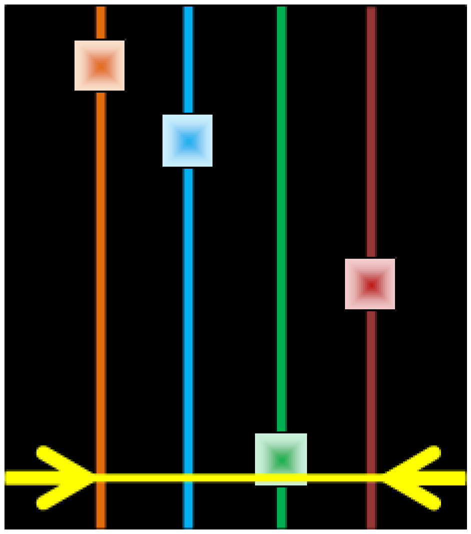
Alexandre Amert
Einar Magnússon

Final Report – Rhythm Game - Fall 2009

12/11/2009



David Money Harris – E155

Abstract

The Rhythm Game is a game where the player simulates a guitarist and tries to play a song as well as possible. The goal is to push the correct buttons at the right time in the song. Our project is divided into two parts: the PIC which manages the whole game and the FPGA which displays the interface of the game on a VGA screen. Thanks to four buttons can play different notes when the VGA screen indicates to do it. Our game is totally functional and the player can train himself with four different songs: Zelda, Mario, Star Wars and Scarborough Fair. Our only regret is we did not have time to improve the mechanical part.

Introduction

Rhythm game is widely inspired by the video game *Guitar Hero*. The user can see different notes parading on a VGA screen from the top to the bottom. When the notes arrive on the white bar, the user has to press the button corresponding to the note. When If the player presses the button at the correct moment, a sound will be generated. If the right button is not pressed at the right time, no sound will be played and the yellow line will blink red. Each different note will generate a different sound. Of course the notes are not chosen randomly, they form a song. If the user plays all the notes correctly he will be able to recognize the song. Because we cannot play all the tones with only 4 notes, each note is not really equal to one tone but to a range of tones. The notes will be chosen to represent the tone that is to be played. The player can try different songs of different difficulty. After each song, the percentage of success is displayed on the screen. The implementation on the VGA screen is shown below.

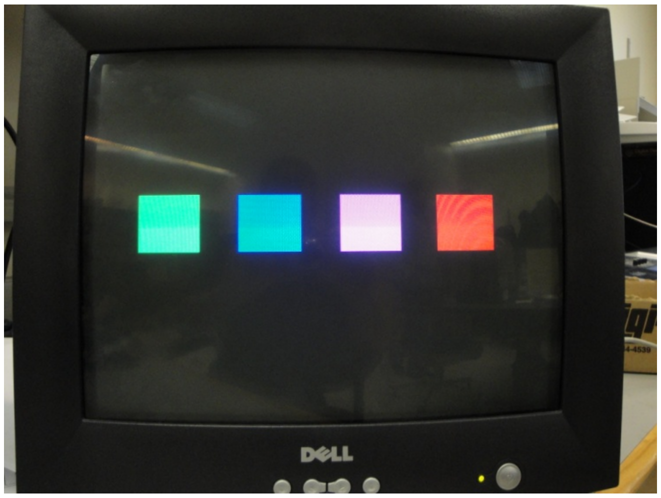


Figure 1 – Start Mode

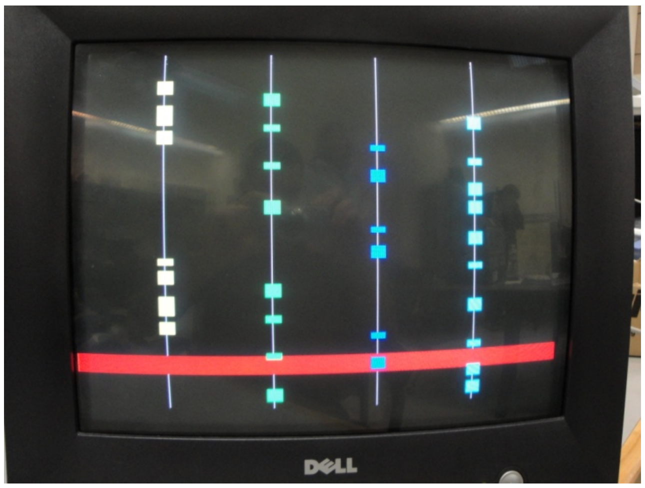


Figure 2 – Game Mode

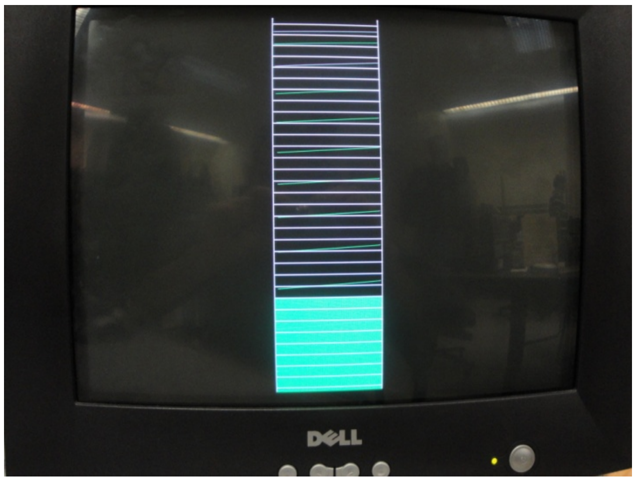


Figure 3 – Score Mode

Our system is composed by two main parts:

- The PIC which manages the whole working of the game, the input buttons, the speaker and sends data to the FPGA to tell it what to display.
- The FPGA which is responsible for displaying the notes, the score and the begin mode as long as the game is run. The FPGA has a parallel connection with the PIC and the screen it is managing is a VGA monitor at a 640x480 resolution.

