WIRELESS OPEN-ACCESS
RESEARCH PLATFORM

Lab 3: Building a Simple Transmitter

Patrick Murphy & Siddharth Gupta

Rice University

WARP Project

Document Revision 8

November 11, 2008

1 Introduction

The goal of this lab exercise is to build a simple transmitter and test it in hardware. The transmitter model will be built in System Generator, converted to a peripheral core and integrated with WARP platform support packages in Xilinx Platform Studio. When complete, the design will transmit a sweeping sinusoid at RF using the WARP FPGA and Radio boards.

Note: All files are stored in `C:\workshop\userN\` where `userN` is your user login location. This location will be referred to as `.\` for the rest of the lab.

2 Sinusoid Generator Model

The first step is to explore the sweeping sinusoid generator model in System Generator.

1. Open MATLAB 2007a and change directory to `.\Lab3_SweepingTx\sysgen\`.
2. Open the System Generator model `sweeping_tx.mdl`.
3. Run the model and view the output on the Simulink scope. You should see a sweeping sinusoid.
4. Double-click the **FreqSweepRate** block and change the **Initial Value** parameter to 10.
5. Run the simulation again and verify that the rate of frequency sweep is slower than before.

3 Generating the Peripheral Core

The next step is to convert this model into a peripheral core with a PLB46 bus interface. This interface will allow the PowerPC processor to write the registers in the model to change its behavior at run-time.

1. Open the Simulink library browser, expand the Xilinx Blockset and click on the Index group. Find the **EDK Processor** block in the block listing. Drag this block into the sweeping_tx model.
2. Double click the **EDK Processor** block and click **Add**. This will populate the memory map with entries for each software-accessible register in the model. In this case, there should be two registers: **FreqSweepEnable** and **FreqSweepRate**. Click OK to dismiss this window.
3. Double-click the **System Generator** block. Make sure its parameters match those shown in Figure 1.
4. Click the **Settings** button. In the EDK Project section click the folder icon. Navigate to `.\Lab3_SweepingTx\xps\system.xmp`, click Open, then click OK.
5. Finally, click Generate. System Generator will construct the VHDL design of your model and export it to the EDK project you'll use below. This process will take a few minutes. If it's successful, the dialog box will say Generation Complete.
6. If the process completes successfully, save and close the model and exit MATLAB.

