Toyon Research Corporation

Lab 5: Carrier Recovery

Chilipepper Tutorial Projects

Version 1.0
1/24/2014

Table of Contents

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Lab 5: Carrier Recovery

Introduction

This lab will show you how to extend the receiver design in the previous lab to allow for Carrier recovery of the transmitted QPSK waveform. 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, verification of the received signal will be done using ChipScope and MATLAB. In future labs, we will continue to increase this verification using 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 2014a
- Xilinx ISE Design Suite 14.7
- 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 MATLAB functions
- Generate an IP core using MATLAB HDL Coder
- 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:

- Implement Carrier Recovery for a received QPSK Waveform
- 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 which are
required to export your design into EDK.

1.1   Supplemental PCores

For this design we require an MCU driver to handle the control signals to and from the Chilipepper,
and an ADC Driver to deinterleave our signal before processing it. Since these PCores have already
been created in the previous labs, we can simply use the same PCores for this lab as well. Refer to
Lab 1 for information on how to create these PCores if needed. Additionally the DC Offset core
designed in the previous lab will be used in this lab as well. Just like the ADC and MCU Driver, we
can copy the core design into our EDK project without having to recreate the HDL Coder project.

1.2   QPSK_RX

The purpose of the QPSK RX core within this lab is to correctly estimate the phase offset of the
carrier modulated received signal. This is an essential first step to recovering the message payload
of the transmitted signal as without proper phase alignment, the receiver will not be able to lock
onto the transmitted carrier signal. Given that the receiver core will consist of several steps along
the path to full message recovery, we will split each individual function into its own MATLAB
function. Each of these functions will then be called sequentially by a central function, called
qpsk_rx.m. The contents of this qpsk_rx.m will thus change with each successive receiver design;
however the individual functions such as the carrier recovery function will stay the same. The
contents of the qpsk_rx function are shown in Figure 1-1 below.

```
%#codegen
function [s_f_i, s_f_q, blinky, f_est] =...
    qpsk_rx(i_in, q_in)

persistent blinky_cnt

if isempty(blinky_cnt)
    blinky_cnt = 0;
end

% frequency offset estimation.
[s_f_i, s_f_q, f_est] = ...
    qpsk_rx_foc(i_in, q_in);

blinky_cnt = blinky_cnt + 1;
if blinky_cnt == 20000000
    blinky_cnt = 0;
end
blinky = floor(blinky_cnt/10000000);
```

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

1. Create a directory for the project under C:\QPSK_Projects\Lab_5.

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

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

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

The function which performs the frequency offset estimates is called qpsk_rx_foc.m. This function is an implementation of a Costas loop. The code required to create the function can be found in Appendix A.

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

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

1.3   MATLAB Test Bench

Now that you have created the code needed to correct for a phase offset, we also need to create a test bench script to test the algorithm. This is done by observing the output graph of a scatter plot of the signal as well as a plot of the estimated phase of the signal. Performing this analysis requires a simulated transmit waveform to fully test the design. Therefore, this script will require several of the MATLAB functions used in Lab 3 to transmit the QPSK waveform. A quick list of the needed files to create the simulated waveform is shown below. The code for the test bench script can be found in Appendix B.

Required files for creating Simulated QPSK waveform
- `make_srrc_lut.m` and `make_trig_lut.m`
- `CreateAppend16BitCRC.m`
- `tx_fifo.m`
- `qpsk_tx.m`
- `qpsk_tx_byte2sym.m`
- `qpsk_srrc.m`
- `mybitget.m`
- `TB_i.m` and `TB_q.m`

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

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

We should expect to see the phase offset change at a constant rate during the time in which the carrier modulated signal is being received. Additionally once the phase lock is established, the scatter plot of the remaining samples should begin to match the expected constellation. For a QPSK

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

waveform, the constellation should have four symbol points, one in each quadrant of the plot. The results of running the testbench script are shown in Figure 1-2 below.
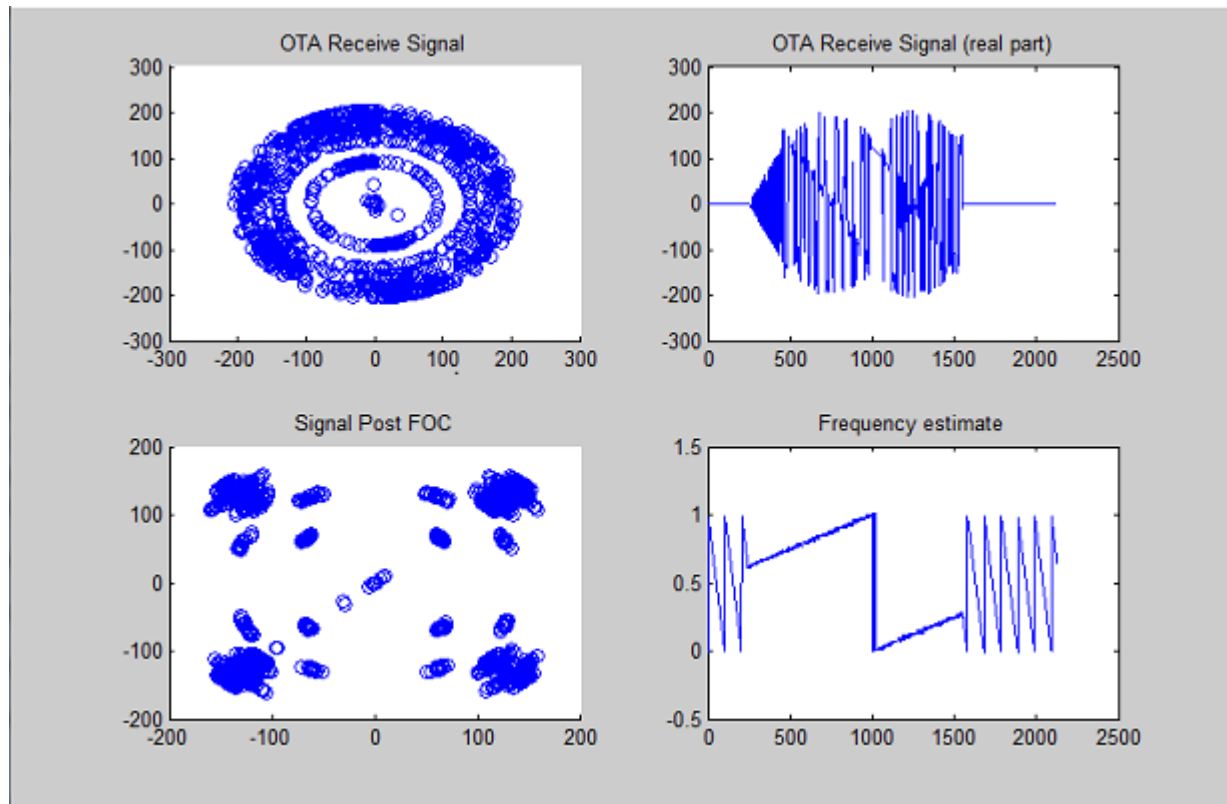

