Toyon Research Corporation

Lab 6: Timing Offset Estimation

Chilipepper Tutorial Projects

Version 0.4
5/24/2013

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Table of Contents

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Lab 6: Timing Offset Estimation

Introduction

This lab will extend the previous lab and allow you to properly recover the timing of the transmitted signal. 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). The frequency estimation and band-pass filtering of the received signal from the previous lab will take place on the FPGA via an exported Simulink Pcore. 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 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 implement a basic signal receiver
- 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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 builds on the MATLAB algorithm used in Lab 5 and adds the timing offset estimation to produce a better estimate the received waveform. The first function used is shown in Figure 1-1.

```matlab
function [r_out, s_f_out, s_c_out, s_t_out, t_est_out, f_est_out] = ...
    qpsk_rx(i_in, q_in, mu_foc_in, mu_toc_in)

    % scale input data coming from the Chilipepper ADC to be purely
    % fractional to avoid scaling issues
    r_in = complex(i_in, q_in);

    % frequency offset estimation. Note that time constant is input as
    % integer
    [s_f_i, s_f_q, fe] = qpsk_rx_foc(i_in, q_in, mu_foc_in);

    % Square-root raised-cosine band-limited filtering
    [s_c_i, s_c_q] = qpsk_rx_srrc(s_f_i, s_f_q);

    % Time offset estimation. Output data changes at the symbol rate.
    [s_t_i, s_t_q, tauh] =  qpsk_rx_toc(s_c_i, s_c_q, mu_toc_in);

    % estimation and correlation values
    t_est_out = tauh;
    f_est_out = fe;

    % original signal out (real version)
    r_out = real(r_in);

    % incremental signal outputs after frequency estimation, filtering, and
    % timing estimation
    s_f_out = complex(s_f_i,s_f_q);
    s_c_out = complex(s_c_i,s_c_q);
    s_t_out = complex(s_t_i,s_t_q);
end
```

Figure 1-1: MATLAB function to analyze received signal.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

This function is almost identical to the one used in the previous lab, however it adds the Timing offset estimation function call.

1. Create a directory for the project under C:\QPSK_Projects\Project_6.

2. Create a MATLAB directory within the main project directory.

3. Create a new **MATLAB function** with the contents of Figure 1-1.

4. **Save** this function as `qpsk_rx.m` inside the project directory.

The first task of this function is implementing the frequency offset correction covered in the previous lab. The code for this function is assumed to have been created previously and is not shown in this lab. The same is true for the raised-cosine band-limited filter function. Please refer to lab 5 for the code used within these functions. The timing estimation is handled by the function `qpsk_rx_toc.m` which can be seen in **Appendix A**. This function implements an algorithm which attempts to maximize the output power of the received waveform, and will adjust the timing offset estimate accordingly until the maximum value is found.

5. Create a new **MATLAB function** with the contents of Appendix A.

6. Save this function as `qpsk_rx_toc.m` inside the MATLAB project directory

1.2　MATLAB Test Bench

Now that you have added the code needed to further analyze the signal, we also need to slightly modify the previously created test bench script for the functions. The objective of the test bench script is still to observe the output graph of the result, we just need to account for more output data. Just as before, this script will need to simulate a transmitted waveform for the analysis. After the algorithm has been verified using simulated data, and the FPGA design completed, you will again need to verify the algorithms with actual data from ChipScope. This will be done later in Section 6 of this lab. The code used for this script is shown in **Appendix B**.

Note	There is a variable called `sim` in the script which allows you to either load your received waveform data from ChipScope or simulate a received QPSK signal in MATLAB and analyze the results. Setting it to 1 simulates the waveform, 0 loads it from a ChipScope prn file.

1. Create a new **MATLAB script** with the contents of Appendix B.

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

These are the only files required for the analysis of a received waveform when using exported ChipScope data. However to use simulated data for initial verification of the algorithm, you must also add the files used in the QPSK lab to your project directory. For your reference a list of the required files is given below. Refer to Lab 3 Output QPSK to recreate these files, or download them from the Chilipepper Github Repo[1].

Required files for creating Simulated QPSK waveform
- `make_train_lut.m`
- `CreateAppend16BitCRC.m`
- `qpsk_tx.m`
- `qpsk_tx_byte2sym.m`
- `qpsk_srrc.m`
- `mybitget.m`

Also be sure you have the required files used in the previous lab for both DC_Offset correction and frequency estimation. Once you have all the required MATLAB files in your project directory, you can run the test bench script to view the waveform analysis. You may have to run the script twice to create the needed LUT files. Your data should look similar to the results shown in Figures 1-2 and 1-3. Be sure you have the `sim` variable in the test bench script set to 1.
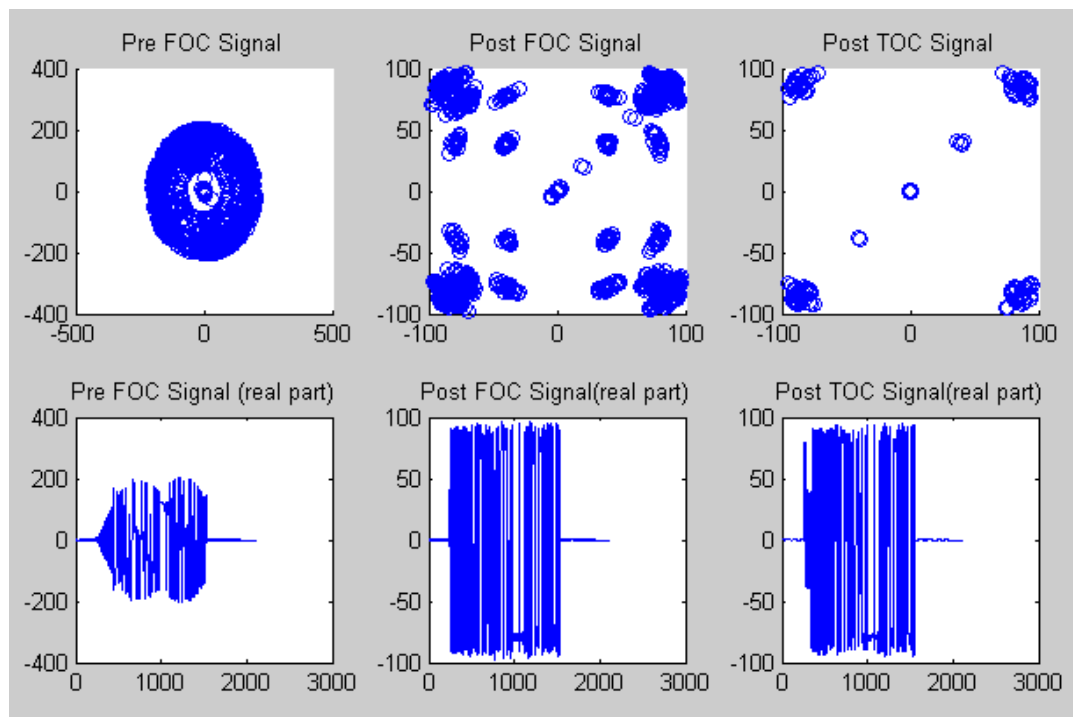

Figure 1-2: Analysis results for simulated QPSK signal

[1] https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_6/Matlab

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper
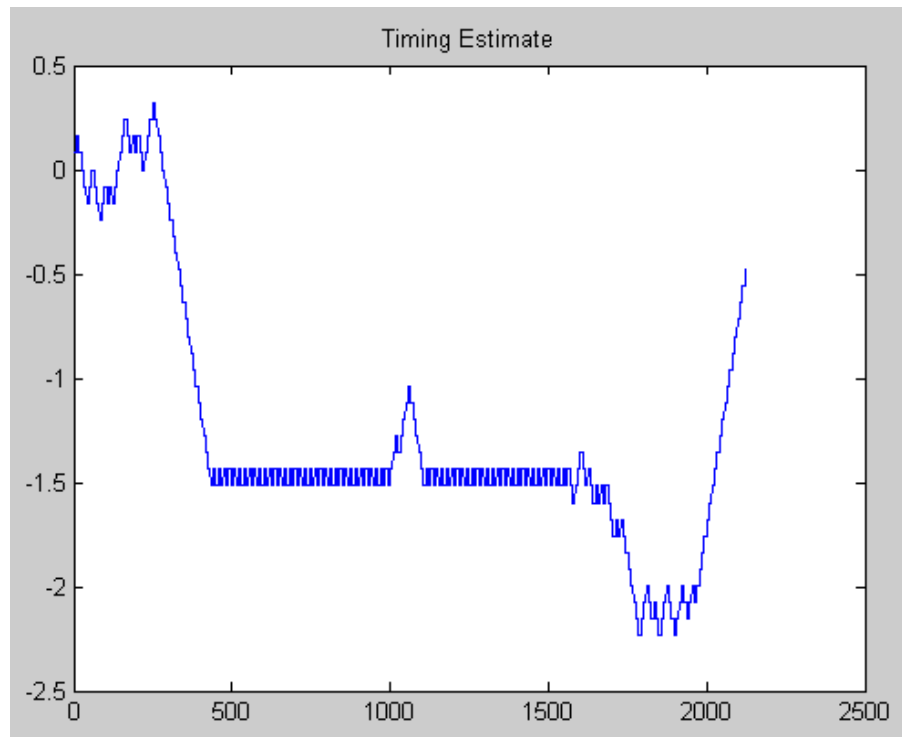

Figure 1-3: Phase estimate for simulated QPSK signal

As you can see from the results of the figures, the new MATLAB function allows for capturing the timing offset, which is used to obtain the correct timing for the QPSK waveform, giving the new signal (post TOC) signal shown in Figure 1-2. This signal is a still closer estimate of the originally transmitted waveform than the ones shown before and after applying the Frequency offset correction (Pre FOC / Post FOC) and filtering.

