San Jose State University
Department of Electrical Engineering
EE178, Fall 2007, Crabill

Laboratory Assignment #2

Objectives

This lab covers simple logic design using familiar building blocks. From your previous coursework, you should already be familiar with flip flops, counters, and multiplexers. The goal of this lab is for you to implement a circuit that generates square waves at specific frequencies based on user input. 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.

Figure 1: Mechanical Music Player

Now that you are familiar with the tools from Laboratory Assignment #1, you should be able to concern yourself with digital design. Figure 2 shows a symbol of the module you will create.

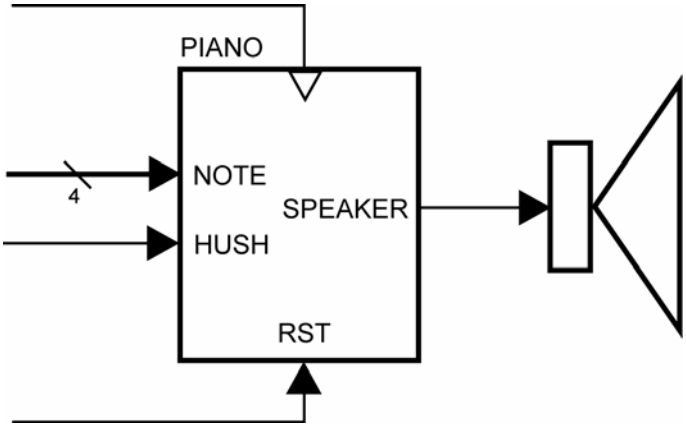


Figure 2: Symbolic Representation

Bibliography

This lab draws heavily from the *Spartan-3E Starter Kit User Guide*. For convenience, I have reproduced portions of the text and figures in this document where applicable.

Module 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-3E Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 2, the module has four inputs. There are clock and reset inputs, plus a four-bit input to select one of 16 tones and an additional input to silence the audio output. There is a single output signal, which is intended to be connected to some form of audio output device.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
note[3:0]	frequency select
hush	stop toggling; be silent
speaker	audio output

The module must drive the speaker output to generate a square wave at the frequencies specified in the next section. The speaker output must stop toggling in response to the assertion of the hush signal. The hush signal is to be treated as an active low enable signal for the speaker output. Assertion of hush should simply disable future transitions – it should not cause the speaker signal to change value. The reset signal, on the other hand, certainly disables transitions on the speaker output, but it also forces the speaker output to a known initial value.

Designing the Module

There are many different techniques for creating audio output. The simplest technique for creating elementary tones is to generate a periodic waveform and then send that waveform to a speaker. For example, if you take a function generator (available in most analog laboratory courses) and connect the output to a speaker, you can actually hear the output when it is in the range of human hearing. This range is approximately 20 Hz to 20,000 Hz. With the function generator, you can control:

- The frequency – which your brain interprets as the pitch, or note heard.
- The amplitude – which your brain interprets as the volume, or loudness heard.
- The waveshape – which your brain interprets as a timbral characteristic, or “instrument” heard.

You are probably familiar with frequency and amplitude. The waveshape changes the quality of the note you hear, because the waveshape changes the number and amplitude of harmonics associated with the fundamental frequency of the waveform. A sinusoidal waveshape gives a perfect tone; that is, if you were to look at the Fourier transform of the time domain signal, you would see a spike at the fundamental frequency of the waveform itself, and no other frequency components. On the other hand, if you were to look at the Fourier transform of a square, saw tooth, or triangle waveshape, you would still see the largest spike at the fundamental frequency of the waveform, but you would see other, smaller spikes at multiples of the fundamental frequency. These are called overtones, or harmonics.

Without the aid of digital to analog converters or other analog circuitry, it is cumbersome to produce anything other than a square wave with digital logic, so we will create square waves as output. Square waves are simple to create; you can periodically toggle a signal, and then send that signal to a speaker to create audio output. For the amplitude parameter, we will simply use the output of the FPGA device directly, with no provision for controlling the amplitude.

