# **Bilkent University**

# **Electrical and Electronics Department**

# **EE102-01 Term Project Report:**

"Guitar Hero"

30/12/2023

Fatih Mehmet Cetin - 22201689

## **Purpose:**

The main purpose of this project is to implement a Guitar Hero game on BASYS3 FPGA board via VHDL. The inspiration behind this project belongs to the popular game series "Guitar Heroes". The original game was first introduced to the market by Harmonix in 2005 and then several companies made contribution to the development of the game. The goal of the game is to "play" the sliding notes of a song in time by pressing the appropriate buttons that match with the ones on the screen on a guitar shaped controller. BASYS3 FPGA Board, TRC-11, a VGA monitor, a pair of speakers and a guitar shaped game controller is used during the project.

### **Design Specifications:**

The design has 5 inputs and 3 outputs. The inputs are reset, start, difficulty, song selection and the note buttons on the guitar shaped controller. The outputs are the VGA output, score output and the audial output. Here is a brief description of each input&output:

- **Reset:** The VGA screen reset button, does not have any effects on the gameplay, belongs in the project just in case if the synchronization of the VGA outputs (hsync, vsync and rgb) go wrong at some point. Reset basically resets these 3 outputs and the game goes on as expected.
- **Start:** The input that starts the game. This input is assigned to a switch; if the switch is on, the gameplay screen comes on; if it is off, the main menu screen appears.
- **Difficulty:** The difficulty input that determines the speed of the sliding notes during the gameplay, is assigned to a switch. 2 difficulty levels are present in the game: easy&hard.
- Song Selection: The input that determines the song that will be played, is assigned to a switch. There are two songs in the game: Smoke on the water and Thunderstruck.

  Different songs have different note combinations during the game.
- **Note Buttons:** The 8-bit input which comes from the guitar shaped main game controller.
- VGA Outputs (hsync, vsync, rgb): The three outputs that are required for the VGA output flowing out of the FPGA board. Hsync and vsync determines the specific pixel that will be painted on the monitor and the 12-bit rgb(red-green-blue) output which basically is the colour output for the chosen pixel via hsync&vsync. A 640\*480 pixel VGA output was used in the project. This is quite a low-resolution but since the game

did not require high-resolution but required a high refresh rate, I kind of sacrificed the resolution rate and went for the screen refresh rate since they are inversely proportioned to each other for the same clock frequency.

- Score: The output that is projected onto the seven-segment display of the BASYS3.
   The score function shows how many notes have been hit correctly at the correct time throughout the entire song.
- Audial Output: The 1-bit output that changes its frequency according to the game logic. This output is connected to the speaker-amplifier of the TRC-11 and then exposed audially via a speaker.

The design requires some external components besides the BASYS3 FPGA Board.

These external components are TRC-11, a VGA monitor, a guitar controller and a pair of speakers. Here is the description of each external component:

• TRC-11: This is the high frequency radio transceiver (transmitter&receiver) that I built during a full semester in another course I take this year EEE-211 Analog Electronics. Only the speaker amplifier of the TRC-11 is used in the project. It is used to amplify&transmit the square wave signal coming from the BASYS3 to a pair of speakers. Here you can see the schematic of the speaker amplifier of the TRC-11 (Figure 1.1):

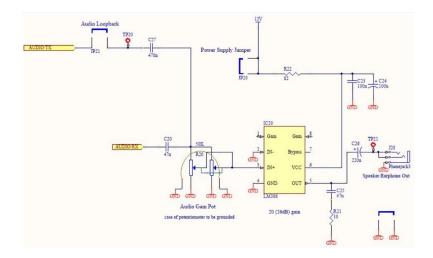


Figure 1.1: The Schematic of the speaker amplifier of TRC-11

- VGA Monitor: The game should be projected onto a monitor. Since BASYS3 has a VGA output, a VGA monitor is used to project the gameplay.
- **Guitar Controller:** The main game controller. Includes 8 buttons which are basically the ones on the gameplay screen.
- **Speakers:** Jack-input speakers that are used to expose the audial signal coming from the TRC-11.

## **Methodology:**

Since I knew very little about VHDL at the start of the project, the very first task I committed myself was to learn the VGA process in VHDL. I did some research and understood the VGA process in VHDL. Then I wrote a VGA driver code by getting help from the internet. Understanding how VGA works in VHDL was definitely one of the trickiest and the most challenging parts within the project.

The second task I committed myself was to write the game logic. I aimed to just see the sliding notes on the monitor. There were no note inputs or a main menu present at this point of the project. After a few hours of hard work, I managed to see the sliding notes on the monitor even though I had some synchronization issues initially. This was a major mental and psychological stepping stone in the project since I started to feel like I could actually do the proposed design.

The third task I committed myself was to create a score function. I had initial thoughts about projecting the score value onto the VGA monitor, but I had second thoughts and decided to project it onto the seven-segment display of the BASYS3. This was mainly due to me wanting to use a variety of features present in the BASYS3 and not sticking with the VGA output only. At this part of the project, the seven-segment lab we did during the semester really helped me and significantly reduced the time I spent on this part. Also, I added the 8 note buttons on a breadboard and managed to successfully get the score data out of the game. At this point I had some initial struggle since the score function incremented more than once within the same note –once per every clock cycle-, but I managed to solve it via using a boolean array and made sure that no note was counted more than once for the score function.

