Toyon Research Corporation

Lab 0: Blink LEDs

Chilipepper Tutorial Projects

Version 1.2
12/11/2013

Table of Contents

Lab 0: Blink LEDs

Introduction

This lab is intended to guide you through the process of creating a simple embedded system on the ZED Board. We will cover each step of the process including creating a PCore using MathWorks HDL Coder with MATLAB code entry, exporting your PCore into Xilinx Platform Studio, and implementing the core in a Xilinx FPGA. To guide you through the process, this lab will teach you how to blink LEDs on the ZED board, using this tool flow.

You should note that software design with MATLAB HDL Coder requires a specific coding style. In this lab, we have provided the demo code for you to use and therefore will not discuss the coding style. Instead, this lab will focus on showing you the workflow.

This lab is created using:

- MATLAB 2013b
- Xilinx ISE Design Suite 14.7
- Windows 7, 64-bit

Procedure

This lab is organized into a series of steps, each including general instructions and supplementary steps, allowing you to take advantage of the lab according to your experience level.

This lab consists of the following basic steps:

- Generate HDL code from a MATLAB Algorithm
- Create a project in EDK
- Convert the HDL design to a PCore
- Interface the core with the MicroBlaze processor
- Building the bitfile and loading it onto the ZED board

Objectives

After completing this lab, you will be able to:

- Translate MATLAB code to HDL using HDL Coder
- Import the HDL Design into EDK as a PCore
- Integrate the resultant PCore in Xilinx EDK
- Load the created bitfile into the ZED board

Generate HDL Code Step 1

This section provides a step by step guide for the configuration process of HDL Coder, as well as a great introduction to the functionality of the tool.

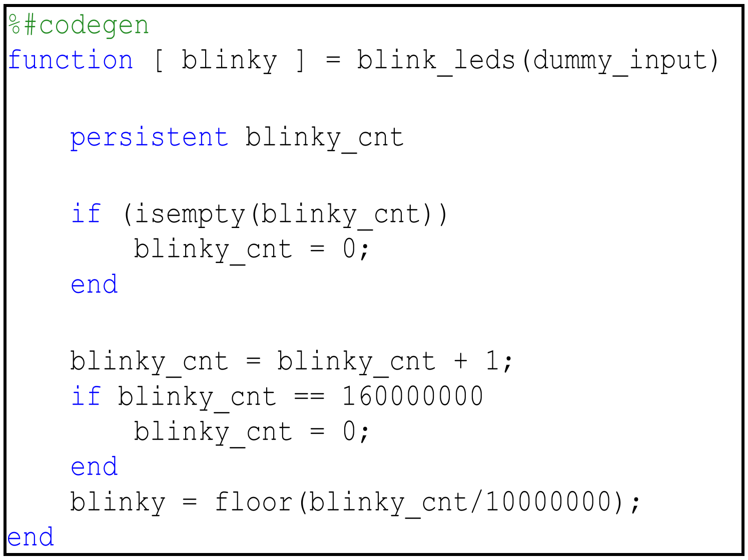
1.1 Getting familiar with HDL Coder

For users new to HDL Coder, reading The *MathWorks HDL Coder* – *Getting Started Guide*[1] for the installation and set up process of HDL Coder is highly recommended. In particular it would be helpful to read through the "HDL Code Generation from a MATLAB Algorithm" (pgs. 2-2 – 2-17) section of the guide.

1.2 MATLAB Function

Your MATLAB function is the source code that will be synthesized into hardware. The function describes the operations in each clock cycle, and it should be noted that the function processes data on a sample-by-sample basis.

To blink the LEDs, we have created a function whose output `blinky` will be connected to the LEDs on the board. This function will increment a counter every clock cycle, and at the interval of ten million cycles will increment the value output to the LEDs. Since the core will be clocked at 40 MHz, the core can effectively change the value at a rate of 20 MHz, which would give the LEDs a new value every half second. The value sent to the LEDs will continue to increase up to a maximum of fifteen. The function is shown in Figure 1-1 below.



```
%#codegen
function [ blinky ] = blink_leds(dummy_input)

    persistent blinky_cnt

    if (isempty(blinky_cnt))
        blinky_cnt = 0;
    end

    blinky_cnt = blinky_cnt + 1;
    if blinky_cnt == 160000000
        blinky_cnt = 0;
    end
    blinky = floor(blinky_cnt/10000000);
end
```

Figure 1-1: MATLAB function for blinking LEDs

1. Create a directory for the project under C:\QPSK_Projects\Project_0.

[1] Can be found at http://www.mathworks.cn/help/pdf_doc/hdlcoder/hdlcoder_gs.pdf

2. Create a folder inside this directory called MATLAB.

3. **Save** this function as `blink_leds.m` inside the MATLAB folder.

You do not have to use the same directory created in this lab, however it is very helpful to have a directory structure which is consistent throughout the labs. In addition, if you use a directory which has spaces, you will not be able to save your EDK project to that directory in Step 3 of this guide. It is therefore recommended that you use underscores instead of spaces for all directory structures in this lab.

1.3 MATLAB Test Bench

The test bench is used to verify the operation of your function. It is used in testing only, meaning none of it will be compiled into hardware. It allows you to provide stimuli to the function (signals, parameters, constants, etc), and analyze the outputs. The test bench used is the MATLAB script, `blink_leds_tb` and the code used is shown in Figure 1-2.

```
for i1 = 1:200
    [blinky_out(i1)] = blink_leds(1);
end

plot(blinky_out,'o')
```

Figure 1-2: MATLAB code for HDL test bench script

Note

The test bench file for this lab requires too many cycles to verify the output to the LEDs. To correct this, you can run the test bench after modifying `blink_leds.m` to wait for only 10 cycles between value changes, rather than 10 million. Don't forget to correct this change however before proceeding to the next steps.

1. Create a new **MATLAB script** with the contents of Figure 1-2.

2. **Save** this script as `blink_leds_tb.m` inside the MATLAB folder.

3. **Run** this script in MATLAB to test the function.

4. Once you have verified that your algorithm is correct, proceed to the next step of the lab.

1.4 Creating the HDL Coder Project

If you have read the MathWorks *Getting Started Guide*, then this section will be fairly straightforward.

1. Under the **Apps** tab, search for MATLAB HDL Coder. You may have click the arrow on the far right ▾ and navigate to **Code Generation**.

2. Create a name for your HDL Coder project. For consistency, we have named the project blink_leds and placed it within the same directory as our MATLAB files. It is important to note that the location of the project cannot contain spaces; use underscores instead.

