ECE241 Digital Systems 2014 fall

Verilog Project – Bass Synthesizer Technical Report

Yuan Ming Chen 1000652493, Victor Ouyang 1000659343

Station #8   Lab Room #3135

December 1, 2014

Introduction

Our final project is a bass synthesizer and sequencer, which takes user inputs from a PS2 keyboard and the DE2 switches, and outputs sounds using the on-board audio codec chip. Our project was inspired by Roland's TB-303 Bass Line, whose characteristic sound had a critical role in the development of electronic music. Many of our design choices, such as the step-time note entering method, were based on those of the 303. We decided early on that it would not be very meaningful to do a graphics related project, as we had both already done so in the past (using software programming languages). We also didn’t own an external personal monitor, this would make this lab more time consuming. Our main goal in this project was to learn new things about both digital systems and the DE2 board, and we felt that the best way to achieve this was to go into the "unexplored territories" involving the PS2 controller, audio controller, as well as PATA connected control knobs and sliders which we had planned for, but unfortunately did not have time to implement. Our secondary goal was to learn about audio signals and synthesis, as it very relevant in the ECE field, and is covered in both current and future courses. We feel that by choosing this particular project, we have gained valuable knowledge which we would not have had we done a graphics based project.

Our Design

The majority of the logic in this project is contained in the state machine module. In addition, it uses 12 other modules which perform integral roles, such as encoding signals from input devices, decoding signals for output devices, and making calculations from the on-board 50MHz clock. The design's block diagram and the state diagram for the state machine can be found in the Appendix.

-State Machine
There are 6 states in the state machine, used to represent the 3 modes of the sequencer: Pitch Mode, where the pitch of the note is entered; Time Mode, where the duration of the note is entered; and Playback Mode, where the user selected sequence is played back. 3 additional states were required: Standby Mode, which waits for the user to select the next state; Limbo Mode, which is used to transition between Pitch/Time Mode and Standby Mode; and Playback Finished Mode, which waits for the user to either playback again, or re-enter Pitch/Time data. Transitions between states are made using SW[2:0] on the DE2 and the Enter key on the keyboard.
-M1: Keyboard encoder (module ps2_keyboard)
This module encodes signals from the PS2 keyboard so that they are usable by the main module.

-M2: Pitch decoder (module pitchdecoder)
This module decodes the note pitch entered from the keyboard
-M3: Length decoder (module lengthdecoder)
This module decodes the note length entered from the keyboard
-M4: Next note calculator (module playNextNote)
This module calculates when the next is to played, given the current tempo and length of note
-M5: Frequency calculator (module noteLookup)
This module calculates the frequency of the note, given a note value
-M6: HEX timer calculator (module mydisplaytimer)
This module calculates the elapsed playback time in seconds for the HEX display
-M7: Audio decoder (module toneGen)
This module decodes the frequency value of a note into a PCM stream using lookup tables. It can also modify the audio waveform, volume, and apply distortion depending on settings passed in from the main logic block
-M8: Audio controller (proprietary modules)
This module handles data transmission to and from the audio codec chip.
-M9: LED decoder (module notedecoder)
This module interprets the position of the current note in the sequence and lights up the corresponding LEDR.
-M10: HEX decoder (module SEG7_LUT_8)
This module translates the elapsed time, selected waveform, volume, tempo, and filter settings into decimal values on the HEX displays.
-M11: LCD decoder (module LCD_TEST)
This module translates the current state into ASCII characters to be displayed on the LCD.
-M12: LCD controller (proprietary modules)
This module handles data transmission to and from the LCD display.

Success

Based on our original goals of replicating a Roland TB-303 bass synthesizer and learning to use different types of I/O, we can confidently say this project was a success. Our DE2 based bass synthesizer can demonstrate the ability to store and playback 2 tracks of up to 16 notes length each. The recording of the note is done through a computer keyboard interface which is similar to a piano layout, using the keys <Z S X D C V G B H N J M ,> with the lower row representing the black keys and middle rows representing the white keys. The user can input the duration of each note by pressing key 1-8 on the keyboard representing 1 – 8 beats respectively. Each note can range in frequency from C1 to C4 and can be selected during recording by changing some switches.

The remaining switches on the DE2 are used for Mode selection, Track selection and audio output effects. The standby mode is the default mode, the play mode plays back the selected track, the time mode allows the user to input the note duration of the selected track and the pitch mode allows the user to input the notes on the selected track. Our bass synthesizer can change the volume level between loud and quiet through the flip of a switch. The synthesizer can also output the music using sine, square, saw tooth and triangular waveforms for various audio effects. There is a built in distortion filter which greatly changes the how the notes will sound.

The first 6 Hex LED displays the information about the user selected options from the switch inputs. The last 2 Hex LED displays a clock that displays in decimal format the time that has elapsed since playback began (e.g. it goes from 00 – 09, 09- 10, 59-00). The green LED are status indicators for debugging purposes and the red LED shows the current number of the note that the user is modifying or playing back (e.g. LEDR[15] lighting up means the third note is being modified). The LCD display shows the name of our project Bass Synthesizer and the current mode of operation.

Although our project cannot be compared directly to a finished product like commercial bass synthesizers, we created a proof of concept of a bass synthesizer simulator using a FPGA and DE2 board. Given our lack of certain resources (e.g. input switches, time, memory) we still were able to replicate the essential functionalities, the sequencers, the note modifiers, the audio effects and the volume level. Given what we learnt from working on this project, we could easily built more functionalities into it, such as 16 tracks, 64 note long track length, more audio effects and filters, more audio levels and spanning higher frequencies.

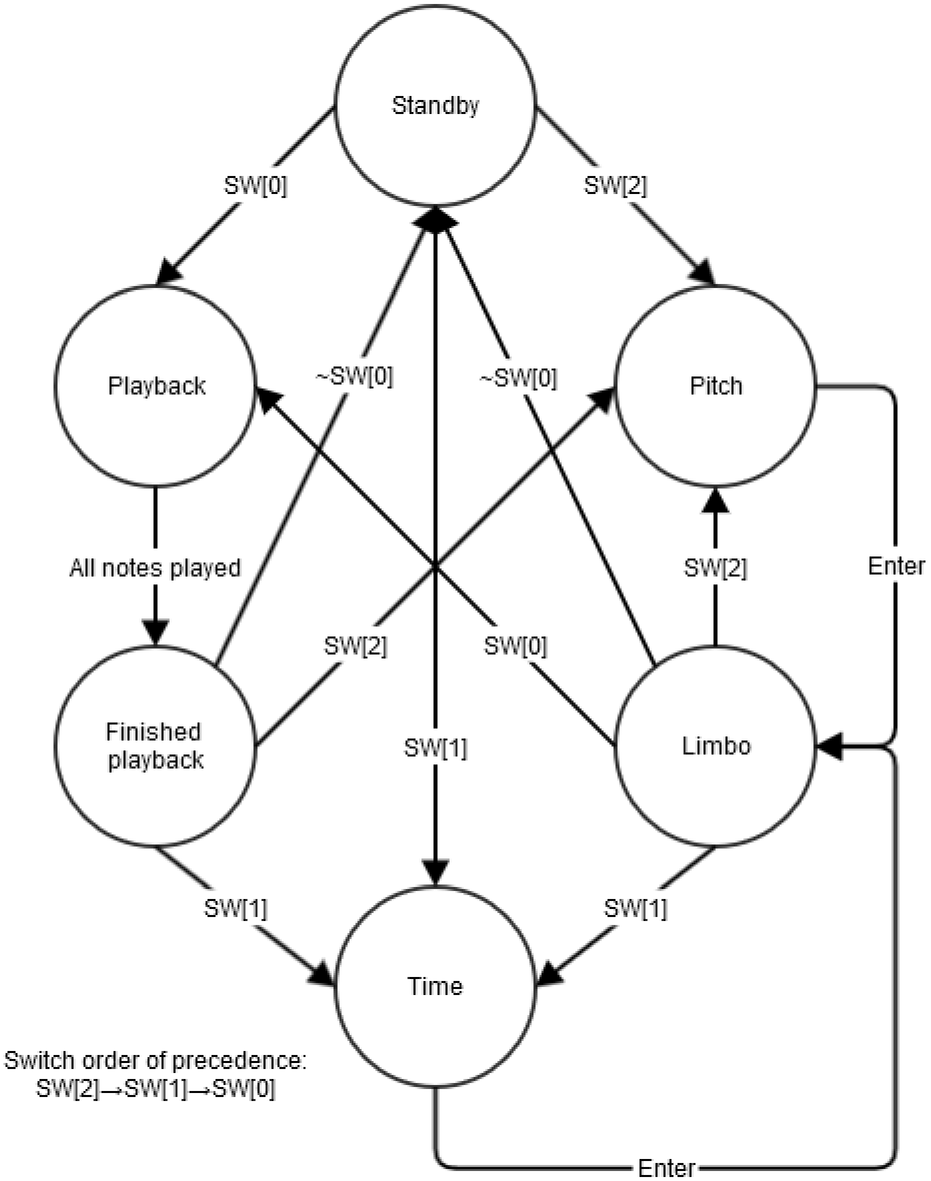
Not only was the end result a success, the journey we took was also bountiful. Throughout the 3 weeks, we learnt how to create sound, manipulate sound, get input from the keyboard and display output on the LCD. The lessons we learnt from interfacing with various types of I/O will surely be useful in the future. When we first started the project all we knew about sound was the theory taught in music and physics class, now we know much more in-depth about how sound is actually stored and reproduced.

Although our project was a success, it isn’t without its fair share of bugs, for the most part we managed to code through the night and tackle them one by one, but there are a couple hard to replicate bugs relating to the initialization of the system and keyboard input. On startup sometimes the DE2 will make a short random sound and mess up the LCD display. The sound quickly stops and the LCD display can be reset by switching states or pressing KEY0 to clear the LCD and audio buffer. Occasionally our state machine would get more than 1 keycode from the Keyboard, this happens on the first note, we suspect this has to do with initialization and buffer not being cleared.

Bass Synthesizer 2.0

If we were given the chance to start all over again one of the thing we would do differently would be the way to integrate our individual work. Victor was responsible for creating modules that dealt with outputting sound, modifying sound and I was responsible for everything else. This division of work was fair but we didn’t collaborate much until the deadline. We had a lot of difficulties trying to merge our code together, we had duplicated audio controller files, multiple instantiations of same modules and sometimes both used the same inout channels. Trying to solve that mess in the end took numerous hours, this problem can be avoided by checking and merging with each other’s code early instead of at the end. We used an approach to coding that is more common in higher level programming, where we would just write modules that are independent of each other and call other modules without giving any afterthought. We learnt that it is much trickier to write larger programs in Verilog because of the intricacies involved with combining modules. The time squeezed out from less debugging related to combining code and less redundant duplicate code would be better spent on learning to use the general purpose I/O pins to interface with an ADC to get input from a potentiometer. If we had additional time we feel it would be worthwhile to explore displaying information using the VGA interface.