The final parameter is the frequency parameter. In designs that only require a buzzer, such as an alarm, any audible frequency will suffice. In this lab, however, the frequency is important since we will later use the resulting tones to play music that was intended for a keyboard instrument, such as a piano. Keyboard instruments are tuned using what is called the equal tempered music scale. In short, what this means is that the frequency of each note is related to the frequency of the adjacent notes by a constant multiple. The result is that music can be played equally well, or equally badly, in any key. Figure 3 shows a keyboard of sixteen notes. The piano design will be capable of playing any note on this keyboard.

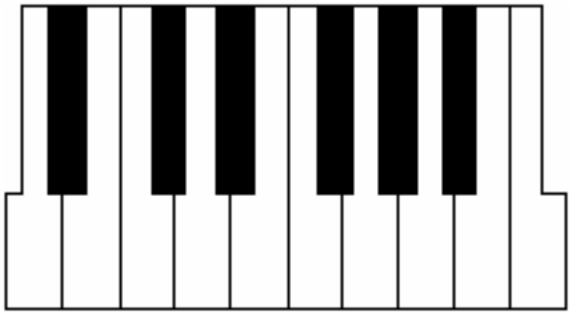


Figure 3: Our Virtual Keyboard

For those of you who play instruments, some of this may be familiar. I do not play an instrument, by the way, and I still made this circuit work correctly! While generating the data shown in Figure 4, I was running under the following assumptions:

1. Sixteen keys cover more than one octave, which should be sufficient.
2. The frequency of a key called "Concert A" or A4 is 440 Hz.
3. The twelfth root of two relates the frequency of adjacent keys.
4. The single clock used in this design is running at a nominal 50 MHz.

Oscillator Frequency	50000000
Oscillator Period	0.00000002

Binary Note Code	Keyboard Note	Desired Frequency	Note Period	Half Period Clk Cycles	Rounded Clk Cycles	Terminal Count
0000	A4	440.00	0.002273	56818.18	56818	16'hDDF1
0001	A#/Bb	466.16	0.002145	53629.22	53629	16'hD17C
0010	B/Cb	493.88	0.002025	50619.25	50619	16'hC5BA
0011	C/B#	523.25	0.001911	47778.21	47778	16'hBAA1
0100	C#/Db	554.37	0.001804	45096.62	45097	16'hB028
0101	D	587.33	0.001703	42565.54	42566	16'hA645
0110	D#/Eb	622.25	0.001607	40176.52	40177	16'h9CF0
0111	E/Fb	659.26	0.001517	37921.59	37922	16'h9421
1000	F/E#	698.46	0.001432	35793.21	35793	16'h8BD0
1001	F#/Gb	739.99	0.001351	33784.29	33784	16'h83F7
1010	G	783.99	0.001276	31888.13	31888	16'h7C8F
1011	G#/Ab	830.61	0.001204	30098.38	30098	16'h7591
1100	A5	880.00	0.001136	28409.09	28409	16'h6EF8
1101	A#/Bb	932.33	0.001073	26814.61	26815	16'h68BE
1110	B/Cb	987.77	0.001012	25309.62	25310	16'h62DD
1111	C/B#	1046.50	0.000956	23889.10	23889	16'h5D50

Figure 4: Calculating Half-Period Cycle Counts

The terminal count column contains integer values that will be used in conjunction with a counter to create the sixteen different notes. In addition, we will also need some method of telling the tone generator to be silent. This is useful for creating short pauses between notes; otherwise, you could not distinguish between two repeated notes and one long note.

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, consider that one possible implementation of this module can be realized using a counter, a multiplexer, and a toggle flip flop. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

To facilitate re-use of your completed design, you must implement it in a single module – you are not allowed to use hierarchical design with sub-modules for this assignment. If you have further questions, or need clarification, consult the instructor.

Testing the Module

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, 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.