3. In the **MATLAB Function** section, click **Add MATLAB function**, select `blink_leds.m` and click **Open.**

4. In the **MATLAB Test Bench** section, click **Add files**, select `blink_leds_tb.m` and click **Open.**

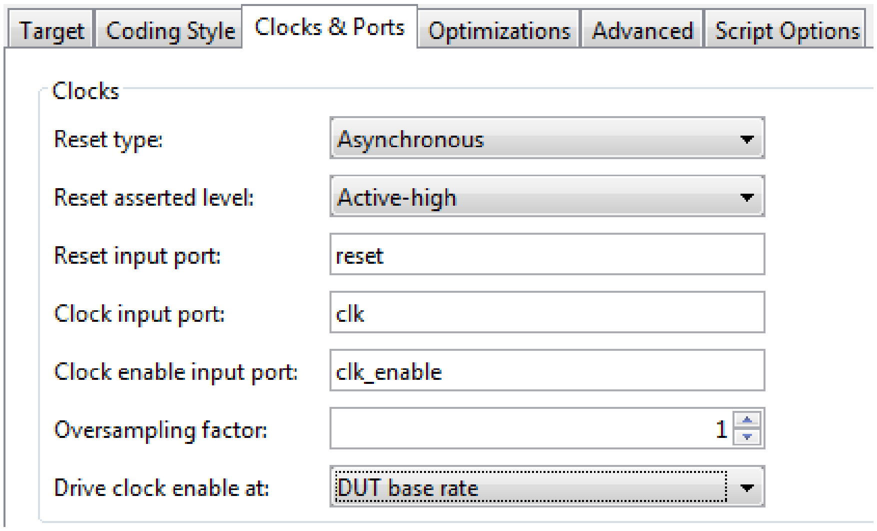
Your function and test bench are now added to the project.

1.5 Create PCore with HDL Coder

1. From the HDL Coder project window, click on **Workflow Advisor**. The left pane shows the tasks in each section of the code generation process. Refer to the MathWorks *Getting Started Guide* for information on each task. In this lab we will simply walk through what is required for each step.

2. Select **HDL Code Generation** in the left hand menu and click on the **Clocks & Ports** tab. Configure the settings as shown in Figure 1-3 below.

   a. Set **Drive clock enable at** to **DUT base rate**



Reset type: Asynchronous

Reset asserted level: Active-high

Reset input port: reset

Clock input port: clk

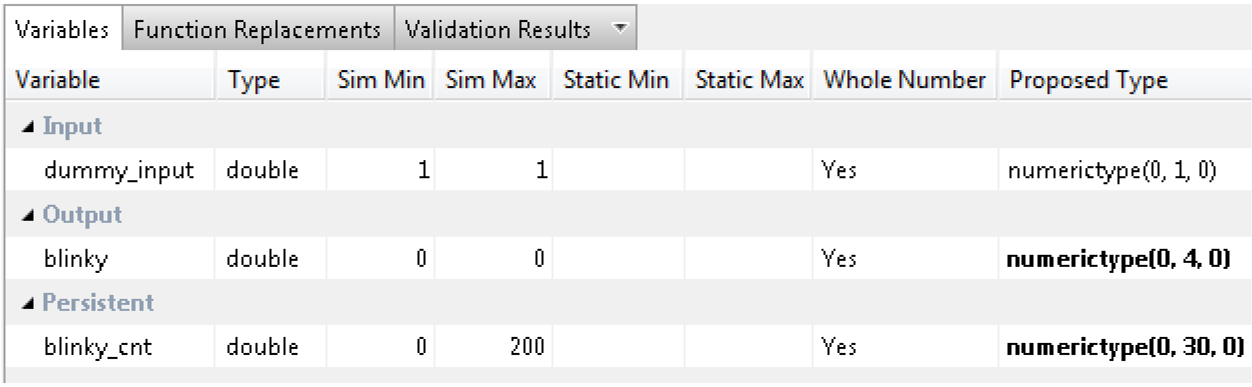
Clock enable input port: clk_enable

Oversampling factor: 1

Drive clock enable at: DUT base rate

Figure 1-3: Clocks & Ports configuration settings

3. Right-click on **Fixed-Point Conversion,** and select **Run to Selected Task**. This step is designed to automate the size of variables in your HDL code. For more information on this process refer to the MathWorks *Getting Started Guide*. For this Lab, the values of your "Type" column should resemble Figure 1-4. If they do not, please change them to match what is shown below.

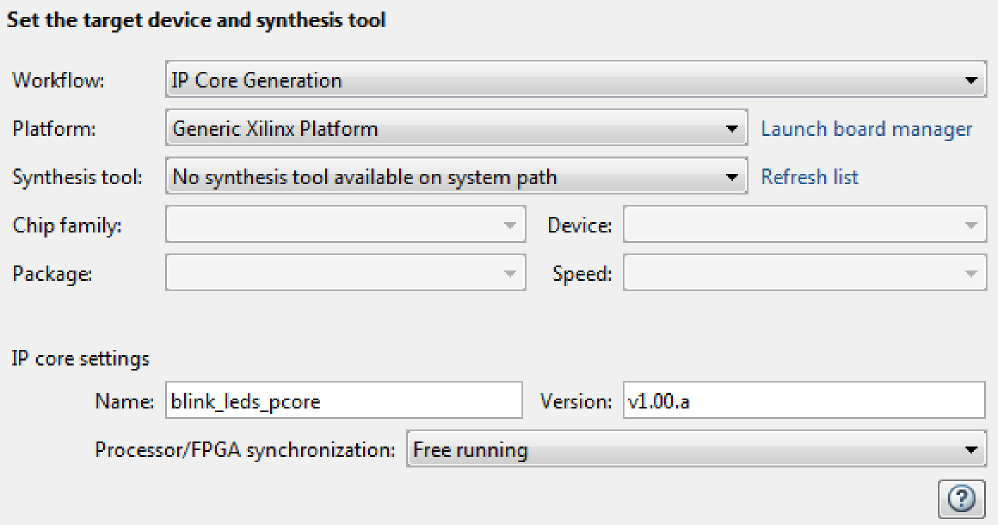


Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
◢ Input							
dummy_input	double	1	1			Yes	numerictype(0, 1, 0)
◢ Output							
blinky	double	0	0			Yes	**numerictype(0, 4, 0)**
◢ Persistent							
blinky_cnt	double	0	200			Yes	**numerictype(0, 30, 0)**

Figure 1-4: Variable types for blink_leds MATLAB function

4. Once you have corrected the Type setting for all your variables, right-click **Fixed-Point Conversion,** and select **Run This Task**. This will create HDL code for your modified variable types.

5. Next, click **Select Code Generation Target**, and modify the settings such that they match Figure 1-5 below. Your Workflow settings should be left at their default values. Also, rename your PCore to blink_leds_pcore.



Set the target device and synthesis tool

Workflow: IP Core Generation

Platform: Generic Xilinx Platform

Synthesis tool: No synthesis tool available on system path

Chip family:

Device:

Package:

Speed:

IP core settings

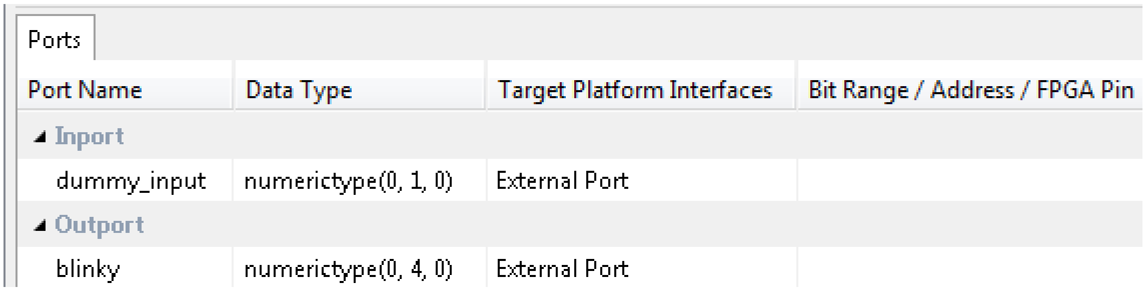
Name: blink_leds_pcore

Version: v1.00.a

Processor/FPGA synchronization: Free running

Figure 1-5: Code Generation Settings for HDL Coder Project

6. Click on Set Target Interface and you will be shown the ports used within this core. We want to send the `blinky` output port to the LEDs and the input port should be left unconnected, therefore set both ports as external ports.



