De La Salle University Manila
Gokongwei College of Engineering
Electronics and Communications Engineering

LBYEC3G - EK2

Design of an HDL-based Rhythm Game

Submitted by:
Dionido, Athena Bianca P.
Lacanilao, Juan Miguel C.
Lamdagan, Ray Vincent Alin B.
Menodiado, Nico Angelo M.
Mercado, John Arvin A.
Siapco, Cyrill Mari L.

Date submitted:
September 26, 2021

Submitted to:
Dr. Carlo Noel Ochotorena

I. Statement of the Problem

The Video Graphic Array (VGA) interface has been widely used in most applications to send signals between computers and monitors so it can be displayed. An integrated circuit, the field-programmable gate array (FPGA), can be configured to run various commands and sequences using Verilog, a hardware description language (HDL), to be displayed with the help of the VGA interface. With the use of HDL and the FPGA, a simple game is developed and must be displayed in the VGA interface to understand the algorithms used. Moreover, a scoring system is also implemented to determine the outcome of the battle between two players. The game is a scrolling-type rhythm game where the players must hit the scrolling blocks in time to obtain points and the winner of the game is determined by the most amount of points achieved in a single battle. A simple interface is also necessary to inform the players of their acquired score and in order to avoid distractions, the point system is placed on the edge of the screen.

II. Objectives

For this project, the group had the following objectives:

1. To implement a simple video game using Verilog.
2. To program a field programmable gate array to run a video game.
3. To utilize the video graphics array (VGA) output of the FPGA.
4. To use Vivado 2020.2 in performing the project.

III. Review of Literature

With the continuous development in the microelectronics industry, FPGA is playing a more complex and more important role in designing a digital circuit. Blokus, a well known and easy to learn board game, already has several implementations making use of Artificial Intelligence (AI) algorithms. Jahanshahi et. al examined various algorithms and proposed a heuristic algorithm that will implement Blokus Duo, the two-player version of Blokus, on an FPGA on which it will also be able to play and make decisions for itself. (Jahanshahi et al., 2013) Most of the existing algorithms are limited due to them violating the game's time constraints when implemented on low-frequency

FPGA. The researchers' proposed algorithm makes use of three main heuristic strategies to be able to make the most optimal move. These include developing the board, disrupting the enemy player, and surviving. They were able to successfully implement this algorithm on an FPGA, employing an effective decision weighting strategy that allows it to beat even the most skilled players.

Moving on, FPGAs have been around for a considerable amount of time, and they've been used as tools for students to learn more about digital design. To familiarize the student with many different aspects of the VHDL, a simple game of "Simon Says" is to be developed on an FPGA with multiple people working on it as a team. Brought about by the complexity of the design, the students were given teams with different tasks to work on. In each team a distribution of tasks is proposed between the components both at design and verification level, which will train communication, leadership and collaborative work skills. (Jiménez-Fernández et al., 2020) As far as the design is concerned, the main objective of the group of students is to create a code based on their knowledge from the course and implement it on the Digilent Nexys4-DDR development board. Upon completion of the game design on the FPGA, the students were able to be more proficient in troubleshooting and debugging the codes, while keeping the writing of the VHDL as intact as possible.

Furthermore, as per VGAs or Video Graphics Arrays, these arrays have already been in use for various applications for years and have been considered as a standard interface. Furthermore, it has been used for certain display modes at any time and is divided into two modes, the character display mode and graphics display mode, however, in most applications, the graphics display mode is the only one that gets attention. Moving on, one issue commonly faced by a wide range of users is the low-resolution display. Thus, the use of VHDL is proposed as it is deemed to have the capability of describing the completion of a high-resolution VGA control module and a display module design that is a resource-conserving string. In addition, it can also provide two main modules of designing ideas and logic diagrams. (Wang et al., 2009)

A rhythm game is an action simulation game that adapts to the presented music. (Song et al., 2019) Also known as the music games, MUGs, it is a type of an electronic

game that is played on mainframe platforms, smear devices, computers, and consoles. In this type of game, players will follow the visual instructions and the rhythm of the music in accordance with the operation button or the on-screen instructions. Usually, game scores are reduced when there are mistakes. (Chen & Lo, 2016)

