

Toyon Research Corporation

# Lab 2: Receive Tone

Chilipepper Tutorial Projects

Version 0.2  
12/17/2012

## Table of Contents

Introduction .....	3
Procedure.....	3
Objectives .....	3
Create and export Simulink models.....	4
1.1 Create MCU Simulink Design .....	4
1.2 Create ADC driver Simulink Design.....	5
Configure cores and export design .....	10
2.1 Needed IP Cores .....	10
2.2 Configuring the ADC Driver Port.....	11
2.3 Configuring the MCU Port .....	11
2.4 Configuring the Clock Generator IP Core .....	11
2.5 Pin Assignments.....	13
Create software project .....	15
3.1 Creating a new C Project .....	15
3.2 Debugging with SDK, iMPACT and ChipScope Pro.....	16
3.3 Debugging with SDK.....	18
Testing and Design Verification.....	19
4.1 Verification with ChipScope Pro .....	19
4.2 MATLAB Analysis .....	20
Conclusion.....	23

# Lab 2: Receive Tone

---

## Introduction

This lab will show you how to receive a single frequency tone on an FPGA Mezzanine Card (FMC) radio board using the Xilinx Zed Board FPGA and the Toyon Chilipepper FMC. The Analog to Digital Conversion (ADC) used to receive the tone 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). This lab assumes prior knowledge of the workings of HDL Coder as well as the Xilinx EDK environment. It is recommended that you complete lab 0 and lab 1 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 control hardware on an FMC
- Create a software application to test your created FPGA hardware
- Verify your results in ChipScope or export to MATLAB

---

**Note**

Please refer to lab 0 to ensure that you have properly configured **System Generator** with your installation of MATLAB **before** starting this lab.

---

## 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 from the signal generator and control the Chilipepper MCU and ADC.

### 1.1 Create MCU Simulink Design

The **Simulink model**<sup>1</sup> in Figure 1-1 will be used for the control signals to and from the **MCU**.

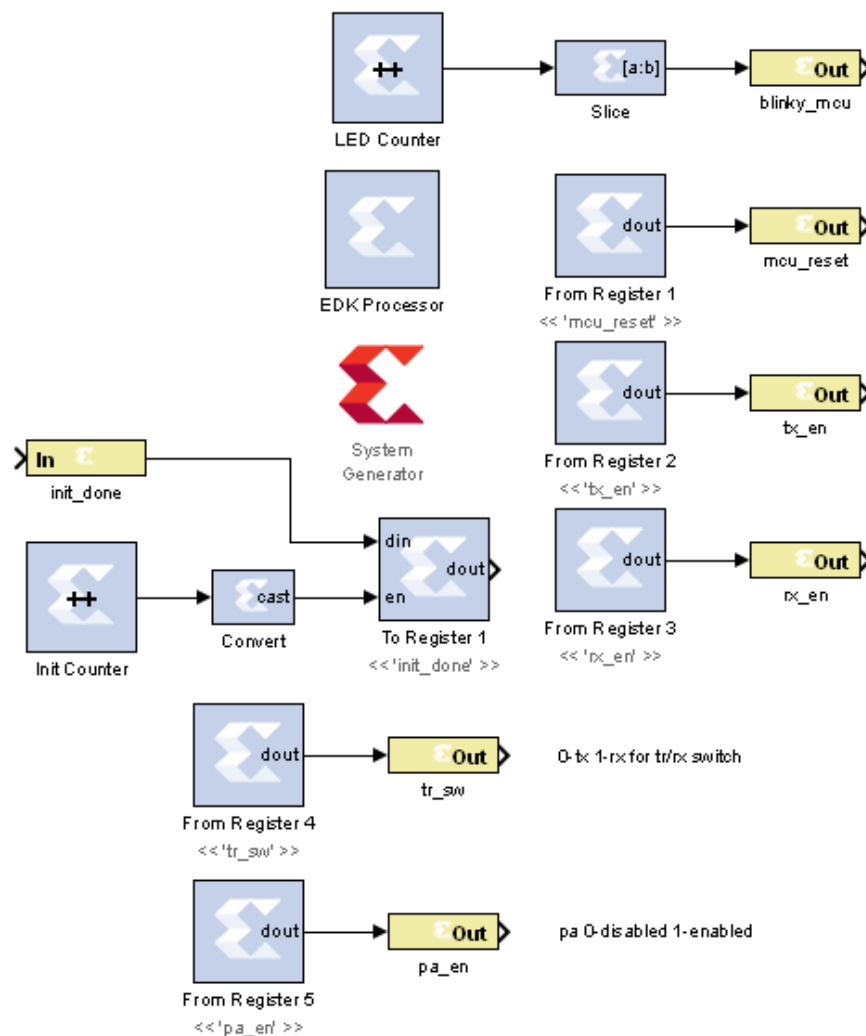


Figure 1-1: Simulink model for MCU control

<sup>1</sup> This model can be downloaded from <https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab%202/sysgen>

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

1. To create a new Simulink model, open MATLAB and click on the **Simulink Library** button in the Home menu.
2. Select **File → New → Model**
3. **Configure** this model and the system generator the same as in Lab 1, and **save** the design. Name the file **mcu.slx** or something similar using the appropriate directory structures.



## 1.2 Create ADC driver Simulink Design

The **Simulink model**<sup>2</sup> In Figure 1-2 will be used for creating the signals which drive the ADC on Chilipepper.

