Toyon Research Corporation

Lab 0: Blink LEDs

Chilipepper Tutorial Projects

Version 0.4

1/24/2012

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 Simulink Block using MathWorks HDL Coder with MATLAB code entry, exporting your Simulink Model as a PCore using Xilinx System Generator, 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 2013a
- Xilinx ISE Design Suite 14.4 with EDK and System Generator
- 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 Simulink Model in System Generator
- Create a project in EDK
- Export the core to EDK
- Interface the core with the 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 Simulink as a Xilinx System Generator Black Box
- Export from Simulink using Xilinx System Generator EDK block integration
- Integrate the resultant PCore in Xilinx EDK and target the bitfile to 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* 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 Configure System Generator for MATLAB 2013a

Before we use the Workflow Advisor tool, there is an integration step which requires executing a few MATLAB commands.

1. First, ensure that you have not configured System Generator for use with MATLAB 2013a. If you are unsure, you should use the following steps to check.
   a. Go to Start Menu →Xilinx Design Tools → ISE Design Suite 14.4 → System Generator MATLAB Configurator.
   b. Check that System generator is not configured with MATLAB 2013a. If it is not, proceed. If it is, remove the configuration.
2. Open MATLAB 2013a, and enter the following lines into the MATLAB command window.

```
>> cd C:\Xilinx\14.4\ISE_DS\ISE\sysgen\bin\nt64

>> xlAddSysgen C:\Xilinx\14.4\ISE_DS\ISE\
```

Figure 1-1: System Generator configuration commands

3. Navigate back to the directory with your HDL project file.

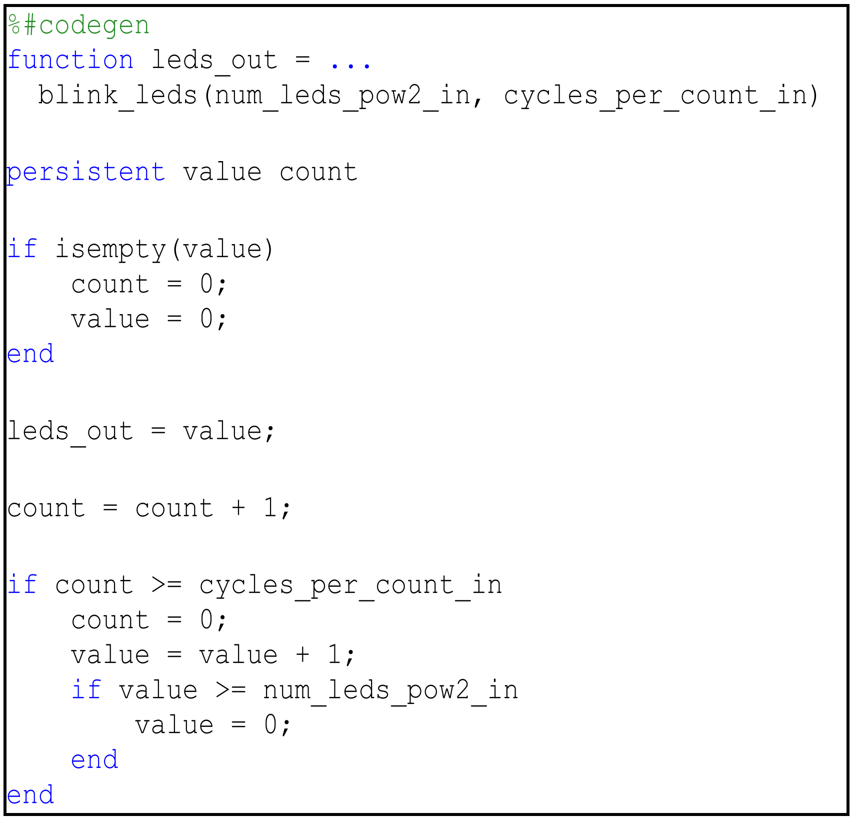
This should configure System Generator for use with MATLAB.

Note	You can also add these two lines to the startup.m file in your MATLAB folder. This will save you the time of having to type the commands when you start MATLAB. However the commands can take some time to execute and may cause your MATLAB to take slightly longer to load.

1.3 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 `leds_out` will be connected to the LEDs on the board. This function will increment a counter every clock cycle, and at some interval defined by `cycles_per_count_in` will also increment the value that is output to the LEDs. This value will set each LED as on or off based on each corresponding bit of the binary value. The function is shown in Figure 1-2 below.



```
%#codegen
function leds_out = ...
  blink_leds(num_leds_pow2_in, cycles_per_count_in)

persistent value count

if isempty(value)
    count = 0;
    value = 0;
end

leds_out = value;

count = count + 1;

if count >= cycles_per_count_in
    count = 0;
    value = value + 1;
    if value >= num_leds_pow2_in
        value = 0;
    end
end
```

Figure 1-2: MATLAB function for blinking LEDs

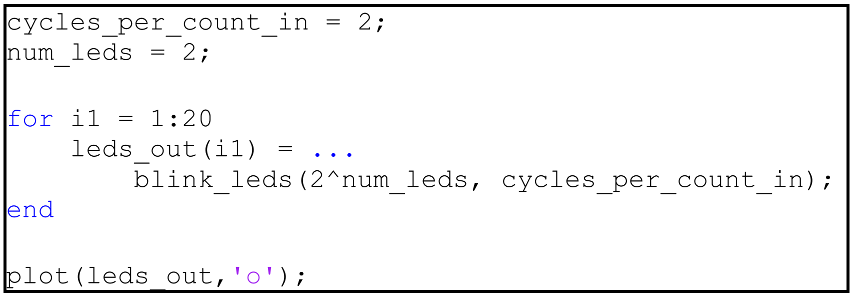
1. Create a directory for the project under C:\QPSK_Projects\Project_0.

2. **Save** this function as `blink_leds.m` inside the project directory.

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 lab. 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.

1.4 MATLAB Testbench

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 1-3.



```
cycles_per_count_in = 2;
num_leds = 2;

for i1 = 1:20
    leds_out(i1) = ...
        blink_leds(2^num_leds, cycles_per_count_in);
end

plot(leds_out,'o');
```

Figure 1-3: MATLAB code for HDL test bench script

1. Create a new **MATLAB script** with the contents of Figure 1-3.
2. **Save** this script as `blink_leds_tb.m` inside the project directory.
3. **Run** this script in MATLAB (be sure the project directory is in the MATLAB PATH variable) to test the function.
4. Once you have verified that your algorithm is correct, proceed to the next step of the lab.

1.5 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 matlab test bench**, select `blink_leds_tb.m` and click **Open.**

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

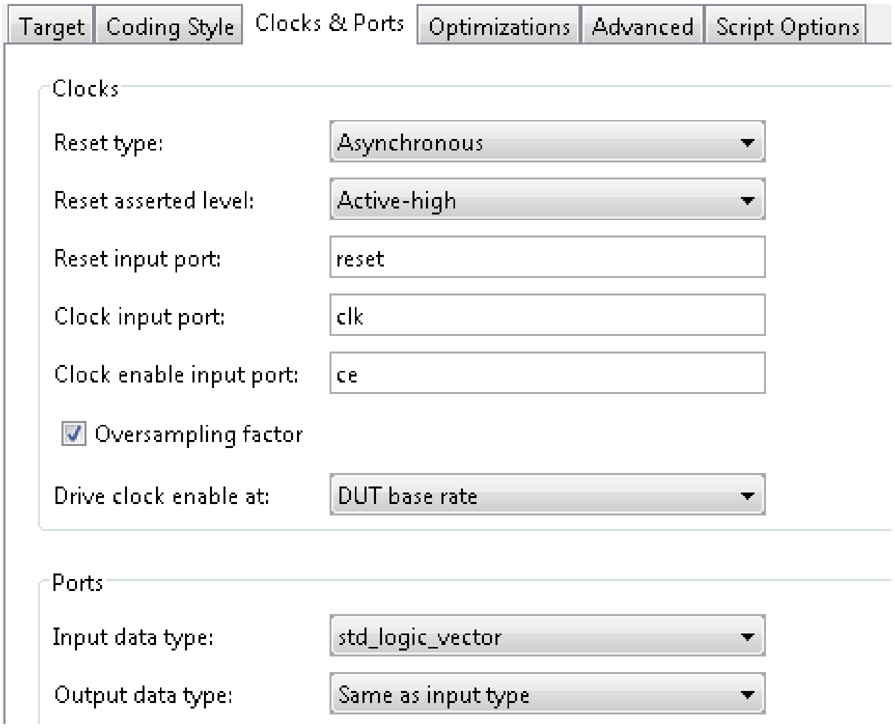
1.6 Synthesize Simulink Block with HDL Coder

