|  |
| --- |
| Toyon Research Corporation |
| **Lab 1: Output Tone** |
| Chilipepper Tutorial Projects |

|  |
| --- |
| Version 1.1  12/11/2013 |

Table of Contents

[Introduction 3](#_Toc374546973)

[Procedure 3](#_Toc374546974)

[Objectives 3](#_Toc374546975)

[Generate HDL code 4](#_Toc374546976)

[1.1 Output Tone MATLAB Files 4](#_Toc374546977)

[1.2 DAC Driver MATLAB Files 7](#_Toc374546978)

[1.3 MCU Driver MATLAB Files 8](#_Toc374546979)

[1.4 HDL Coder Project 10](#_Toc374546980)

[Configure Cores and Export Design 16](#_Toc374546981)

[2.1 Needed IP Cores 16](#_Toc374546983)

[2.2 Configuring the DAC Driver Port 17](#_Toc374546984)

[2.3 Configuring the Tone Generator Port 17](#_Toc374546985)

[2.4 Configuring the MCU Port 17](#_Toc374546986)

[2.5 Configuring the TX Clock Generator IP Core 18](#_Toc374546987)

[2.6 Pin Assignments 19](#_Toc374546988)

[2.7 Adding ChipScope Peripheral 20](#_Toc374546989)

[Create software project 22](#_Toc374546990)

[3.1 Creating a new C Project 22](#_Toc374546992)

[3.2 Programming the Board 23](#_Toc374546993)

[3.3 Debugging with SDK 25](#_Toc374546994)

[Testing and Design Verification 26](#_Toc374546995)

[4.1 Verification with ChipScope Pro 26](#_Toc374546996)

Lab 1: Output Tone

# Introduction

This lab will show you how to output a single frequency tone on an FPGA Mezzanine Card (FMC) radio board using the Xilinx Zed Board FPGA and the Toyon Chilipepper FMC. The components used to generate the tone will be created in hardware on the FPGA by exporting your MATLAB code directly to Xilinx as an FPGA PCore. 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 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 a MATLAB algorithm to test your design
* Generate HDL code from your MATLAB algorithm
* Configure your created PCores and export the design into SDK
* Create software to run your design

## Objectives

After completing this lab, you will be able to:

* Translate an algorithm from MATLAB code into an FPGA PCore design
* Configure The I/O of your design using EDK
* Create a software application to test your created FPGA hardware
* Run your design in ChipScope Pro for Design Verification

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 Output Tone MATLAB Files

Your MATLAB functions will eventually become Xilinx PCores that will be synthesized into hardware. The algorithms within these functions describe the operations in each clock cycle, and processes data on a sample-by-sample basis. The primary function used to create the tone in this lab is shown in Figure 1-1.



Figure 1‑1: MATLAB algorithm for 1 MHz tone generation

In the above algorithm, phi is an offset used to grab the correct value from the sine and cosine look up table. The i\_out and q\_out outputs are the signals that are sent directly to the DAC. Since the DAC on the Chilipepper is 12 bits, these outputs are multiplied by a scaling factor to allow for the full 12 bit resolution (12 bit 2’s Compliment). Additionally, an input variable is required for all designs to be converted properly into HDL. Therefore, the myInput parameter is required, even though it’s not used directly in the algorithm.

1. Create a directory for the project under C:\QPSK\_Projects\Lab\_1.
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 output\_tone.m inside the MATLAB directory.

There are two lines in Figure 1-1 which refer to SIN and COS variables. Since the tone generation will be implemented in hardware, we will pre-define our sine and cosine waveforms, and save them into MATLAB files called SIN.m and COS.m. The values of each waveform will be used as a **lookup table** to correctly output a sample of the sine and cosine. The selection of each sample value will depend directly on the value phi, which is used as an offset within the table. To create this table an **additional MATLAB function** must be created and run prior to creating the core in the workflow advisor. The code used for this function is shown in Figure 1-2.



Figure 1‑2: MATLAB code for creating sine and cosine lookup tables

1. Create a new **MATLAB function** with the contents of Figure 1-2.
2. **Save** this function as make\_trig\_lut inside the MATLAB folder.