Figure 1-2: Output of QPSK Carrier recovery simulation in MATLAB test bench script

1.4   HDL Coder Project

Now that the MATLAB files have been created, we can turn them into PCores. As mentioned earlier, we will reuse the previously created DC Offset, MCU and ADC Driver PCores, thus the only core we need to create for this lab is the qpsk_rx PCore. Using the same steps outlined in the previous labs, create a new HDL coder project called `qpsk_rx`. Add both your `qpsk_rx.m` file and your `qpsk_rx_tb.m` files to the **MATLAB Function** and **MATLAB Test Bench** categories respectively.

1. Once inside the workflow advisor screen, click on **HDL Code Generation** on the left hand side, and be sure to set the clock to be driven at the **DUT base rate** as in the previous labs.

2. Right-click **Fixed-Point Conversion,** and select **Run to Selected Task.**

3. Both the qpsk_rx.m and the qpsk_rx_foc.m functions require modifications to their variable's proposed types. Modify your HDL Coder design to match the following Fixed-Point conversions for each function.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
▲ Input							
i_in	double	-205	205			Yes	**numerictype(1, 12, 0)**
q_in	double	-204	205			Yes	**numerictype(1, 12, 0)**
▲ Output							
blinky	double	0	0			Yes	numerictype(0, 1, 0)
f_est	double	-0.01⊙	1.01⊙			No	numerictype(1, 14, 12)
s_f_i	double	-160.32⊙	158.9⊙			No	**numerictype(1, 26, 12)**
s_f_q	double	-160.46⊙	157.84⊙			No	**numerictype(1, 26, 12)**
▲ Persistent							
blinky_cnt	double	0	2128			Yes	**numerictype(0, 25, 0)**

Figure 1-3: Proposed variable types for qpsk_rx function

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
▲ Input							
y_i	double	-205	205			Yes	**numerictype(1, 12, 0)**
y_q	double	-204	205			Yes	**numerictype(1, 12, 0)**
▲ Output							
fe	double	-0.01⊙	1.01⊙			No	numerictype(1, 14, 12)
z_i_out	double	-160.32⊙	158.9⊙			No	**numerictype(1, 26, 12)**
z_q_out	double	-160.46⊙	157.84⊙			No	**numerictype(1, 26, 12)**
▲ Persistent							
phi	double	-0.01⊙	1.01⊙			No	numerictype(1, 14, 12)
▲ Local							
bf	double	-157.92⊙	160.46⊙			No	**numerictype(1, 26, 12)**
c	double	-0.01⊙	0.01⊙			No	**numerictype(1, 12, 12)**
e	double	-1	1			Yes	numerictype(1, 2, 0)
f_i	double	-1⊙	1⊙			No	numerictype(1, 14, 12)
f_q	double	-1⊙	1⊙			No	numerictype(1, 14, 12)
ICos	4096 x 1 double	-1	1			No	numerictype(1, 14, 12)
ISin	4096 x 1 double	-1	1			No	numerictype(1, 14, 12)
phi12	double	1	4089			Yes	**numerictype(0, 13, 0)**
phiNew	double	-0.01⊙	1.01⊙			No	**numerictype(1, 26, 12)**
tf	double	-160.32⊙	157.92⊙			No	**numerictype(1, 26, 12)**
ti1	double	-176.32⊙	177.24⊙			No	**numerictype(1, 26, 12)**
ti2	double	-178.43⊙	176.8⊙			No	**numerictype(1, 26, 12)**
time_diff	double	-83.34⊙	82.15⊙			No	**numerictype(1, 26, 12)**
tq1	double	-166.39⊙	173.1⊙			No	**numerictype(1, 26, 12)**
tq2	double	-175.79⊙	177.48⊙			No	**numerictype(1, 26, 12)**
z_i	double	-160.32⊙	158.9⊙			No	**numerictype(1, 26, 12)**
z_q	double	-160.46⊙	157.84⊙			No	**numerictype(1, 26, 12)**

Figure 1-4: Proposed variable types for qpsk_rx_foc function

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

4. Once you have corrected the **Type** setting for all your variables, click **Select Code Generation Target**. Here you can select the FPGA you will use for your design. For this Lab, we will not be using any of the built-in Zynq board functionality within our MATLAB PCores. Therefore you can leave the default settings. Ensure your Workflow settings resemble figure 1-5 below

1-5: Settings for Xilinx Zed Board HDL Coder Design

5. Just below the synthesis tool settings, **rename your PCore** to `qpsk_rx_pcore` or something similar. This is optional as MATLAB will give its default name for each of your cores, as well as a default version, however it is helpful to rename your core for easier netlist configuration later in the lab.

6. Once the platform and synthesis tool are set, you can click **Set Target Interface** to configure the input and output ports of the design. For this Lab, follow the settings shown in Figure 1-6 below.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Port Name	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
◢ Inport			
i_in	numerictype(1, 12, 0)	External Port	
q_in	numerictype(1, 12, 0)	External Port	
◢ Outport			
s_f_i	numerictype(1, 26, 12)	External Port	
s_f_q	numerictype(1, 26, 12)	External Port	
blinky	numerictype(0, 1, 0)	External Port	
f_est	numerictype(1, 14, 12)	External Port	

Figure 1-6: Port Interface settings for the qpsk_rx HDL Coder project

7. Once the ports are set, right-click **HDL Code Generation** and select Run This Task. This will create a PCore for your design that can be used directly within Xilinx EDK. By default, the PCore is created in <Project Directory/MATLAB folder/codegen/ipcore>.