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 **Code Generation** in the left hand menu and click on the **Clocks & Ports** tab. Configure the settings as shown in Figure 1-4 below.

   a. Set **Clock enable input port** to **ce**

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



Clocks

Reset type: Asynchronous

Reset asserted level: Active-high

Reset input port: reset

Clock input port: clk

Clock enable input port: ce

☑ Oversampling factor

Drive clock enable at: DUT base rate

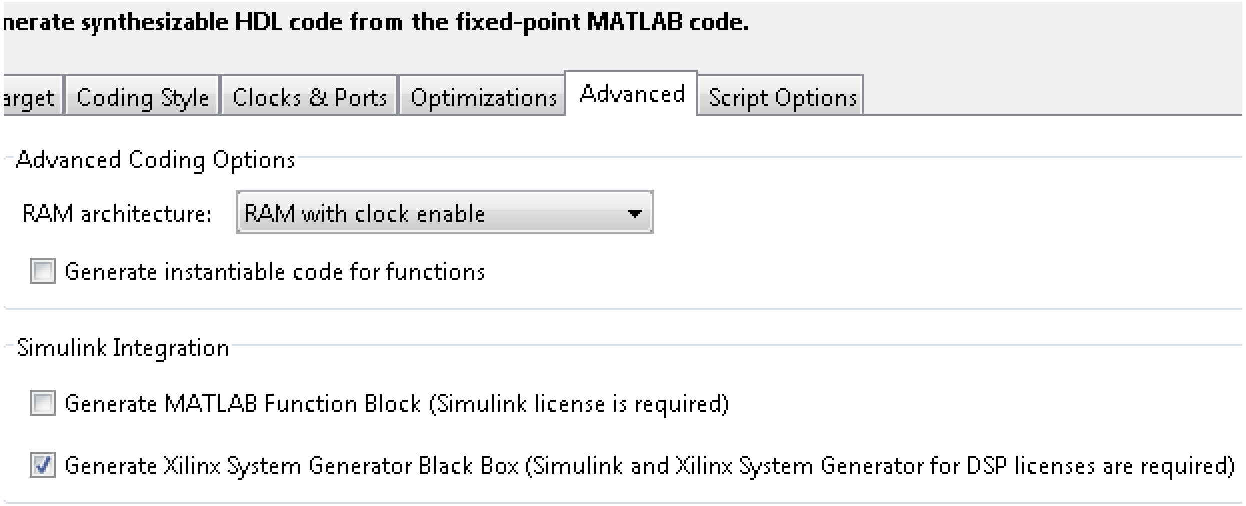
Ports

Input data type: std_logic_vector

Output data type: Same as input type

Figure 1-4: Clocks & Ports configuration settings

3. Under the **Advanced** tab, check the box for **Generate Xilinx System Generator Black Box.** Be sure your settings match those in Figure 1-5 below.



nerate synthesizable HDL code from the fixed-point MATLAB code.

Advanced Coding Options

RAM architecture: RAM with clock enable

☐ Generate instantiable code for functions

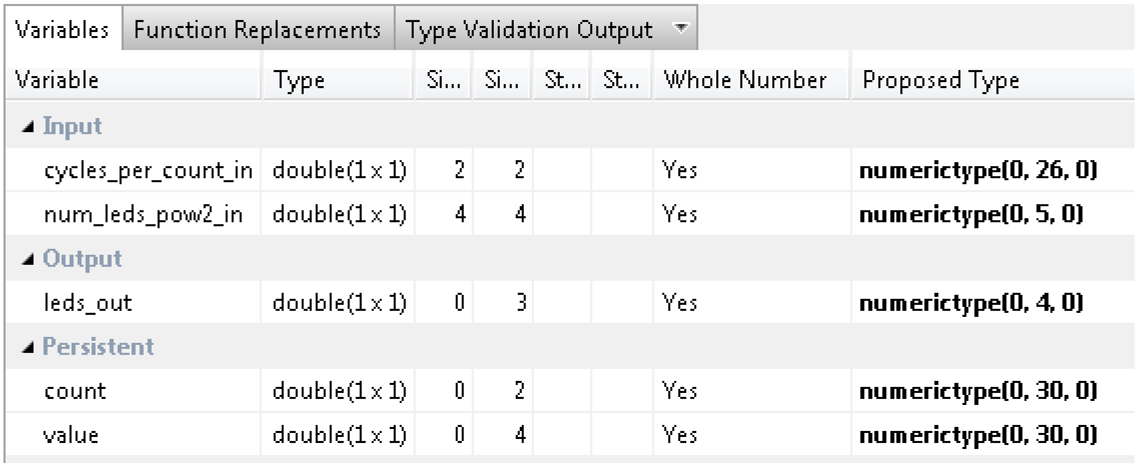
Simulink Integration

☐ Generate MATLAB Function Block (Simulink license is required)

☑ Generate Xilinx System Generator Black Box (Simulink and Xilinx System Generator for DSP licenses are required)

Figure 1-5: Advanced HDL Generator settings

4. Right-click **Propose Fixed-Point Types,** 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 1-6. If they do not, please change them to match what is shown below.



Variable	Type	Si...	Si...	St...	St...	Whole Number	Proposed Type
Input							
cycles_per_count_in	double(1 x 1)	2	2			Yes	numerictype(0, 26, 0)
num_leds_pow2_in	double(1 x 1)	4	4			Yes	numerictype(0, 5, 0)
Output							
leds_out	double(1 x 1)	0	3			Yes	numerictype(0, 4, 0)
Persistent							
count	double(1 x 1)	0	2			Yes	numerictype(0, 30, 0)
value	double(1 x 1)	0	4			Yes	numerictype(0, 30, 0)

Figure 1-6: Variable types for blink_leds MATLAB function

5. Once you have corrected the Type setting for all your variables, right-click **Code Generation,** and select **Run to Selected Task.** A Simulink model similar to Figure 1-7 containing your design should appear after the task is completed.

