

Toyon Research Corporation

# Lab 2: Receive Tone

Chilipepper Tutorial Projects

Version 1.2

1/9/2014

# Table of Contents

---

Introduction .....	3
Procedure.....	3
Objectives .....	3
Generate HDL code .....	4
1.1  ADC Driver MATLAB Files .....	4
1.2  MCU Driver MATLAB Files .....	5
1.3  HDL Coder Project .....	5
Configure Cores and Export Design .....	9
2.1  Needed IP Cores .....	9
2.2  Configuring the ADC Driver Port .....	10
2.3  Configuring the MCU Driver Port .....	10
2.4  Configuring the MCU UART.....	10
2.5  Configuring the Clock Generator IP Core.....	10
2.6  Pin Assignments.....	12
2.7  Adding ChipScope Peripheral .....	13
Create software project .....	15
3.1  Creating a new C Project.....	15
3.2  Programming the Board.....	16
3.3  Debugging with SDK.....	17
Testing and Design Verification.....	19
4.1  Verification with ChipScope Pro .....	19
4.2  MATLAB Analysis.....	20

# Lab 2: Receive Tone

---

## Introduction

This lab will show you how to receive a single frequency tone on an FPGA Mezzanine Card (FMC) radio board using the Xilinx Zed Board FPGA and the Toyon Chilipepper FMC. The Analog to Digital Conversion (ADC) used to receive the tone will take place on the Chilipepper board. The FMC initialization and microcontroller (MCU) signal control will be handled in software using the Xilinx Software Development Kit (SDK). This lab assumes prior knowledge of the workings of HDL Coder as well as the Xilinx EDK environment. It is recommended that you complete lab 0 and lab 1 before completing this lab.

This lab is created using:

- MATLAB 2013b
- 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:

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

## Objectives

After completing this lab, you will be able to:

- Create a Simulink model to control hardware on an FMC
- Create a software application to test your created FPGA hardware
- Verify your results in ChipScope or export to MATLAB

## 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 ADC Driver MATLAB Files

The purpose of the ADC driver within this lab is to properly format the I/Q interleaved signal provided by the Chilipepper. The Chilipepper FMC provides the signal sample data, as well as a single toggle line to tell the FPGA whether the current input is I channel or Q channel. Once separated, the I and Q channels can be used by other cores to process the input signal. Given that both the I and Q channels are clocked at the same rate, the ADC driver PCore must be clocked at twice the frequency of the remaining receiver chain to deinterleave the data properly. The MATLAB function used to create the ADC Driver PCore is shown in figure 1-1 below.

```
%#codegen
function [rx_i, rx_q, blinky] = adc_driver(rxd, rx_iq_sel)
% deinterleave txd into I and Q channels
persistent i_out q_out q_out_delay blinky_cnt

if isempty(i_out)
    i_out = 0;
    q_out = 0;
    q_out_delay = 0;
    blinky_cnt = 0;
end

if rx_iq_sel == 0
    q_out_delay = rxd;
else
    i_out = rxd;
    q_out = q_out_delay;
end
% ADC runs at 2x the rate the output changes at
rx_i = i_out;
rx_q = q_out;

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

Figure 1-1: MATLAB Driver to deinterleave I and Q channels

1. Create a directory for the project under C:\QPSK\_Projects\Lab\_2.

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

2. Create a folder inside this directory called MATLAB.
3. Create a new **MATLAB function** with the contents of Figure 1-1.
4. **Save** this function as `adc_driver.m` inside the MATLAB folder.

In addition to the MATLAB function, a script must be created both for testing the MATLAB algorithm, and as required for the PCore generation. Figure 1-2 below shows a basic test bench for the adc driver function.

```
for i1 = 0:2:512
    rxd = mod(i1,4)/2;
    rx_iq_sel = mod(i1,4)/2;
    [ rx_i(i1+1), rx_q(i1+1) ] = adc_driver( rxd, rx_iq_sel );
    [ rx_i(i1+2), rx_q(i1+2) ] = adc_driver( rxd, rx_iq_sel );
end

