|  |
| --- |
| Toyon Research Corporation |
| Lab 9: QPSK Receiver/Transmitter |
| Chilipepper Tutorial Projects |

|  |
| --- |
| Version 0.2  6/18/2013 |

Table of Contents

[Introduction 4](#_Toc359329552)

[Procedure 4](#_Toc359329553)

[Objectives 4](#_Toc359329554)

[Generate HDL Code 5](#_Toc359329555)

[1.1 MATLAB Functions 5](#_Toc359329556)

[1.2 MATLAB Test Bench 5](#_Toc359329557)

[1.3 RX HDL Coder Project 6](#_Toc359329558)

[1.4 dc\_offset HDL Coder Project 7](#_Toc359329559)

[1.5 TX HDL Coder Project 8](#_Toc359329560)

[Export Simulink models 12](#_Toc359329561)

[2.1 Create MCU Simulink Design 12](#_Toc359329562)

[2.2 Create ADC driver Simulink Design 13](#_Toc359329563)

[2.3 Create DAC driver Simulink Design 14](#_Toc359329564)

[2.4 Create Transmitter Simulink Design 15](#_Toc359329565)

[2.5 Create Receiver Simulink Design 16](#_Toc359329566)

[Configure Cores and Export Design 19](#_Toc359329567)

[3.1 Needed IP Cores 19](#_Toc359329569)

[3.2 Configuring the Pcore Ports 20](#_Toc359329570)

[3.3 Configuring the GPIO Ports 20](#_Toc359329571)

[3.4 Configuring the Clock Generator IP Core 20](#_Toc359329572)

[3.5 Pin Assignments 22](#_Toc359329573)

[Create software project 23](#_Toc359329574)

[4.1 Creating a new C Project 23](#_Toc359329576)

[4.2 Adding Supporting files 24](#_Toc359329577)

[4.3 Loading Hardware Platform with iMPACT 25](#_Toc359329578)

[4.4 Debugging with SDK 26](#_Toc359329579)

[Testing and Design Verification 27](#_Toc359329580)

[5.1 Verification with Terminal 27](#_Toc359329581)

[Appendix A main.c 28](#_Toc359329582)

[Appendix B UCF file 30](#_Toc359329583)

Lab 9: QPSK Receiver/Transmitter

# Introduction

This lab will extend the previous labs and allow you to combine your receive core designed in lab 8 with the transmitter designed in lab 3. The focus of this combination will be on sending a message to one FPGA via terminal, sending this message to the Chilipepper for transmission, receiving the transmitted QPSK waveform using a second Chilipepper board, and outputting this message directly to another terminal as done in the previous lab. Both the transmit and receive core created in the previous labs will be used in this lab to create a single EDK project, and therefore prior knowledge of the workings of HDL Coder as well as the Xilinx EDK environment is assumed. It is recommended that you complete the previous labs before completing this lab.

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 and export Simulink models using System Generator
* Configure your created PCores and export the design into SDK
* Create software to run your design
* Test and verify your results

## Objectives

After completing this lab, you will be able to:

* Create a Simulink model to interface your PCore to the MicroBlaze Processor
* Send and Receive a QPSK Waveform using the Chilipepper FMC
* Send/Receive output to/from the MicroBlaze Processor Serial Port
* Create a software application to test your design

Generate HDL Code Step 1

This section will show you how to create your MATLAB function and test bench files as well as the process for generating the HDL code used in the Simulink model.

## 1.1 MATLAB Functions

Your MATLAB functions will eventually become a core that will be synthesized into hardware. The algorithm describes the operations in each clock cycle, and processes data on a sample-by-sample basis. This lab uses the same MATLAB algorithm used in Lab 8 for the receiver core. Therefore, the MATLAB files are identical and can be simply copied from the previous Lab or downloaded from the GitHub Repo. For your convenience, a list of the required MATLAB files is given below.

* CreateAppend16BitCRC.m
* make\_srrc\_lut.m
* make\_train\_lut.m
* make\_trig\_lut.m
* mybitget.m
* qpsk\_rx.m
* qpsk\_rx\_correlator.m
* qpsk\_rx\_foc.m
* qpsk\_rx\_srrc.m
* qpsk\_rx\_toc

Additionally, the MATLAB files used for the transmitter core are identical to those used in lab 3. They can be copied directly from lab 3 or downloaded from the GitHub repo. The required files are listed below.