Figure 4 shows the block diagram of the system:

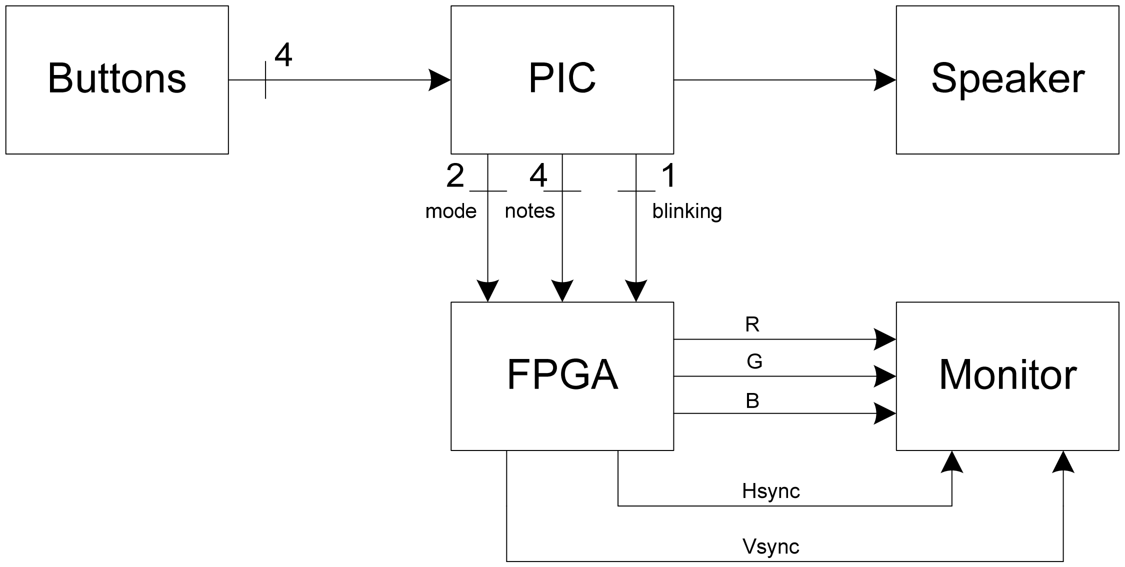
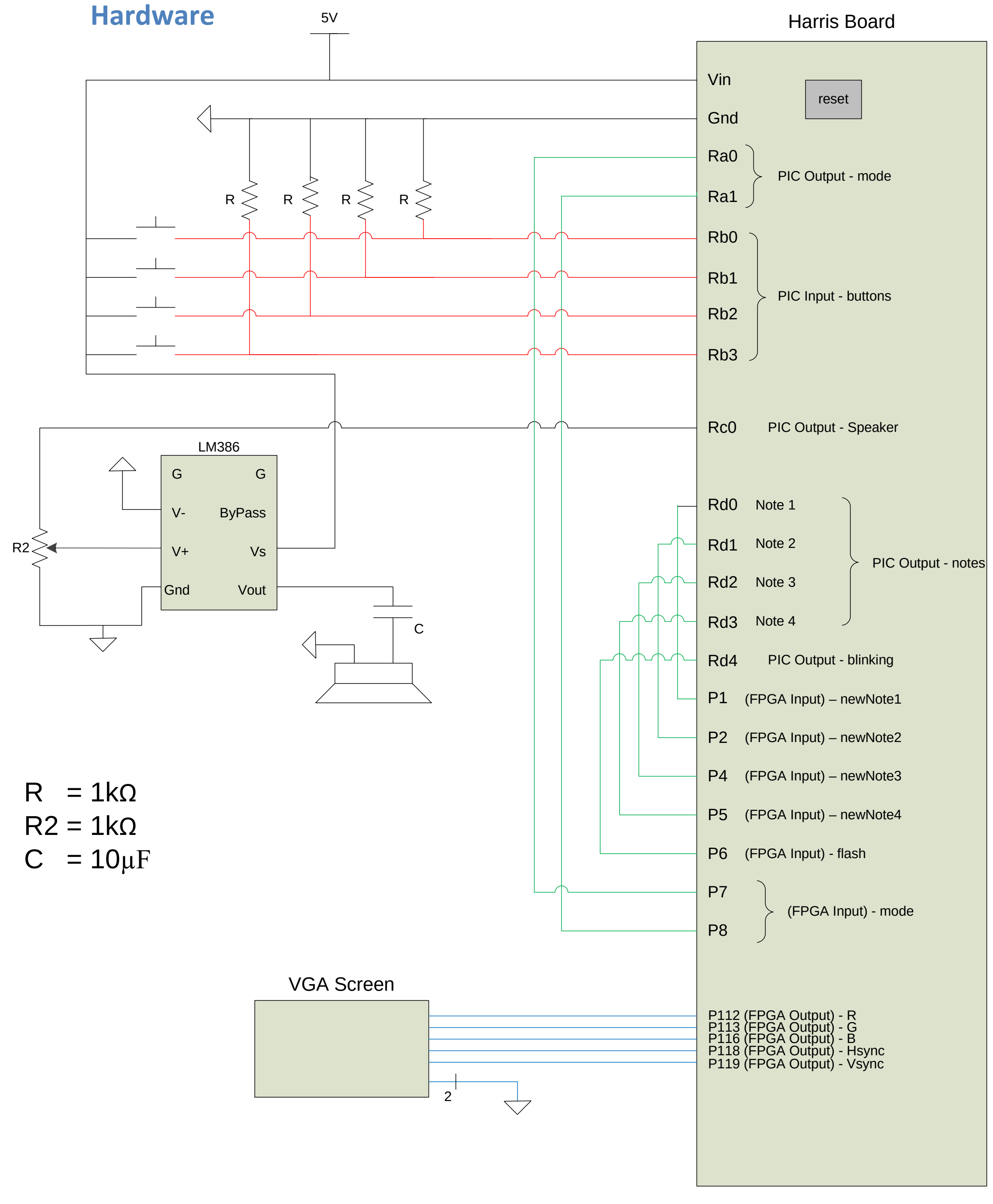


Figure 4 – Block Diagram

PIC

The PIC has to manage the whole game and send the necessary data to the FPGA. It has four inputs (the push buttons) and eight outputs (one for the speaker and seven for the FPGA).

Hardware



R = 1kΩ
R2 = 1kΩ
C = 10µF

Figure 5 – Schematic

Input

The goal is, for each input, to be high when the corresponding button is pressed. To be sure the input will be seen as a zero when the button is not pressed, we added a pull-down resistor. If we remove the resistor and the wire which connects the input to the ground, we would not be sure about the state of the input when it is not connected to the 5V. Different buttons may be pressed at the same time without disturbing the system. Indeed, the user must sometimes press several buttons at the same time if he wants to play the correct note.

