San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #3

Objectives

The objective of this lab is to understand and use vendor specific primitives for Xilinx devices. Vendor specific primitives allow you to access resources in a device that may be cumbersome or difficult to infer through HDL description. In some cases where HDL description is possible, it may be advantageous to use vendor specific primitives if you need precise control over the implementation of the circuit.

In this lab, you will complete a design that plays a selected waveform through a digital to analog converter in response to a pushbutton event. Figure 1 shows a block diagram of the project.

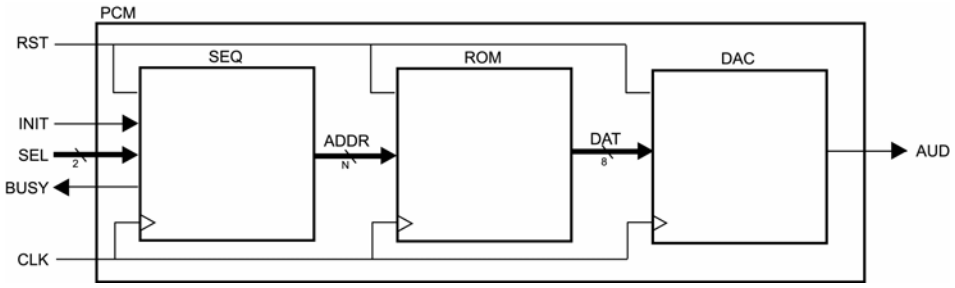

Figure 1: Project Block Diagram

The waveform samples will be stored in ROM located on the FPGA. To access the ROM resources, you will use vendor specific primitives. When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to re-use in the future.

Bibliography

This lab uses *Xilinx Application Note 154, Virtex Synthesizable Delta-Sigma DAC.* I have reproduced portions of the text, figures, and source code related to the use of this module on the Spartan-3 Starter Kit board. Completion of this lab requires the use of an instructor-provided design utility and an audio waveform editing program named GoldWave, or an equivalent substitute.

Unisim Library

The library of vendor specific primitives for Xilinx devices is called the unisim library. There are over 1,000 primitives in this library. A primitive from this library may be instantiated in your design if the primitive is supported for the specific FPGA device you are using. The unisim library contains functional simulation models of the primitives so that you can properly simulate your design. A properly configured synthesis tool will recognize unisim primitives and write them directly into the design netlist.

The Xilinx Software Manuals are a useful source of information. One book in this collection of manuals is called the *Libraries Guide*. The *Libraries Guide* is an encyclopedia of information on the available unisim primitives. For each primitive, the *Libraries Guide* provides a list of ports, a description of behavior, and tables indicating which devices support the primitive.

During design entry, you may instantiate unisim primitives in your design in the same way you instantiate other sub-modules. There is no difference whatsoever.

Project Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3 Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 1, the design has four inputs. There is a clock and a reset input, a 2-bit waveform select, plus a single control signal that is used to initiate audio play.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
init	initiate audio play
sel[1:0]	select one of four waveforms

Also shown in Figure 1 are the two outputs. The busy output is to be observable on an LED. The audio output should be filtered but for this assignment, we will omit the filter and connect to a speaker or headphone directly. You will need to provide your own headphone or speaker.

aud	pwm audio output from delta-sigma dac
busy	indicates waveform playback in progress

The design must not generate audible output until a user elects to initiate audio play. The design must initiate audio play from the beginning of the selected waveform whenever init is asserted, regardless of what it is currently doing (if it is already playing, restart at the beginning). When the design plays to the end of the selected waveform, it must cease generating audible output.

Project Design

There is a downloadable support package for this lab. In it, you are provided with a test bench, the DAC module, and tools to generate the ROM module. You must design the SEQ module, which performs the address sequencing. You must also design the PCM module, which is the top level module of the design.

As you work on your project, start by creating a new source file for the top level PCM module and the SEQ module. Copy the test bench, DAC, and ROM module files into your project directory, and add them as existing sources. You can then instantiate and interconnect them as required.

To facilitate re-use of your completed design, you must implement it in the four modules as described above – it will be a hierarchical design. However, you are not allowed to create additional sub-modules or change the design hierarchy. If you have further questions, or need clarification, consult the instructor.

Digital to Analog Converter Module

You are provided with a delta-sigma digital to analog converter module. Do not modify this module. The module accepts an 8-bit signed binary input, as well as a clock and a reset. It generates a single-bit pulse-width-modulated output signal. When this signal is filtered by an appropriate low pass filter, the resulting analog signal amplitude is linearly related to the 8-bit signed binary input value. If you are interested in how it actually works, read *Xilinx Application Note 154, Virtex Synthesizable Delta-Sigma DAC* which is also included with the downloadable support package for this lab.

Read Only Memory Module

The FPGA on the Spartan-3 Starter Kit board contains a number of 18-kilobit synchronous static RAMs, called BlockRAMs. Both read and write operations are synchronous. BlockRAMs are very useful as they

provide an efficient means for storing small amounts of data without having to use an off-chip memory device. They may be configured as single port or dual port, and may be independently configured on each port in a variety of organizations. As an exercise, locate and read about them in the *Libraries Guide*.