Ports

Port Name	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
◢ Inport			
dummy_input	numerictype(0, 1, 0)	External Port	
◢ Outport			
blinky	numerictype(0, 4, 0)	External Port	

1-6: Interface Ports for blink_leds PCore

7. The lat step on the HDL coder project is to generate the code. Right-click on HDL Code Generation and select run this task. Once you see the green check box appear next to that option, your code is created and ready for import into an EDK project.

1.6 Troubleshooting

1. If during **Code Generation**, you receive the following errors:

```
Error: failed to run post code generation tasks:
hdlcoder:matlabhdlcoder:dutbaserateforXSG In order to work
with Xilinx System Generator for DSP, "Drive clock enable
at" must be set to "Dut base rate".
```

   a. Check the **Clocks & Ports** settings. The **Drive clock enable at** should be set to **DUT base rate**

```
Expecting top level function name ('blink_leds') to match the design filename
blink_leds.
```

   b. Be sure that your MATLAB function and test bench files **do not contain spaces**; use underscores instead.

2. If you receive a message saying either your **MATLAB function file** or your **test bench file** cannot be found, be sure that the path you saved your files to **does not contain spaces** and is the same directory as your MATLAB and project files.

You have now generated the HDL code for your design from your MATLAB algorithm.

Configure Cores and Export Design Step 2

Before exporting the design to Xilinx EDK, we must create an XPS project. Xilinx Platform Studio (XPS) is a part of EDK (the embedded development kit). XPS is used here for integration of the embedded processor and IP cores within the FPGA.

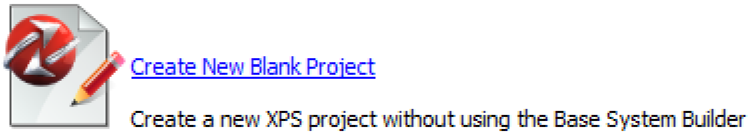
2.1 Create an XPS Project

This section will walk you through how to use Base System Builder (BSB) to create our XPS project.

1. Open XPS:

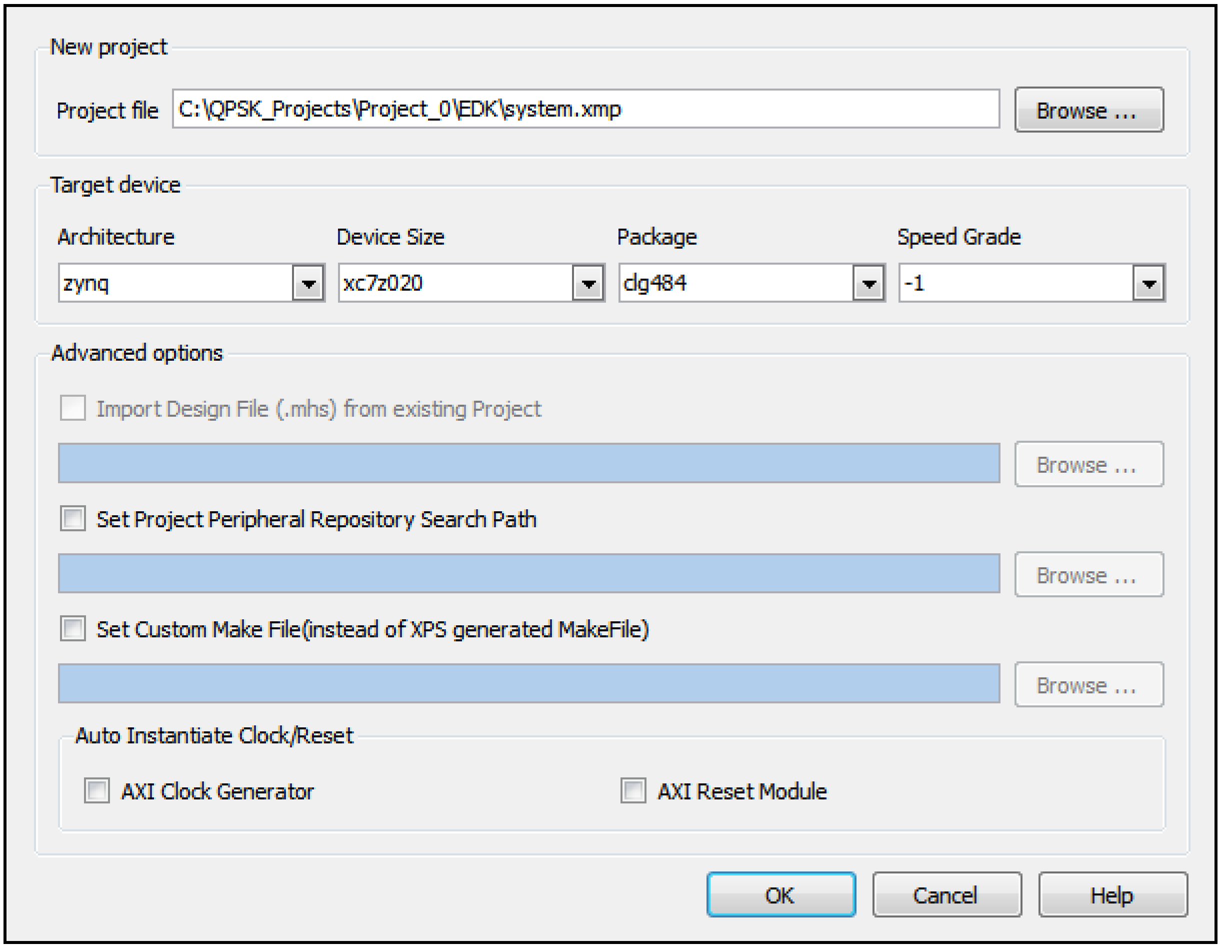
   Start menu → All programs → Xilinx Design Tools → ISE Design Suite 14.7 → EDK → Xilinx Platform Studio.

2. Click **Create New Blank Project**.



Create New Blank Project

Create a new XPS project without using the Base System Builder



New project

Project file C:\QPSK_Projects\Project_0\EDK\system.xmp Browse ...

Target device

Architecture	Device Size	Package	Speed Grade
zynq	xc7z020	clg484	-1

Advanced options

☐ Import Design File (.mhs) from existing Project

Browse ...

☐ Set Project Peripheral Repository Search Path

Browse ...

☐ Set Custom Make File(instead of XPS generated MakeFile)

Browse ...

Auto Instantiate Clock/Reset

☐ AXI Clock Generator ☐ AXI Reset Module

Figure 2-1: New XPS project settings

3. Next to **Project File,** click **Browse.**

4. Create a folder to contain the EDK project. It would be convenient to place this folder in the same directory as your MATLAB files and HDL Coder project, and name it something like **EDK**. Double-click to enter the folder.

