Toyon Research Corporation

Lab 1: Output Tone

Chilipepper Tutorial Projects

Version 1.2
1/9/2014

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Table of Contents

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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

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   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.

```matlab
%#codegen
function[i_out, q_out, blinky] = output_tone(myInput)
%Output a tone at 1MHz
    persistent i_hold q_hold phi blinky_cnt
    if (isempty(phi))
        phi=1;
        blinky_cnt = 0;
    end

    %ROMs *not* declared persistent
    lSin = SIN;
    lCos = COS;

    i_hold = lSin(phi);
    q_hold = lCos(phi);

    phi = phi + 20;
    if phi > 400
        phi = phi-400;
    end

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

    i_out = (2^11-1)*i_hold;
    q_out = (2^11-1)*q_hold;
end
```

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

```matlab
function make_trig_lut
    % Generate LUT values
    ii = 0:(400-1);
    c = cos(2*pi*ii/400);
    s = sin(2*pi*ii/400);
    % Create cosine LUT
    fid = fopen('COS.m','w+');
    fprintf(fid,'function y = COS\n');
    fprintf(fid,'%%#codegen\n');
    fprintf(fid,'y = [\n');
    fprintf(fid,'%14.12f\n',c);
    fprintf(fid,'];\n');
    fclose(fid);
    % Create sine LUT
    fid = fopen('SIN.m','w+');
    fprintf(fid,'function y = SIN\n');
    fprintf(fid,'%%#codegen\n');
    fprintf(fid,'y = [\n');
    fprintf(fid,'%14.12f\n',s);
    fprintf(fid,'];\n');
    fclose(fid);
end
```

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

5. Create a new **MATLAB function** with the contents of Figure 1-2.

6. **Save** this function as `make_trig_lut` inside the MATLAB folder.

Note	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.

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 $\frac{Ftone}{phi*Tone\ Generator\ Clock\ Frequency} = \frac{1MHz}{20*20MHz} = \frac{1}{400}$.

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.

```matlab
Fs = 20e6;          % Sampling frequency (tone – 20MHz)
L = 500;            % Length of signal
t = (0:L-1)/Fs;     % Time vector

make_trig_lut;

for i1 = 1:L
    [I(i1), Q(i1)] = output_tone(L);
    y(i1) = I(i1) – Q(i1);
end

subplot(2,1,1);
plot(t,y);
title('y(t)')
xlabel('time (milliseconds)')
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);
% Plot single-sided amplitude spectrum.
subplot(2,1,2);
plot(f,2*abs(Y(1:NFFT/2+1)))
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')
```

Figure 1-3: MATLAB code for Output_Tone test bench script

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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

8. **Save** this script as `output_tone_tb.m` inside the MATLAB directory.

9. **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.

```matlab
%#codegen
function [ tx_iq_sel, txd, blinky] = dac_driver( tx_i, tx_q, dac_en )
%interleave I and Q channels into txd
    persistent count blinky_cnt
    if isempty(count)
        count = 0;
        blinky_cnt = 0;
    end
    %DAC runs at 2x the input rate
    if dac_en == 1
        tx_iq_sel = count;
        if tx_iq_sel == 0
            txd = tx_i;
        else
            txd = tx_q;
        end
    else
        tx_iq_sel = 0;
        txd = 0;
    end
    count = +~count;

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

Figure 1-4: MATLAB Driver to interleave I and Q channels

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

```matlab
dac_en = 0;

for i1 = 0:2:256
    tx_i = i1;
    tx_q = -(i1+1);
    [tx_iq_sel(i1+1), txd(i1+1), blinky] = dac_driver(tx_i, tx_q, dac_en);
    [tx_iq_sel(i1+2), txd(i1+2), blinky] = dac_driver(tx_i, tx_q, dac_en);
    if i1 > 10
        dac_en = 1;
    end
end
%plot the result
subplot(2,1,1)
plot(txd)
title('Interleaved I and Q signal');
subplot(2,1,2)
plot(tx_iq_sel)
title('I/Q select line');
```

Figure 1-5: MATLAB code for dac_driver test bench script

3. Create a new **MATLAB script** with the contents of Figure 1-5.

4. **Save** this script as `dac_driver_tb.m` inside the MATLAB directory.

5. **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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