1.3 RX HDL Coder Project

Using the same steps outlined in the previous labs, create a new HDL coder project called rx_qpsk. Add your MATLAB function `qpsk_rx.m` and your 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 1-4 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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper
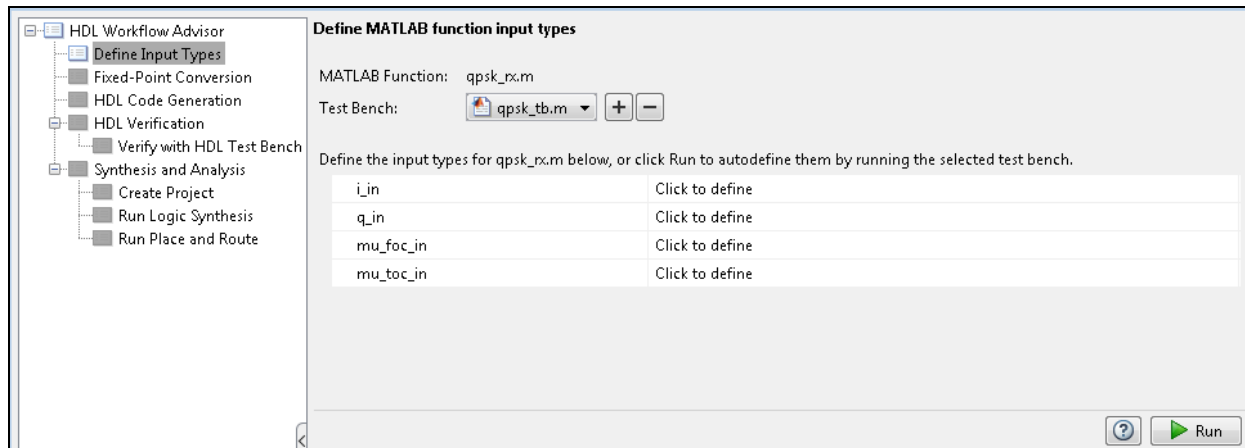

Figure 1-4: HDL Code Generation Workflow Advisor

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 a popup that says "Validation succeeded". This means that MATLAB has successfully analyzed your design and selected fixed point types to replace the floating point arithmetic required in your algorithm. However, not all of the automatic selections are sufficient for our FPGA design; therefore several of the conversions will need to be modified.

3. Using the function dropdown menu at the top of the HDL Code Generation screen, select each of the functions in the design and make the following modifications.

Note	Many of the Fixed-Point conversions are the same as those used in the previous labs. For your convenience, this lab only shows the Functions which have more input/output variables than the previous labs, and whose proposed types need to be modified. Refer to the previous lab for information on all other proposed types.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

qpsk_rx_toc

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
▲ Input							
mu_in	double	327	327			Yes	**numerictype(0, 12, 0)**
r_i	double	-820.43	852.68			No	**numerictype(1, 26, 12)**
r_q	double	-806.16	885.61			No	**numerictype(1, 26, 12)**
▲ Output							
s_i	double	-820.43	852.68			No	**numerictype(1, 26, 12)**
s_q	double	-774.88	856.18			No	**numerictype(1, 26, 12)**
tauh	double	-0.08	5.03			No	**numerictype(1, 20, 12)**
▲ Persistent							
counter	double	0	8			Yes	numerictype(0, 4, 0)
rBuf_i	1 x 32 double	-820.43	852.68			No	**numerictype(1, 26, 12)**
rBuf_q	1 x 32 double	-806.16	885.61			No	**numerictype(1, 26, 12)**
symLatch_i	double	-820.43	852.68			No	**numerictype(1, 26, 12)**
symLatch_q	double	-774.88	856.18			No	**numerictype(1, 26, 12)**
tEst	double	-0.08	5.03			No	**numerictype(1, 20, 12)**
tau	double	-0.08	5.03			No	**numerictype(1, 20, 12)**
▲ Local							
OS_RATE	double	8	8			Yes	numerictype(0, 4, 0)
mu	double	0.08	0.08			No	**numerictype(0, 12, 12)**
od_i	double	-556.8	527.22			No	**numerictype(1, 26, 12)**
od_r	double	-236.07	495.08			No	**numerictype(1, 26, 12)**
oe	double	-1	1			Yes	**numerictype(1, 2, 0)**
oe_i	double	-133666.29	412316.33			No	**numerictype(1, 38, 12)**
oe_r	double	-189117.41	30356.7			No	**numerictype(1, 38, 12)**
os	double	-195994.39	339140.82			No	**numerictype(1, 38, 12)**
taur	double	0	5			Yes	**numerictype(0, 8, 0)**
ze_i	double	-652.88	667.63			No	**numerictype(1, 26, 12)**
ze_q	double	-806.16	885.61			No	**numerictype(1, 26, 12)**
zl_i	double	-768.57	811.66			No	**numerictype(1, 26, 12)**
zl_q	double	-645.16	770.76			No	**numerictype(1, 26, 12)**
zo_i	double	-820.43	852.68			No	**numerictype(1, 26, 12)**
zo_q	double	-774.88	856.18			No	**numerictype(1, 26, 12)**

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

qpsk_rx

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
◢ Input							
i_in	double	-204	205			Yes	numerictype(1, 12, 0)
mu_foc_in	double	40	40			Yes	numerictype(0, 12, 0)
q_in	double	-205	205			Yes	numerictype(1, 12, 0)
◢ Output							
s_c_out	complex double	-97.73	97.11			No	numerictype(1, 26, 12)
s_f_out	complex double	-161.66	160.86			No	numerictype(1, 26, 12)
f_est_out	double	-0.01	0.99			No	numerictype(1, 26, 12)
r_out	double	-204	205			Yes	numerictype(1, 12, 0)
◢ Local							
r_in	complex double	-205	205			Yes	numerictype(1, 12, 0)
fe	double	-0.01	0.99			No	numerictype(1, 26, 12)
s_c_i	double	-97.37	96.43			No	numerictype(1, 26, 12)
s_c_q	double	-97.73	97.11			No	numerictype(1, 26, 12)
s_f_i	double	-161.66	160.86			No	numerictype(1, 26, 12)
s_f_q	double	-160.38	159.95			No	numerictype(1, 26, 12)

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.

4. Select Validate Types to verify the new design

5. 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.

6. Once created, copy the black box and System generator Blocks to a new Model just as in the previous labs.

7. Save the new model as rx.slx into the sysgen directory "Lab_6\sysgen".

8. Copy the `qpsk_rx_FixPt_xsgbbxcfg.m` file into your Sysgen folder just as in the previous labs.

9. 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.

Create and export Simulink models   Step 2

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.

2.1   Create MCU Simulink Design

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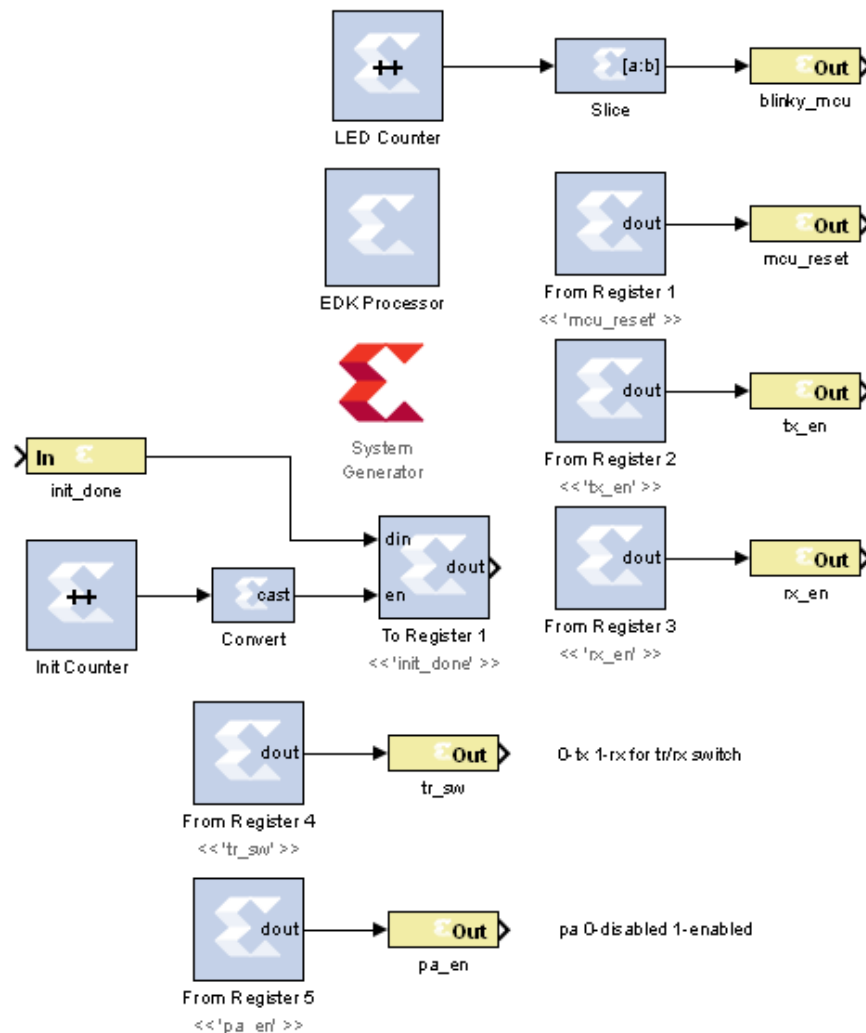

