WIRELESS OPEN-ACCESS
RESEARCH PLATFORM

Lab 3: Building a Simple Transceiver

Patrick Murphy & Siddharth Gupta

Rice University

WARP Project

Document Revision 9

March 27, 2010

1 Introduction

The goal of this lab exercise is to build a simple transceiver and test it in hardware. The transmitter and receiver models will be built in System Generator, converted to peripheral cores and integrated with WARP platform support packages in Xilinx Platform Studio. When complete, the design will use the WARP FGPA and Radio boards to transmit a sweeping sinusoid and measure RSSI in real-time.

This lab exercise has five steps:

1. Design a transmitter in System Generator which generates a complex sinusoid with a sweeping frequency.
2. Integrate the transmitter core with WARP support packages in a Xilinx Platform Studio (XPS) project.
3. Test the transmitter in hardware.
4. Design a System Generator model with computes the running sum of RSSI.
5. Integrate the RSSI summing core into the same XPS project and test it in hardware.

Each of these steps is explained in detail below.

Note: All files are stored in `C:\workshop\`. This location will be referred to as `.\` below.

2 Sinusoid Generator Model

The first step is to build the sweeping sinusoid generator model in System Generator. This model will implement a simple transmitter which generates a complex sinusoid whose frequency is constantly sweeping. The block diagram in Figure 1 shows the basic design for your transmitter.

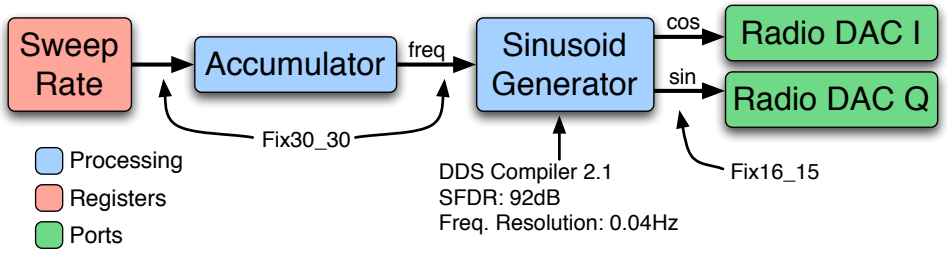

Figure 1: Block diagram of sweeping sinusoid transmitter

1. Open MATLAB 2008a and change directory to `.\Lab3_TxRx\sysgen\`.
2. Open the System Generator model `sweeping_tx.mdl`.
3. This model contains just the registers and DAC output ports for your design. Build the sweeping sinusoid transmitter using blocks from the Xilinx blockset. Figure 1 provides some suggested datatypes and blocks. A complete model is shown in Figure 2. A copy of the complete model is also saved in `.\Lab3_TxRx\sysgen_solution\`.

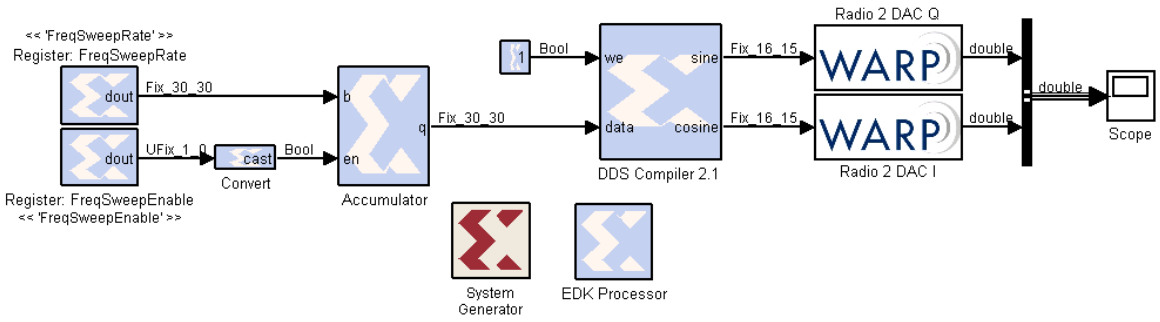

Figure 2: System Generator design of sweeping sinusoid transmitter

3 Generating the Transmitter 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 3.
4. Click the **Settings** button. In the EDK Project section click the folder icon. Navigate to `.\Lab3_TxRx\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. When the process completes successfully, save and close the model and exit MATLAB.