```matlab
function [mcu_reset_out, tr_sw, pa_en, tx_en, rx_en,...
    init_done_reg, latch_done, reg_reset_done, blinky] = ...
    mcu_driver(init_done, mcu_reset_in, tr_sw_reg,...
    pa_en_reg, tx_en_reg, rx_en_reg, latch, mcu_reg_reset)

    persistent reset pa_enable tx_enable rx_enable tr_switch blinky_cnt

    reg_reset_done = 0;
    if (isempty(reset) || mcu_reg_reset == 1)
        reset = 1;
        tr_switch = 0;
        pa_enable = 0;
        tx_enable = 0;
        rx_enable = 0;
        reg_reset_done = 1;
        blinky_cnt = 0;
    end

    latch_done = 0;
    if (latch == 1)
        reset = mcu_reset_in;
        tr_switch = tr_sw_reg;
        pa_enable = pa_en_reg;
        tx_enable = tx_en_reg;
        rx_enable = rx_en_reg;
        latch_done = 1;
    end

    init_done_reg = init_done;
    mcu_reset_out = reset;
    tr_sw = tr_switch;
    pa_en = pa_enable;
    tx_en = tx_enable;
    rx_en = rx_enable;

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

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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

3. Create a new **MATLAB script** with the contents of Figure 1-7.

4. **Save** this script as `mcu_driver_tb.m` inside the MATLAB directory.

```
mcu_reset_reg = 0;
tr_sw_reg = 1;
pa_en_reg = 0;
tx_en_reg = 1;
rx_en_reg = 1;
init_done = 0;
for i1 = 0:2^16
    [mcu_reset,tr_sw, pa_en, tx_en, rx_en,...
    init_done_reg, latch_done, blinky] =...
    mcu_driver(init_done, mcu_reset_reg,...
    tr_sw_reg, pa_en_reg, tx_en_reg, rx_en_reg, 1, 0);
end
```

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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
◢ Input							
myInput	double	500	500			Yes	numerictype(0, 10, 0)
◢ Output							
blinky	double	0	0			Yes	numerictype(0, 1, 0)
i_out	double	-2047	2047			No	**numerictype(1, 12, 0)**
q_out	double	-2047	2047			No	**numerictype(1, 12, 0)**
◢ Persistent							
blinky_cnt	double	0	500			Yes	**numerictype(0, 25, 0)**
i_hold	double	-1	1			No	numerictype(1, 14, 12)
phi	double	1	401			Yes	numerictype(0, 9, 0)
q_hold	double	-1	1			No	numerictype(1, 14, 12)
◢ Local							
ICos	400 × 1 double	-1	1			No	numerictype(1, 14, 12)
ISin	400 × 1 double	-1	1			No	numerictype(1, 14, 12)

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**.

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

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 1-9 below

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

5. 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.

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, all I/O can be configured as External Ports as shown below.

Port Name	Data Type	Target Platform Interfaces
▲ Inport		
myInput	numerictype(0, 10, 0)	External Port
▲ Outport		
i_out	numerictype(1, 12, 0)	External Port
q_out	numerictype(1, 12, 0)	External Port
blinky	numerictype(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. 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.

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Number	Proposed Type
◢ Input							
dac_en	double	0	1			Yes	numerictype(0, 1, 0)
tx_i	double	0	256			Yes	**numerictype(1, 12, 0)**
tx_q	double	-257	-1			Yes	**numerictype(1, 12, 0)**
◢ Output							
blinky	double	0	0			Yes	numerictype(0, 1, 0)
tx_iq_sel	double	0	1			Yes	numerictype(0, 1, 0)
txd	double	-257	256			Yes	**numerictype(1, 12, 0)**
◢ Persistent							
blinky_cnt	double	0	258			Yes	**numerictype(0, 25, 0)**
count	double	0	1			Yes	numerictype(0, 1, 0)