Output - Speaker

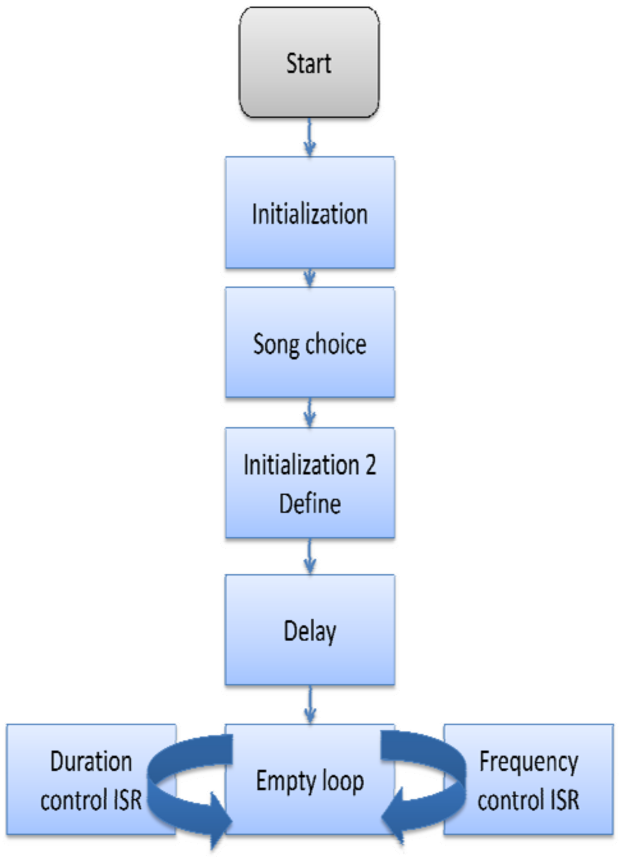
The notes are played on a speaker. Each note is characterized by its frequency and its duration which are stored in the Data Memory. The output is a square signal which varies as a function of the note played. The output current from the PIC is not sufficient for the sound to be audible. That is why we added the audio amplifier which has a gain of twenty. The potentiometer allows the player to regulate the volume. A capacitor is used to make the speaker oscillate around its equilibrium point.

Output – FPGA

The game is divided in three parts: the Start Mode when the user chooses his song (see Figure 1), the Game Mode when the song is running (see Figure 2) and the Score Mode when the player see his percentage of success (see Figure 3). At any time, the FPGA must know which mode the game is in, so we need 2 wires to indicate the current mode. The value of mode is zero at the beginning which represents the start mode. When the user chooses a song, mode turned into 0x01 which is the Game Mode. At the end of the song, when the score is sent, the mode is equal to 0x02.

The five other wires are uses to send the notes and manage the bar blinking. Each time a note has to be displayed, the value of the corresponding button is sent. For instance, if for the next note the user will have to press button 1 and 3 the value sent will be 0x05 which means the state of wire one and wire three will be high while the three others will be low. When the bar has to blink we do not want to resend a new note but we have to modify the value of the last bit. To do that we use an assembly line code which modifies only the desired bit.

```
_asm BSF PORTD, 4, 0 _endasm
```



Software

Overview of the program

The difficulty of our program was to manage at the same time the note to display and the note to play. To do that we used the timers and interrupts.

Details of the functions

Initialization

The main goal of this function is to initialize the PORTs, variables and enable interruptions. The PORTB is used as an input for the buttons whereas the PORTA (mode), the PORTC (speaker) and the PORTD (notes and blink) are outputs.

Song choice

In this part the program waits for player input to choose the song. The player has to press one of the four buttons. If he presses two buttons at the same time, the song number 4 will be selected.

Initialization 2 - Define

When the song is chosen the data from the ROM is stored in the program memory to be used. The function define is necessary to decide which buttons will be used for each note. At the beginning we simply divided in a general way all the tones by the number of combinations of buttons we can press (that is to say 10 – 4 singles and 6 pairs). But because the tone may be really different from a song to another one it did not work correctly. For instance, one song could be interesting to play whereas for another one the user had to play only two different buttons. This is why we finally decided the definitions for the buttons more manually. The most frequent notes are played with only one button and the less common ones need two buttons at the same time. To play the notes, we need two timers: one for the period and one for the duration. So we configure Timer3 with a prescale of 8 so each cycle is about 0.105 seconds. Similarly, we configure Timer1 with a prescale of 4 so that each count is 0.8 µs, or half a period unit.

Check button

This function has a double use. The first one is to ensure that the correct button was pressed. If it did and only if it did, the variable enable will be 1 instead of 0 and the sound of the note will be emitted. The second use is to avoid a parasitical noise to be played when the frequency is zero (pause).

Percent

Because the FPGA and the PIC are connected with only 5 wires for sending data, the score sent by the PIC cannot exceed 2^5-1=31. This is why instead of sending a score, we send the percentage of success times 31:

score_sent = 31*score/number

where number is the total number of notes of the song.

High priority interrupt function

When all initialization is finished, the main code enters an empty while loop. All the work is performed in interrupt service routines. The high priority interrupt function is the one that controls the frequency of the note being played on the speaker. When Timer 1 overflows, the routine is triggered. If enable is 1, the output bit which controls the speaker is toggled with assembly code, and a variable that indicates a successfully pushed button is set. After that, Timer 1 is reloaded with the value 0xFFFF-“note period” so that it keeps overflowing once per half period. Then the function checkbutton is used to check if the user is still holding the right buttons.

Low priority interrupt function

The low priority interrupt function is triggered when Timer 3 overflows. Timer 3 is configured to run a whole cycle in a certain time which is used as a duration unit for the songs. Each time the interrupt is triggered, similar things are done for the notes being played and those being displayed. The function checks to see if the current note has been played/displayed for the right amount of cycles. If not, it increases the cycle counter. If it has, then it steps to the next note and resets the counter. For the notes being displayed, when it steps to a new note the corresponding button value is written to PORT D. For notes being played, it defines which buttons are to be pressed next, and writes a new value to Timer 3 so that it plays the right frequency. Also, if the last button combination was never pushed correctly, the fifth bit of PORT D is set to make the bar go red.

FPGA

The FPGA is used to display the game graphics on a VGA monitor. See appendix for a block diagram visualization.

Inputs

The inputs to the FPGA are the clock, reset, the four bits that create new notes, the bit that flashes the bottom bar and the two bits that control the game mode.