IV. Design Considerations

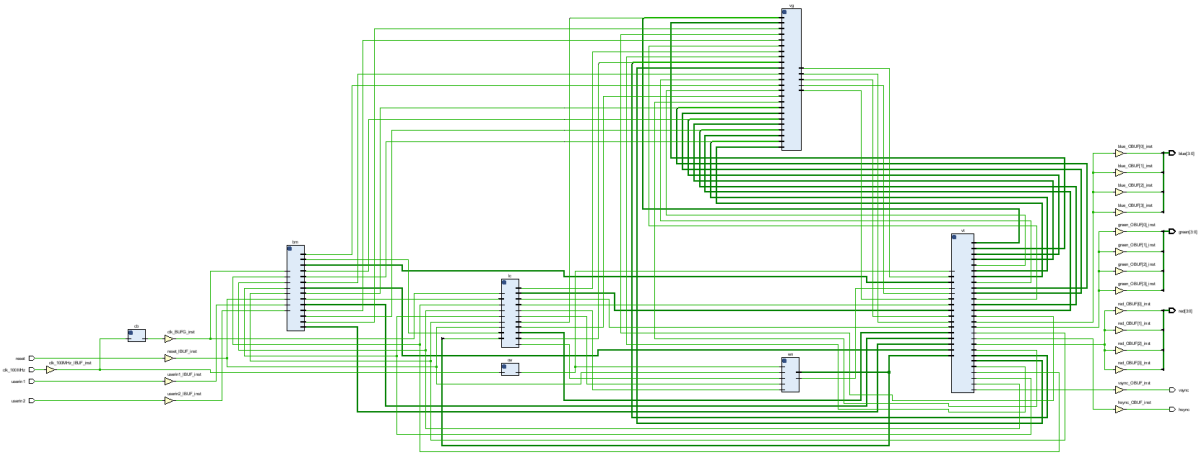

Figure 1. Schematic of the game

The game utilizes seven different modules for different functions of the game. These modules are linked together by a top module that serves as the main interface of the game to the FPGA.

Blockmaster

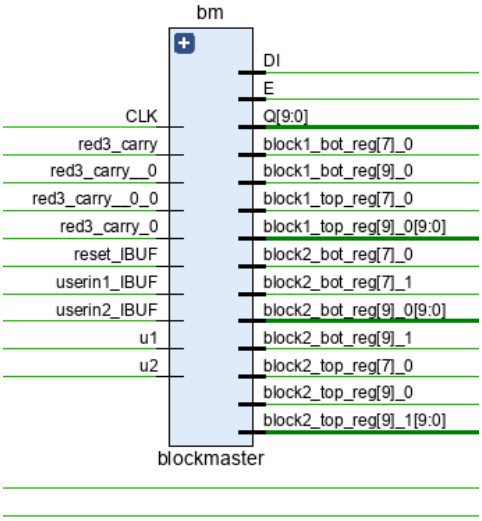

Figure 2. Blockmaster module

In a scrolling-type rhythm game, notes are hit accordingly to match a song or a beat. This concept was implemented by generating blocks of color to represent the music notes of the game. With this, the module entitled ‘blockmaster’ generates the coordinates of the blocks relative to the screen resolution. It controls the increments of the y-coordinate of all music notes through a slower clock pulse as well as resets to the original position whenever a note is considered a hit or when it reaches the bottom of the music line. The rectangular blocks have dimensions of 50 by 300 pixels with initial positions of 50 pixels from the top and the y-coordinate of both boxes are incremented by 5 pixels per clock pulse. It takes both player inputs and a reset as well as a slower clock as the module’s input while its outputs are both players’ blocks’ top, bottom, left, and right limits for the VGA generator module.