Appendix



Switch order of precedence:
SW[2]→SW[1]→SW[0]

Figure 1

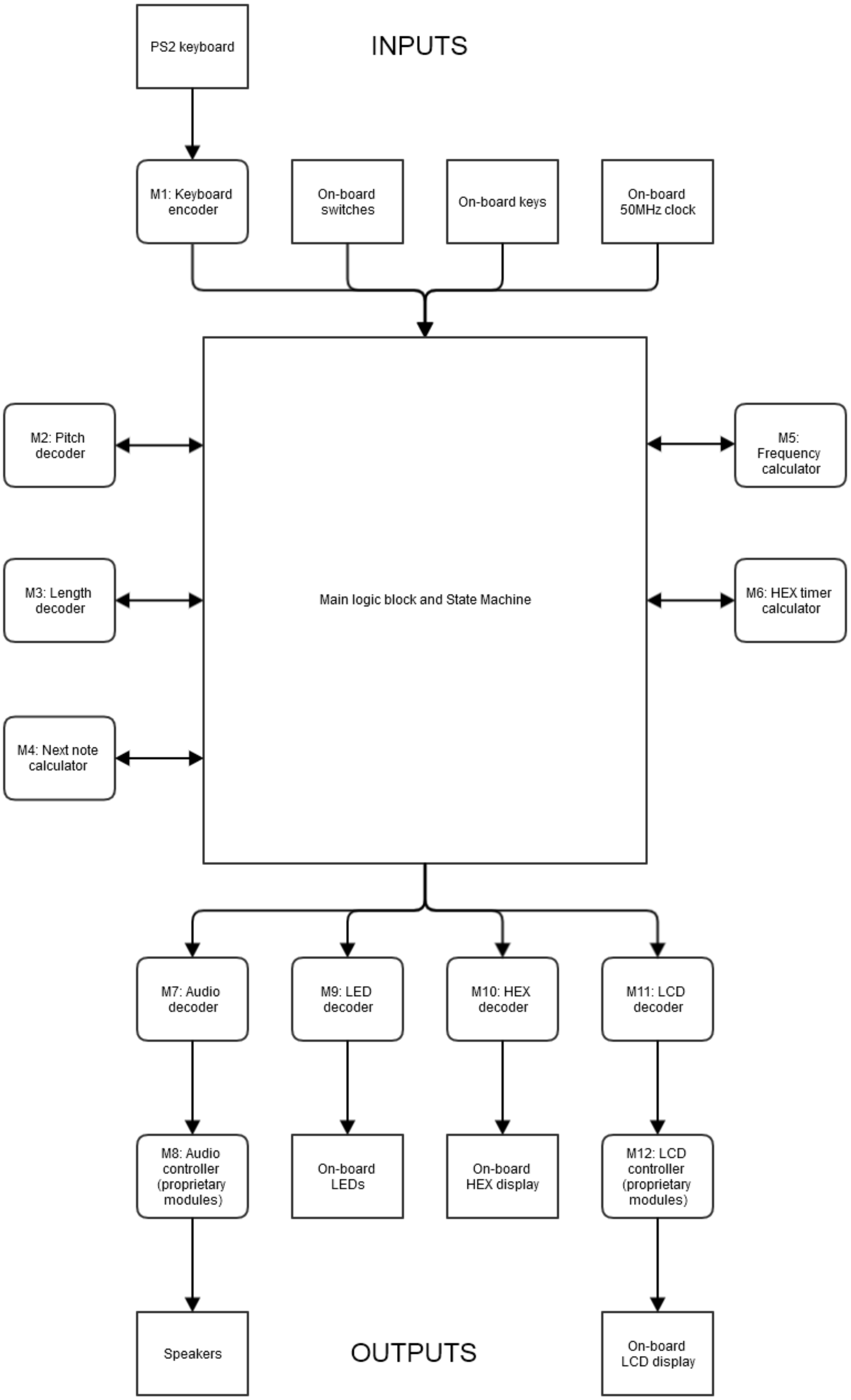

PS2 keyboard
INPUTS
M1: Keyboard encoder
On-board switches
On-board keys
On-board 50MHz clock
M2: Pitch decoder
M5: Frequency calculator
M3: Length decoder
Main logic block and State Machine
M6: HEX timer calculator
M4: Next note calculator
M7: Audio decoder
M9: LED decoder
M10: HEX decoder
M11: LCD decoder
M8: Audio controller (proprietary modules)
On-board LEDs
On-board HEX display
M12: LCD controller (proprietary modules)
Speakers
OUTPUTS
On-board LCD display