Outputs

The only outputs of the FPGA are the ones that go to the VGA monitor; the horizontal sync signal, the vertical sync signal and three bits for R,G and B.

FPGA function

The FPGA's DCM is used to generate a 25 MHz clock which is needed to send information correctly to the VGA monitor. The module hvsync_generator takes in the 25 MHz clock and generates the horizontal and vertical sync signals. In addition, it outputs the coordinates of the current pixel and a variable inDisplayArea that is set when the current pixel is in the display area of the monitor. The module operates on the usual VGA standard; the horizontal sync signals when the electron beam should go to the beginning of the next line and the vertical sync signals when it should go to the beginning of the first line. A few lines at the top and bottom and pixels at the beginning and end of each line are defined as outside the display area.

The new note signals are received by shift registers. They are 480 bits wide, each bit corresponding to a row. The speed is stepped down inside the shift register modules so that a bit is transferred through the 480 seats in about 13 seconds.

The modules inside makePixel; isString, isBar, isStringXnote, isScoreBar, isScoreBarOutline, isDiffBox look at the coordinates of the current pixel and output a bit which states whether the current pixel is part of said object. The strings and the bar are static, so the modules just check whether the coordinates are within certain limits. The notes, however, are traveling objects. The module isStringXnote compares the current coordinates to the corresponding shift register, and if the bit in the shift register corresponding to the current row is set and the current column is in a certain interval around string X, then the output is set. The way to make a note that is more than one row high is then to input a string of 1's into the shift register.

The module PixelColor then takes as input the outputs of the above modules along with the mode variable and decides based on some priorities what the color of the current pixel is.

The simplicity of the design means that we can use a parallel connection between the PIC and the FPGA.

Results

Our project works. We created a light version of the Guitar Hero Game as expected using a PIC and a FPGA. We did everything we planned to and we even did two things which were not part of the basic requirements: we had the score part which allows the player to know his score and the difficulty of pressing two buttons at the same time.
One of the main difficulties we had was to synchronize the notes traveling on the FPGA and the PIC. Indeed the PIC has to send the new notes to the FPGA and, at the same time, check if the correct button is pressed and play the sound. To fix that problem we used different interrupts.

References

[1] FPGA4Fun's Pong Game, `http://www.fpga4fun.com/PongGame.html`

Parts List

Part	Source	Vendor Part #	Cost
1xSpeaker	Stock room		Supplied by HMC
1xAudio amplifier LM386	Stock room		Supplied by HMC
4xResistors 1kΩ	Stock room		Supplied by HMC
1xPotentiometer 1kΩ	Stock room		Supplied by HMC
4xButtons	Stock room		Supplied by HMC
1xCapacitor 10µF	Stock room		Supplied by HMC
1xVGA monitor	Stock room		Supplied by HMC