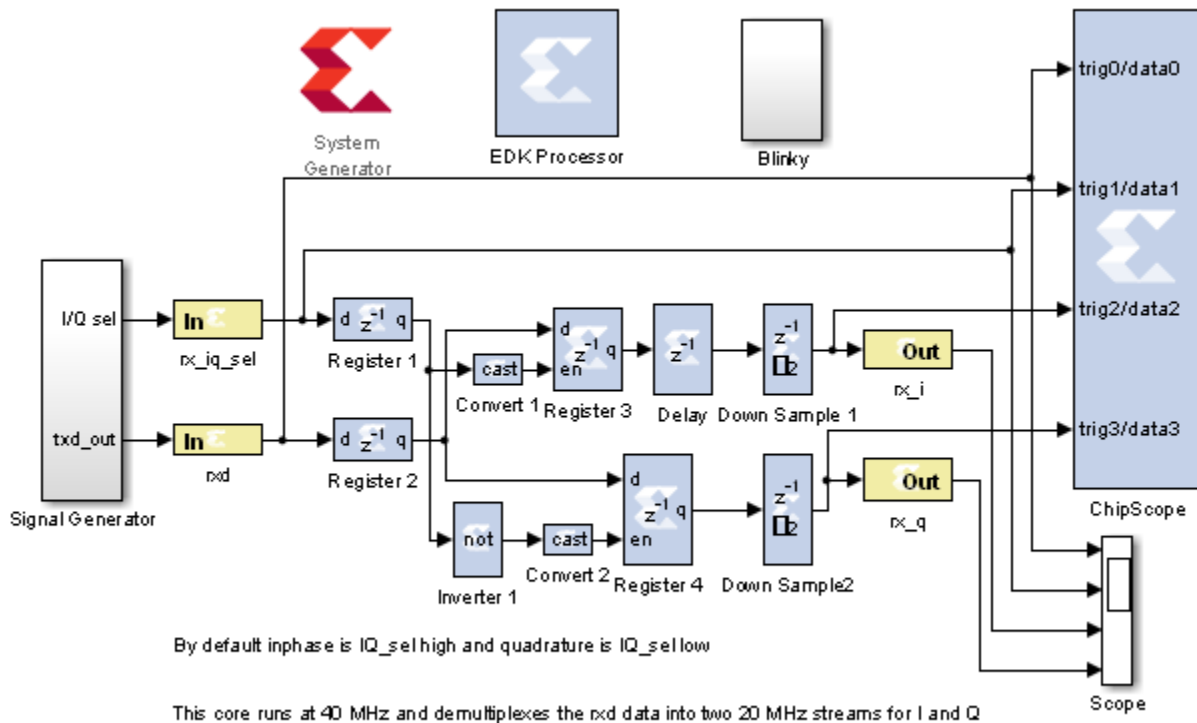


Figure 1-2: Simulink model for ADC control

1. **Create** a new Simulink model and add the components from the Simulink blockset.

<sup>2</sup> This model can be downloaded from <https://github.com/rcagley/Chilipepper/tree/master/Labs/Lab%202/sysgen>

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

- 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. You can create a subsystem from any number of blocks in your design by first adding the blocks to the design, selecting them, and pressing Ctrl+G, or right clicking and selecting **Create Subsystem from Selection**. Configure the Blinky subsystem identically to the other LED out systems in the previous labs.