This function will create **two** more MATLAB files called SIN.m and COS.m. These files are required to execute the test bench script. Therefore it may be required to run the test bench twice for proper execution.

**Note**

Both the SIN.m and COS.m files are created by sampling a pure sine and cosine wave at some pre-calculated interval. The value of this interval should be proportional to the frequency of the wave we wish to generate, the clock frequency of our tone generator core, and the offset (phi) we wish to use in our lookup table (LUT). A small offset value can be used to conserve memory space, however for this lab we want to make the LUT larger than the minimum size to reduce aliasing of our **1MHz tone**; therefore we will use an **offset of 20**. This means our sine and cosine waves should be sampled at a rate of .

The last MATLAB file required for this core is the test bench script. This script is required for HDL generation, but also allows you to test the functionality of the MATLAB algorithm. The code used for this script is shown in Figure 1-3.



Figure 1‑: MATLAB code for Output\_Tone test bench script

1. Create a new **MATLAB script** with the contents of Figure 1-3.
2. **Save** this script as output\_tone\_tb.m inside the MATLAB directory.
3. **Run** this script in MATLAB to test the algorithm and the lookup table.

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

## 1.2 DAC Driver MATLAB Files

The purpose of the DAC driver within this lab is to properly format the in-phase and quadrature channel signals provided by the output of the tone PCore. The Chilipepper FMC expects a single interleaved signal containing the sample data, as well as a single toggle line to tell the board whether the current input is I channel or Q channel. Given that both the I and Q channels are clocked at the same rate, the DAC driver PCore must be clocked at twice the frequency of the output tone PCore to interleave the data properly. The MATLAB function used to create the DAC Driver PCore is shown in figure 1-4 below.



Figure 1‑: MATLAB Driver to interleave I and Q channels

1. Create a new **MATLAB function** with the contents of Figure 1-4.
2. **Save** this function as dac\_driver.m inside the MATLAB directory.

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 1-5 below shows a basic test bench for the dac driver function.



Figure 1‑: MATLAB code for dac\_driver test bench script

1. Create a new **MATLAB script** with the contents of Figure 1-5.
2. **Save** this script as dac\_driver\_tb.m inside the MATLAB directory.
3. **Run** this script in MATLAB to test the interleave algorithm.

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

## 1.3 MCU Driver MATLAB Files

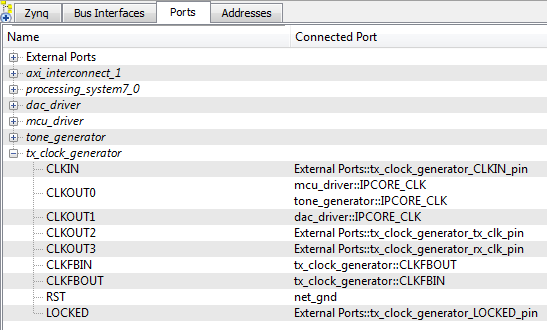
Using the same approach as Section 1.2, we must create an MCU Driver MATLAB function. The purpose of this function is to allow for reading from and writing to various registers within the Chilipepper microcontroller. Due to differences in clock frequencies within the design, we will also establish a handshaking protocol for this MATLAB function which will ensure data is properly latched to/from the microcontroller at the correct time. The MATLAB function for this driver is shown in Figure 1-6 below.



Figure 1‑6: MATLAB Driver for Chilipepper MCU Integration

1. Create a new **MATLAB function** with the contents of Figure 1-6.
2. **Save** this function as mcu\_driver.m inside the MATLAB directory.

The test bench script for the mcu driver is shown in Figure 1-7 below. This script does not exhaustively test the Handshaking used within the core, however as mentioned previously it is required for translation into HDL.



1. Create a new **MATLAB script** with the contents of Figure 1-7.
2. **Save** this script as mcu\_driver\_tb.m inside the MATLAB directory.



Figure 1‑7: Testbench Script for mcu driver PCore

## 1.4 HDL Coder Project

Now that all of the MATLAB files have been created, we can start to turn them into PCores. Using the same steps outlined in Lab 0, create a new HDL coder project called Tone. Add both your output\_tone.m file and your output\_tone\_tb.m file to the **MATLAB Function** and **MATLAB Test Bench** categories respectively. The two lookup table files will be automatically added to the project later; just be sure they are located in the same directory as your output tone files.