Figure 2-1: Simulink model for MCU control

---

[2] This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_6/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 into the Sysgen folder. Name the file **mcu.slx** or something similar.

2.2   Create Receiver Simulink Design

The **Simulink model**[3] In Figure 2-2 will be used for receiving the ADC output and sending the data to ChipScope for extraction.
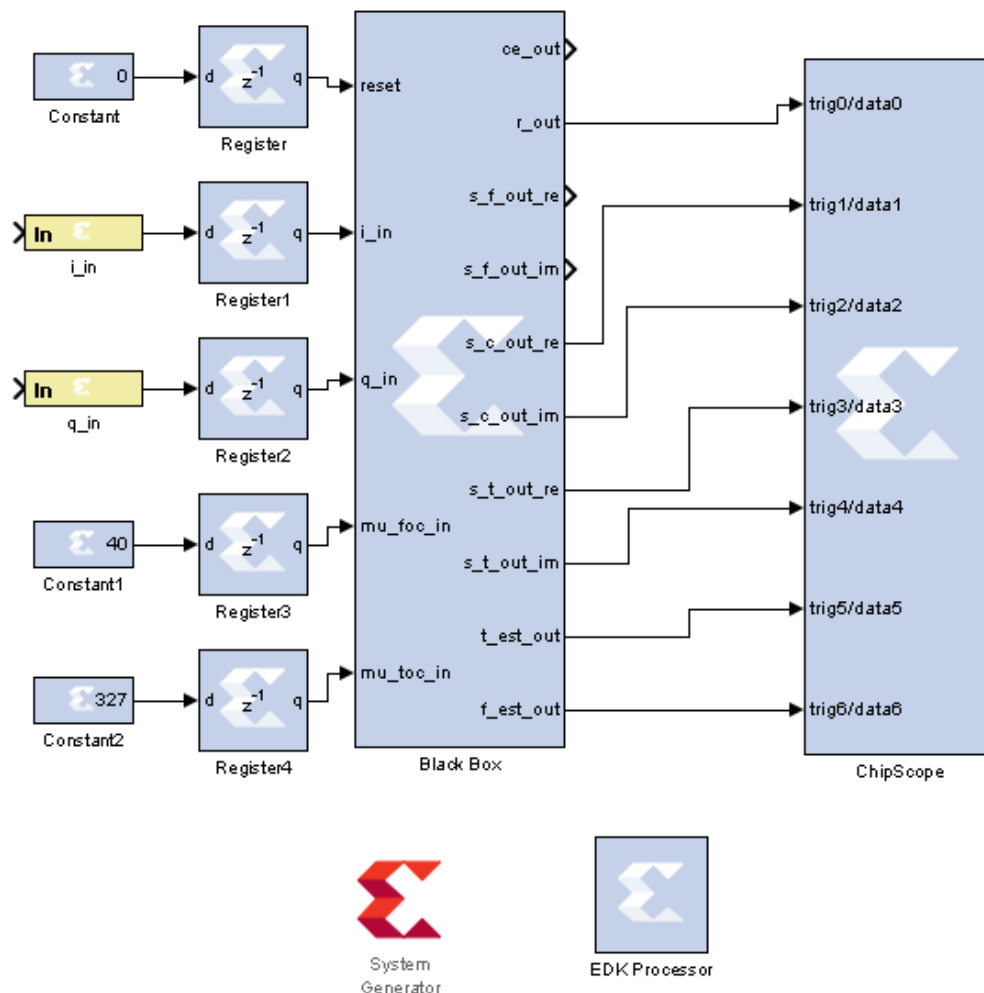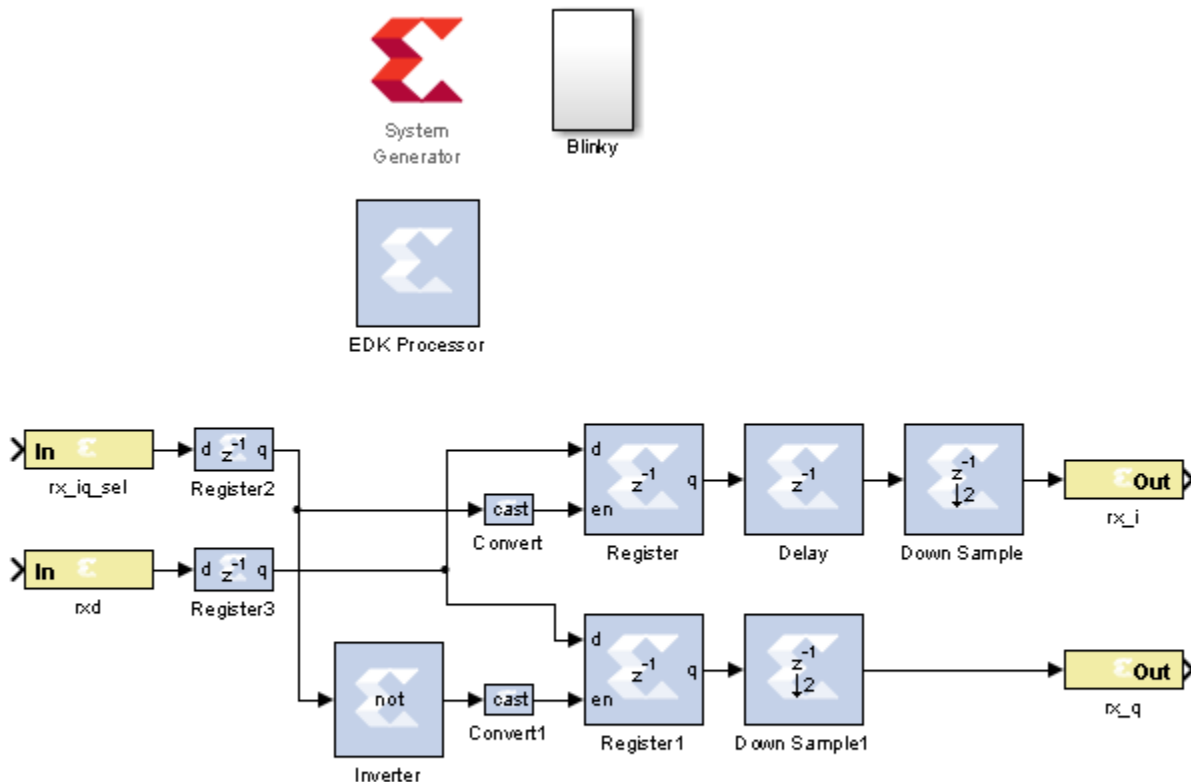

Figure 2-2: Simulink model for receiving DC Offset output

---

[3] This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_6/sysgen

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

1. **Modify** the Simulink model `rx.slx` created earlier to look similar to Figure 2.2.

2. Both `i_in` and `q_in` should be signed 12 bit (0 decimal bits) inputs. The constant for `mu_foc_in` should be set to `floor(.01*2^12)` and for `mu_toc_in` should be `floor(.01*8*2^12)`.

3. **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 **rx.slx** or something similar.

2.3   Create ADC driver Simulink Design

The **Simulink model**[4] In Figure 2-3 will be used for creating the signals which drive the **ADC** on Chilipepper.