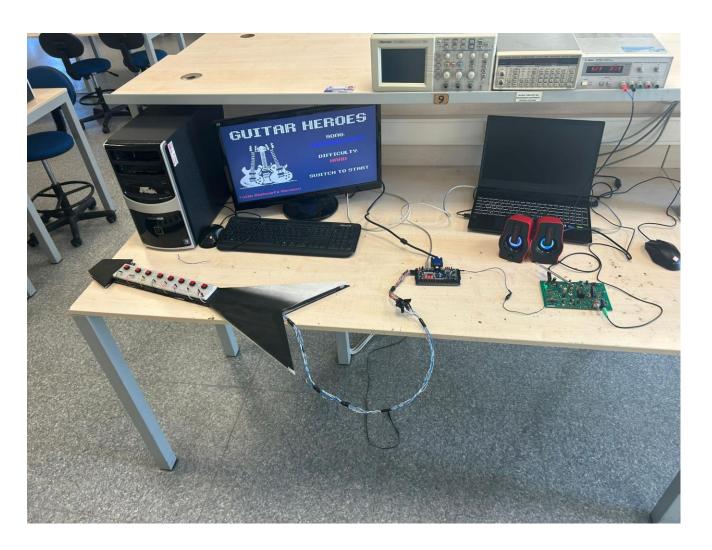
The fourth task I committed myself was to create a main-menu screen and using a start input for switching back and forth the gameplay screen and the main menu. I used bitmap images and vector arrays for designing the main menu. I found out that designing a screen with VHDL is super time-consuming. For the main menu screen, I used image generating AI DALL-E 3 and the pixelion writing style. I converted jpg images to binary arrays via a website. Then, I manually determined the horizontal and vertical pixel information for each text or picture I put to the main menu screen.

The fifth task I committed myself was to obtain the audial output. I used digital square wave signals and transformed to soundwaves via TRC-11's speaker amplifier and a pair of speakers. In one of the labs of EEE-211, we connected the signal generator to the speaker amplifier of TRC-11 and generated square wave signals and listened to the sound of each frequency. This made me think using TRC-11 as the audial output amplifier instead of purchasing external audio synthesizers compatible with BASYS3. This approach was going to be more economic for a student. Different notes have different frequencies and I've manually coded the notes of the song in such a way that the melody of the real song could be heard from the speakers.

The last task I committed myself was to create the guitar shaped controller. This process involved only mechanical work and no electronical&programming work. I first drew the guitar hardware on a thin wooden plate. Then I shaped the wooden plate via a wood saw. I obtained two examples of this guitar shape which was going to be the two sides of the guitar. I wanted to pass the cables in between the two thin guitar plates. Therefore, I also obtained small but thick and strong wooden cubic shapes to support the two thin plates mechanically. Then I spray-painted and varnished both sides of the guitar shaped wooden plates and the supporting material. This helped me getting a smoother feeling and a better look for the guitar controller. Then I placed the breadboards with buttons on the guitar and passed the note button cables within the two plates. Finally, I sticked everything together via a two-component glue. The guitar controller was ready and set for the game.

## **Results:**

This was the general setup after everything was ready (**Figure 1.2**). You can see the guitar controller, the VGA monitor, TRC-11, BASYS3 FPGA board and the cable connections clearly.



**Figure 1.2: The General Setup for the Game** 

And here you can see me holding the controller (Figure 1.3):



Figure 1.3: Me Holding the Guitar Shaped Controller

### **Conclusion:**

This project is a significant milestone in my career. It is truly incredible what can we do with FPGA and VHDL. I aimed to make a guitar hero game and eventually managed to do so.

The biggest challenge throughout the project was definitely building the guitar hardware. It involved heavy work and consumed lots of hours. Besides that, the synchronization of the game logic was another major issue I faced. Therefore, I've written everything in a single module even though my initial intentions were using sub-modules. The final major drawback I'd like to mention is the memory issue I encountered. I had to reduce the number of notes per song to 80 because I encountered a warning about the number of LUT-cells present in the BASYS3. This memory issue was also the reason why I had to stick with two songs. When I tried to add more songs, I encountered the same warning from Vivado.

I believe having a lab about the VGA output of the BASYS3 is necessary during the course. It could have saved me and my friends a lot of time during the term project if we had that lab.

### **References:**

screen design

https://studio.code.org/projects/applab/qiyLvNCBDuOYbaBB8oe0isTwNDYTOeGA5cpWlh
HNTzM - very useful visualization of 12-bit rgb coding, helped me through the gameplay

https://www.dcode.fr/binary-image - the site I used for jpg to binary array conversion
https://openai.com/dall-e-3 - the AI that generated the guitar image on the main menu
https://www.youtube.com/watch?v=eJMYVLPX0no&t=228s - VGA driver code source:
highlighted with yellow in the appendix part.