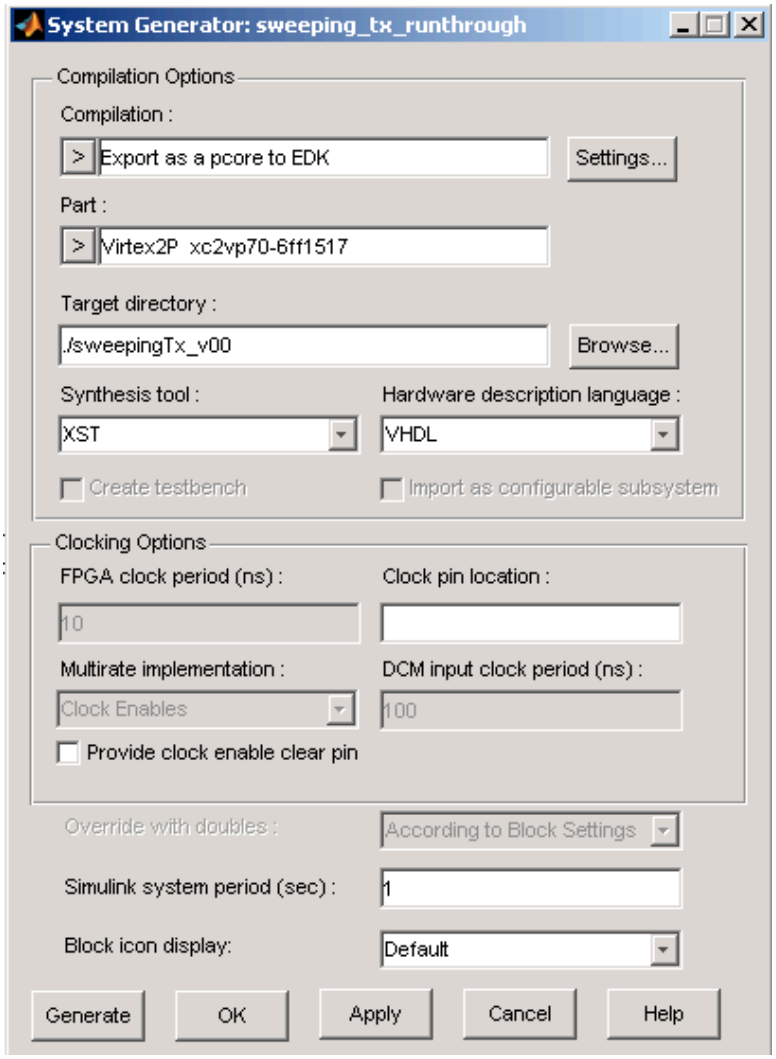

Figure 1: System Generator parameters for PLB core generation

4 Integrating the Transmitter Model

We have provided a pre-built project in Xilinx Platform Studio for this exercise. We built this project using Base System Builder, just like you used in the previous lab exercise. This step integrates your custom transmitter with the hardware/software platform in the EDK project.

1. Open Xilinx Platform Studio. When prompted, select **Open a recent project**, click OK, then navigate to `.\Lab3_SweepingTx\xps\system.xmp`.
2. Click the **IP Catalog** tab on the left of the screen, and expand the **Project Local pcores** / **USER** category.
3. Double-click the **sweeping_tx_plbw** core and click 'Yes'.
4. Click the **Bus Interface** button in the middle of the screen and look for the **sweeping_tx_plbw** core in the list of included peripherals. Expand the core's entry and click the hollow yellow circle to attach the core to the PLB.
5. Switch to the **Ports** view and scroll down to the **sweeping_tx_plbw** entry. Expand the entry. You'll see two ports: **radio2_DAC_I** and **radio2_DAC_Q**. These ports represent the two DAC

blocks in the System Generator model. For each port, click its drop-down list and choose New Connection. This will create a new net for each port.

6. Scroll to the **radio_bridge_slot_2** core and expand its entry. Scroll down the list of ports to **user_DAC_I** and **user_DAC_Q**. In the drop-down list for each, select the corresponding new net created in the previous step; the nets will probably be named sweeping_tx_plbw_0_radio2_dac_i and sweeping_tx_plbw_0_radio2_dac_q. By choosing these nets, you're connecting the outputs of the transmitter model to the FPGA pins mapped to the radio board's D/A converters.

7. When complete, your ports list should look like those shown in Figure 2 (the order of the ports may be different in your project).

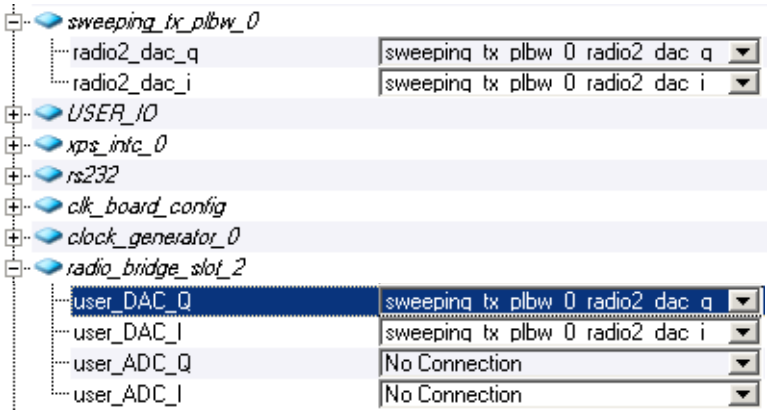

Figure 2: Custom core and radio bridge ports

8. Switch to the **Addresses** view and click **Generate Addresses**. When this process finishes, the **sweeping_tx_plbw** core will have an automatically assigned base address.

9. Choose Hardware → Generate Bitstream to begin the hardware implementation process. As in the previous lab exercise, this step will take 10-15 minutes. While it's running, you can look through the software project (described in the next section).

5 Driving the System from Software

1. We have provided a software project which handles the required radio configuration. It also writes suitable values to the two registers in the sweeping_tx_plbw core.

2. Switch to the **Applications** tab on the left of the screen. Expand the Sources list and double-click the one file entry.