* qpsk\_srrc.m
* qpsk\_tx.m
* qpsk\_tx\_byte2sym.m

Lastly, the MATLAB file needed for DC Offset Correction (dc\_offset\_correction.m) can also be downloaded from the GitHub Repo or copied from a previous lab.

## 1.2 MATLAB Test Bench

Now that you have all of the MATLAB files required for the design, you also need the associated test bench scripts to create the HDL coder projects. This lab requires 2 test bench scripts, 1 for tx/rx and the other for the dc offset core. These test bench scripts are identical to the ones used in lab 8 and lab 4 respectively. You can either copy these files from their lab locations, or download them from the GitHub repo.

## 1.3 RX HDL Coder Project

Using the same steps outlined in the previous labs, we will create 3 new HDL coder projects called rx\_qpsk, tx\_qpsk, and dc\_offset\_correction.

The first project will be rx\_qpsk.prj. Add the MATLAB function qpsk\_rx.m and the test bench script qpsk\_tb.m to the **MATLAB Function** and **MATLAB Test Bench** categories respectively.

Once you open your Workflow Advisor, you should be greeted with a screen similar to Figure 1-1 which allows you to define input types for your function. You can also allow them to be auto-defined by simply selecting run, and letting MATLAB analyze your design. For the inputs listed for the qpsk\_rx function, the auto-defined types are fine.

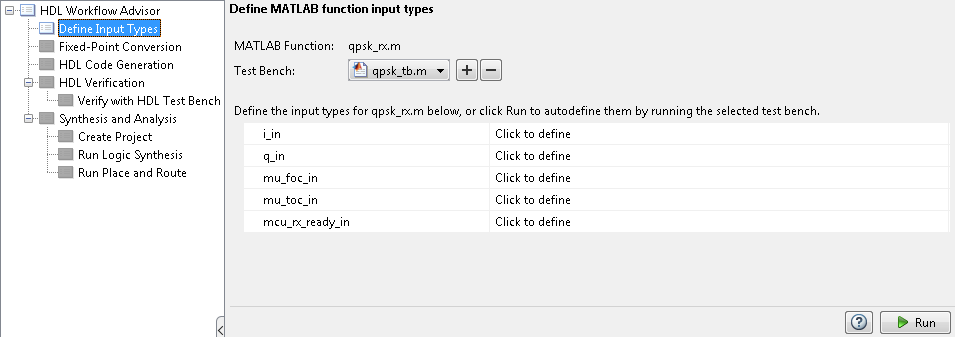


Figure 1‑1: HDL Code Generation Workflow Advisor

1. Open Workflow Advisor and select “Run” to define the input types.
2. Click on Fixed-Point Conversion and select run this task. This process may take awhile.

Once the process is completed, you should receive the “Validation succeeded” popup as usual. As in the previous labs, not all of the automatic selections are sufficient for our FPGA design; therefore several of the conversions will need to be modified.

1. Using the function dropdown menu at the top of the HDL Code Generation screen, select each of the functions in the design and modify them with the same settings used in Lab 8.

Once all modifications have been made, select “Validate Types” in the top right area of the top toolbar to verify the design for your modified Fixed-Point conversions. Again, once the process is complete, you should get a message saying Validation Succeeded.

1. Click on HDL Code Generation and modify the settings according to the previous labs. There is no pipelining required for this project. Right Click and select run this task to generate your Xilinx Block Box Design.
2. Once created, copy the black box and System Generator blocks to a new Model just as in the previous labs.
3. Save the new model as rx.slx into the sysgen directory “Lab\_9\sysgen”.
4. Copy the qpsk\_rx\_FixPt\_xsgbbxcfg.m file into your Sysgen folder just as in the previous labs.
5. Create the hdl folder inside the Sysgen folder and copy your vhd files into this directory. Make sure you modify the previously copied m file to point to the new location of the vhd files.