By default inphase is IQ_sel high and quadrature is IQ_sel low

This core runs at 40 MHz and demultiplexes the rxd data into two 20 MHz streams for I and Q

Figure 2-3: Simulink model for ADC control

---

[4] This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_6/sysgen

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 2-4. Configure
    the Blinky subsystem identically to the other LED out systems in the previous labs.
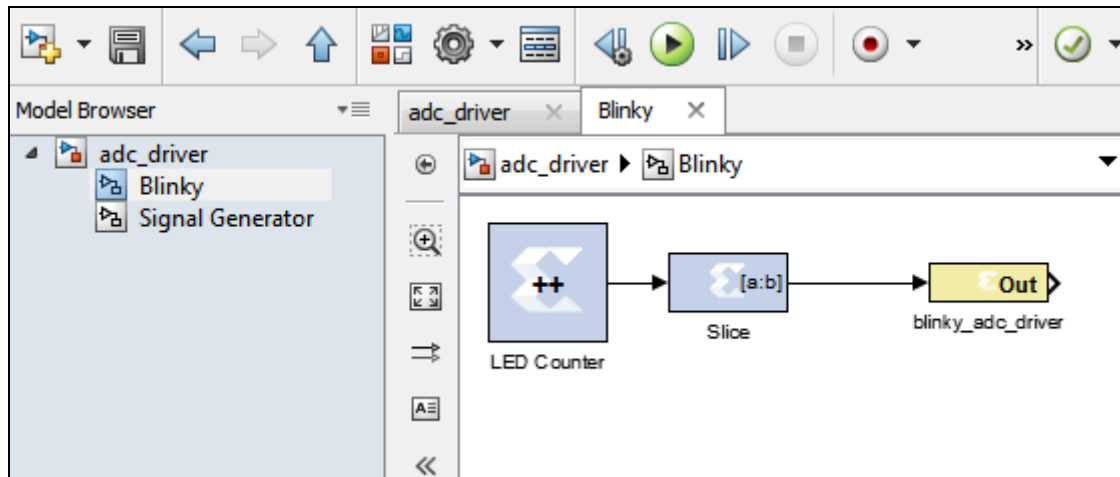

Figure 2-4: Blinky Subsystem

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

2.4   DC Offset Core

The last **Simulink model** [5]needed for this FPGA design is the DC Offset model created in a previous lab. For your reference this model is shown in Figure 2-5 below.
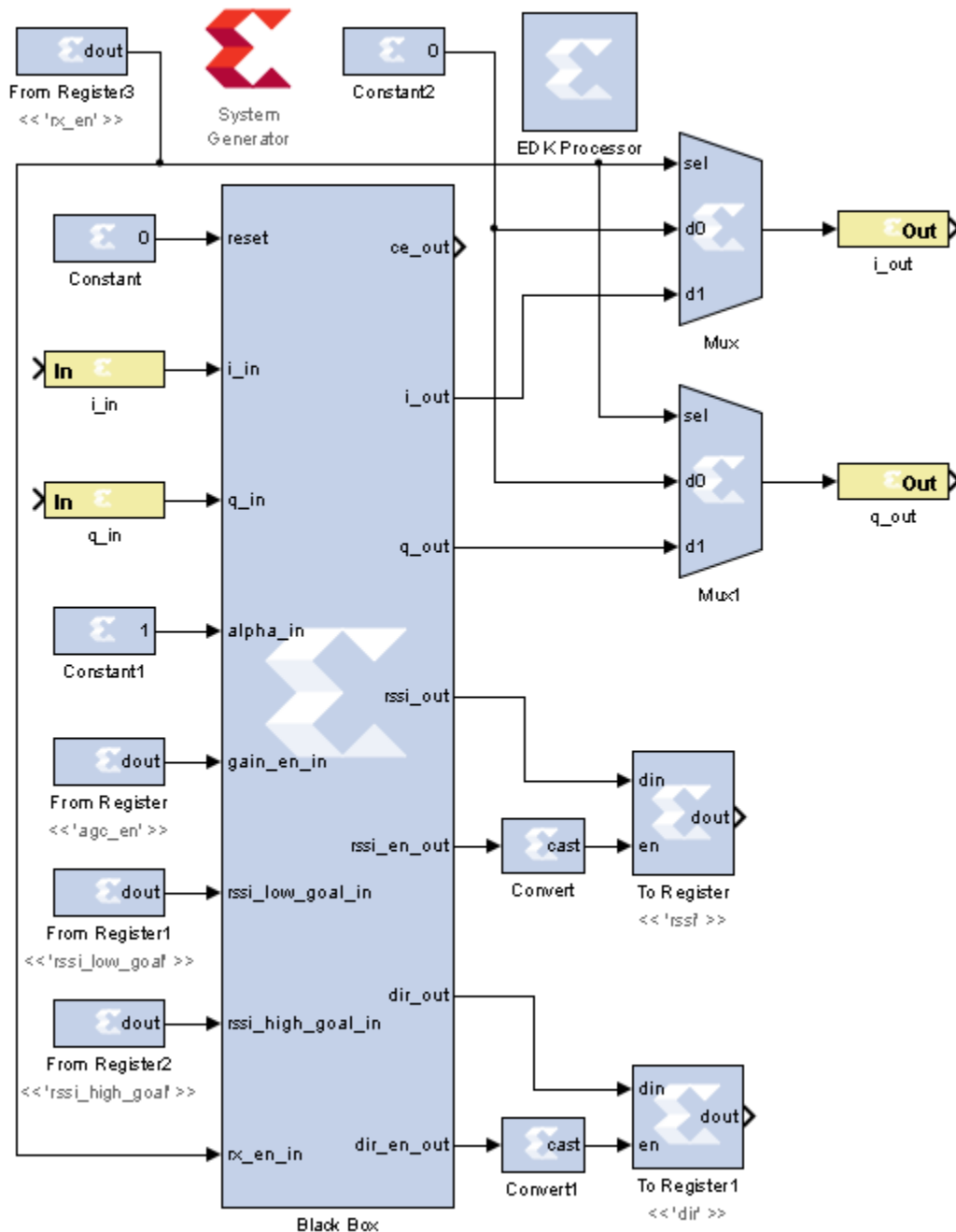

Figure 2-5: Simulink model for receiving ADC output and applying DC Offset Correction

---

[5] This model can be downloaded from https://github.com/Toyon/Chilipepper/tree/master/Labs/Lab_6/sysgen

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

1. Copy the `DC_Offset` Simulink files into your new project directory.

Note	Be sure to copy the vhd and config files to the new directory. You may use the Simulink files on the GitHub Repo as a reference. Additionally, you can open the old `DC_Offset` Simulink model and reconfigure its Pcore settings to point to the new EDK project directory, then recreate it.

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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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

- DC Offset 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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 for more information on individual port configuration.

3.3   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. Figure 3-2 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 3-2 below as a reference.

> 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.

5. Prior to EDK version 14.4, Xilinx had a <u>documented issue</u>[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.

---

[6] Issue can be found ay http://www.xilinx.com/support/answers/51739.htm

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```
############################################### 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 clock_generator_0_tx_clk_pin                 LOC = C17        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
############################################## MCU Interface ###############################################
NET axi_uartlite_0_RX_pin                        LOC = R19        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET axi_uartlite_0_TX_pin                        LOC = L21        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
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 axi_gpio_led_GPIO_IO_pin                             LOC = T22  | IOSTANDARD=LVCMOS33;  # "LD0"
NET axi_gpio_led_GPIO2_IO_pin                           LOC = T21  | IOSTANDARD=LVCMOS33;  # "LD1"
NET mcu_axiw_0_blinky_mcu_pin                           LOC = U22  | IOSTANDARD=LVCMOS33;  # "LD2"
NET adc_driver_axiw_0_blinky_adc_driver_pin             LOC = U21  | IOSTANDARD=LVCMOS33;  # "LD3"
NET clock_generator_0_LOCKED_pin                        LOC = V22  | IOSTANDARD=LVCMOS33;  # "LD4"
```

Figure 3-2: EDK project pin assignments

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 QPSK receiver project.

1. Select **File → New → Application Project**.

2. Name the project `qpsk_rx` or something similar and leave the other settings at their defaults. Be sure to select **Hello World** from the **Select Project Template** section.

3. Click **Finish**. You should now see your project folder, as well as a **board support package** (bsp) folder.

4. If you navigate into the 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. Feel free to give the file a more descriptive name such as `main.c` or something similar.

5. **Double click** the file to open it and **replace** all of its contents with the code in Figure 4-1.

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

```c
#include <stdio.h>
#include "platform.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xparameters.h"
#include "xstatus.h"
#include "chilipepper.h"
#include "xuartps.h"
#include "xil_printf.h"
#include "xscugic.h"
#include "xil_exception.h"

XGpio gpio_blinky;
int SetupPeripherals( void );

int main()
{
    int gain;
    int aliveLed = 0;
    static int BlinkCount = 0;
    init_platform();
    if(SetupPeripherals() != XST_SUCCESS)
      return -1;

    if ( Chilipepper_Initialize() != 0 )
      return -1;

    Chilipepper_SetPA( 1 );
    Chilipepper_SetTxRxSw( 1 ); // 0- transmit, 1-receive
    while (1)
    {
        gain = Chilipepper_ControlAgc(); //update the Chilipepper AGC
        BlinkCount += 1;
        if (BlinkCount > 500000)
        {
            if (aliveLed == 0)
                aliveLed = 1;
            else
                aliveLed = 0;
            BlinkCount = 1;
            XGpio_DiscreteWrite(&gpio_blinky, 2, aliveLed);  //blink LEDs
            XGpio_DiscreteWrite(&gpio_blinky, 1, ~aliveLed);
        }
    }
    cleanup_platform();
    return 0;
}

int SetupPeripherals( void )
{
    XGpio_Initialize(&gpio_blinky, XPAR_AXI_GPIO_LED_DEVICE_ID);
    XGpio_SetDataDirection(&gpio_blinky, 2, 0);
    XGpio_SetDataDirection(&gpio_blinky, 1, 0);
    XGpio_DiscreteWrite(&gpio_blinky, 1, 0);
    XGpio_DiscreteWrite(&gpio_blinky, 2, 0);
    return XST_SUCCESS;
}
```

Figure 4-1: Code outline for SDK project

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 githib repo. Place these files in the src directory of your project workspace.

1. Chilipepper.c – 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 – This file holds the function prototypes for the Chilipepper.c functions.

Note	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.

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 8-12 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 except the TX_DRIVER for this Lab.