Clk_blk

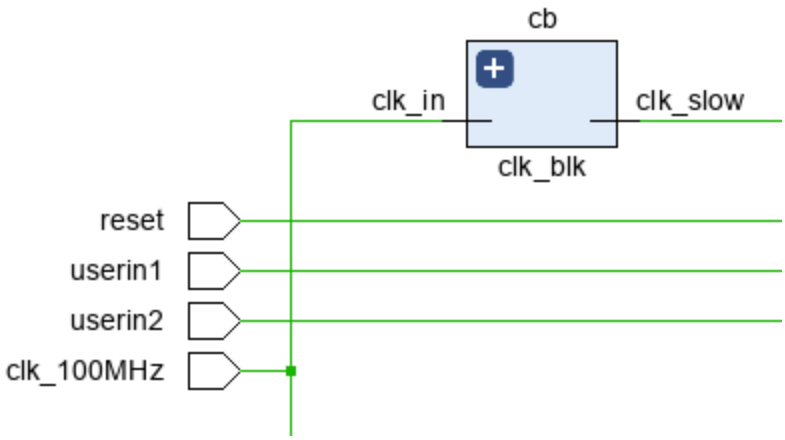

Figure 3. Clk_blk module

A slower clock was generated in order to further decrease the usual clock speeds used in FPGAs. This was used solely in the blockmaster module in order to slow down the increments on the y-coordinates of the generated blocks. Using the usual 100MHz or 74.25MHz in the previous module would result in a video output where the block travels downward very fast that players would not be able to see much.

Linecheck

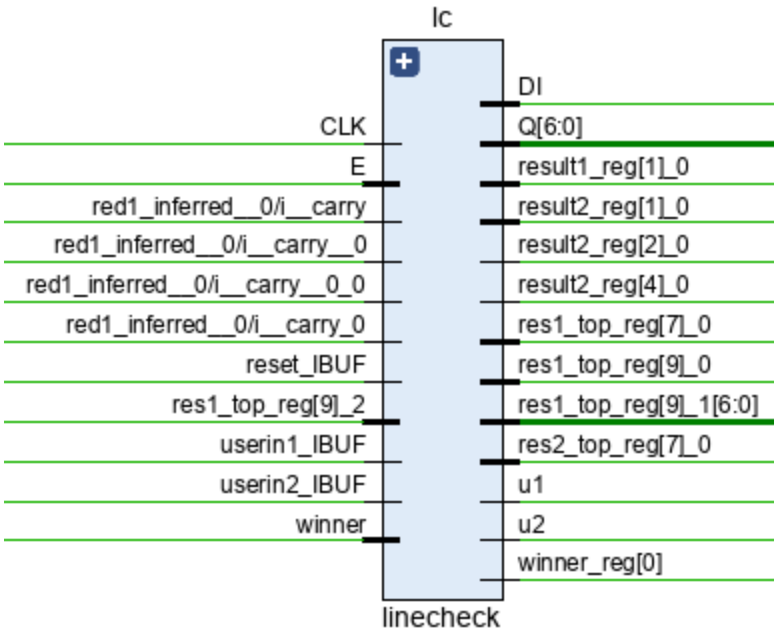

Figure 4. Linecheck module

In order to judge the accuracy of a note hit, a ‘linecheck’ module checks when the blocks’ bottom position is within the intended ‘hit’ zone of the game. This was set to a boundary of 50 pixels in height from 600 to 650 pixels horizontally. For every correct hit of a player, an internal register saves the score which would be used further down the circuit. Additionally, the module generates the dimensions of the scoring bar of each player. The scoring bar is a vertical bar that increases by 36 pixels for every correct hit made by a player. Both bars are 20 pixels wide and are situated on both ends of the screen for minimal distraction. With this, the module takes in clock, reset, user inputs, and current y-coordinates of both boxes as the inputs. It outputs the scores of the players, as well as the coordinates of the vertical scoring bar for both players.