By default, BlockRAMs are initially cleared, with all storage locations set to zero. However, it is possible to specify the initial state of the storage elements. This makes the BlockRAMs very flexible and also enables the implementation of BlockROMs. A BlockROM is an initialized BlockRAM with the write enable signal permanently de-asserted.

In this project, you will create and initialize a BlockROM using two utilities provided in the downloadable support package. In order to do this successfully, you first need to generate a binary data file containing the desired BlockROM contents. The amount of storage you may use for your complete set of four waveforms is 8192 bytes, which consumes four BlockRAM resources.

With a sample rate of 8192 Hz, the total play time of the combined four waveforms must be less than or equal to 1.000 seconds. It is possible to reduce the sample rate in exchange for increased play time. For example, reducing the sample rate to 4096 Hz increases the total available play time to 2.000 seconds but the quality of the audio is lower. Short and recognizable sounds like beeps, bells, chimes, and cuss words are suggested. The waveform manipulation can be done with the GoldWave program:

- Locate or create four waveforms that satisfy the design requirements and concatenate.
- Trim as much "silence" from the resulting concatenated waveform as possible.
- Maximize the waveform amplitude so that you are using the full dynamic range.
- Decide if re-sampling is required, and if so, what the new sample rate should be.
- Perform a low-pass filter with a cut-off frequency of half the new sample rate.
- Perform the re-sampling operation; verify the waveform is 8192 or fewer samples.
- Using the viewer, make note of the start and end sample numbers for each of the four waveforms in the composite. This *must* be done after re-sampling, not before!
- Save the audio sample in raw data format, 8-bit, mono, signed.

Once you have a binary data file, do the following:

- Locate and study the raw2ver.c file provided in the downloadable support package. This program takes a binary data file as input and outputs a Verilog-HDL file implementing a BlockROM initialized with the data in the input file. The compiled version of this program, raw2ver.exe, is also provided.
- Use raw2ver.exe to turn your data file into a Verilog-HDL file for your BlockROM. There is a text file containing short instructions on how to do this, as well as a previously generated example so you can know what to expect in the output file. Notice how the BlockRAMs are instantiated, and also how they are initialized. That would have been a lot of typing!

At this point, you will have an additional source file to use with your design. This BlockROM has clock, reset, and address inputs. The width of the address input will depend on the size of the BlockROM you have created; check the file to determine the width of the address input. There is an 8-bit data output.

Address Sequencer Module

You must create the address sequencer. The address sequencer provides an address to the BlockROM. Every sample period during playback, the address sequencer should increment its address output. The sample period is set by the sample rate of your final composite waveform.

Before you begin writing any code, you must sit down with scratch paper and draw a block diagram of a circuit that will satisfy the design requirements. As a hint, one possible implementation can be realized using two "fancy" counters, two large multiplexers, and comparators. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

Project Verification

You must perform some minimal functional simulation of the design. This is important for two reasons. First, it will give you confidence your design is working properly before you implement it. Second, if the design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug logic problems (incorrect design behavior) unless you have a block diagram and are able to run a simulation.

A simple test bench is included with the downloadable support package for this lab. The test bench can be used for functional simulation as well as timing simulation after the implementation step. Feel free to enhance the basic test bench as you see fit.

Project Synthesis

Synthesize your design exactly as you have done before. Do not forget to check the synthesis report. This report will tell you how many clocks exist in your design, under the "clock information" summary. If you have more than one clock, you need to go back and correct your design. Also check to see if any latches were used, you should not have any. These can be found in the "cell usage" summary. If you see anything starting with LD (Latch, D-type) then you need to go back and correct your design.

Project Implementation

Before you implement your design, you will need to add a constraints file and edit the I/O locations and properties. Figure 3 shows how and where to connect your speaker or headphone to the Spartan-3 Starter Kit board. Pin 6 on Expansion Port A2 is connected to pin C5 on the FPGA.

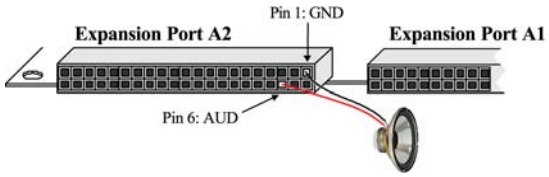

Figure 3: Speaker or Headphone Connection

You must properly constrain the clock input. Use BTN3 for reset and BTN0 for init. Use SW1 down to SW0 for the sel inputs. Use LD0 for the busy output. For all signals, do not forget to properly specify the I/O standard as LVCMOS33. Verify your design in hardware. When you are satisfied with the results, generate a programming file for the PROM and then load your design into the PROM.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within six hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. Use WinZIP to archive the entire project directory, and name the archive l3_yourlastname.zip. For example, if I were to make a submission, it would be l3_crabill.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove:

- The xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should turn in what you have to receive partial credit.