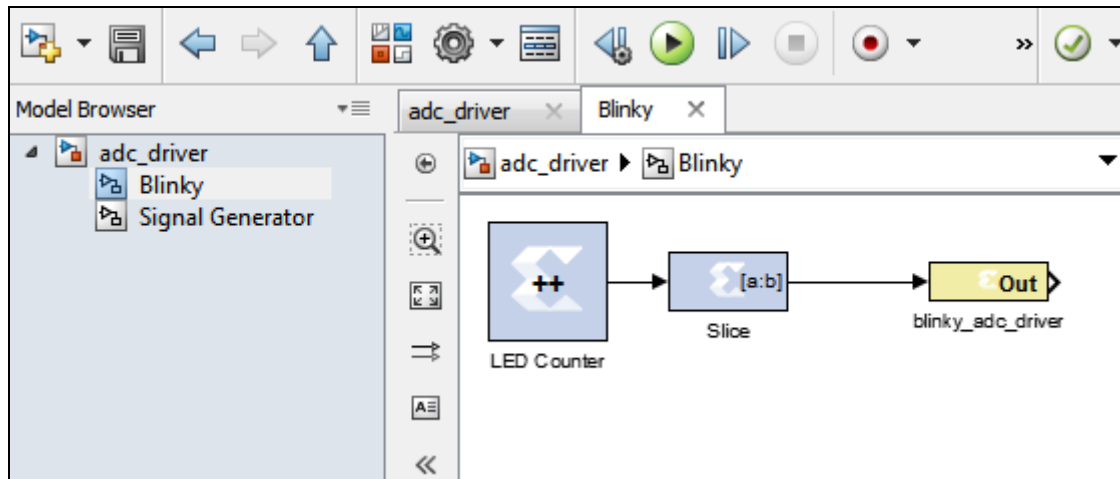


Figure 1-3: Blinky Subsystem

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

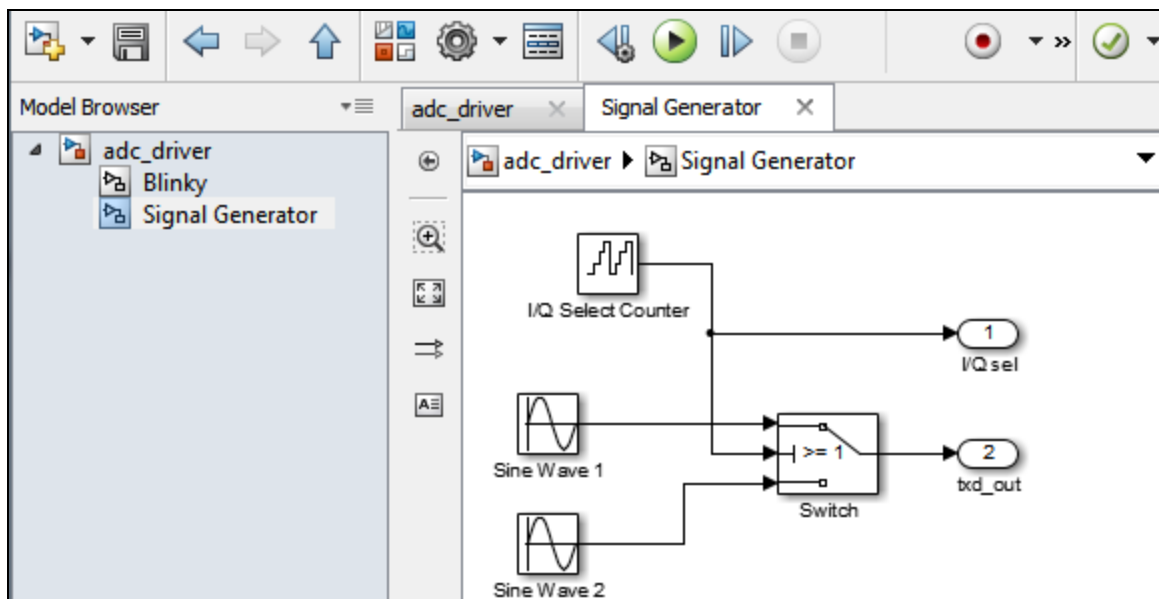


Figure 1-4: Signal Generator Subsystem


**Note**

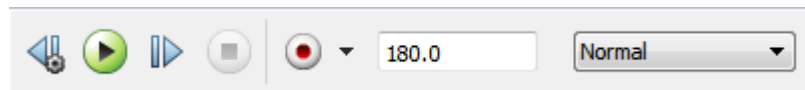
All of the blocks used in the Signal Generator subsystem should come directly from the Simulink library blockset. This blockset is useful for testing your design in Simulink; however the blocks are dropped from the PCore design when exporting into EDK.

---

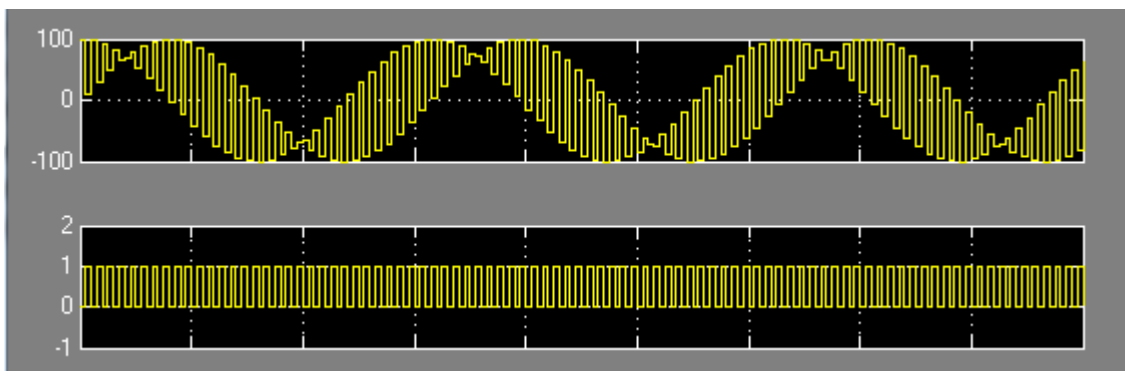
- a. The I/Q Select Counter is a **Free-Running Counter** block. Set the parameters as 1 bit with a sample period of 1.
  - b. The **Sine Wave 1** block is used to represent the in-phase portion of the test signal. It is mixed into the output whenever the I/Q Select Counter is high. Configure this block as follows: **Sine type** Time based; **Time (t)** Use simulation time; **Amplitude** 100; **Bias** 0; **Frequency** 0.1; **Phase (rad)** and **Sample time** 0; **Interpret Vector parameters as 1-D** checked.
  - c. The **Sine Wave 2** block is used to represent the quadrature portion of the test signal. It is essentially a cosine waveform, and it is mixed into the  $t_{xd}$  signal whenever the I/Q Select Counter is low. Configure this block with the same settings as the previous block with the following change: **Phase (rad)**  $\pi/2$ .
  - d. The **Switch** block does the mixing of the two sinusoids. Configure it as follows: **Criteria for passing first input**  $u2 \geq \text{Threshold}$ ; **Threshold** 1; **Enable zero-crossing detection** checked; **Sample time** -1.
4. Return to the top level of your model browser. Be sure to save your design.
  5. The **rx\_iq\_sel input gateway** is the signal that selects which of the two signals (I or Q) is currently on the  $rx_d$  line. By default when **rx\_iq\_sel** is high, the in-phase signal is on the  $rx_d$  line, and when it's low, the quadrature signal is on the line.
  6. The  $rx_d$  **input gateway** is the multiplexed input signal received by the Chilipepper board from the signal generator.
  7. The four **register** blocks are simply D Flip-Flops used to latch the value of the analog signals. They don't require any configuration settings, just be sure that the ones which require enable ports have them provided.
  8. The **not** (inverter) is just used to toggle selection of the I and Q channel outputs. No configuration settings are required.
  9. The **cast** blocks are used to convert the latched **rx\_iq\_sel** line into a boolean value to enable/disable the registers. Just be sure they are set to a boolean output type.