Winner

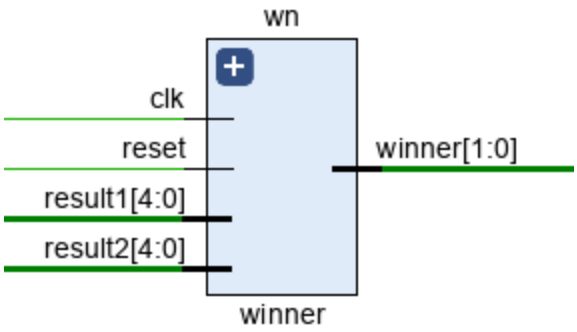

Figure 5. Winner module

In order to determine the winner, a separate module was created. It determines if a player’s score is already above the set scoring limit of 20. With this, it outputs a value of 0 when nobody is winning, 1 if player 1 has won, 2 if player 2 has won, and 3 when both players have achieved the limit simultaneously making it a tie. This was used to control the game state which is further used in the generation of the pixel in the VGA generator module.

Vga_timing

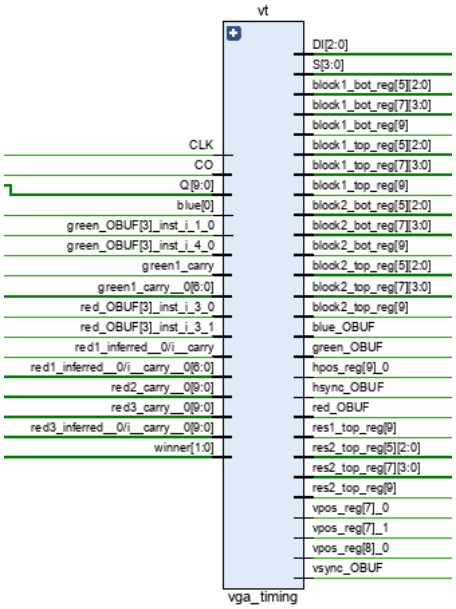

Figure 6. Vga_timing module

Video graphics arrays (VGA) light up individual pixels horizontally then vertically. Manufacturers' data sheets indicate the visible resolution when in reality, they implement a border at the ends of the display in order to synchronize the rate at which the vertical and horizontal pixels are changed. With this, the pixels change colors smoothly giving off an illusion that images change in real time when in fact the pixels of the screen change line by line. A module named 'vga_timing' keeps track of what horizontal and vertical pixels should be changed at a given clock pulse. This is utilized by the VGA generator module which directly interfaces with the monitor used by the FPGA.

vga_gen

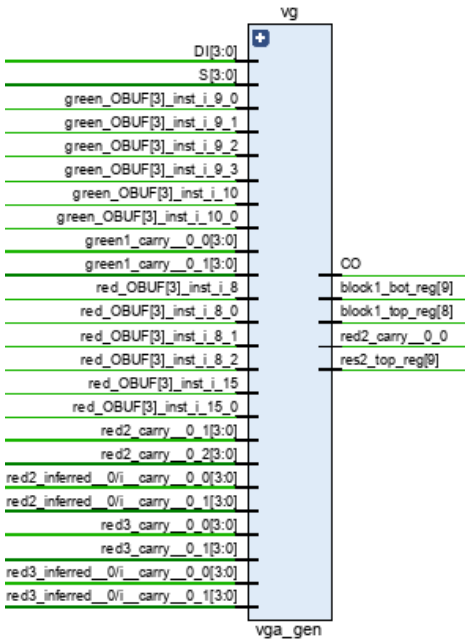

Figure 7. Vga_gen module

In order to properly interface with the monitor to be connected to the FPGA, a module ‘vga_gen’ was used. It is controlled by a clock pulse in order to go through each pixel one by one. With this, the module dictates the color that would be assigned per pixel and generates the shapes that are used in the game. Its inputs are the current x and y pixel position and active state dictated by the vga_timing module, current game state dictated by the winner module, and current coordinates of both players’ blocks and scoring bars. It outputs the red, green, and blue intensities of each pixel as it is synchronized with the timing of the VGA.