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 lab.
2. Right-click **Fixed-Point Conversion,** and select **Run to Selected Task.** For this Lab, the values of your “Propose Type” column should resemble the settings below in Figure 1-8.

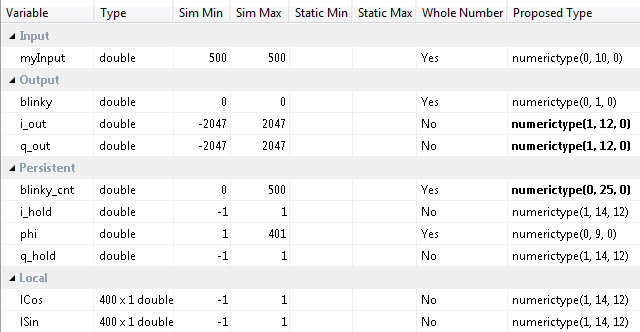
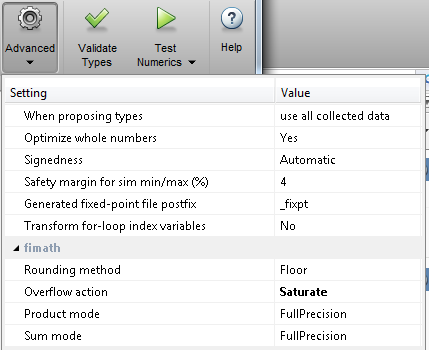


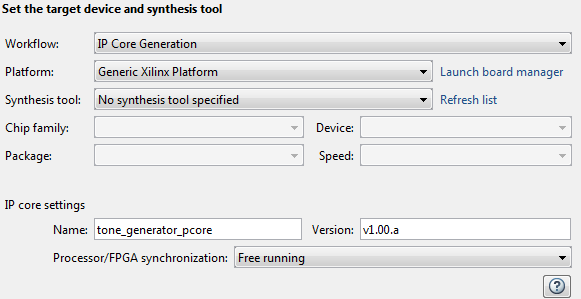
Figure 1‑8: Variable types for Tone MATLAB algorithm

As shown in Figure 1-8, the i\_out and q\_out variables are set to (1,12,0). This formats these variables as **12 bit signed values** (the first value is for signed or unsigned) with 0 bits used for a decimal value. You can change this format if you would like your precision to include decimal values, however remember to change the multiplier used in the MATLAB algorithm accordingly to ensure you get full **12 bit resolution**.

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

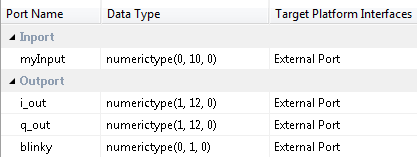


1. Once you have corrected the **Type** setting for all your variables, click **Select** **Code Generation Target**. Here you can select the FPGA you will use for your design. For this Lab, we will not be using any of the built-in Zynq board functionality within our MATLAB PCores. Therefore you can leave the default settings. Ensure your Workflow settings resemble figure 1-9 below



1‑9: Settings for Xilinx Zed Board HDL Coder Design

1. Just below the synthesis tool settings, **rename your PCore** to tone\_generator\_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.
2. 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.



1. 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>.
2. Repeat this process for both the DAC Driver MATLAB function as well as the MCU Driver function. **Name the PCores** as dac\_driver\_pcore and mcu\_driver\_pcore respectively. Verify your **Fixed-Point variable** conversions and your **Target interface port** settings using the Figures below. Also don’t forget to set both projects to use the **DUT base** clock rate.