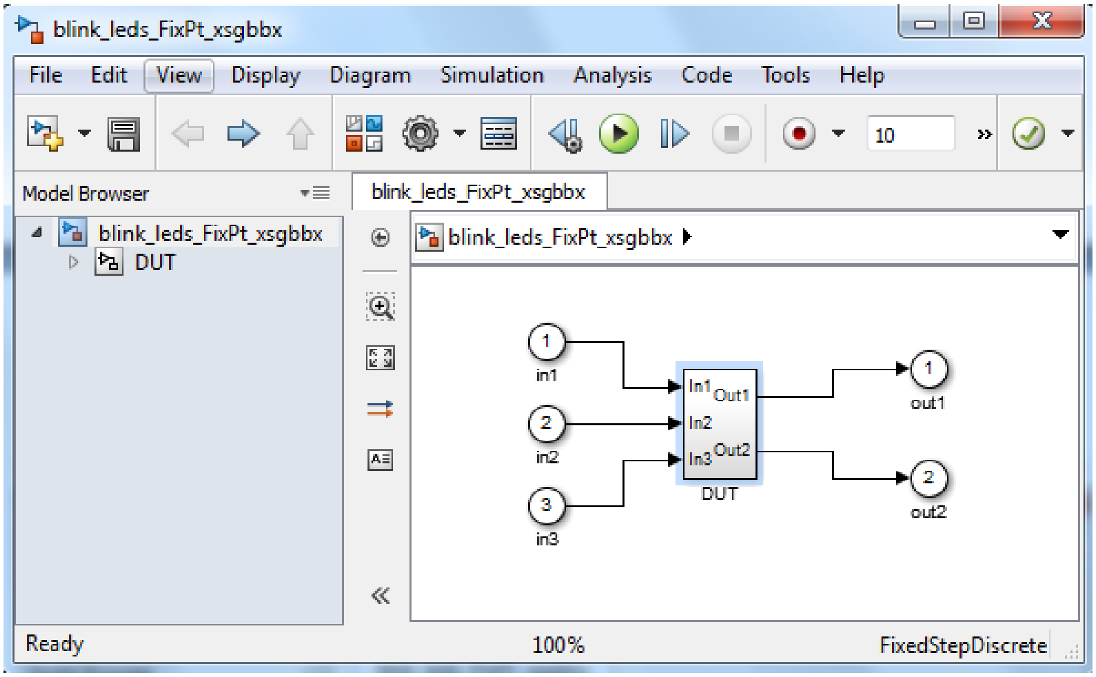


Figure 1-7: Simulink model for blink_leds MATLAB function

1.7 Troubleshooting

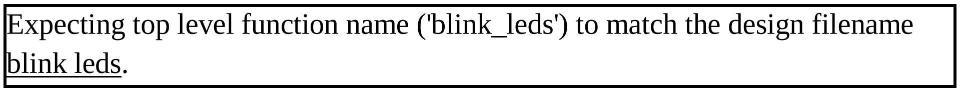
1. If during **Code Generation**, you receive the following errors:

```
Error occurred when running post codegeneration tasks
Error: failed to run post code generation tasks:
hdlcoder:matlabhdlcoder:clkcenameforXSG In order to work
with Xilinx System Generator for DSP, clock and clock
enable must be named as "clk" and "ce".
```

a. Check the **Clocks & Ports** settings. The **Clock enable input port** should be set to **ce**

```
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".
```

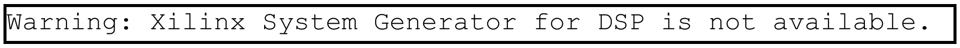
b. Check the **Clocks & Ports** settings. The **Drive clock enable at** should be set to **DUT base rate**



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

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

2. If code generation has completed successfully, but no Simulink model appears, look through the log in the HDL Code Generation window. If you see this line ~4 lines down from the top:



```
Warning: Xilinx System Generator for DSP is not available.
```

a. You have not configured System Generator for use with MATLAB. Refer back to Step 1.2 Configure System Generator for MATLAB 2012b.

i. If you have already done this, then restart MATLAB and try it again.

ii. If you have not done Step 1.2, please do so.

3. 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 within the **PATH settings** of your MATLAB installation.

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

Import Core to System Generator Step 2

2.1 Modify Simulink Design for your Core

Now that you've generated the Simulink Design Block, let's customize it for our application.

1. Navigate to the bottom-most level of the Simulink model. Select all (Ctrl + A) and copy and paste this into a new Simulink model (**File → New → Model**). You can close the old model without saving it as we won't be using it again.

2. The embedded system design process requires creating several project files and directory trees. It is very helpful to create a consistent structure for all of the files within your design files, so they can be easily accessed in the future. To do this, please use the following:

    a. In the directory with your MATLAB and HDL Coder project files, create a new folder named **SysGen**.

    b. **Save** the new model you created into this SysGen folder. Name it something descriptive like **blink_leds**. If you receive a message about **shadowing**, just click save to continue.

    c. Copy the blink_leds_FixPt_xsgbbxcfg.m file from within the \codegen\blink_leds\hdlsrc directory into the SysGen folder.

    d. Open the cfg file, and change line 39 to the following.

    ```
    this_block.addFile('hdl\blink_leds_FixPt.vhd');
    ```

    This will direct Simulink to the new location of the configuration file.

    e. Create a folder named **hdl** within the SysGen folder.

    f. From \codegen\blink_leds\hdlsrc, move the blink_leds_FixPt.vhd file to the \SysGen\hdl directory.

NOTE Be sure to repeat this step any time you regenerate your HDL code.

3. Add blocksets from the Xilinx library to create a model similar to the one in Figure 2-1 below.

    a. Select **View → Library Browser** to show the list of possible block set libraries

    b. Inside the Xilinx Blockset on the left hand menu pane, double click the Index block. The components used to create the model can be dragged from the library browser into the Simulink model.

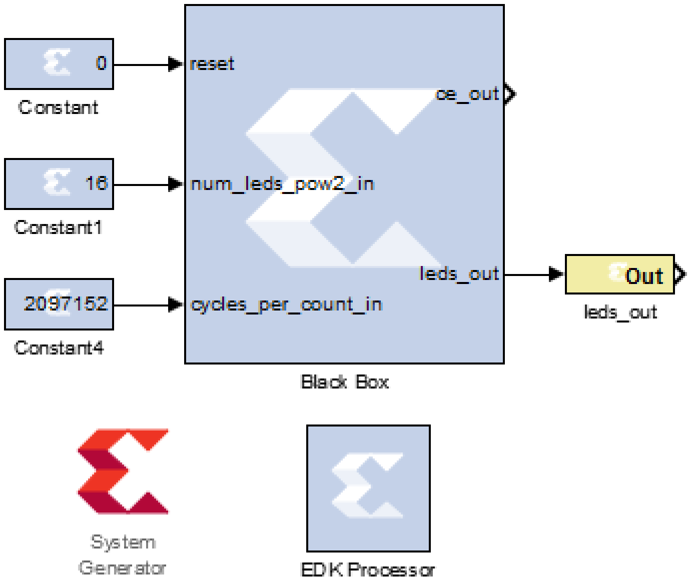
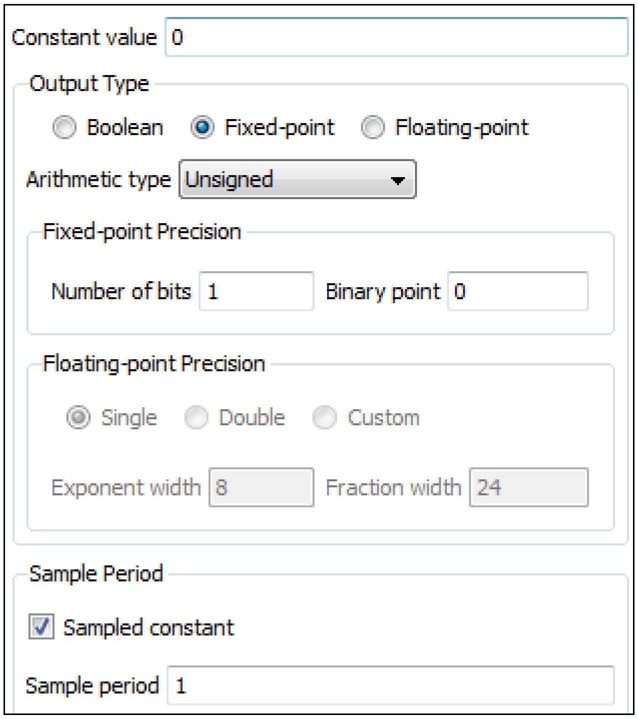