V. Simulation Results

Blockmaster Testbenching

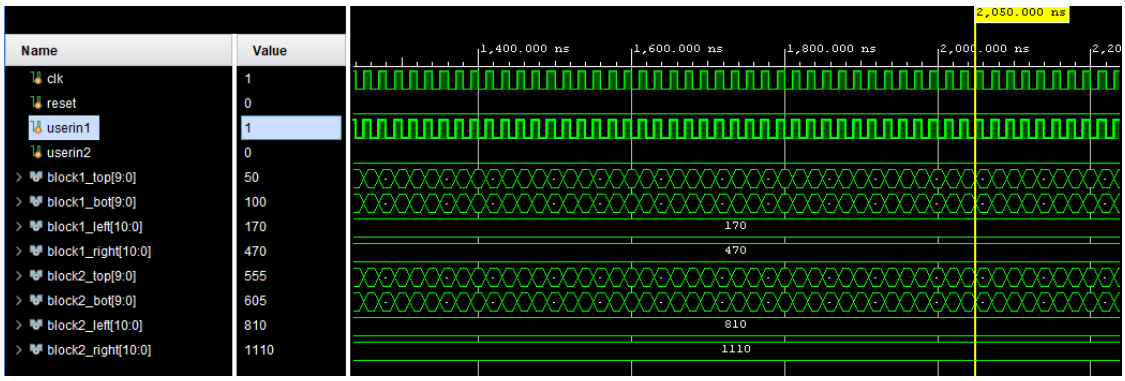

Figure 8. Resulting waveform for blockmaster testbench

The module ‘blockmaster’ was simulated with a clock pulse and an alternating input for just Player 1. The output of the testbench indicates that the top and bottom coordinates of both players’ blocks changes with an increment of 5 pixels per clock pulse as shown in Figure 8. Timing Player 1’s input when the bottom of his block has reached 600 pixels resets its position making it now asynchronous with Player 2’s block. With this, the module was proven to work given the behavioural simulation results.

Linecheck Test Benching

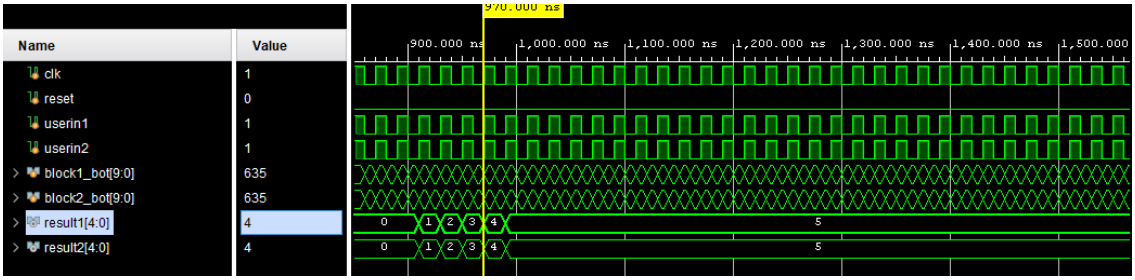

Figure 9. Resulting waveform for linecheck testbench

The module ‘linecheck’ was tested for functionality by creating a testbench of increasing block1 and block2 y-coordinates and pulsating inputs for both Player 1 and 2. With this, the module was able to properly count the score for both players once the y-coordinates of the blocks are between 600 to 650 pixels as shown in Figure 9. It must be noted that the multiple scores counted by the module when the block is within the indicated range is due to the fact that y-coordinate resetting function is not included in the

‘linecheck’ module but rather in the ‘blockmaster’ module thus incrementing the score multiple times while the block is within the range.