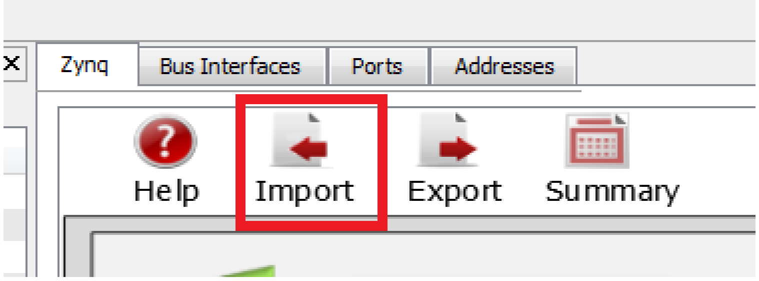
5. Keep the default name "system", and press **Save.**

6. Select Architecture/Device Size/Package/Speed Grade as shown in Figure 2-1 above.

7. Uncheck the boxes for AXI Clock Generator and AXI Reset Module.

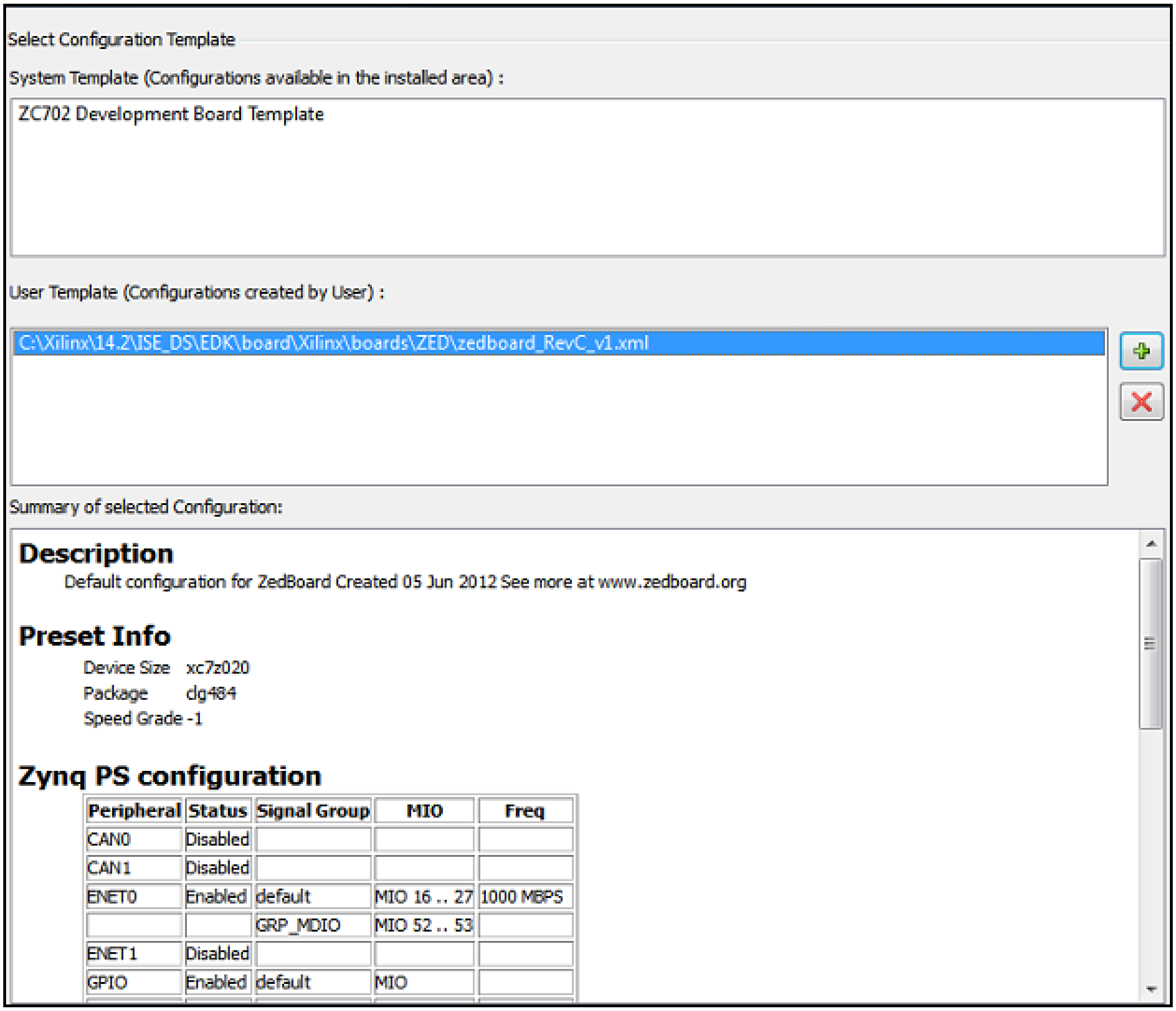
8. Press **OK.**

9. You will now need to import the Zynq board definition. On the Zynq tab select **Import**.



10. Now select the User Template as shown in Figure 2-2 below and click **OK** and then **YES** to accept the configuration.

Note	If your User template is not listed on the Import Configurations screen, you may have to manually select the path where your template is stored. By default the Template should be located in the Path C:\Xilinx\14.4\ISE_DS\EDK\board\Xilinx\boards\ZED. If you cannot find it there or the path doesn't exist, you can download the file at zedboard.com/misc/files/zedboard_RevC_v1.xml. After downloading the file, save it to the default directory given above and select it by clicking the plus symbol under User Template in the Import Configuration screen.



Select Configuration Template

System Template (Configurations available in the installed area) :

ZC702 Development Board Template

User Template (Configurations created by User) :

C:\Xilinx\14.2\ISE_DS\EDK\board\Xilinx\boards\ZED\zedboard_RevC_v1.xml

Summary of selected Configuration:

Description

Default configuration for ZedBoard Created 05 Jun 2012 See more at www.zedboard.org

Preset Info

Device Size xc7z020
Package clg484
Speed Grade -1

Zynq PS configuration

Peripheral	Status	Signal Group	MIO	Freq
CAN0	Disabled			
CAN1	Disabled			
ENET0	Enabled	default	MIO 16 .. 27	1000 MBPS
		GRP_MDIO	MIO 52 .. 53	
ENET1	Disabled			
GPIO	Enabled	default	MIO	

Figure 2-2: XPS Project Configuration Template

3.2 Import and add your PCore to the project

Now that your EDK Project is created, we can add the custom PCore, and configure the design.

1. First you need to place your custom PCore where your EDK project can find it. C**opy the PCore folder** called `blink_leds_pcore_v1_00_a` from the MATLAB/codegen/ipcore folder into the PCores folder of your **EDK Project**.

2. Next, from the menu bar, simply select **Project → Rescan User Repositories** to show your newly added User PCore within your EDK project.

In the **IP Catalog** tab on the left, under **Project Local PCores → User**, you should see your PCore listed as shown in Figure 2-3.