Figure 2-1: Simulink model for blink_leds.m

4. Connect a **Constant** to the **reset** input and configure it as shown in Figure 2-2.



Constant value 0

Output Type

○ Boolean ● Fixed-point ○ Floating-point

Arithmetic type Unsigned

Fixed-point Precision

Number of bits 1 Binary point 0

Floating-point Precision

● Single ○ Double ○ Custom

Exponent width 8 Fraction width 24

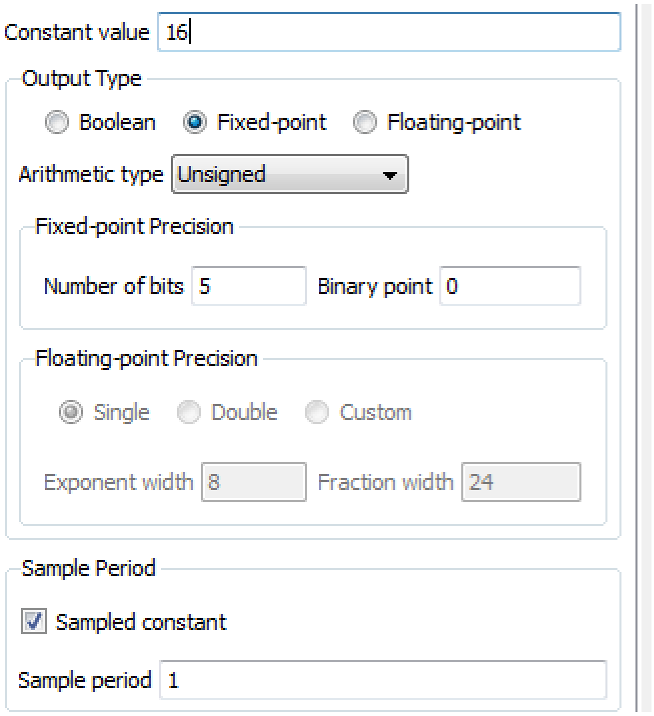
Sample Period

☑ Sampled constant

Sample period 1

Figure 2-2: Settings for reset input constant

a. Connect a **Constant** to **num_leds_pow2_in** and configure it as shown in Figure 2-3.



Constant value 16

Output Type

○ Boolean ● Fixed-point ○ Floating-point

Arithmetic type Unsigned

Fixed-point Precision

Number of bits 5 Binary point 0

Floating-point Precision

● Single ○ Double ○ Custom

Exponent width 8 Fraction width 24

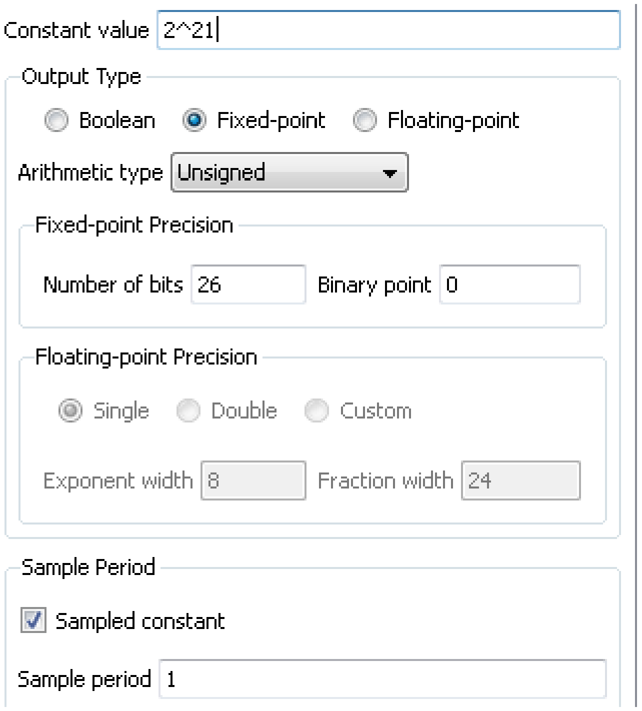
Sample Period

☑ Sampled constant

Sample period 1

Figure 2-3: Settings for num_leds_pow2 input constant

b. Connect a **Constant** to **cycles_per_count** and configure it as shown in Figure 2-4



Constant value 2^21

Output Type

○ Boolean ● Fixed-point ○ Floating-point

Arithmetic type Unsigned

Fixed-point Precision

Number of bits 26 Binary point 0

Floating-point Precision

● Single ○ Double ○ Custom

Exponent width 8 Fraction width 24

Sample Period

☑ Sampled constant

Sample period 1

Figure 2-4: Settings for cycles_per_count_in input constant

5. Connect a **Gateway Out** block to the output **leds_out**. Click on the **Gateway Out** label on the bottom of the block to rename it. Rename the block to **leds_out.**

6. Add an **EDK Processor** block to the design. Double-click on the block, to let it synchronize memories with your design. Press **OK.**

The Simulink model for your core is now complete.

2.2 Troubleshooting

1. If after selecting **View → Library Browser** you don't see Xilinx Blockset on the left hand menu pane, it means your System Generator is not configured correctly for MATLAB 2013a. Refer to section 1.2 and carefully perform the steps outlined there.

Create Project in Xilinx EDK Step 3

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.

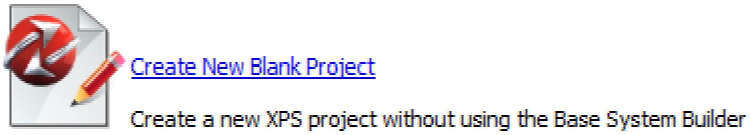
3.1 Create an XPS Project

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

1. Open XPS:

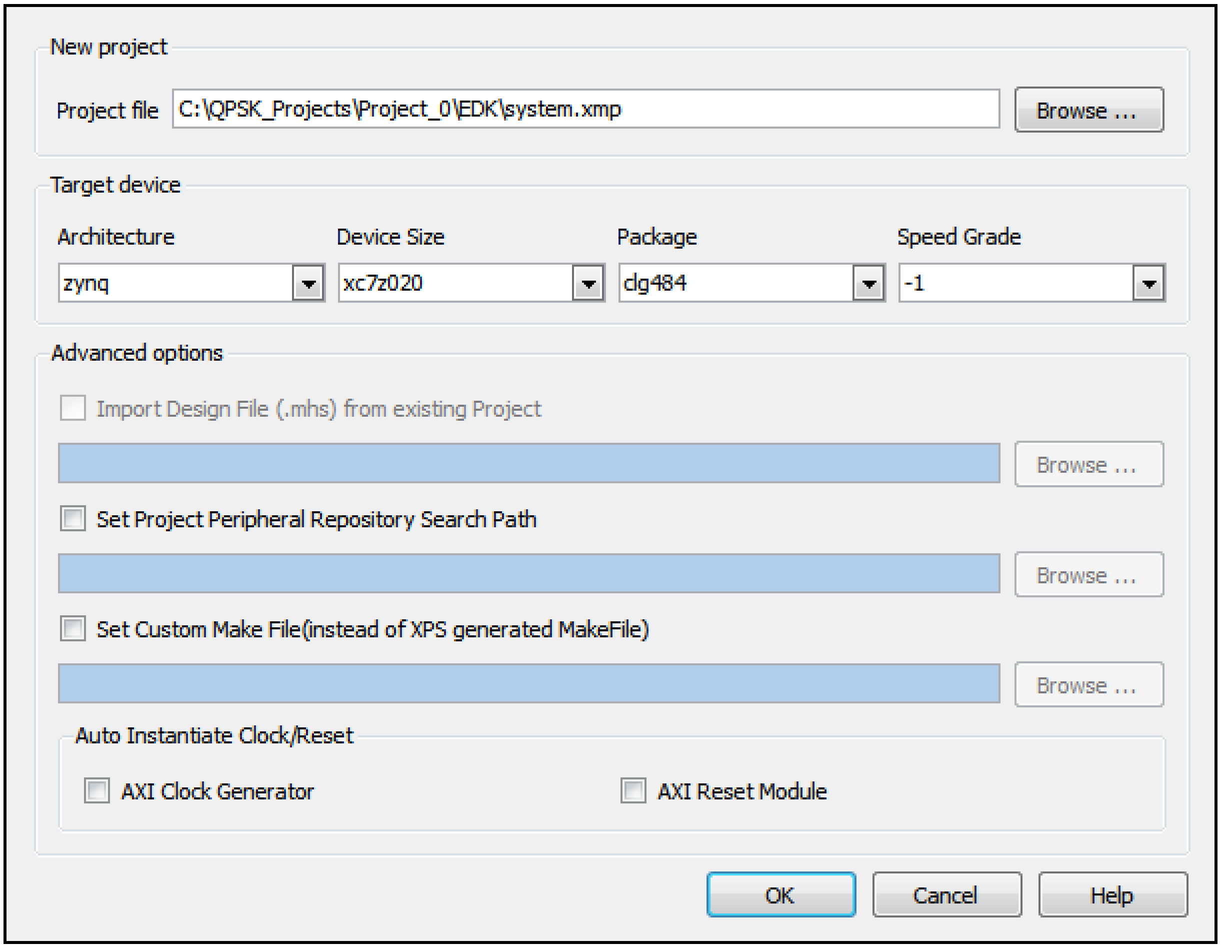
   Start menu → All programs → Xilinx Design Tools → ISE Design Suite 14.4 → EDK → Xilinx Platform Studio.