## 1.4 dc\_offset HDL Coder Project

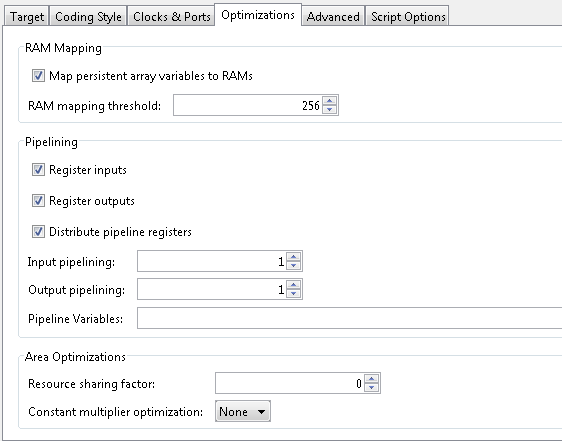
The next project will be dc\_offset.prj. Add the MATLAB function dc\_offset\_correction.m and the test bench script dc\_offset\_correction\_tb.m to the **MATLAB Function** and **MATLAB Test Bench** categories respectively.

Once you open your Workflow Advisor, you will again be greeted with a screen similar to Figure 1-1. For the inputs listed for the dc\_offset\_correction function, the auto-defined types are fine.

1. Open Workflow Advisor and select “Run” to define the input types.
2. Click on Fixed-Point Conversion and select run this task. This process may take awhile. You may need to add the DC.prn file to your MATLAB directory to get the test bench file to fully compile.

As in the previous labs, not all of the automatic selections are sufficient for our FPGA design; therefore several of the conversions will need to be modified.

1. Using the function dropdown menu at the top of the HDL Code Generation screen, select each of the functions in the design and modify them with the same settings used in Lab 4.
2. Once all modifications have been made, select “Validate Types” in the top right area of the top toolbar to verify the design for your modified Fixed-Point conversions.
3. Click on HDL Code Generation and modify the settings according to the previous labs. **This project requires pipelining** to meet timing constraints just as in lab 4. Configure these settings using lab 4 as a reference. Your pipelining settings should look similar to Figure 1-2 below.



1‑2: Modified optimization settings to allow for register pipelining

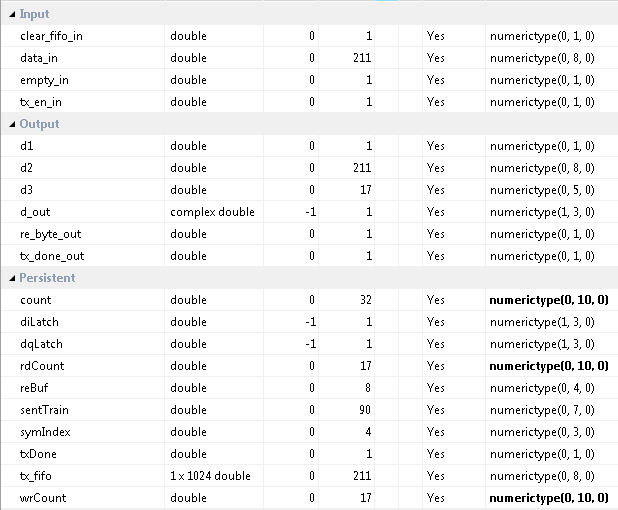
1. Once created, **copy the black box into the rx.slx model**. Both of these MATLAB algorithms cores will be exported using the same Simulink model.
2. Copy the qpsk\_rx\_FixPt\_xsgbbxcfg.m file into your Sysgen folder just as in the previous labs.
3. Copy your vhd files into the hdl directory. Make sure you modify the previously copied m file to point to the new location of the vhd files.

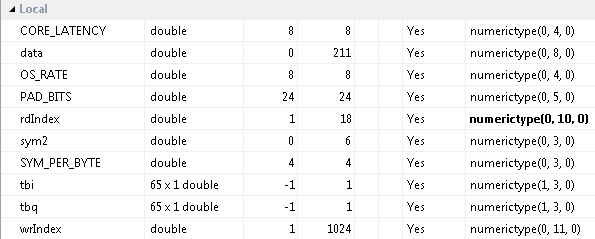
## 1.5 TX HDL Coder Project

The Last project will be qpsk\_tx.prj. Add the MATLAB function qpsk\_tx.m and the test bench script qpsk\_tb.m to the **MATLAB Function** and **MATLAB Test Bench** categories respectively.