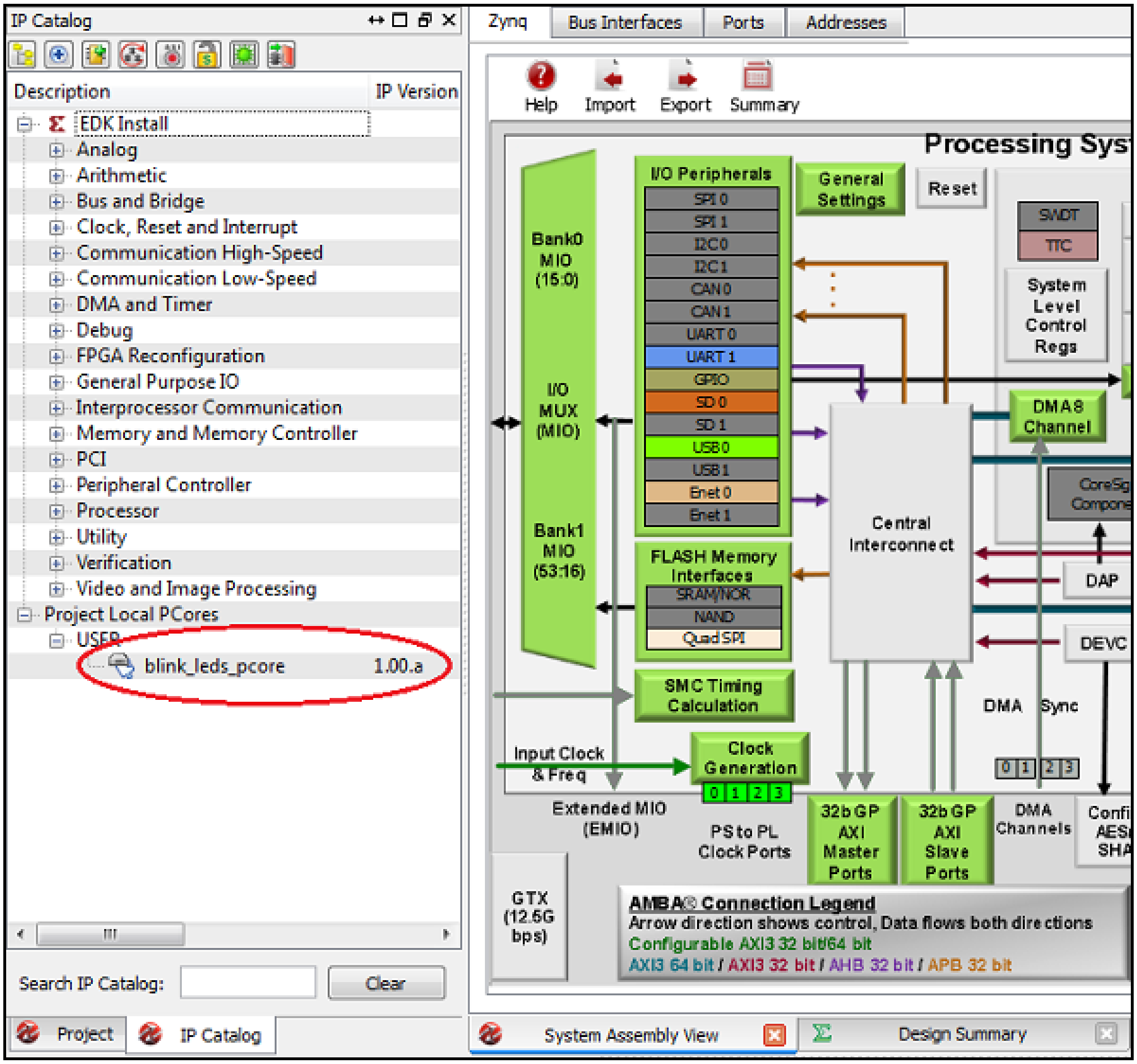


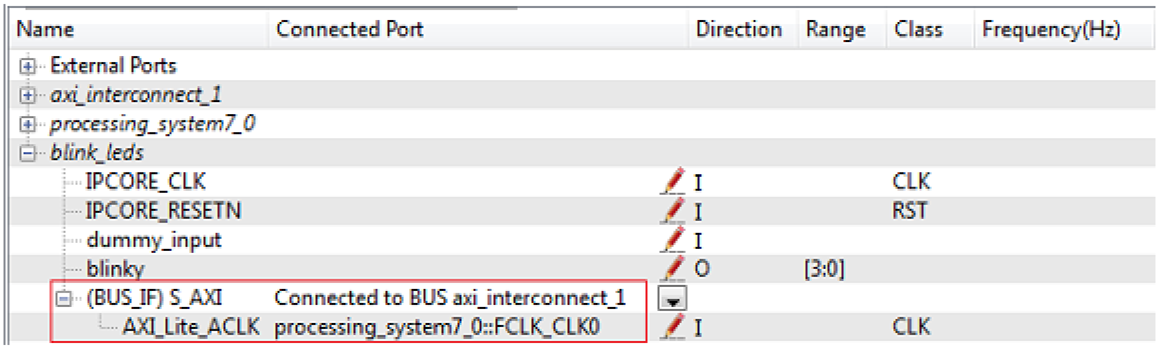
Figure 2-3: User PCore imported from HDL Coder

3. **Double-click** on your PCore and select **yes** to add it to the design.

4. On the Core Configuration screen, change the name of the core from `blink_leds_pcore_0` to `blink_leds`, as we will only have one instance of this core.

5. Click ok, as well as ok on the next screen that appears to accept the default Interface.

6. Verify your PCore by making sure it looks similar to Figure 2-4. In particular, be sure that your design has the BUS_IF and axi_aclk shown in the Figure.



Name	Connected Port	Direction	Range	Class	Frequency(Hz)
⊞ External Ports					
⊞ *axi_interconnect_1*					
⊞ *processing_system7_0*					
⊟ *blink_leds*					
IPCORE_CLK		I		CLK	
IPCORE_RESETN		I		RST	
dummy_input		I			
blinky		O	[3:0]		
⊟ (BUS_IF) S_AXI	Connected to BUS axi_interconnect_1				
AXI_Lite_ACLK	processing_system7_0::FCLK_CLK0	I		CLK	

2-4: Pots Interface for blink_leds PCore

3.3 Integrate your PCore into the Design

1. While in **System Assembly View**, go to the **Ports** tab and find your PCore listed. Expand your PCore to see all the inputs/outputs of the core.

2. Right-click on the signal **blinky port** and select **Make External.** This will direct the 4 bits of the blinky port to a peripheral.