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

10. The **Down Sample** blocks are used to ensure that the signal which was latched into the register is held for the full duration until the next signal is ready to be latched. Since there is one input signal which has two signals mixed into one, the Down Sample block should hold each value for 2 cycles. Configure both Down Sample blocks as follows: **Sampling rate 2**; **Sample** Last value of frame; **Provide enable port** checked; **Latency 1**.
11. The **Delay** block is used to ensure that even though the inputs latch at different times, the outputs are placed on the I and Q output channels at the same time. Its settings should be unchecked with **Latency** set to 1.
12. The **output gateway** blocks are used to output the resultant signals to the Scopes and for verification of the design.
13. The **Scope** block is used to verify the Simulink Model. Since there is no MATLAB testbench script for this project, this essentially becomes the method for testing the design. To configure it properly, add it to the design, and double click to open the scope. Click on the  button (parameters). Under the General tab, set the **Number of axes** to the number of input lines you want to monitor and for best results, set the **Time range** to 180.
14. Test your design! At the top of the Simulink model Screen, find the Toolbar shown below.



Fill in the box next to the record button with a value of 180. This is the number of time samples that will be used for your scope. With the scope open, click the play button at the top of the Simulink Model screen. This will compile and run your simulation. Your results should look similar to Figures 1-5 and 1-6. You may get a warning about using Simulink blocks to drive your input gateways, you can ignore it. Once the design runs, you may need to right click on the axes and select AutoScale to properly view the waveform.

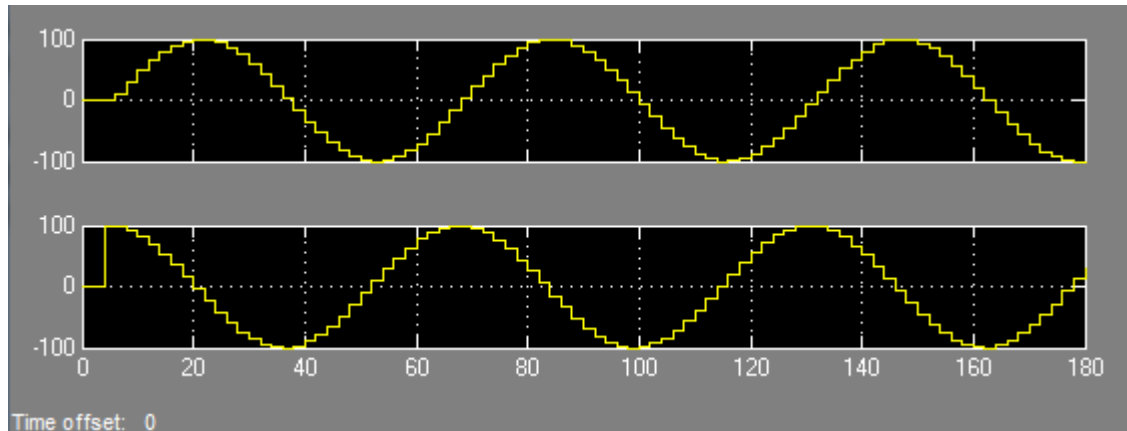


1-5: Scope view of Input Signals for ADC Driver Simulink Model



## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

As you can see from the figure plots, the input is a multiplexed sinusoid, and the `iq_select` toggles from high to low with every clock cycle. The toggling should be exactly in sync with the switching of the input signal.



1-6: Scope view of output Signals for ADC Driver Simulink Model

Figure 1-6 shows the output of the model. A correctly working ADC driver model should take in a multiplexed signal composing of the two generated sinusoids and output the original waveforms on the `rx_i` and `rx_q` lines.

15. The **ChipScope** block will be used to verify the received signal once the design is completed. It is not necessary to configure it with all 4 input signals as the `rx_i` and `rx_q` signals are what you will be verifying; however it can be helpful for troubleshooting to view both input and output signals. Configure the block as: **Number of trigger ports 4; Display settings for trigger port 0; Number of match units 1; Match type Basic; Use trigger port as data** Checked; **Depth of capture buffer 1024; Use SRL16s** Checked; **Add BUFG to JTAG clock** Checked.
16. Add the **System Generator** and **EDK Processor** blocks to your design. Configure the system generator the same as in the previous labs, and **save** the design. You are now 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.

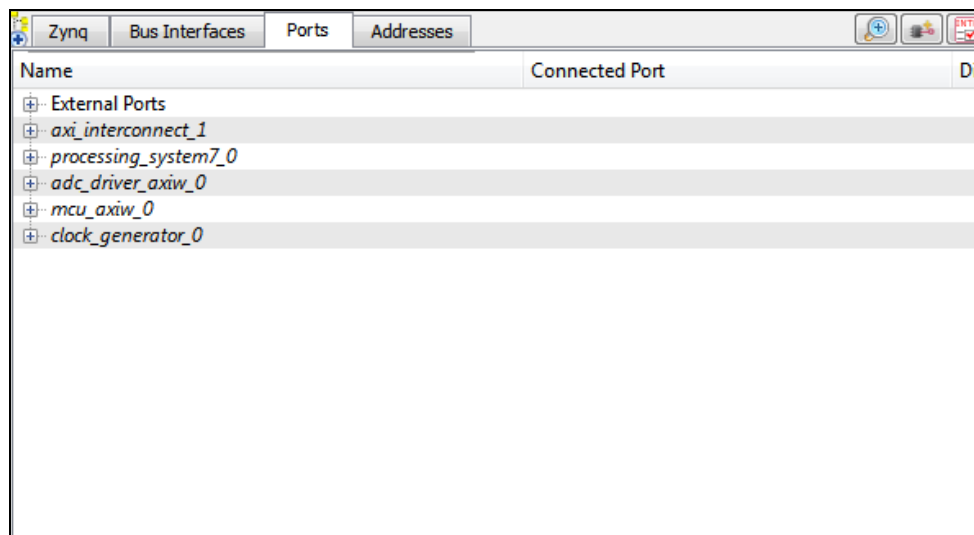


Figure 2-1: EDK project ports list

## 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 `rx_d` 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**. 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 **External 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 `p11_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**