### **Appendices:**

Youtube Video Link: <a href="https://www.youtube.com/watch?v=hGeJhr4xRpw&t=324s">https://www.youtube.com/watch?v=hGeJhr4xRpw&t=324s</a>

```
VHDL Code:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tester is
  Port (clk: in STD_LOGIC; --100Mhz internal BASYS3 clock
      reset: in STD_LOGIC; --reset button
      start: in std_logic; --start switch
      difficulty: in std_logic; --difficulty selection switch
      select_song: in std_logic; --song selection switch
      note_buttons: in std_logic_vector (7 downto 0); --guitar buttons
      hsync : out STD_LOGIC; --horizontal sync output for vga driver
      vsync : out STD_LOGIC; --vertical sync output for vga driver
      audio_output: out std_logic; --the audial output
      anodes: out std_logic_vector(3 downto 0); --anodes of seven segment
      cathodes: out std_logic_vector(6 downto 0); -- cathodes of seven segment
      rgb: out STD_LOGIC_VECTOR (11 downto 0)); --the 12-bit RedGreenBlue colour
code
      --very useful visualization of 12-bit rgb coding at
https://studio.code.org/projects/applab/qiyLvNCBDuOYbaBB8oe0isTwNDYTOeGA5cpWlh
HNTzM
end tester;
architecture Behavioral of tester is
```

```
--25 MHz clock
signal clk25: std_logic := '0';
--50MHz clock
signal clk50: std_logic := '0';
-- the horizontal constants for the 640*480 vga screen
constant hd: integer:= 639; --horizontal display
constant hfp: integer:= 16; --horizontal front porch
constant hsp: integer:= 96; --horizontal sync pulse
constant hbp: integer:= 48; --horizontal back porch
signal hpos: integer:= 0; --horizontal position
--the vertical constants for the 640*480 vga screen
constant vd: integer:= 479; --vertical display
constant vfp: integer:= 10; --vertical front porch
constant vsp: integer:= 2; --vertical sync pulse
constant vbp: integer:= 33; --vertical back porch
signal vpos: integer:= 0; --vertical position
--every single note height and widths in pixels
constant note_height: integer := 20;
constant note_width: integer := 50;
--horizontal and vertical difference between two notes in pixels
constant notes_height_diff: integer := 20;
constant notes_width_diff: integer := 10;
--different colours in 12-bit rgb representation for the notes
constant colour_green: std_logic_vector(11 downto 0):= "000011110000";
constant colour_red: std_logic_vector(11 downto 0):= "111100000000";
```

```
constant colour_yellow: std_logic_vector(11 downto 0):= "1111111110000";
constant colour_blue: std_logic_vector(11 downto 0):= "000000001111";
constant colour_orange: std_logic_vector(11 downto 0):= "111110000000";
constant colour_violet: std_logic_vector(11 downto 0):= "110000001111";
constant colour_white: std_logic_vector(11 downto 0):= "111111111111";
constant colour_pink: std_logic_vector(11 downto 0):= "111100001100";
--different colours in 12-bit rgb representation for background screen (NOT THE NOTES)
constant colour_gray: std_logic_vector(11 downto 0):= "000100010001";
constant colour_black: std_logic_vector(11 downto 0):= "000000000000";
-- the speed of the notes that are sliding down the screen
signal vertical_speed: integer;
--horizontal pixel information of each note according to the colour
constant note_green_horizontal: integer:= 85;
constant note_red_horizontal: integer:= 145;
constant note_yellow_horizontal: integer:= 205;
constant note_blue_horizontal: integer:= 265;
constant note_orange_horizontal: integer:= 325;
constant note_violet_horizontal: integer:= 385;
constant note_white_horizontal: integer:= 445;
constant note_pink_horizontal: integer:= 505;
-- the vertical pixel information of the strum area where the buttons must be hit exactly when
the notes are there
constant strum_upper_limit: integer:= 390;
constant strum_lower_limit: integer:= 430;
constant strum_limit_height: integer:= 10;
-- the array that contains the vertical pixel information of each note
type note_vertical_properties_array is array (0 to 79) of integer; --each element of the array
contains the vertical pixel info of every note
signal note_vertical_properties: note_vertical_properties_array;
```

```
--the arrays that contain the horizontal pixel information and the colour of each note
type note_horizontal_colours_string_array is array (0 to 79) of string(1 to 1);--each element
of the array is one of "grybovwp" (note colours' initials)
signal note horizontal colours string: note horizontal colours string array;
type note horizontal properties array is array (0 to 79) of integer; -- each element of the array
contains the horizontal pixel info of every note
signal note horizontal properties: note horizontal properties array;
type note horizontal colours array is array (0 to 79) of std logic vector(11 downto 0);--
each element of the array is a 12-bit rgb representation of the note colour
signal note_horizontal_colours: note_horizontal_colours_array;
--the following 11 bitmaps are displayed in the main menu / scroll all the way down and
check the binary arrays at the very end of the behavioral part!
--shoutout to https://www.dcode.fr/binary-image which is the perfect site for jpg to binary
array conversion
signal gamenamevisible: boolean;
type bitmap1 is array (0 to 47) of std_logic_vector(0 to 639);
signal gamename: bitmap1;
signal photovisible: boolean;
type bitmap2 is array (0 to 260) of std_logic_vector(0 to 299);
signal photo: bitmap2;
signal songvisible: boolean;
type bitmap3 is array (0 to 18) of std_logic_vector(0 to 339);
signal song: bitmap3;
signal thunderstruckvisible: boolean;
type bitmap4 is array (0 to 18) of std_logic_vector(0 to 339);
signal thunderstruck: bitmap4;
signal smokevisible: boolean;
type bitmap5 is array (0 to 18) of std_logic_vector(0 to 339);
```

```
signal smoke: bitmap5;
signal difficultyvisible: boolean;
type bitmap6 is array (0 to 18) of std_logic_vector(0 to 339);
signal difficulty_2: bitmap6;
signal easyvisible: boolean;
type bitmap7 is array (0 to 18) of std_logic_vector(0 to 339);
signal easy: bitmap7;
signal hardvisible: boolean;
type bitmap8 is array (0 to 18) of std_logic_vector(0 to 339);
signal hard: bitmap8;
signal startswitchvisible: boolean;
type bitmap9 is array (0 to 18) of std_logic_vector(0 to 339);
signal startswitch: bitmap9;
signal fatihmehmetvisible: boolean;
type bitmap10 is array (0 to 13) of std_logic_vector(0 to 299);
signal fatihmehmet: bitmap10;
signal lightningvisible: boolean;
type bitmap11 is array (0 to 144) of std_logic_vector(0 to 84);
signal lightning: bitmap11;
--the array that makes sure you only get 1 point from each note
--p.s. without this array you would get 1 point in each clock cycle -which means if you
pressed the correct button for 1 sec, you would get 25 million points :)
type NoteScoredArray is array (0 to 79) of boolean;
signal note_scored : NoteScoredArray:= (others=>False); --set the initial value to False for all
80 elements of the array
```

```
--the score output which will be displayed on the seven segment display and its' three
decimals
signal score: integer:= 0;
signal score0: integer;--least significant figure of the score signal
signal score1: integer;
signal score2: integer;
signal score3: integer; --most significant figure of the score signal
-- the necessary signals for creating a 3000Hz clock for seven segment display
signal ss_counter: std_logic_vector(19 downto 0):= "00000000000000000000";
signal clk3000: std_logic_vector(1 downto 0);--3000Hz clock
signal ss_leds: integer;
-- the frequencies of the 8 different notes
--p.s. if you divide each value to 25MHz, you find the corresponding frequency value
--Ex: 25 \text{million}/23889 = 1046.50 \text{ Hz for freq} 8
signal freq8 : integer := 23889; --pink
signal freq7: integer := 25316; --white
signal freq6 : integer := 26409; --violet
signal freq5: integer := 31887; --orange
signal freq4: integer := 35790; --blue
signal freq3 : integer := 37936; --yellow
signal freq2: integer := 42589; --red
signal freq1 : integer := 47801; --green
signal freqbuzz : integer := 150000; --buzzer
signal frequull: integer := 100000000; -- no sound
-- the necessary signals for the sound output
signal freq_counter: integer := 0;
signal frequency: integer:= 0;
signal audio_signal: std_logic:= '0';
```

```
begin
```

```
--this is the process where we give an output of a square wave signal with the selected
frequency
audio_output <= audio_signal;</pre>
audio: process(clk25)
begin
  if rising_edge(clk25) then
     if freq_counter >= frequency then
       audio_signal <= not audio_signal;</pre>
       freq_counter <= 0;
    else
       freq_counter <= freq_counter + 1;</pre>
     end if:
  end if;
end process;
--obtain a nearly 3kHz clock which will be used in the seven segment display from the
internal 100MHz BASYS3 clock
clkdivider2: process(clk)
begin
  if (rising_edge(clk)) then
     ss_counter <= ss_counter + 1;</pre>
  end if;
clk3000 <= ss_counter(16 downto 15);
end process;
--calculate the buzzing output and the score value, p.s. be very careful to not increase the
score more than once at the same note:)
score_count: process(clk25)
begin
  if rising_edge(clk25) then
```

```
if start = '1' then
for i in 0 to 79 loop
```