4 Integrating the Transmitter Model

We have provided a template project in Xilinx Platform Studio for this exercise. This step integrates your custom transmitter with the hardware/software platform in the XPS project.

1. Open Xilinx Platform Studio. When prompted, select **Open a recent project**, click OK, then navigate to `.\Lab3_TxRx\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.

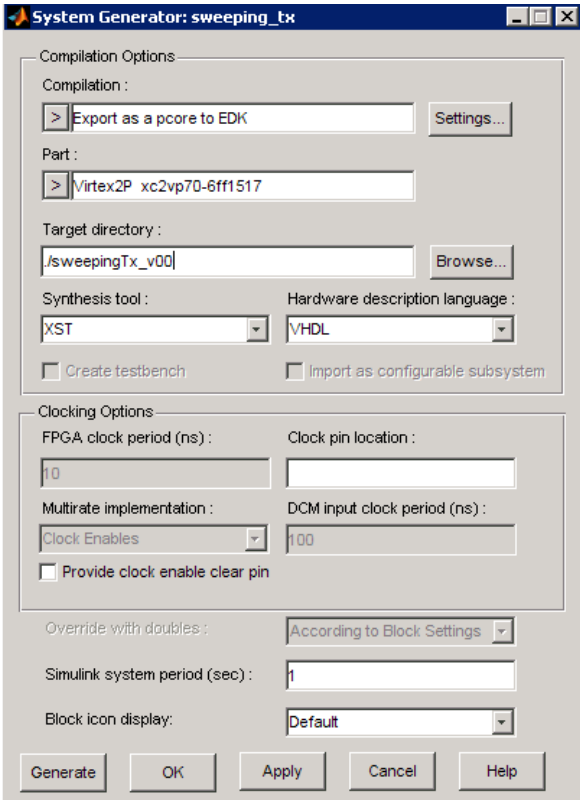

Figure 3: System Generator parameters for PLB core generation

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 4 (the order of the ports may be different in your project).

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

9. Choose Hardware → Generate Bitstream to begin the hardware implementation process. As in the previous lab exercise, this step will take around 5 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. This software template is the minimum necessary to initialize the WARP radio boards and can be

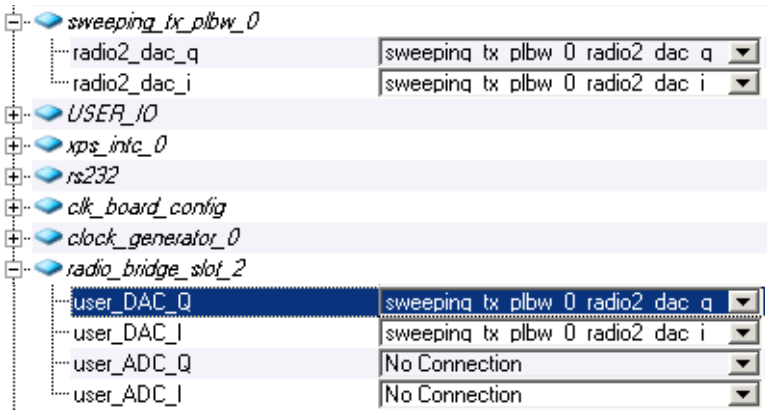

Figure 4: Port connections between sweeping sinusoid transmitter and radio bridge

extended to support custom peripherals. 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 `Lab3_TxOnly` project, then expand its 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 and driver. 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 Transmitter 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. If your C code has any errors, the compiler will print messages in the XPS console. Correct these and re-run the Update Bitstream process.

3. Choose Device Configuration→Download Bitstream to program your WARP kit with your design. This process will print a bunch of status messages to the console. A successful download will finish with `Checking done pin...done`.

4. If everything works, the serial terminal will print out boot messages. Type `T` to enable the radio's transmit path. 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 5. Type `t` to disable the transmitter.

7 Adding the Receiver Model

For this step, you'll build a simple receiver model which calculates the running sum of RSSI samples. Your model should implement the block diagram in Figure 6, which calculates the sum of the 64 most recent RSSI samples, and saves the sum to a register accessible from software.

