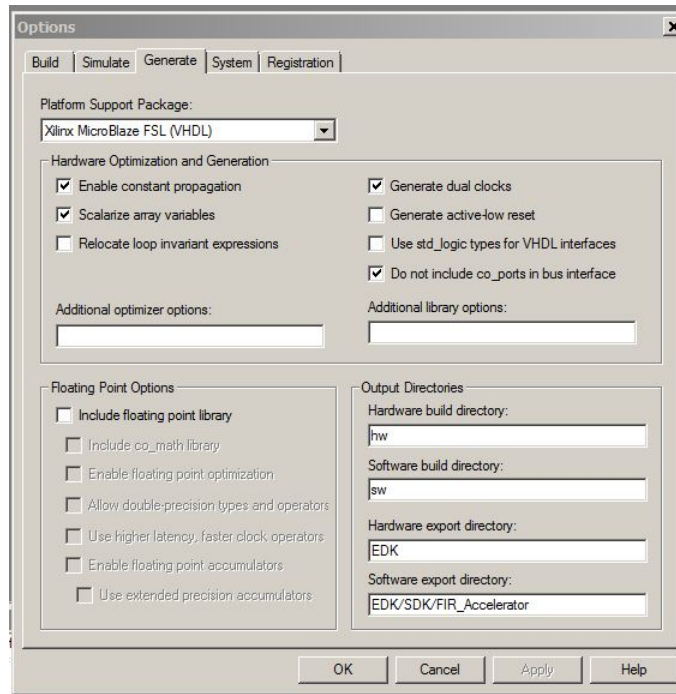
## 1.2.4 Building the Application for the Target Platform

### Complex FIR Filter Tutorial for MicroBlaze, Step 4

The next step in the tutorial is to create FPGA hardware and related files from the C code found in the **Filter\_hw.c** source file. This requires that we select a platform target, specify any needed options, and initiate the hardware compilation process.

### Specifying the Platform Support Package

To specify a platform target, open the **Generate** tab of the **Options** dialog as shown below:

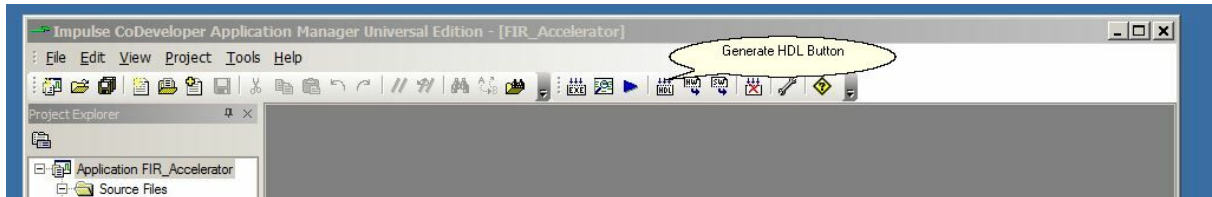


Specify **Xilinx MicroBlaze FSL (VHDL)**. Also specify **hw** and **sw** for the hardware and software directories as shown, and specify **EDK** for the hardware and software export directories. Also ensure that the **Generate dual clocks** option is checked.

Click **OK** to save the options and exit the dialog.

### Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces and library files, select **Generate HDL** from the **Project** menu, or select the **Generate HDL** toolbar button as shown below:



A series of processing steps will run in a command window as shown below:



```

Build
| Block #10:
| Stages: 1
| Max. Unit Delay: 0
| Block #11:
| Stages: 1
| Max. Unit Delay: 0
-----
| Operators:
| 10 Adder(s)/Subtractor(s) (32 bit)
| 4 Multiplier(s) (16 bit)| 1 Comparator(s) (2 bit)
| 5 Comparator(s) (32 bit)
-----
| Total Stages: 19
| Max. Unit Delay: 64
| Estimated DSPs: 4
-----
Writing output ... done
"C:/Impulse/CoDeveloper3/bin/impulse_genvhdl.exe" FIR_Accelerator.xhw hw/FIR_Accelerator_comp.vhd
Impulse C RTL Component Generator
Copyright 2002-2009, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating pe0/complex_fir ...
Component generation complete
---Software activated---
"C:/Impulse/CoDeveloper3/bin/impulse_arch" "-aC:/Impulse/CoDeveloper3/Architectures/xilinx_mb_fsl.xml" -dc -no_port_bus_connect -swdirsw files
"FIR_Accelerator_comp.vhd FIR_Accelerator_top.vhd" FIR_Accelerator.xic hw/FIR_Accelerator_top.vhd
Impulse C HDL Design Generator
Copyright 2002-2012, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/xilinx_mb_fsl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Xilinx/FSL/bus.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/target.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Xilinx/technology.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Xilinx/MicroBlaze/FSL/standalone.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Xilinx/OPB/bus.xml ...
Loading FIR_Accelerator.xic ...
libCoreGenCommon.tcl:-GenerateCoreGenFilesForEDK():Starting
ngcString =
libCoreGenCommon.tcl:-GenerateCoreGenFilesForEDK():Exiting
Design generation complete
chmod -R +rw hw
mkdir sw
"C:/Impulse/CoDeveloper3/bin/impulse_lib" "-aC:/Impulse/CoDeveloper3/Architectures/xilinx_mb_fsl.xml" -hwdirhw files "ComplexFilter.c Filter_sw.c"
FIR_Accelerator.xic sw/co_init.c
Impulse C Software Interface Generator
Copyright 2002-2012, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/xilinx_mb_fsl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Xilinx/MicroBlaze/FSL/cpu.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Xilinx/MicroBlaze/FSL/standalone.xml ...Loading FIR_Accelerator.xic ...
for i in ComplexFilter.c Filter_sw.c; do cp $i sw; done
for i in ComplexFilter.h Filter.h; do cp $i sw; done
chmod -R +rw sw

===== Build of target 'build' complete =====
Build Find in Files System
NUM

```

*Note: the processing of this example may require a few minutes to complete, depending on the performance of your system.*

When processing has completed you will have a number of resulting files created in the **hw** and **sw** subdirectories of your project directory.

## See Also

[Exporting Files from CoDeveloper](#)

## 1.2.5 Exporting Files from CoDeveloper

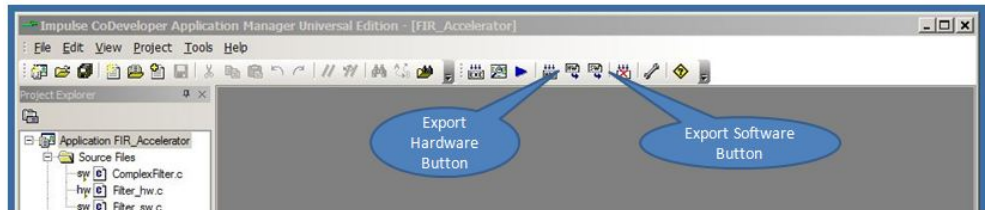
### Complex FIR Filter Tutorial for MicroBlaze, Step 5

Recall that in [Step 4](#) you specified the directory **EDK** as the export target for hardware and software. These export directories specify where the generated hardware and software processes are to be copied when the **Export Software** and **Export Hardware** features of **CoDeveloper** are invoked.

Within these target directories (in this case **EDK**), the specific destination (which may be a subdirectory under **EDK**) for each file previously generated is determined from the **Platform Support Package** architecture library files. It is therefore important that the correct **Platform Support Package** (in this case **Xilinx MicroBlaze FSL**) is selected prior to starting the export process.

To export the files from the build directories (in this case **hw** and **sw**) to the export directories (in this case the **EDK** directory), select **Project -> Export Generated Hardware (HDL)** and **Project -> Export Generated Software**, or select the **Export Generated Hardware** and **Export Generated Software** buttons from the toolbar.

### Export the Hardware and Software Files



*Note: you must select BOTH **Export Software** and **Export Hardware** before going onto the next step, it does not matter in what order you export them.*

You have now exported all necessary files from **CoDeveloper** to the Xilinx tools environment.

