Simple Speaker with Inputs using Verilog on FPGA

Alan Huang, Umar Khan, and Michael Kim[1]
[1]*PHYS-234: Digital Electronics (Professor Brody)*

I. INTRODUCTION

For this project, we used a medley of equipment. This includes the Basys 2 Spartan-3E FPGA Trainer Board, a computer running Windows, a connecting cable from the FPGA board to the computer, the Xilinx ISE Design Suite, the Digilent Adept software, and a 2-pin speaker. Furthermore, we used Verilog syntax posted on Stack Overflow, ASIC World, and FPGA4Fun, all websites with helpful comments on the writing of Verilog code. FPGA4Fun in particular motivated our brief use of modular arithmetic.

We began with the goal of manipulating a small speaker using the input switches on the FPGA board. To accomplish this, we connected the FPGA board to the computer with the connecting cable and used Xilinx ISE Design Suite to write Verilog code, and finally "flashed" the generated *.bit* file to the FPGA board using Digilent Adept.

II. PROCEDURE

After preparing the FPGA board by connecting it to the computer, we installed our 2-pin speaker to the board. Fortunately, the dimensions of the speaker made it easy to attach. One pin on the speaker represents a data bit, setting the speaker to emit noise if and only if the voltage is high. This pin was inserted into the output "A3" on the board. The other pin on the speaker represents a voltage control pin, inserted into *VCC* (voltage at the common collector). This pin serves as a ground for the speaker, completing the circuit.

III. VERILOG MODULE CODE

We coded six inputs including four buttons, one switch, and a clock with frequency 50 MHz. The only output on the board was the speaker.

```
module waves(input[3:0] button,
input switch, input clk,
output reg speaker);

 // first create a 32-bit binary counter
 reg [31:0] counter;

 always @(posedge clk) begin

  // switch adds an octave to frequency
  // counter incremented on rising edges
  counter <= counter+1+switch;

  // use most significant bit of counter
  case(button)
   0: speaker = 0;
   1: speaker = counter[14];
   2: speaker = counter[15];
   4: speaker = counter[16];
   8: speaker = counter[17];
  endcase

 end

endmodule
```

IV. VERILOG CONSTRAINTS CODE

The constraints file was programmed with the FPGA board-specific inputs and outputs. For instance, the four buttons were treated as a bus and given index values 0 through 4 from right to left (mapped to "A7," "M4," "C11," "G12," respectively).

```
net speaker loc = "A3";
net clk loc = "B8";
net button[0] loc = "A7";
net button[1] loc = "M4";
net button[2] loc = "C11";
net button[3] loc = "G12";
net switch loc = "P11";
```

V. COMMENTS ON VERILOG CODE

Clock **clk** serves the purpose of continuously oscillating in a step-wise manner between high and low voltage. We recognized that the initial frequency of the clock was too high for human hearing. Given that it is reasonable to hear notes in concert-A on the order of 440 Hz, we had to access a bit which oscillated from high to low at a much slower frequency. The simplest way to accomplish this was to create another variable **counter** which incremented on each rising edge of the clock. Allocating to **counter** 32 bits means that the first bit **counter[0]** will have the same frequency as the clock. It follows then that the next bit **counter[1]** will oscillate at half the frequency as the preceding one (25 MHz).

By simple experimentation, we found that **counter[15]** produced an audible sound. The switch served the purpose of modulating the frequency after

we selected a frequency using the buttons. If the switch was activated, then for each increment of **counter** we added one least-significant binary digit. This has the effect of raising the sound by one octave, because the binary number **counter** is increasing twice as fast.

VI. FURTHER REMARKS

We make some additional remarks about the project. Although not implemented in this project, we recognize that it is possible to emit sounds that are not in our generated class of sounds (up to octave-shift). That is, when we emit the sound by using the frequency of oscillation of **counter[15]** as an output, we are really playing a sound of frequency:

$$freq = \frac{50MHz}{2^{n-1}}$$

where n is the index of **counter**. Then, we can reach a note of specific frequency (e.g. concert-A, which is 440 Hz) by dividing the clock frequency by some integer d to reach 440 Hz. In this case of 440 Hz, we let d be 113636. Then, we can implement two always blocks:

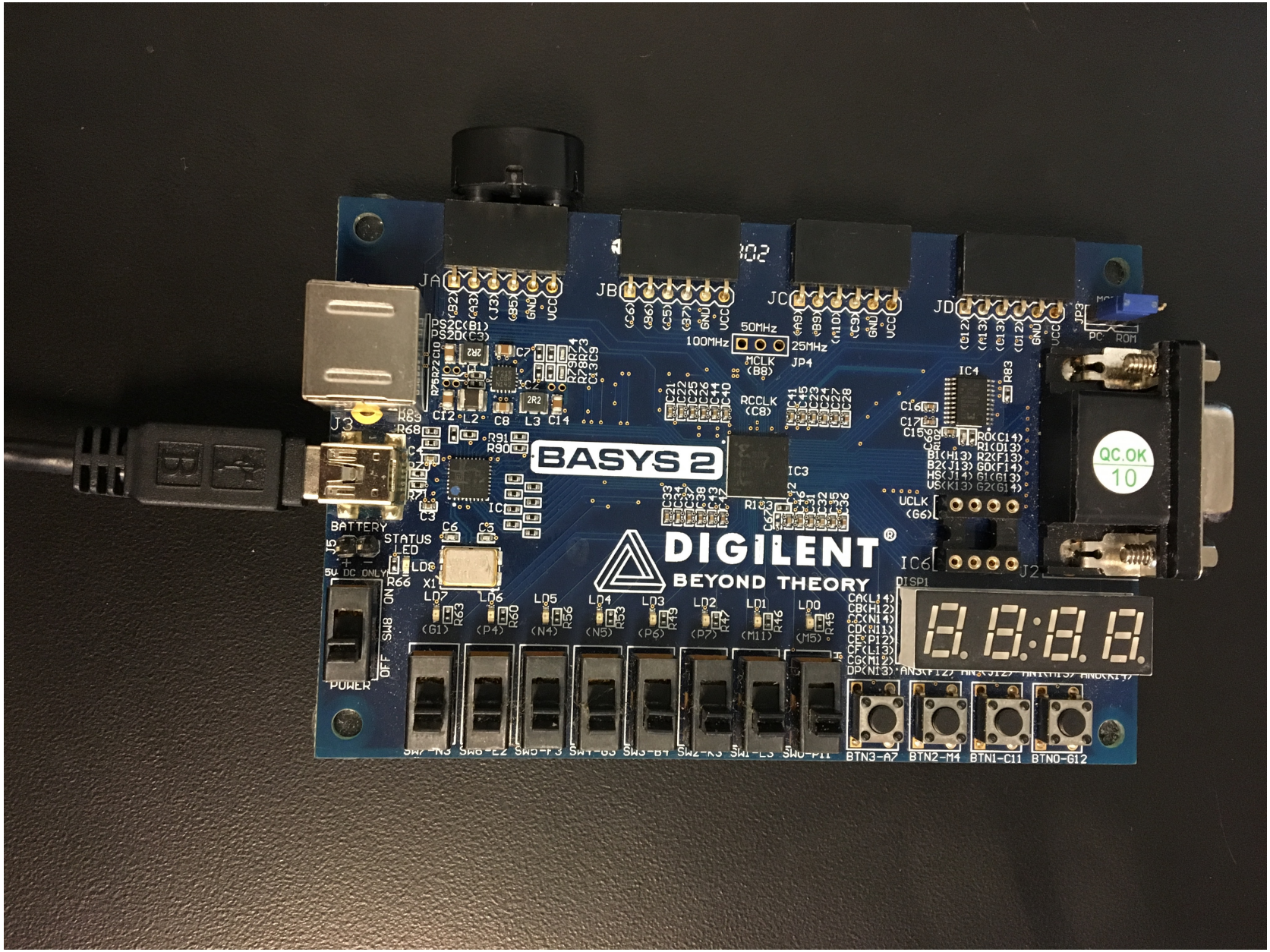
```
module waves(input clk,
output reg speaker);

  reg [31:0] counter;

  always @(posedge clk)
    if(counter == 56818)
      counter <= 0;
    else
      counter <= counter+1;

  always @(posedge clk)
    if(counter == 56818)
      speaker <= ~speaker;

endmodule
```

Note that we now have values for **counter** in the reduced residue class mod 56818, which effectively divides the clock frequency by that number. We use d divided by 2 because we note that using d allows flipping the speaker bit only for the rising edges of the clock input, whereas we want two such changes for each rising edge. Now, when **counter** reaches value 56818, it is reset to 0. Finally, the speaker bit is flipped only when **counter** reaches value 56818. This way, we give the speaker sound a frequency of 440 Hz.

VII. CONCLUSIONS

We see that the small 2-pin speaker can be used to generate almost any frequency (up to the floating-point precision of arithmetic operations). The immediate application of our implementation is that we have created an octave keyboard with four input buttons and a switch that changes the frequency.

This can be used to create simple musical tunes as well as act as a tuner for musicians. Furthermore, if we were to implement the selection of different octave classes (as in the prior section), then we would allow for a keyboard with more possible notes.


BASYS 2
DIGILENT®
BEYOND THEORY
QC.OK
10
JA
JB
JC
JD
50MHz
100MHz
25MHz
MCLK
(B8)
JP4
RCCLK
(C8)
IC1
IC2
IC3
IC4
IC6
J2
J3
J5
BATTERY
STATUS
LED
LD8
LD7
LD6
LD5
LD4
LD3
LD2
LD1
LD0
POWER
ON
OFF
SW8
DISP1
UCLK
(G6)
BTN3-A7
BTN2-M4
BTN1-C11
BTN0-G12