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

OK Cancel Help

Figure 3-1: 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.

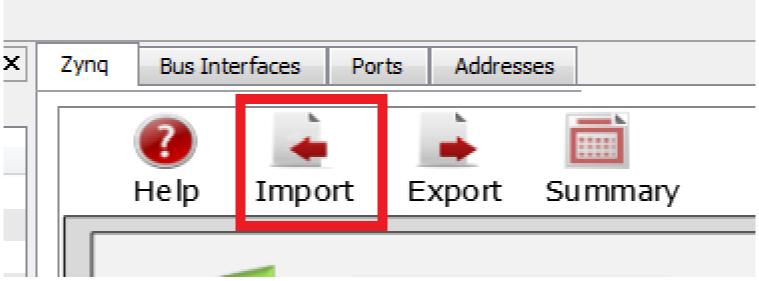
5. Keep the default name "system", and press **Save.**

6. Select Architecture/Device Size/Package/Speed Grade as shown in Figure 3-1 above.

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

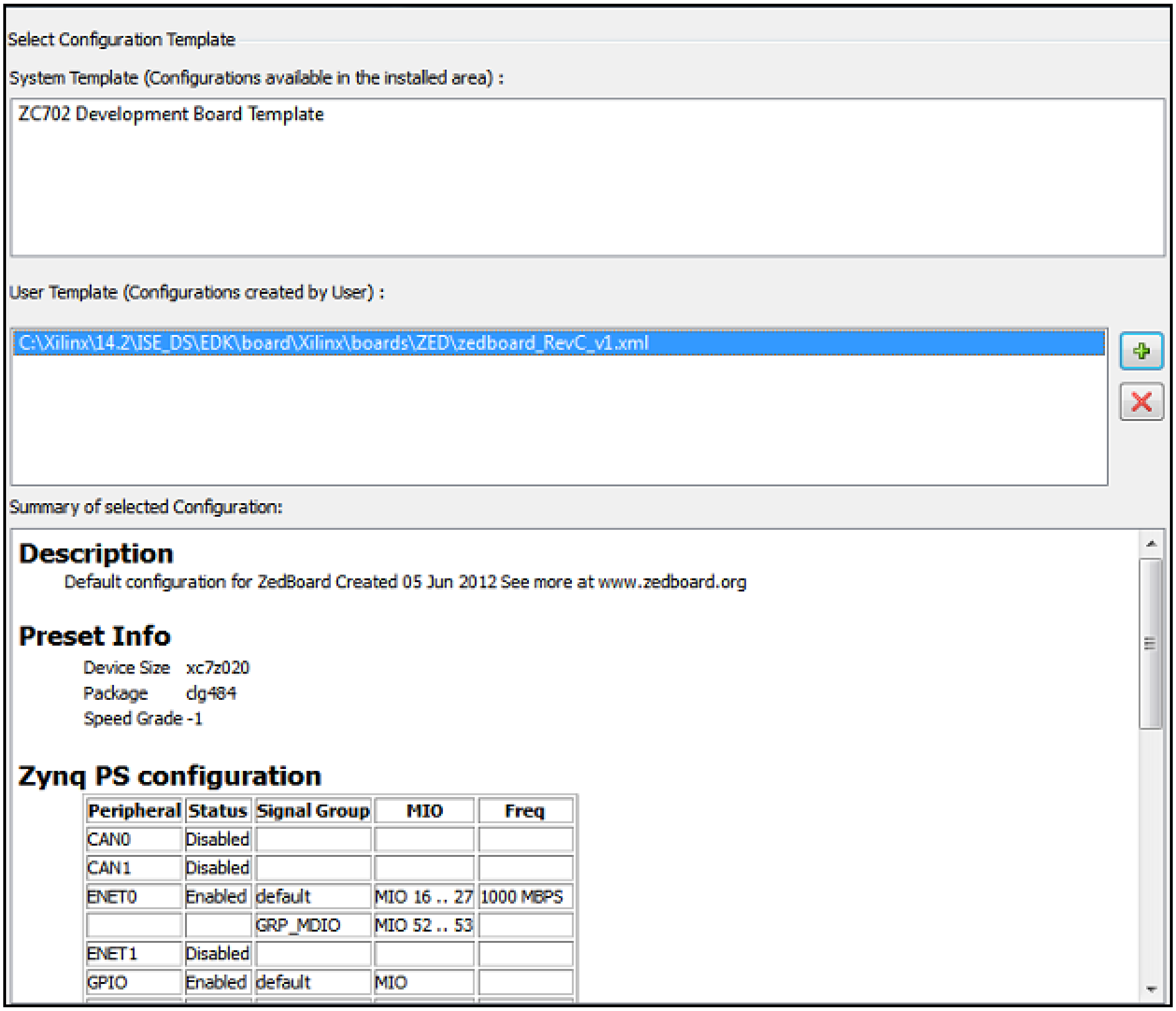
8. Press **OK.**

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 3-2 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 3-2: XPS Project Configuration Template

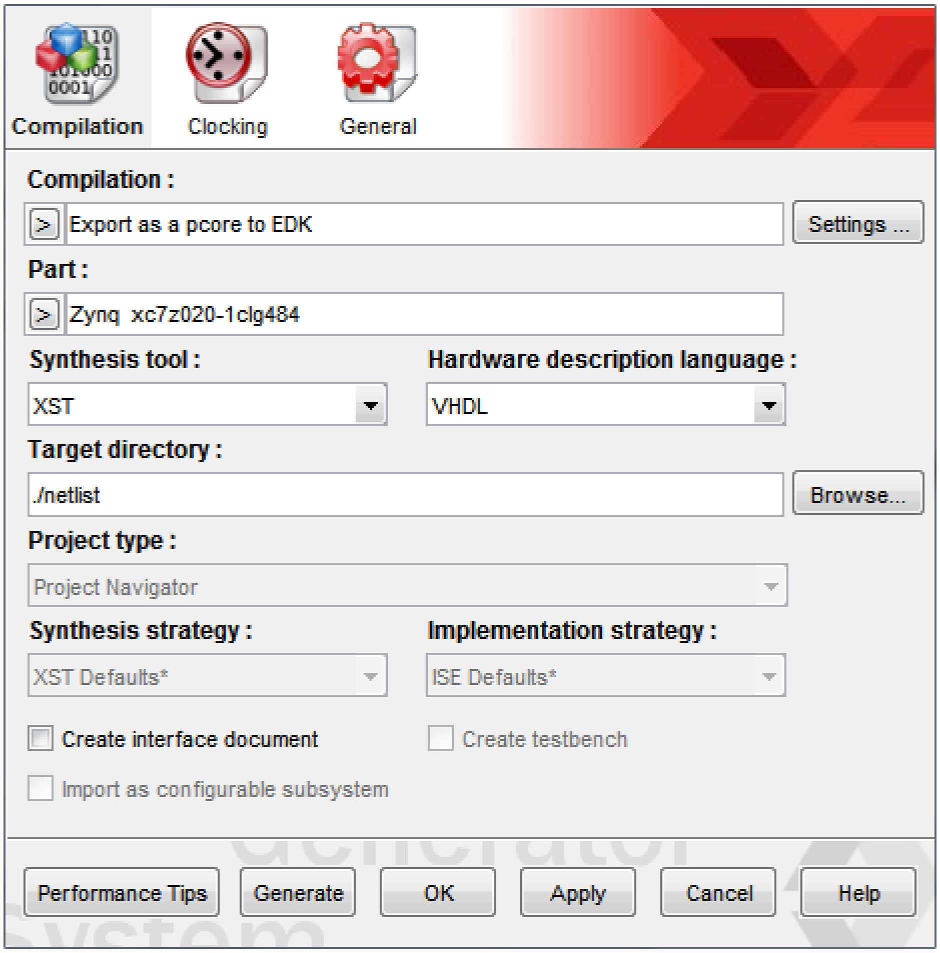
You have now created a blank XPS project for the Zedboard!