### See Also

[Creating the Platform Using the Xilinx Tools](#)

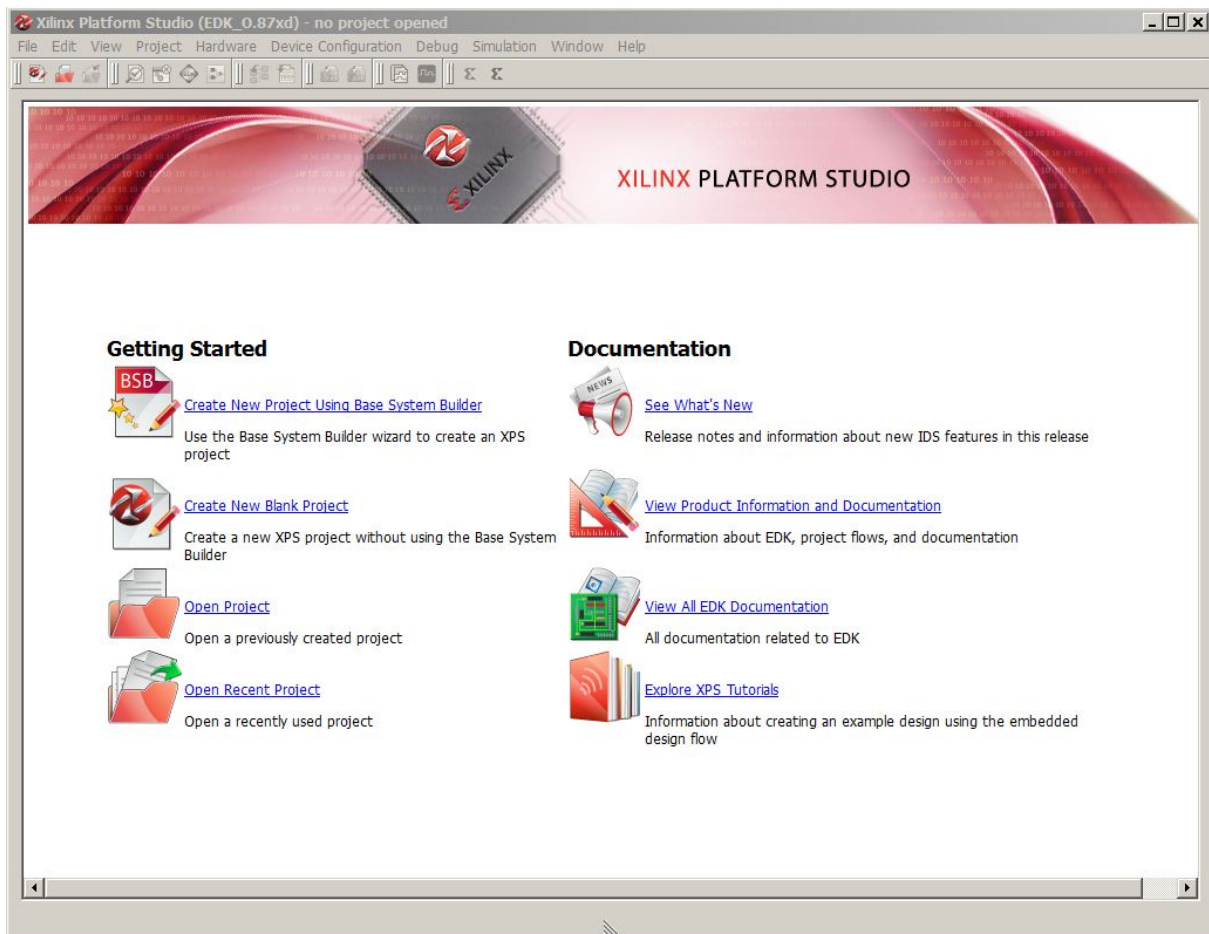
## 1.2.6 Creating a Platform Using Xilinx Tools

### Complex FIR Filter Tutorial for MicroBlaze, Step 6

As you learned in the previous [Hello World tutorial](#), **CoDeveloper** creates a number of hardware and software-related output files that must all be used to create a complete hardware/software application on the target platform (in this case a Xilinx FPGA with an embedded **MicroBlaze** processor). This section will walk you through the file export/import process for this example, using the EDK System Builder (Platform Studio) project.

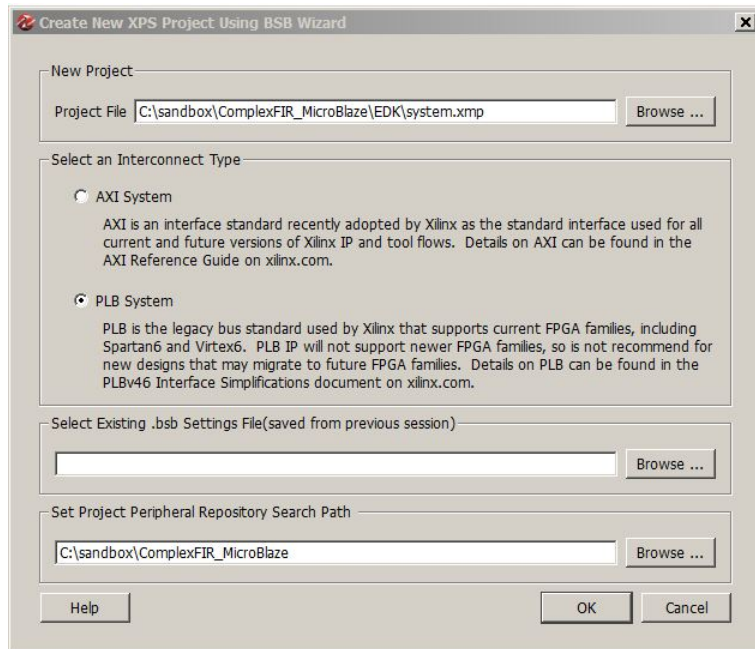
### Creating a New Xilinx Platform Studio Project

Now we'll move into the Xilinx tool environment. Begin by launching **Xilinx Platform Studio** (from the **Windows Start -> Xilinx ISE Design Suite 13.4 -> EDK -> Xilinx Platform Studio**) and creating a new project. The **Xilinx Platform Studio** dialog appears as shown below:



Select the **Base System Builder wizard (recommended)**, and click **OK**.

Next, in the **Create New XPS Project Using BSB Wizard** dialog, click **Browse** and navigate to the directory you created for your **Xilinx EDK** project files. For this tutorial we choose the directory name **EDK**, which is also the directory name we specified earlier in the **Generate Options** dialog. Click **Open** to create a project file called **system.xmp** (you can specify a different project name if desired):



Now click **OK** in the **Create New XPS Project Using BSB Wizard** dialog. The **Base System Builder - Welcome** page will appear. Select **I would like to create a new design** (the default), then click **Next** to choose your target board.

Choose your development board from the dropdown boxes. This example will use the following board (you should choose the reference board you have available for this step):

**Board Vendor: Xilinx**  
**Board Name: Virtex 6 ML605 Evaluation Platform**  
**Board Revision: D**

The screenshot shows the 'Base System Builder' wizard window, specifically the 'Board' tab. The window has a title bar with a question mark and a close button. Below the title bar is a navigation bar with tabs: 'Welcome', 'Board', 'System', 'Processor', 'Peripheral', 'Cache', and 'Summary'. The 'Board' tab is selected.

**Board Selection**  
Select a target development board.

**Board**

I would like to create a system for the following development board

Board Vendor: Xilinx  
Board Name: Virtex 6 ML605 Evaluation Platform  
Board Revision: D

I would like to create a system for a custom board

**Board Information**

Architecture	Device	Package	Speed Grade
virtex6	xc6vx240t	ff1156	-1

Use Stepping  
Reset Polarity: Active High

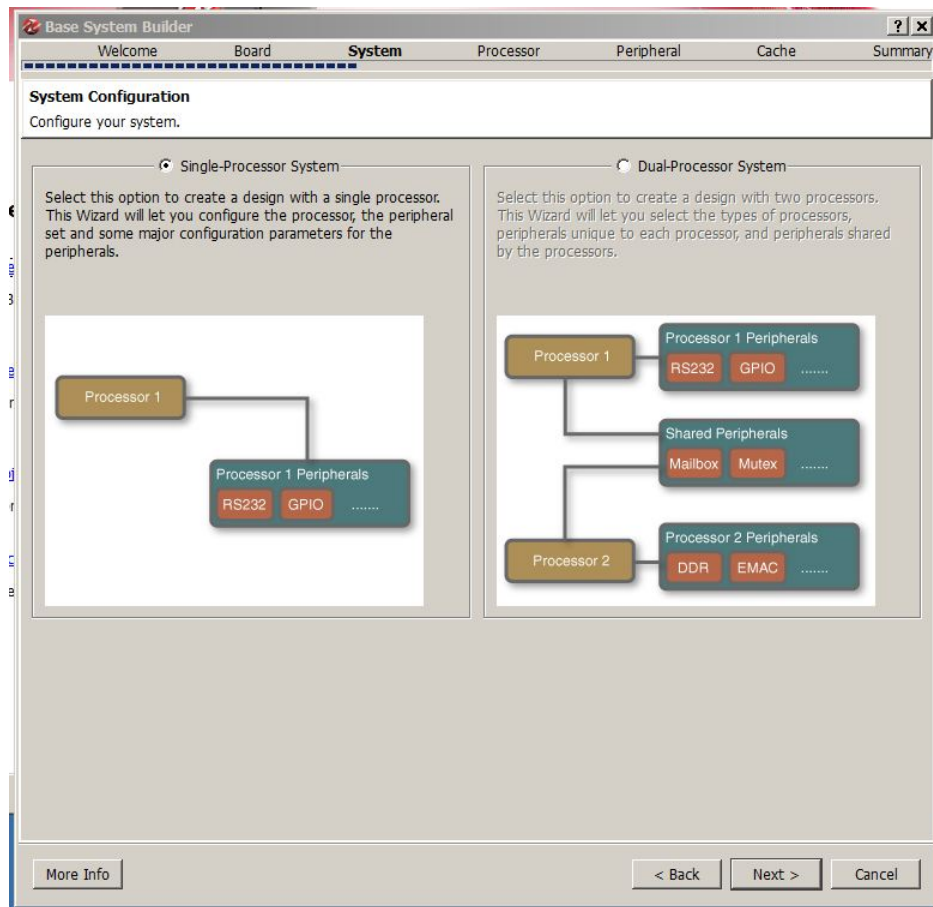
**Related Information**

[Vendor's Website](#)  
[Vendor's Contact Information](#)  
[Third Party Board Definition Files Download Website](#)