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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

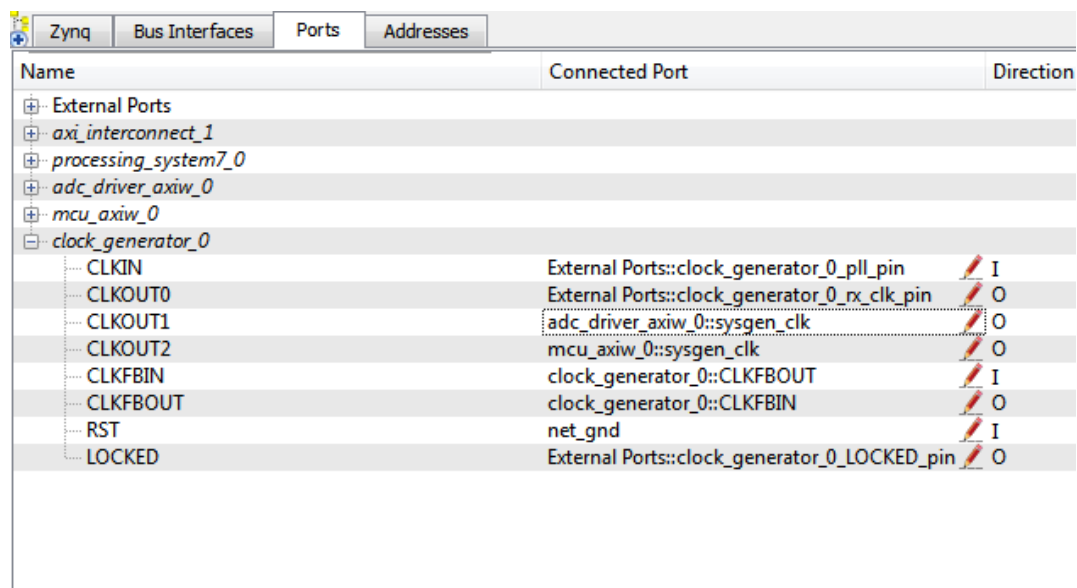
2. **Connect** the pins according to the following.

- CLKIN → External Ports
- CLKOUT0 → External Ports
- CLKOUT1 → adc\_driver::sysgen\_clk
- CLKOUT2 → mcu::sysgen
- CLKFBIN → CLKFBOUT
- RST → net\_gnd
- LOCKED → External Port

**Note**

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.

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



Name	Connected Port	Direction
External Ports		
axi_interconnect_1		
processing_system7_0		
adc_driver_axiw_0		
mcu_axiw_0		
clock_generator_0		
CLKIN	External Ports::clock_generator_0_pll_pin	I
CLKOUT0	External Ports::clock_generator_0_rx_clk_pin	O
CLKOUT1	adc_driver_axiw_0::sysgen_clk	O
CLKOUT2	mcu_axiw_0::sysgen_clk	O
CLKFBIN	clock_generator_0::CLKFBOUT	I
CLKFBOUT	clock_generator_0::CLKFBIN	O
RST	net_gnd	I
LOCKED	External Ports::clock_generator_0_LOCKED_pin	O

Figure 2-2: Clock Generator port configuration

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

## 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.

```
##### 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;
##### Rx – FMC interface at 2.5V #####
NET clock_generator_0_rx_clk_pin        LOC = J18 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET adc_driver_axiw_0_rx_iq_sel_pin     LOC = N19 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[0]        LOC = M21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[1]        LOC = J21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[2]        LOC = M22 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[3]        LOC = J22 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[4]        LOC = T16 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[5]        LOC = P20 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[6]        LOC = T17 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[7]        LOC = N17 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[8]        LOC = J20 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[9]        LOC = P21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[10]       LOC = N18 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[11]       LOC = J16 | IOSTANDARD = LVCMOS25;
##### MCU Interface #####
NET mcu_axiw_0_mcu_reset_pin            LOC = K20 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_axiw_0_tx_en_pin                LOC = D22 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_axiw_0_tr_sw_pin                LOC = D20 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_axiw_0_rx_en_pin                LOC = C22 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_axiw_0_pa_en_pin                LOC = E21 | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_axiw_0_init_done_pin            LOC = K19 | IOSTANDARD = LVCMOS25;
##### LEDs #####
NET clock_generator_0_LOCKED_pin        LOC = T22 | IOSTANDARD=LVCMOS33; # "LD0"
NET mcu_axiw_0_blinky_mcu_pin           LOC = T21 | IOSTANDARD=LVCMOS33; # "LD1"
NET adc_driver_axiw_0_blinky_adc_driver LOC = V22 | IOSTANDARD=LVCMOS33; # "LD4"
```

Figure 2-3: EDK project pin assignments

**Note**

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](#)<sup>3</sup> 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.

---

<sup>3</sup> Issue can be found at <http://www.xilinx.com/support/answers/51739.htm>

## 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.

---

**Note**

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.