Export Core to EDK Step 4

This section will show you how to export your PCore (peripheral core) from Simulink to EDK. It will start out by showing you what needs to be done in Simulink, then take you into XPS to verify that the core has been exported successfully.

4.1 Configure System Generator

1. Going back to your Simulink model, double-click the **System Generator** token.



Compilation : > Export as a pcore to EDK

Part : > Zynq xc7z020-1clg484

Synthesis tool : XST

Hardware description language : VHDL

Target directory : ./netlist

Project type : Project Navigator

Synthesis strategy : XST Defaults*

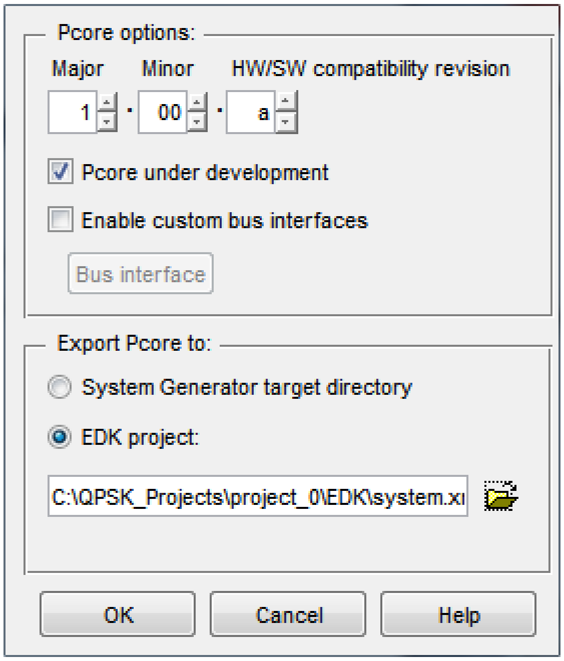
Implementation strategy : ISE Defaults*

☐ Create interface document ☐ Create testbench

☐ Import as configurable subsystem

Figure 4-1: Settings for System generator

2. Under **Compilation**: click > and select **EDK Export Tool**.
3. Click on **Settings**:



Pcore options:

Major: 1 · Minor: 00 · HW/SW compatibility revision: a

☑ Pcore under development

□ Enable custom bus interfaces

Export Pcore to:

○ System Generator target directory

● EDK project:

C:\QPSK_Projects\project_0\EDK\system.xı

Figure 4-2: System Generator compilation settings

a. Under **PCore options:** Check the **PCore under development** box.

b. Under **Export PCore to:** Select **EDK project**. Click 📂 and browse to the directory where you saved your XPS project. Select the **system.xmp** file in that directory.

c. Be sure your EDK export settings look similar to Figure 4-2 above. Press **OK.**

4. Under **Part:** click > and select **Zynq → xc7z020 → -1 → clg484** to select the FPGA for the ZedBoard.

5. Press **Apply.**

6. Press **Generate** to generate the PCore.

7. Once you see the Generation Completed message, you've successfully compiled the Simulink model!

4.2 Troubleshooting

It is possible that you will get an error after pressing Generate. This is especially likely if this is your first time generating a design with System Generator, after configuring System Generator for MATLAB R2012b.

1. If you get a fatal error (typically within a fairly simple block), then exit and restart MATLAB, go through the System Generator configuration process outlined in section 2.1 and try generating the PCore again.

4.3 Verify Export to EDK

Now, let's check if we've successfully exported the core or not. Going back to XPS…

1. From the menu bar, click on **Project → Rescan User Repositories**.

In the **IP Catalog** tab on the left, under **Project Local PCores → User**, you should see your PCore listed as shown in Figure 4-3.

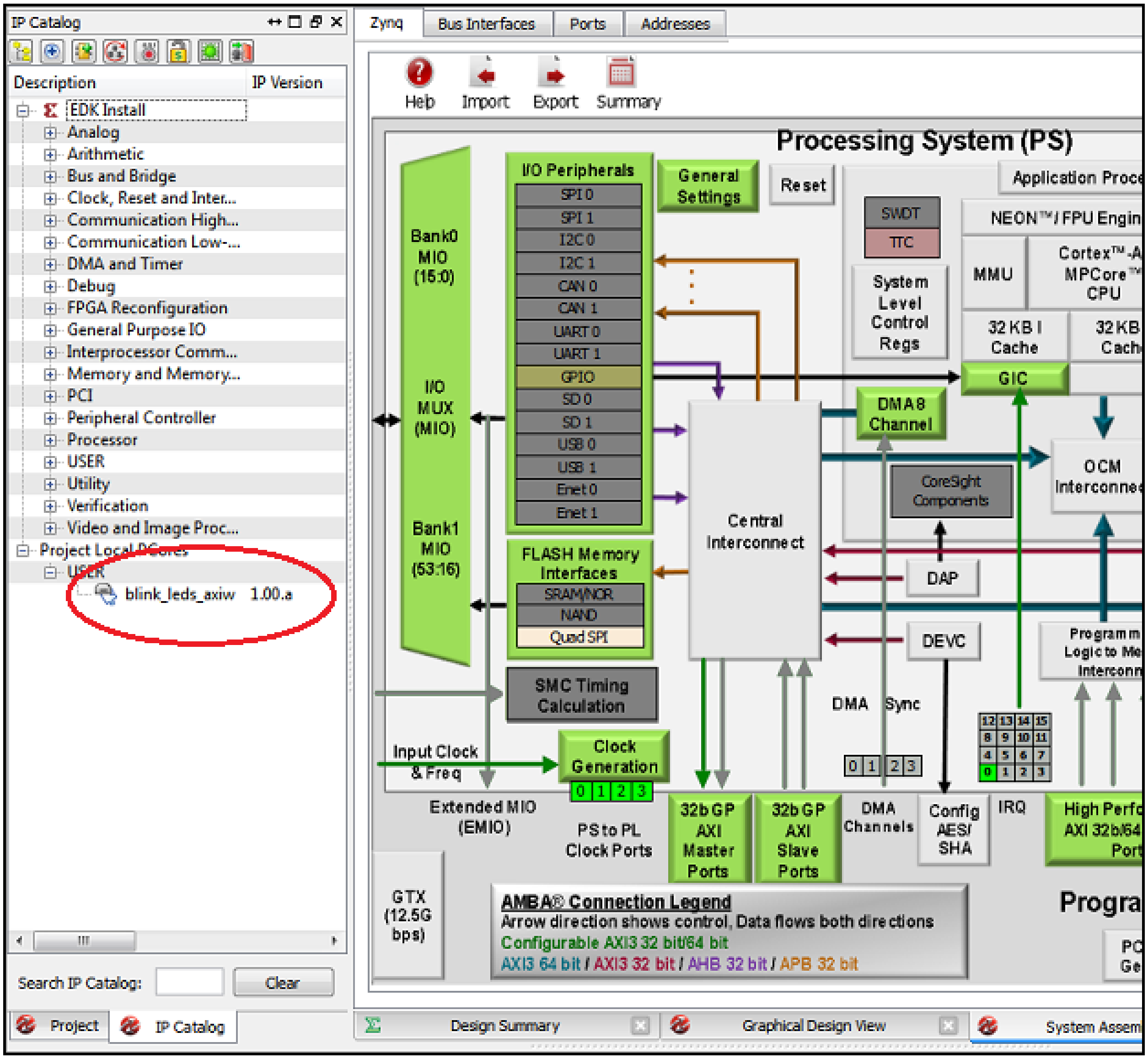


Figure 4-3: User PCore imported from Simulink

You have now exported your PCore into EDK, and you are ready to integrate it with your embedded processor.