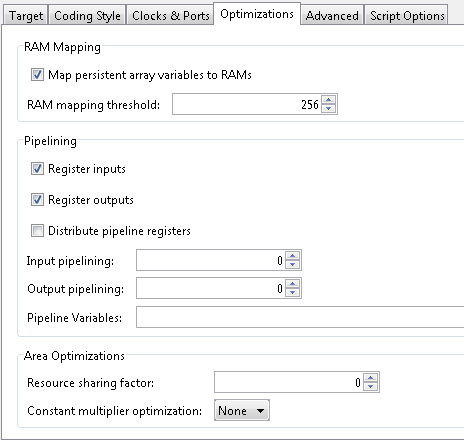
For the inputs listed for the qpsk\_tx function, the auto-defined types are fine.

1. Open Workflow Advisor and select “Run” to define the input types.
2. Click on Fixed-Point Conversion and select run this task. This process may take awhile.
3. Again, several of the conversions will need to be modified. Using the function dropdown menu at the top of the HDL Code Generation screen, select each of the functions in the design and modify them with the same settings used in Lab 3, shown in Figure 1-3 and 1-4 below.





1. Once all modifications have been made, select “Validate Types” in the top right area of the top toolbar to verify the design for your modified Fixed-Point conversions.
2. Click on HDL Code Generation and modify the settings according to the previous labs. **Pipelining is required for this project**, and the settings are shown in Figure 1-3 below. Right Click and select run this task to generate your Xilinx Block Box Design.



1‑3: Modified optimization settings to allow for register pipelining

1. Once created, copy the black box and System Generator blocks to a new Model just as in the previous labs.
2. Save the new model as tx.slx into the sysgen directory “Lab\_9\sysgen”.
3. Copy the qpsk\_rx\_FixPt\_xsgbbxcfg.m file into your Sysgen folder.
4. Copy your vhd files into the hdl directory. Make sure you modify the previously copied m file to point to the new location of the vhd files.

Export Simulink models Step 2

This section will show you which of the previously created Simulink Models will be used for this lab. It is assumed you have completed the previous labs and have already created the needed Simulink models.

## 2.1 Create MCU Simulink Design

The **Simulink model[[1]](#footnote-1)** in Figure 2-1 will be used for the control signals to and from the **MCU**.

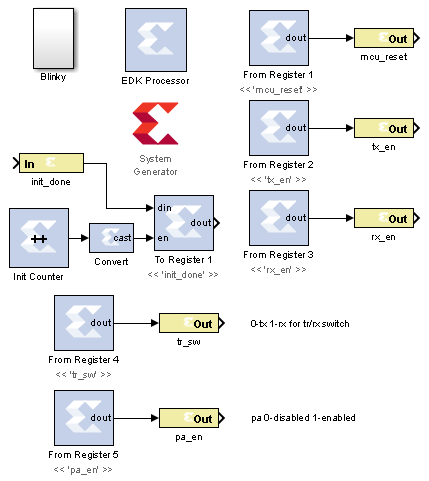


Figure 2‑1: Simulink model for MCU control

1. **Configure** this model and the system generator the same as in Lab 1, and **save** the design into the Sysgen folder. Name the file **mcu.slx** or something similar.

## 2.2 Create ADC driver Simulink Design

The **Simulink model[[2]](#footnote-2)** In Figure 2-2 will be used for creating the signals which drive the **ADC** on Chilipepper.



Figure 2‑2: Simulink model for ADC control

1. **Configure** this model and the system generator the same as in Lab 1, and **save** the design into the Sysgen folder. Name the file **mcu.slx** or something similar.

## 2.3 Create DAC driver Simulink Design

The [Simulink model](https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab_3/sysgen)**[[3]](#footnote-3)** In Figure 2-3 will be used for creating the signals which interface to the DAC. This is the same model used in Lab 2 and 3.

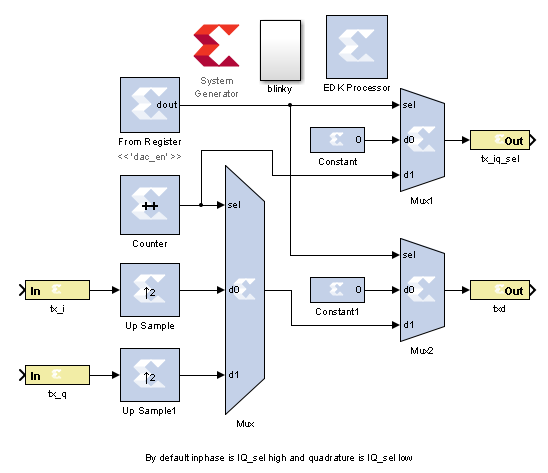


Figure 2‑3: Simulink model for DAC control

1. **Configure** this model and the system generator the same as in previous labs, and **save** the design. Name the file **dac\_driver.slx** or something similar using the appropriate directory structures.

