|  |
| --- |
| Toyon Research Corporation |
| Lab 4: Carrier Recovery |
| Chilipepper Tutorial Projects |

|  |
| --- |
| Version 0.1  2/6/2013 |

Table of Contents

[Introduction 3](#_Toc345686907)

[Procedure 3](#_Toc345686908)

[Objectives 3](#_Toc345686909)

[Generate HDL Code 4](#_Toc345686910)

[1.1 MATLAB Functions 4](#_Toc345686911)

[1.2 MATLAB Test bench 9](#_Toc345686912)

[1.3 HDL Coder Project 11](#_Toc345686913)

[Create and Export Simulink Models 13](#_Toc345686914)

[2.1 Modify Tone Simulink Design 13](#_Toc345686915)

[2.2 Create MCU Simulink Design 15](#_Toc345686917)

[2.3 Create DAC driver Simulink Design 16](#_Toc345686918)

[Configure Cores and Export Design 18](#_Toc345686919)

[3.1 Needed IP Cores 18](#_Toc345686921)

[3.2 Configuring the TX Port 19](#_Toc345686922)

[3.3 Configuring the DAC Driver Port 19](#_Toc345686923)

[3.4 Configuring the MCU Port 19](#_Toc345686924)

[3.5 Configuring the Clock Generator IP Core 19](#_Toc345686925)

[3.6 Pin Assignments 21](#_Toc345686926)

[Create software project 23](#_Toc345686927)

[4.1 Creating a new C Project 23](#_Toc345686929)

[4.2 Debugging with SDK, iMPACT and ChipScope Pro 24](#_Toc345686930)

[4.3 Debugging with SDK 25](#_Toc345686931)

[Testing and Design Verification 26](#_Toc345686932)

[5.1 Verification with ChipScope Pro 26](#_Toc345686933)

[5.2 MATLAB Analysis 28](#_Toc345686934)

[Appendix A MATLAB Core Functions 30](#_Toc345686935)

[Appendix B MATLAB Test Bench 33](#_Toc345686936)

Lab 4: Carrier Recovery

# Introduction

This lab will show you how to recover your QPSK Waveform generated in Lab 3 using a second Xilinx Zed Board FPGA and Toyon Chilipepper FMC. The Analog to Digital Conversion (ADC) used to receive the signal will take place on the Chilipepper board. The FMC initialization and microcontroller (MCU) signal control will be handled in software using the Xilinx Software Development Kit (SDK). Finally, the testing of results will be done using ChipScope and MATLAB. This lab assumes prior knowledge of the workings of HDL Coder as well as the Xilinx EDK environment. It is recommended that you complete the previous labs before completing this lab.

This lab is created using:

* MATLAB 2012b
* Xilinx ISE Design Suite 14.3 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:

* 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 implement an ADC
* Receive a QPSK Waveform using the Chilipepper FMC
* Create a software application to test your design
* Verify your results in ChipScope and analyze them using MATLAB

Create and export Simulink models Step 1

This section will show you how to create and customize your Simulink models to properly receive a signal and control the Chilipepper MCU and ADC. For additional information on this process, see lab 2.

## 1.1 Create MCU Simulink Design

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



Figure 1‑1: Simulink model for MCU control

1. To create a new Simulink model, open MATLAB and click on the **Simulink Library** button in the Home menu.



1. Select **File 🡪 New 🡪 Model**
2. **Configure** this model and the system generator the same as in Lab 1, and **save** the design into the Sysgen folder. **Be sure to change the cfg file as well to find the files in your new directory structure.** Name the file **mcu.slx** or something similar.

## 1.2 Create ADC driver Simulink Design

The **Simulink model[[2]](#footnote-2)** In Figure 1-2 will be used for creating the signals which drive the **ADC** on Chilipepper.

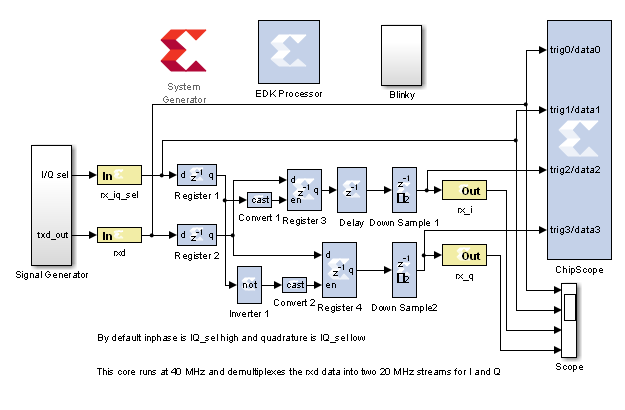


Figure 1‑2: Simulink model for ADC control

1. **Create** a new Simulink model and add the components from the Simulink blockset.
2. The white box labeled “Blinky” is simply a subsystem of the **Counter** **Slice** and LED **Gateway Out** blocks. The Blocks used for this subsystem are shown in Figure 1-3. Configure the Blinky subsystem identically to the other LED out systems in the previous labs.