The ML605 board is intended to showcase and demonstrate Virtex-6 technology. The ML605 board utilizes Xilinx Virtex 6 XC6VLX240T-FF1156 device. The board includes Gigabit Tri-Mode Ethernet MAC/PHY, 512MB DDR3 SDRAM SODIMM memory, 32MB BPI Linear Flash, 128MB of Platform Flash, 1KB IIC EEPROM, USB peripheral, CPU Debug connectors, System ACE CF controller and RS232 serial port. Additional user desired features can be added through daughter cards attached to the on-board high speed FMC VITA-57 high pin count (HPC) expansion connector or the on-board FMC VITA-57 low pin count (LPC) connector. Due to Answer Record 39430 (<http://www.xilinx.com/support/answers/39430.htm>), PCI Express was removed as an option from ML605 board in IDS 13.1 release.

More Info      < Back      Next >      Cancel

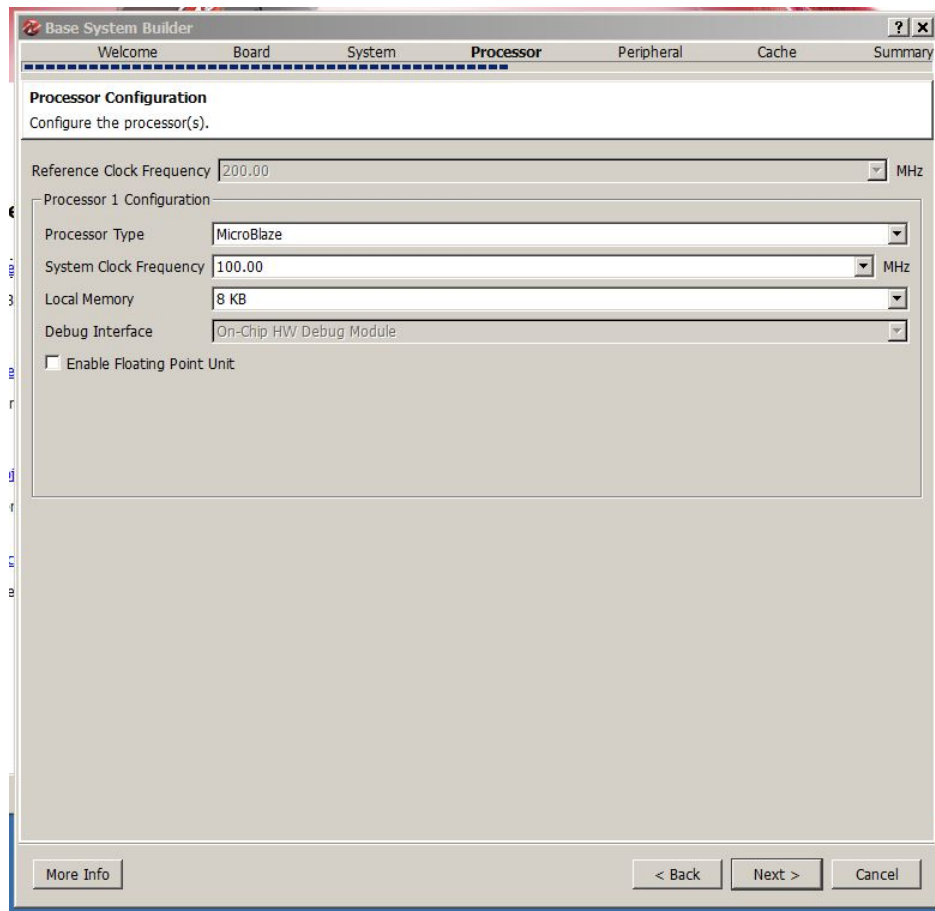
Click **Next** to continue with the **Base System Builder** wizard. In the next wizard page, make sure that **Single-Processor System** is selected:



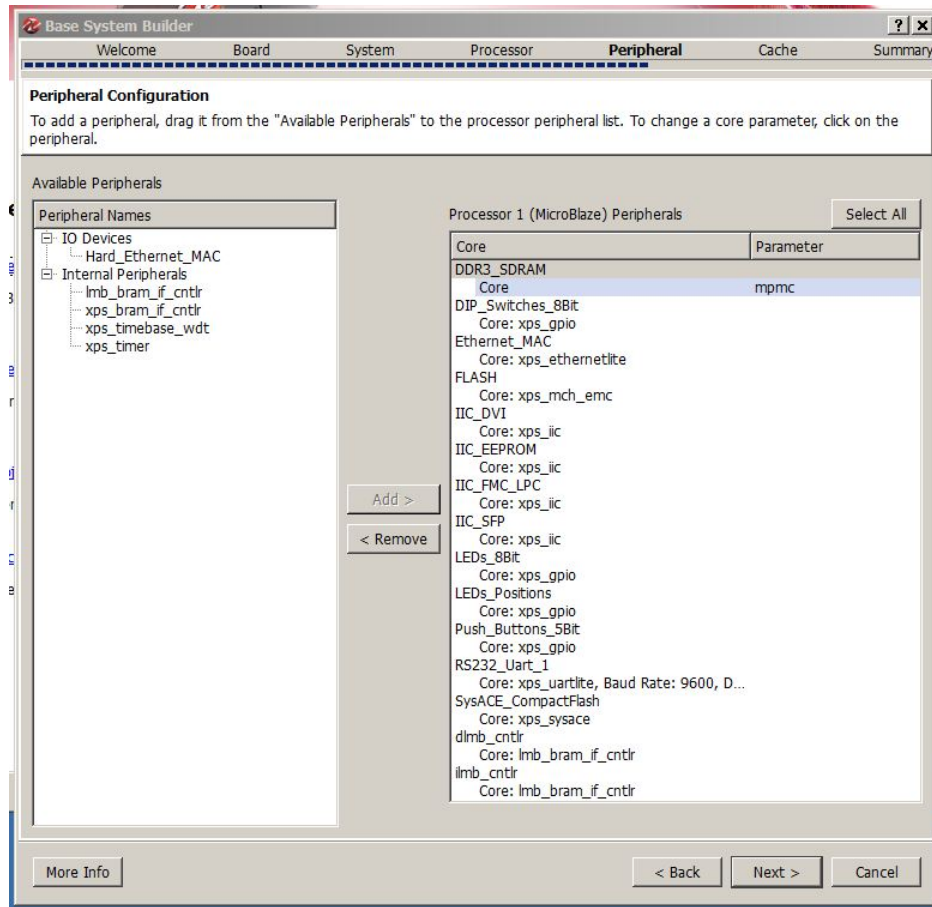
Click **Next** to continue with the **Base System Builder** wizard.

*Note: the **Base System Builder** options that follow may be different depending on the development board you are using.*

In this tutorial we are not using any floating point arithmetic so make sure the "Enable Floating Point Unit" is NOT selected.