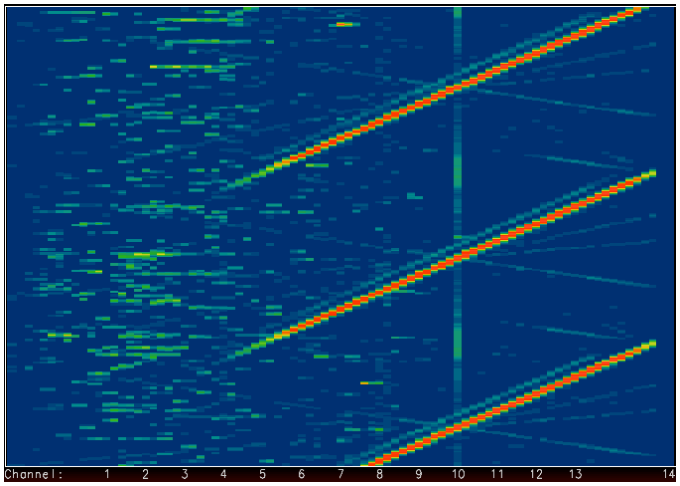

Figure 5: Spectrum Analyzer Output

1. Open MATLAB 2008a and change directory to `.\Lab3_TxRx\sysgen\`.
2. Open the System Generator model `rssi_sum.mdl`.
3. This model contains just the register and RSSI ports for your design. Build the 64-length running sum using blocks from the Xilinx blockset. Figure 6 provides some suggested datatypes and blocks. A complete model is shown in Figure 7. A copy of the complete model is also saved in `.\Lab3_TxRx\sysgen_solution\`.

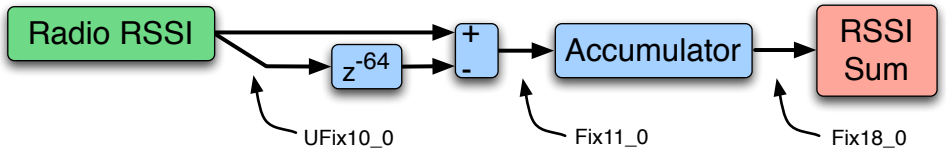

Figure 6: Block diagram of RSSI running sum calculation

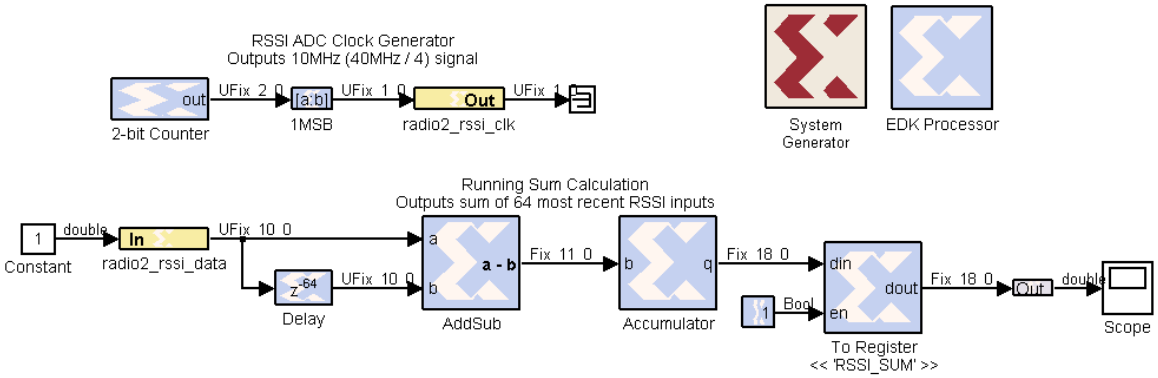

Figure 7: System Generator design of running RSSI sum calculator

4. Generate a pcore for this model and integrate it in your XPS project following the same flow as for the transmitter. You should connect your core's `radio2_rssi_data` and `radio2_rssi_clk` ports to the corresponding ports on the `radio_bridge_slot2` core in XPS.

5. Starting with the `Lab3_TxRx` software project, add a keyboard command to print the RSSI sum by reading the register in your core.

8 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.
- Add keyboard commands to change the radio's center frequency and transmit gains.
- Add a keyboard command which triggers a sequential search of each channel, measuring RSSI and selecting the quietest one for your transmission.