Figure 2

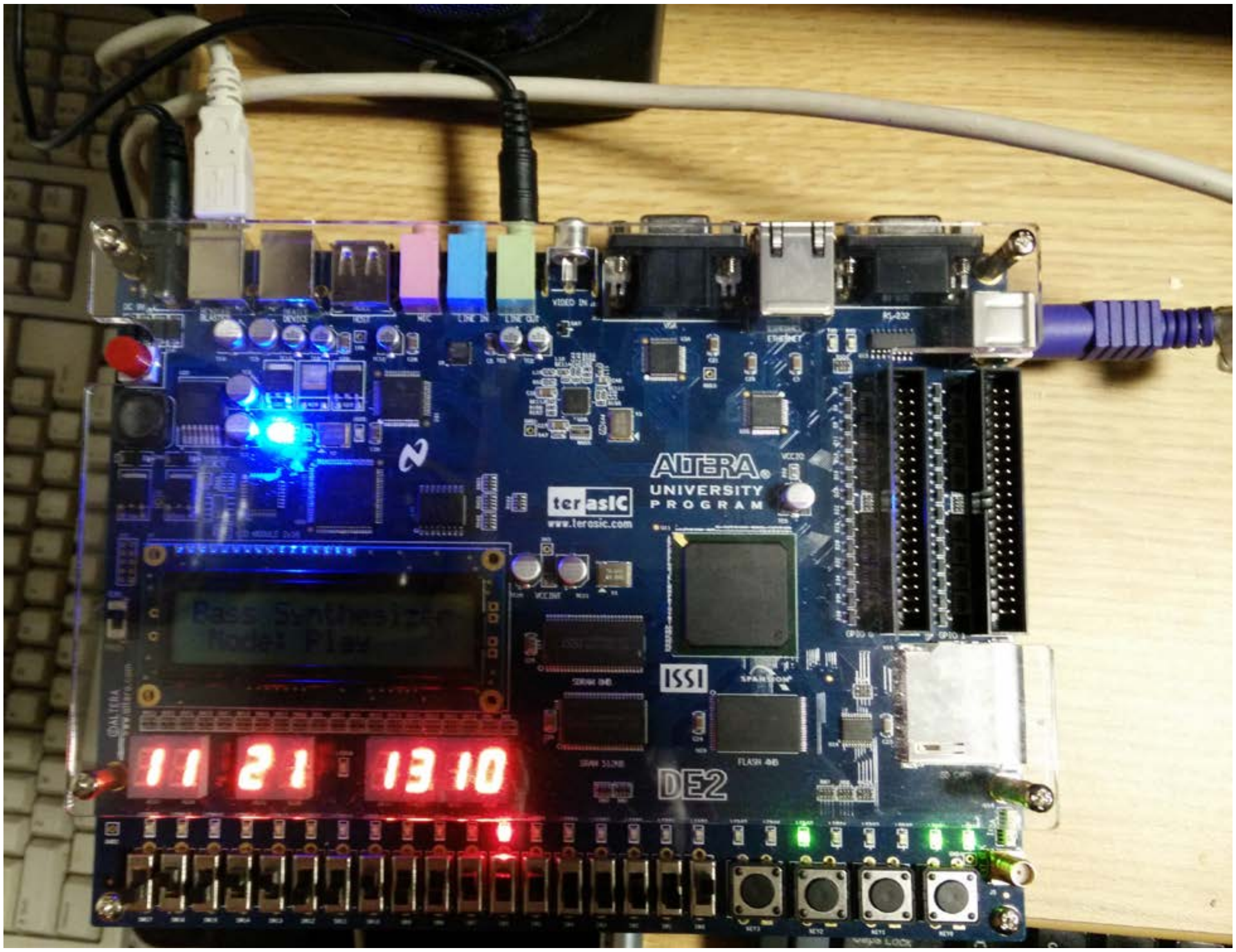


Figure 3

Verilog Code

Only the files we made major changes to are included here, so the audio controller and LCD controller files are not included for length reasons. We also tweaked around with the audio controller and I2C files which allowed the project to compile correctly. The full project folder can be found in the .rar file included in the email.

```
/*

SW[17] ->pitch shift

SW[16] -> filter

SW[15] & SW[14] -> tempo

SW[13] & SW[12] -> waveforms

SW[11] & SW[10] -> Volume

SW[9] & SW[8] & SW[7]-> Track

SW[6] & SW[5] -> Octave
```

```
SW[4] - SW[0] -> Mode selection
LEDR -> Current Note #
LEDG -> State machine display debugging
LCD -> Display name and mode
HEX[7] pitch shift
HEX[6] tempo
HEX[5] waveforms
HEX[4] volume
HEX[3] track
HEX[2] octave
HEX[1] & HEX [0] timer;
KEY[0] resets audio and lcd buffer
*/

//statemachine module

module DE2_synthesizer (

                ///////////////////     Clock Input             ///////////////////
                CLOCK_27,                                                       //      27 MHz
                CLOCK_50,                                                       //      50 MHz
                EXT_CLOCK,                                              //      External Clock
                ///////////////////     Push Button             ///////////////////
                KEY,                                                    //      Button[3:0]
                ///////////////////     DPDT Switch             ///////////////////
                SW,                                             //      DPDT Switch[17:0]
                ///////////////////     7-SEG Dispaly   ///////////////////
                HEX0,                                                   //      Seven Segment Digital 0
                HEX1,                                                   //      Seven Segment Digital 1
                HEX2,                                                   //      Seven Segment Digital 2
                HEX3,                                                   //      Seven Segment Digital 3
                HEX4,                                                   //      Seven Segment Digital 4
                HEX5,                                                   //      Seven Segment Digital 5
                HEX6,                                                   //      Seven Segment Digital 6
                HEX7,                                                   //      Seven Segment Digital 7
                //////////////////////  LED             //////////////////////
                LEDG,                                           //      LED Green[8:0]
                LEDR,                                           //      LED Red[17:0]
                ////////////////////            LCD Module 16X2         ////////////////
                LCD_ON,                                                         //      LCD Power ON/OFF
                LCD_BLON,                                                       //      LCD Back Light ON/OFF
                LCD_RW,                                                         //      LCD Read/Write Select, 0 =
Write, 1 = Read
                LCD_EN,                                                         //      LCD Enable
                LCD_RS,                                                         //      LCD Command/Data Select, 0 =
Command, 1 = Data
                LCD_DATA,                                                       //      LCD Data bus 8 bits

                ////////////////////            I2C             /////////////////////////////
                I2C_SDAT,                                                       //      I2C Data
                I2C_SCLK,                                                       //      I2C Clock
                ////////////////////            PS2             /////////////////////////////
                PS2_DAT,                                                //      PS2 Data
                PS2_CLK,                                                //      PS2 Clock

                ////////////////Audio CODEC             /////////////////////////
                AUD_ADCLRCK,                                            //      Audio CODEC ADC LR Clock
                AUD_ADCDAT,                                                     //      Audio CODEC ADC Data
                AUD_DACLRCK,                                            //      Audio CODEC DAC LR Clock
                AUD_DACDAT,                                                     //      Audio CODEC DAC Data
                AUD_BCLK,                                                       //      Audio CODEC Bit-Stream Clock
                AUD_XCK                                                         //      Audio CODEC Chip Clock

        );

////////////////////////        Clock Input             ////////////////////////
        input                   CLOCK_27;                                               //      27 MHz
        input                   CLOCK_50;                                               //      50 MHz
        input                   EXT_CLOCK;                                      //      External Clock
////////////////////////        Push Button             ////////////////////////
        input    [3:0]  KEY;                                            //      Button[3:0]
////////////////////////        DPDT Switch             ////////////////////////
        input    [17:0] SW;                                     //      DPDT Switch[17:0]
////////////////////////        7-SEG Dispaly   ////////////////////////
        output  [6:0]   HEX0;                                           //      Seven Segment Digital 0
        output  [6:0]   HEX1;                                           //      Seven Segment Digital 1
        output   [6:0]    HEX2;                             //        Seven Segment Digital 2
        output   [6:0]    HEX3;                             //        Seven Segment Digital 3
        output   [6:0]    HEX4;                             //        Seven Segment Digital 4
        output   [6:0]    HEX5;                             //        Seven Segment Digital 5
        output   [6:0]    HEX6;                             //        Seven Segment Digital 6
        output   [6:0]    HEX7;                             //        Seven Segment Digital 7
////////////////////////////    LED             ////////////////////////////
        output   [7:0]    LEDG;                     //        LED Green[8:0]
        output  [17:0]    LEDR;                     //        LED Red[17:0]
////////////////////    LCD Module 16X2    ////////////////////////////
        inout     [7:0]   LCD_DATA;                         //        LCD Data bus 8 bits
        output                    LCD_ON;                           //        LCD Power ON/OFF
        output                    LCD_BLON;                         //        LCD Back Light ON/OFF
        output                    LCD_RW;                           //        LCD Read/Write Select, 0 =
Write, 1 = Read
        output                    LCD_EN;                           //        LCD Enable
        output                    LCD_RS;                           //        LCD Command/Data Select, 0 =
Command, 1 = Data
////////////////////////    I2C             ////////////////////////////////
        inout                       I2C_SDAT;                       //        I2C Data
        output                    I2C_SCLK;                         //        I2C Clock
////////////////////////    PS2             ////////////////////////////////
        inout                       PS2_DAT;                        //        PS2 Data
        input                       PS2_CLK;                        //        PS2 Clock

////////////////////    Audio CODEC             ////////////////////////////
inout                            AUD_ADCLRCK;               //        Audio CODEC ADC LR Clock
        inout                       AUD_DACLRCK;            //        Audio CODEC DAC LR Clock
        input                       AUD_ADCDAT;               //      Audio CODEC ADC Data
        output                    AUD_DACDAT;                       //        Audio CODEC DAC Data
        inout                       AUD_BCLK;                       //        Audio CODEC Bit-Stream Clock
        output                    AUD_XCK;                          //        Audio CODEC Chip Clock

//KEY0 go to restart state;
//one hot encoding
//SW0 playback on
//SW1 pitch mode on
//SW2 length mode on
//all off standby mode;
```

```
//SW0  0 standby state  1 -> not standby
//SW1  0 playback mode  1 -> edit mode
//SW2  0 edit note  1 -> edit length
parameter sstandby = 5'd00000, splay = 5'b00011, srecordnote = 5'b00101, srecordlength = 5'b01001, slimbo = 5'b10000, sendplay = 5'b10011;
/*parameter nc1 = 11'd65, nc2 = 11'd69, nd1 = 11'd73, nd2 = 11'd78, ne = 11'd82, nf1 = 11'd87, nf2 = 11'd92, ng1 = 11'd98, ng2 = 11'd104, na1 = 11'd110,
na2 = 11'd117, nb = 11'd123, nc3 = 11'd131, nn = 11'd1, nl=11'd2;
*/
reg [8:0] t0n0,t0n1,t0n2,t0n3,t0n4,t0n5,t0n6,t0n7,t0n8,t0n9,t0n10,t0n11,t0n12,t0n13,t0n14,t0n15;
reg [8:0] t1n0,t1n1,t1n2,t1n3,t1n4,t1n5,t1n6,t1n7,t1n8,t1n9,t1n10,t1n11,t1n12,t1n13,t1n14,t1n15;
reg [8:0] t2n0,t2n1,t2n2,t2n3,t2n4,t2n5,t2n6,t2n7,t2n8,t2n9,t2n10,t2n11,t2n12,t2n13,t2n14,t2n15;
reg [8:0] t3n0,t3n1,t3n2,t3n3,t3n4,t3n5,t3n6,t3n7,t3n8,t3n9,t3n10,t3n11,t3n12,t3n13,t3n14,t3n15;
reg [8:0] t4n0,t4n1,t4n2,t4n3,t4n4,t4n5,t4n6,t4n7,t4n8,t4n9,t4n10,t4n11,t4n12,t4n13,t4n14,t4n15;
reg [8:0] t5n0,t5n1,t5n2,t5n3,t5n4,t5n5,t5n6,t5n7,t5n8,t5n9,t5n10,t5n11,t5n12,t5n13,t5n14,t5n15;
reg [8:0] t6n0,t6n1,t6n2,t6n3,t6n4,t6n5,t6n6,t6n7,t6n8,t6n9,t6n10,t6n11,t6n12,t6n13,t6n14,t6n15;
reg [8:0] t7n0,t7n1,t7n2,t7n3,t7n4,t7n5,t7n6,t7n7,t7n8,t7n9,t7n10,t7n11,t7n12,t7n13,t7n14,t7n15;
reg [3:0] t0l,t1l,t2l,t3l,t4l,t5l,t6l,t7l,t8l,t9l,t10l,t11l,t12l,t13l,t14l,t15l;
//8:6 length
//5:4 octave
//3:0 pitch
//

reg [4:0] ns;
reg [4:0] cs;
assign LEDG[4:0] = cs;
assign LEDG[5] = cplay;

always @(negedge screenrefresh,negedge enterpress,negedge cplay)
case(cs)
        sstandby:
                        if(SW[2]) ns = srecordnote;
                        else if(SW[1]) ns = srecordlength;
                        else if(SW[0]) ns = splay;
                        else ns = sstandby;
        srecordlength:
                        if(SW[2])
                        ns = srecordnote;
                        else if(SW[1])
                        begin
                        if(~enterpress)ns = slimbo;