plot(rx_i)
plot(rx_q)
```

Figure 1-2: MATLAB code for `adc_driver` test bench script

5. Create a new **MATLAB script** with the contents of Figure 1-2.
6. **Save** this script as `adc_driver_tb.m` inside the MATLAB folder.
7. **Run** this script in MATLAB to test the deinterleave algorithm.

Once you have verified that your algorithm is correct, proceed to the next step of the lab.

## 1.2 MCU Driver MATLAB Files

Just like the previous lab, we need an MCU driver to handle the control signals to and from the Chilipepper. Since the PCore has already been created in the previous lab, we can simply use the same PCore for this lab as well. Refer to Lab 1 for information on how to create this PCore if needed.

## 1.3 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 MCU PCore, thus the only core we need to create for this lab is the ADC Driver PCore. Using the same steps outlined in the previous labs, create a new HDL coder project called `adc_driver`. Add both your `adc_driver.m` file and your `adc_driver_tb.m` files to the **MATLAB Function** and **MATLAB Test Bench** categories respectively.





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

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**. For this Lab, the values of your “Proposed Type” column should resemble the settings below in Figure 1-3.

Variables		Function Replacements		Type Validation Output			
Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
<b>Input</b>							
rx_iq_sel	double	0	1			Yes	numerictype(0, 1, 0)
rx_d	double	0	1			Yes	<b>numerictype(1, 12, 0)</b>
<b>Output</b>							
blinky	double	0	0			Yes	numerictype(0, 1, 0)
rx_i	double	0	1			Yes	<b>numerictype(1, 12, 0)</b>
rx_q	double	0	0			Yes	<b>numerictype(1, 12, 0)</b>
<b>Persistent</b>							
blinky_cnt	double	0	514			Yes	<b>numerictype(0, 25, 0)</b>
i_out	double	0	1			Yes	<b>numerictype(1, 12, 0)</b>
q_out	double	0	0			Yes	<b>numerictype(1, 12, 0)</b>
q_out_delay	double	0	0			Yes	<b>numerictype(1, 12, 0)</b>

Figure 1-3: Variable types for ADC Driver MATLAB algorithm

3. Set the **Overflow Mode** to **saturate** as shown below. This will provide a more accurate sine and cosine waveform within our design.

<div>     </div>	
Setting	Value
When proposing types	use all collected data
Optimize whole numbers	Yes
Signedness	Automatic
Safety margin for sim min/max (%)	4
Generated fixed-point file postfix	_fixpt
Transform for-loop index variables	No
<b>fimath</b>	
Rounding method	Floor
Overflow action	<b>Saturate</b>
Product mode	FullPrecision
Sum mode	FullPrecision

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,

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

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 1-4 below

**Set the target device and synthesis tool**

Workflow: IP Core Generation

Platform: Generic Xilinx Platform [Launch board manager](#)

Synthesis tool: No synthesis tool available on system path [Refresh list](#)

Chip family:  Device:

Package:  Speed:

**IP core settings**

Name: adc\_driver\_pcore Version: v1.00.a

Processor/FPGA synchronization: Free running

[?](#)

1-4: Settings for Xilinx Zed Board HDL Coder Design

- Just below the synthesis tool settings, **rename your PCore** to `adc_driver_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.
- 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, all I/O can be configured as External Ports as shown below.

▲ Inport		
rx_d	numeric_type(1, 12, 0)	External Port
rx_iq_sel	numeric_type(0, 1, 0)	External Port
▲ Outport		
rx_i	numeric_type(1, 12, 0)	External Port
rx_q	numeric_type(1, 12, 0)	External Port
blinky	numeric_type(0, 1, 0)	External Port

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

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 labs. 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

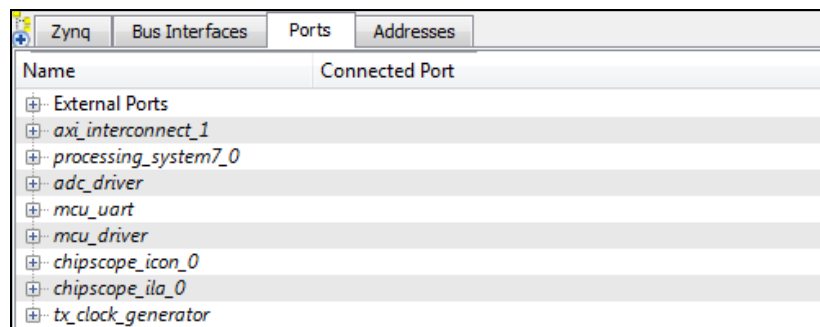
## Step 2

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
- Clock Generator
- 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.



Name	Connected Port
External Ports	
axi_interconnect_1	
processing_system7_0	
adc_driver	
mcu_uart	
mcu_driver	
chipscope_icon_0	
chipscope_ila_0	
tx_clock_generator	

Figure 2-1: EDK project ports list

## 2.2 Configuring the ADC Driver Port

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

1. First we will configure the `rx_iq_sel`, the `rx_d` and the `bliky` pins. Each of these pins can be assigned as **External ports**.
2. Next are the `rx_i` and the `rx_q` output pins. There is no further logic required to connect these pins to, therefore they can be left **unconnected**. Later we will connect these pins to ChipScope to be monitored in software.