Appendices

FPGA Block Diagram

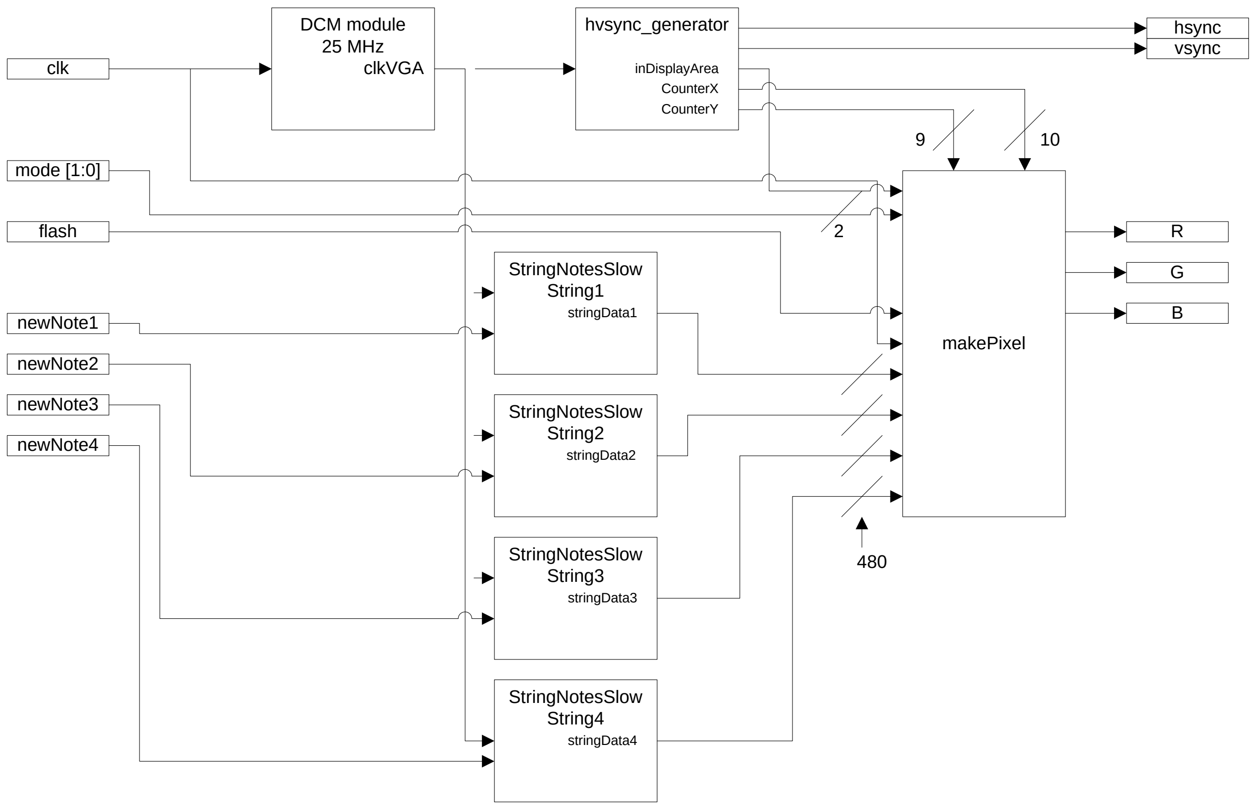


Figure 6: Block diagram of the top module

C code and Verilog code

```
/************************************ Final Project ***********************************/
/* final.c                                                                            */
/* Alexandre_Amert@hmc.edu                       last update:   Tuesday, 08 December  */
/* Einar_Magnusson@HMC.Edu                                                            */
/**************************************************************************************/


// Use the 18F452 PIC
#include <p18f452.h>
// Use the usart and stdio library
#include <stdio.h>
#include <timers.h>
#include <delays.h>

//Functions
void main(void);
void checkb(int);
int define(int);
int percent(int);


void lowisr(void);
void highisr(void);

// Variables
int note,duree,mode,song,init,i,j,button,enable,enable2,tempo,k,i32, startsound, countdurVGA,
countdurSPK, displaying,pushed,score,number;
int delay =1255;
#pragma udata sectionname1
int durat[70]; //this receives the durations of the notes of the selected song
#pragma udata sectionname2
int freq[70];  //this receives the frequencies of the notes of the selected song
#pragma udata sectionname3
int butto[70]; //this recieves the button combinations of the notes selected
int SWb[33];
rom int SWf[32] = {    //Frequencies of the notes of the Star Wars song
0x37E,
0x37E,
0x37E,
0x29E,
0x1BF,
0x1F6,
0x213,
0x255,
0x14F,
0x1bf,
0x1F6,
0x213,
0x255,
0x14F,
0x1bf,
0x000,
0x37E,
0x37E,
0x37E,
0x29E,
0x1BF,
0x1F6,
0x213,
0x255,
0x14F,
0x1bf,
0x1F6,
0x213,
0x255,
0x14F,
0x1bf,
0x14F
};
```

```
rom int SWd[32] = {    //Durations of the notes of the Star Wars song
1,
1,
1,
2,
2,
1,
1,
1,
2,
1,
1,
1,
1,
2,
1,
1,
1,
1,
1,
2,
2,
1,
1,
1,
2,
1,
1,
1,
1,
2,
2,
2
};

#pragma udata section5
rom int MAf[46] = {    //Frequencies of the notes of the Mario song
0x1da,
0x1da,
0x1da,
0x255,
0x1da,
0x18e,
0x000,
0x4AB,
0x000,
0x255,
0x31d,
0x000,
0x3b4,
0x000,
0x26c,
0x278,
0x29e,
0x26c,
0x000,
0x31d,
0x1da,
0x18e,
0x163,
0x1da,
0x18e,
0x1da,
0x255,
0x1da,
0x278,
0x255,
0x31d,
0x000,
0x3b4,
0x000,
0x26c,
0x278,
0x29e,
0x26c,
0x000,
0x31d,
0x1da,
0x18e,
0x163,
0x1da,
0x18e,
0x1da
};

#pragma udata section6
rom int MAd[46] = {    //Durations of the notes of Mario song
1,
2,
2,
1,
2,
2,
1,
2,
2,
1,
1,
2,
2,
1,
2,
2,
1,
1,
1,
1,
2,
1,
2,
1,
2,
2,
1,
1,
2,
2,
1,
2,
2,
1,
2,
2,
1,
1,
1,
1,
2,
1,
2,
1,
2,
2
};

rom int ZDf[70]={      //Frequencies of the notes of Zelda song
0x14F,
0x1BE,
0x000,
0x14F,
0x12A,
0x10A,
0x0FB,
0x0DF,
0x000,
0x0DF,
0x0DF,
0x0D3,
0x0BC,
0x0A7,
0x000,
0x0A7,
0x0A7,
0x000,
0x0BC,
0x0D3,
0x0BC,
0x0D3,
0x0DF,
0x000,
0x0DF,
0x0FB,
0x0FB,
0x0DF,
0x0D3,
0x000,
0x0DF,
0x0FB,
0x119,
0x119,
0x0FB,
0x0DF,
0x000,
0x0FB,
0x119,
0x12A,
0x12A,
0x10A,
0x0ED,
0x000,
0x0C7,
0x0DF,
0x0DF,
0x000,
0x0DF,
0x0FB,
0x0FB,
0x0DF,
0x0D3,
0x000,
0x0DF,
0x0FB,
0x119,
0x119,
0x0FB,
0x0DF,
0x000,
0x0FB,
0x119,
0x12A,
0x12A,
0x10A,
0x0ED,
0x000,
0x0C7,
0x0DF
};

rom int ZDd[70]={      //Durations of the notes of Zelda song
4,
4,
1,
1,
1,
1,
1,
4,
1,
2,
2,
1,
1,
4,
1,
2,
1,
1,
1,
1,
2,
1,
4,
1,
4,
2,
1,
1,
4,
1,
2,
2,
2,
1,
1,
4,
1,
2,
2,
2,
1,
1,
4,
1,
4,
8,
4,
1,
4,
2,
1,
1,
4,
1,
2,
2,
2,
1,
1,
4,
1,
2,
2,
2,
1,
1,
4,
1,
4,
8
};

rom int SFf[33]={      //Frequencies of the notes of SF song
0x850,
0x850,
0x58C,
0x58C,
0x768,
0x6FD,
0x768,
0x850,
0x000,
0x58C,
0x4AA,
0x428,
0x4AA,
0x58C,
0x4F1,
0x63A,
0x58C,
0x428,
0x428,
0x4AA,
0x58C,
0x58C,
0x63A,
0x6FD,
0x768,
0x850,
0x58C,
0x63A,
0x6FD,
0x768,
0x850,
0x954,
0x850
};

rom int SFd[33]={      //Frequencies of the notes of SF song
2,
1,
2,
1,
2,
1,
1,
3,
1,
1,
1,
2,
1,
1,
1,
1,
3,
2,
1,
2,
1,
1,
1,
1,
3,
2,
1,
2,
1,
1,
1,
1,
3
};

void main ()
```

```
{
    RCONbits.IPEN=1;            //Enable interrupt priorities
    INTCONbits.GIEH=1;          //Enable high priority interrupts
    INTCONbits.GIEL=1;          //enable low priority

    TRISB = 0xFF;      //PortB Input, button input
    TRISD = 0x00;      //PortD Output, sends notes to FPGA
    TRISC = 0x00;      //PORTC Output, speaker output
    TRISA = 0x00;      //PORTA output, game mode selection (start - game - end)
    PORTD = 0;
    PORTC = 0;
    PORTA = 0;                  //begin in start mode
    enable=0;                   //enable: tells whether the right buttons are being pressed
    startsound=0;               //startsound: when set, the notes of the chosen song are played
    pushed=0;                   //goes high if the right buttons have been pushed during a note
    score=0;                    //counts number of right notes

    i=0;
       song=0;

       while(song==0)
       {
               song=PORTB;            //push button to select song
       }

       PORTA=0x01;                    //enter game mode

       if(song==1)
       {
               number=32;
               for(init=0;init<number;init++)
               {
                       durat[init]=SWd[init]*6;      //scale duration to have suitable difficulty
                       freq[init]=SWf[init];
                       butto[init]=define(SWf[init]);

               }
       }
       else if(song==2)
       {
               number=46;
               for(init=0;init<number;init++)
               {
                       durat[init]=MAd[init]*5; //slowdown the song
                       freq[init]=MAf[init];
                       butto[init]=define(MAf[init]);
               }
       }
       else if(song==4)
       {
               number=33;
               for(init=0;init<number;init++)
               {
                       durat[init]=SFd[init]*5; //slowdown the song
                       freq[init]=SFf[init];
                       butto[init]=define(SFf[init]);
               }
       }
       else
       {
               number=70;
               for(init=0;init<number;init++)
               {
                       durat[init]=ZDd[init]*3; //slowdown the song
                       freq[init]=ZDf[init];
                       butto[init]=define(ZDf[init]);
               }
       }

    /*Timer1 controls the frequency
    Interrupts Off-16 bit mode-instru cycle clk-presc of 4-no external oscil*/
    OpenTimer1( TIMER_INT_ON &
                T1_16BIT_RW &
                T1_SOURCE_INT &
                T1_PS_1_4 &
                T1_OSC1EN_OFF &
                T1_SYNC_EXT_OFF );

    PIR1bits.TMR1IF=0; //clear flag
                                    //(notes are not played at first)
    PIE1bits.TMR1IE=0; //disable Timer1 interrupt for the time being
    IPR1bits.TMR1IP=1; //high priority


    //Timer3 to control the duration of the notes, both on VGA and on speaker
    //one complete cycle of timer is 2048 * 51.2µs
    OpenTimer3(    TIMER_INT_ON &
                T3_16BIT_RW &
                T3_SOURCE_INT &
                T1_PS_1_8 &
                T3_OSC1EN_OFF &
                T3_SYNC_EXT_OFF);

    IPR2bits.TMR3IP=0;        //low priority interrupt Timer3
    PIR2bits.TMR3IF=0;  //clear Timer3 interrupt flag
    PIE2bits.TMR3IE=1;  //enable Timer3 interrupt


    i=0;    //this tells which note is being PLAYED
    j=0;    //this tells which note is being DISPLAYED

    countdurSPK=0;            //counts the duration of notes played on speaker
    countdurVGA=0;            //counts the duration of notes displayed on VGA

    //start by displaying the first note
    displaying=1;             //displaying: when set, the notes are displayed
    PORTD=butto[j];           //output to FPGA notes to be displayed
    WriteTimer3(0);           //start counting duration of notes


    //Einar: 12.75 sec delay:
       //We now use the 40Mhz clk so all the delays are doubled
    k=1;
    while(k<28)
    {
        Delay10KTCYx(delay);
               Delay10KTCYx(delay);
        k++;
    }
    Delay10KTCYx(890);
       Delay10KTCYx(890);

    //after delay: start playing notes on speaker
    startsound=1;                    //Enable the playing of notes
    tempo=0xFFFF-freq[i];     //so that timer1 overflows after one (half) period
    WriteTimer1(tempo);
    PIE1bits.TMR1IE=1;        //enable interrupts for frequency
    button=butto[i];          //button to be pushed
    enable=0;                        //start by assuming that the right buttons are not pushed
       while(1){}                           //endless loop: the stuff happens in the interrupts

} //main

void checkb(int but)        //check if the correct button is pressed
{
    if(PORTB==but & but!=0)   //allows to play the note only if the corrected button is pressed
    {                                       //and the frequency is not 0
        enable=1;
    }
    else
        enable=0;
}//checkb
```

```
int define(int frequency)    //Defines which buttons should be pushed
{                                                    //depending on the frequency
    if (frequency==0)
               return 0x00;
        else if(frequency==0x1DA){
        return 0x01;
        }
    else if(frequency==0x255 || frequency==0x26C){
        return 0x8;
        }
    else if(frequency==0x4AB){
        return 0x05;
    }
    else if(frequency==0x18E || frequency==0x29E){
        return 0x02;
    }
    else if(frequency==0x31D || frequency==0x278){
        return 0x04;
        }
    else if(frequency==0x3B4){
        return 0xA;
        }
    else if(frequency==0x163){
        return 0x09;
        }
//Zelda
        else if(frequency==0x14F){
        return 0x06;
        }
    else if(frequency==0x12A || frequency==0x0D3 || frequency==0x0C7){
        return 0x2;
        }
    else if(frequency==0x1BE){
        return 0x09;
    }
    else if(frequency==0x0FB){
        return 0x04;
    }
    else if(frequency==0x0BC || frequency==0x0A7 || frequency==0x119 ||frequency==0x0ED){
        return 0x08;
        }
    else if(frequency==0x10A){
        return 0xA;
        }
    else if(frequency==0x0DF){
        return 0x01;
        }
//SF
        else if(frequency==0x850){
        return 0x01;
        }
    else if(frequency==0x768 || frequency==0x428){
        return 0x4;
        }
    else if(frequency==0x58C){
        return 0x02;
    }
    else if(frequency==0x800){
        return 0x05;
    }
    else if(frequency==0x6FD || frequency==0x4AA || frequency==0x63A){
        return 0x08;
        }
    else if(frequency==0x4F1){
        return 0x3;
        }
    else if(frequency==0x954){
        return 0x0A;
        }
//SW
```

```
        else if(frequency==0x37E){
        return 0x01;
        }
    else if(frequency==0x1F6){
        return 0x8;
        }
    else if(frequency==0x1BF){
        return 0x02;
    }
    else if(frequency==0x213){
        return 0x01;
    }
    else if(frequency==0x14F || frequency==0x29E){
        return 0x04;
        }
    else{
               return 0x0C;
    }
}//define





int percent(int score)
{
        int score_sent;
        score_sent = score=31*score/number;
        mode = score_sent;
        return score_sent;
}


#pragma code highinterruptvector = 0x08
void highinterruptvector(void)
{
    _asm goto highisr _endasm
}
#pragma code

#pragma code lowinterruptvector =0x18
void lowinterruptvector(void)
{
    _asm goto lowisr _endasm
}
#pragma code

#pragma interrupt highisr     //interrupt routine for toggling speaker output
void highisr(void)                    //- generate frequency
{
    if(enable)    //toggle speaker output using assembly, probably the fastest way
    {
        _asm BTG PORTC, 0,0 _endasm          //toggle the speaker output bit
        pushed=1;                            //the right combination has been pushed
    }

    checkb(button);           //check if the right buttons are still pushed

    WriteTimer1(tempo);                            //"reset" timer1
    PIR1bits.TMR1IF=0;                             //clear interrupt flag
}

#pragma interrupt lowisr    //interrupt routine
void lowisr(void)
{

    if(startsound)                    //if sound should be played
    {
        countdurSPK++;                //Timer3 has counted one cycle
        if(countdurSPK==durat[i])     //if right number of cycles for current note being played
        {
```

```
        i++;                            //next note
        if(i==number)                   //if song is finished
        {
                       PORTA=2    ;     //enter end mode
            PIE1bits.TMR1IE=0;          //turn off both interrupts
            PIE2bits.TMR3IE=0;
            PIR2bits.TMR3IF=0;
            PIR1bits.TMR1IF=0;
                          PORTD=percent(score);  //send score to FPGA

        }
        else
        {
            if(!pushed)                //if the right buttons were not pushed
            {                                               //-during the last note
                _asm BSF PORTD, 4, 0 _endasm    //turn bar red
            }
            else
            {
                  _asm BCF PORTD, 4, 0 _endasm  //turn bar white
                  if(button!=0)
                          score++       //increase score if the right button is pushed
            }
            pushed=0;                 //reset pushed-variable for next note
                          button=butto[i];         //new button to be played
                          tempo=0xFFFF-freq[i];    //new frequency to be played
                          countdurSPK=0;           //reset count

        }
    }
}

if(displaying)                                  //if notes are to be displayed
{
    countdurVGA++;              //Timer3 has counted one cycle
    if(countdurVGA==durat[j])     //if right number of cycles for current
    {                                                   //-note being displayed
        j++;                    //display next
        countdurVGA=0;          //reset count
        if(j==number)           //if finished
        {
            PORTD=0;            //display no notes
            displaying=0;       //stop displaying new notes
        }
        else
        {
            PORTD=butto[j];     //display next note
        }
    }
}


PIR2bits.TMR3IF=0;              //clear Timer3 interrupt flag

}
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

top.v:
Top module for FPGA part of the game system.
*/
//////////////////////////////////////////////////////////
module top(
      input clk, reset,newNote1, newNote2, newNote3, newNote4, flash,
      input [1:0] mode,
      output R,G,B, hsync, vsync);


   wire clkVGA;
   wire inDisplayArea;
   wire [9:0] CounterX;
   wire [8:0] CounterY;
   wire [479:0] string1data, string2data, string3data, string4data;


      // Instantiate the clock manager
   clkmod25 clkmng(
    clk,
    reset,
    clkdv_out,
    clkfx_out,
    clkVGA,
    locked_out
    );

   //Make four instances, one for each string
   stringNotesSlow string1(clkVGA,reset, newNote1, string1data);
   stringNotesSlow string2(clkVGA,reset, newNote2, string2data);
   stringNotesSlow string3(clkVGA,reset, newNote3, string3data);
   stringNotesSlow string4(clkVGA,reset, newNote4, string4data);


   hvsync_generator hvsync(clkVGA, hsync, vsync, inDisplayArea, CounterX, CounterY);

   makePixel pixel(clk,newNote1,newNote2,newNote3,newNote4, inDisplayArea,mode, CounterX
CounterY, string1data, string2data, string3data, string4data, flash, R, G, B);


endmodule