                        else ns = srecordlength;
                        end
                        else if(SW[0]) ns = splay;
                        else ns = sstandby;

        splay:
                        if(SW[2]) ns = srecordnote;
                        else if(SW[1]) ns = srecordlength;
                        else if(SW[0])
                        begin
                        if(~cplay) ns = sendplay;
                        else ns = splay;
                        //ns = splay;
                        end
                        else ns = sstandby;

        srecordnote:
                        if(SW[2])
                        begin
                        if(~enterpress)ns = slimbo;
                        else ns = srecordnote;
                        end
                        else if(SW[1]) ns = srecordlength;
                        else if(SW[0]) ns = splay;
                        else ns = sstandby;

        slimbo:
                        if(screenrefresh)ns = slimbo;
                        else
                        begin
                        if(SW[2]) ns = srecordnote;
                        else if(SW[1]) ns = srecordlength;
                        else if(SW[0]) ns = splay;
                        else ns = sstandby;
                        end
        sendplay:
                        if(screenrefresh)ns = sendplay;
                        else
                        begin
                        if(SW[2]) ns = srecordnote;
                        else if(SW[1]) ns = srecordlength;
                        else if(SW[0]) ns = splay;
                        else ns = sstandby;
                        end
        default:
                ns = sstandby;
endcase

//wire tochangestate = (entered & screenrefresh);

reg [4:0] cn;
reg [4:0] playn;
reg [4:0] pitchn;
reg [4:0] lengthn;

always @(cs)
begin
case(cs)
        splay: cn = playn;
        srecordlength:cn= lengthn;
        srecordnote:cn = pitchn;
        default: cn = 5'b00000;
endcase
end

wire [15:0] oneledout;
notedecoder maindisplaycurrentnote(cn,oneledout);

reg [15:0]tled;
assign LEDR[17:2] = tled;
assign LEDR[1:0] = t0n0[1:0];
assign LEDG[7:6] = t0l[1:0];
```

```
//assign LEDR[1:0] = cn[3:2];
//assign LEDG[7:6] = cn[1:0];
always @(CLOCK_50)
begin
case(cs)
        splay: tled = oneledout;
        srecordlength:tled = oneledout;
        srecordnote:tled = oneledout;
        default: tled = 16'h0000;
endcase
end
/*
always @ (negedge key1_on,negedge screenrefresh)
begin
case(cs)
        srecordlength:
                if(~screenrefresh) cn <=0;
                else
                begin
                if(key1_code!=8'h5a)cn <= cn + 1;
                end
        srecordnote:
                if(~screenrefresh) cn <=0;
                else
                begin
                if(key1_code!=8'h5a)cn <= cn + 1;
                end
        default:cn <= 0;
        endcase
end
*/

//wire clearn;
//assign clearn = (~key1_on & screenrefresh);
always @ (negedge key1_on)
begin
case(cs)
        srecordnote:
        begin
                if(key1_code!=8'h5a)pitchn <= pitchn + 1;
                lengthn <= 0;
        end

        srecordlength:
        begin
                if(key1_code!=8'h5a)lengthn <= lengthn + 1;
                pitchn <= 0;
        end

        default:
        begin
        pitchn <= 0;
        lengthn <= 0;
        end
        endcase
end

wire [8:0] tsilence;
assign tsilence = 4'b000001111;
always @(posedge tochangenote)
begin
case(cs)
splay:
playn <= playn +1;

default:
playn <=0;
endcase
end

reg[8:0] currentnote;

always @(posedge tochangenote)
case (cs)
        splay:
        case (SW[9:7])
```

```
3'b000:
        case(playn)
        5'd0:currentnote = t0n0;
        5'd1:currentnote = t0n1;
        5'd2:currentnote = t0n2;
        5'd3:currentnote = t0n3;
        5'd4:currentnote = t0n4;
        5'd5:currentnote = t0n5;
        5'd6:currentnote = t0n6;
        5'd7:currentnote = t0n7;
        5'd8:currentnote = t0n8;
        5'd9:currentnote = t0n9;
        5'd10:currentnote = t0n10;
        5'd11:currentnote = t0n11;
        5'd12:currentnote = t0n12;
        5'd13:currentnote = t0n13;
        5'd14:currentnote = t0n14;
        5'd15:currentnote = t0n15;
        default:currentnote = tsilence;
        endcase
3'b001:
        case(playn)
        5'd0:currentnote = t1n0;
        5'd1:currentnote = t1n1;
        5'd2:currentnote = t1n2;
        5'd3:currentnote = t1n3;
        5'd4:currentnote = t1n4;
        5'd5:currentnote = t1n5;
        5'd6:currentnote = t1n6;
        5'd7:currentnote = t1n7;
        5'd8:currentnote = t1n8;
        5'd9:currentnote = t1n9;
        5'd10:currentnote = t1n10;
        5'd11:currentnote = t1n11;
        5'd12:currentnote = t1n12;
        5'd13:currentnote = t1n13;
        5'd14:currentnote = t1n14;
        5'd15:currentnote = t1n15;
```

```
                default:currentnote = tsilence;
                endcase
        default:currentnote = tsilence;
        endcase
        default:
        currentnote = tsilence;
endcase
wire [13:0] frequency;
noteLookup calculatecurrentfrequency(currentnote[3:0],currentnote[5:4],frequency);

wire tochangenote;
playNextNote whentoplaynextnote(
        CLOCK_50,
        currentnote[8:6],
        SW[15:14],
        tochangenote
);
//assign cplay =1;

always @(playn , t0l,t1l)
        if(cs == splay| cs == sendplay)
        case (SW[9:7])
        3'b000:
                if(playn > t0l)cplay <= 0;
                else cplay <= 1;
        3'b001:
                if(playn > t1l)cplay <= 0;
                else cplay <= 1;
        default: cplay <=1;
        endcase
        else
        cplay <=1;

reg cplay;//0 finished 1 notfinished

        reg [4:0]swreg;
        reg screenrefresh;
        always @(posedge CLOCK_50)
```

```
                begin
                if(swreg != SW[4:0])
                begin
                screenrefresh = 0;
                swreg = SW[4:0];



                end
                else
                begin
                swreg = SW[4:0];
                screenrefresh = 1;
                end
                end



wire [3:0]mnote;


wire [2:0]mlength;
wire  [5:0]mfnote = {SW[6:5],mnote};
lengthdecoder getlengthfromkey(key1_code,mlength);
                pitchdecoder getpitchfromkey(key1_code,mnote,key1_on);
/*
always @ (*)
begin
 t0n0[5:0] = 6'b000000;
 t0n1[5:0] = 6'b000001;
 t0n2[5:0] = 6'b000010;
 t0n3[5:0] = 6'b000011;
 t0n4[5:0] = 6'b000100;
 t0n5[5:0] = 6'b000101;
 t0n6[5:0] = 6'b000110;
 end
*/

always @(posedge key1_on)
begin
case (cs)
```

```
srecordnote:
if(key1_code!=8'h5a)
begin
case (SW[9:7])
3'b000:
        begin
        case (pitchn)
        4'd00: t0n0[5:0] <= mfnote;
        4'd01: t0n1[5:0] <= mfnote;
        4'd02: t0n2[5:0] <= mfnote;
        4'd03: t0n3[5:0] <= mfnote;
        4'd04: t0n4[5:0] <= mfnote;
        4'd05: t0n5[5:0] <= mfnote;
        4'd06: t0n6[5:0] <= mfnote;
        4'd07: t0n7[5:0] <= mfnote;
        4'd08: t0n8[5:0] <= mfnote;
        4'd09: t0n9[5:0] <= mfnote;
        4'd10: t0n10[5:0] <= mfnote;
        4'd11: t0n11[5:0] <= mfnote;
        4'd12: t0n12[5:0] <= mfnote;
        4'd13: t0n13[5:0] <= mfnote;
        4'd14: t0n14[5:0] <= mfnote;
        4'd15: t0n15[5:0] <= mfnote;
        endcase
        t0l <= pitchn;
        end
3'b001:
        begin
        case (pitchn)
        4'd00: t1n0[5:0] <= mfnote;
        4'd01: t1n1[5:0] <= mfnote;
        4'd02: t1n2[5:0] <= mfnote;
        4'd03: t1n3[5:0] <= mfnote;
        4'd04: t1n4[5:0] <= mfnote;
        4'd05: t1n5[5:0] <= mfnote;
        4'd06: t1n6[5:0] <= mfnote;
        4'd07: t1n7[5:0] <= mfnote;
        4'd08: t1n8[5:0] <= mfnote;
        4'd09: t1n9[5:0] <= mfnote;

        4'd10: t1n10[5:0] <= mfnote;

        4'd11: t1n11[5:0] <= mfnote;

        4'd12: t1n12[5:0] <= mfnote;

        4'd13: t1n13[5:0] <= mfnote;

        4'd14: t1n14[5:0] <= mfnote;

        4'd15: t1n15[5:0] <= mfnote;

        endcase

        t1l <= pitchn;

        end

        endcase

end

srecordlength:

if(key1_code!=8'h5a)

begin

case (SW[9:7])

3'b000:

        case (lengthn)

        4'd00: t0n0[8:6] <= mlength;

        4'd01: t0n1[8:6] <= mlength;

        4'd02: t0n2[8:6] <= mlength;

        4'd03: t0n3[8:6] <= mlength;

        4'd04: t0n4[8:6] <= mlength;

        4'd05: t0n5[8:6] <= mlength;

        4'd06: t0n6[8:6] <= mlength;

        4'd07: t0n7[8:6] <= mlength;

        4'd08: t0n8[8:6] <= mlength;

        4'd09: t0n9[8:6] <= mlength;

        4'd10: t0n10[8:6] <= mlength;

        4'd11: t0n11[8:6] <= mlength;

        4'd12: t0n12[8:6] <= mlength;

        4'd13: t0n13[8:6] <= mlength;

        4'd14: t0n14[8:6] <= mlength;

        4'd15: t0n15[8:6] <= mlength;

        endcase

3'b001:

        case (lengthn)

        4'd00: t1n0[8:6] <= mlength;
```

```
                    4'd01: t1n1[8:6] <= mlength;
                    4'd02: t1n2[8:6] <= mlength;
                    4'd03: t1n3[8:6] <= mlength;
                    4'd04: t1n4[8:6] <= mlength;
                    4'd05: t1n5[8:6] <= mlength;
                    4'd06: t1n6[8:6] <= mlength;
                    4'd07: t1n7[8:6] <= mlength;
                    4'd08: t1n8[8:6] <= mlength;
                    4'd09: t1n9[8:6] <= mlength;
                    4'd10: t1n10[8:6] <= mlength;
                    4'd11: t1n11[8:6] <= mlength;
                    4'd12: t1n12[8:6] <= mlength;
                    4'd13: t1n13[8:6] <= mlength;
                    4'd14: t1n14[8:6] <= mlength;
                    4'd15: t1n15[8:6] <= mlength;
                    endcase
                    endcase
          end
          endcase
end

 toneGen soundgenerator(
          // Inputs
          CLOCK_50,
          AUD_ADCDAT,
          KEY,
          SW,
          frequency,

          // Bidirectionals
          AUD_BCLK,
          AUD_ADCLRCK,
          AUD_DACLRCK,
          I2C_SDAT,

          // Outputs
          AUD_XCK,
          AUD_DACDAT,
            I2C_SCLK
);

reg enterpress;
always @ (posedge CLOCK_50)
begin
if(key1_on & key1_code==8'h5a)enterpress = 0;
else enterpress = 1;
end

always @(posedge CLOCK_50)
begin
                        cs <= ns;
end

//  7-SEG

            SEG7_LUT_8                                  u0          (
            HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7,hexoutput);//testkeycounter);//31'h00001112 );

            reg    [31:0]VGA_CLK_o;

            assign   keyboard_sysclk = VGA_CLK_o[12];
            //assign   demo_clock      = VGA_CLK_o[18];
            //assign   VGA_CLK         = VGA_CLK_o[0];

            always @( posedge CLOCK_50 )
  begin
                        VGA_CLK_o <= VGA_CLK_o + 1;
            end