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 Clock Generator IP Core

The Clock Generator is used in this project to distribute the appropriate clock signals to each of the PCores required for transmitting the tone, 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) and the `TX_CLK` and `RX_CLK` signals; which 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**
- CLKFBIN Required Frequency of **40Mhz** with **no Clock Deskew**
- CLKFBOUT Required Frequency of **40Mhz**, Required Group **PLLE0**, and **Buffered True**
- CLKOUT0 Required Frequency of **20MHz**, 0 Phase, **PLLE0** group and **Buffered True**
- CLKOUT1 Required Frequency of **40MHz**, 0Phase, **PLLE0** group and **Buffered True**
- CLKOUT2 Required Frequency of **40Mhz**, 0Phase, **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      →      External Ports
- CLKOUT0 →      mcu::sysgen
- CLKOUT1 →      External Ports
- CLKOUT2 →      External Ports
- CLKFBIN →      CLKFBOUT
- RST        →      net\_gnd
- LOCKED    →      External Port

The Last component which needs a clock is the ADC Driver PCore. Since the data for this core comes directly from the Chilipepper, a clock signal is also provided with the data to ensure proper latching of the RXD signal.

3. Configure the adc\_driver IPCORE\_CLK as an External port. Name the port as  
rx\_clk\_return\_pin.

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

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

adc_driver		
IPCORE_CLK	External Ports::rx_clk_return_pin	net_rx_clk_return_pin
IPCORE_RESETN	processing_system7_0::FCLK_RESETO_N	processing_system7_0::FCLK_RESETO_N
rx_d	External Ports::adc_driver_rx_d_pin	adc_driver_pcore_fixp
rx_iq_sel	External Ports::adc_driver_rx_iq_sel_pin	adc_driver_pcore_fixp
rx_i		No Connection
rx_q		No Connection
blinky	External Ports::adc_driver_blinky_pin	adc_driver_blinky
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1	Connected to BUS axi_interconnect_1
mcu_driver		
tx_clock_generator		
CLKIN	External Ports::tx_clock_generator_CLKIN_pin	tx_clock_generator_CLKIN_pin
CLKOUT0	mcu_driver::IPCORE_CLK	tx_clock_generator_CLKOUT0
CLKOUT1	External Ports::tx_clock_generator_tx_clk_pin	tx_clock_generator_CLKOUT1
CLKOUT2	External Ports::tx_clock_generator_rx_clk_pin	tx_clock_generator_CLKOUT2
CLKFBIN	tx_clock_generator::CLKFBOUT	tx_clock_generator_CLKFBIN
CLKFBOUT	tx_clock_generator::CLKFBIN	tx_clock_generator_CLKFBOUT
RST	net_gnd	net_gnd
LOCKED	External Ports::tx_clock_generator_LOCKED_pin	tx_clock_generator_LOCKED_pin

Figure 2-2: Clock Generator port configuration

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

## 2.6 Pin Assignments

Once the clock generator is configured correctly, the `IPCORE_CLK` clock 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_pll_pin          LOC = D18 | IOSTANDARD = LVCMOS25;
NET tx_clock_generator_pll_pin          TNM_NET = tx_clock_generator_CLKIN;
TIMESPEC TS_tx_clock_generator_pll = PERIOD tx_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;
NET rx_clk_return_pin                   LOC = L18 | IOSTANDARD = LVCMOS25;
#####Rx – FMC interface at 2.5V #####
NET adc_driver_axiw_0_rx_iq_sel_pin     LOC = N19 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[0]        LOC = M21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[1]        LOC = J21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[2]        LOC = M22 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[3]        LOC = J22 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[4]        LOC = T16 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[5]        LOC = P20 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[6]        LOC = T17 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[7]        LOC = N17 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[8]        LOC = J20 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[9]        LOC = P21 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[10]       LOC = N18 | IOSTANDARD = LVCMOS25;
NET adc_driver_axiw_0_rxd_pin[11]       LOC = J16 | IOSTANDARD = LVCMOS25;
##### MCU Interface #####
NET mcu_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_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 tx_clock_generator_LOCKED_pin        LOC = T22 | IOSTANDARD = LVCMOS33; # "LD0"
NET adc_driver_blinky_pin                 LOC = T21 | IOSTANDARD = LVCMOS33; # "LD1"
NET mcu_driver_blinky_pin                 LOC = U22 | IOSTANDARD = LVCMOS33; # "LD2"
```