Winner Test Benching

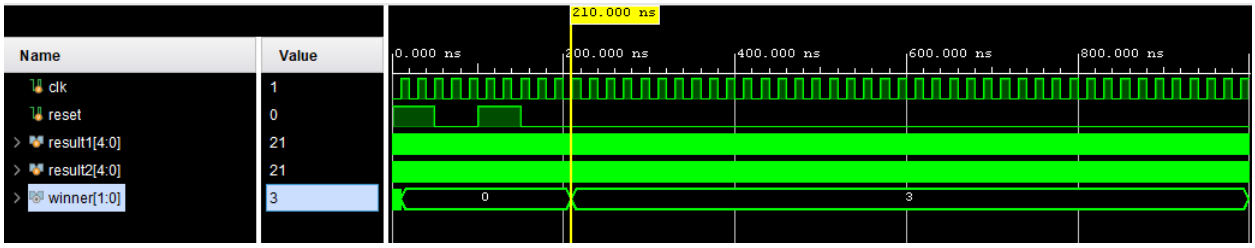

Figure 10. Resulting waveform for winner testbench

The module ‘winner’ was simulated by creating a testbench of increasing Player 1 and 2 scores. The resulting waveform indicates that the module was able to function properly by giving a game state of 0 when the scores are both less than 20 and a game state of 3 when both players are tied at a score more than 20.

Actual Implementation

The game controls were wired with Player 1’s input to the left push button onboard of the Zedboard, Player 2’s input to the right push button, and reset button to the center push button. In order to test the game, a monitor was utilized and connected to the VGA port of the FPGA. Programming the device from Vivado starts the game immediately as there are no loading or welcome screens included in the game.

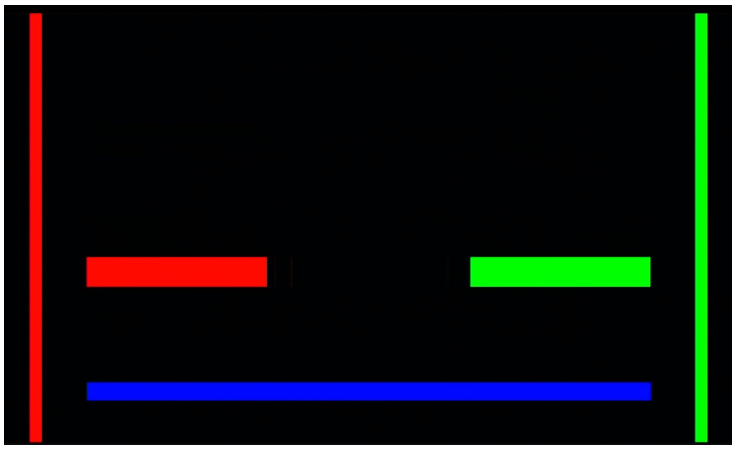

Figure 11. Initial synchronized state of the blocks

The game starts with both players' blocks synchronized. The vertical bar at both sides of the screen represents the score of the players. As shown in Figure 11, the vertical bar is already at max even in an initial state which was then fixed in the latest version of the game.

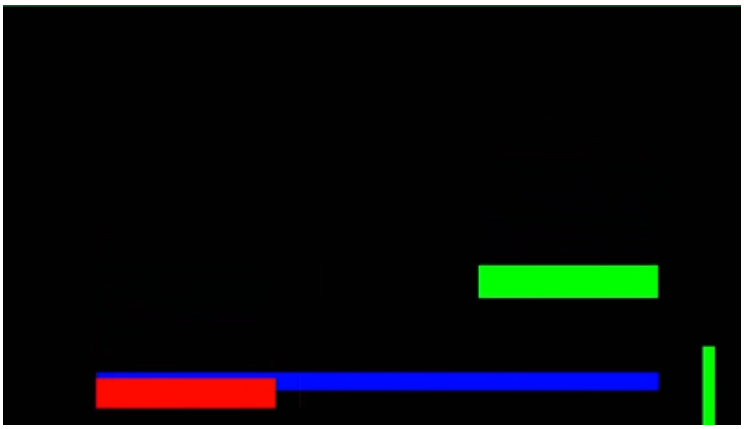

Figure 12. Out of sync blocks when a hit occurs

Hitting the button precisely when the block reaches the blue line resets its position immediately back to the top. This causes the blocks to be out of sync as shown in Figure 12. As the game progresses and there is an apparent difference in the players' reflexes, the blocks get more and more out of sync from each other, gaining the winning player a lead in the scoring system of the game.