8. Once the PCore has been created, make a **new EDK project** using the same method used in the previous lab. Be sure that you **import** the correct system configuration file.

9. Once the project is created, **copy each of the PCore folders** from the MATLAB directory into the PCores folder of your **EDK Project**. Don't forget to also copy any previously created cores you may be reusing as well. Then simply select project -> **rescan user repositories** to show your newly added user PCores within your EDK project.

Configure Cores and Export Design   Step 2
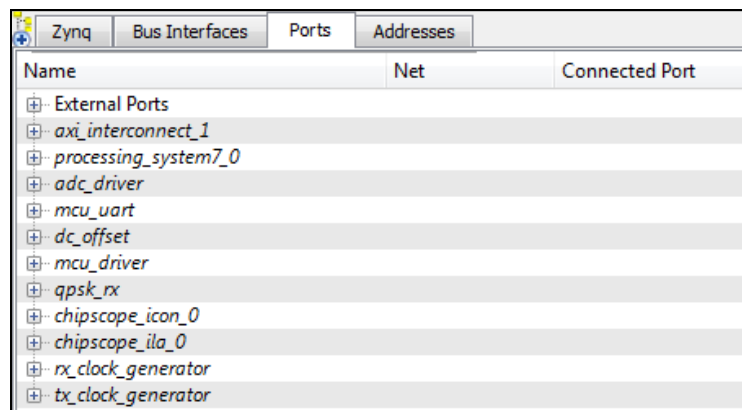
This section will show you how to integrate your PCores into your FPGA design using EDK. There are several components that must be configured for the design of this project. A quick list of the cores needed is given below. Refer to lab 0 sections 4.3 and 5.1 for information on how to add cores to the design.

2.1   Needed IP Cores

- ADC Driver

- MCU Driver

- MCU UART

- DC Offset

- QPSK RX

- Clock Generator (one for RX and one for TX)

- Processing System

- AXI Interconnect

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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. First we will configure the `rx_iq_sel`, the `rxd` and the `bliky` pins. Each of these pins can be assigned as **External ports**.

2. Next are the `rx_i` and the `rq_q` output pins. Connect these pins to the `i_in` and `q_in` pins of the dc_offset PCore.

3. Connect the IPCORE_RESETN port to the processing_system7 FCLK_RESET0_N port.

4. The `IPCORE_CLK` pin can be skipped for now and will be connected later in **section 2.5**

2.3   Configuring the MCU Driver Port

Expand the **MCU Driver** core. There are 9 individual I/O pins which need to be routed on this core.

1. Configuring this core is very simple as **all of the pins** with the exception of the `IPCORE_CLK` and the `IPCORE_RESETN` are simply **assigned** as **external ports**.

2. Connect the IPCORE_RESETN port to the processing_system7 FCLK_RESET0_N Port and skip the `IPCORE_CLK` for now.

2.4   Configuring the MCU UART

1. Under the Communications Low-Speed section, add the AXI UART (Lite) to your design

2. Name the core mcu_uart as shown in Figure 2-1. Keep all configuration settings as default.

3. This core requires no other customization; just verify the RX and TX pins are set as External ports.

2.5   Configuring the DC Offset

Expand the **DC Offset** core. There are 7 individual I/O pins which need to be routed on this core.

1. If the ADC driver was previously configured correctly, the i_in and q_in pins of the dc_offset core should already be set.

2. The i_out and q_out pins should be connected to the qpsk_rx i_in and q_in pins respectively.

3. Set the blinky pin as an External port.

4. Connect the IPCORE_RESETN port to the processing_system7 FCLK_RESET0_N Port and skip the `IPCORE_CLK` for now.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

2.6   Configuring the QPSK RX

Expand the **QPSK RX** core. There are 8 individual I/O pins which need to be routed on this core.

1. If the DC Offset core was previously configured correctly, the i_in and q_in pins of the qpsk_rx core should already be set.

2. Set the blinky pin as an External port.

3. The s_f_i, s_f_q and f_est pins should be left unconnected for now and will eventually be connected to ChipScope for further analysis.

4. Connect the IPCORE_RESETN port to the processing_system7 FCLK_RESET0_N Port and skip the `IPCORE_CLK` for now.

2.7   Configuring the TX Clock Generator IP Core

The TX Clock Generator is used in this project to distribute the appropriate clock signals to each of the PCores required for Chilipepper initialization, as well as any external hardware which may require a clock signal. For this project, the TX 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; 1 PCore (MCU Driver) and the `TX_CLK` and `RX_CLK` signals. The TX and RX clock signals are used to latch data from the TXD and RXD lines to the DAC and ADC respectively on the radio board. Although no DAC is used within the design, the clock is required for proper initialization of the Chilipepper FMC. For this lab, the Clock Generator has been named tx_clock_generator.

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

- Input Clock Frequency of **40Mhz**

- CLKOUT0 Required Frequency of **20MHz**, 0 Phase, **PLLE0** group and **Buffered True**

- CLKOUT1 Required Frequency of **40MHz**, 180 Phase, **PLLE0** group and **Buffered True**

- CLKOUT2 Required Frequency of **40Mhz**, 0 Phase, **PLLE0** group and **Buffered True**

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 $\longrightarrow$ External Ports

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

- CLKOUT0 $\longrightarrow$ mcu:: IPCORE_CLK

- CLKOUT1 $\longrightarrow$ External Ports

- CLKOUT2 $\longrightarrow$ External Ports

- RST $\longrightarrow$ net_gnd

- LOCKED $\longrightarrow$ External Port

2.8   Configuring the RX Clock Generator IP Core

In addition to the TX Clock Generator, another clock generator is required for this design. As mentioned in Lab 2 and the Chilipepper User's Guide, the receiver chain is to be clocked using the RX return clock on the Chilipepper board to ensure data is latched properly from the ADC. In this design, there are three cores which must be clocked using the RX return clock; therefore a new clock generator called rx_clock_generator is used to distribute the clock signal.