Figure 2-3: EDK project pin assignments

## 2.7 Adding ChipScope Peripheral

The last step is to setup the ChipScope peripheral to verify the functionality of the ADC Driver.

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.

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

4. Add the rx\_i and rx\_q pins from the adc\_driver Port. Additionally, you should set the clock to the same clock used for the core, which for this design is Clockout\_0.
5. 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 ADC ports have the ChipScope peripherals in the correct locations.

+	axi_interconnect_1		
+	processing_system7_0		
-	adc_driver		
	IPCORE_CLK	External Ports::rx_clk_return_pin	net_rx_clk_return
	IPCORE_RESETN	processing_system7_0::FCLK_RESETO_N	processing_syste
	rxd	External Ports::adc_driver_rxd_pin	adc_driver_pcore
	rx_iq_sel	External Ports::adc_driver_rx_iq_sel_pin	adc_driver_pcore
	rx_i	chipscope_ila_0::TRIG0	adc_driver_rx_i_tc
	rx_q	chipscope_ila_0::TRIG0	adc_driver_rx_q_t
	blinky	External Ports::adc_driver_blinky_pin	adc_driver_blinky
	(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1	Connected to BU
+	mcu_driver		
+	chipscope_icon_0		
+	chipscope_ila_0		
-	tx_clock_generator		
	CLKIN	External Ports::tx_clock_generator_CLKIN_pin	tx_clock_generat
	CLKOUT0	mcu_driver::IPCORE_CLK	tx_clock_generat
	CLKOUT1	External Ports::tx_clock_generator_tx_clk_pin	tx_clock_generat
	CLKOUT2	External Ports::tx_clock_generator_rx_clk_pin	tx_clock_generat
	CLKFBIN	tx_clock_generator::CLKFBOUT	tx_clock_generat
	CLKFBOUT	tx_clock_generator::CLKFBIN	tx_clock_generat
	RST	net_gnd	net_gnd
	LOCKED	External Ports::tx_clock_generator_LOCKED_pin	tx_clock_generat

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

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.

## Create software project

## Step 3

---

Once the design is compiled and exported, you'll be greeted with a screen asking you where you would like to store your software project. It is very helpful to create the workspace folder in the same directory as your Sysgen and EDK folders. Doing this will keep all relevant files in the same location.

### 3.1 Creating a new C Project

This section will show you how to create a C program to test your receive tone project. Since our logic for receiving the signal is handled by PCores, all we need to do in software is initialize the hardware.

1. Select **File → New → Application Project**.
2. Name the project "receive\_tone" 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 tone project folder, as well as a **board support package (bsp)** folder.
5. If you navigate into the tone 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<sup>1</sup> 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 the `MCU_Driver` and the `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.

---

---

<sup>1</sup> Can be found at [https://github.com/Toyon/Chilipepper/tree/QPSK\\_pcore/ChilipepperSupport/Library%20Files](https://github.com/Toyon/Chilipepper/tree/QPSK_pcore/ChilipepperSupport/Library%20Files)

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

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.