Figure 1‑3: Blinky Subsystem

1. The **Signal Generator** subsystem shown in Figure 1-4 is used to test the functionality of the ADC driver.

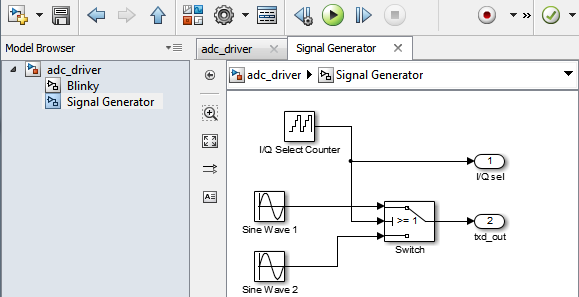


Figure 1‑4: Signal Generator Subsystem

All of the blocks used in the **Signal Generator subsystem** should come directly from the Simulink library blockset. This blockset is useful for testing your design in Simulink; however the blocks are dropped from the PCore design when exporting into EDK.

**Note**

1. Run and test your ADC design using the same method used in Lab 2. Once you have verified it is correct, you are ready to create your EDK project and Export your models as PCore IP!

Refer to Lab 0 Step 3 to **Create a New Blank EDK Project**. Be sure to follow the directory structure used. Once your project is created, **export** each model 1 by 1 into the newly created EDK project. Be sure your **Compilation Settings** are correct as shown in Lab 0 Section 4.1. Once each Simulink model has been exported successfully, you’re ready to configure your FPGA design.

Configure Cores and Export Design Step 2

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.

## 2.1 Needed IP Cores

* ADC Driver PCore created in Simulink
* MCU PCore created in Simulink
* Clock Generator IP Core
* Processing System IP Core
* AXI Interconnect IP 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 2-1 Below.

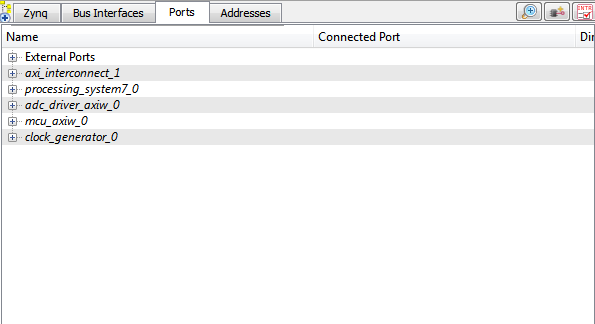


Figure 2‑1: EDK project ports list

## 2.2 Configuring the ADC Driver Port

Expand the **ADC Driver** port. There are 6 individual I/O pins which need to be routed on this port.

1. The first three are the rx\_iq\_sel, the rxd and the bliky\_adc\_driver pins. Each of these pins can be assigned as **External ports**.
2. Next are the rx\_i and the rq\_q output pins. There is no further logic required to connect these pins to, therefore they can be left **unconnected**. For this lab, we will only monitor these pins in software using the ChipScope previously configured in Simulink.
3. The sysgen\_clk pin can be skipped for now and will be connected later in **section 2.4**

## 2.3 Configuring the MCU Port

Expand the **MCU** port. There are 8 individual I/O pins which need to be routed on this port.

1. Configuring this port is very simple as all of the pins with the exception of the sysgen\_clk are assigned as E**xternal ports**.

## 2.4 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**). This signal is then distributed to 3 other devices; 2 PCores (MCU and ADC) and the RX\_CLK signal which latches data from the RXD line to the ADC on the radio board.

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 **40MHz**, **90 Phase**, **PLLE0** group and **Buffered true**
* CLKOUT1 Required Frequency of **40MHz**, 0Phase, **PLLE0** group and **Buffered true**
* CLKOUT2 Required frequency of **20Mhz**, 0Phase, **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 External Ports
* CLKOUT1 adc\_driver::sysgen\_clk
* CLKOUT2 mcu::sysgen
* CLKFBIN CLKFBOUT
* RST net\_gnd
* LOCKED External Port

The CLKOUT0 pin has a 90 degree phase shift as mentioned in the **Chilipepper user’s guide**. This line will be used as the RX\_CLK signal from the FPGA to the radio board.