```
#define MCU_UART
#define MCU_DRIVER
#define DC_OFFSET
//#define TX_DRIVER
#define RX_DRIVER
```

⚠	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)

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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. On one board, you should run the latest version of Lab 3 which allows you to either send a single packet via button press, or multiple packets using the switch on the FPGA.

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.

Note	If you are running lab 3 from a second PC, you will need to repeat this process for the second board using the Lab 3 system.bit file. Alternatively, you can run Lab 3 directly from the SD card by loading a standard SD card with the Boot.bin file for lab 3, which can be found on the github repo.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper
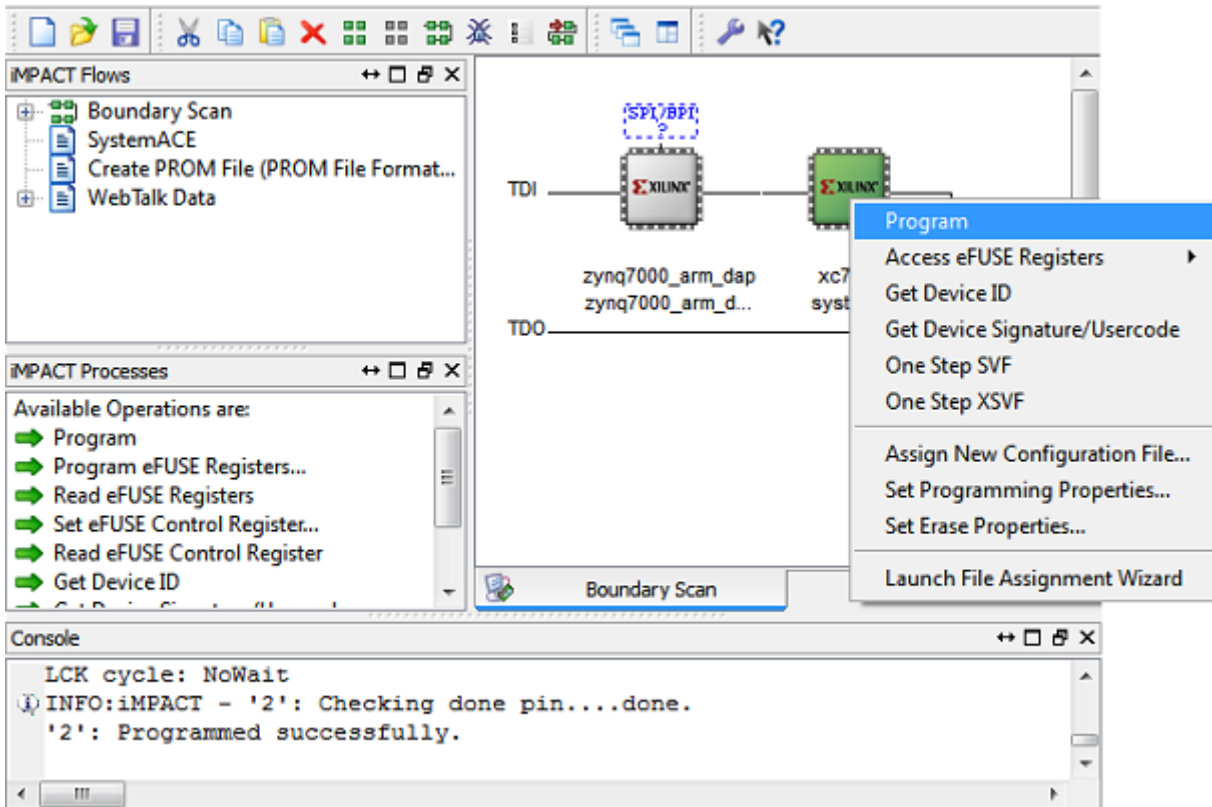
4-2: iMPACT configuration screen

To load Lab 3 via SD card:

1. Place the file on the SD card, and place the card inside the SD slot of the FPGA.

2. Configure the jumpers on the FPGA as shown in Figure 4-3.

3. Turn on the board, and the program should load after about 30 seconds. Check for the blue light, indicated the load was successful.
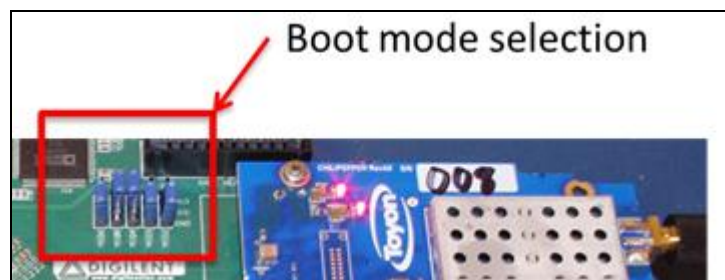

Figure 4-3: Jumper configuration needed to load a project via SD card

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

4.4   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 should also see the LED blinking on the second FPGA indicating the Lab 3 project is working properly. 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.

If the software initialization worked, you should see a green light on the Chilipepper as well as LED0 and LED1 blinking alternatively.
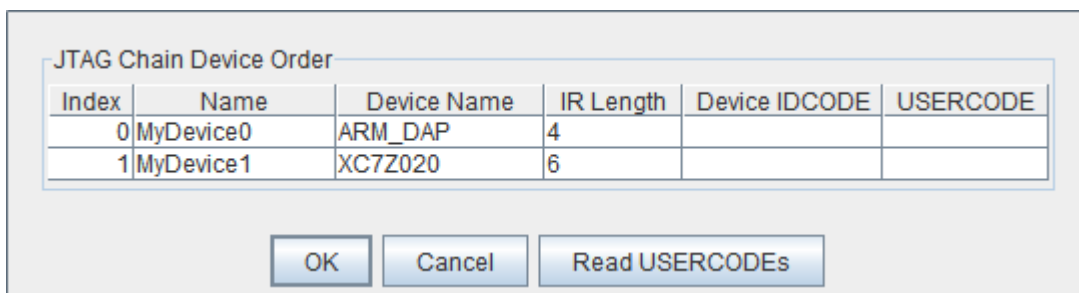
Testing and Design Verification   Step 5

5.1   Verification with ChipScope Pro

There are several methods available for verifying the received QPSK transmission. This lab focuses on verification using ChipScope Pro, as well as exporting to MATLAB for further analysis.

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.

JTAG Chain Device Order					
Index	Name	Device Name	IR Length	Device IDCODE	USERCODE
0	MyDevice0	ARM_DAP	4		
1	MyDevice1	XC7Z020	6		

OK   Cancel   Read USERCODEs

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 ports, and change their **bus radix** to **signed decimal**. Click OK to accept the default decimal values.

7. 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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