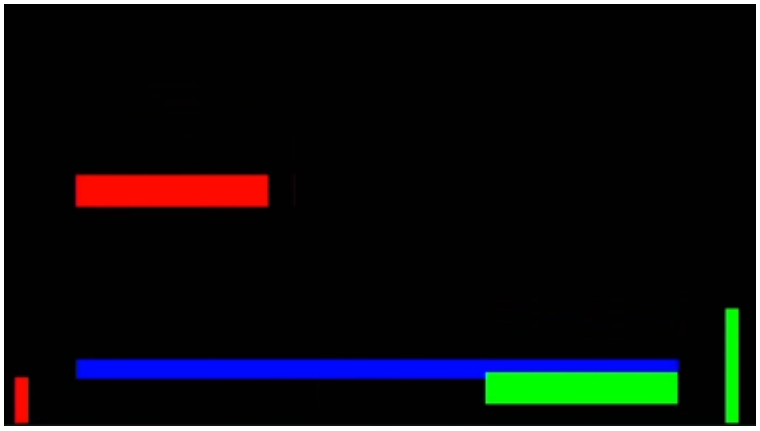

Figure 13. Increasing vertical score bars when a hit occurs

Hitting the note precisely when it touches the blue line increases the score of a player. This was visualized by having vertical score bars indicating the score as shown in Figure 13. Once a player reaches a score of 20 or the vertical bar reaches the top, the whole screen immediately turns into a solid color representing the winner. On the other hand, the whole screen turns yellow when both players have reached a score of 20 simultaneously - turning it into a tie.

VI. Conclusion

The group was able to implement a simple rhythm game using Verilog and programmed it into a Zedboard device. The game implements the basic concepts of a scrolling-type rhythm game such as accuracy of hitting a note when it passes a line threshold and a scoring system. It utilizes several modules that control the logic of the game as well as the creation of graphical components of the game. Each module was tested prior to the final implementation of the game which resulted in the expected behavior of each module. Lastly, the final implementation of the game was done on an FPGA which resulted into a basic rhythm game as seen from the results above.

The code of the game utilizes concepts that were taught in the previous classes. However, the graphical implementation and conceptualization of the project heavily relied on the members' experiences with a rhythm game. The created program was fully functional according to the members' expectations of a simple rhythm game. For future development of the game, future programmers can look into the improvement of the graphics. Based on the final implementation, there is a slight glitch on the display that may be caused by some slight error in the 'vga_timing' and 'vga_gen modules'. In addition to this, a randomizer and a preset timing info can be created in order to either randomize the timing of the blocks or mimic an actual music when hitting the notes. Lastly, further improvements in the gameplay such as the speed and other gameplay mechanics can be made by adjusting the parameters that were set in the code.

References

A. Jahanshahi, M. K. Taram and N. Eskandari, "Blokus Duo game on FPGA," *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, 2013, pp. 149-152, doi: 10.1109/CADS.2013.6714256.

C. H. Chen and C. S. Lo, "The development of a music rhythm game with a higher level of playability," *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, 2016, pp. 132-135, doi: 10.1109/ICAMSE.2016.7840256.

C. J. Jiménez-Fernández, C. B. Oliva, P. P. Fernández, A. G. Soto, F. E. P. Ordóñez and M. V. Barrero, "Learning VHDL through teamwork FPGA game design," *2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE)*, 2020, pp. 1-5, doi: 10.1109/TAEE46915.2020.9163756.

G. Wang, Y. Guan and Y. Zhang, "Designing of VGA Character String Display Module Based on FPGA," *2009 International Symposium on Intelligent Ubiquitous Computing and Education*, 2009, pp. 499-502, doi: 10.1109/IUCE.2009.12.