## 2.4 Create Transmitter Simulink Design

The **Simulink model[[4]](#footnote-4)** In Figure 2-4 will be used for transmitting the data from the DAC to the receiver core on the other board.

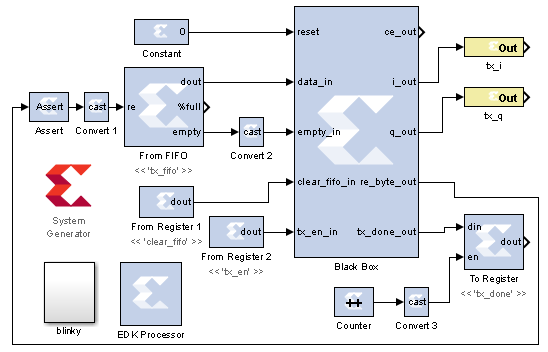


Figure 2‑4: Simulink model for receiving DC Offset output

1. This model is the same one used in Lab 3 without the ChipScope model. **Configure** this model and the system generator the same as in the previous lab, and **save** the design. Name the file **dac\_driver.slx** or something similar using the appropriate directory structures.

## 2.5 Create Receiver Simulink Design

The last **Simulink model** [[5]](#footnote-5)needed for this FPGA design is the receiver design which includes both the rx.slx and dc\_offset.slx files from previous labs. The Model and its sub parts are shown in Figures 2.5 2.6 and 2.7.

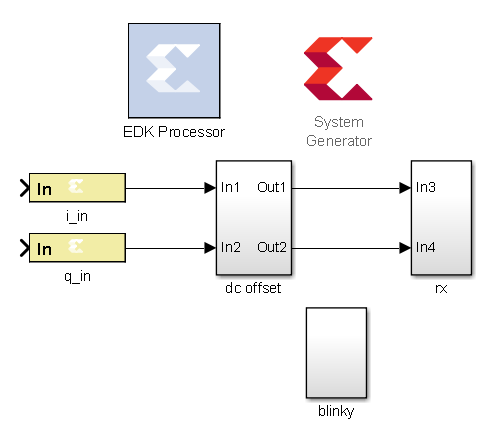
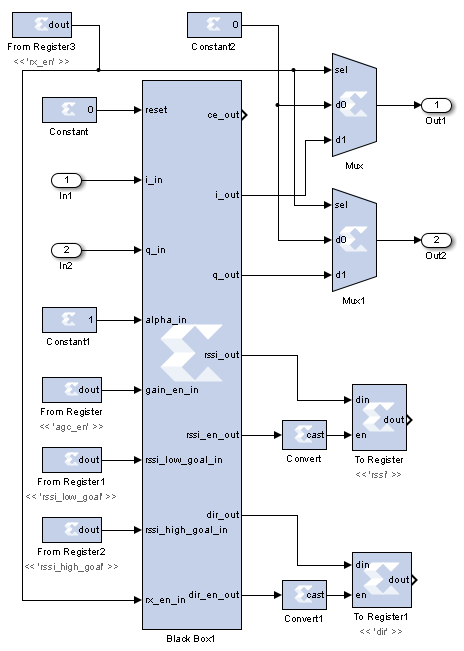
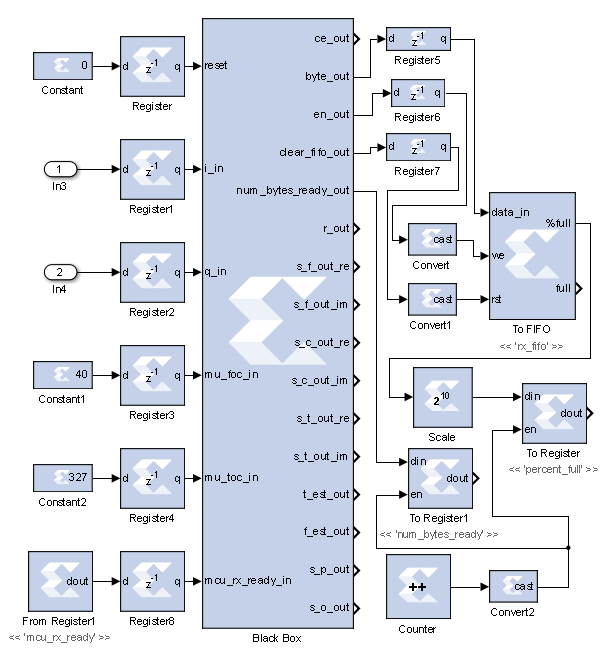