---

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```
#include <stdio.h>

int main()
{
    // mcu registers
    int *chili_init_done, *chili_pa_en, *chili_tr_sw;
    int *chili_mcu_reset, *chili_rx_en, *chili_tx_en;

    // MCU core
    chili_init_done = (int *) (0x61600000 + 0x800);
    chili_pa_en = (int *) (0x61600000 + 0x800);
    chili_tr_sw = (int *) (0x61600000 + 0x804);
    chili_mcu_reset = (int *) (0x61600000 + 0x808);
    chili_rx_en = (int *) (0x61600000 + 0x80C);
    chili_tx_en = (int *) (0x61600000 + 0x810);

    // enable Tr/Rx cores in the RFIC
    *chili_rx_en = 1;
    *chili_tx_en = 1;

    // toggle reset on MCU - active low
    *chili_mcu_reset = 1;
    *chili_mcu_reset = 0;
    *chili_mcu_reset = 1;

    // wait for the MCU to finish calibration
    while(*chili_init_done == 0)
        ;

    // put RF front-end in state for receive
    *chili_pa_en = 0; // active high
    *chili_tr_sw = 1; // 0- transmit, 1-receive

    while (1);

    return 0;
}
```

Figure 3-1: Code outline for SDK project

### 3.2 Debugging with SDK, IMPACT and ChipScope Pro

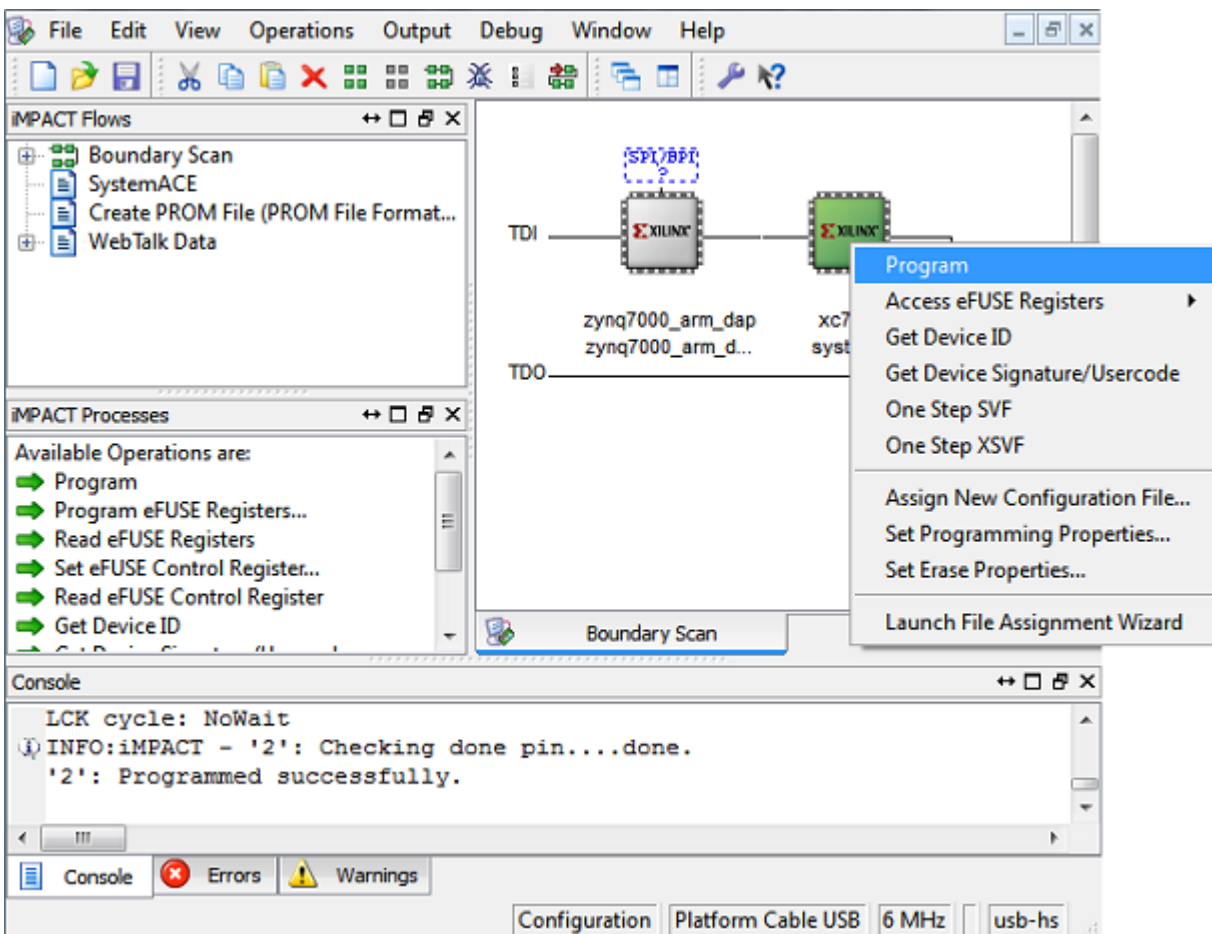
Once your program is written and compiled you are ready to test the design! This is done by programming the FPGA with your hardware descriptions defined in the bit file generated in EDK, and running your software on top of this design.

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.



## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper


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.


## Testing and Design Verification

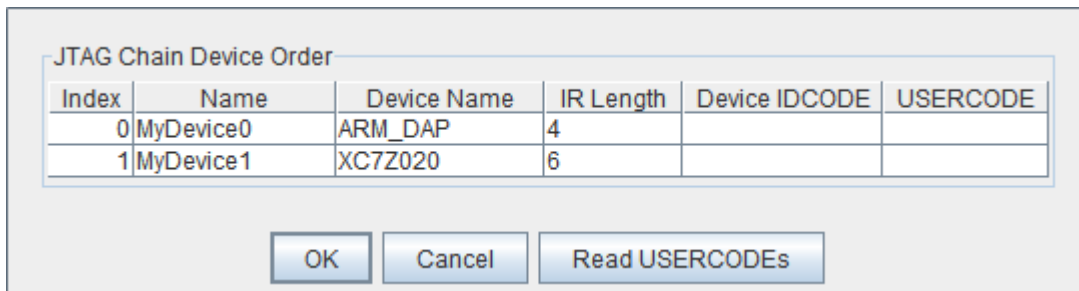
## Step 4

---

### 4.1 Verification with ChipScope Pro

There are several methods available for verifying the received tone. This lab focuses on verification using ChipScope Pro, as well as exporting to MATLAB for further analysis. In addition the single frequency tone received was generated using a signal generator; However if you have multiple boards available, you can also create the tone using lab 1, and receive on your second board using lab 2.

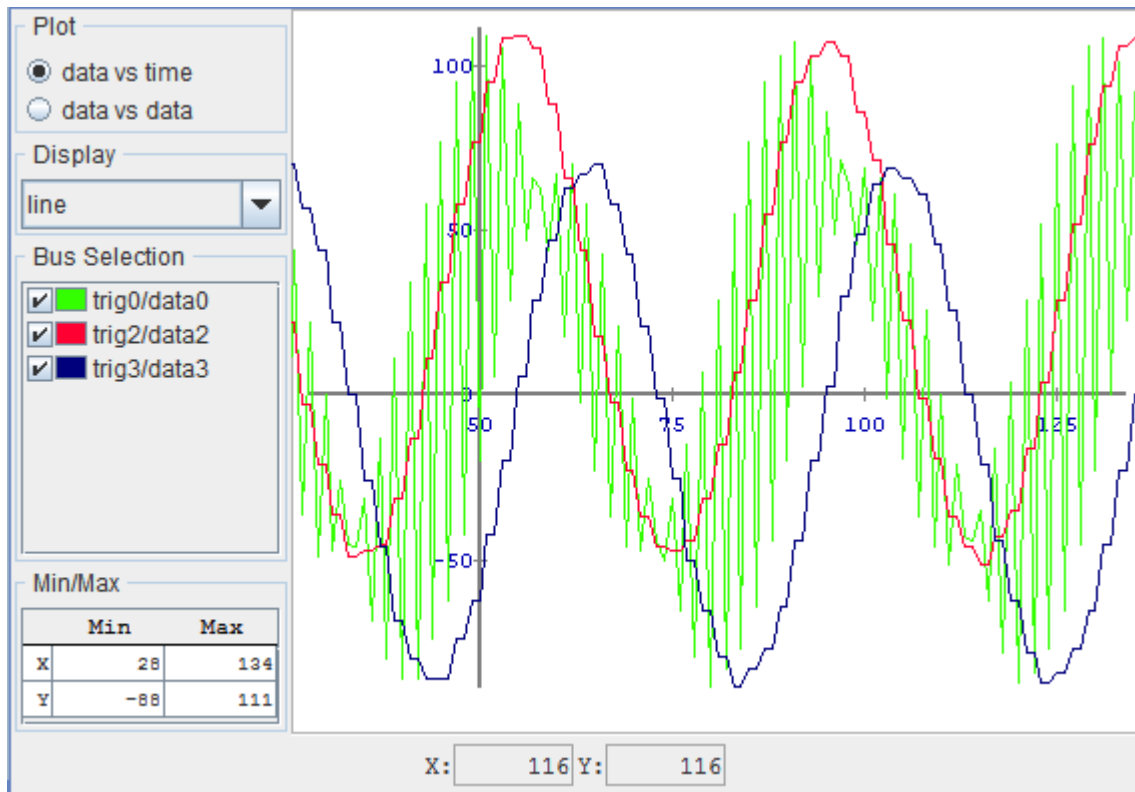
1. To verify the received signal, you will need to open **ChipScope Pro Analyzer**. Be sure that the JTAG cable is connected to the FPGA board properly (or the 6 pin output of the Chilipepper).