- --this part needs a very clear explanation; first we basically set the limits and increased the score if the
- --correct button has been hit at the right time. In order to not increase the score multiple times at the
- --same note, we use the note\_scored array and initialized it at false for every element.
- --Then when a note is hit that element's note\_scored property is locked to true until the restart of the game!

```
--Note that the buzzer signal is not dependent on the note_scored array

if ((note_vertical_properties(i) + (note_height/2)) > strum_upper_limit) and
((note_vertical_properties(i) - (note_height/2)) < strum_lower_limit) then

case note_horizontal_colours_string(i) is

when "g" =>

if note_buttons(6)= '1' or note_buttons(5)= '1' or note_buttons(4)= '1' or
note_buttons(3)= '1' or note_buttons(2)= '1' or note_buttons(1)= '1' or note_buttons(0)= '1'
then

frequency <= freqbuzz;
elsif note_buttons(7) = '1' then
frequency <= freq1;
```

```
frequency <= freq1;
else
  frequency <= freqnull;
end if;
if not note_scored(i) then
  if note_buttons(7) = '1' then
    score <= score + 1;
    note_scored(i) <= True;
  end if;
end if;</pre>
```

if note\_buttons(7)= '1' or note\_buttons(5)= '1' or note\_buttons(4)= '1' or note\_buttons(3)= '1' or note\_buttons(2)= '1' or note\_buttons(1)= '1' or note\_buttons(0)= '1' then

```
frequency <= freqbuzz;
elsif note_buttons(6) = '1' then
```