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

int main()
{
    init_platform();

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

    // by default we are in transmit
    Chilipepper_SetPA( 0 );
    Chilipepper_SetTxRxSw( 1 ); // 0- transmit, 1-receive

    while (1)
    {}

    cleanup_platform();

    return 0;
}
```

Figure 3-1: Code outline for 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.

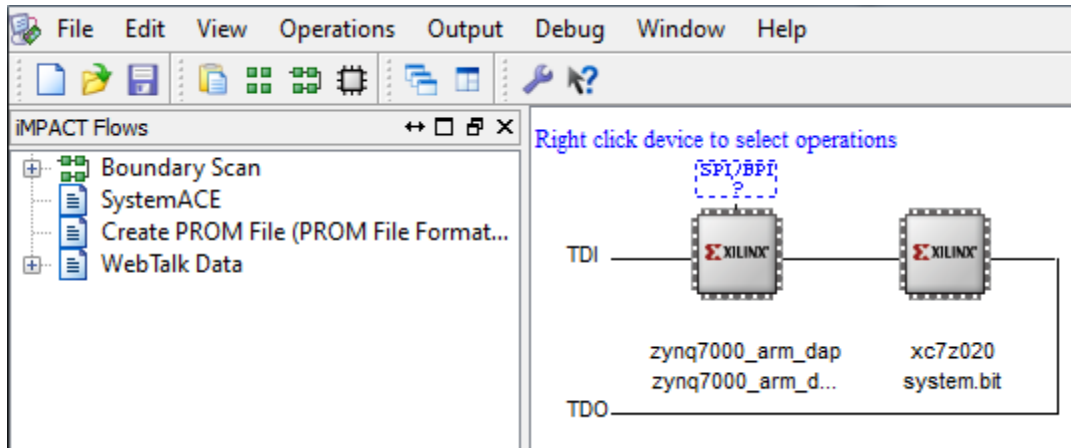
1. Connect the Chilipepper to the FPGA board and verify all cables are connected properly and the jumper settings are correct. Verify this by using the *Chilipepper Getting Started Guide*<sup>2</sup> as a reference. Also See Lab 0 for details on Jumper Configuration.
2. Once the FPGA and radio board 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.

<sup>2</sup> Can be found at [https://github.com/Toyon/Chilipepper/tree/master/QPSK\\_Radio/DemoFilesAndDocumentation](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.

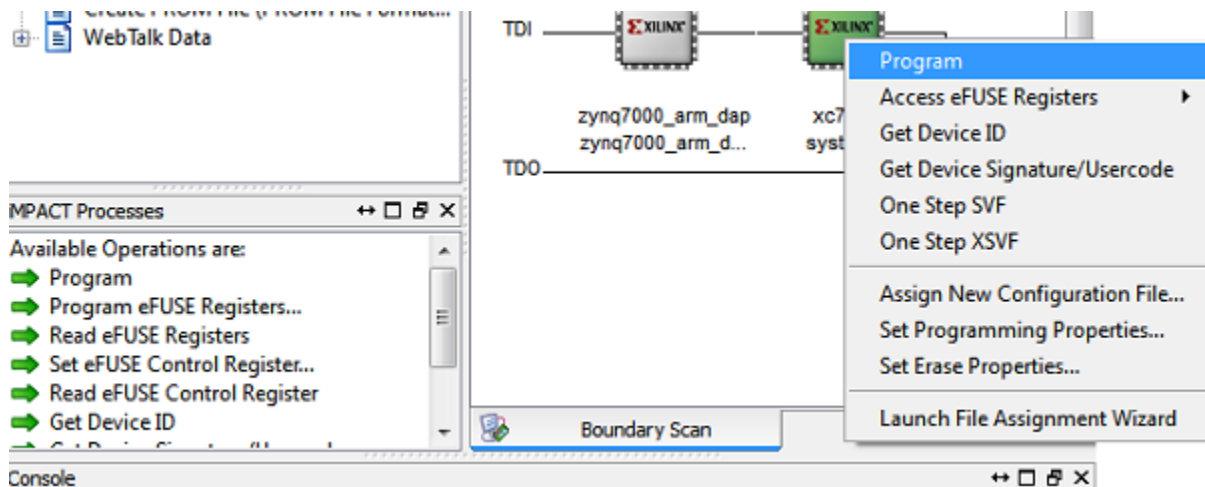


Figure 3-3: iMPACT configuration screen

### 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.

---

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

1. Test it by **right clicking** the `tone` 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 Tone, MCU and ADC PCores.

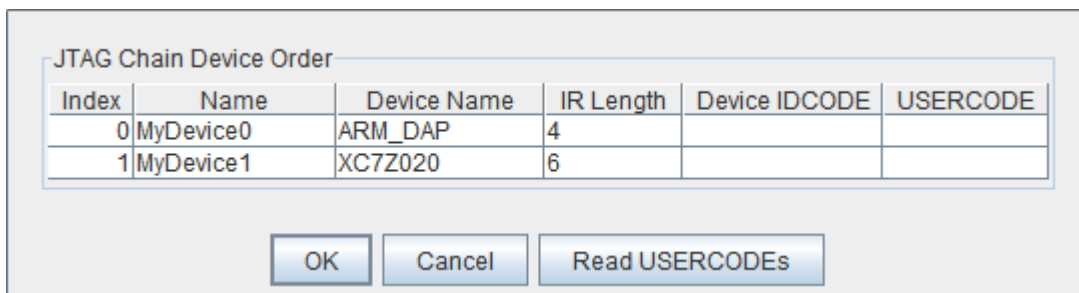