// KeyBoard Scan //

            wire [7:0]scan_code;

            wire get_gate;
wire key1_on;

wire key2_on;

wire [7:0]key1_code;

wire [7:0]key2_code;
wire [30:0] keycounter;
wire [31:0] hexoutput;
wire entered;
/*
assign hexoutput[31:28] = mnote;
assign hexoutput[27:24] = t0n1[3:0];
assign hexoutput[23:20] = t0n2[3:0];
assign hexoutput[19:16] = t0n3[3:0];
assign hexoutput[15:12] = t0n4[3:0];
assign hexoutput[11:8] = t0n5[3:0];
*/

assign hexoutput[29:28] = SW[17:16];
assign hexoutput[31:30] = 2'b00;
assign hexoutput[25:24] = SW[15:14];
assign hexoutput[27:26] = 2'b00;
assign hexoutput[21:20] = SW[13:12];
assign hexoutput[23:22] = 2'b00;
assign hexoutput[17:16] = SW[11:10];
assign hexoutput[19:18] = 2'b00;
assign hexoutput[14:12] = SW[9:7];
assign hexoutput[15] = 1'b0;
assign hexoutput[9:8] = SW[6:5];
assign hexoutput[11:10] = 2'b00;
mydisplaytimer showclock(~SW[0],CLOCK_50,hexoutput[7:0]);
ps2_keyboard keyboard(
   .iCLK_50  (CLOCK_50),
            .ps2_dat  ( PS2_DAT ),                      //ps2bus data
            .ps2_clk  ( PS2_CLK ),                      //ps2bus clk
            .sys_clk  ( keyboard_sysclk ),  //system clock
            .reset    ( KEY[3] ),                //system reset
```

```
        .reset1   ( KEY[2] ),           //keyboard reset
        .scandata ( scan_code ),        //scan code
        .key1_on  ( key1_on ),                  //key1 triger
        .key2_on  ( key2_on ),                  //key2 triger
        .key1_code( key1_code ),        //key1 code
        .key2_code( key2_code ),        //key2 code
                .keycounter(keycounter),
                .entered(entered)
        );

//reg[13:0] frequency;
//assign frequency =14'd0440;

//      LCD

        assign  LCD_ON          =       1'b1;

        assign  LCD_BLON        =       1'b1;


        LCD_TEST                        u5      (

                                                        //      Host Side

                                                        .iCLK   ( CLOCK_50 ),
                                                        .iRST_N (( KEY[0] & screenrefresh)),
                                                        .LCD_MODE(SW[4:0]),

                                                        //      LCD Side

                                                        .LCD_DATA( LCD_DATA ),
                                                        .LCD_RW  ( LCD_RW ),
                                                        .LCD_EN  ( LCD_EN ),
                                                        .LCD_RS  ( LCD_RS )

                                                        );
```

```
Endmodule
module noteLookup (
        note,
        octave,
        frequency
);

input [3:0] note; //0: C, 1: C#, 2: D, 3: D#, 4: E, 5: F, 6: F#, 7: G, 8: G#, 9: A, 10: A#, 11: B, 12: C, default: silence
input [2:0] octave;
output reg[13:0] frequency;

always @ *
        case (octave)
        0: begin //default range C2 -> C3
                case (note)
                0 : frequency = 13'd65;
                1 : frequency = 13'd69;
                2 : frequency = 13'd73;
                3 : frequency = 13'd78;
                4 : frequency = 13'd82;
                5 : frequency = 13'd87;
                6 : frequency = 13'd92;
                7 : frequency = 13'd98;
                8 : frequency = 13'd104;
                9 : frequency = 13'd110;
                10: frequency = 13'd117;
                11: frequency = 13'd123;
                12: frequency = 13'd131;
                default: frequency = 13'd0;
                endcase
        end
        1: begin //transpose up 1 octave C3 -> C4
                case (note)
                0 : frequency = 13'd131;
                1 : frequency = 13'd139;
                2 : frequency = 13'd147;
                3 : frequency = 13'd156;
                4 : frequency = 13'd165;
                5 : frequency = 13'd175;
                6 : frequency = 13'd185;
                7 : frequency = 13'd196;
                8 : frequency = 13'd208;
                9 : frequency = 13'd220;
                10: frequency = 13'd233;
                11: frequency = 13'd247;
                12: frequency = 13'd262;
                default: frequency = 13'd0;
                endcase
end
2: begin //transpose down 1 octave C1 -> C2
                case (note)
                0 : frequency = 13'd33;
                1 : frequency = 13'd35;
                2 : frequency = 13'd37;
                3 : frequency = 13'd39;
                4 : frequency = 13'd41;
                5 : frequency = 13'd44;
                6 : frequency = 13'd46;
                7 : frequency = 13'd49;
                8 : frequency = 13'd52;
                9 : frequency = 13'd55;
                10: frequency = 13'd58;
                11: frequency = 13'd62;
                12: frequency = 13'd65;
                default: frequency = 13'd0;
                endcase
end
default: begin //also goes to default range
                case (note)
                0 : frequency = 13'd65;
                1 : frequency = 13'd69;
                2 : frequency = 13'd73;
                3 : frequency = 13'd78;
                4 : frequency = 13'd82;
                5 : frequency = 13'd87;
                6 : frequency = 13'd92;
                7 : frequency = 13'd98;
                8 : frequency = 13'd104;
                9 : frequency = 13'd110;
                10: frequency = 13'd117;
                11: frequency = 13'd123;
                12: frequency = 13'd131;
                default: frequency = 13'd0;
                endcase
        end
        endcase
endmodule
module lengthdecoder(pressedkey,mlength);
input [7:0]pressedkey;
output reg [2:0] mlength;
always @(pressedkey)
begin
case (pressedkey)
8'h16: mlength <= 3'd00;  //Key 1
8'h1E: mlength  <= 3'd01; //Key 2
8'h26:  mlength  <= 3'd02; //Key 3
8'h25: mlength  <= 3'd03;          //Key 4
8'h2E: mlength  <= 3'd04;          //Key 5
8'h36: mlength <= 3'd05;//Key 6
8'h3D: mlength  <= 3'd06;          //Key 7
8'h3E: mlength  <= 3'd07;          //Key 8
default:  mlength  <= 3'd00;
endcase
end
endmodule module lengthdecoder(pressedkey,mlength);
input [7:0]pressedkey;
output reg [2:0] mlength;
always @(pressedkey)
begin
case (pressedkey)
8'h16: mlength <= 3'd00;  //Key 1
8'h1E: mlength  <= 3'd01; //Key 2
8'h26:  mlength  <= 3'd02; //Key 3
8'h25: mlength  <= 3'd03;          //Key 4
```

```
8'h2E: mlength  <= 3'd04;          //Key 5
8'h36: mlength <= 3'd05;//Key 6
8'h3D: mlength  <= 3'd06;          //Key 7
8'h3E: mlength  <= 3'd07;          //Key 8
default:  mlength  <= 3'd00;
endcase
end
endmodule
module pitchdecoder(pressedkey,mnote,key1_on);
input [7:0]pressedkey;
input key1_on;
output reg [3:0] mnote;
always @(posedge key1_on)
begin
case (pressedkey)
8'h1a: mnote <= 4'd00;  //Key z
8'h1b:mnote <= 4'd01; //Key s
8'h22: mnote <= 4'd02; //Key x
8'h23:mnote <= 4'd03;              //Key d
8'h21:mnote <= 4'd04;              //Key c
8'h2a:      mnote <= 4'd05;//Key v
8'h34:mnote <= 4'd06;              //Key g
8'h32:mnote <= 4'd07;              //Key b
8'h33:mnote <= 4'd08;              //Key h
8'h31:mnote <= 4'd09;              //Key n
8'h3b:      mnote <= 4'd10;//Key j
8'h3a: mnote <= 4'd11; //Key m
8'h41: mnote <= 4'd12; //Key ,
default: mnote <= 4'd13;
endcase
end
endmodule
module ps2_keyboard(
                                                      input       iCLK_50,
                                                      inout   ps2_dat,
                                                      input   ps2_clk,
                                                      input   sys_clk,
                                                      input   reset,
                    input   reset1,
                    output  reg [30:0]keycounter,
                    output  reg[7:0]scandata,
                    output reg key1_on,
                    output reg key2_on,
                    output reg [7:0]key1_code,
                    output reg [7:0]key2_code,
                    output reg entered
                    );



////////////Keyboard Initially/////////
    reg [10:0] MCNT;
    always @(negedge reset or posedge sys_clk)
    begin
        if (!reset)
            MCNT=0;
        else if(MCNT < 500)
            MCNT=MCNT+1;
//      else if ( !is_key && (key1_code!=keycode_o))
//          MCNT=0;
    end

///// sequence generator /////
//  reg     [7:0]   revcnt;`
    wire rev_tr=(MCNT<12)?1:0;

//  always @(posedge rev_tr or posedge ps2_clk)
//  begin
//      if (rev_tr)
//          revcnt=0;
//      else if (revcnt >=10)
//          revcnt=0;
//      else
//          revcnt=revcnt+1;
//  end

//////KeyBoard serial data in /////
```

```
//      reg [9:0]keycode_o;
//      always @(posedge ps2_clk) begin
//              case (revcnt[3:0])
//              1:keycode_o[0]=ps2_dat;
//              2:keycode_o[1]=ps2_dat;
//              3:keycode_o[2]=ps2_dat;
//              4:keycode_o[3]=ps2_dat;
//              5:keycode_o[4]=ps2_dat;
//              6:keycode_o[5]=ps2_dat;
//              7:keycode_o[6]=ps2_dat;
//              8:keycode_o[7]=ps2_dat;
//              endcase
//      end
        wire [7:0]rc=keycode_o;
        wire HOST_ACK=(revcnt==10)?~(rc[7]^rc[6]^rc[5]^rc[4]^rc[3]^rc[2]^rc[1]^rc[0]) :1;
//////////PS2 InOut/////////
//      assign   ps2_dat =(HOST_ACK)?1'bz:1'b0;



///////KeyBoard Scan-Code trigger//////
        reg keyready;
        always @(posedge rev_tr or negedge ps2_clk) begin
                if (rev_tr)
                        keyready=0;
                else if (revcnt[3:0]==10)
                        keyready=1;
                else
                        keyready=0;
  end
////////////////////////////////////Key1-Key2 Output//////////////////////////
        wire is_key=(
                (keycode_o==8'h1a)?1:(  //Key z
                (keycode_o==8'h1b)?1:(  //Key s
                (keycode_o==8'h22)?1:(  //Key x
                (keycode_o==8'h23)?1:(          //Key d
                (keycode_o==8'h21)?1:(          //Key c
                (keycode_o==8'h2a)?1:(          //Key v
                (keycode_o==8'h34)?1:(          //Key g
                (keycode_o==8'h32)?1:(          //Key b
                (keycode_o==8'h33)?1:(          //Key h
                (keycode_o==8'h31)?1:(          //Key n
                (keycode_o==8'h3b)?1:(          //Key j
                (keycode_o==8'h3a)?1:(  //Key m
                (keycode_o==8'h41)?1:(  //Key ,
                (keycode_o==8'h5a)?1:(  //Key Enter
                (keycode_o==8'h29)?1:( //Key space
                (keycode_o==8'h16)?1:( //Key 1
                (keycode_o==8'h1E)?1:( //Key 2