when "r" =>

```
frequency <= freq2;
                  else
                    frequency <= frequull;</pre>
                  end if;
                  if not note_scored(i) then
                    if note\_buttons(6) = '1' then
                       score \le score + 1;
                       note_scored(i) <= True;</pre>
                    end if;
                  end if;
               when "y" =>
                  if note_buttons(7)= '1' or note_buttons(6)= '1' or note_buttons(4)= '1' or
note_buttons(3)= '1' or note_buttons(2)= '1' or note_buttons(1)= '1' or note_buttons(0)= '1'
then
                    frequency <= freqbuzz;</pre>
                  elsif note_buttons(5) = '1' then
                    frequency <= freq3;
                  else
                    frequency <= frequull;</pre>
                  end if:
                  if not note_scored(i) then
                    if note_buttons(5) = '1' then
                       score \le score + 1;
                       note_scored(i) <= True;</pre>
                    end if:
                  end if:
               when "b" =>
                  if note_buttons(7)= '1' or note_buttons(6)= '1' or note_buttons(5)= '1' or
note_buttons(3)= '1' or note_buttons(2)= '1' or note_buttons(1)= '1' or note_buttons(0)= '1'
then
                    frequency <= freqbuzz;
                  elsif note_buttons(4) = '1' then
                    frequency <= freq4;
```

```
else
                     frequency <= frequull;</pre>
                  end if;
                  if not note_scored(i) then
                     if note\_buttons(4) = '1' then
                       score \le score + 1;
                       note_scored(i) <= True;</pre>
                     end if;
                  end if;
               when "o" =>
                  if note_buttons(7)= '1' or note_buttons(6)= '1' or note_buttons(5)= '1' or
note_buttons(4)= '1' or note_buttons(2)= '1' or note_buttons(1)= '1' or note_buttons(0)= '1'
then
                     frequency <= freqbuzz;</pre>
                  elsif note_buttons(3) = '1' then
                     frequency <= freq5;
                  else
                     frequency <= frequull;</pre>
                  end if:
                  if not note_scored(i) then
                     if note\_buttons(3) = '1' then
                       score \le score + 1;
                       note_scored(i) <= True;</pre>
                     end if:
                  end if:
               when "v" =>
                  if note buttons(7)= '1' or note buttons(6)= '1' or note buttons(5)= '1' or
note_buttons(4)= '1' or note_buttons(3)= '1' or note_buttons(1)= '1' or note_buttons(0)= '1'
then
                     frequency <= freqbuzz;
                  elsif note_buttons(2) = '1' then
                     frequency <= freq6;
                  else
```

```
frequency <= frequull;</pre>
                  end if;
                  if not note_scored(i) then
                     if note\_buttons(2) = '1' then
                        score \le score + 1;
                       note_scored(i) <= True;</pre>
                     end if:
                  end if;
               when "w" =>
                  if note_buttons(7)= '1' or note_buttons(6)= '1' or note_buttons(5)= '1' or
note_buttons(4)= '1' or note_buttons(3)= '1' or note_buttons(2)= '1' or note_buttons(0)= '1'
then
                     frequency <= freqbuzz;</pre>
                  elsif note_buttons(1) = '1' then
                     frequency <= freq7;
                  else
                     frequency <= frequull;</pre>
                  end if;
                  if not note_scored(i) then
                     if note_buttons(1) = '1' then
                        score \le score + 1;
                       note_scored(i) <= True;</pre>
                     end if;
                  end if:
               when others => --when "p" =>
                  if note buttons(7)= '1' or note buttons(6)= '1' or note buttons(5)= '1' or
note_buttons(4)= '1' or note_buttons(3)= '1' or note_buttons(2)= '1' or note_buttons(1)= '1'
then
                     frequency <= freqbuzz;</pre>
                  elsif note_buttons(0) = '1' then
                        frequency <= freq8;
                  else
                     frequency <= frequull;</pre>
```

```
end if;
                  if not note_scored(i) then
                    if note\_buttons(0) = '1' then
                       score \le score + 1;
                       note_scored(i) <= True;</pre>
                    end if;
                  end if;
             end case;
          end if;
       end loop;
--set the score&buzzer values to zero when game is stopped or not initialized
     else
       score \leq 0;
       frequency <= freqnull;</pre>
       for i in 0 to 79 loop
          note_scored(i) <= False;</pre>
       end loop;
     end if;
  end if;
end process;
--display the score signal at the seven segment display for BASYS3
seven_segment: process(clk3000)
begin
--seperate each decimal of the score function
--remark that since #notes per song is 80, the score function can be max 80.
score0 <= score mod 10;
score1 <= (score mod 100/10);
score2 <= (score / 100);
score3<= (score/1000);
```

--match the anodes and each decimal of the score function

```
case clk3000 is --3000Hz clock for seven segment display
    when "00" =>
       anodes <= "1111";
       ss_leds <= score3;
    when "01" =>
       anodes <= "1011";
       ss_leds <= score2;
    when "10" =>
       anodes <= "1101";
       ss_leds <= score1;
    when others => --when "11"
       anodes <= "1110";
       ss_leds <= score0;
  end case;
--determine the 7-bit vector cathode signals for each integer
  case ss_leds is
    when 1 => cathodes <= "1001111";
    when 2 =  cathodes <= "0010010";
    when 3 =  cathodes <= "0000110";
    when 4 => cathodes <= "1001100";
    when 5 =  cathodes <= "0100100";
    when 6 =  cathodes <= "0100000";
    when 7 =  cathodes <= "00011111";
    when 8 =  cathodes <= "00000000";
    when 9 =  cathodes <= "0000100";
    when others \Rightarrow cathodes \iff "0000001"; --when 0 \implies
  end case;
end process;
```