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

Port Name	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
◢ Inport			
tx_i	numerictype(1, 12, 0)	External Port	
tx_q	numerictype(1, 12, 0)	External Port	
dac_en	numerictype(0, 1, 0)	AXI4-Lite	x"100"
◢ Outport			
tx_iq_sel	numerictype(0, 1, 0)	External Port	
txd	numerictype(1, 12, 0)	External Port	
blinky	numerictype(0, 1, 0)	External Port	

Figure 1-11: Port settings for DAC Driver PCore

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							
init_done	double	0	0			Yes	numerictype(0, 1, 0)
latch	double	1	1			Yes	numerictype(0, 1, 0)
mcu_reg_reset	double	0	0			Yes	numerictype(0, 1, 0)
mcu_reset_in	double	0	0			Yes	numerictype(0, 1, 0)
pa_en_reg	double	0	0			Yes	numerictype(0, 1, 0)
rx_en_reg	double	1	1			Yes	numerictype(0, 1, 0)
tr_sw_reg	double	1	1			Yes	numerictype(0, 1, 0)
tx_en_reg	double	1	1			Yes	numerictype(0, 1, 0)
⊿ Output							
blinky	double	0	0			Yes	numerictype(0, 1, 0)
init_done_reg	double	0	0			Yes	numerictype(0, 1, 0)
latch_done	double	0	1			Yes	numerictype(0, 1, 0)
mcu_reset_out	double	0	0			Yes	numerictype(0, 1, 0)
pa_en	double	0	0			Yes	numerictype(0, 1, 0)
reg_reset_done	double	0	1			Yes	numerictype(0, 1, 0)
rx_en	double	1	1			Yes	numerictype(0, 1, 0)
tr_sw	double	1	1			Yes	numerictype(0, 1, 0)
tx_en	double	1	1			Yes	numerictype(0, 1, 0)
⊿ Persistent							
blinky_cnt	double	0	65537			Yes	**numerictype(0, 25, 0)**
pa_enable	double	0	0			Yes	numerictype(0, 1, 0)
reset	double	0	1			Yes	numerictype(0, 1, 0)
rx_enable	double	0	1			Yes	numerictype(0, 1, 0)
tr_switch	double	0	1			Yes	numerictype(0, 1, 0)
tx_enable	double	0	1			Yes	numerictype(0, 1, 0)

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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

Port Name	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
⊿ Inport			
init_done	numerictype(0, 1, 0)	External Port	
mcu_reset_in	numerictype(0, 1, 0)	AXI4-Lite	x"100"
tr_sw_reg	numerictype(0, 1, 0)	AXI4-Lite	x"104"
pa_en_reg	numerictype(0, 1, 0)	AXI4-Lite	x"108"
tx_en_reg	numerictype(0, 1, 0)	AXI4-Lite	x"10C"
rx_en_reg	numerictype(0, 1, 0)	AXI4-Lite	x"110"
latch	numerictype(0, 1, 0)	AXI4-Lite	x"114"
mcu_reg_reset	numerictype(0, 1, 0)	AXI4-Lite	x"118"
⊿ Outport			
mcu_reset_out	numerictype(0, 1, 0)	External Port	
tr_sw	numerictype(0, 1, 0)	External Port	
pa_en	numerictype(0, 1, 0)	External Port	
tx_en	numerictype(0, 1, 0)	External Port	
rx_en	numerictype(0, 1, 0)	External Port	
init_done_reg	numerictype(0, 1, 0)	AXI4-Lite	x"11C"
latch_done	numerictype(0, 1, 0)	AXI4-Lite	x"120"
reg_reset_done	numerictype(0, 1, 0)	AXI4-Lite	x"124"
blinky	numerictype(0, 1, 0)	External Port	

Figure 1-13: Port settings for MCU Driver PCore

9. 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.

10. 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.

Note	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.

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

- MCU UART

- 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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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 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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

2.5   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.

2.6   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.

2. **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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

- CLKOUT3 $\longrightarrow$ External Ports

- CLKFBIN $\longrightarrow$ CLKFBOUT

- RST $\longrightarrow$ net_gnd

- LOCKED $\longrightarrow$ 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

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

Note	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.