## Testing and Design Verification

## Step 4

### 4.1 Verification with ChipScope Pro

There are several methods available for verifying the MATLAB functions. For verification of the MCU Driver, ChipScope is recommended as it provides the most useful feedback of the Latch Handshaking.

1. To verify the MCU 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.



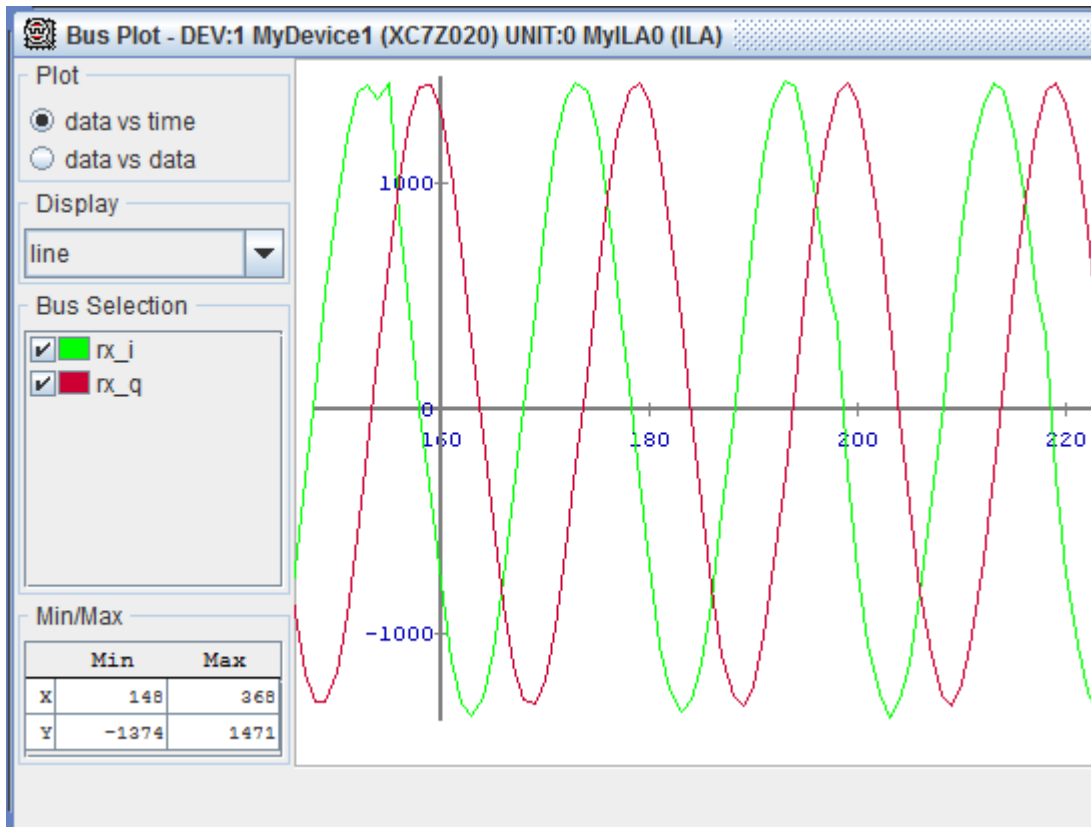
#### Note

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

- Click the **play button** in the top menu bar to display the signal. Additionally you can set up triggering options for periodic or continuous playback of the received signal. Your received signal should look similar to either Figure 4-1.



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

## 4.2 MATLAB Analysis

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

- It may be helpful later in your MATLAB code if you rename your **Data Port variables**. Right click on each of the trig/data Ports, and **change the names** to something more descriptive, such as `rx_i` and `rx_q` respectively.
- Open the file menu and select **Export**.
- Click the Ascii radio box and change the Signal to Export to the Bus Plot. Then click export.

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

4. Save the file in a convenient location. It is recommended you save this file in the same location as your MATLAB files. Call it something descriptive such as **ADC.prn**.

**Note**

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

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