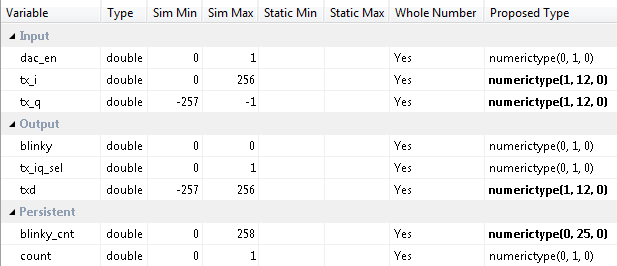


Figure 1‑10: Fixed-Point Variables for DAC Driver PCore

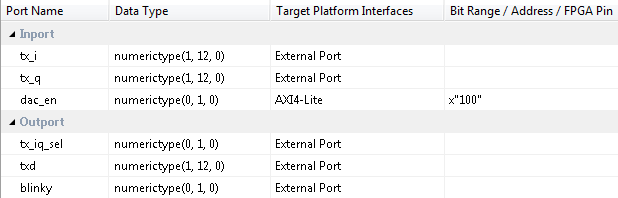


Figure 1‑11: Port settings for DAC Driver PCore

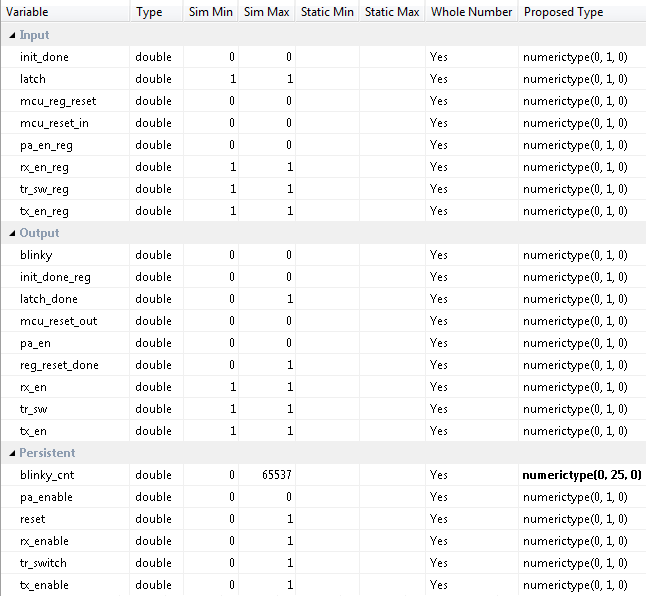


Figure 1‑12: Fixed-Point Variables for MCU Driver PCore

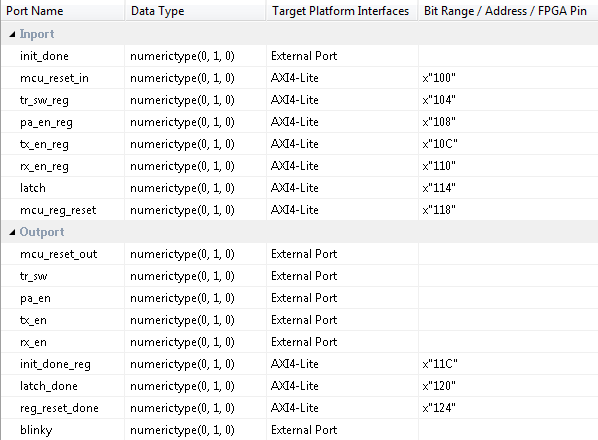


Figure 1‑13: Port settings for MCU Driver PCore

1. Once all of the PCores have 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.
2. Once the project is created, **copy each of the PCore folders** from the MATLAB directory into the PCores folder of your **EDK Project**. Then simply select project -> **rescan user repositories** to show your newly added User PCores within your EDK project.

The PCore folder created by HDL Coder for the MCU and DAC Driver Projects can be reused for future labs. It is recommended that you save a default copy of each of these cores to speed up your design process in the remaining labs.

**Note**

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

* DAC Driver
* MCU Driver
* Tone Generator
* Clock Generator
* Processing System
* AXI Interconnect

1. Create a new folder called EDK within the Project directory to store your new EDK project.
2. Import the configuration template for the ZED Board as done in Lab 0 section 2.1.9.
3. Add each of the PCores listed above to your design using Figure 2-1 below as a guide to name your cores appropriately. The remainder of this step will assist you in configuring each of these cores.

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 to verify the cores have all been added to the design.