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_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;
#########################################Tx – FMC interface at 2.5V ##########################################
NET dac_driver_tx_iq_sel_pin                    LOC = B16        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[0]                       LOC = A18        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[1]                       LOC = A19        | IOSTANDARD = LVCMOS25 | DRIVE0 = 4 | SLEW = FAST;
NET dac_driver_txd_pin[2]                       LOC = E20        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[3]                       LOC = G21        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[4]                       LOC = F19        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[5]                       LOC = G15        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[6]                       LOC = E19        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[7]                       LOC = G16        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[8]                       LOC = G19        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[9]                       LOC = A16        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_xd_pin[10]                       LOC = A17        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
NET dac_driver_txd_pin[11]                      LOC = C18        | IOSTANDARD = LVCMOS25 | DRIVE = 4 | SLEW = FAST;
############################################# 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 tone_generator_blinky_pin                   LOC = T21        | IOSTANDARD=LVCMOS33;  # "LD1"
NET mcu_driver_blinky_pin                       LOC = U22        | IOSTANDARD=LVCMOS33;  # "LD2"
NET dac_driver_blinky_pin                       LOC = U21        | IOSTANDARD=LVCMOS33;  # "LD3"
```

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.

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

⊞ processing_system7_0	
⊞ dac_driver	
⊟ mcu_driver	
IPCORE_CLK	tx_clock_generator::CLKOUT0
IPCORE_RESETN	processing_system7_0::FCLK_RESET0_N
init_done	External Ports::mcu_driver_init_done_pin
mcu_reset_out	External Ports::mcu_driver_mcu_reset_out_pin chipscope_ila_0::TRIG0
tr_sw	External Ports::mcu_driver_tr_sw_pin chipscope_ila_0::TRIG0
pa_en	External Ports::mcu_driver_pa_en_pin chipscope_ila_0::TRIG0
tx_en	External Ports::mcu_driver_tx_en_pin chipscope_ila_0::TRIG0
rx_en	External Ports::mcu_driver_rx_en_pin chipscope_ila_0::TRIG0
blinky	External Ports::mcu_driver_blinky_pin
⊞ (BUS_IF) S_AXI	Connected to BUS axi_interconnect_1
⊞ tone_generator	
⊞ chipscope_icon_0	
⊞ chipscope_ila_0	
⊟ tx_clock_generator	
CLKIN	External Ports::tx_clock_generator_CLKIN_pin
CLKOUT0	mcu_driver::IPCORE_CLK tone_generator::IPCORE_CLK chipscope_ila_0::CLK
CLKOUT1	dac_driver::IPCORE_CLK
CLKOUT2	External Ports::tx_clock_generator_tx_clk_pin
CLKOUT3	External Ports::tx_clock_generator_rx_clk_pin
CLKFBIN	tx_clock_generator::CLKFBOUT
CLKFBOUT	tx_clock_generator::CLKFBIN
RST	net_gnd

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.

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 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.

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

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] 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, the DAC_Driver, and the MCU_UART.

---

[1] Can be found at http://www.wiki.xilinx.com/Getting Started
[2] Can be found at https://github.com/Toyon/Chilipepper/tree/QPSK_pcore/ChilipepperSupport
[3] 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

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.

```c
#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( 1 );
    Chilipepper_SetTxRxSw( 0 ); // 0- transmit, 1-receive

    // continuously send single frequency tone
    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[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] 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