2. Once the program opens, click the  (open cable) button to open your JTAG connection to the board. If your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



3. Select ok to get to the Analyzer main screen. Open the file menu and select **Import**.
4. 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.
5. 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.
6. Under Data Port in the Signals Dev menu on the left side of the screen, right click on each of the trig/data ports, and change their **bus radix** to **signed decimal**. Click OK to accept the default decimal values.

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

- 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.
- 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 3-3 below.



4-1: Plot of 1MHz received signal from ChipScope Pro

## 4.2 MATLAB Analysis

Now that you have verified the received signal, you can get a pretty good idea of what frequency your radio board is receiving by using the sampling frequency and dividing it by an estimate of the period shown in ChipScope. However, ChipScope allows you to export the data received directly into MATLAB for further analysis.

- It may be helpful later in your MATLAB code if you rename your **Data Port variables**. Double click on each of the trig/data Ports, and **change the names** to something more descriptive, such as `rx_d`, `rx_i`, and `rx_q` respectively. If needed, you can use the Simulink model to find which signal each port has.

## Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

2. Open the file menu and select **Export**.
3. Click the Ascii radio box, then click export.
4. Save the file in a convenient location. It is recommended you save this file into the Sysgen folder with your MATLAB files. Call it something descriptive such as **ADC.prn**.

**Note**

This file contains all the information used to display the Bus Plots in ChipScope, and can be opened directly in a MATLAB script to display, as well as analyze using built in MATLAB spectral analysis functions.

5. To display the data in MATLAB, create a script with the code shown below.