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

- Input Clock Frequency of **40Mhz**

- CLKOUT0 Required Frequency of **40MHz**, 180 Phase, **PLLE0** group and **Buffered True**

- CLKOUT1 Required Frequency of **20MHz**, 180 Phase, **PLLE0** group and **Buffered True**

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 $\longrightarrow$ External Ports

- CLKOUT0 $\longrightarrow$ adc_driver::IPCORE_CLK

- CLKOUT1 $\longrightarrow$ dc_offset:: IPCORE_CLK and qpsk_rx::IPCORE_CLK

- RST $\longrightarrow$ net_gnd

- LOCKED $\longrightarrow$ External Port

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Figure 2-2: Clock Generator port configurations

Be sure your External Port pins, as well as your PCores match the names shown in the figures above.

2.9   Pin Assignments

Once the clock generator is configured correctly, the `IPCORE_CLK` for the other cores should be set as well. The next step is to setup the **pin assignments** for the external ports.

1. Open the **Project** tab.

2. Double-click on the **UCF File: data\system.ucf** from this panel, to open the constraints file.

3. Fill in the pin out information for your design using Figure 2-3 below as a reference.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```
######################################### PL clocks and reset #########################################
NET tx_clock_generator_CLKIN_pin            LOC = D18   | IOSTANDARD = LVCMOS25;
NET tx_clock_generator_CLKIN_pin            TNM_NET = tx_clock_generator_CLKIN;
TIMESPEC TS_tx_clock_generator_CLKIN = PERIOD tx_clock_generator_CLKIN 40.000 MHz;
######################################################################################################
NET rx_clock_generator_CLKIN_pin            LOC = L18   | IOSTANDARD = LVCMOS25;
NET rx_clock_generator_CLKIN_pin            TNM_NET = rx_clock_generator_CLKIN;
TIMESPEC TS_rx_clock_generator_CLKIN = PERIOD rx_clock_generator_CLKIN 40.000 MHz;
###################################Chilipepper Rx and Tx clock lines###################################
NET tx_clock_generator_tx_clk_pin           LOC = C17        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET tx_clock_generator_rx_clk_pin           LOC = J18        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
#########################################Rx – FMC interface at 2.5V ###################################
NET adc_driver_rx_iq_sel_pin                LOC = N19        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[0]                   LOC =M21         | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[1]                   LOC = J21        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[2]                   LOC = M22        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[3]                   LOC = J22        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[4]                   LOC = T16        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[5]                   LOC = P20        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[6]                   LOC = T17        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[7]                   LOC = N17        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[8]                   LOC = J20        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[9]                   LOC = P21        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[10]                  LOC = N18        | IOSTANDARD = LVCMOS25;
NET adc_driver_rxd_pin[11]                  LOC = J16        | IOSTANDARD = LVCMOS25;
######################################### MCU Interface ###############################################
NET mcu_uart_RX_pin                         LOC = R19        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_uart_TX_pin                         LOC = L21        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_mcu_reset_out_pin            LOC = K20        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_tx_en_pin                    LOC = D22        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_tr_sw_pin                    LOC = D20        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_rx_en_pin                    LOC = C22        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_pa_en_pin                    LOC = E21        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET mcu_driver_init_done_pin                LOC = K19        | IOSTANDARD = LVCMOS25;
######################################### LEDs ########################################################
NET tx_clock_generator_LOCKED_pin           LOC = T22        | IOSTANDARD =LVCMOS33;  # "LD0"
NET rx_clock_generator_LOCKED_pin           LOC = T21        | IOSTANDARD =LVCMOS33;  # "LD1"
NET adc_driver_blinky_pin                   LOC = U22        | IOSTANDARD =LVCMOS33;  # "LD2"
NET mcu_driver_blinky_pin                   LOC = U21        | IOSTANDARD =LVCMOS33;  # "LD3"
NET dc_offset_blinky_pin                    LOC = V22        | IOSTANDARD =LVCMOS33;  # "LD4"
NET qpsk_rx_blinky_pin                      LOC = W22        | IOSTANDARD =LVCMOS33;  # "LD5"
```

Figure 2-3: EDK project pin assignments

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

2.10 Adding ChipScope Peripheral

The last step is to setup the ChipScope peripheral which will be used to capture the output of the qpsk_rx core for further analysis in MATLAB.

1. Select Debug -> **Debug Configuration** from the top menu

2. Click the **Add ChipScope Peripheral** button on the bottom left hand side of the screen

3. Select To **monitor arbitrary system level signals** (middle option) from the list.

4. Add the s_f_i, s_f_q and f_est pins from the qpsk_rx Port. Additionally, you should set the clock to the same clock used for the core, which for this design is rx_clock_generator_clockout_1.

5. (optional) you can also add the rx_i and rx_q signals from the DAC Driverto see the before and after affect of the frequency offset correction.

6. Click ok to finish configuration of your ChipScope peripheral. Your new port list should look similar to Figure 2-4 below. Be sure your Clock and qpsk_rx ports have the ChipScope peripherals in the correct locations.

qpsk_rx				
IPCORE_CLK	rx_clock_generator::CLKOUT1	rx_clock_generator_CLKOUT1		I
IPCORE_RESETN	processing_system7_0::FCLK_RESET0_N	processing_system7_0_FCLK_...		I
i_in	dc_offset::i_out	dc_offset_i_out		I
q_in	dc_offset::q_out	dc_offset_q_out		I
s_f_i	chipscope_ila_0::TRIG0	qpsk_rx_s_f_i_to_chipscope_il...		O
s_f_q	chipscope_ila_0::TRIG0	qpsk_rx_s_f_q_to_chipscope_i...		O
blinky	External Ports::qpsk_rx_blinky_pin	qpsk_rx_blinky		O
f_est	chipscope_ila_0::TRIG0	qpsk_rx_f_est_to_chipscope_il...		O
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1	Connected to BUS axi_interc...		
chipscope_icon_0				
chipscope_ila_0				
rx_clock_generator				
CLKIN	External Ports::rx_clock_generator_CLKIN_pin	rx_clock_generator_CLKIN		I
CLKOUT0	adc_driver::IPCORE_CLK	rx_clock_generator_CLKOUT0		O
CLKOUT1	dc_offset::IPCORE_CLK qpsk_rx::IPCORE_CLK chipscope_ila_0::CLK	rx_clock_generator_CLKOUT1		O
RST	net_gnd	net_gnd		I
LOCKED	External Ports::rx_clock_generator_LOCKED_pin	rx_clock_generator_LOCKED		O
tx_clock_generator				
CLKIN	External Ports::tx_clock_generator_CLKIN_pin	tx_clock_generator_CLKIN		I
CLKOUT0	mcu_driver::IPCORE_CLK	tx_clock_generator_CLKOUT0		O
CLKOUT1	External Ports::tx_clock_generator_tx_clk_pin	tx_clock_generator_CLKOUT1		O
CLKOUT2	External Ports::tx_clock_generator_rx_clk_pin	tx_clock_generator_CLKOUT2		O
RST	net_gnd	net_gnd		I
LOCKED	External Ports::tx_clock_generator_LOCKED_pin	tx_clock_generator_LOCKED		O

Figure 2-4: Ports list after adding ChipScope peripheral to monitor qpsk_rx signals

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Once completed, 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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 SDK folder in the same directory as your MATLAB 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 QPSK RX project.

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

2. Name the project "qpsk_rx" or something similar and leave the other settings at their defaults. Click next.

3. On the next screen, be sure to select **Hello World** from the list of Available Templates.

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

5. If you navigate into the qpsk_rx project folder, and into the src folder, you should see a `helloworld.c` file. Feel free to rename this file to main.c or something more appropriate.

6. **Double click** the file to open it and **replace** all of its contents with the code in Figure 3-1.

7. **Download** the **Chilipepper.c** and **Chilipepper.h** files from the GitHub repository[1] if you don't already have them. Copy them into the source directory with your main.c file.

8. Open the Chilipepper.c file and modify it for this lab. The only PCores that should be defined at the top of the file are MCU_DRIVER, DC_OFFSET, and MCU_UART.

Note	You may be required to add the Math Library to the project to define the pow function used in the Chilipepper.c Library file. If so, follow the optional step 9 listed below.

9. (Optional) Click on **Project →Properties.** Open the **C/C++ Build** arrow and click the settings option. Under **ARM gcc linker**, click the Libraries folder. Click the   button, type the letter **m** into the prompt and select ok. **Apply** and hit ok.

---

[1] Can be found at https://github.com/Toyon/Chilipepper/tree/QPSK_pcore/ChilipepperSupport/Library%20Files

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```c
#include <stdio.h>
#include "platform.h"
#include "chilipepper.h"
#include "xuartps.h"

XUartPs uartPs;
XUartPs_Config *pUartPsConfig;

int main()
{

        init_platform();

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

        Chilipepper_SetPA(0);
        Chilipepper_SetTxRxSw(1);    // 0- transmit, 1-receive
        Chilipepper_SetDCOC(1);      // enable dc offset correction

        while (1)
        {
                Chilipepper_ControlAgc(); //update the Chilipepper AGC
        }
        cleanup_platform();
        return 0;
}
```