                (keycode_o==8'h26)?1:( //Key 3
                (keycode_o==8'h25)?1:( //Key 4
                (keycode_o==8'h2E)?1:( //Key 5
                (keycode_o==8'h36)?1:( //Key 6
                (keycode_o==8'h3D)?1:( //Key 7
                (keycode_o==8'h3E)?1:0 //Key 8
        //        (keycode_o==8'h35)?1:(
                //(keycode_o==8'h2c)?1:(
                //(keycode_o==8'h24)?1:(
        //        (keycode_o==8'h1d)?1:(
        //        (keycode_o==8'h15)?1:0
                ))))))))))))))))))))))
        );

//////////////key1 & key2 Assign///////////
        wire keyboard_off=((MCNT==200) || (!reset1))?0:1;


        always @(posedge keyready) scandata = keycode_o;
always @(negedge key1_on)
begin
keycounter = keycounter +1;
//if(key1_code==8'h5a) entered = 0;
//else entered = 1;
end

always @(posedge keyready)
begin
if(keycode_o==8'h5a) entered = 0;
```

```
else entered = 1;
end



always @(negedge keyboard_off  or posedge keyready)
begin
        if (!keyboard_off)
        begin
                key1_on=0;
                key2_on=0;
                key1_code=8'hf0;
                key2_code=8'hf0;
        end
        else if (scandata==8'hf0)
        begin
                if (keycode_o==key1_code)
                begin
                        //testkeycounter =testkeycounter +1;

                        key1_on=0;
                        key1_code=8'hf0;
                end
                else if (keycode_o==key2_code)
                begin

                        key2_on=0;
                        key2_code=8'hf0;
                end
        end
        else if (is_key)
        begin
                if ((!key1_on) && (key2_code!=keycode_o))
                begin
                        key1_code=keycode_o;
                        key1_on=1;

                end
                else if ((!key2_on) && (key1_code!=keycode_o))
        begin
            key2_code=keycode_o;
            key2_on=1;

        end
    end
end


reg     ps2_clk_in,ps2_clk_syn1,ps2_dat_in,ps2_dat_syn1;
wire     clk,ps2_dat_syn0,ps2_clk_syn0;
//tristate output control for PS2_DAT and PS2_CLK;
assign ps2_clk_syn0 = ps2_clk;
assign ps2_dat_syn0 = ps2_dat;

//clk division, derive a 97.65625KHz clock from the 50MHz source;
reg [8:0] clk_div;
always@(posedge iCLK_50)
    begin
        clk_div <= clk_div+1;
    end

assign clk = clk_div[8];
//multi-clock region simple synchronization
always@(posedge clk)
begin
    ps2_clk_syn1 <= ps2_clk_syn0;
    ps2_clk_in   <= ps2_clk_syn1;
    ps2_dat_syn1 <= ps2_dat_syn0;
    ps2_dat_in   <= ps2_dat_syn1;
end
reg [7:0]   keycode_o;
reg     [7:0]     revcnt;

always @( posedge ps2_clk_in or negedge keyboard_off)
    begin
        if (!keyboard_off)
            revcnt=0;
        else if (revcnt >=10)
            revcnt=0;
        else
            revcnt=revcnt+1;
    end

always @(posedge ps2_clk_in)
begin
    case (revcnt[3:0])
        1:keycode_o[0]=ps2_dat_in;
        2:keycode_o[1]=ps2_dat_in;
        3:keycode_o[2]=ps2_dat_in;
        4:keycode_o[3]=ps2_dat_in;
        5:keycode_o[4]=ps2_dat_in;
        6:keycode_o[5]=ps2_dat_in;
        7:keycode_o[6]=ps2_dat_in;
        8:keycode_o[7]=ps2_dat_in;
    endcase
end
endmodule
module notedecoder(cn,oneledout);
input [4:0]cn;
output reg[15:0] oneledout;
always @(cn)
begin
case(cn)
    5'd0:oneledout <=16’b1000000000000000;
    5'd1:oneledout <=16’b0100000000000000;
    5'd2:oneledout <=16’b0010000000000000;
    5'd3:oneledout <=16’b0001000000000000;
    5'd4:oneledout <=16’b0000100000000000;
    5'd5:oneledout <=16’b0000010000000000;
    5'd6:oneledout <= 16’b0000001000000000;
    5'd7:oneledout <= 16’b0000000100000000;
    5'd8:oneledout <= 16’b0000000010000000;
    5'd9:oneledout <= 16’b0000000001000000;
    5'd10:oneledout <= 16’b0000000000100000;
    5'd11:oneledout <= 16’b0000000000010000;
```

```
        5'd12:oneledout <= 16'b0000000000001000;
        5'd13:oneledout <= 16'b0000000000000100;
        5'd14:oneledout <= 16'b0000000000000010;
        5'd15:oneledout <= 16'b0000000000000001;
 default:oneledout <= 16'b0000000000000000;
        endcase;
end

endmodule
module mydisplaytimer(myreset,CLOCK_50,timerdisplay);
input myreset;
input CLOCK_50;
output reg[7:0] timerdisplay;
reg [27:0] clockcounter;
always @ (posedge CLOCK_50)
begin
if(myreset)
begin
timerdisplay = 8'h00;
clockcounter = 28'h0000000;
end
else clockcounter = clockcounter + 1;
if(clockcounter == 28'd50000000)
begin
clockcounter = 28'h0000000;
timerdisplay = timerdisplay +1;
if(timerdisplay[3:0]== 4'ha)timerdisplay = timerdisplay + 6;
if(timerdisplay==8'h60)timerdisplay = 8'h00;
end
end

endmodule
//test module used to test audio effects
module DE2_Audio_Example (
        // Inputs
        CLOCK_50,
        AUD_ADCDAT,
        KEY,
        SW,

        // Bidirectionals
        AUD_BCLK,
        AUD_ADCLRCK,
        AUD_DACLRCK,
        I2C_SDAT,

        // Outputs
        AUD_XCK,
        AUD_DACDAT,
        I2C_SCLK,
        LEDR

);

/*****************************************************************************
 *                         Port Declarations                         *
 *****************************************************************************/
// Inputs
input                                   CLOCK_50;
input       [3:0]               KEY;
input       [17:0]      SW;
input                                   AUD_ADCDAT;

// Bidirectionals
inout                                   AUD_BCLK;
inout                                   AUD_ADCLRCK;
inout                                   AUD_DACLRCK;
inout                                   I2C_SDAT;

// Outputs
output                          AUD_XCK;
output                          AUD_DACDAT;
output                          I2C_SCLK;
output [12:0]           LEDR;

// Internal Registers
```

```
/*
reg     [13:0]  frequency;


always @ *
        if (SW[0])
                frequency = 19'd65;
        else if (SW[1])
                frequency = 19'd69;
        else if (SW[2])
                frequency = 19'd73;
        else if (SW[3])
                frequency = 19'd78;
        else if (SW[4])
                frequency = 19'd82;
        else if (SW[5])
                frequency = 19'd87;
        else if (SW[6])
                frequency = 19'd92;
        else if (SW[7])
                frequency = 19'd98;
        else if (SW[8])
                frequency = 19'd104;
        else if (SW[9])
                frequency = 19'd110;
        else if (SW[10])
                frequency = 19'd117;
        else if (SW[11])
                frequency = 19'd123;
        else if (SW[12])
                frequency = 19'd131;
        else
                frequency = 19'd0;
                */


        //geneatingsound module


        toneGen testgensound(
                // Inputs
                CLOCK_50,
                AUD_ADCDAT,
                KEY,
                SW,
                14'd0440,

                // Bidirectionals
                AUD_BCLK,
                AUD_ADCLRCK,
                AUD_DACLRCK,
                I2C_SDAT,

                // Outputs
                AUD_XCK,
                AUD_DACDAT,
                I2C_SCLK
        );

endmodule



module toneGen (
        // Inputs
        CLOCK_50,
        AUD_ADCDAT,
        KEY,
        SW,
        frequency,

        // Bidirectionals
        AUD_BCLK,
        AUD_ADCLRCK,
        AUD_DACLRCK,
        I2C_SDAT,

        // Outputs
        AUD_XCK,
        AUD_DACDAT,
        I2C_SCLK
);


/*****************************************************************************
 *                             Port Declarations                             *
 *****************************************************************************/
// Inputs
input                           CLOCK_50;
input                           AUD_ADCDAT;
input       [0:0]       KEY;
input       [17:0]  SW;
input [13:0]        frequency;


// Bidirectionals
inout                           AUD_BCLK;
inout                           AUD_ADCLRCK;
inout                           AUD_DACLRCK;
inout                           I2C_SDAT;

// Outputs
output                  AUD_XCK;
output                  AUD_DACDAT;
output                  I2C_SCLK;
/*****************************************************************************
 *                 Internal Wires and Registers Declarations                 *
 *****************************************************************************/
// Internal Wires
wire                            audio_in_available;
wire                            read_audio_in;
wire                            audio_out_allowed;
wire        [31:0]  left_channel_audio_out;
wire        [31:0]  right_channel_audio_out;
wire                            write_audio_out;
wire        [1:0]   waveform;
```

```
// Internal Registers

//reg       [18:0]      frequency;
reg         [18:0]      counter;
reg         [18:0]      tick;
reg         [6:0]       tablecount;
reg         [31:0]      sound;
reg         [31:0]      sound2;
reg         [31:0]      sound3;
reg                                             silence;

/*****************************************************************************
 *                      Sequential Logic                      *
 *****************************************************************************/

always @(posedge CLOCK_50)
            if (tick == counter) begin
                        tick <= 0;
                        tablecount <= tablecount + 1;
                        if (tablecount == 64) tablecount <= 0;
            end else tick <= tick + 1;

/*****************************************************************************
 *                      Combinational Logic                      *
 *****************************************************************************/
assign waveform[1:0] = SW[13:12];

//processes special case of frequency = 0 (silence)
always @ *
            if (frequency != 0) begin
                        silence = 0;
                        counter = (50000000/frequency)/64; //64 samples per period
            end else begin
                        silence = 1;
            end

//lookup tables for waveforms
always @ (tablecount)
```

```
if (silence != 0)
        sound = 32'd0;
else begin
        case (waveform)
        2'b00: begin //square wave
                case (tablecount)
                0  :sound = 32'd10000000;
                1  :sound = 32'd10000000;
                2  :sound = 32'd10000000;
                3  :sound = 32'd10000000;
                4  :sound = 32'd10000000;
                5  :sound = 32'd10000000;
                6  :sound = 32'd10000000;
                7  :sound = 32'd10000000;