Interface Core with Processor Step 5

In this step, we will integrate the newly exported pcore with the embedded processor. The role of MATLAB in this work flow is done; these next steps use only XPS.

5.1 Add IP to Design

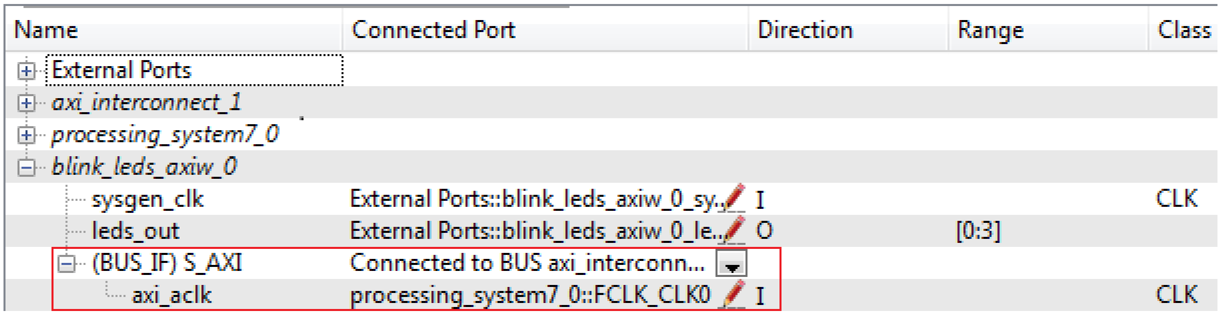
1. Double-click on your PCore.

2. Select **Yes** to add it to the design**.**

3. Click **OK** on the next two screens that appear to accept defaults.

5.2 Verify your PCore Design

1. Verify your PCore by making sure it looks similar to Figure 5.1. In particular, be sure that your design has the BUS_IF and axi_aclk shown in the Figure.



Name	Connected Port	Direction	Range	Class
⊞ External Ports				
⊞ *axi_interconnect_1*				
⊞ *processing_system7_0*				
⊟ *blink_leds_axiw_0*				
sysgen_clk	External Ports::blink_leds_axiw_0_sy...	I		CLK
leds_out	External Ports::blink_leds_axiw_0_le...	O	[0:3]	
⊟ (BUS_IF) S_AXI	Connected to BUS axi_interconn...			
axi_aclk	processing_system7_0::FCLK_CLK0	I		CLK

5-1: Pots Interface for blink_leds PCore

At the time this lab was created there was a synchronization issue between the Simulink EDK processor and the XPS software. The bug prevents Simulink from adding the bus interface to your PCore design. If you are missing the bus interface shown in the figure above, return to section 2.1 and add a **From Register** into the Simulink design. Sync as outlined in step 6 (click **add** after opening the EDK processor to show the register), then remove the register and re-sync the design.

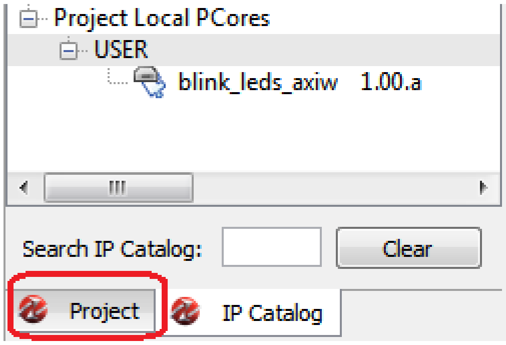
5.3 Integrate IP into Design

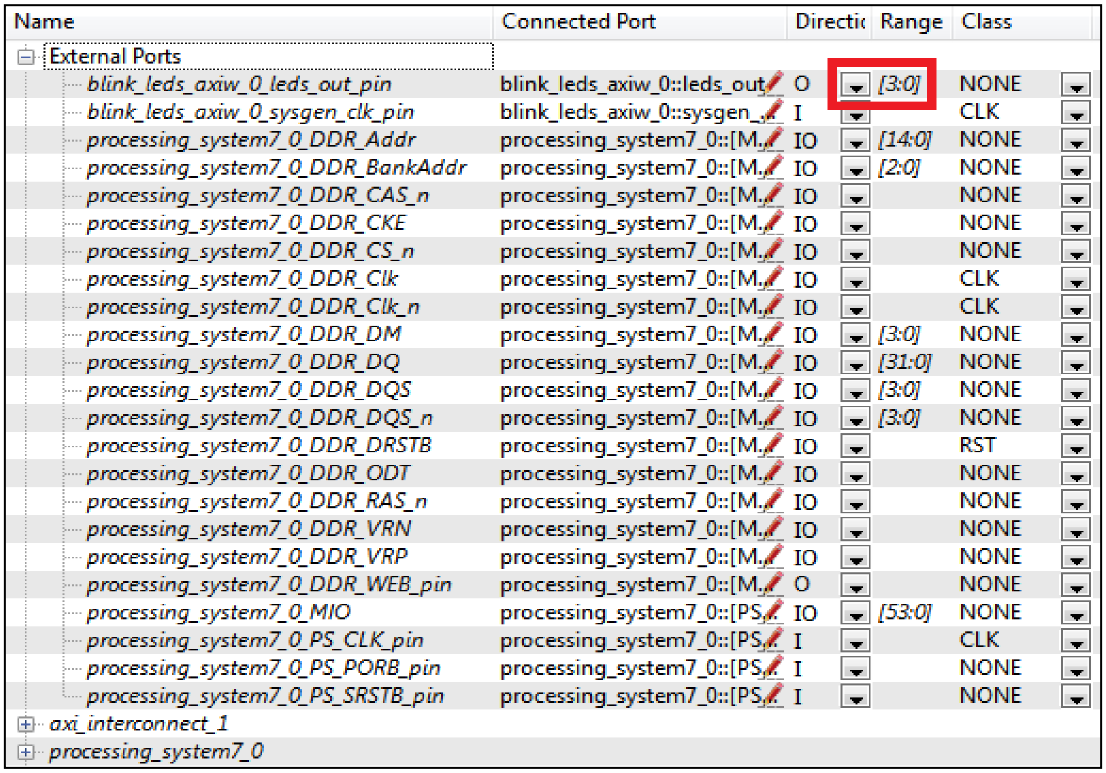
1. While in **System Assembly View**, go to the **Ports** tab. Scroll to the bottom of this window, to find your PCore listed. Expand your pcore to see all the inputs/outputs of the core.

2. Right-click on the signal **leds_out** and **sysgen_clk**, and select **Make External.** This will direct the leds signal to a peripheral.

3. Your External Ports and system configuration should look similar to Figure 5.2 below. Note that we have reversed the bit range for the LEDs as the default behavior reverses bit order.

4. Open the **Project** tab.





Name	Connected Port	Directic	Range	Class
External Ports				
blink_leds_axiw_0_leds_out_pin	blink_leds_axiw_0::leds_out	O	*[3:0]*	NONE
blink_leds_axiw_0_sysgen_clk_pin	blink_leds_axiw_0::sysgen_	I		CLK
processing_system7_0_DDR_Addr	processing_system7_0::[M.	IO	*[14:0]*	NONE
processing_system7_0_DDR_BankAddr	processing_system7_0::[M.	IO	*[2:0]*	NONE
processing_system7_0_DDR_CAS_n	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_CKE	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_CS_n	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_Clk	processing_system7_0::[M.	IO		CLK
processing_system7_0_DDR_Clk_n	processing_system7_0::[M.	IO		CLK
processing_system7_0_DDR_DM	processing_system7_0::[M.	IO	*[3:0]*	NONE
processing_system7_0_DDR_DQ	processing_system7_0::[M.	IO	*[31:0]*	NONE
processing_system7_0_DDR_DQS	processing_system7_0::[M.	IO	*[3:0]*	NONE
processing_system7_0_DDR_DQS_n	processing_system7_0::[M.	IO	*[3:0]*	NONE
processing_system7_0_DDR_DRSTB	processing_system7_0::[M.	IO		RST
processing_system7_0_DDR_ODT	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_RAS_n	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_VRN	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_VRP	processing_system7_0::[M.	IO		NONE
processing_system7_0_DDR_WEB_pin	processing_system7_0::[M.	O		NONE
processing_system7_0_MIO	processing_system7_0::[PS.	IO	*[53:0]*	NONE
processing_system7_0_PS_CLK_pin	processing_system7_0::[PS.	I		CLK
processing_system7_0_PS_PORB_pin	processing_system7_0::[PS.	I		NONE
processing_system7_0_PS_SRSTB_pin	processing_system7_0::[PS.	I		NONE
axi_interconnect_1				
processing_system7_0				