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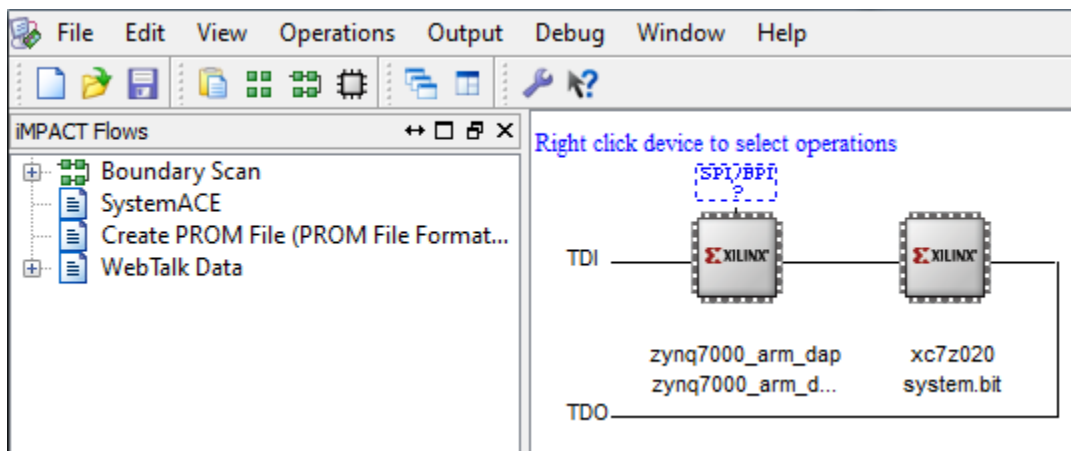
3-2: configuration for Zed Board System.bit file

9. Right click on the xc7z020 board icon (should be on the right), select program and hit ok.
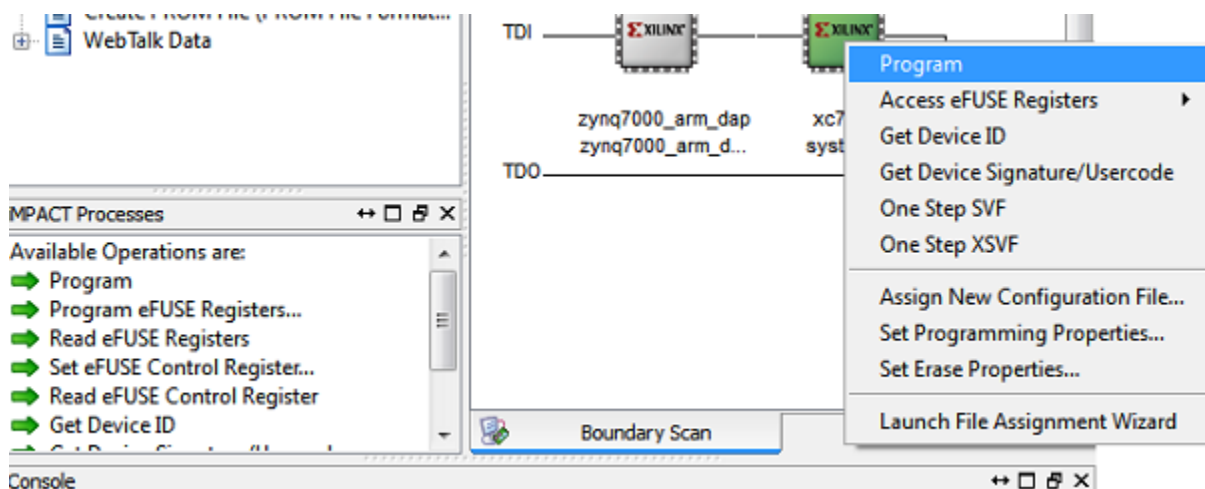

Figure 3-3: iMPACT configuration screen

Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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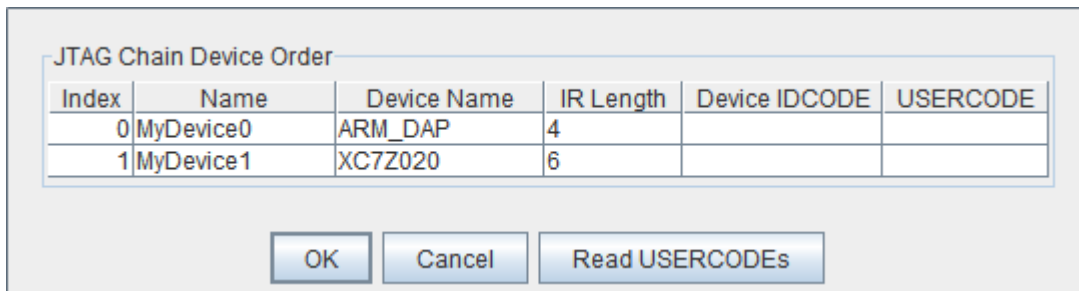
Creating Wireless Transceivers Using MATLAB to HDL translation and Toyon Chilipepper

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.