--obtain a 25MHz clock from the internal 100MHz clock of BASYS3

```
clkdivider1: process(clk)
begin
  if rising_edge(clk) then
    clk50 <= not clk50;
  end if;
  if rising_edge(clk50) then
    clk25 \le not clk25;
  end if;
end process;
--create a horizontal position counter for the vga output
--very useful info on vga drivers at
https://www.youtube.com/watch?v=eJMYVLPX0no&t=228s
horizontal_position_counter: process(clk25, reset)
begin
  if reset = '1' then
    hpos \le 0;
  elsif rising_edge (clk25) then
    if (hpos = hd + hfp + hbp + hsp) then
       hpos \le 0;
    else
       hpos \le hpos + 1;
    end if;
  end if;
end process;
--create a vertical position counter which updates the vertical pos every time the horizontal
pos is resetted for the vga output
--very useful info on vga drivers at
https://www.youtube.com/watch?v=eJMYVLPX0no&t=228s
vertical_position_counter: process(clk25, reset)
begin
```

```
if reset = '1' then
     vpos \le 0;
  elsif rising_edge (clk25) then
     if (hpos = hd + hfp + hbp + hsp) then
       if (vpos = vd + vfp + vbp + vsp) then
          vpos \le 0;
       else
          vpos \le vpos + 1;
       end if;
    end if;
  end if;
end process;
-- the horizontal sync output for basys3
--very useful info on vga drivers at
https://www.youtube.com/watch?v=eJMYVLPX0no&t=228s
horizontal_synchronization: process(clk25, reset, hpos)
begin
  if reset = '1' then
    hsync <= '0':
  elsif rising_edge(clk25) then
     if (hpos \le (hd + hfp)) or (hpos > (hd + hfp + hsp)) then
       hsync <= '1';
     else
       hsync <= '0';
    end if;
  end if;
end process;
-- the vertical sync output for basys3
--very useful info on vga drivers at
https://www.youtube.com/watch?v=eJMYVLPX0no&t=228s
```

```
vertical_synchronization: process(clk25, reset, vpos)
begin
  if reset = '1' then
     vsync <= '0';
  elsif rising_edge(clk25) then
     if (vpos \le (vd + vfp)) or (vpos > (vd + vfp + vsp)) then
       vsync <= '1';
     else
       vsync <= '0';
     end if;
  end if;
end process;
-- the initialization of the vertical pixel infos of the notes and the sliding mechanism logic of
the game
vertical_pos: process(clk25, vpos, hpos)
begin
  if rising_edge(clk25) then
--increase the vertical pixel value of each note according to the vertical speed
     if start = '1' then
       if vpos = 0 and hpos = 0 then
          for i in 0 to 79 loop
             if note_vertical_properties(i) < vd then
               note_vertical_properties(i) <= note_vertical_properties(i) + vertical_speed;</pre>
--set the vertical value info to a value which is greater than max screen vertical pixel (480 in
our case)
             else
               note_vertical_properties(i) <= 1923; --the founding year of the Turkish
Republic <3
             end if:
          end loop;
       end if:
--initialization of vertical pixels of the notes, each of them starts at a negative value!
```

```
else
       for i in 0 to 79 loop
          note_vertical_properties(i) <= -1 * (i+1) * (note_height + notes_height_diff);</pre>
       end loop;
     end if;
  end if;
end process;
--set the horizontal pixel value and the 12-bit rgb colour representation of each note
according to the initials array
--i am lazy and since the notes are manually inputted, i just coded the initials
horizontal_pos_and_colour: process(clk25)
begin
     for i in 0 to 79 loop
       if note_horizontal_colours_string(i) = "g" then
          note_horizontal_properties(i) <= note_green_horizontal;</pre>
          note_horizontal_colours(i)<= colour_green;</pre>
       elsif note_horizontal_colours_string(i) = "r" then
          note_horizontal_properties(i) <= note_red_horizontal;</pre>
          note_horizontal_colours(i)<= colour_red;</pre>
       elsif note_horizontal_colours_string(i) = "y" then
          note_horizontal_properties(i) <= note_yellow_horizontal;</pre>
          note_horizontal_colours(i)<= colour_yellow;</pre>
       elsif note_horizontal_colours_string(i) = "b" then
          note_horizontal_properties(i) <= note_blue_horizontal;</pre>
          note_horizontal_colours(i)<= colour_blue;</pre>
       elsif note_horizontal_colours_string(i) = "o" then
          note_horizontal_properties(i) <= note_orange_horizontal;</pre>
          note_horizontal_colours(i)<= colour_orange;</pre>
       elsif note_horizontal_colours_string(i) = "v" then
          note_horizontal_properties(i) <= note_violet_horizontal;</pre>
          note_horizontal_colours(i)<= colour_violet;</pre>
```

```
elsif note horizontal colours string(i) = "w" then
          note horizontal properties(i) <= note white horizontal;
          note_horizontal_colours(i)<= colour_white;
       else --elsif note_horizontal_colours_string(i) = "p" then
          note_horizontal_properties(i) <= note_pink_horizontal;</pre>
          note_horizontal_colours(i)<= colour_pink;</pre>
       end if:
     end loop;
end process;
--change the initials array (Therefore the horizontal position and colour arrays) according to
the song selection by the user
--remark that the notes are coded in a form such you would play on a real guitar if you
actually played the real song
--easier to write the colour initials manually rather than the integer hpos and 12-bit vector rgb
values:)
with select_song select note_horizontal_colours_string <=
       ("g","b","g","b","g","b","g","b","g","b","r","o","r","o","r","o","r","o","r","o",--first 20
notes of seek&destroy
       "g","b","g","b","g","b","g","b","r","o","r","o","r","o","r","o", --next 16 notes
       "p","v","o","v","o","y","o","r","y","g","r","g","r","g","r","g", --next 16 notes
       "p","v","o","v","o","y","o","r","y","g","r","g","r","g","r","g", --next 16 notes
       "p","p","w","w","v","v","o","b","y","r","g","g") when '0', --last 12 notes
       ("g","y","o","o","g","y","v","o","g","y","o","o","y","g",--first\ 14\ notes\ of\ raining
blood
        "r","b","v","v","r","b","w","v","r","b","v","v","b","r", --next 14 notes
        "y","o","w","w","y","o","p","w","y","o","w","w","o","y", --next 14 notes
        "r","b","v","v","r","b","w","v","r","b","v","v","b","r", --next 14 notes
        "g","y","o","o","g","y","v","o","g","y","o","o","y","g", --next 14 notes
        "p","w","v","o","b","y","r","g","r","g") when others; --last 10 notes
```

```
--change the note speeds according to the difficulty input by the user
with difficulty select vertical_speed <=
  1 when '0',
  2 when others;
--this is the huge&complex process where we finally get to draw things on the VGA display
draw: process(clk25, reset, vpos, hpos)
begin
  if start = '1' then
--It is always nice to have a reset if the synchronization goes wrong at some point :)
     if reset = '1' then
       rgb <= colour_black;</pre>
     elsif rising_edge(clk25) then
       rgb <= colour_black;</pre>
       if (((vpos > (strum_upper_limit - strum_limit_height)) and (vpos <=
strum_upper_limit)) or
       ((vpos > strum_lower_limit) and (vpos <= (strum_lower_limit +
strum limit height))))
       then
--paint the strum area to the note colour when the player has hit a correct note
          if frequency = freq1 and ((hpos > note_green_horizontal) and (hpos <
note_green_horizontal + note_width)) then
            rgb<= colour_green;</pre>
          elsif frequency = freq2 and ((hpos > note_red_horizontal) and (hpos <
note_red_horizontal + note_width)) then
            rgb<= colour_red;
          elsif frequency = freq3 and ((hpos > note_yellow_horizontal) and (hpos <
note_yellow_horizontal + note_width)) then
            rgb<= colour_yellow;</pre>
          elsif frequency = freq4 and ((hpos > note_blue_horizontal) and (hpos <
note_blue_horizontal + note_width)) then
            rgb<= colour blue;
          elsif frequency = freq5 and ((hpos > note_orange_horizontal) and (hpos <
note orange horizontal + note width)) then
```

```
rgb<= colour_orange;</pre>
         elsif frequency = freq6 and ((hpos > note_violet_horizontal) and (hpos <
note_violet_horizontal + note_width)) then
            rgb<= colour violet;
         elsif frequency = freq7 and ((hpos > note_white_horizontal) and (hpos <
note white horizontal + note width)) then
            rgb<= colour black;
         elsif frequency = freq8 and ((hpos > note_pink_horizontal) and (hpos <
note pink horizontal + note width)) then
            rgb<= colour_pink;
--paint the background of the gameplay screen
         elsif (((hpos > note_green_horizontal) and (hpos < note_green_horizontal +
note_width)) or
         ((hpos > note_red_horizontal) and (hpos < note_red_horizontal + note_width)) or
         ((hpos > note_yellow_horizontal) and (hpos < note_yellow_horizontal +
note width)) or
         ((hpos > note_blue_horizontal) and (hpos < note_blue_horizontal + note_width)) or
         ((hpos > note_orange_horizontal) and (hpos < note_orange_horizontal +
note_width)) or
         ((hpos > note_violet_horizontal) and (hpos < note_violet_horizontal + note_width))
or
         ((hpos > note_white_horizontal) and (hpos < note_white_horizontal + note_width))
or
         ((hpos > note_pink_horizontal) and (hpos < note_pink_horizontal + note_width)))
         then
            rgb <= colour_white;
         end if:
       end if;
       if lightningvisible = True then
         rgb<= colour_red;
       end if:
--paint the necessary positions of the screen where notes are currently in with the necessary
note colours
       for i in 0 to 79 loop
         if note_vertical_properties(i) > (-1 * note_height) and note_vertical_properties(i)
<= vd then
```

```
if ((vpos > note_vertical_properties(i)) and (vpos <= note_vertical_properties(i)
+ note_height)) and
            ((hpos > note_horizontal_properties(i)) and (hpos <=
note_horizontal_properties(i) + note_width))
            then
               rgb <= note_horizontal_colours(i);</pre>
            end if:
          end if:
       end loop;
     end if:
  else
--this is the code for the main menu
     rgb <= colour_black;
     if (gamenamevisible = True) or (photovisible = True) or (songvisible = True) or
     (difficulty visible = True) or (starts witch visible = True)
     then
       rgb<=colour white;
     elsif (fatihmehmetvisible = True) or (hardvisible = True) then
       rgb<=colour_red;
     elsif (thunderstruckvisible = True)or (smokevisible = True)then
       rgb<=colour_blue;
     elsif (easyvisible = True) then
       rgb<=colour_green;
     end if:
  end if:
end process;
```