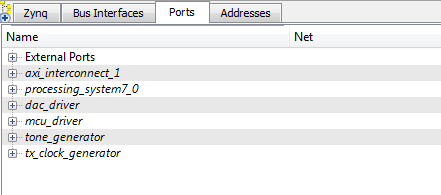


Figure 2‑1: EDK project ports list

## 2.2 Configuring the DAC Driver Port

**Expand** the **DAC Driver** core. There are 7 individual I/O pins which need to be routed in this core.

1. First we will configure the tx\_i and tx\_q pins. These are input pins which are used to create the **TXD output** by interleaving an I and Q channel. The signals come directly from the lookup table created by the MATLAB tone generator algorithm. **Assign** these two pins to the i\_out and q\_out output pins from the **Tone Generator PCore** respectively.
2. Next are the txd, tx\_iq\_sel, and blinky output pins. These pins carry signals which should be routed directly to physical components on the FPGA as **external pins**. The first two are sent to the **FMC** connector port and into the Chilipepper radio board, while the blinky pin connects to an LED on the FPGA. **Assign** all of these pins as **external ports.**
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 Tone Generator Port

**Expand** the **Tone Generator** core. There are 6 individual I/O pins which need to be routed in this core.

1. If the DAC Driver core was configured correctly in section 2.2, the Tone Generator core should already have its i\_out and q\_out pins assigned.
2. The blinky pin can be **assigned** as an **external port**, and the myInput port can simply be left with No Connection.
3. Again, connect the IPCORE\_RESETN port to the processing\_system7 FCLK\_RESET0\_N port.
4. Also, skip the IPCORE\_CLK for now as it is assigned later in **section 2.5**

## 2.4 Configuring the MCU 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.

## 2.5 Configuring the TX 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 5 other devices; 3 PCores (MCU, DAC and Tone Generator) 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 ADC is used within the design, the clock is required for proper initialization of the Chilipepper FMC.

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**
* CLKOUT3 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.

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

* CLKIN External Ports
* CLKOUT0 tone\_generator:: IPCORE\_CLK **and** mcu\_driver:: IPCORE\_CLK
* CLKOUT1 dac\_driver::IPCORE\_CLK
* CLKOUT2 External Ports
* CLKOUT3 External Ports
* CLKFBIN CLKFBOUT
* RST net\_gnd
* LOCKED External Ports

Although there are 5 sources which need to be clocked, **two** of the PCores use the same clock frequency, and can thus **share** the same clock signal from the Clock Generator. The Tone Generator and MCU Driver clocks are only half the clock speeds of the DAC and the CLKIN pin. This is necessary for the creation of the TXD signal as both the I and Q channels must be **interleaved** into the signal at a rate **equal to** that of the TXD\_CLK rate. Your Clock Generator port should look similar to Figure 2-2.

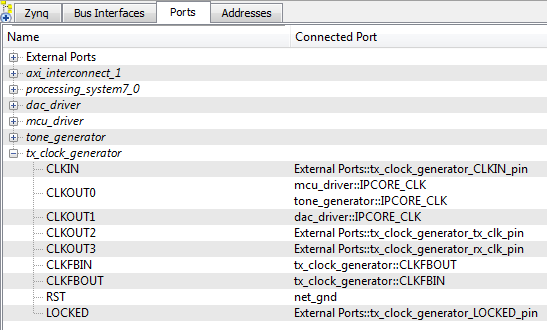


Figure 2‑2: Clock Generator port configuration

## 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.

You can rename the external port pins to more appropriate names when you setup the UCF file. Just be sure the names in the file match what is given in the External Ports section of the Ports tab.

**Note**

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**.



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 MCU 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.
4. Add all available signals from the mcu\_driver Port. This should be all **5 output external ports** in the Core. 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 MCU ports have the ChipScope peripherals in the correct locations.

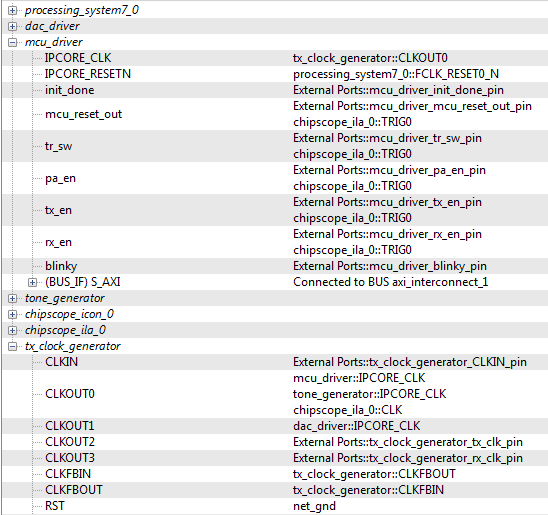


Figure 2‑4: Ports list after adding ChipScope peripheral to monitor MCU signals

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 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 tone generation project. Since our algorithms for the tone generation are written in MATLAB, and the mixing is handled by PCores, all we need to do in software is initialize the hardware.