**Note**

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

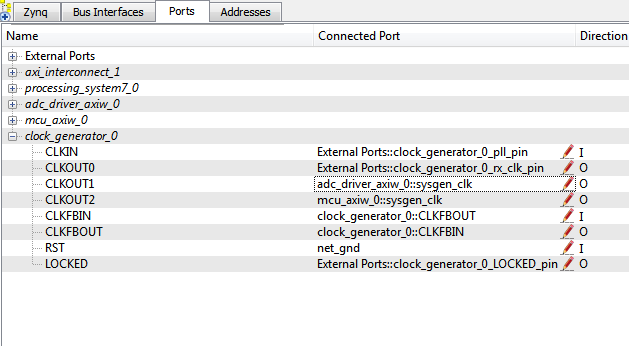


Figure 2‑2: Clock Generator port configuration

## 2.5 Pin Assignments

Once the clock generator is configured correctly, the sysgen clock for the other 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. Figure 2-3 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 Figure 2-3 below as a reference.



Figure 2‑3: EDK project pin assignments

****

Be sure that the **orientation** of the RXD pins is set correctly. If you follow the pin list in the figure above, you must **reverse** the RXD pins in the external ports assignment section. This is done using the same method used in Lab 0 Section 5.2 for the LEDs.

At the time of this tutorial, Xilinx had a [documented issue](http://www.xilinx.com/support/answers/51739.htm)[[3]](#footnote-3) 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 for our project. The steps to perform are identical to those in the previous labs; however they must be performed for **both** of the PCores used in this lab.

Once the fix is applied, you’re ready to generate your bitstream file! 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 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 Sysgen 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 test your receive tone project. Since our logic for receiving the signal is handled by PCores, all we need to do in software is initialize the hardware.

1. Select **File 🡪 New 🡪 Project**.
2. Select **Xilinx C Project,** and hit next.
3. Name the project hello\_world and leave the other settings at their defaults. Be sure to select **Hello World** from the **Select Project Template** section.
4. Click **Finish**. You should now see your hello world project folder, as well as a **board support package** (bsp) folder.
5. If you navigate into the hello world project folder, and into the src folder, you should see a helloworld.c file. This is the file we will be using to create our software design.
6. **Double click** the file to open it and **replace** all of its contents with the code in Figure 3-1.

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**

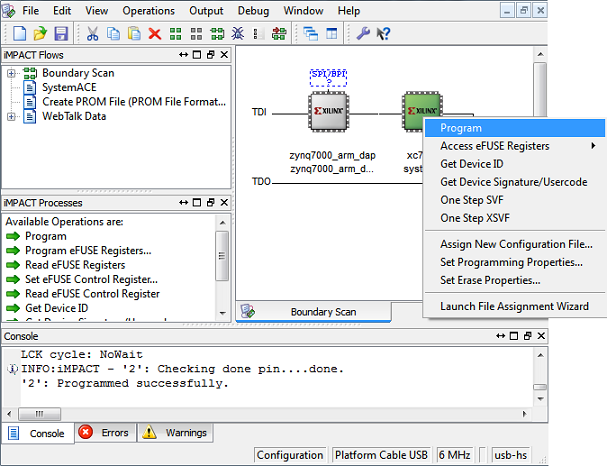


Figure 3‑1: Code outline for SDK project

## 3.2 Debugging with SDK, iMPACT and ChipScope Pro

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.

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 3-2 below.
9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.



3‑2: iMPACT configuration screen

## 3.3 Debugging with SDK

If the hardware design is correct, you should see the LEDs start blinking on the board, 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 hello\_world project 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.

If the software initialization worked, you should see a green light on the Chilipepper.

****

Verify that the **orientation** of the TXD pins is set correctly. If you follow the pin list in the figure above, you must **reverse** the TXD pins in the external ports assignment section. This is done using the same method used in Lab 0 Section 5.2 for the LEDs.

At the time of this tutorial, Xilinx had a [documented issue](http://www.xilinx.com/support/answers/51739.htm)[[4]](#footnote-4) 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 for our project. The steps to perform are identical to those in the previous labs; however they must be performed for **both** of the PCores used in this lab.