```

```verilog
////////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155
Names: Einar B Magnusson and Alexandre Amert

hvsync_generator:
Generates the horizontal and vertical sync and outputs the coordinates
of the current pixel and whether it is in the display area.

Reference: fpga4fun.com
*/
////////////////////////////////////////////////////////////

module hvsync_generator2(
      input clk,
      output vga_h_sync, vga_v_sync,
      output inDisplayArea,
      output [9:0] CounterXout,
      output [8:0] CounterYout);


reg [9:0] CounterX;
reg [8:0] CounterY;
wire CounterXmaxed = (CounterX==10'd800);
wire CounterYmaxed = (CounterY==10'd525);
reg   vga_HS, vga_VS;

assign CounterXout=CounterX-40;
assign CounterYout=CounterY-25;

always @(posedge clk)
   if(CounterXmaxed)
      CounterX <= 0;
   else
      CounterX <= CounterX + 1;

always @(posedge clk)
if(CounterXmaxed) CounterY <= CounterY + 1;

always @(posedge clk)
begin
   vga_HS <= (CounterX>704);
   vga_VS <= (CounterY==500);
end

reg inDisplayArea;
always @(posedge clk)
if(CounterX>40 & CounterX<680 & CounterY>25 & CounterY<505)
   inDisplayArea <=1;
else
   inDisplayArea <=0;

assign vga_h_sync = ~vga_HS;
assign vga_v_sync = ~vga_VS;

endmodule

```

```verilog
//////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

stringNotesSlow:
Shift register to keep the notes of each string,
slowed down to shift a bit through in about 13 seconds
*/
//////////////////////////////////////////////////////////
module stringNotesSlow(
      input clkDisp, reset, newNote,
      output reg [479:0] notes);

   reg [24:0] count;

   always @(posedge clkDisp, posedge reset)
   if(reset)
   begin
      notes<=0;
      count <=0;
   end
   else
   begin
      if(count == 1250000)
      begin
         notes <= {notes[478:0],newNote};
         count <=0;
      end
      else
      begin
         count <= count +1;
      end
   end


endmodule