Figure 2‑5: Simulink model for receiving ADC output and processing QPSK signal



2‑6: Simulink model for performing dc offset correction.



2‑7: Simulink design for processing the received QPSK waveform.

1. **Configure** this model and the system generator the same as in the previous labs, and **save** the design. Name the file **rx.slx** or something similar using the appropriate directory structures.

**Don’t forget to sync your memory using EDK before exporting your designs!**

Configure Cores and Export Design Step 3

This section will show you how to integrate your PCores into your FPGA design using EDK. There are several components that must be configured for the design of this project. A quick list of the cores needed is given below. Refer to lab 0 sections 4.3 and 5.1 for information on how to add cores to the design.



## 3.1 Needed IP Cores

* ADC Driver PCore created in Simulink
* MCU PCore created in Simulink
* rx PCore created in Simulink (includes DC offset)
* tx Pcore created in Simulink
* DAC Driver PCore created in Simulink
* Clock Generator IP Core
* Processing System IP Core
* AXI Interconnect IP Core
* GPIO Cores for LEDs
* AXI\_UART (Lite) Core

In addition, several of these cores will require external ports. Be sure that you have access to modifying the external port settings. Refer to Figure 3-1 Below.

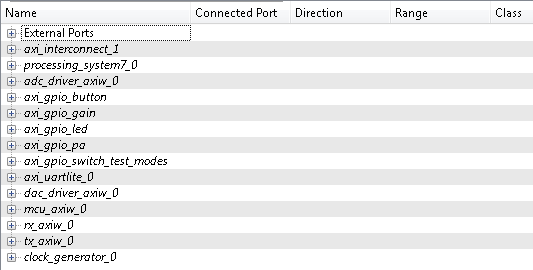


Figure 3‑1: EDK project ports list

## 3.2 Configuring the Pcore Ports

Each of the ports used in this lab can be configured exactly as the previous labs design. Refer to Lab 5 (RX) and lab 3 (TX) for more information on individual port configuration.

## 3.3 Configuring the GPIO Ports

There are several GPIO cores that are used in this lab to help with program verification. The required cores and their configurations are given below.

1. Add a GPIO core for initiating packet transmission via button press. Configure it as Channel 1 only, with a bit width of just 1 bit. Call it axi\_gpio\_button or something similar. Expand the port, right click the GPIO\_IO\_I input pin, and select “Make External”. Configure the others as “no connection”.
2. Add a GPIO core for setting the LEDs with the gain information. This is useful to determine what the approximate value of your current gain is set to. Configure it as Channel 1 only with a bit width of 6 bits. Call it axi\_gpio\_gain or something similar. Expand the port, right click the GPIO\_IO\_O input pin, and select “Make External”. Configure the others as “no connection”.
3. In addition to the LEDs for setting the gain value, this lab will have 2 general purpose LEDs that can be used for program debugging if necessary. Configure these LEDs the same as in the previous labs, 2 channels each with a bit width of 1 bit.
4. The next GPIO used will be an input for the user to manually set the transmit gain using the FPGA switches. Configure this input as single channel with a 5 bit width (input only), and name it axi\_gpio\_pa or something similar. Expand the port, right click the GPIO\_IO\_I input pin, and select “Make External”. Configure the others as “no connection”.
5. The last GPIO used for this lab is for changing the mode of the demo from continuous transmit to button transmit to normal operation etc. Configure this core as a single channel with a width of 2 bits. Call it axi\_gpio\_switch\_test\_modes or something similar.

Once all of your cores have been added, you design should look similar to Figure 3.1 shown above.