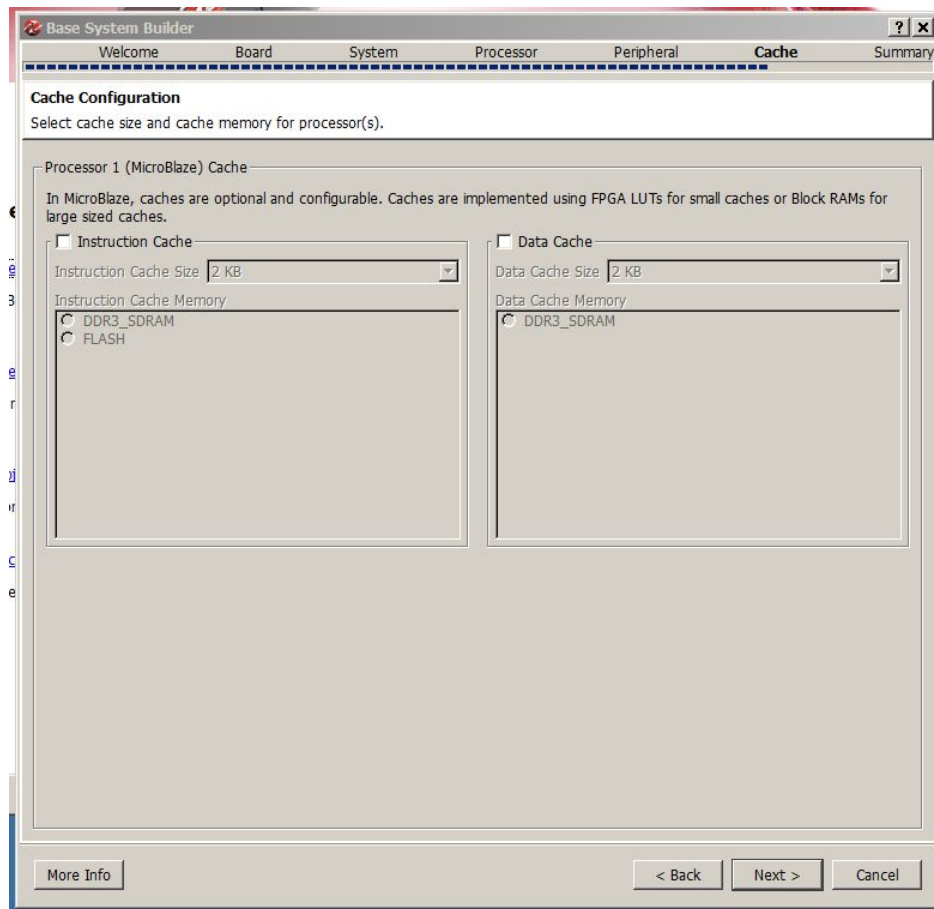


Click "Next" to continue with the Wizard which will display a peripheral selection dialog. Add the `xps_timer` to the configuration by selecting `xps_timer` from the Peripheral Names pane and clicking the Add button.

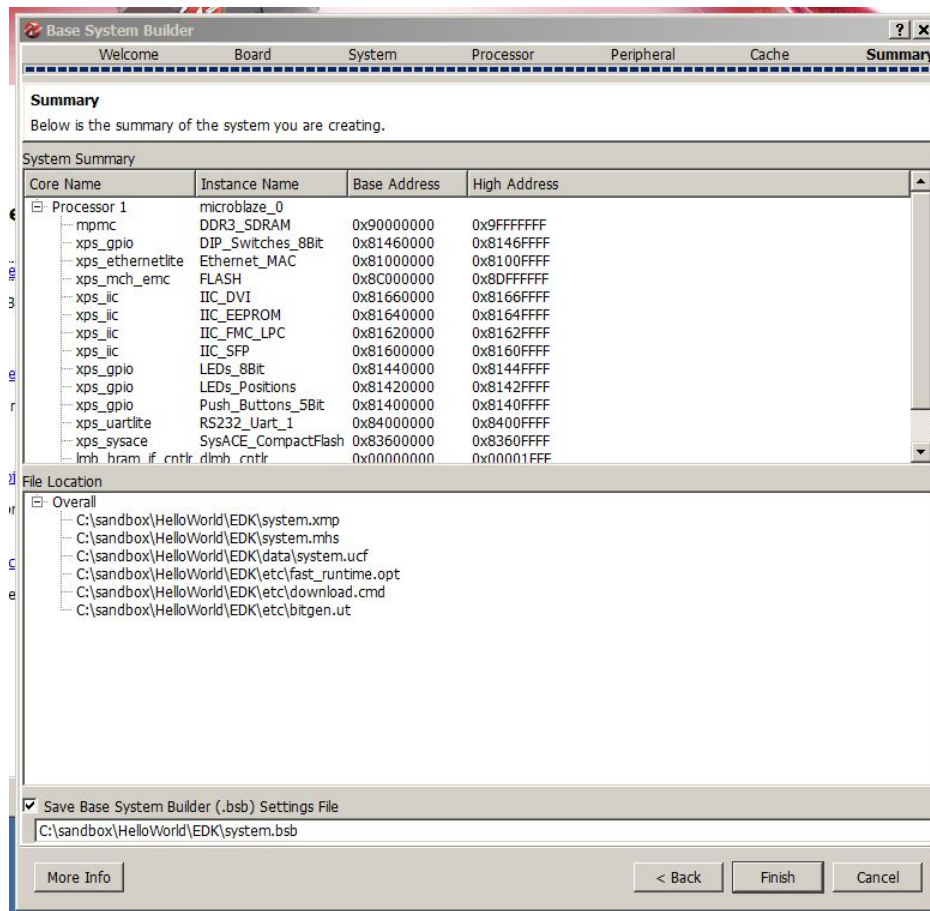


Clicking "Next" will display the Cache Configuration dialog. The MicroBlaze is a "Harvard Architecture" processor having separate data and instruction buses and therefore separate cache's if used. This dialog is used to enable cache and to set the size of each cache. Since this tutorial uses a very small application it is not necessary to use either instruction or data caching and the dialog may be left in its default state.





Click the "Next" button to display the base system Summary dialog.



This summary page shows the location of the various output files that will be used later to prepare the FPGA bit (programming) file. It also lists all of the software cores to be instantiated within the MicroBlaze and their base address. These addresses will be automatically used later, you do not need to record them. Click the "Finish" button.

When you clicked the "Finish" button the Wizard created and opened a Xilinx Platform Studio (EDK) project and populated it with the base system you specified with the Wizard.

The next steps will demonstrate how to configure the **MicroBlaze** processor and create the necessary I/O interfaces for our sample application.

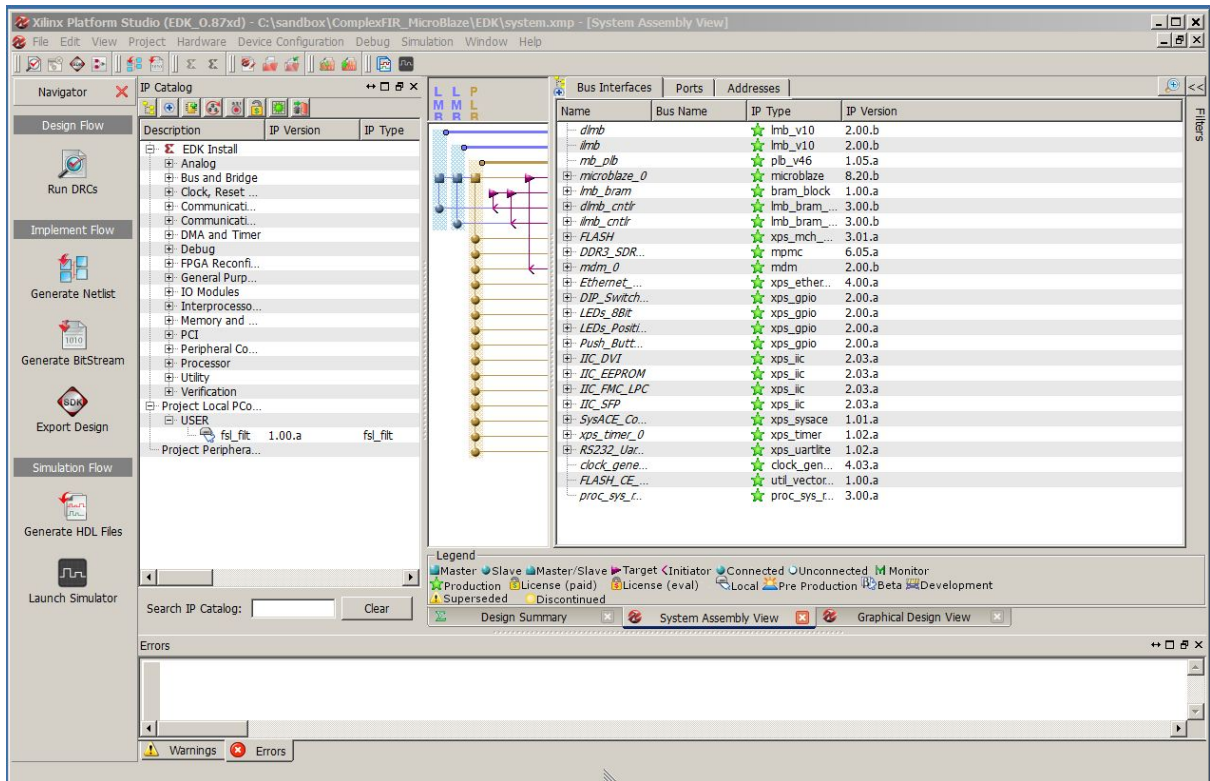
**See Also**

[Configuring the New Platform](#)

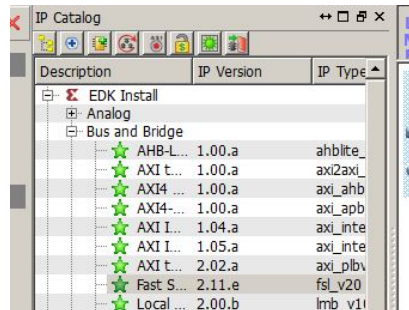
**1.2.7 Configuring the New Platform**

**Complex FIR Filter Tutorial for MicroBlaze, Step 7**

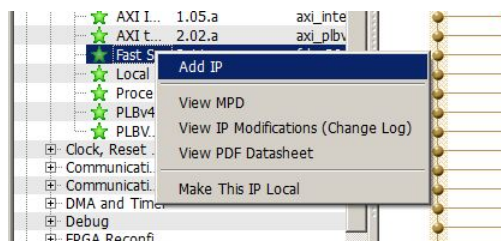
After completing the previous tutorial step you will be presented with a window similar to this:



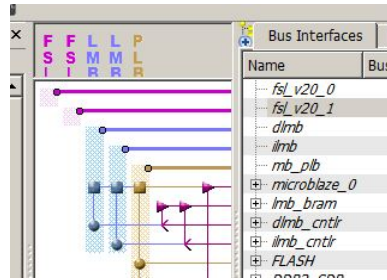
It is now time to connect the hardware peripheral to the MicroBlaze processor. We want to use the FSL (fast simplex link) but there is none attached to the processor. In the IP Catalog pane, expand the Bus and Bridge class and select Fast Simplex Link.