```
xlLoadChipScopeData('ADC.prn');
plot(1:1024, rxd, 1:1024, rx_i, 1:1024, rx_q)
```

6. This plots the waveforms in MATLAB. To view the FFT plot of the wave forms, write a MATLAB script similar to the one in Figure 4-2 below.

```
%Load scope data
xlLoadChipScopeData('ADC.prn');

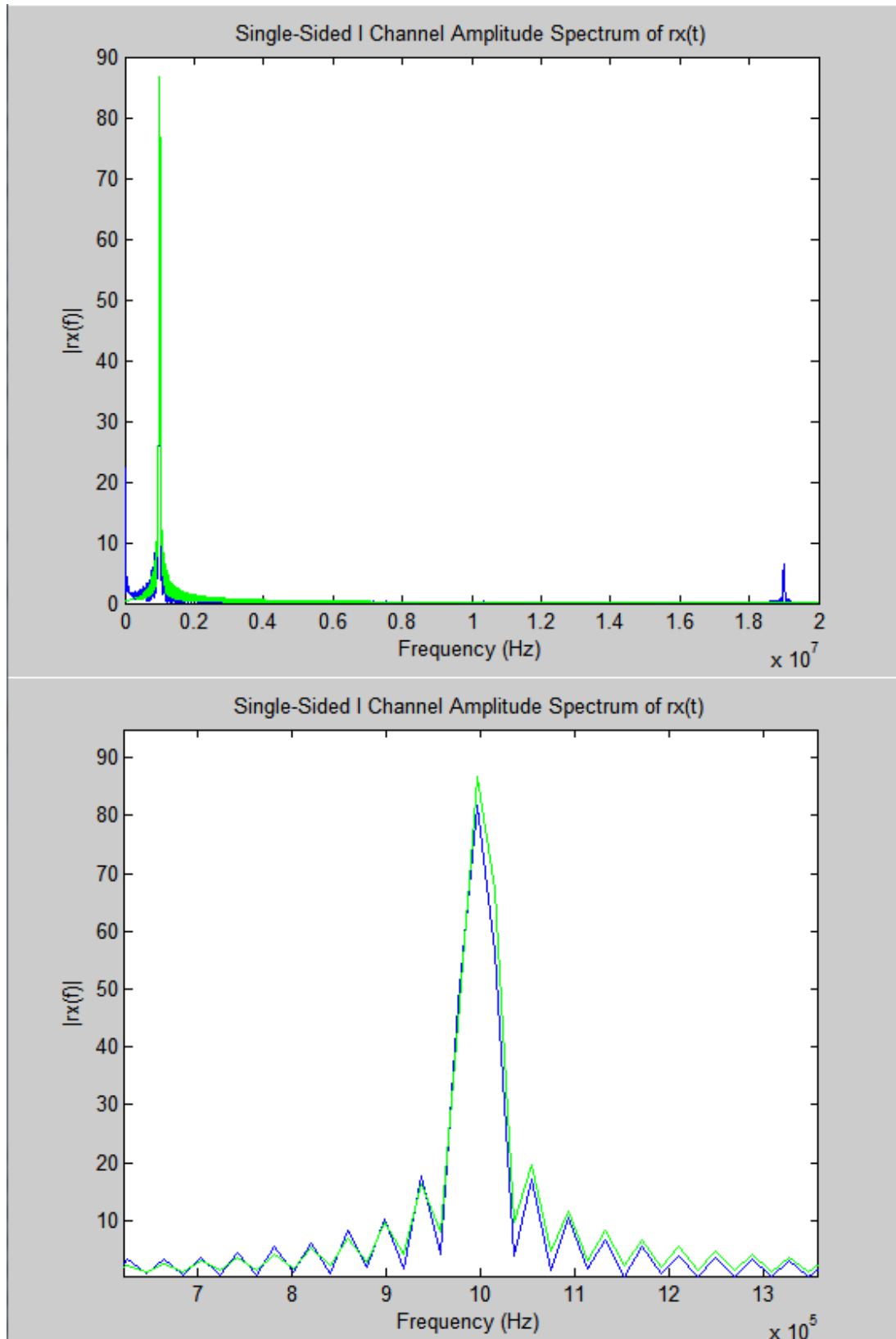
%Process signal data
Fs=40E6;
L=length(rx_i)-1;
NFFT=2*(2^nextpow2(L));
Y_i=fft(rx_i,NFFT)/L;
f = Fs/2* linspace(0,1,NFFT/2+1);

%Create theoretical signal
t=(0:L-1)/Fs;
X=88*sin(2*pi*1e6*t);
X_i=fft(X,NFFT)/L;

% Plot single-sided amplitude spectrum.
plot(f,2*abs(Y_i(1:NFFT/2+1)),f,2*abs(X_i(1:NFFT/2+1)),'green');
title('Single-Sided I Channel Amplitude Spectrum of rx(t)')
xlabel('Frequency (Hz)')
ylabel('|rx(f)|')
```

4-2: Matlab code to display FFT plot of the waveforms

Looking at the `rx_i` and `rx_q` frequency plots, you should be able to see a spike at the frequency you sent to the radio board. The results of the code used above are shown below in Figure 4-3. The signal in blue is the I channel of the received waveform, while the signal in green is a 1 MHz waveform created in MATLAB and used as a reference.



4-3: FFT Plot of the I channel

## Conclusion

---

Now that you were able to send and receive signals using Chilipepper, see if you can send other types of signals from one board to another, and verify the results using MATLAB spectral plots.