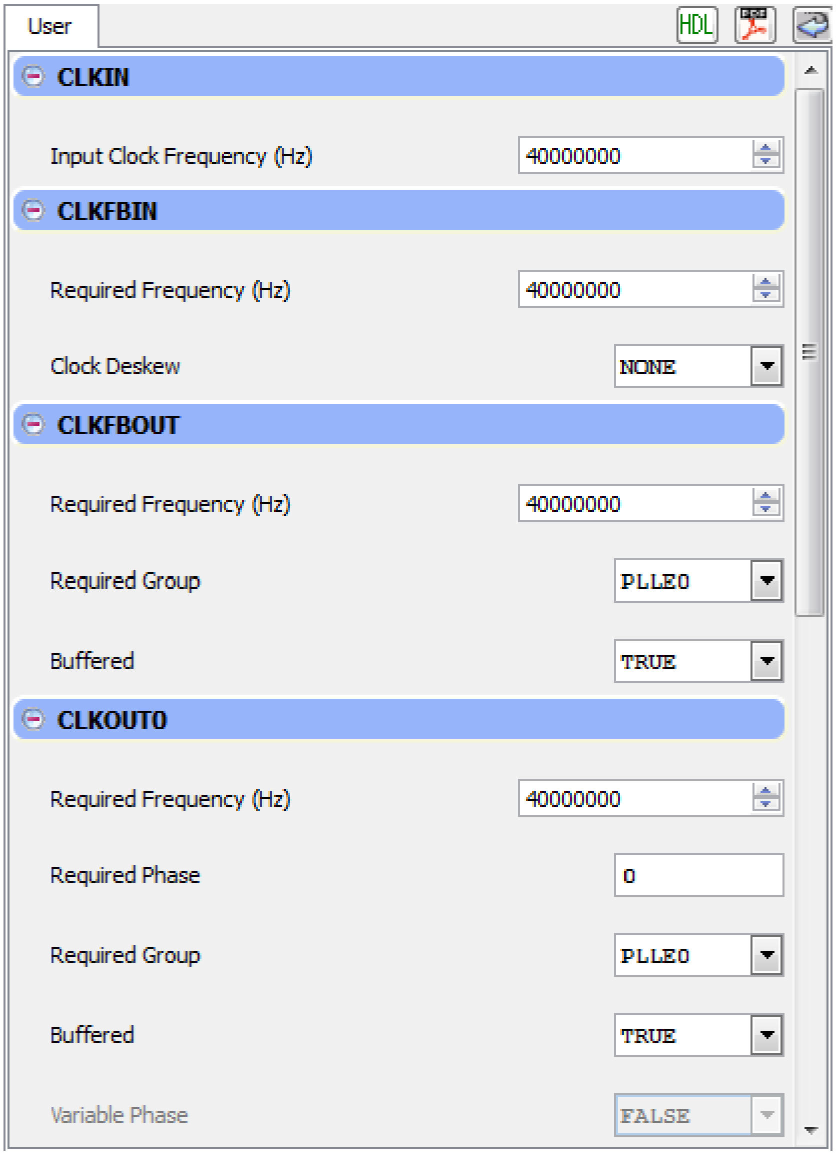
3. In the IP Catalog on the left, expand the Clock, Reset and Interrupt section. Double click the Clock Generator core to add this to your design.

4. At the Core Configuration screen, fill in the settings according to Figure 2-5 below.

    a. The Input frequency should be 40 MHz, which is the clock input from the Chilipepper board to the FPGA

    b. Both the Clock BIN and BOUT should be set to 40 MHz as well. The clock BOUT port also requires a group setting of PLLE0 and a buffered setting of true.

    c. The Clockout0 port will be used to clock our blink_leds core. This frequency should also be set to 40MHz with no phase, a group of PLLE0, and a buffered setting of true. No other settings are required for this core



CLKIN

Input Clock Frequency (Hz) 40000000

CLKFBIN

Required Frequency (Hz) 40000000

Clock Deskew NONE

CLKFBOUT

Required Frequency (Hz) 40000000

Required Group PLLE0

Buffered TRUE

CLKOUT0

Required Frequency (Hz) 40000000

Required Phase 0

Required Group PLLE0

Buffered TRUE

Variable Phase FALSE

2-5: Core Configuration for Clock Generator Core.

5. Expand your newly created clock_generator core, and connect its ports according to the following.

    a. CLKIN should be set as an external port. Set the name of this port to `clock_generator_0_pll_pin`.

    b. CLKOUT0 should be connected to the blink_leds::IPCORE_CLK.

    c. CLKFBIN should connect directly to CLKFBOUT.

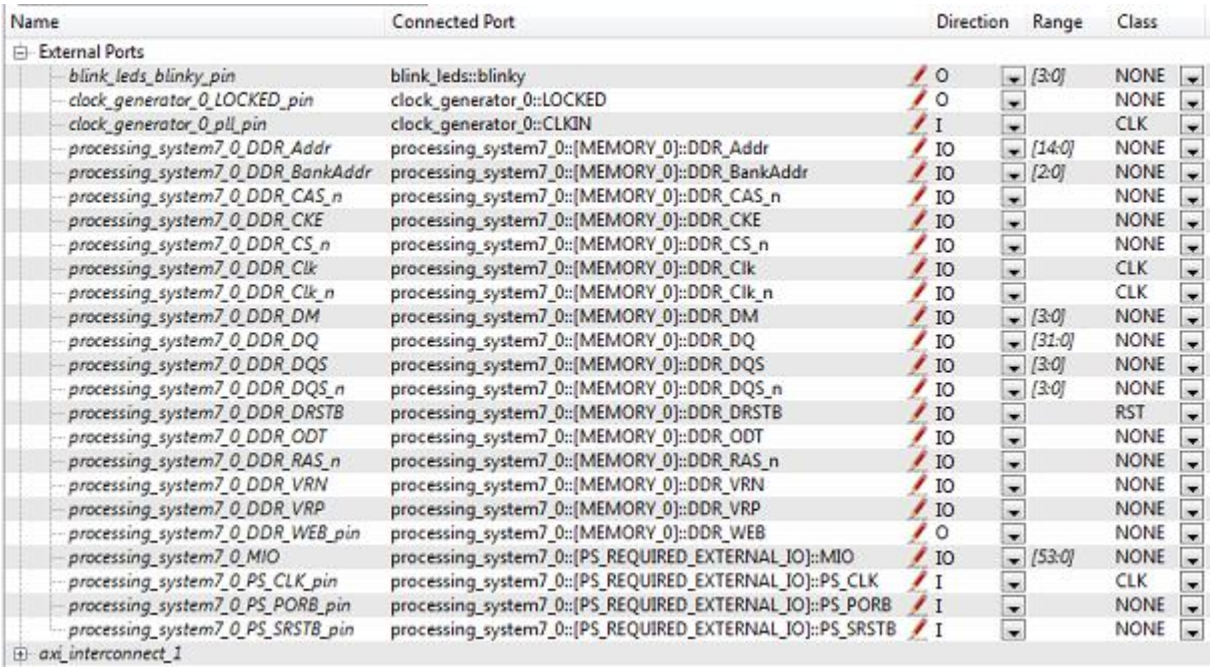
    d. The RST pin should connect to net_gnd.

e. The Locked pin should be set as an external port. The default name is fine for this pin.

6. Next, connect the IPCORE_RESETN pin of your blink_leds port to the processing_system7::FCLK_CLK0 pin.

7. The dummy_input pin can be left blank.

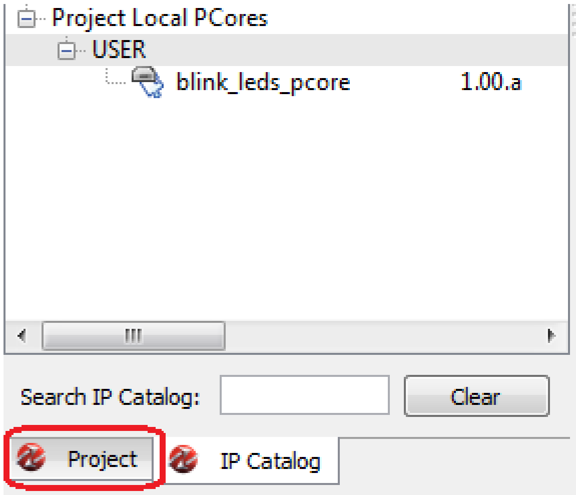
Your External ports should now resemble figure 2-6 below.