8. To correctly view the received the signal in ChipScope, you must catch the signal in the window which is currently being viewed in ChipScope. This is much easier to do if you flip the switch on the Lab 3 demo to allow for continuous packet transmission.

9. Click the **play button** in the top menu bar until you get a full display of 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.
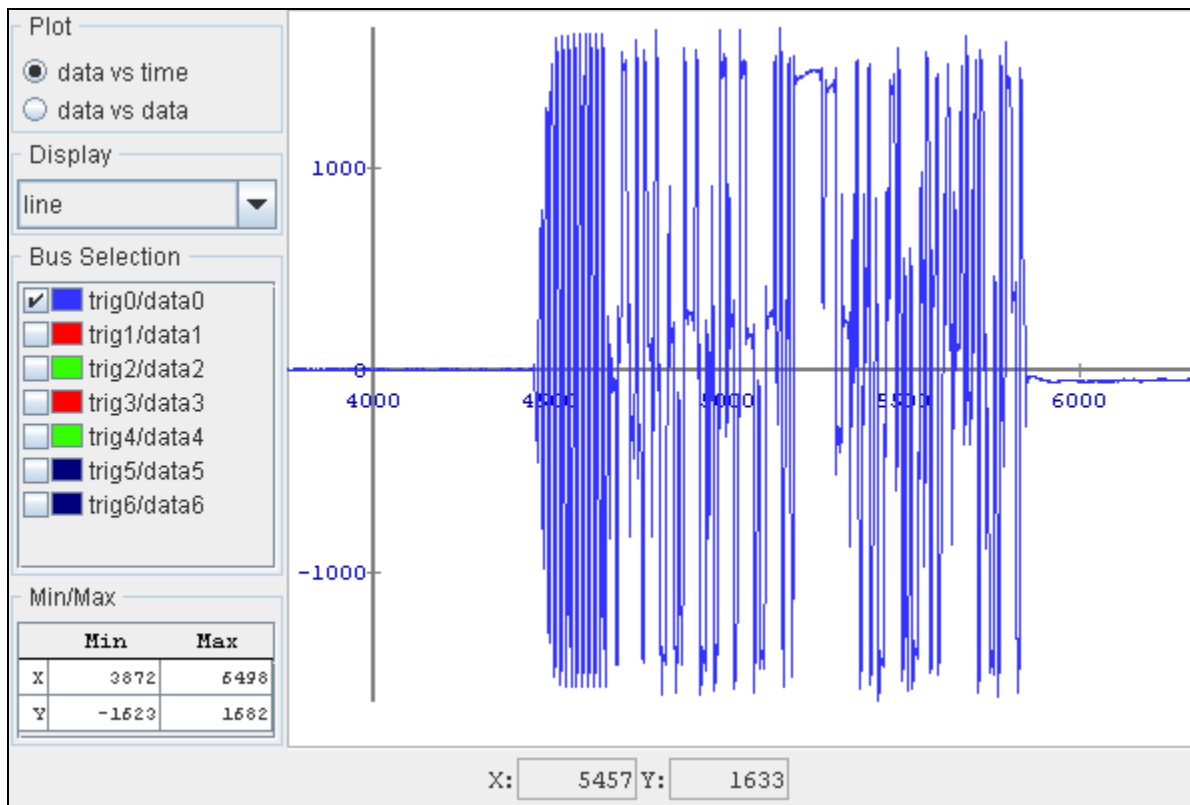
5-1: real part of the received QPSK waveform in ChipScope Pro

This is only the real part of the received signal, and is before any of the signal processing. To see the signal post processing, change the bus plot view to see the data1 and data2 values or the data 2 and data 3 values. Be sure to set the correct bus radix for your signal. Your i and q signals post processing should resemble the Figures below.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper
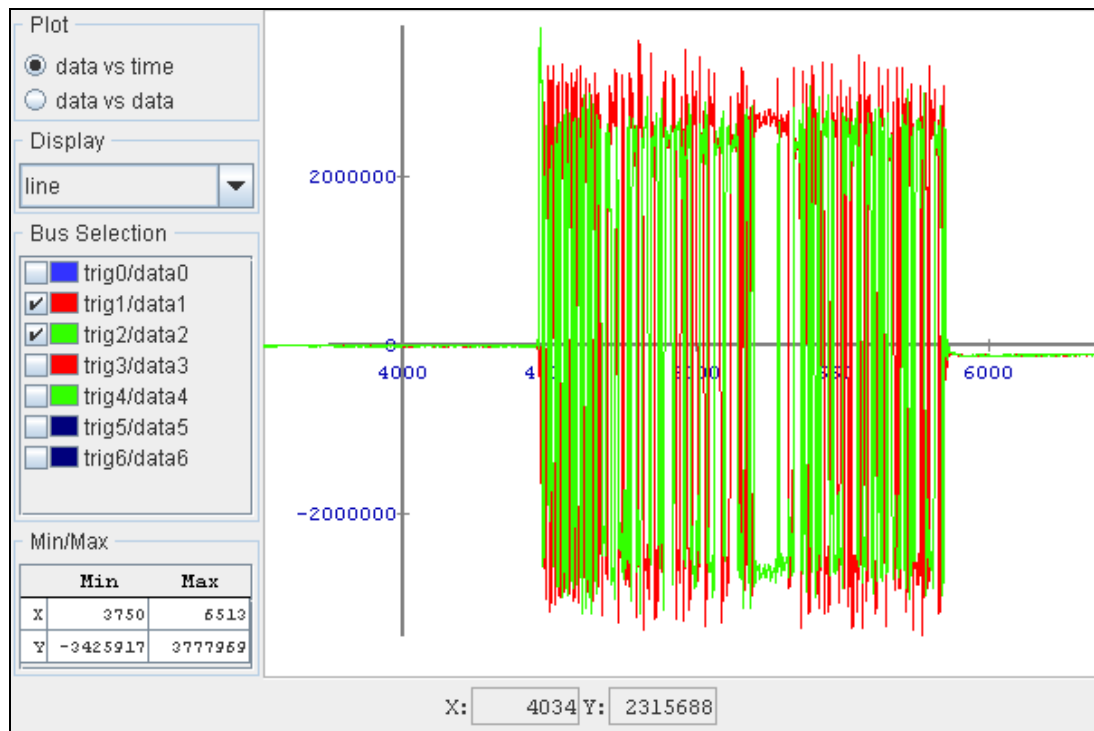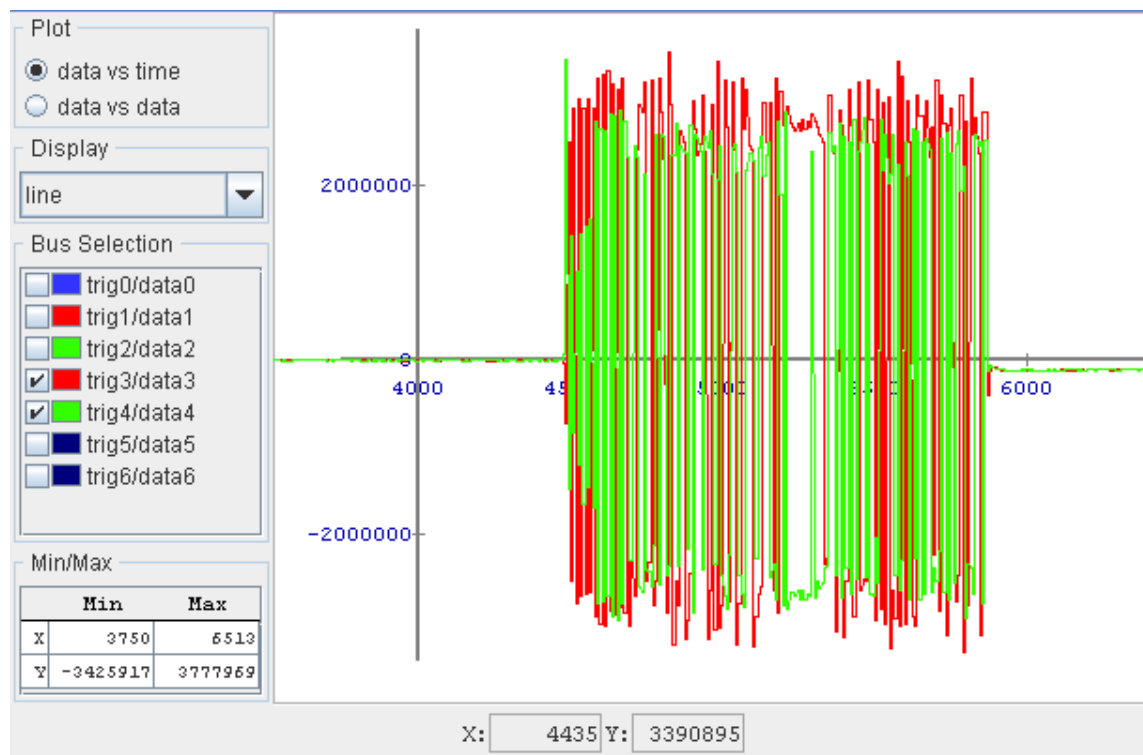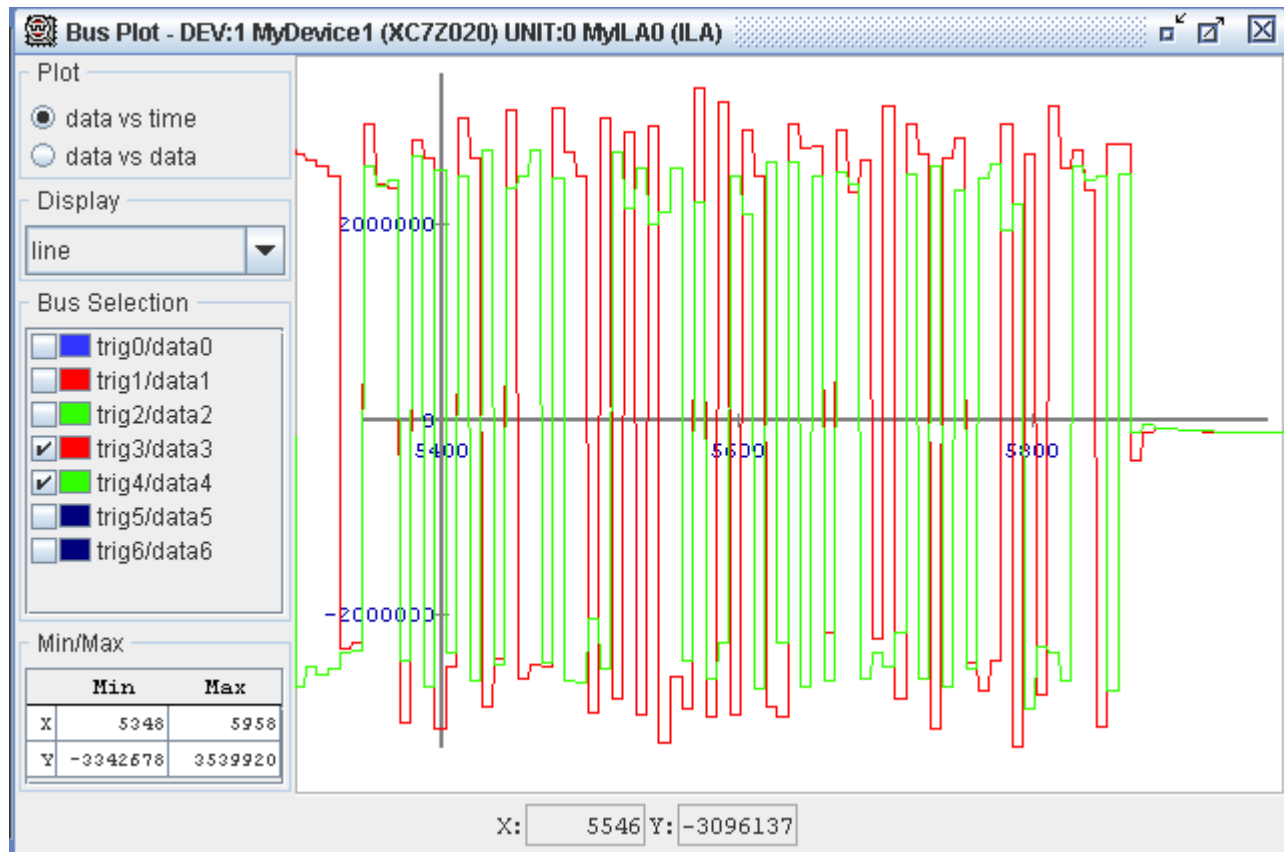

Figure 5-2: Plot of the i and q components of the received QPSK signal post F.O.C. and filtering

5-3: Plot of the i and q components of the received QPSK signal post T.O.C.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

5-4: Post T.O.C. The qpsk waveform starts to develop its square wave-like shape from the sample and hold.

5.2   Exporting into MATLAB

Now that you have verified the received 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. It will be helpful later in your MATLAB code if you rename your **Data Port variables**. Right click on the Ports, and **change the names** to something more descriptive, such as `real_out, rx_i_foc, rx_q_foc, rx_i_toc and rx_q_toc` respectively. If needed, you can use the Simulink model to find which signal each port has.

2. Open the file menu and select **Export**.

3. Click the **ASCII** radio box, select **Bus Plot Buses** under Signals to export, and then click **export**.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

4. It is recommended that you save this file into the project directory with your MATLAB files. Call it something descriptive such as **rx.prn**.

5.3   MATLAB Analysis

The last step is to verify the correctness of the rx pcore by plotting the resultant data received from the core. The rx module designed earlier in MATLAB returns the i and q channels of a QPSK waveform which has gone through Frequency and Timing offset estimation. Therefore, we would expect that a scatter plot of the results should yield an increasingly better constellation plot that has a majority of its signal data at the correct pre-specified symbol locations.

1. Verify this plot by running the following MATLAB script. You may have to change the load values depending on which variables you exported from ChipScope.