Figure 3-1: main.c file for DC Offset Correction SDK Project

3.2   Programming the Board

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 Getting Started Guide[2]* 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.

---

[2] Can be found at https://github.com/Toyon/Chilipepper/tree/master/QPSK_Radio/DemoFilesAndDocumentation

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.
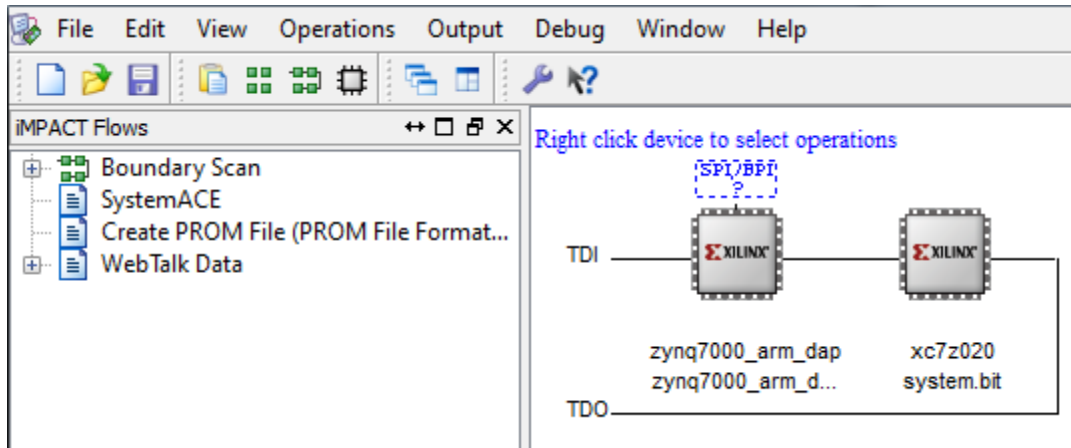
3-2: configuration for Zed Board System.bit file

9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.
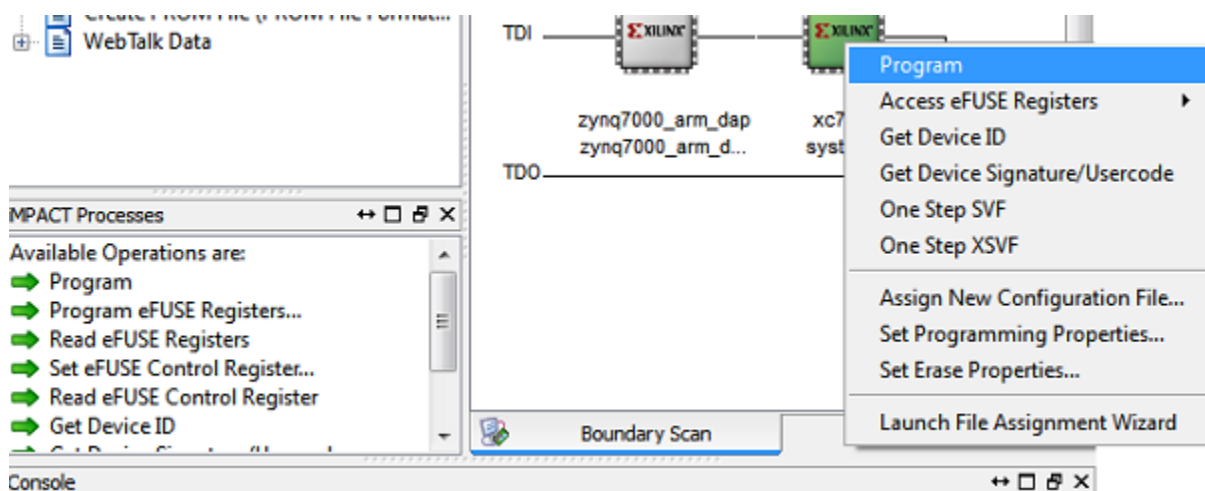
Figure 3-3: iMPACT configuration screen