```
fid = fopen('ADC.prn');
M = textscan(fid, '%d %d %d %d', 'Headerlines', 1);
fclose(fid);
rx_i = double(M{3})';
rx_q = double(M{4})';
plot(1:1024, rx_i, 1:1024, rx_q)
```

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

```
%Load scope data
fid = fopen('ADC.prn');
M = textscan(fid, '%d %d %d %d', 'Headerlines', 1);
fclose(fid);
rx_i = double(M{3})';
rx_q = double(M{4})';

%Process signal data
Fs=20E6;
L=length(rx_i)-1;
NFFT=2*(2^nextpow2(L));
Y_i=fft(rx_i,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);

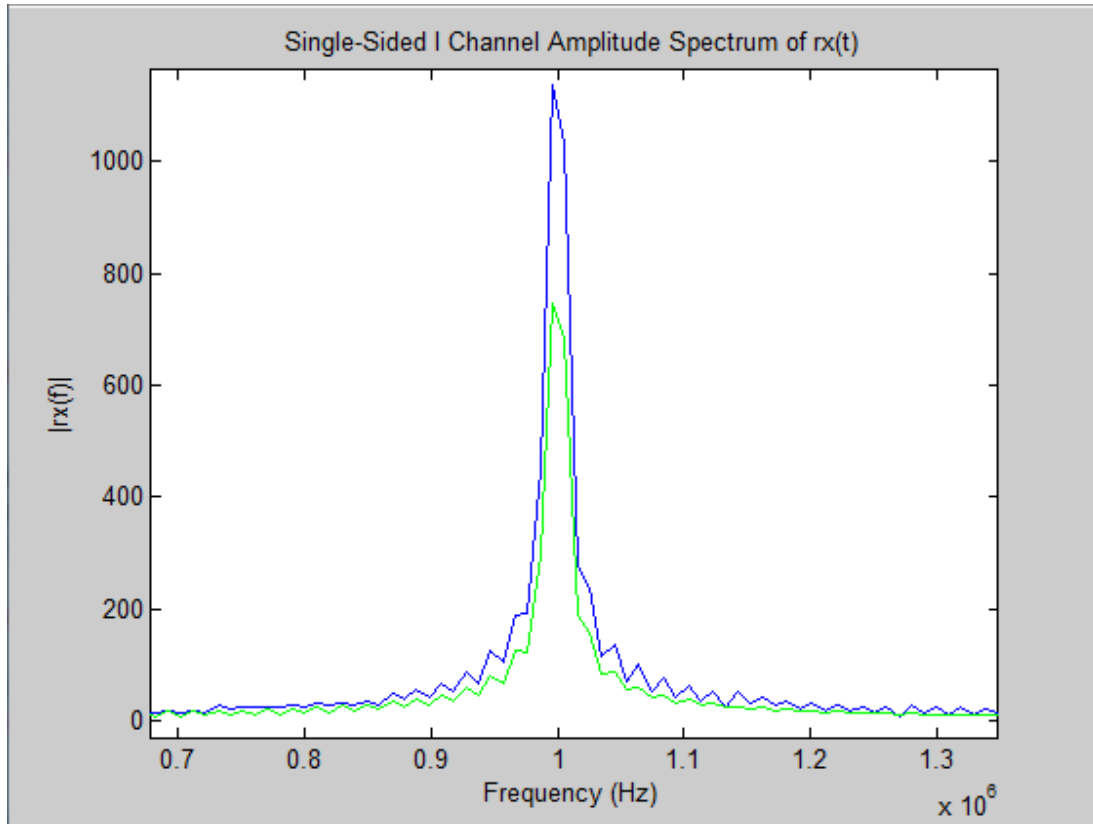
%Create theoretical signal
t=(0:L-1)/Fs;
X=88*sin(2*pi*1e6*t);
X_i=fft(X,NFFT)/L;

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

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

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

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



4-3: FFT Plot of the I channel