## 

## 3.4 Configuring the Clock Generator IP Core

The Clock Generator is used in this project to distribute the appropriate clock signals to each of the PCores, as well as any external hardware which may require a clock signal. For this project, the Clock Generator is sourced from the 40 MHz pll\_clk\_out on the Chilipepper radio board (as described in the **Chilipepper user’s guide**).

1. **Double click** the Clock Generator PCore and **configure** the settings as follows

* Input Clock Frequency of **40Mhz**
* CLKFBIN Required Frequency of **40Mhz** with **no Clock Deskew**
* CLKFBOUT Required Frequency of **40Mhz**, Required Group **PLLE0**, and **Buffered True**
* CLKOUT0 Required frequency of **20Mhz**, 0Phase, **PLLE0** group and **Buffered true**
* CLKOUT1 Required Frequency of **40MHz**, 0Phase, **PLLE0** group and **Buffered true**
* CLKOUT2 Required Frequency of **40MHz**, **90 Phase**, **PLLE0** group and **Buffered true**

Now that the settings are configured you should have several clocks in your clock generator list.

1. **Connect** the pins according to the following.

* CLKIN External Ports
* CLKOUT0 mcu::sysgen

rx\_axiw::Sysgen

tx\_axiw::sysgen

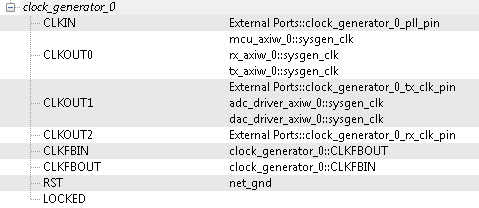
* CLKOUT1 Ports::clock\_generator\_0\_tx\_clk

dac\_driver::Sysgen\_clk

adc\_driver::sysgen\_clk

* CLKOUT2 External Ports::clock\_generator\_0\_rx\_clk\_pin
* CLKFBIN CLKFBOUT
* RST net\_gnd
* LOCKED

Your Clock Generator port should look similar to Figure 3-2 below.



3‑2: Clock Settings for QPSK radio Lab EDK project

## 3.5 Pin Assignments

Once the ports are configured correctly, the sysgen clock for the cores should be set as well. The last step is to setup the **pin assignments** for the external ports.

1. Rename the pins of the external ports so they are easily identifiable. Appendix B shows the names used in this demo, however you don’t have to use the same naming convention.
2. Open the **Project** tab.
3. Double-click on the **UCF File: data\system.ucf** from this panel, to open the constraints file.
4. Fill in the pin out information for your design using Appendix B as a reference.
5. Prior to EDK version 14.4, Xilinx had a [documented issue](http://www.xilinx.com/support/answers/51739.htm)[[6]](#footnote-6) with AXI-bus generation for Simulink PCores targeting the Zynq FPGA. Refer to this issue for more information. As in Lab 0 section 5.2, this bug must be corrected if your **EDK version** is **14.3 or lower**. The steps to perform are identical to those in the previous labs; however they must be performed for **all** of the PCores used in this lab.
6. Select the **Export Design** button from the navigator window on the left. Click the **Export and Launch SDK** button. This process may take awhile.

Create software project Step 4

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 Sysgen and EDK folders. Doing this will keep all relevant files in the same location.



## 4.1 Creating a new C Project

This section will show you how to create a C program to test your FPGA design. The primary objectives of this program are to add processor integration on both the transmitter and receiver cores. Additionally, this code will add some Layer two features to allow for packet id and acknowledgment.

It would be helpful if you have completed the Embedded System Design tutorial in the *ZedBoard AP SoC Concepts Tools and Techniques Guide*. Refer to Lab 1 for more information on the MCU signal control using C code within SDK.

**Note**

1. Select **File 🡪 New 🡪 Project**.
2. Select **Xilinx 🡪 Application Project,** and hit next.
3. Name the project qpsk\_example or something similar and leave the other settings at their defaults. Hit next.
4. Select **Hello World** from the **Select Project Template** section.
5. Click **Finish**. You should now see your hello\_world.c file in your project src folder, as well as a **board support package** (bsp) folder. 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 Appendix A.