```

```

	////////////////////////////////////////////////////////
	/*
	Project: Rhythm Game, Final project of E155

	Names: Einar B Magnusson and Alexandre Amert

	isStringXnote:
	Determines whether the pixel is on a note on String centered at
	center.
	*/
	////////////////////////////////////////////////////////
	module isStringXnote(
		input [9:0] center,
		input [9:0] CounterX,
		input [8:0] CounterY,
		input [479:0] stringXdata,
		output stringout);

	   assign stringout = (CounterX < center+10) & (CounterX > center+10) &
	                        stringXdata[CounterY];

	endmodule

	////////////////////////////////////////////////////////
	/*
	Project: Rhythm Game, Final project of E155

	Names: Einar B Magnusson and Alexandre Amert

	isString:
	Determines whether the pixel is on a string
	*/
	////////////////////////////////////////////////////////
	module isString(
		input [9:0] CounterX,
		output string);

	   assign string = (CounterX==128 | CounterX==256 | CounterX==384 | CounterX==512);

	endmodule

	////////////////////////////////////////////////////////
	/*
	Project: Rhythm Game, Final project of E155

	Names: Einar B Magnusson and Alexandre Amert

	isBar:
	Determines whether the pixel is on the bottom bar
	*/
	////////////////////////////////////////////////////////
	module isBar(
		input [8:0] CounterY,
		output bar);

	   assign bar = (CounterY<433 & CounterY>407);

	endmodule

	////////////////////////////////////////////////////////
```

```
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

isScoreBarOutline:
Determines whether the pixel is on the outline of the
score bar
*/
/////////////////////////////////////////////////////////
module isScoreBarOutline(
      input [9:0] CounterX,
      input [8:0] CounterY,
      output isScoreOutline);

   assign isScoreOutline = ( CounterX==256 | CounterX==384 |
                          (CounterX<384 & CounterX>256 &(CounterY%15==0)));

endmodule

/////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

isScoreBar:
Determines whether the pixel is on the colored part
of the score bar
*/
/////////////////////////////////////////////////////////
module isScoreBar(
      input [9:0] CounterX,
      input [8:0] CounterY,
      input [4:0] data,
      output isScoreBar);

   assign isScoreBar = ((CounterX<384 & CounterX>256)& (CounterY>(480-15*data)));

endmodule

/////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

isBar:
Determines whether the pixel is on the difficulty
selection box centered at center
*/
/////////////////////////////////////////////////////////
module isDiffBox(
      input [9:0] center,
      input [9:0] CounterX,
      input [8:0] CounterY,
      output isBox);

   assign isBox = ( CounterX>center-40 & CounterX <center+40 &
                  CounterY<280 & CounterY>200);

```

```verilog
endmodule


/////////////////////////////////////////////////////////
/*
Project: Rhythm Game, Final project of E155

Names: Einar B Magnusson and Alexandre Amert

PixelColor:
Depending on the which elements the pixel is on and their

different priorities, determines the color.
*/
/////////////////////////////////////////////////////////
module PixelColor(
      input clk,newNote1,newNote2,newNote3,newNote4, inDisplayArea,
         string, string1, string2, string3, string4, bar, flash,isscoreline,
         isscorebar,iseasy,ismedium,ishard,isveryhard,
      input [1:0] mode,
      output reg [2:0] RGB);

   always @(posedge clk)
   begin
      if(mode==1)
      begin
         if(~inDisplayArea)
            RGB<=0;
         else if(string1)
            RGB<=3'b110;
         else if(string2)
            RGB<=3'b010;
         else if(string3)
            RGB<=3'b001;
         else if(string4)
            RGB<=3'b011;
         else if(bar)
         begin
            if(flash)
               RGB<=3'b100;
            else
               RGB<=3'b111;
         end
         else if(string)
            RGB<=3'b111;
         else
            RGB<=3'b000;
      end
      else if(mode==2)
      begin
         if(isscoreline)
            RGB<=3'b111;
         else if(isscorebar)
            RGB<=3'b010;
         else
            RGB<=0;

      end
      else
      begin
         if(iseasy)
```

```
                RGB<=3'b010;
             else if(ismedium)
                RGB<=3'b001;
             else if(ishard)
                RGB<=3'b101;
             else if(isveryhard)
                RGB<=3'b100;
             else
                RGB<=0;
          end


       end

    endmodule

    /*
    module RGBsignal(input [2:0] RGB, output R, G, B);

       assign R = RGB[2];
       assign G = RGB[1];
       assign B = RGB[0];

    endmodule
    */
    /////////////////////////////////////////////////////////
    /*
    Project: Rhythm Game, Final project of E155

    Names: Einar B Magnusson and Alexandre Amert

    makePixel:
    Takes in all relevant data to determine what the color

    of the current pixel should be.
    */
    /////////////////////////////////////////////////////////
    module makePixel(
          input clk, newNote1, newNote2, newNote3, newNote4, inDisplayArea,
          input [1:0] mode,
          input [9:0] CounterX,
          input [8:0] CounterY,
          input [479:0] string1data, string2data, string3data, string4data,
          input flash,
          output R,G,B
                );

       wire string1, string2, string3, string4, string, bar,isscoreline;
    // wire [2:0] RGB;

       wire [9:0] center1= 128;
       wire [9:0] center2= 256;
       wire [9:0] center3= 384;
       wire [9:0] center4= 512;

       //check if the current pixel is on a note
       isStringXnote string1note(center1, CounterX, CounterY, string1data, string1);
       isStringXnote string2note(center2, CounterX, CounterY, string2data, string2);
       isStringXnote string3note(center3, CounterX, CounterY, string3data, string3);
       isStringXnote string4note(center4, CounterX, CounterY, string4data, string4);

```

```
        //check if the current pixel is on a string or the bar
        isString isstring(CounterX, string);
        isBar isbar(CounterY, bar);

        //for end-of-game mode, check if pixel is on score outlines or score bar
        isScoreBarOutline isscoreoutl(CounterX,CounterY,isscoreline);
        isScoreBar isscoreb(CounterX,CounterY,{flash,newNote4,newNote3,newNote2,
           newNote1},isscorebar);

        //for beginning-of-game mode
        isDiffBox isE(center1,CounterX,CounterY,iseasy);
        isDiffBox isM(center2,CounterX,CounterY,ismedium);
        isDiffBox isH(center3,CounterX,CounterY,ishard);
        isDiffBox isVH(center4,CounterX,CounterY,isveryhard);

        //determine the color of the current pixel
        PixelColor determineColor(clk,newNote1,newNote2,newNote3,newNote4, inDisplayArea,
           string, string1, string2, string3, string4, bar, flash, isscoreline,isscorebar,
           iseasy,ismedium,ishard,isveryhard,mode, {R,G,B});
     // RGBsignal splitThem(RGB,R,G,B);

     endmodule
```