Figure 5-2: External ports for the FPGA design

5. 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 5.3 as a guideline.

```
#######################################################################
# PL clocks and reset
######################################################################
NET blink_leds_axiw_0_sysgen_clk_pin LOC = Y9    | IOSTANDARD = LVCMOS25;
NET blink_leds_axiw_0_sysgen_clk_pin TNM_NET = blink_leds_axiw_0_sysgen_clk;
TIMESPEC TS_blink_leds_axiw_0_sysgen_clk = PERIOD blink_leds_axiw_0_sysgen_clk
100.000 MHz;


######################################################################
# LEDs
######################################################################
NET blink_leds_axiw_0_leds_out_pin[0] LOC = T22  |  IOSTANDARD = LVCMOS33;
NET blink_leds_axiw_0_leds_out_pin[1] LOC = T21  |  IOSTANDARD = LVCMOS33;
NET blink_leds_axiw_0_leds_out_pin[2] LOC = U22  |  IOSTANDARD = LVCMOS33;
NET blink_leds_axiw_0_leds_out_pin[3] LOC = U21  |  IOSTANDARD = LVCMOS33;
```

5-3: Pin assignments for Blink LEDs

6. Previous versions of EDK have a documented issue with AXI-bus generation for Simulink PCores targeting the Zynq FPGA. **This error has been corrected as of version 14.4**.
   a. To fix the bug, open up the .vhd file in the directory, PCores / <ip_name>_axiw_v1_00_a / hdl / vhdl / <ip_name>_axiw.vhd and make the following changes.

   b. Comment out everything from line 37 to line 73. Also comment out everything from line 217 to line 232.

   c. Save and close the file.

7. Be sure to save and close the **UCF** file as well.

Note	The lines commented out in step 6b may not be the same lines you need to comment if your design has more or fewer components than what was specified in the lab. To verify exactly what lines you should comment out, please see the blink_leds vhd file.

8. Select **Project → Rescan User Repositories** as we did one time before. It is good practice to Rescan User Repositories every time you make changes to a core.

5.4 Generate Bitstream

1. It is optional, but some recommend going through **Project →Design Rule Check** to quickly check for errors.

2. In the Navigator pane on the left, click **Generate BitStream** (alternatively, you can go to Hardware→Generate Bitstream). This process may take a while.

You are now ready to program the FPGA with your bitfile!

Program FPGA Step 6

6.1 Set up your ZedBoard for Programming

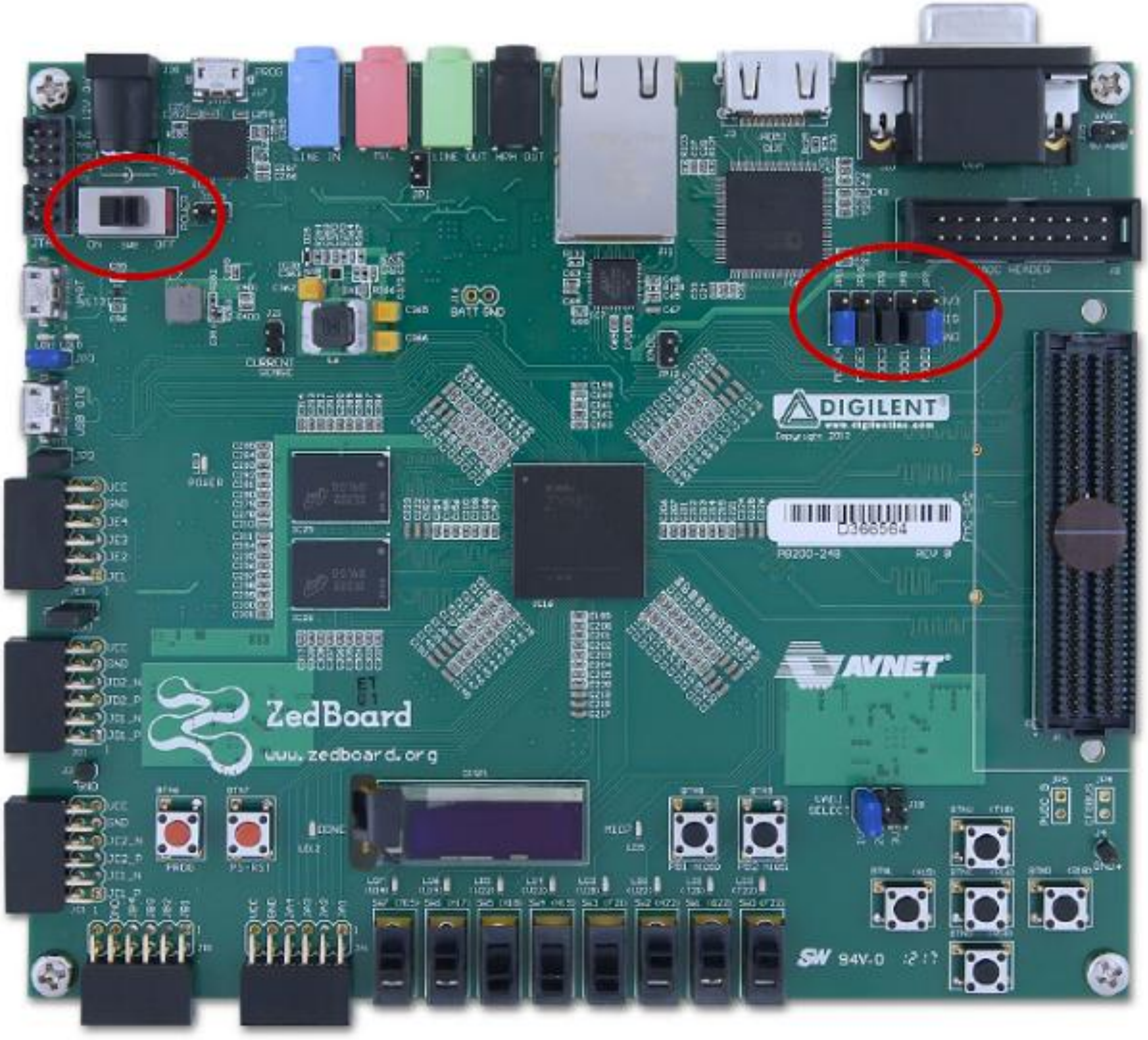


Figure 6-1. Jumper Configuration for the ZedBoard. Image is from Zynq ZedBoard Concepts, Tools, and Techniques

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. Note: To program the Zedboard, we used the Xilinx Platform Cable and connected it to the JTAG port on the left of the power switch.
2. Flip the power switch to power on the board.

Refer to the ZedBoard Basic Setup and Operation Guide for more details:

6.2 Download Bitstream

Now that your bitstream has successfully been generated, go to **Device Configuration → Download Bitstream.** This will program your FPGA with your design. The blue Done LED will illuminate when the FPGA has been programmed.

You should see that the 4 LEDs you pointed the core's outputs to are blinking.

Congratulations! You've created a core which blinks LEDs!

Conclusion

HDL Coder helps bridge the gap between algorithm and FPGA implementation significantly. After the core has been developed in MATLAB, the tool is used to generate HDL code. The HDL code generated is placed into a Black Box in System Generator, and exported to XPS. There, it is interfaced with an embedded processor, and downloaded onto the FPGA.

The use of system generator to export cores to EDK is a well-established work flow, and the addition of HDL Coder bringing the MATLAB algorithm to System Generator is invaluable.