- -- the rest of the code is the design signals&bitmaps that belong to the main menu
- --the guitar picture is designed by an AI and the writings are designed in canva app with writing style 'Pixelion'
- --shoutout to https://www.dcode.fr/binary-image is the site I used for jpg to binary array conversion

#### gamename

gamenamevisible <= (vpos > 35) and (vpos < 85) and (gamename(vpos - 36)(hpos) = '1');

photo <=(

111111111111111111111111111111111111
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000

"000000000000000000000000000000000000	111
"000000000000000000000000000000000000	111
"000000000000000000000000000000000000	0000
"000000000000000000000000000000000000	0000
"0000000000000000000000000000000000000	111
"000000000000000000000000000000000000	111
"000000000000000000000000000000000000	101
"0000000000000000000000000000000000000	101

0000000000000000000011111110000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"0000000000000000000000000000000000111111

0",

"0000000000000000000000000000000000011111	001111000000000000 001001110111001111
"0000000000000000000000000000000000111111	000111000000000000 011101110111001111
"0000000000000000000000000000000000000	000111111111111111 011101110111001111
"0000000000000000000000000000000000000	000111111111111111 011101110111000111
"0000000000000000000000000000000000000	000111111111111111 011101110111000111
"0000000000000000000000000000000000000	000111111111111111 01111111111111000000
"0000000000000000000000000000000000000	000111111111111111 11111111110000000000
"0000000000000000000000000000000000000	0001111111111111111