3. Look through the C code to understand how the radio is setup and controlled through the **radio_controller** core. Scroll down in the code and look for the two lines which write registers in the transmitter model. These lines start with `XIo_Out32`. Notice the first arguments in these function calls: `XPAR_SWEEPING_TX_PLBWP_0_MEMMAP_FREQSWEEPENABLE` and `XPAR_SWEEPING_TX_PLBWP_0_MEMMAP_FREQSWEEPRATE`. These constants define the memory addresses of the two registers in the transmitter model. The tools create these constants automatically.

6 Testing the Design in Hardware

1. Make sure your WARP FPGA board is connected to power and USB.

2. In XPS, Choose Device Configuration → Update Bitstream. This will compile any code changes and update the FPGA programming file.
3. Using iMPACT on your local PC, download the bitstream `W:\Lab3_SweepingTx\xps\implementation\download.bit` to your FPGA board.
4. If everything works, the radio's green LED will illuminate to indicate the radio is transmitting, and your node will be generating a sweeping sinusoid at RF. You can observe the waveform on the spectrum analyzer in the lab. It should look something like Figure 3.

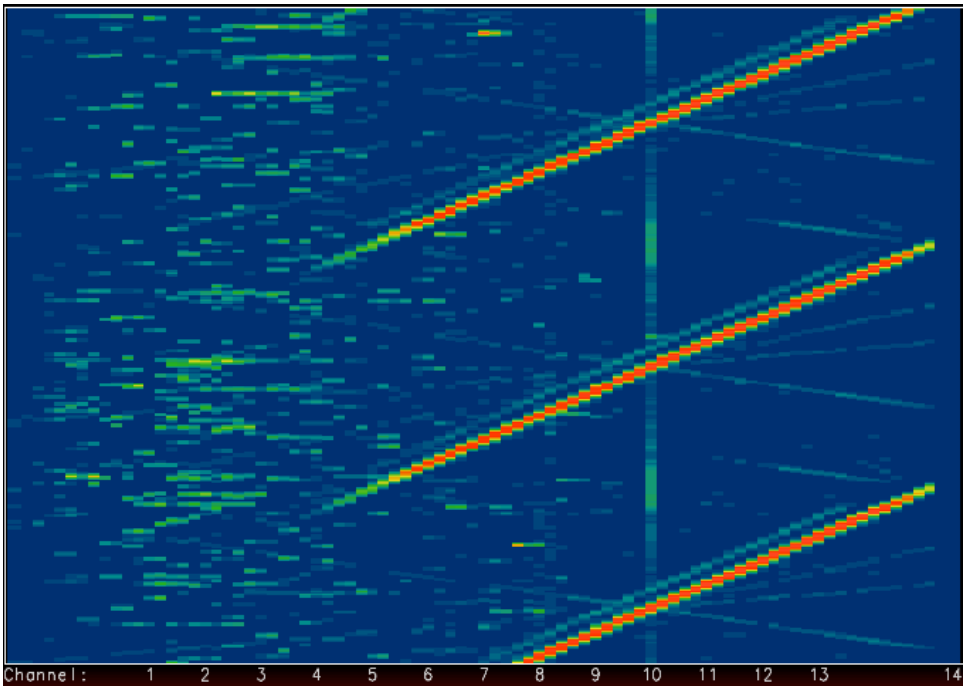

Figure 3: Spectrum Analyzer Output

7 Optional Exercises

If you finish the lab with extra time, here are a few extensions to try:

- Change the frequency sweep rate to switch the slope of your transmitted signal on the spectrogram. You'll have to figure out what value to write to the FreqSweepRate register that effectively negates the frequency accumulator input.
- Use the left/right push buttons to change the radio center frequency. This will involve modifying the callback functions called by the user I/O interrupt for each push button. The code provided already uses the center/up buttons to dis/enable the radio; these are a good model for using the other buttons.
- Try altering some of your radio's settings and observe the changes on the spectrum analyzer. You will need extra functions from the radio controller driver (see http://warp.rice.edu/WARP_API)
  - Change the radio transceiver's center frequency to a different channel.
  - Increase/decrease your radio's transmit gain.
  - Change the transmit low-pass filter bandwidth to its highest setting to see a wider sweep of your transmitted sinusoids.