3.3   Debugging with SDK

If the hardware design is correct, you should see a blue light on the ZED Board 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 `qpsk_rx` project folder and selecting **Debug As → Launch on Hardware (GDB)**.

2. You should now be taken to a screen which shows the `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, as well as the Blinking LEDs on the FPGA from the PCore blinky pins.

Testing and Design Verification   Step 4

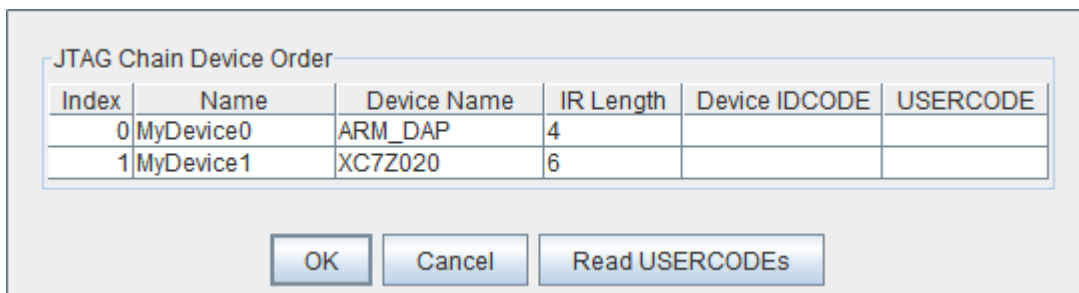## 4.1 Verification with ChipScope Pro

There are several methods available for verifying the MATLAB functions. For verification of the qpsk_rx_foc design, ChipScope is recommended as it provides the most useful view of the signal port frequency offset, especially when compared to the output of the dc offset.

1. To verify the qpsk_rx signals, you will need to open **ChipScope Pro Analyzer**. Be sure that the JTAG cable is connected to the FPGA board properly.

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.

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

OK   Cancel   Read USERCODEs

Note	If you receive an error from ChipScope stating that you either cannot detect or cannot open the cable, try using the optional Step 3 to configure your cable setup correctly.

3. (**Optional**)Click JTAG Chain in the top menu selection. Select the option for **Open Plug-in**... You will be greeted with a Plug-in Parameters screen. Enter the following in the box, and hit ok. "**xilinx_tcf URL=tcp::3121**". Then click the open cable button and proceed as usual.

4. Select ok to get to the Analyzer main screen. Open the **file menu** and select **Import**.

5. Click **Select New File**, and browse to the location of your ChipScope **CDC file**, which is located in the <EDK/implementation/chipscope_ila_0_wrapper> folder of your project directory. This file was created for you when you generated your bit file in EDK, assuming you added the ChipScope peripheral appropriately. It tells the ChipScope program how to interpret the data it is receiving from the JTAG port.

6. On the Bus Plot screen, you can view the I and Q channel signals that you connected to your ChipScope peripheral previously. Right click on a signal to change its features such as bus radix, name or color. For this Lab, both signals should be set to the signed decimal bus radix.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