Once the fix is applied, you’re ready to generate your bitstream file! 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 and Test 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

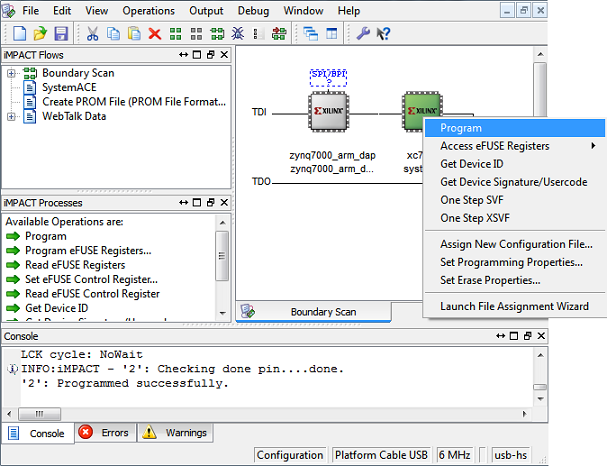
This section will show you how to create a C program to test your QPSK project. There are several source files[[5]](#footnote-5) which are required to support your software application. Some of the files are created when you create your new C project and do not need to be modified. A list of the files which are not created automatically and/or need to be modified is listed below with a link to download them.

1. [helloworld.c](https://github.com/rcagley/Chilipepper/blob/master/Labs/Lab%203/workspace/hello_world/src/helloworld.c) – This file is created when you create your Hello World template project. (see previous labs). It should be modified to look similar to the file found on the website.
2. [Chilipepper.c](https://github.com/rcagley/Chilipepper/blob/master/Labs/Lab%203/workspace/hello_world/src/Chilipepper.c) – This file needs to be created. There are three main functions associated with this file. The first function is Chilipepper\_Initialize, which is handled similarly to the previous labs. The next is Chilipepper\_AppendCrc which handles adding the CRC to the end of the packet. The last important function within this file is Chilipepper\_WriteTestPacket which handles creating a packet to transmit based on a user input string.
3. [Chilipepper.h](https://github.com/rcagley/Chilipepper/blob/master/Labs/Lab%203/workspace/hello_world/src/Chilipepper.h) – This file holds the function prototypes for the Chilipepper.c functions.
4. [xbasictypes.c](https://github.com/rcagley/Chilipepper/blob/master/Labs/Lab%203/workspace/hello_world/src/xbasic_types.c) – This file defines some assert functions which can be used to assist in troubleshooting your software application.
5. [Xbasictypes.h](https://github.com/rcagley/Chilipepper/blob/master/Labs/Lab%203/workspace/hello_world/src/xbasic_types.h) - This file is needed to define some of the variable types used throughout the code.
6. In addition to the files given, you also need to include a Math library which contains the pow function that is used when creating the CRC.
   1. Right click on your project, and select C/C++ Build Settings.
   2. Under ARM gcc linker, select Libraries.
   3. Click the  button in the Libraries (-l) section, and type ‘m’ then hit ok. This adds the math library needed to use this function. Hit apply and Ok to return to your project.

## 4.2 Debugging with SDK, iMPACT and ChipScope Pro

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.

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 4-1 below.
9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.



4‑1: iMPACT configuration screen

## 4.3 Debugging with SDK

If the hardware design is correct, you should see the LEDs start blinking on the board, 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 hello\_world project folder and selecting **Debug As** 🡪 **Launch on Hardware**.
2. You should now be taken to a screen which shows the first 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 a green light on the Chilipepper.

Testing and Design Verification Step 5

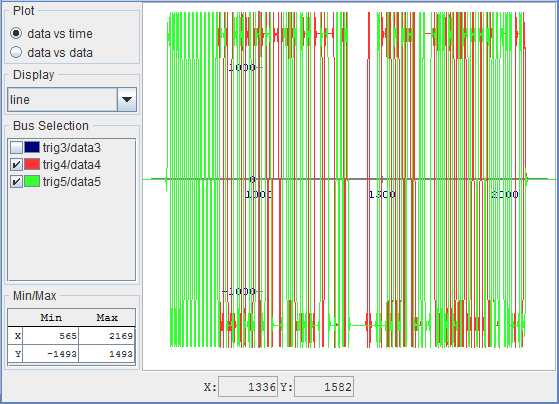
## 5.1 Verification with ChipScope Pro

There are several methods available for verifying the QPSK transmission. This lab focuses on verification using ChipScope Pro, as well as exporting to MATLAB for further analysis. In addition, if you have multiple boards available, you can also verify the transmission by receiving the QPSK packet on your second board (more is covered on this topic in future labs).