It would be helpful if you have completed the Embedded System Design tutorial in the *ZedBoard AP SoC Concepts Tools and Techniques Guide[[1]](#footnote-1)*. In addition, you should be familiar with the *Chilipepper user guide[[2]](#footnote-2)* to ensure proper MCU Control.

**Note**

1. Select **File 🡪 New** 🡪 **Application Project**.
2. Name the project “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[[3]](#footnote-3) 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 DAC\_Driver.

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.

**Note**

1. (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.





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[[4]](#footnote-4)* as a reference. Also See Lab 0 for details on Jumper Configuration.
2. Once the FPGA and radio board are connected correctly, turn on the board.
3. Open iMPACT in the ISE Design tools.
4. Select no if Impact asks you to load the last saved project.
5. Select yes to allow iMPACT to automatically create a new project for you. If you receive any connection errors, verify your USB or JTAG programmer cables are connected properly.
6. Select the Automatic option for the JTAG boundary scan setting and click ok.
7. Hit yes to assign configuration files. Bypass the first file selection, but for the second selection, browse to the location of your system.bit file. It should be inside the “Implementation” folder of your EDK project folder.
8. Select ok on the next screen verifying that the board displayed is your Zynq xc7z020 board. It should look similar to Figure 3-2 below.



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

1. 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 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 DAC PCores. Verify that the output of the antennae is in fact a 2.4 GHz + 1 MHz sine wave using a spectrum analyzer.

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.



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.

**Note**

1. (**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.
2. Select ok to get to the Analyzer main screen. Open the **file menu** and select **Import**.
3. 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.
4. On the Waveform screen, you can view all of the 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.
5. 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 debug signals should look similar to either Figure 4-1 or Figure 4-2 below.

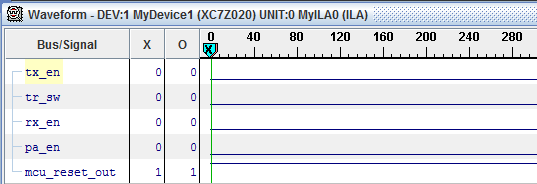


Figure 4‑1: MCU Driver signals in ChipScope Pro (before initialization)

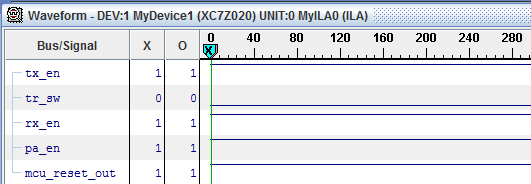


Figure 4‑2: MCU Driver signals in ChipScope Pro (after initialization)

In addition you can have ChipScope monitor the DAC driver to get a better understanding of how the interleaver is working. Figure 4-3 below shows a ChipScope capture of the DAC Driver for this lab.

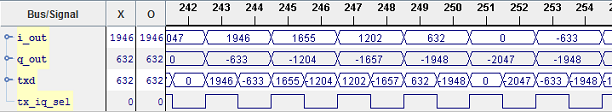


Figure 4‑3: DAC Driver signals in ChipScope Pro

1. Can be found at [http://www.wiki.xilinx.com/Getting Started](http://www.wiki.xilinx.com/Getting%20Started) [↑](#footnote-ref-1)
2. Can be found at <https://github.com/Toyon/Chilipepper/tree/QPSK_pcore/ChilipepperSupport> [↑](#footnote-ref-2)
3. Can be found at <https://github.com/Toyon/Chilipepper/tree/QPSK_pcore/ChilipepperSupport/Library%20Files> [↑](#footnote-ref-3)
4. Can be found at <https://github.com/Toyon/Chilipepper/tree/master/QPSK_Radio/DemoFilesAndDocumentation> [↑](#footnote-ref-4)