****

If your SDK project cannot find your include files such as “xuartlite.h” and “xparameters.h” you can add these include files manually by selecting Xilinx Tools from the top menu and selecting Repositories. Under Global Repositories, click New and select your EDK project directory. See [here](http://www.xilinx.com/support/answers/35443.htm) [[7]](#footnote-7)for more information

## 4.2 Adding Supporting files

In addition to the main c file, you need the library files for the Chilipepper board. The 2 required files for this Lab are Chilipepper.c and Chilipepper.h and can be found on the GitHub[[8]](#footnote-8) repo. Place these files in the src directory of your project workspace.

1. [Chilipepper.c](https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_8/workspace/qpsk_rx/src) – This file is the primary library file for the Chilipepper board. It contains functions for modifying the MCU registers as well as basic helper functions for tasks such as initialization, transmitting, and receiving.
2. [Chilipepper.h](https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_8/workspace/qpsk_rx/src) – This file holds the function prototypes for the Chilipepper.c functions.

In addition to the Library files, you also need to include a Math library which contains the pow function that is used when creating the CRC. See Lab 3 section 4.1 for more information on how to add the Math Library to your project.

**Note**

The Chilipepper.c library file is configured for both TX and RX cores as well as a UART to talk to the on board MCU and configure its settings. To use the library file properly, you must specify which of these features you will use. To do this, modify lines 8-13 of the Chilipepper.c file to define a variable for the cores you will be using. Your code should resemble the following, as we will be using all cores for this Lab.



****

If you are still not able to compile your C design due to include errors, you may need to tell SDK where your PCore drivers are stored. If you click on Xilinx Tools **🡪** Repositories, you can specify (in Global Repositories) where the EDK directory of your project is. (This will need to be changed for each new project)

## 4.3 Loading Hardware Platform with iMPACT

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. For this lab, you can verify your design by connecting two Chilipepper boards together using an attenuator.

1. Connect the Chilipepper to the FPGA board and verify all cables are connected properly and the jumper settings are correct. Verify this by using the *Chilipepper user guide* and the *ZED Board Hardware users guide* as a reference. Also See Lab 0 for details on Jumper Configuration.
2. Once the FPGA and radio board are connected correctly, turn on the board.
3. Open iMPACT in the ISE Design tools.
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 4-2 below.
9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.
10. Repeat this process for each board you would like to test.



4‑2: iMPACT configuration screen

## 4.4 Debugging with SDK

If the hardware design is correct, you should see the LEDs on the board light up, as well as a blue light 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 project name folder and selecting **Debug As** 🡪 **Launch on Hardware**.
2. You should now be taken to a screen which shows the first pointer initialization as highlighted. You can now start the software program by clicking the  (play) button in the top menu.

Testing and Design Verification Step 5

## 5.1 Verification with Terminal

Once you have both labs running successfully, the next step is to verify functionality by connecting the FPGAs which are running your Lab 8 design to terminals to view the TX and RX QPSK character packet.

1. Connect your FPGA to the PC using a micro USB cable. The cable should be plugged into the UART port on the FPGA, shown in Figure 5-1 below.



Figure 5‑‑: Circled in this figure is the UART port of the Xilinx Zed Board FPGA.

1. With the board powered on, open up a hyper terminal window such as Tera Term, and configure it with the following settings

**Baud: 115200; Data: 8 bit; No Parity;1 Stop bit; No Flow Conrol.**

Notice that the baud rate used is configured in the SetupPeripherals function in our main.c function created earlier. The other settings are all defaults for the XUartPs port.

1. Once the terminal is configured, you should be able to view your packets by clicking the button on the Lab 3 Demo. In addition, if you flip the switch for continuous packet transmission, you shoud see several hello world packets on your terminal output.
2. main.c

SDK function main.c



1. UCF file

EDK UCF file system.ucf





1. This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab\_9/sysgen [↑](#footnote-ref-1)
2. This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab\_8/sysgen [↑](#footnote-ref-2)
3. This model can be downloaded from https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab\_3/sysgen [↑](#footnote-ref-3)
4. This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab\_9/sysgen [↑](#footnote-ref-4)
5. This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab\_8/sysgen [↑](#footnote-ref-5)
6. Issue can be found at http://www.xilinx.com/support/answers/51739.htm [↑](#footnote-ref-6)
7. See http://www.xilinx.com/support/answers/35443.htm [↑](#footnote-ref-7)
8. <https://github.com/Toyon/Chilipepper> [↑](#footnote-ref-8)