```
fid = fopen('rx.prn');
M = textscan(fid,'%d %d %d %d %d %d %d','Headerlines',1);
fclose(fid);
i_in  = double(cell2mat(M(3)));
i_out_foc = double(M{4});
q_out_foc = double(M{5});
i_out_toc = double(M{6});
q_out_toc = double(M{7});

figure(1)
subplot(1,2,1)
scatter(i_out_foc,q_out_foc)
title('Scatter Plot of rx with FOC');
subplot(1,2,2)
scatter(i_out_toc,q_out_toc)
title('Scatter Plot of rx with TOC');

figure(2)
subplot(1,3,1)
plot(i_in);
title('OTA Receive Signal (real part)');
subplot(1,3,2)
plot(i_out_foc);
title('Signal Post FOC (real part)');
subplot(1,3,3)
plot(i_out_foc);
title('Signal Post TOC (real part)');
```

Your plots should look similar to Figures 5-5 and 5-6 below.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper
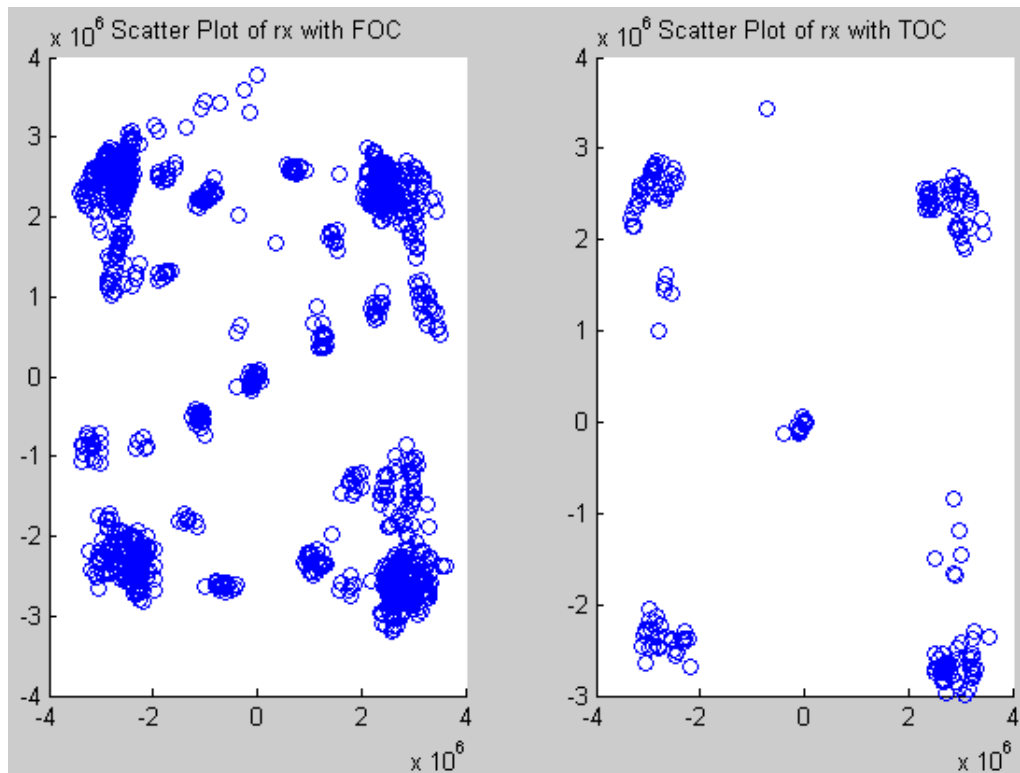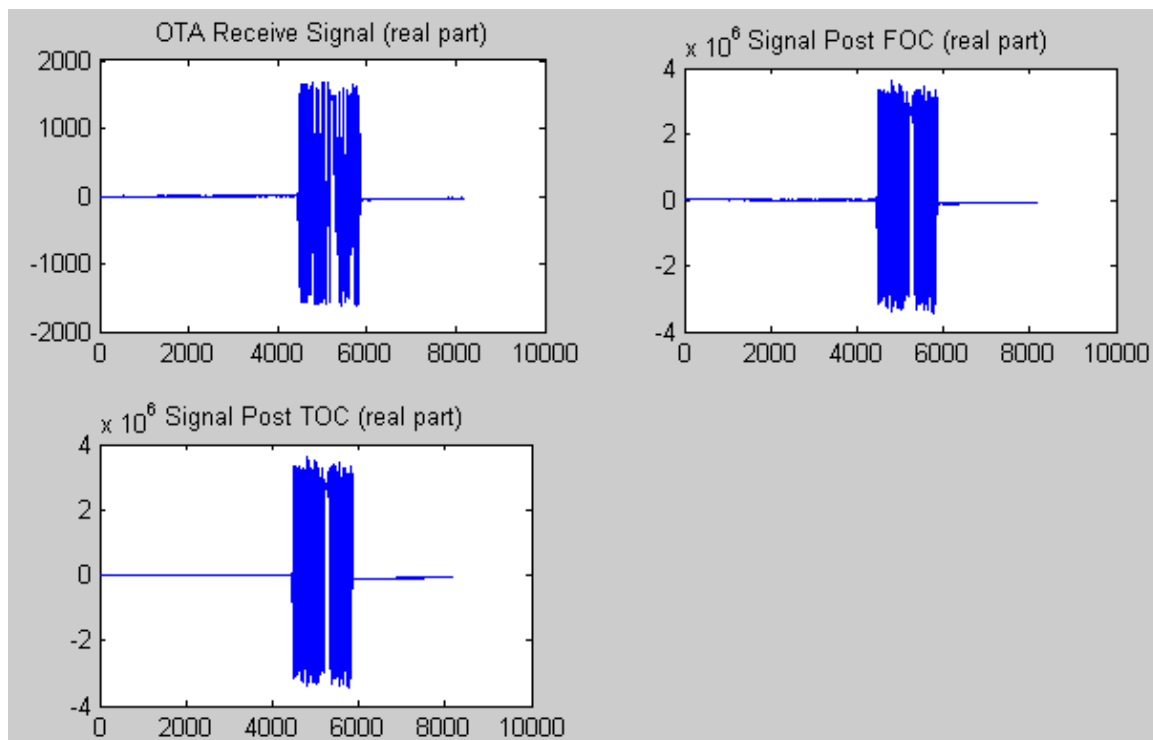

Figure 5-5: Scatter Plot of the transmitted and received QPSK waveform post Frequency Offset Correction.

5-6: Plot of the real component of the QPSK signal before and after frequency and timing estimation.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Appendix A   MATLAB Timing Offset Correction

MATLAB function `qpsk_rx_toc.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QPSK demonstration packet-based transceiver for Chilipepper
% Toyon Research Corp.
% http://www.toyon.com/chilipepper.php
% Created 10/17/2012
% embedded@toyon.com
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%#codegen
function [s_i, s_q, tauh] = qpsk_rx_toc(r_i, r_q, mu_in)


persistent counter
persistent tau
persistent rBuf_i rBuf_q
persistent symLatch_i symLatch_q
persistent tEst

OS_RATE = 8;
if isempty(counter)
    counter = 0;
    tau = 0;
    rBuf_i = zeros(1,4*OS_RATE);
    rBuf_q = zeros(1,4*OS_RATE);
    symLatch_i = 0; symLatch_q = 0;
    tEst = 0;
end

mu = mu_in/2^12;

rBuf_i = [rBuf_i(2:end) r_i];
rBuf_q = [rBuf_q(2:end) r_q];

if counter == 0
    taur = round(tau);
    % if we shift out of the window just exit
    if abs(taur) >= OS_RATE
        tau = 0;
        taur = 0;
    end

    % Determine lead/lag values and compute offset error
    zl_i = rBuf_i(2*OS_RATE+taur-1);
    zo_i = rBuf_i(2*OS_RATE+taur);
    ze_i = rBuf_i(2*OS_RATE+taur+1);
    zl_q = rBuf_q(2*OS_RATE+taur-1);
    zo_q = rBuf_q(2*OS_RATE+taur);
    ze_q = rBuf_q(2*OS_RATE+taur+1);
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
    od_r = ze_i-zl_i;
    od_i = ze_q-zl_q;
    oe_r = zo_i*od_r;
    oe_i = zo_q*od_i;
    % using sign of error in order to make gain invariant
    os = oe_r+oe_i;
    if os < 0
        oe = -1;
    else
        oe = 1;
    end

    % update tau
    tau = tau + mu*oe;

    tEst = tau;

    symLatch_i = zo_i;
    symLatch_q = zo_q;
end

s_i = symLatch_i;
s_q = symLatch_q;
tauh = tEst;

counter = counter + 1;
if counter >= OS_RATE
    counter = 0;
end
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Appendix B   MATLAB Test Bench

MATLAB script `qpsk_tb.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Model/simulation parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sim=1;
OS_RATE = 8;
SNR = 1;
fc = 10e3/20e6; % sample rate is 20 MHz, top is 10 kHz offset
muFOC = floor(.01*2^12)/2^12;
muTOC = floor(.01*8*2^12)/2^12;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize LUTs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
make_srrc_lut;
make_train_lut;
make_trig_lut;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (sim)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Emulate microprocessor packet creation
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % data payload creation
    messageASCII = 'hello world!';
    message = double(unicode2native(messageASCII));
    % add on length of message to the front with four bytes
    msgLength = length(message);
    messageWithNumBytes =[ ...
        mod(msgLength,2^8) ...
        mod(floor(msgLength/2^8),2^8) ...
        mod(floor(msgLength/2^16),2^8) ...
        1 ... % message ID
        message];
    % add two bytes at the end, which is a CRC