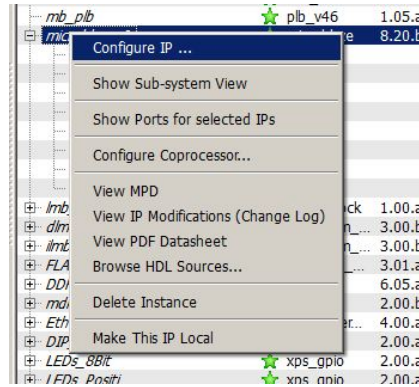
Then right-click the Fast Simplex Link IP to show the instance menu and select Add IP. Do this twice so that two instances of the FSL is added to the system assembly.



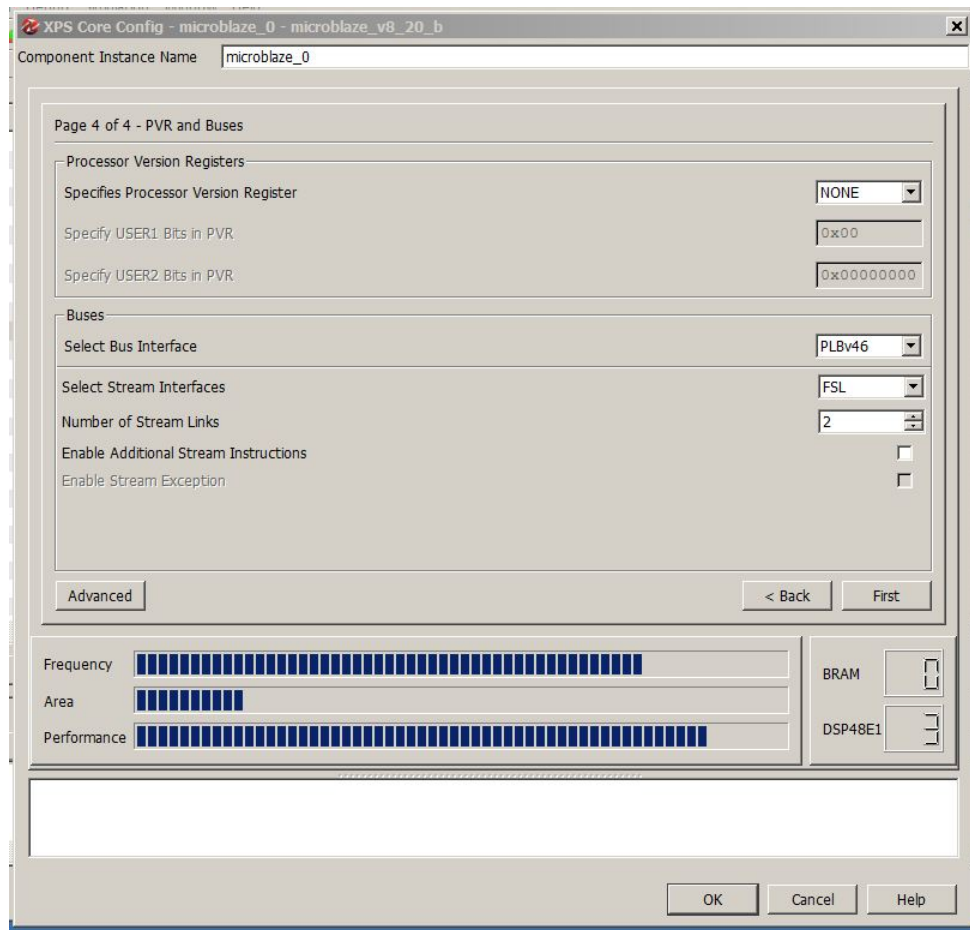
The System Assembly should now be changed to appear similar to this:



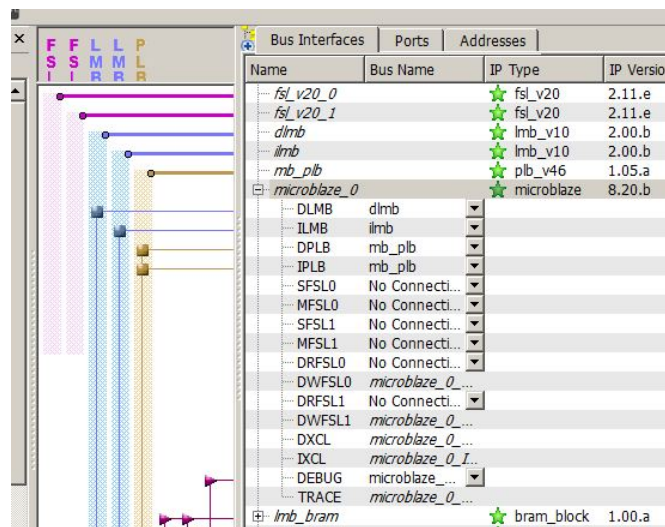
At this point the two FSL modules do not go anywhere, now it is time to connect them to the MicroBlaze processor. Select the microblaze\_0 instance from the System assembly, right-click it and select Configure IP.



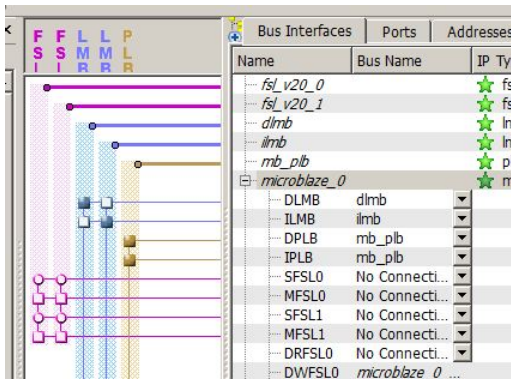
The Configure IP Wizard for the microblaze will pop-up. Click "Next" until you get to page 4 (PVR and Buses). Then change the number of stream links from zero to two, as shown here:



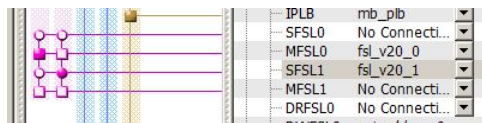
Click "OK" and after a few moments the System Assembly will look similar to this:



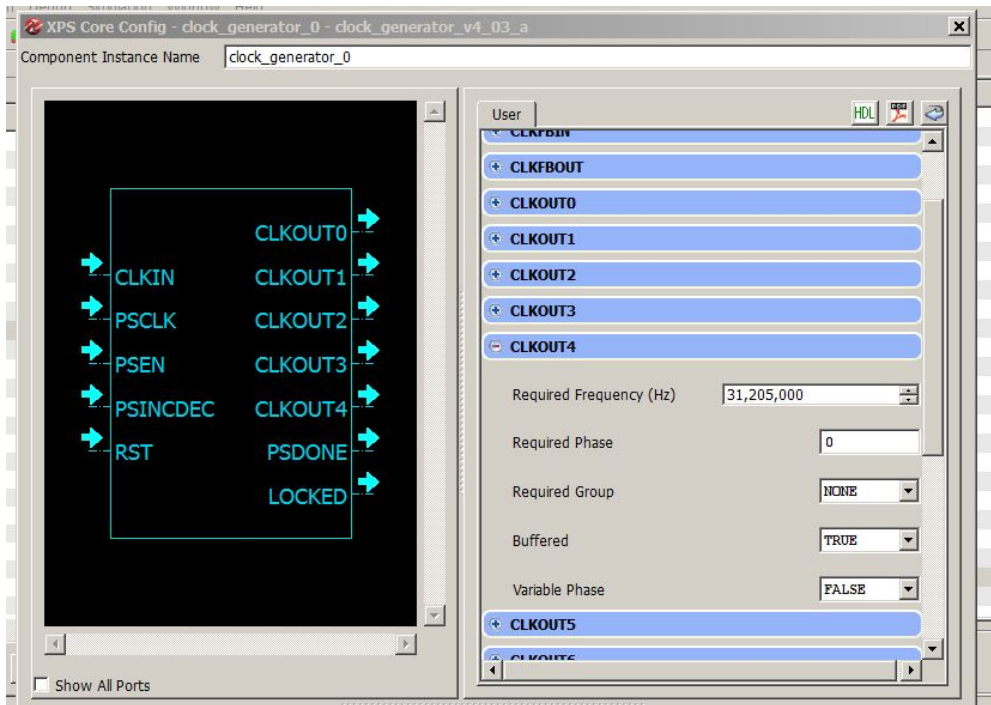
Now we can connect the MicroBlaze to the two FSL modules. The easy way to do this is to move the cursor into the pink area for one of the FSL modules. This will cause the potential connections to appear, like this:



From the legend at the bottom of the window you may note that a square indicates a Master and a circle indicates a Slave. Click the circles and squares as shown here to connect fsl\_0 and fsl\_1 to the correct MicroBlaze ports. The MicroBlaze will be the master on fsl\_0 and the slave on fs\_1:



The base system is almost complete, but the FSL IP is not connected to a clock or reset. The next step is to configure the clock\_generator IP to create the 31.250MHz clock needed for the user logic. Right-click the clock\_generator\_0 instance in the System Assembly to launch the Configure IP wizard for the clock generator. Configure the CLKOUT4 output as shown here then click "OK".



At this point the base system is fully defined and it is time to import the USER IP and connect the two FSL's , clocks, and resets to the user defined hardware.

## See Also

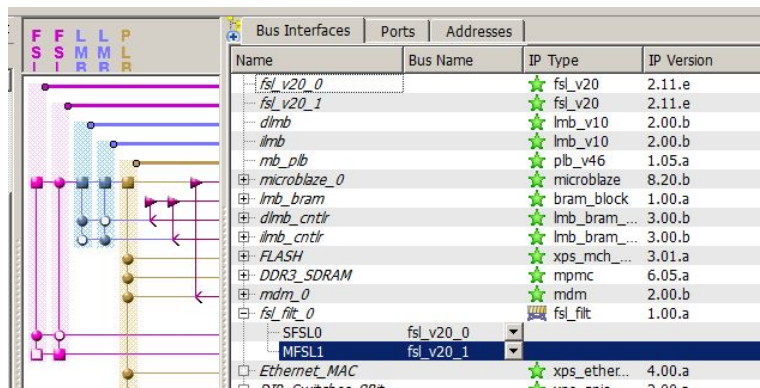
[Importing the Generated Hardware](#)

### 1.2.8 Importing the Generated Hardware

#### Complex FIR Filter Tutorial for MicroBlaze, Step 8

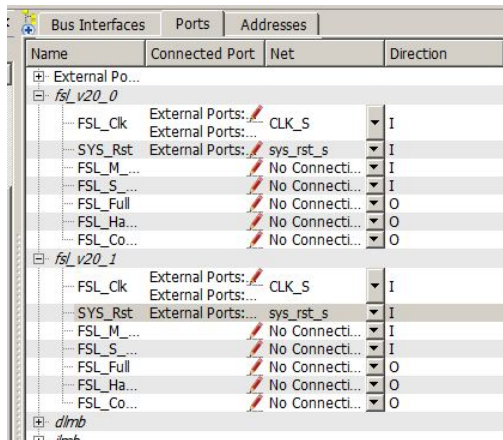
##### Import the User Defined Hardware

Return to the Bus Interfaces tab of the System Assembly View. Expand the Project Local PCores and USER modules. Then right-click the `fsl_filt` IP and select Add IP. Then click OK to put the IP into the system assembly. Expand the `fsl_filt_0` IP to display the two related FSL elements. Then as with hooking-up the MicroBlaze in the last topic, connect the user logic to the FSL as shown here. Note that you must match a slave to the MicroBlaze master and a master to the MicroBlaze slave:

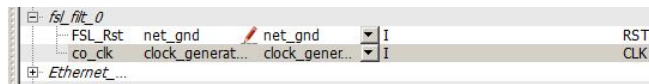


##### Connecting Clock and Reset Ports

Next, you need to configure the clock and reset signals for each **FSL** IP Core. Click on the **Ports** filter in the **System Assembly View** and expand `fsl_v20_0` and `fsl_v20_1`. For each FSL link, set **FSL\_Clk** to **CLK\_S** and set **SYS\_Rst** to **net\_gnd** as shown.



Select the **Ports** filter in the **System Assembly View** and expand **fsl\_filt\_0**. This should reveal ports **co\_clk** and **FSL\_Rst**. The **co\_clk** has to be connected to the **CLK\_S** clock that we configured in the previous steps. The **FSL\_Rst** should be tied to **net\_gnd**.



*Note: if **co\_clk** is missing from the **fsl\_filt\_0** section, then will need to return to [step 3](#) of this tutorial and specify the **Dual Clock** option in the **CoDeveloper Generate Options** page.*

Now it is time to create a new clock net and connect the 31.250MHz clock and the **co\_clk** to it.

### Specify the Addresses

Now you will need to set the addresses for each of the peripherals specified for the platform. This can be done simply by selecting the **Addresses** tab and clicking on the **Generate Addresses** button. The addresses will be assigned for you automatically:

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dlimb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SPLB	dlimb	<input type="checkbox"/>
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K	SPLB	ilmb	<input type="checkbox"/>
Ethernet_MAC	C_BASEADDR	0x81000000	0x8100FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
Push_Buttons_SBit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_Positions	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
LEDs_8Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
DIP_Switches_8Bit	C_BASEADDR	0x81460000	0x8146FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_SFP	C_BASEADDR	0x81600000	0x8160FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_FMC_LPC	C_BASEADDR	0x81620000	0x8162FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_EEPROM	C_BASEADDR	0x81640000	0x8164FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
IIC_DVI	C_BASEADDR	0x81660000	0x8166FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
xps_timer_0	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
FLASH	C_MEM0_BASE...	0x8DC00000	0x8DFFFFFF	32M	SPLB	mb_plb	<input type="checkbox"/>
DDR3_SDRAM	C_MPMC_BASE...	0x90000000	0x9FFFFFFF	256M	SPLB0	mb_plb	<input type="checkbox"/>

You have now exported all necessary hardware files from **CoDeveloper** to the **Xilinx** tools environment and have configured your new platform. The next step will be to generate FPGA bitstream. Make sure



that you save your project.

## See Also

[Generating the FPGA Bitmap](#)

### 1.2.9 Generating the FPGA Bitmap

#### Complex FIR Filter Tutorial for MicroBlaze, Step 9

At this point, if you have followed the tutorial steps carefully you have successfully:

- Generated hardware and software files from the **CoDeveloper** environment.
- Created a new **Xilinx Platform Studio** project and created a new **MicroBlaze**-based platform.
- Imported your **CoDeveloper**-generated files to the **Xilinx Platform Studio** environment.
- Connected and configured the **Impulse C** hardware process to the **MicroBlaze** processor via the **FSL** bus.

You are now ready to generate the bitmap.

First, from within Platform Studio select the button **Generate Netlist**. Note that this will take several minutes to execute depending on how fast your computer is.

Second, if there are no errors in the netlist generation step then click the **Generate Bitstream** button. Depending on your computer, this may take well over an hour to execute depending on how fast your computer is.

When the bitstream has been generated, click **Export Design** to launch the Xilinx SDK to build your software-side application.



From the pop-up box, select "Export & Launch SDK". This will package the necessary files and export them to the correct locations for the SDK (software development kit) environment. The SDK will then be automatically launched and you can begin the next topic.

## See Also

[Importing the Application Software](#)

### 1.2.10 Importing the Application Software

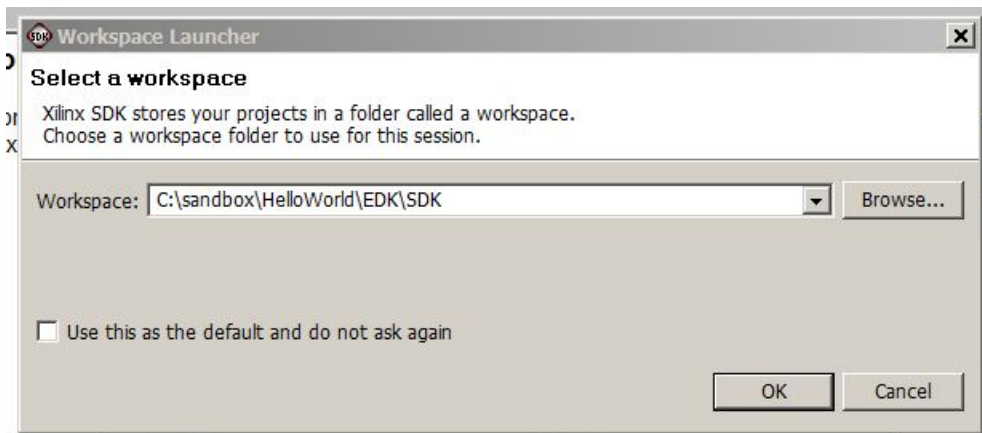
#### Complex FIR Filter Tutorial for MicroBlaze, Step 10

After clicking the Export Design button the pop-up (shown) is displayed giving you the option to perform an Export only or to Export files and start the Software Development Kit (SDK), for this tutorial please click "Export & Launch SDK".