In order to help you get started, here is a template for a test bench that works with the module you are designing. Feel free to enhance this basic test bench as you see fit.

```
// File:  testbench.v
// This is the top level testbench for EE178 Lab #2.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator timestep should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module testbench;

  // Generate a free running 50 MHz clock
  // signal to mimic what is on the board
  // provided for prototyping.

  reg clk;

  always
  begin
    clk = 1'b1;
    #10;
    clk = 1'b0;
    #10;
  end

  // Now, generate a reset assertion that
  // takes place at time zero and then
  // deasserts 100 ns later.  In this block,
  // also include a mechanism to exercise
  // the design and then finally stop the
  // simulation.

  reg rst;
  reg [3:0] note;
  reg hush;
  integer loopvar;

  initial
```

```
  begin
    $display("Simulation starting...");
    rst = 1'b1;
    note = 4'h0;
    hush = 1'b1;
    #100;
    $display("Reset signal released.");
    rst = 1'b0;
    // Loop through all 16 possible notes
    // and also exercise the hush signal.
    for (loopvar = 0; loopvar < 16; loopvar = loopvar + 1)
    begin
      hush = 1'b0; // make noise
      note = loopvar[3:0]; // assign note
      #10000000; // allow it to run
      hush = 1'b1; // go quiet
      #1000000; // allow it to run
     end
    $display("Simulation finished.");
    $stop;
  end

  // Now instantiate the top level design.

  wire speaker;

  piano piano_inst (
    .rst(rst),
    .clk(clk),
    .hush(hush),
    .note(note),
    .speaker(speaker)
    );

endmodule
```

Figure 5 shows what you might expect to see after running this testbench. At this zoom level, the waveform display does not show the clock or speaker signals toggling; instead, it shows a cross-hatch pattern indicating that the signal is active.