Name	Connected Port	Direction	Range	Class
⊟ External Ports				
blink_leds_blinky_pin	blink_leds::blinky	O	*[3:0]*	NONE
clock_generator_0_LOCKED_pin	clock_generator_0::LOCKED	O		NONE
clock_generator_0_pll_pin	clock_generator_0::CLKIN	I		CLK
processing_system7_0_DDR_Addr	processing_system7_0::[MEMORY_0]::DDR_Addr	IO	*[14:0]*	NONE
processing_system7_0_DDR_BankAddr	processing_system7_0::[MEMORY_0]::DDR_BankAddr	IO	*[2:0]*	NONE
processing_system7_0_DDR_CAS_n	processing_system7_0::[MEMORY_0]::DDR_CAS_n	IO		NONE
processing_system7_0_DDR_CKE	processing_system7_0::[MEMORY_0]::DDR_CKE	IO		NONE
processing_system7_0_DDR_CS_n	processing_system7_0::[MEMORY_0]::DDR_CS_n	IO		NONE
processing_system7_0_DDR_Clk	processing_system7_0::[MEMORY_0]::DDR_Clk	IO		CLK
processing_system7_0_DDR_Clk_n	processing_system7_0::[MEMORY_0]::DDR_Clk_n	IO		CLK
processing_system7_0_DDR_DM	processing_system7_0::[MEMORY_0]::DDR_DM	IO	*[3:0]*	NONE
processing_system7_0_DDR_DQ	processing_system7_0::[MEMORY_0]::DDR_DQ	IO	*[31:0]*	NONE
processing_system7_0_DDR_DQS	processing_system7_0::[MEMORY_0]::DDR_DQS	IO	*[3:0]*	NONE
processing_system7_0_DDR_DQS_n	processing_system7_0::[MEMORY_0]::DDR_DQS_n	IO	*[3:0]*	NONE
processing_system7_0_DDR_DRSTB	processing_system7_0::[MEMORY_0]::DDR_DRSTB	IO		RST
processing_system7_0_DDR_ODT	processing_system7_0::[MEMORY_0]::DDR_ODT	IO		NONE
processing_system7_0_DDR_RAS_n	processing_system7_0::[MEMORY_0]::DDR_RAS_n	IO		NONE
processing_system7_0_DDR_VRN	processing_system7_0::[MEMORY_0]::DDR_VRN	IO		NONE
processing_system7_0_DDR_VRP	processing_system7_0::[MEMORY_0]::DDR_VRP	IO		NONE
processing_system7_0_DDR_WEB_pin	processing_system7_0::[MEMORY_0]::DDR_WEB	O		NONE
processing_system7_0_MIO	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::MIO	IO	*[53:0]*	NONE
processing_system7_0_PS_CLK_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_CLK	I		CLK
processing_system7_0_PS_PORB_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_PORB	I		NONE
processing_system7_0_PS_SRSTB_pin	processing_system7_0::[PS_REQUIRED_EXTERNAL_IO]::PS_SRSTB	I		NONE
⊞ *axi_interconnect_1*				

2-6: External ports for the Blink_LEDs EDK project

8. Open the **Project** tab.



9. Double-click on the **UCF File: data\system.ucf** from this panel, to open the constraints file. Fill in your variable names and pin locations for all pins assignments using Figure 2-7 as a guideline.

```
#######################################################################
# PL clocks and reset
######################################################################
NET clock_generator_0_pll_pin         LOC = D18  | IOSTANDARD = LVCMOS25;
NET clock_generator_0_pll_pin         TNM_NET = clock_generator_0_pll;
TIMESPEC TS_clock_generator_0_pll =   PERIOD clock_generator_0_pll 40.000 MHz;
######################################################################
# LEDs
######################################################################
NET clock_generator_0_LOCKED_pin      LOC = T22   |  IOSTANDARD = LVCMOS33; # “LD0”
NET blink_leds_blinky_pin[0]          LOC = T21   |  IOSTANDARD = LVCMOS33; # “LD1”
NET blink_leds_blinky_pin[1]          LOC = U22   |  IOSTANDARD = LVCMOS33; # “LD2”
NET blink_leds_blinky_pin[2]          LOC = U21   |  IOSTANDARD = LVCMOS33; # “LD3”
NET blink_leds_blinky_pin[3]          LOC = V22   |  IOSTANDARD = LVCMOS33; # “LD4”
```

2-7: Pin assignments for Blink LEDs

10. Be sure to save and close the **UCF** file.

3.4 Export your design

1. It is optional, but recommended that you use **Project →Design Rule Check** to quickly check for errors. Proceed when no errors are found within your design.

2. In the Navigator pane on the left, click **Export Design**. Click **Export & Launch SDK**. This process may take a while.

Once the process finishes, you'll be ready to proceed to the next step to finalize your design.

Create software project Step 3

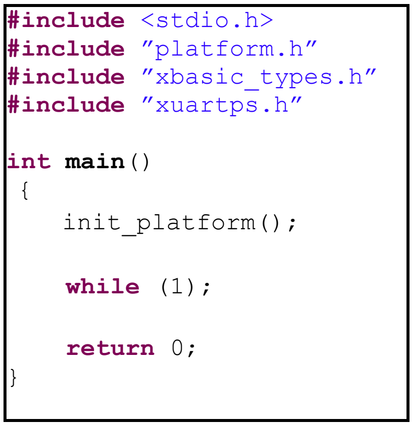
Once the design is compiled and exported, you'll be greeted with a screen asking you where you would like to store your software project. It is very helpful to create the workspace folder in the same directory as your MATLAB and EDK folders. Doing this will keep all relevant files in the same location.

3.1 Creating a new C Project

This section will show you how to create a C program to run your blink LEDs design. Since our blinking was programmed in hardware, all we need to do in software is initialize the FPGA hardware.

Note	It would be helpful if you have completed the Embedded System Design tutorial in the *ZedBoard AP SoC Concepts Tools and Techniques Guide*.

1. Select **File → New → Application Project**.
2. Name the project "blinky" or something similar and leave the other settings at their defaults. Click next.
3. On the next screen, be sure to select **Hello World** from the list of Available Templates.
4. Click **Finish**. You should now see your tone project folder, as well as a **board support package** (bsp) folder.
5. If you navigate into the blinky project folder, and into the src folder, you should see a `helloworld.c` file. Feel free to rename this file to main.c or something more appropriate.
6. **Double click** the file to open it and **replace** all of its contents with the code in Figure 3-1.



```
#include <stdio.h>
#include ”platform.h”
#include ”xbasic_types.h”
#include ”xuartps.h”

int main()
 {
    init_platform();

    while (1);

    return 0;
}
```

Figure 3-1: Code outline for SDK project

3.2 Programming the Board

Once your program is written and compiled you are ready to test the design! This is done by programming the FPGA with your hardware descriptions defined in the bit file generated in EDK, and running your software on top of this design.