0111111001111111110000000011111111111
"0000000000000000000000000000000000011111
"0000000000000000000000000000000000011111
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"0000000000000000000000000000000000011111
"000000000000000000000000000000000111111
"0000000000000000000000000000000000111111

"00000000000000000000000000000011111111	111000111111111111111 1111111111111111
0",	
"00000000000000000000000000000000000000	000000001111111111111
11111111111111111111100000000011111111	111000111111111111111
1111111100011111111111111111111111111	
111111111111111111111111111111111111111	000000000000000000000000000000000000000
0",	
"00000000000000000000000000000000000000	1111111111111111111111
111111111111111111000111111111111111111	
1111111110000011111111111111111111111	
111111111111111111111111111111111111	000000000000000000000000000000000000000
0",	
"00000000000000000000000000000000000000	1111111111111111111111
111111111111111111111111111111111111111	
111111110000011111111111111111111111111	
1111111111111111111100011111111111111	000000000000000000000000000000000000000
0",	
"00000000000000000000000000000000000000	1011111111111111111111
111111111111111111111111111111111111111	
111111110000011111111111111111111111111	
111111111111111111111111111111111111111	
"00000000000000000000000000000011111111	
111111111111111111111111111111111111	
111111111111111111111111111111111111111	
0",	000000000000000000000000000000000000000
"00000000000000000000000000000000000000	01111111111111111111111
111111111111111111111111111111111111	
1111111100000111111111111111111111111	
111111111111111100000000111111111111111	000000000000000000000000000000000000000
"00000000000000000000000000000000000000	0000000000000111111111
111111111111111111111111111111111111111	
111111110000011111111111111111111111111	

1111111111111110000000011111111111110000
"0000000000000000000000000000000000001111
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"0000000000000000000000000000000000000
"0000000000000000000000000000000000000
"0000000000000000000000000000000000000

"0000000000 111111111 111111111 100000000	11111 11110	1111 0011	1111 1111	1111	1111 1111	1111 1111	1111 1111	1111	1111 1111	111 111	1111 1111	100 111	)01 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 111111111 111111111 100000000	11111 11110	1111 0011	1111 1111	.111 .111	1111 1111	100 1111	0000 1111	00001	1111 1111	111 111	1111 1111	100 111	)01 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 111111111 111111111 1111000111 0",	11111 11111	1111 0000	1111 1111	1111	1111 1111	100 1111	0000 1111	0000 1111	1111 1111	111 111	1111 1111	100 111	)01 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 1111111111 1111111111 1111000111 0",	11111 11111	1111 1100	1111 0011	1111	1111 1111	100 1111	0000 1111	0000	1111 1111	111 111	1100 1111	)011   111	111 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 0111111111 1111111111 1111101111 0",	11111 11111	1111 1100	1111 0011	1111	1111 1111	100 1111	0000 1111	0000	1111 1111	111 111	1100 1111	)011   111	111 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 1111111111 1111111111 1111111	11111 11111	1111 1100	1111 0011	.111 .111	1111 1111	100 1111	0000 1111	0000.	1111 1111	111 111	1100 1111	)011   111	111 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 1100111111 1111111111 1111111	11111 11111	1111 1111	1111 1100	.111. 0011.	1111 1111	100 1111	0000 1111	)111  111	1111 1111	111 111	0000 1111	)011   111	111 111	111 111	111 111	111 111	111 111	111 111	11111 11111
"0000000000 000001111 111111111	11111	0011	1111	111	1111	100	0000	)111	1111	111	0000	001	111	111	111	111	111	111	11111

11111111111111111111111111000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000
"000000000000000000000000000000000000

"000000000000000000000000000000000000	0000
"000000000000000000000000000000000000	0000
"000000000000000000000000000000000000	0000
"000000000000000000000000000000000000	0000
"0000000000000000000000111101111111111	110 111
"0000000000000000000000111101111110111111	110 111
"000000000000000000000011110111110111111	110 111

0",

photovisible  $\leq$  (vpos > 130) and (vpos < 392) and (hpos < 300) and (photo(vpos-131)(hpos) = '1');

song <= (

songvisible  $\leftarrow$  (vpos>133) and (vpos<153) and (hpos>299) and (song(vpos-134)(hpos-300) = '1');

### thunderstruck<=

thunderstruckvisible  $\leq$  (vpos > 172) and (vpos < 192) and (hpos > 299) and (thunderstruck(vpos-173)(hpos-300) = '1') and (select\_song = '0');

# smoke<=

smokevisible  $\neq$  (vpos > 172) and (vpos < 192) and (hpos > 299) and (smoke(vpos-173)(hpos-300) = '1') and (select\_song = '1');

## difficulty\_2

difficulty visible  $\leq$  (vpos > 247) and (vpos < 267) and (hpos > 299) and (difficulty\_2(vpos-248)(hpos-300) = '1');

easy

easyvisible  $\neq$  (vpos > 286) and (vpos < 306) and (hpos > 299) and (easy(vpos-287)(hpos-300) = '1') and (difficulty = '0');

#### hard

hardvisible  $\neq$  (vpos > 286) and (vpos < 306) and (hpos > 299) and (hard(vpos-287)(hpos-300) = '1') and (difficulty = '1');

startswitch <=(

startswitchvisible <= (vpos > 366) and (vpos<386) and (hpos>299) and (startswitch(vpos-367)(hpos-300)='1');

#### fatihmehmet <=(

fatihmehmetvisible<= (vpos>440) and (vpos<456) and (hpos<300) and (fatihmehmet(vpos-441)(hpos)='1');

lightning <=(

0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000",

0000000000",

0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000", 0000000000",

0000000000",

end Behavioral;