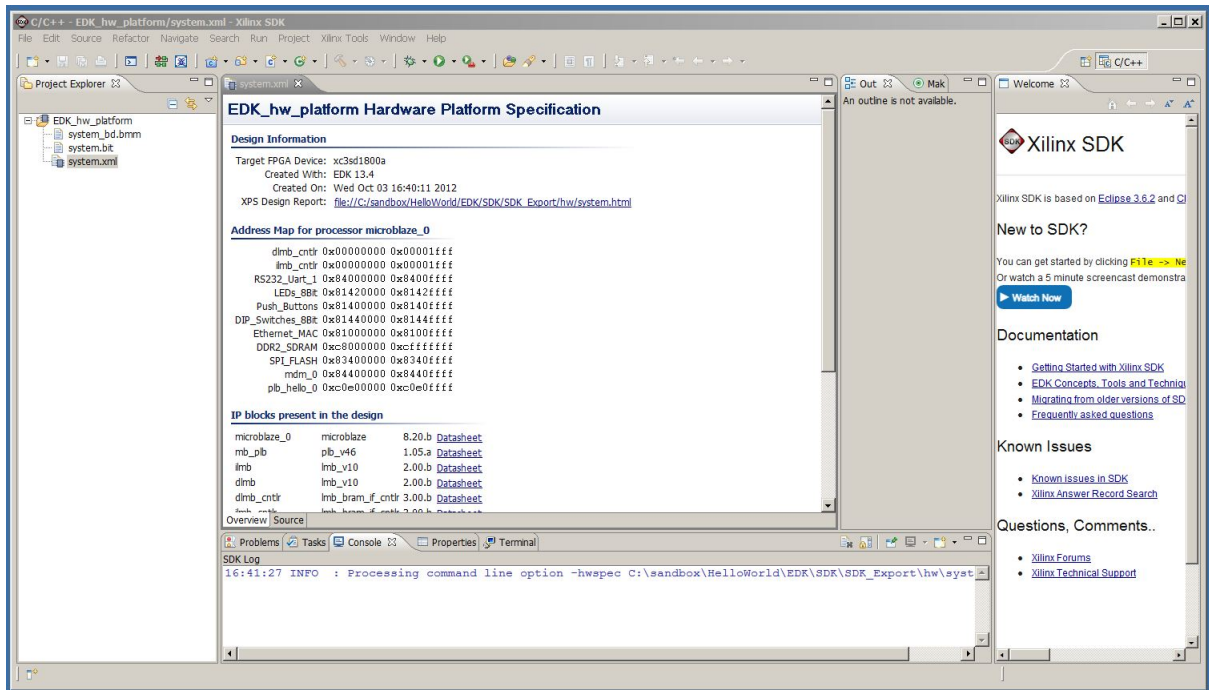
**NOTE:** Screen images in this tutorial re-use images from the Hello World tutorial. Where the project "HelloWorld" is seen, please replace it with the project Complex\_FIR\_Microblaze.



After the SDK tool starts, the following pop-up will be displayed. The export tool will have correctly filled in the project workspace path data. We do not recommend setting this as the "default" since it is not likely to be the default for other projects.

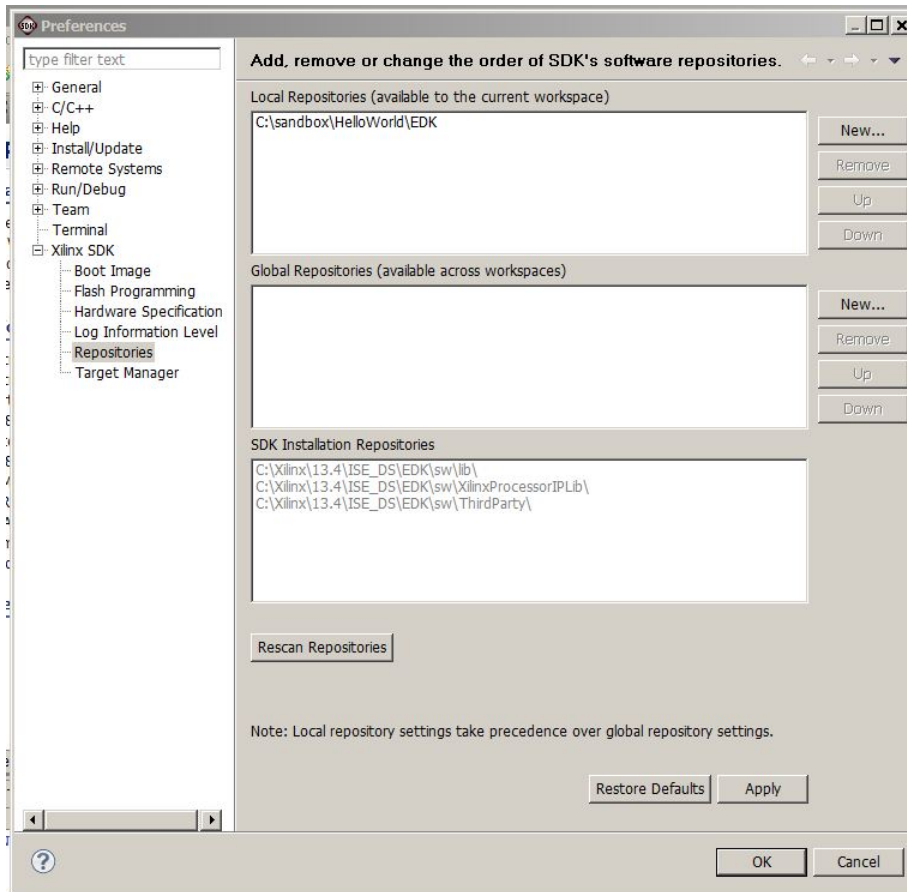


Once the SDK window is open you can close the EDK window as it will not be needed further for this tutorial. Your screen should be similar to the one shown:



## Configure SDK

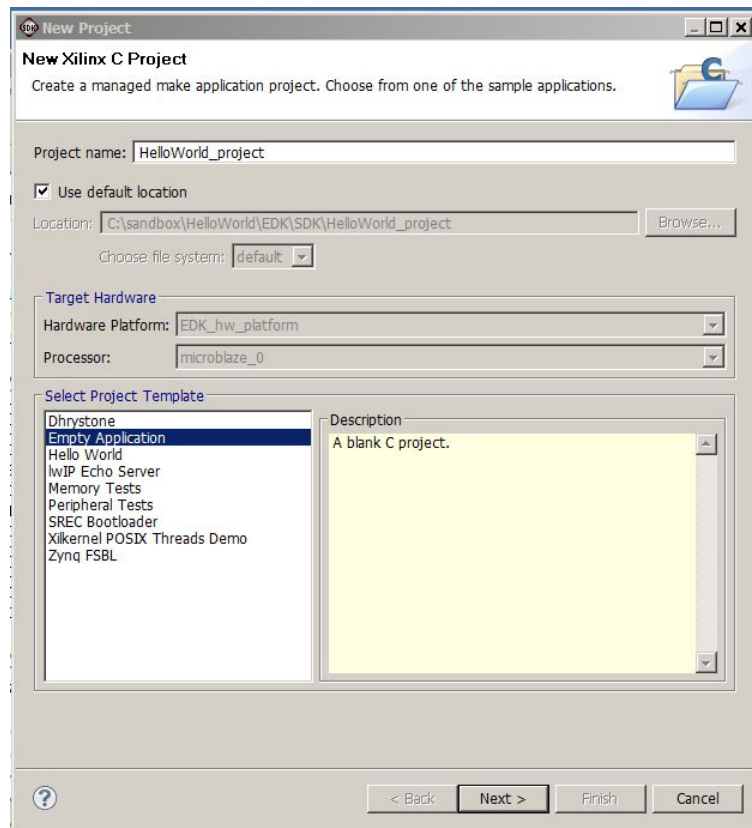
It is first necessary to configure the SDK environment to reference the EDK project directory as a "repository" in order to reference the FPGA logic driver by choosing the "Repositories" menu item from the "Xilinx Tools" menu. Click the "New" button next to the "Local Repositories" then navigate to the project's EDK directory and select it. The Preferences dialog should be similar to that shown here:



Click "Apply" and then "OK" to continue.