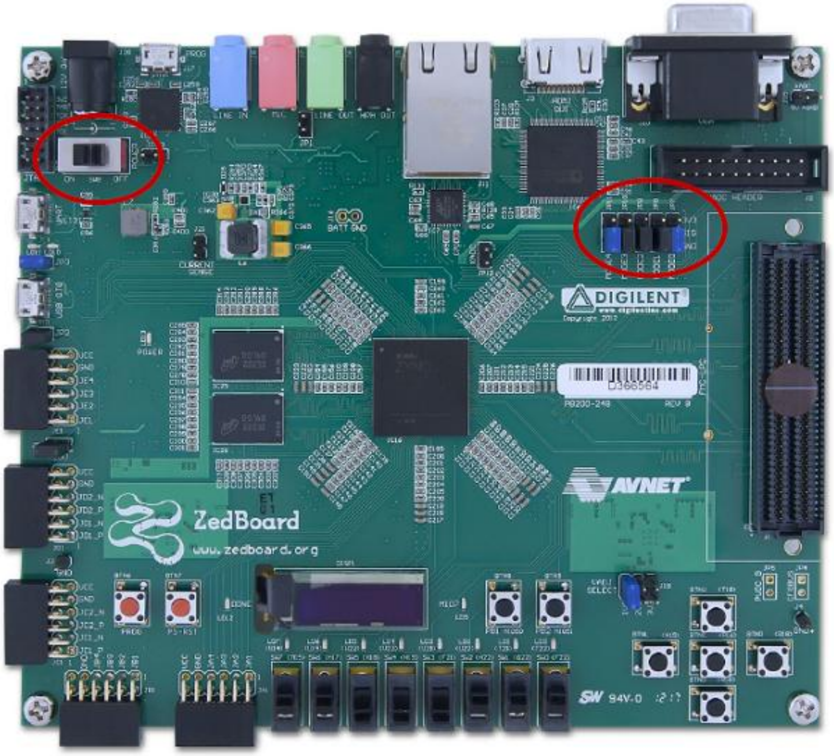


Figure 3-2. Jumper Configuration for the ZedBoard. Image is from Zynq ZedBoard Concepts, Tools, and Techniques[2]

Note	For the setup instructions below, it is assumed you have the jumpers in the locations indicated above (the default jumper configuration). All mentions of left/right are made with respect to the image above.

1. Connect a programming cable between the JTAG port of ZedBoard Board and PC. To program the Zedboard, we used a micro USB cable and connected it to the JTAG port on the left of the power switch
2. Once the FPGA and radio board are connected correctly, turn on the board.

[2] Can be found at http://forums.xilinx.com/xlnx/attachments/xlnx/zaps/615/1/zedboard_CTT_v2013_2_130807.pdf

3. Open iMPACT in the ISE Design tools (Start menu → All programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Tools → iMPACT.)

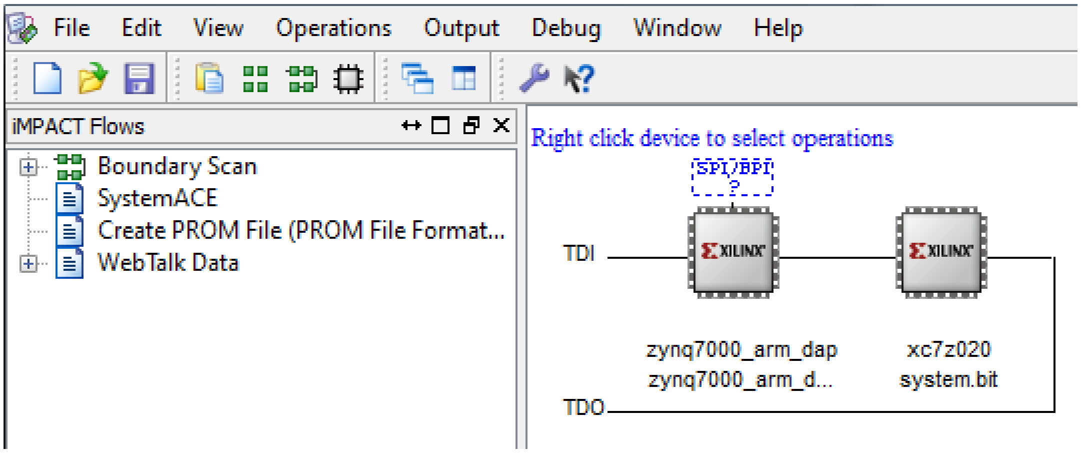
4. Select no if Impact asks you to load the last saved project.

5. Select yes to allow iMPACT to automatically create a new project for you. If you receive any connection errors, verify your USB or JTAG programmer cables are connected properly.

6. Select the Automatic option for the JTAG boundary scan setting and click ok.

7. Hit yes to assign configuration files. Bypass the first file selection, but for the second selection, browse to the location of your system.bit file. It should be inside the "Implementation" folder of your EDK project folder.

8. Select ok on the next screen verifying that the board displayed is your Zynq xc7z020 board. It should look similar to Figure 3-3 below.



3-3: proper configuration of the Zed Board system.bit file

9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.

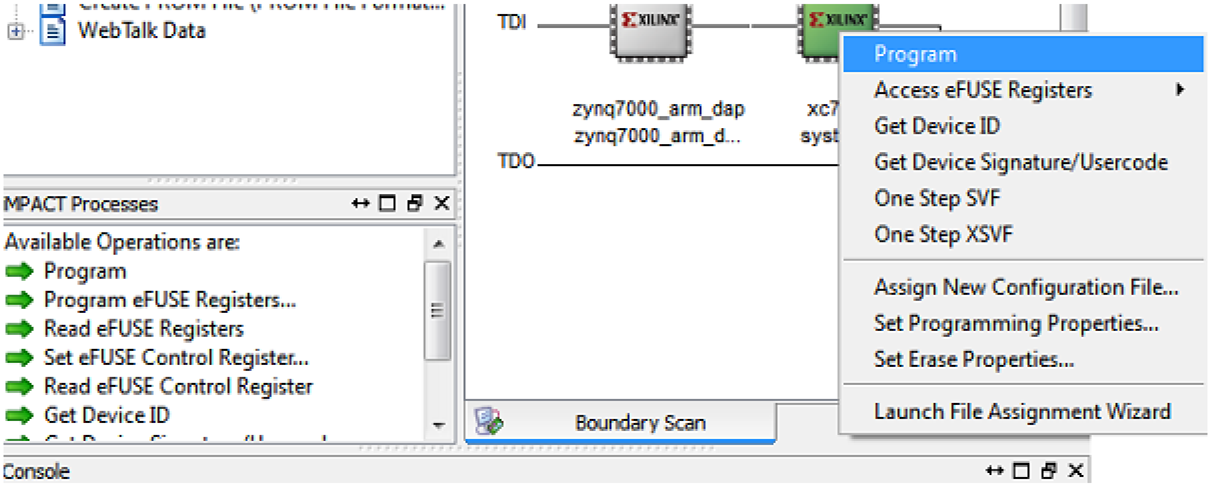

WebTalk Data
TDI
TDO
zynq7000_arm_dap
zynq7000_arm_d...
xc7
syst
iMPACT Processes
Available Operations are:
Program
Program eFUSE Registers...
Read eFUSE Registers
Set eFUSE Control Register...
Read eFUSE Control Register
Get Device ID
Boundary Scan
Console
Program
Access eFUSE Registers
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard

Figure 3-4: iMPACT configuration screen

3.3 Debugging with SDK

If the hardware design is correct, you should see a blue light on the ZED Board indicating the program was successful. You can now return to the SDK project screen to test your software.

1. Test it by **right clicking** the `blinky` project folder and selecting **Debug As → Launch on Hardware (GDB)**.

2. You should now be taken to a screen which shows the `init_platform()` function as highlighted. You can now start the software program by clicking the **play** button in the top menu.

If the software initialization worked, you should see the Blinking LEDs on the FPGA counting from 0 to 15 just as in your MATLAB algorithm.