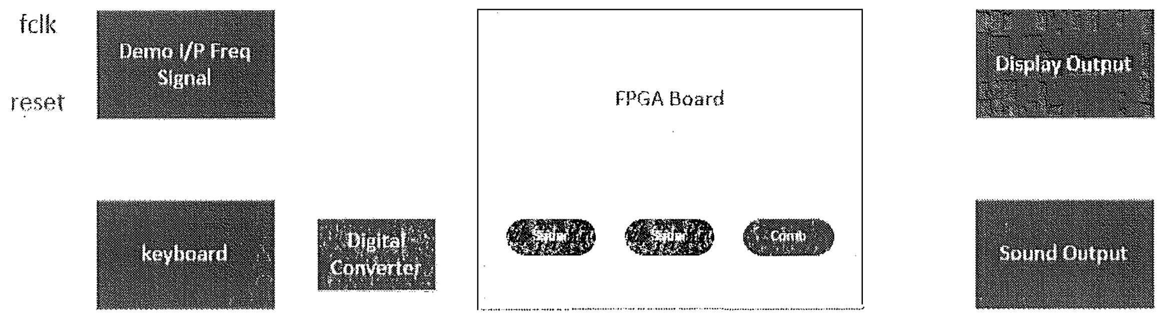
ELEX 7660: Digital System Design
Guitar Synthesizer

Student Names: Aaron Fernandes Daniel Liu		Set: S

Summary

This project is a guitar synthesizer which allows you to press a button on a keypad and outputs sound. The notes correspond to the first four frets of a standard 4 string bass guitar. It is based off of a similar keyboard synthesizer referenced below.

Block Diagrams



State Machine

Schematics

Code

```
/* Description: To combine a row and column element for finding key pressed
location
*      Input: 4 bit i/p
*      Output: 4 bit o/p number and an o/p for indicating active high or low
*/
module kpdecode ( input logic [3:0] kpr, kpc,
                  output logic kphit,
                  output logic [3:0] fre_next2 ) ;

   logic [7:0] key_pressed;

   always_comb begin

      key_pressed = {kpr , kpc};

      if (kpr == 4'b1111) begin
         kphit = 0;
       fre_next2 = 0;
      end

      else begin
         kphit = 1;   //   key pressed
         unique case (key_pressed)
            //  G String
          8'b0111_0111: fre_next2 = 392; // 1
          8'b0111_1011: fre_next2 = 416; // 2
```

```
            8'b0111_1101: fre_next2 = 440; // 3
            8'b0111_1110: fre_next2 = 466; // A
              //  D String
            8'b1011_0111: fre_next2 = 294; // 4
            8'b1011_1011: fre_next2 = 312; // 5
            8'b1011_1101: fre_next2 = 330; // 6
            8'b1011_1110: fre_next2 = 350; // B
              //  A String
            8'b1101_0111: fre_next2 = 220; // 7
            8'b1101_1011 : fre_next2 = 234; // 8
            8'b1101_1101: fre_next2 = 246; // 9
            8'b1101_1110: fre_next2 = 262; // C
              //  E String
            8'b1110_0111: fre_next2 = 164; // E
            8'b1110_1011: fre_next2 = 174; // 0
            8'b1110_1101: fre_next2 = 184; // F
            default: fre_next2 = 196; // D //num

         endcase
      end
   end
endmodule


/* Description: Combine 4 bits keypad row element input with a clock and an active
low input
*                   into 4 bits keypad output.
*
*      Input: 4 bit keypad row, a clock and an active low rest i/p
*      Output: 4 bit o/p keypard column
*/
module colseq ( input logic [3:0] kpr,
                input logic clk,reset_n,
              output logic [3:0] kpc ) ;

   logic [3:0] kpc_next ;

   always_comb begin

      if (!reset_n)
         kpc_next = 4'b0111;
      else if (kpr == 4'b1111)
         unique case (kpc)
          4'b0111: kpc_next = 4'b1011;
          4'b1011 : kpc_next = 4'b1101;
          4'b1101: kpc_next = 4'b1110;
            default: kpc_next = 4'b0111;
         endcase
      else
         kpc_next = kpc;
   end

   always_ff@(posedge clk)
      kpc <= kpc_next;

endmodule

/* tonegen.sv - tone generator for ELEX 7660
*  Author: Daniel Liu (6S A00930876)
*  Module: tonegen - Creates a square wave on the spkr (speaker) output
*                              at a frequency given by the 'freq' control register.
*  input: 4 bit kpr, clock(clk) and an active low reset(reset_n) input
*  output: 8 bit leds for 7-LED segement
*/

module tonegen
  #( logic [31:0] fclk )          // clock frequency, Hz
   ( input logic [31:0] kphit,
     input logic [3:0] fre_next2,
     output logic spkr,           // on/off output for audio
     input logic reset, clk ) ;

      logic signed [31:0] count;
      logic signed[31:0] count_next;
      logic [31:0] fre, fre_next;

      logic spkr_next; // set up frequency register

      always_comb begin
            if(reset)
            begin
                  spkr_next = 0;
                  count_next = fclk;
                  fre_next = 0;
            end
            else if(kphit)
            begin
                  fre_next = fre_next2;
                  count_next = count;
                  spkr_next = spkr;
            end
            else if(count >0)
            begin
                  fre_next = fre;
                  count_next = count - (2*fre);
                  spkr_next = spkr;
            end
            else
            begin
                  fre_next = fre;
                  count_next = fclk;
                  spkr_next = ~spkr;
            end
      end
      always_ff@(posedge clk) begin

            count <= count_next;
            fre <= fre_next;
            spkr <= spkr_next;
      end
endmodule // end module
```

Suggestions for Future Work

Using a PWM signal connected to a simple low pass filter (RC) would allow you to create waveforms other than square waves. Also displaying the output onto a monitor.

References

Reference:

1. (Alec Steinkraus ECE 385 final project at University of Illinois Urbana-champaign) https://www.youtube.com/watch?v=cXik7ouF2zE

2. Litmanovich,V, https://www.youtube.com/watch?v=ANxHyCAYGp0