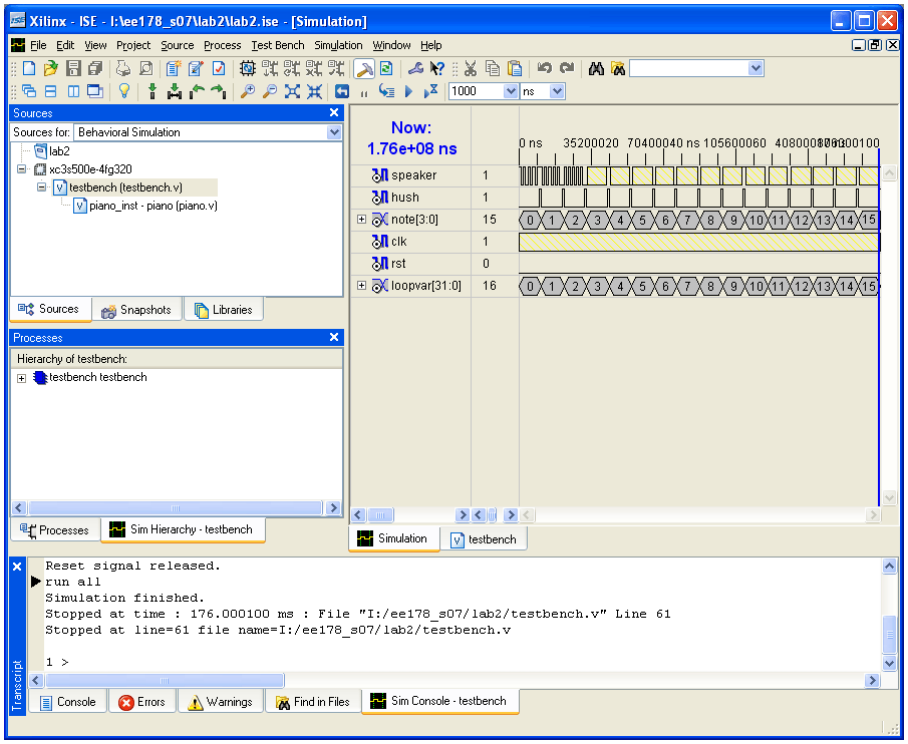


Figure 5: Simulation Results

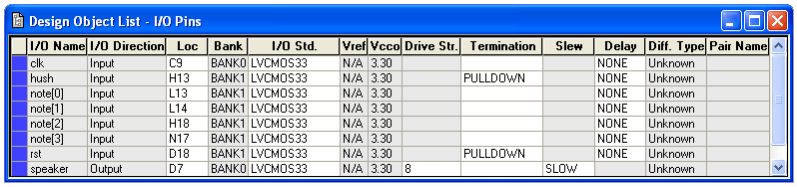
To confirm the circuit operates properly, you can zoom in and measure the period of the output waveform for each value of the note input. You should also visually confirm that the output waveform stops toggling when hush is asserted.

Synthesizing the Module

Synthesize your design exactly as you did in the tutorial. Do not forget to review 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.

Implementing the Module

Before you implement your design, you will need to add a constraints file and edit the I/O locations and properties using PACE. Use the details in Figure 6 when entering your constraints.



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name
clk	Input	C9	BANK0	LVCMOS33	N/A	3.30				NONE	Unknown	
hush	Input	H13	BANK1	LVCMOS33	N/A	3.30		PULLDOWN		NONE	Unknown	
note[0]	Input	L13	BANK1	LVCMOS33	N/A	3.30				NONE	Unknown	
note[1]	Input	L14	BANK1	LVCMOS33	N/A	3.30				NONE	Unknown	
note[2]	Input	H18	BANK1	LVCMOS33	N/A	3.30				NONE	Unknown	
note[3]	Input	N17	BANK1	LVCMOS33	N/A	3.30				NONE	Unknown	
rst	Input	D18	BANK1	LVCMOS33	N/A	3.30		PULLDOWN		NONE	Unknown	
speaker	Output	D7	BANK0	LVCMOS33	N/A	3.30	8		SLOW		Unknown	

Figure 6: Details, Details…

The clock input is assigned to the correct location to receive the 50 MHz clock signal from the on-board oscillator. The note inputs are assigned to the switches, and the reset and hush inputs are assigned to buttons. Review the pin assignments in Figure 6, and then inspect the annotations on your board to find the location of each button and switch.

The speaker output pin assignment is not immediately obvious. Locate the 6-pin accessory header named J4. This header is shown in Figure 7 and has 6 pins – four signal pins, plus power and ground. You should be able to identify which pin on this header is used for the speaker output. You will connect your audio output device between this pin and ground.

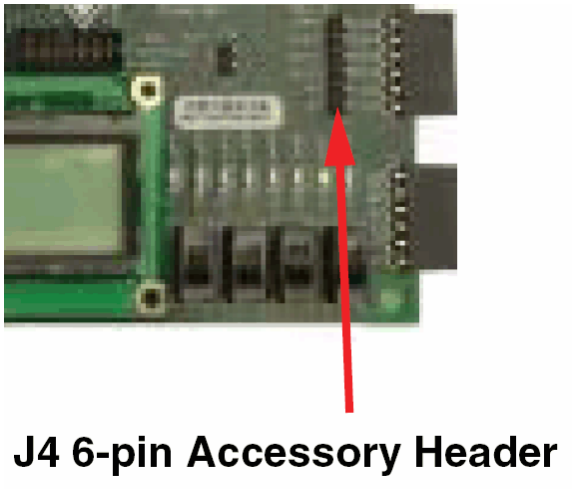


Figure 7: Speaker Connection Location

Once you have entered the pin assignments, save the constraint file, exit PACE, and implement the design. After a successful design implementation, generate a programming file and launch iMPACT to test the design in hardware. When you are satisfied with your result, program it 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 nine 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 l2_yourlastname.zip. That is "l2" as in Lab 2, **not** "12" as in twelve. For example, if I were to make a submission, it would be l2_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 subdirectories in the project folder.

Demonstrations must be made on the due date. If your circuit is not completely functional by the due date, you should demonstrate and submit what you have to receive partial credit.