7. 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 Figures 4-1 and 4-2.
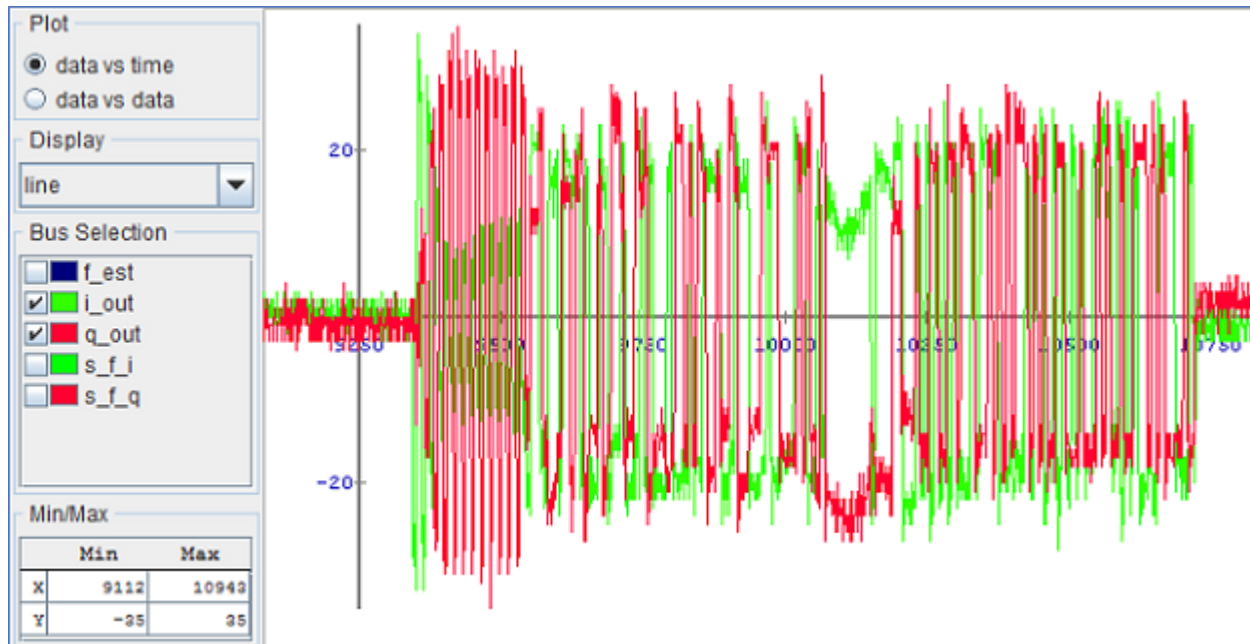
Figure 4-1: Plot of the i and q components of the received QPSK signal pre F.O.C.
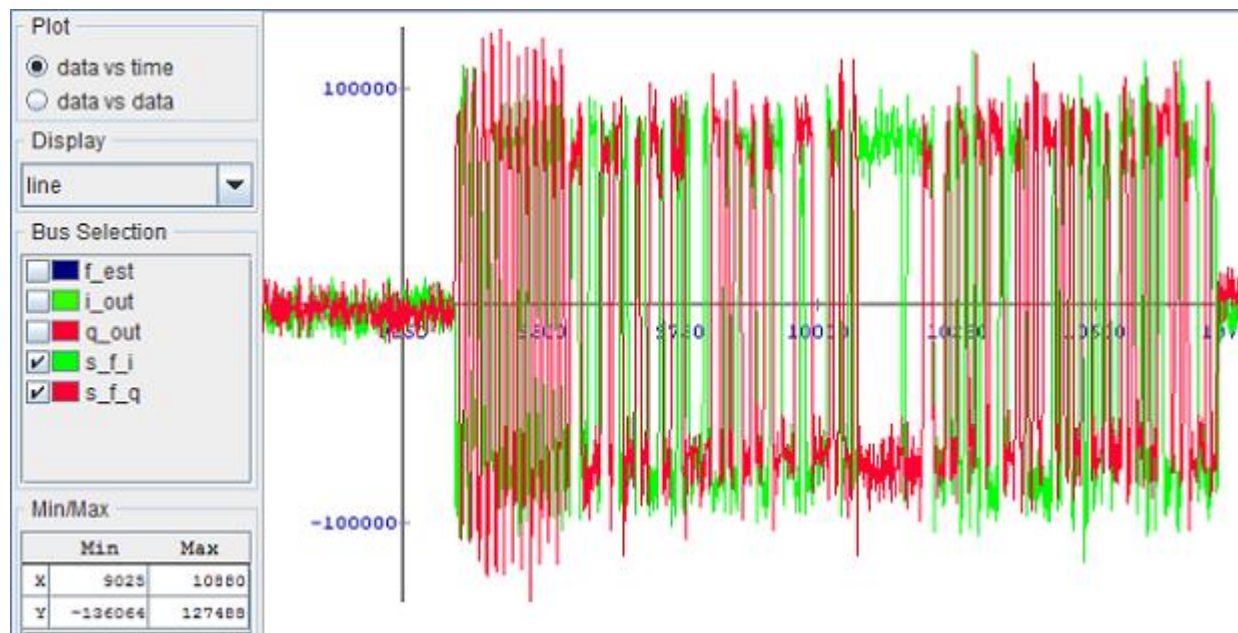
Figure 3-2: Plot of the i and q components of the received QPSK signal post F.O.C.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

When compared to the dc offset output, you can see that one aspect of the qpsk_rx_foc function is that it greatly amplifies the received signal. However despite this amplification, the relative structure of the signal is very similar. In addition, the plots below shows that a carrier phase lock was established, and as mentioned earlier a scatter plot of all the sample points is beginning to match the expected constellation.
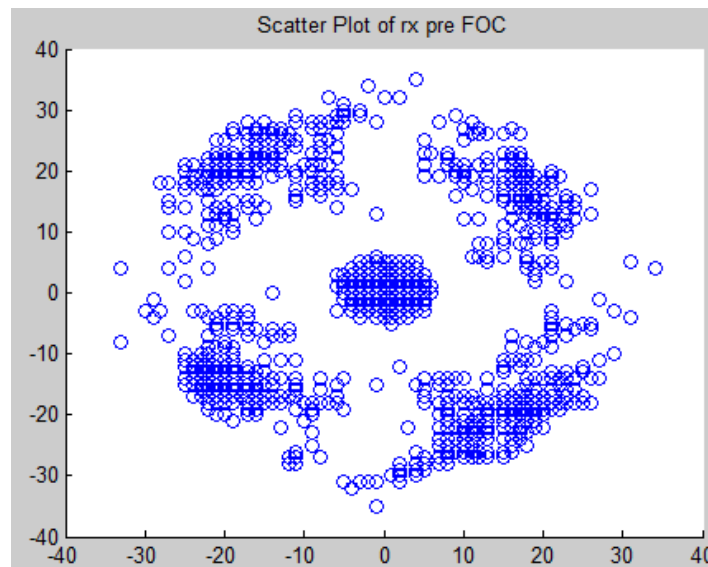
4-3: Scatter plot of the received QPSK signal before applying the frequency offset correction.
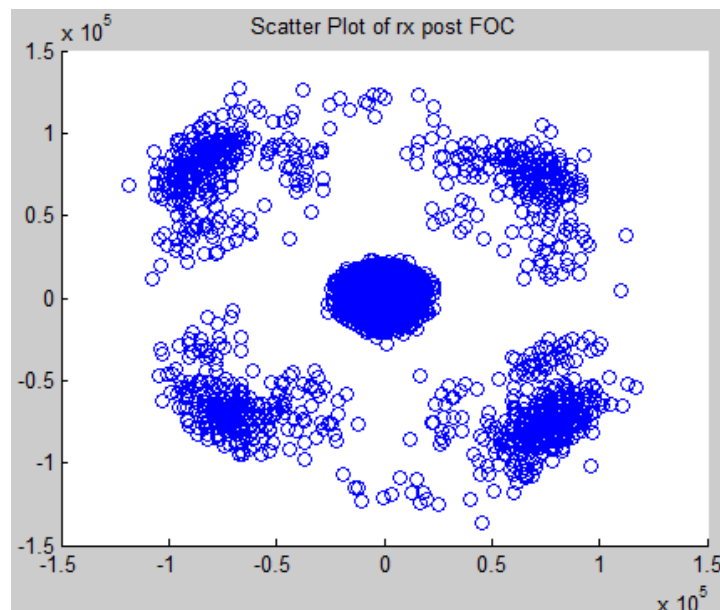
4-4: Scatter plot of the received QPSK signal after applying the frequency offset correction.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Appendix A   MATLAB Frequency Offset