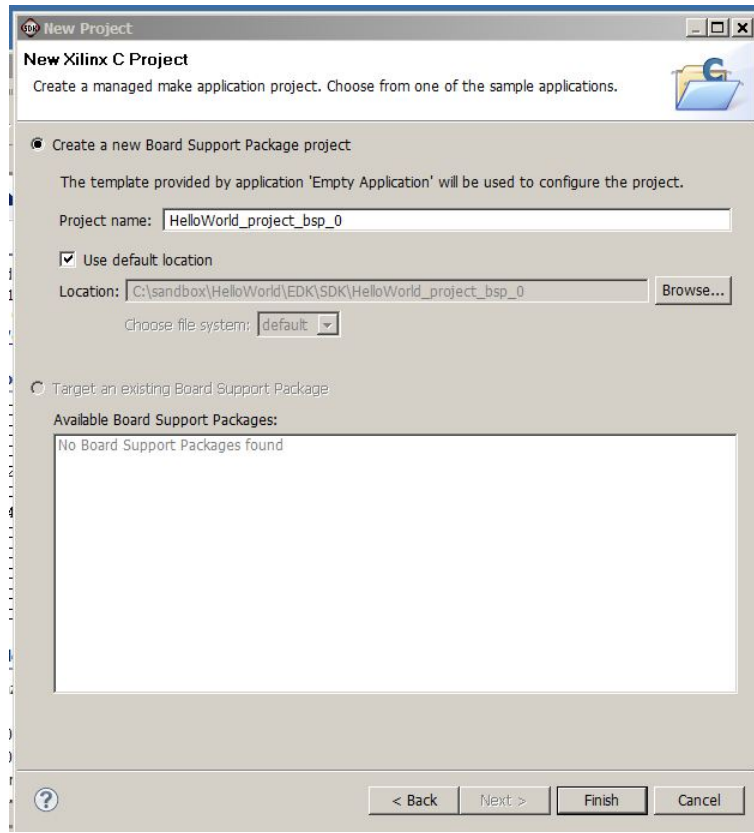
### Create the Complex\_FIR\_Microblaze Project

Choose **File -> New -> Xilinx C Project** from the File menu. Then select **Empty Application** and set the **Project name:** to `Complex_FIR_Microblaze_project`. The window will appear similar to what is shown here:



Click **Next** to continue.

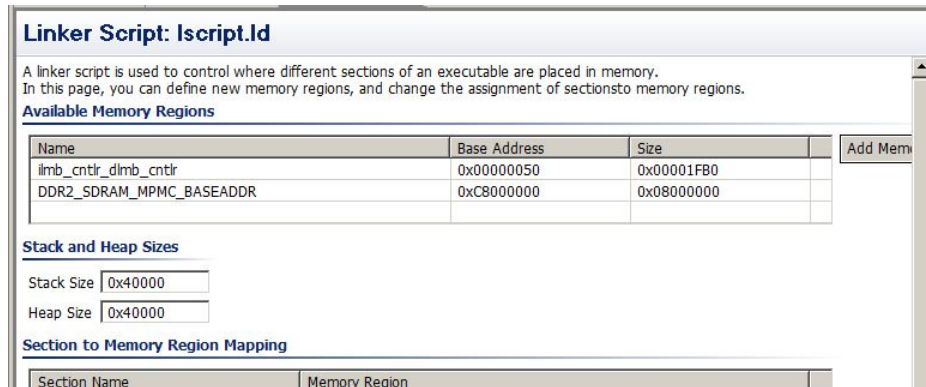
On the next dialog window set the Project name: to Complex\_FIR\_Microblaze\_project\_bsp\_0; the window will now appear similar to what is shown here:



Click **Finish** to continue.

The **Project Explorer** will now include the two new projects, `Complex_FIR_Microblaze_project` and `Complex_FIR_Microblaze_project_bap_0`. Notice that as soon as the files were read into the project that the entire project was built. This is the default behavior of SDK. If you followed this tutorial accurately there should have been no errors requiring edits.

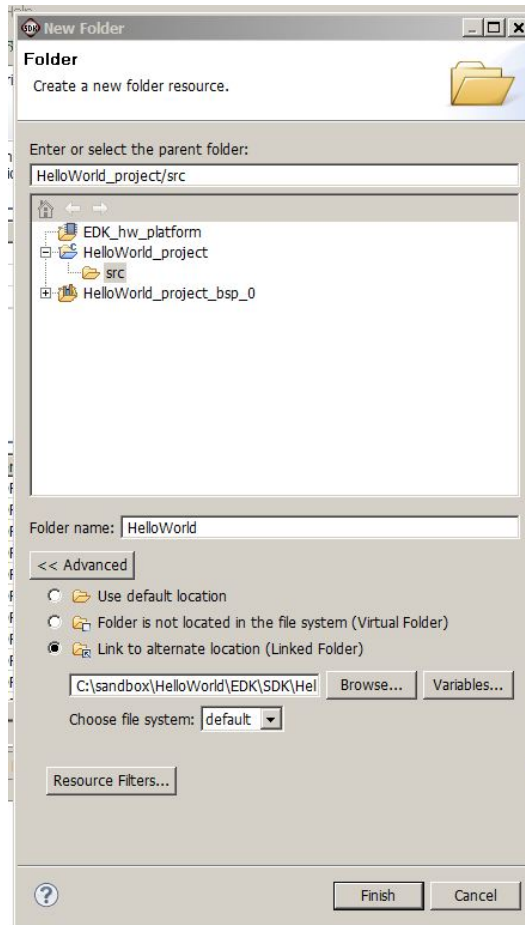
It is now necessary to increase the Stack and Heap memory sizes using the linker script. From the Project Explorer open the `lscript.ld` file, you will find it in the `Complex_FIR_Microblaze_project src` directory. Change the Stack and Heap sizes to `0x4000` as shown here:



Either click the file save icon or enter `<ctrl>s` to save the modified linker script.

Now the software files exported from CoDeveloper must be added to the project. We will do this by linking to their location as follows:

Right click the **src** directory in the HelloWorld\_project and select **New -> Folder**. Click the **Advanced** button and select **Link to alternate location (Linked Folder)**. Use the browse button to locate the Complex\_FIR\_Microblaze\EDK\SDK\Complex\_FIR\_Microblaze directory. The New Folder dialog should look similar to that shown here:



When you click Finish to continue the SDK will automatically rebuild the entire project and prepare the Complex\_FIR\_Microblaze\_project.elf file. In the next step, the BitStream and ELF files will be downloaded into the FPGA.

The Export Design process (previous topic) created an SDK directory and populated it with files needed to add the software to your project. One interesting file is the "system.html" file that is a browsable representation of the overall system and is useful in visualizing and understanding the interactions of the system.

If you accurately followed the tutorial steps the export to SDK process will be fully automated and result in a complete software application with all source files, include files, and libraries correctly setup.

After the SDK opens you will see a number of build activities scrolling in the Console window. It is NOT UNEXPECTED to have the process end with two compiler errors. The first error is an undefined

symbol error and the second error is just a notice that a prior build step failed.

The undefined symbol error will be in the file `Filter_sw.c` and is an artifact of version differences in the Xilinx tool flow. To correct the error search for the text string "XPAR\_XPS\_TIMER\_1\_DEVICE\_ID" (without the quotes) and change the "1" to "0" (zero). When you click the file save button in the toolbar the project will be automatically recompiled successfully.

You can edit other source files and on each file save the project will be automatically updated and incrementally recompiled.

Next, you will run and debug the application using SDK.

## See Also

[Running the Application](#)

## 1.2.11 Running the Application

### Complex FIR Filter Tutorial for MicroBlaze, Step 11

#### Setting up Terminal Window and Connecting Cables

Open preferred terminal emulation program. Use the same communication settings you chose when defining the peripheral in **Base System Builder (15200 baud, 8-N-1)**. Turn off **flow control**, if available. These settings were the defaults used.

Connect the serial port of your development machine to the **RS232** interface on your development board. Make sure the download (**JTAG**) cables are connected on the development board. Also ensure that the board is configured to be programmed. Turn on the power to the board. Refer to your development board documentation to correctly configure the board for programming.

Use the **Xilinx Tools -> Program FPGA** menu option to load the bit file into the FPGA. All necessary files will be automatically selected and downloaded. You can change the download files if necessary.

#### Running Application from SDK

Now let's run the application on the development board.

Select menu **Run -> Debug Configurations** select the desired debug configuration from the menu and click **Debug**.

A **Cygwin** bash shell will come up. It runs a script, connecting to the **MicroBlaze** processor and the debugger inside the FPGA. We can learn the base address of the **DDR2\_SDRAM** is **0x88000000**.

Now watch your terminal window again. You should see the messages generated by the software process indicating that the test data has been successfully filtered. The execution with hardware acceleration is **122** times faster than software only running on **MicroBlaze** microprocessor.

Congratulations! You have successfully completed this tutorial and run the generated hardware on the



development board.

### See Also

[Tutorial 1: Hello World on the MicroBlaze platform](#)