                8  :sound = 32'd10000000;
                9  :sound = 32'd10000000;
                10 :sound = 32'd10000000;
                11 :sound = 32'd10000000;
                12 :sound = 32'd10000000;
                13 :sound = 32'd10000000;
                14 :sound = 32'd10000000;
                15 :sound = 32'd10000000;
                16 :sound = 32'd10000000;
                17 :sound = 32'd10000000;
                18 :sound = 32'd10000000;
                19 :sound = 32'd10000000;
                20 :sound = 32'd10000000;
                21 :sound = 32'd10000000;
                22 :sound = 32'd10000000;
                23 :sound = 32'd10000000;
                24 :sound = 32'd10000000;
                25 :sound = 32'd10000000;
                26 :sound = 32'd10000000;
                27 :sound = 32'd10000000;
                28 :sound = 32'd10000000;
                29 :sound = 32'd10000000;
                30 :sound = 32'd10000000;
                31 :sound = 32'd10000000;
        32 :sound = -32'd10000000;
        33 :sound = -32'd10000000;
        34 :sound = -32'd10000000;
        35 :sound = -32'd10000000;
        36 :sound = -32'd10000000;
        37 :sound = -32'd10000000;
        38 :sound = -32'd10000000;
        39 :sound = -32'd10000000;
        40 :sound = -32'd10000000;
        41 :sound = -32'd10000000;
        42 :sound = -32'd10000000;
        43 :sound = -32'd10000000;
        44 :sound = -32'd10000000;
        45 :sound = -32'd10000000;
        46 :sound = -32'd10000000;
        47 :sound = -32'd10000000;
        48 :sound = -32'd10000000;
        49 :sound = -32'd10000000;
        50 :sound = -32'd10000000;
        51 :sound = -32'd10000000;
        52 :sound = -32'd10000000;
        53 :sound = -32'd10000000;
        54 :sound = -32'd10000000;
        55 :sound = -32'd10000000;
        56 :sound = -32'd10000000;
        57 :sound = -32'd10000000;
        58 :sound = -32'd10000000;
        59 :sound = -32'd10000000;
        60 :sound = -32'd10000000;
        61 :sound = -32'd10000000;
        62 :sound = -32'd10000000;
        63 :sound = -32'd10000000;
        default: sound = 32'd0;
        endcase
end
2'b01: begin //sawtooth wave
        case (tablecount)
        0  : sound = -32'd17320508;
```

```
1  : sound = -32'd16779242;
2  : sound = -32'd16237976;
3  : sound = -32'd15696710;
4  : sound = -32'd15155444;
5  : sound = -32'd14614178;
6  : sound = -32'd14072912;
7  : sound = -32'd13531646;
8  : sound = -32'd12990381;
9  : sound = -32'd12449115;
10 : sound = -32'd11907849;
11 : sound = -32'd11366583;
12 : sound = -32'd10825317;
13 : sound = -32'd10284051;
14 : sound = -32'd9742785;
15 : sound = -32'd9201519;
16 : sound = -32'd8660254;
17 : sound = -32'd8118988;
18 : sound = -32'd7577722;
19 : sound = -32'd7036456;
20 : sound = -32'd6495190;
21 : sound = -32'd5953924;
22 : sound = -32'd5412658;
23 : sound = -32'd4871392;
24 : sound = -32'd4330127;
25 : sound = -32'd3788861;
26 : sound = -32'd3247595;
27 : sound = -32'd2706329;
28 : sound = -32'd2165063;
29 : sound = -32'd1623797;
30 : sound = -32'd1082531;
31 : sound = -32'd541265;
32 : sound = 32'd0;
33 : sound = 32'd541265;
34 : sound = 32'd1082531;
35 : sound = 32'd1623797;
36 : sound = 32'd2165063;
37 : sound = 32'd2706329;
38 : sound = 32'd3247595;
```

```
        39 : sound = 32'd3788861;
        40 : sound = 32'd4330127;
        41 : sound = 32'd4871392;
        42 : sound = 32'd5412658;
        43 : sound = 32'd5953924;
        44 : sound = 32'd6495190;
        45 : sound = 32'd7036456;
        46 : sound = 32'd7577722;
        47 : sound = 32'd8118988;
        48 : sound = 32'd8660254;
        49 : sound = 32'd9201519;
        50 : sound = 32'd9742785;
        51 : sound = 32'd10284051;
        52 : sound = 32'd10825317;
        53 : sound = 32'd11366583;
        54 : sound = 32'd11907849;
        55 : sound = 32'd12449115;
        56 : sound = 32'd12990381;
        57 : sound = 32'd13531646;
        58 : sound = 32'd14072912;
        59 : sound = 32'd14614178;
        60 : sound = 32'd15155444;
        61 : sound = 32'd15696710;
        62 : sound = 32'd16237976;
        63 : sound = 32'd16779242;
        default: sound = 32'd0;
        endcase
end
2'b10: begin //sine wave
        case (tablecount)
        0  : sound = 32'd0;
        1  : sound = 32'd1386171;
        2  : sound = 32'd2758993;
        3  : sound = 32'd4105245;
        4  : sound = 32'd5411961;
        5  : sound = 32'd6666556;
        6  : sound = 32'd7856949;
        7  : sound = 32'd8971676;
```

```
8  : sound = 32'd10000000;
9  : sound = 32'd10932018;
10 : sound = 32'd11758756;
11 : sound = 32'd12472250;
12 : sound = 32'd13065629;
13 : sound = 32'd13533180;
14 : sound = 32'd13870398;
15 : sound = 32'd14074037;
16 : sound = 32'd14142136;
17 : sound = 32'd14074037;
18 : sound = 32'd13870398;
19 : sound = 32'd13533180;
20 : sound = 32'd13065629;
21 : sound = 32'd12472250;
22 : sound = 32'd11758756;
23 : sound = 32'd10932018;
24 : sound = 32'd10000000;
25 : sound = 32'd8971676;
26 : sound = 32'd7856949;
27 : sound = 32'd6666556;
28 : sound = 32'd5411961;
29 : sound = 32'd4105245;
30 : sound = 32'd2758993;
31 : sound = 32'd1386171;
32 : sound = 32'd0;
33 : sound = -32'd1386171;
34 : sound = -32'd2758993;
35 : sound = -32'd4105245;
36 : sound = -32'd5411961;
37 : sound = -32'd6666556;
38 : sound = -32'd7856949;
39 : sound = -32'd8971676;
40 : sound = -32'd10000000;
41 : sound = -32'd10932018;
42 : sound = -32'd11758756;
43 : sound = -32'd12472250;
44 : sound = -32'd13065629;
45 : sound = -32'd13533180;
```

```
        46 : sound = -32'd13870398;
        47 : sound = -32'd14074037;
        48 : sound = -32'd14142136;
        49 : sound = -32'd14074037;
        50 : sound = -32'd13870398;
        51 : sound = -32'd13533180;
        52 : sound = -32'd13065629;
        53 : sound = -32'd12472250;
        54 : sound = -32'd11758756;
        55 : sound = -32'd10932018;
        56 : sound = -32'd10000000;
        57 : sound = -32'd8971676;
        58 : sound = -32'd7856949;
        59 : sound = -32'd6666556;
        60 : sound = -32'd5411961;
        61 : sound = -32'd4105245;
        62 : sound = -32'd2758993;
        63 : sound = -32'd1386171;
        default: sound = 32'd0;
        endcase
end
2'b11: begin //triangle wave
        case (tablecount)
        0  : sound = 32'd0;
        1  : sound = 32'd1082531;
        2  : sound = 32'd2165063;
        3  : sound = 32'd3247595;
        4  : sound = 32'd4330127;
        5  : sound = 32'd5412658;
        6  : sound = 32'd6495190;
        7  : sound = 32'd7577722;
        8  : sound = 32'd8660254;
        9  : sound = 32'd9742785;
        10 : sound = 32'd10825317;
        11 : sound = 32'd11907849;
        12 : sound = 32'd12990381;
        13 : sound = 32'd14072912;
        14 : sound = 32'd15155444;
```

```
15 : sound = 32'd16237976;

16 : sound = 32'd17320508;

17 : sound = 32'd16237976;

18 : sound = 32'd15155444;

19 : sound = 32'd14072912;

20 : sound = 32'd12990381;

21 : sound = 32'd11907849;

22 : sound = 32'd10825317;

23 : sound = 32'd9742785;

24 : sound = 32'd8660254;

25 : sound = 32'd7577722;

26 : sound = 32'd6495190;

27 : sound = 32'd5412658;

28 : sound = 32'd4330127;

29 : sound = 32'd3247595;

30 : sound = 32'd2165063;

31 : sound = 32'd1082531;

32 : sound = 32'd0;

33 : sound = -32'd1082531;

34 : sound = -32'd2165063;

35 : sound = -32'd3247595;

36 : sound = -32'd4330127;

37 : sound = -32'd5412658;

38 : sound = -32'd6495190;

39 : sound = -32'd7577722;

40 : sound = -32'd8660254;

41 : sound = -32'd9742785;

42 : sound = -32'd10825317;

43 : sound = -32'd11907849;

44 : sound = -32'd12990381;

45 : sound = -32'd14072912;

46 : sound = -32'd15155444;

47 : sound = -32'd16237976;

48 : sound = -32'd17320508;

49 : sound = -32'd16237976;

50 : sound = -32'd15155444;

51 : sound = -32'd14072912;

52 : sound = -32'd12990381;
```

```
            53 : sound = -32'd11907849;
            54 : sound = -32'd10825317;
            55 : sound = -32'd9742785;
            56 : sound = -32'd8660254;
            57 : sound = -32'd7577722;
            58 : sound = -32'd6495190;
            59 : sound = -32'd5412658;
            60 : sound = -32'd4330127;
            61 : sound = -32'd3247595;
            62 : sound = -32'd2165063;
            63 : sound = -32'd1082531;
            default: sound = 32'd0;
            endcase
        end
        endcase
    end

//distortion effect
always @ *
    if (SW[16]) begin
        if (sound > 32'd9000000)
            sound2 = 32'd9000000;
        else if (sound < -32'd9000000)
            sound2 = -32'd9000000;
    end else
        sound2 = sound;

//volume adjustment
always @ *
    if (!SW[10])
        sound3 = sound2 * 10;
    else
        sound3 = sound2;

//assign LEDR = SW;


assign read_audio_in = audio_in_available & audio_out_allowed;
assign left_channel_audio_out   = sound3;
assign right_channel_audio_out = sound3;
assign write_audio_out = audio_in_available & audio_out_allowed;

/*****************************************************************************
 *               Internal Modules                        *
 *****************************************************************************/

Audio_Controller Audio_Controller (
        // Inputs
        .CLOCK_50                                               (CLOCK_50),
        .reset                                                  (~KEY[0]),

        .clear_audio_in_memory          (),
        .read_audio_in                                  (read_audio_in),

        .clear_audio_out_memory         (),
        .left_channel_audio_out         (left_channel_audio_out),
        .right_channel_audio_out        (right_channel_audio_out),
        .write_audio_out                        (write_audio_out),

        .AUD_ADCDAT                                             (AUD_ADCDAT),

        // Bidirectionals
        .AUD_BCLK                                               (AUD_BCLK),
        .AUD_ADCLRCK                                    (AUD_ADCLRCK),
        .AUD_DACLRCK                                    (AUD_DACLRCK),


        // Outputs
        .audio_in_available             (audio_in_available),
        .left_channel_audio_in          (),
        .right_channel_audio_in         (),
        .audio_out_allowed              (audio_out_allowed),

        .AUD_XCK                                                        (AUD_XCK),
        .AUD_DACDAT                                             (AUD_DACDAT)
);

avconf #(.USE_MIC_INPUT(1)) avc (
        .I2C_SCLK                                       (I2C_SCLK),
        .I2C_SDAT                                       (I2C_SDAT),