MATLAB function `qpsk_rx_foc.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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Demonstration of a Costas Loop. Refer to:
% Telecommunications Breakdown: Concepts of Communication Transmitted via
% Software-Defined Radio C. Richard Johnson
% We employ a hard-decision feedback in order to get rid of the loop
% filters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%#codegen
function [z_i_out, z_q_out, fe] = qpsk_rx_foc(y_i, y_q)


persistent phi

lSin = SIN;
lCos = COS;

if isempty(phi)
    phi = 0;
end


% create the VCO signal
if phi >= 1
    phi = phi - 1;
end
if phi < 0
    phi = phi + 1;
end
phi12 = round(phi*2^12)+1;
if phi12 >= 2^12
    phi12 = 1;
end
if phi12 < 0
    phi12 = 0;
end
f_i = lCos(phi12+1);
f_q = lSin(phi12+1);
ti1 = y_i*f_i;
ti2 = y_q*f_q;
tq1 = y_q*f_i;
tq2 = -y_i*f_q;
z_i = ti1 + ti2;
z_q = tq1 + tq2;
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
% generate the error term to drive VCO generateion
if z_q < 0
    tf = -z_i;
else
    tf = z_i;
end
if z_i < 0
    bf = -z_q;
else
    bf = z_q;
end
% using sign of error in order to make it gain invariant
time_diff = tf-bf;
if time_diff < 0
    e = -1;
else
    e = 1;
end

% update with hard coded mu 40/2^12
c = (0.009765625)*e;
phiNew = phi - c;
phi = phiNew;

fe = phiNew;

z_i_out = z_i;
z_q_out = z_q;
```

Appendix B   MATLAB QPSK RX Test Bench Script

MATLAB script `qpsk_tb.m`

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Model/simulation parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
OS_RATE = 8;
SNR = 100;
fc = 10e3/20e6; % sample rate is 20 MHz, top is 10 kHz offset
sim = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize LUTs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
make_srrc_lut;
make_trig_lut;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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;
store_byte = 0;
numBytesFromFifo = 0;
num_samp = ml*8*2*2*3;
x = zeros(1,num_samp);
CORE_LATENCY = 4;
data_buf = zeros(1,CORE_LATENCY);
store_byte_buf = zeros(1,CORE_LATENCY);
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
clear_buf = zeros(1,CORE_LATENCY);
tx_en_buf = zeros(1,CORE_LATENCY);
re_byte_out(1) = 0;
reset_fifo = 0;
byte_request = 0;
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

    data_buf = [data_buf(2:end) data_in];
    store_byte_buf = [store_byte_buf(2:end) store_byte];
    clear_buf = [clear_buf(2:end) clear_fifo_in];
    tx_en_buf = [tx_en_buf(2:end) tx_en_in];

    [new_data_in, empty_in, byte_recieved, full, percent_full] = ...
    tx_fifo(byte_request, store_byte_buf(1), data_buf(1), reset_fifo);

    [i_out, q_out, tx_done_out, request_byte, clear_fifo_in_done] = ...
        qpsk_tx(new_data_in,empty_in,clear_buf(1),tx_en_buf(1));
    x_out = complex(i_out,q_out)/2^11;
    x(i1) = x_out;
    byte_request = request_byte;

    %%% Emulate write to FIFO interface
    if mod(i1,8) == 1 && numBytesFromFifo < length(messageWithCRC)
        data_in = messageWithCRC(numBytesFromFifo+1);
        numBytesFromFifo = numBytesFromFifo + 1;
    end
    %%% Software lags a but on the handshaking signals %%%
    if (0 < mod(i1,8) && mod(i1,8) < 5) && tx_en_in == 0
        store_byte = 1;
    else
        store_byte = 0;
    end
    % processor loaded all bytes into FIFO so begin transmitting
    if (numBytesFromFifo == length(messageWithCRC) && mod(i1,8) > 5)
        empty_in = 1;
        tx_en_in = 1;
    end
end
if ~sim % load data that was transmitted and captured from chipscope
    if 1
        fid = fopen('tx.prn');
        M = textscan(fid,'%d %d %d %d %d %d %d %d %d %d','Headerlines',1);
        fclose(fid);
        iFile = double(M{3})'/2^11;
        qFile = double(M{4})'/2^11;
    else
        M = load('dac.prn');
        if M(1,end-1) == 0
            iFile = M(1:2:end,end)'/2^11;
            qFile = M(2:2:end,end)'/2^11;
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
        else
            qFile = M(1:2:end,end)'/2^11;
            iFile = M(2:2:end,end)'/2^11;
        end
    end
    x = complex(iFile,qFile);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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);
r = rC;
if ~sim
    %M = load('rx.prn');
    fid = fopen('rx.prn');
    %M = textscan(fid,'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d',...
%d','Headerlines',1);
    M = textscan(fid,'%d %d %d %d','Headerlines',1);
    fclose(fid);
    is = double(M{3});
    qs = double(M{4});
    r = complex(is,qs);
    figure(1)
    subplot(2,1,1);
    plot(is);
    subplot(2,1,2);
    plot(qs)
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main receiver core
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r_out = zeros(1,length(r));
s_f_i = zeros(1,length(r));
s_f_q = zeros(1,length(r));
fe = 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) = real(complex(i_in,q_in));
```

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
    [dc_i_out, dc_q_out, rssi_out, rssi_en_out, dir_out, dir_en_out] = ...
        dc_offset_correction(i_in, q_in, mod(i1,2), 500, 1500, +(i1>3000));

    [s_f_i(i1), s_f_q(i1), blinky, fe(i1)] =...
        qpsk_rx(dc_i_out, dc_q_out);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)
subplot(2,2,1)
scatter(real(r),imag(r))
title('OTA Receive Signal');
subplot(2,2,2)
plot(real(r_out));
title('OTA Receive Signal (real part)');
subplot(2,2,3)
scatter(s_f_i,s_f_q)
title('Signal Post FOC');
subplot(2,2,4)
plot(fe);
title('Frequency estimate');
```