    messageWithCRC = CreateAppend16BitCRC(messageWithNumBytes);
    ml = length(messageWithCRC);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % FPGA radio transmit core
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    data_in = 0;
    empty_in = 1;
    tx_en_in = 0;
    numBytesFromFifo = 0;
    num_samp = ml*8*2*2*3;
    x = zeros(1,num_samp);
    CORE_LATENCY = 4;
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
    data_buf = zeros(1,CORE_LATENCY);
    empty_buf = ones(1,CORE_LATENCY);
    clear_buf = zeros(1,CORE_LATENCY);
    tx_en_buf = zeros(1,CORE_LATENCY);
    for i1 = 1:num_samp
        % first thing the processor does is clear the internal tx fifo
        if i1 == 1
            clear_fifo_in = 1;
        else
            clear_fifo_in = 0;
        end

        if i1 == 5 % wait a little bit then begin to load the fifo
            empty_in = 0;
            numBytesFromFifo = 0;
        end

        data_buf = [data_buf(2:end) data_in];
        empty_buf = [empty_buf(2:end) empty_in];
        clear_buf = [clear_buf(2:end) clear_fifo_in];
        tx_en_buf = [tx_en_buf(2:end) tx_en_in];
        [i_out, q_out, re_byte_out, tx_done_out, d1, d2, d3] = ...
            qpsk_tx(data_buf(1),empty_buf(1),clear_buf(1),tx_en_buf(1));
        x_out = complex(i_out,q_out)/2^11;
        x(i1) = x_out;

        %%% Emulate read FIFO AXI interface
        if re_byte_out == 1 && numBytesFromFifo < length(messageWithCRC)
            data_in = messageWithCRC(numBytesFromFifo+1);
            numBytesFromFifo = numBytesFromFifo + 1;
        end
        % processor loaded all bytes into FIFO so begin transmitting
        if numBytesFromFifo == length(messageWithCRC)
            empty_in = 1;
            tx_en_in = 1;
        end
    end

    index = find(abs(x) > sum(SRRC))+24*8; % constant is pad bits
    offset = index(1)+6+length(TB_i)*OS_RATE;
    idx = offset:8:(offset+8*ml*4-1);
    y = x(idx); % four symbos per byte of data
    sc = zeros(1,18*8);
    sc(1:2:end) = real(y);
    sc(2:2:end) = imag(y);
    sh = sign(sc);
    sb = (sh+1)/2;
    d = zeros(1,ml);
    for i1 = 1:ml
        si = sb(1+(i1-1)*8:i1*8);
        d(i1) = bi2de(si);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Emulate channel
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % pad on either side with zeros
    p = complex(zeros(1,100),zeros(1,100));
    xp = [p x p]; % pad

    % Apply frequency offset and receive/over-the-air AWGN
    y = xp.*exp(1i*2*pi*fc*(0:length(xp)-1));
    rC = y/max(abs(y))*.1*2^11; % this controls receive gain
    r = awgn(rC,SNR,0,1);
    r1 = rC;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load Chipscope samples
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (~sim)
    fid = fopen('rx.prn');
    M = textscan(fid,'%d %d %d %d','Headerlines',1);
    fclose(fid);
    is = double(M{3});
    qs = double(M{4});
    r = complex(is,qs);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main receiver core
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r_out = zeros(1,length(r));
s_f = zeros(1,length(r));
s_t = zeros(1,length(r));
s_c = zeros(1,length(r));
f_est = zeros(1,length(r));
t_est = zeros(1,length(r));
for i1 = 1:length(r)+200
    if i1 > length(r)
        r_in = 0;
    else
        r_in = r(i1);
    end
    i_in = round(real(r_in));
    q_in = round(imag(r_in));
    [r_out(i1), s_f(i1), s_c(i1), s_t(i1), t_est(i1), f_est(i1)] = ...
        qpsk_rx(i_in, q_in, floor(muFOC*2^12), floor(muTOC*2^12));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
figure(2)
subplot(2,3,1)
scatter(real(r),imag(r))
title('Pre FOC Signal');
subplot(2,3,4)
plot(real(r_out));
title('Pre FOC Signal (real part)');
subplot(2,3,2)
scatter(real(s_c),imag(s_c))
title('Post FOC Signal');
subplot(2,3,5)
plot(real(s_c));
title('Post FOC Signal(real part)');
subplot(2,3,3)
scatter(real(s_t),imag(s_t))
title('Post TOC Signal');
subplot(2,3,6)
plot(real(s_t));
title('Post TOC Signal(real part)');

figure(3)
plot(t_est);
title('Timing Estimate');
```