        .CLOCK_50                                       (CLOCK_50),
        .reset                                          (~KEY[0])
);

Endmodule
module playNextNote (
        CLOCK_50,
        length,
        tempo,
        playNext
);

input                           CLOCK_50;
input   [2:0]           length;
input   [1:0]           tempo;

output reg              playNext;

reg     [32:0]          counter;
reg     [32:0]          tick;
reg             [3:0]           len;
reg             [7:0]           bpm;

//assigns bpms depending on the tempo input
always @ *
        case (tempo)
        2'b00: bpm = 8'd120;
        2'b01: bpm = 8'd240;
        default: bpm = 8'd240;
        endcase
always @(bpm)
begin
 counter = 32'd3000000000/bpm;
```

```
end
always @(posedge CLOCK_50)
        if (tick == counter) begin
                //playNext <= 0;
                tick <= 0;
                len <= len + 1;
                if (len == length)  begin
                        len <= 0;
                        playNext <= 1;
                end
                else playNext <= 0;

        end else begin
                tick <= tick + 1;
                playNext <= 0;
        end

endmodule
module SEG7_LUT_8 (     oSEG0,oSEG1,oSEG2,oSEG3,oSEG4,oSEG5,oSEG6,oSEG7,iDIG );
input   [31:0]  iDIG;
output  [6:0]   oSEG0,oSEG1,oSEG2,oSEG3,oSEG4,oSEG5,oSEG6,oSEG7;

SEG7_LUT        u0      (       oSEG0,iDIG[3:0]                 );
SEG7_LUT        u1      (       oSEG1,iDIG[7:4]                 );
SEG7_LUT        u2      (       oSEG2,iDIG[11:8]        );
SEG7_LUT        u3      (       oSEG3,iDIG[15:12]       );
SEG7_LUT        u4      (       oSEG4,iDIG[19:16]       );
SEG7_LUT        u5      (       oSEG5,iDIG[23:20]       );
SEG7_LUT        u6      (       oSEG6,iDIG[27:24]       );
SEG7_LUT        u7      (       oSEG7,iDIG[31:28]       );

endmodule
module hextodec (iDIG);

input   [31:0]  iDIG;

endmodule
```

```
module SEG7_LUT         (           oSEG,iDIG               );
input       [3:0]       iDIG;
output      [6:0]       oSEG;
reg                     [6:0]       oSEG;

always @(iDIG)
begin
                        case(iDIG)
                        4'h1: oSEG = 7'b1111001;    // ---t----
                        4'h2: oSEG = 7'b0100100;    // |        |
                        4'h3: oSEG = 7'b0110000;    // lt      rt
                        4'h4: oSEG = 7'b0011001;    // |        |
                        4'h5: oSEG = 7'b0010010;    // ---m----
                        4'h6: oSEG = 7'b0000010;    // |        |
                        4'h7: oSEG = 7'b1111000;    // lb      rb
                        4'h8: oSEG = 7'b0000000;    // |        |
                        4'h9: oSEG = 7'b0011000;    // ---b----
                        4'ha: oSEG = 7'b0001000;
                        4'hb: oSEG = 7'b0000011;
                        4'hc: oSEG = 7'b1000110;
                        4'hd: oSEG = 7'b0100001;
                        4'he: oSEG = 7'b0000110;
                        4'hf: oSEG = 7'b0001110;
                        4'h0: oSEG = 7'b1000000;
                        endcase
end

endmodule
module      LCD_TEST (              //          Host Side
                                                            iCLK,iRST_N,LCD_MODE,
                                                            //          LCD Side
                                                            LCD_DATA,LCD_RW,LCD_EN,LCD_RS);
//          Host Side
input                               iCLK,iRST_N;
input                   [4:0]LCD_MODE;
//          LCD Side
output      [7:0]       LCD_DATA;
output                              LCD_RW,LCD_EN,LCD_RS;
```

```
//          Internal Wires/Registers
reg         [5:0]       LUT_INDEX;
reg         [8:0]       LUT_DATA;
reg         [5:0]       mLCD_ST;
reg         [17:0]      mDLY;
reg                                 mLCD_Start;
reg         [7:0]       mLCD_DATA;
reg                                 mLCD_RS;
wire                    mLCD_Done;

parameter   LCD_INTIAL              =           0;
parameter   LCD_LINE1               =           5;
parameter   LCD_CH_LINE             =           LCD_LINE1+16;
parameter   LCD_LINE2               =           LCD_LINE1+16+1;
parameter   LUT_SIZE                =           LCD_LINE1+32+1;

//parameter standbym = 4'd0000,playm = 4'd0100,recordlengthm = 4'd0010, recordpitchm = 4'd0011;

always@(posedge iCLK or negedge iRST_N)
begin
            if(!iRST_N)
            begin
                        LUT_INDEX               <=          0;
                        mLCD_ST                 <=          0;
                        mDLY                    <=          0;
                        mLCD_Start              <=          0;
                        mLCD_DATA               <=          0;
                        mLCD_RS                             <=          0;
            end
            else
            begin
                        if(LUT_INDEX<LUT_SIZE)
                        begin
                                    case(mLCD_ST)
                                    0:          begin
                                                            mLCD_DATA               <=          LUT_DATA[7:0];
                                                            mLCD_RS                             <=          LUT_DATA[8];
                                                            mLCD_Start              <=          1;
```

```
                                        mLCD_ST         <=      1;
                                end
                        1:      begin
                                        if(mLCD_Done)
                                        begin
                                                mLCD_Start      <=      0;
                                                mLCD_ST         <=      2;

                                        end
                                end
                        2:      begin
                                        if(mDLY<18'h3FFFE)
                                        mDLY    <=      mDLY+1;
                                        else
                                        begin
                                                mDLY    <=      0;
                                                mLCD_ST<=       3;
                                        end
                                end
                        3:      begin
                                        LUT_INDEX       <=      LUT_INDEX+1;
                                        mLCD_ST<=       0;
                                end
                        endcase
                end
        end
end

always
begin
        begin
        case(LUT_INDEX)
        //      Initial
        LCD_INTIAL+0:   LUT_DATA        <=      9'h038;
        LCD_INTIAL+1:   LUT_DATA        <=      9'h00C;
        LCD_INTIAL+2:   LUT_DATA        <=      9'h001;
        LCD_INTIAL+3:   LUT_DATA        <=      9'h006;
        LCD_INTIAL+4:   LUT_DATA        <=      9'h080;
        //          Line 1
        LCD_LINE1+0:     LUT_DATA        <=      9'h142;//B
        LCD_LINE1+1:     LUT_DATA        <=      9'h161;//a
        LCD_LINE1+2:     LUT_DATA        <=      9'h173;//s
        LCD_LINE1+3:     LUT_DATA        <=      9'h173;//s
        LCD_LINE1+4:     LUT_DATA        <=      9'h120;//
        LCD_LINE1+5:     LUT_DATA        <=      9'h153;//S
        LCD_LINE1+6:     LUT_DATA        <=      9'h179;//y
        LCD_LINE1+7:     LUT_DATA        <=      9'h16E;//n
        LCD_LINE1+8:     LUT_DATA        <=      9'h174;//t
        LCD_LINE1+9:     LUT_DATA        <=      9'h168;//h
        LCD_LINE1+10:    LUT_DATA        <=      9'h165;//e
        LCD_LINE1+11:    LUT_DATA        <=      9'h173;//s
        LCD_LINE1+12:    LUT_DATA        <=      9'h169;//i
        LCD_LINE1+13:    LUT_DATA        <=      9'h17A;//z
        LCD_LINE1+14:    LUT_DATA        <=      9'h165;//e
        LCD_LINE1+15:    LUT_DATA        <=      9'h172;//r
        //LCD_LINE1+15:  LUT_DATA        <=      9'h120;//
        //          Change Line
        LCD_CH_LINE:     LUT_DATA        <=      9'h0C0;
        //          Line 2
        LCD_LINE2+0:     LUT_DATA        <=      9'h120;


        //
        LCD_LINE2+1:     LUT_DATA        <=      9'h14D;//M
        LCD_LINE2+2:     LUT_DATA        <=      9'h16F;//o
        LCD_LINE2+3:     LUT_DATA        <=      9'h164;//d
        LCD_LINE2+4:     LUT_DATA        <=      9'h165;//e
        LCD_LINE2+5:     LUT_DATA        <=      9'h13A;//:
        LCD_LINE2+6:     LUT_DATA        <=      9'h120;//
        LCD_LINE2+7:     LUT_DATA        <=      ((LCD_MODE[2])?9'h150:((LCD_MODE[1])?
9'h154:((LCD_MODE[0])?9'h150:9'h153)));//S P T P
        LCD_LINE2+8:     LUT_DATA        <=      ((LCD_MODE[2])?9'h169:((LCD_MODE[1])?
9'h169:((LCD_MODE[0])?9'h16C:9'h174)));//t l i i
        LCD_LINE2+9:     LUT_DATA        <=      ((LCD_MODE[2])?9'h174:((LCD_MODE[1])?
9'h16D:((LCD_MODE[0])?9'h161:9'h161)));//a a m t
        LCD_LINE2+10:    LUT_DATA        <=      ((LCD_MODE[2])?9'h163:((LCD_MODE[1])?
9'h165:((LCD_MODE[0])?9'h179:9'h16E)));//n y e c
        LCD_LINE2+11:    LUT_DATA        <=      ((LCD_MODE[2])?9'h168:((LCD_MODE[1])?
9'h120:((LCD_MODE[0])?9'h120:9'h164)));//d     h
```

```
        LCD_LINE2+12:   LUT_DATA        <=      ((LCD_MODE[2])?9'h120:((LCD_MODE[1])?
9'h120:((LCD_MODE[0])?9'h120:9'h162)));//b

        LCD_LINE2+13:   LUT_DATA        <=      ((LCD_MODE[2])?9'h120:((LCD_MODE[1])?
9'h120:((LCD_MODE[0])?9'h120:9'h179)));//y

        LCD_LINE2+14:   LUT_DATA        <=      9'h120;//

        LCD_LINE2+15:   LUT_DATA        <=      9'h120;//

        default:        LUT_DATA        <=      9'hxxx;

        endcase


        end

end

/*

 is_key=(

                (keycode_o==8'h1a)?1:(  //Key z

                (keycode_o==8'h1b)?1:(

                */

LCD_Controller          u0      (       //      Host Side

                                                .iDATA(mLCD_DATA),

                                                .iRS(mLCD_RS),

                                                .iStart(mLCD_Start),

                                                .oDone(mLCD_Done),

                                                .iCLK(iCLK),

                                                .iRST_N(iRST_N),

                                                //      LCD Interface

                                                .LCD_DATA(LCD_DATA),

                                                .LCD_RW(LCD_RW),

                                                .LCD_EN(LCD_EN),

                                                .LCD_RS(LCD_RS)   );


Endmodule
```