1. To verify the received signal, you will need to open **ChipScope Pro Analyzer**. Be sure that the JTAG cable is connected to the FPGA board properly (or the 6 pin output of the Chilipepper).
2. Once the program opens, click the  (open cable) button to open your JTAG connection to the board. If your jumpers are configured correctly, you should see the following devices on the cable.



1. Select ok to get to the Analyzer main screen. Open the file menu and select **Import**.
2. Click **Select New File**, and browse to the location of your ChipScope **CDC file**, which is located in the Sysgen/netlist folder of your project directory. This file was created for you when you generated your PCores from your Simulink Model design. It tells the ChipScope program how to interpret the data it is receiving from the JTAG port.
3. Next double click on the **bus plot** option in the New Project menu in the top left hand side of the screen. This will open a window which allows you to view a signal **value vs. time** plot of your waveforms.
4. Under Data Port in the Signals Dev menu on the left side of the screen, right click on the trig4/data4 and trig5/data5 ports, and change their **bus radix** to **signed decimal**. Click OK to accept the default decimal values.
5. On the Bus Plot screen, you can change the color of each of the signals to get a better view of each individual signal. Click the **check box** next to any of the signals you wish to see on the plot.
6. Click the **play button** in the top menu bar to display the signal. Additionally you can set up triggering options for periodic or continuous playback of the received signal. Your received signal should look similar to Figure 5-1 below.



5‑1: both i and q channels of the QPSK waveform in ChipScope Pro

If you changed the message within your C program, your waveform may look slightly different than the one shown here, however you should have the same header padding which can be seen at the front portion of the waveform.

## 5.2 MATLAB Analysis

Now that you have verified the transmitted signal, you can get a pretty good idea of what your QPSK waveform looks like in the time domain. However, ChipScope allows you to export the data received directly into MATLAB for further analysis.

1. Before we export the data, we need to format it as best we can using the triggering options within ChipScope.
   1. To start, under **Trigger Setup**, set the **depth** of the capture to **4096**. Your depth may be longer if you created a long test message, however this should be fine for the Hello World example presented here.
   2. For this example, we will set our **trigger** to occur when **trig2/data2** is **not equal to zero**. This occurs anytime the FIFO buffer is not empty, and indicates that we are sending data to the buffer to be transmitted.
   3. In the **Match** section of **Trigger Setup**, change the triggering function for **Match** Unit **M2** to <> and the Value to one.
   4. In the **Trig** section, make sure your condition is active, and set the **Condition** equation to **M2**.
2. Now when you hit play, your QPSK data should be near the beginning of your capture.

The MATLAB test bench function will search for any QPSK signal in the variables exported. Be sure that only one broadcast of your ASCII string is within you capture of ChipScope to prevent possible data interference in MATLAB.

**Note**

1. It will be helpful later in your MATLAB code if you rename your **Data Port variables**. Right click on the **trig4/data4** and **trig5/data5** Ports, and **change the names** to something more descriptive, such as tx\_i and tx\_q respectively. If needed, you can use the Simulink model to find which signal each port has.

Confirm that the names you used for the i and q channel data ports in ChipScope match the names of the i and q channel variables in the MATLAB Test Bench script crated earlier.

**Note**

1. Open the file menu and select **Export**.
2. Click the **ASCII** radio box, select **Bus Plot Buses** under Signals to export, and then click **export**.
3. It is recommended that you save this file into the project directory with your MATLAB files. Call it something descriptive such as **TX.prn**.
4. To test the data in MATLAB, run your test bench script by first navigating to your project directory (this is necessary to load the prn file correctly).
5. Run the test bench with the sim parameter set to 0. You should see your message output with a plot in MATLAB of the QPSK waveform.

****

Experiment with other ASCII strings in your C code to verify the timing of the buffers. If you notice that your message is incomplete, it’s possible you could be overloading your FIFO buffer before you have a chance to load your message to transmit.

1. This model can be downloaded from https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab%202/sysgen [↑](#footnote-ref-1)
2. This model can be downloaded from https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab%202/sysgen [↑](#footnote-ref-2)
3. Issue can be found ay http://www.xilinx.com/support/answers/51739.htm [↑](#footnote-ref-3)
4. Issue can be found at http://www.xilinx.com/support/answers/51739.htm [↑](#footnote-ref-4)
5. The source files can be found at https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab\_3/workspace/hello\_world/src [↑](#footnote-ref-5)