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

OK   Cancel   Read USERCODEs

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 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.

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 debug signals should look similar to either Figure 4-1 or Figure 4-2 below.
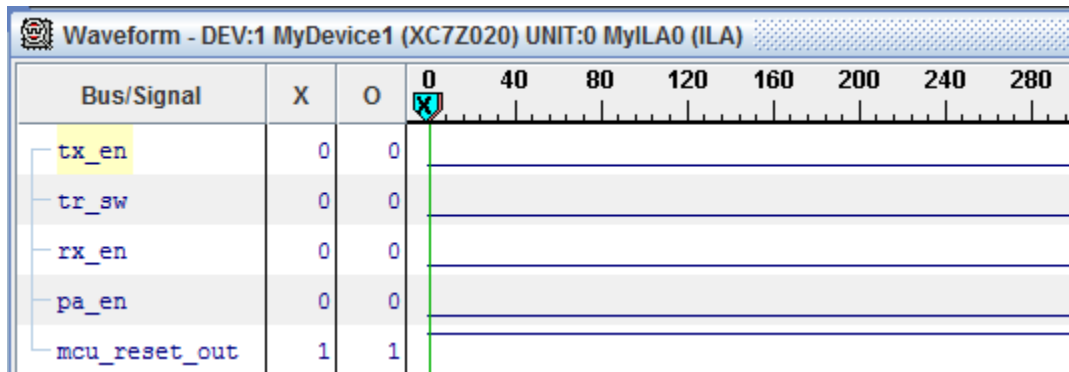

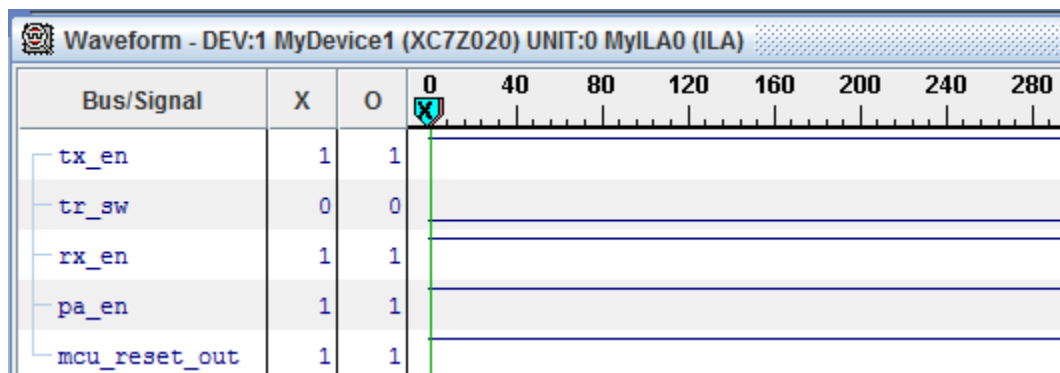**Figure 4-1: MCU Driver signals in ChipScope Pro (before initialization)**

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.
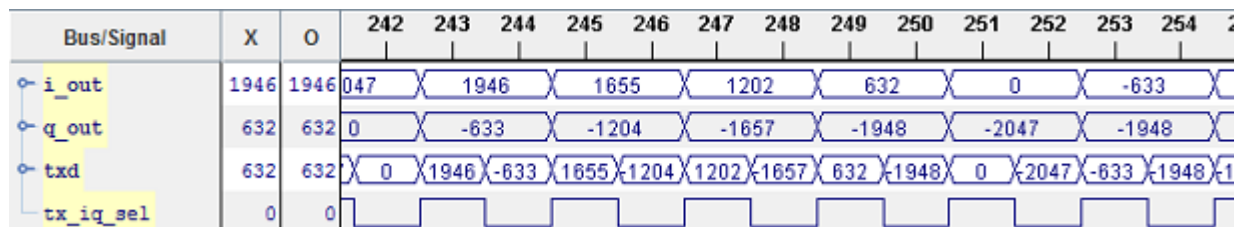

Figure 4-3: DAC Driver signals in ChipScope Pro

Page 29