Song, D. H., Kim, K. B., & Lee, J. H. (2019). Analysis and evaluation of mobile rhythm games : Game structure and playability. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(6), 5263. https://doi.org/10.11591/ijece.v9i6.pp5263-5269

Appendix

clk_blk module

```
module clk_blk(
    input clk_in,
    output reg clk_slow
    );

    reg [19:0]count;
    always@(posedge clk_in)
    begin
        count <= count+1;
        if (count == 1_000_000)begin
        count <= 0;
        clk_slow <= ~clk_slow;
        end
    end

endmodule
```

blockmaster module

```
module blockmaster(
    input clk_blk,
    input reset,
    input userin1,
    input userin2,

    output reg [9:0] block1_top,
    output reg [9:0] block1_bot,
    output reg [10:0] block1_left, //width of 300px
    output reg [10:0] block1_right,

    output reg [9:0] block2_top,
    output reg [9:0] block2_bot,
    output reg [10:0] block2_left, //width of 300px
    output reg [10:0] block2_right
    );
    reg u1;
```

```
reg u2;
wire p1;
wire p2;

assign p1 = userin1 && ~u1;
assign p2 = userin2 && ~u2;

always@(posedge clk_blk)
begin
    u1 <= userin1;
    u2 <= userin2;

    if (reset||(p1&&((block1_bot>=600)&&(block1_bot<650)))) begin
        block1_bot <= 100;
        block1_top <= 50;
    end else begin
        if(block1_bot != 720)begin
            block1_bot <= block1_bot + 5;
            block1_top <= block1_top + 5;
        end else begin
            block1_bot <= 100;
            block1_top <= 50;
        end
    end



    if (reset||(p2&&((block2_bot>=600)&&(block2_bot<650)))) begin
        block2_bot <= 100;
        block2_top <= 50;
    end else begin
        if(block2_bot != 720)begin
            block2_bot <= block2_bot + 5;
            block2_top <= block2_top + 5;
        end else begin
            block2_bot <= 100;
            block2_top <= 50;
        end
    end
```

```
    end
endmodule
```

linecheck module

```
module linecheck(
    input clk,
    input reset,
    input userin1,
    input userin2,
    input [9:0] block1_bot,
    input [9:0] block2_bot,
    output reg [4:0] result1,
    output reg [4:0] result2,

    output reg [9:0] res1_top,
    output reg [9:0] res1_bot,
    output reg [10:0] res1_left,  //20px width scoring
    output reg [10:0] res1_right,

    output reg [9:0] res2_top,
    output reg [9:0] res2_bot,
    output reg [10:0] res2_left, //20px width scoring visual
    output reg [10:0] res2_right

    );
    reg u1;
    reg u2;
    wire p1;
    wire p2;

    assign p1 = userin1 && ~u1;
    assign p2 = userin2 && ~u2;

    always@(posedge clk)
    begin
        u1 <= userin1;
```

```
        u2 <= userin2;

        if (reset)begin
            result1 <= 0;
            result2 <= 0;
            res1_top <= 720;
            res2_top <= 720;
        end else begin

            if(p1 && ((block1_bot<650)&&(block1_bot>=600))) begin
                result1 <= result1 +1;
                res1_top <= 720-(36*result1);
            end

            if(p2 && ((block2_bot<650)&&(block2_bot>=600))) begin
                result2 <= result2 +1;
                res2_top <= 720-(36*result2);
            end

        end
    end
endmodule
```

winner module

```
module winner(
    input clk,
    input reset,
    input [4:0] result1,
    input [4:0] result2,
    output reg [1:0] winner
    );

    always@(posedge clk)
    begin
        if(reset)begin

            winner <= 0;
```

```
        end else begin

            if ((result1 >= 20) && (result1 > result2))begin
                winner <= 1;
            end else if ((result2 >= 20) && (result2 > result1))begin
                winner <= 2;
            end else if ((result1 >= 20) && (result1 == result2)
)begin
                winner <= 3;
            end

        end
    end
endmodule
```

vga_timing module

```
module vga_timing(
    input clk,
    output reg hsync,
    output reg vsync,
    output reg active,
    output wire [10:0] x, // 0 to 1279 (max of 2047)
    output wire [9:0] y   // 0 to 719  (max of 1023)
);
    parameter total_width = 1650;
    parameter total_height = 750;

    parameter h_active = 1280;
    parameter h_front_porch = 110;
    parameter h_back_porch = 220;
    parameter h_sync_pulse = 40;

    parameter v_active = 720;
    parameter v_front_porch = 5;
    parameter v_back_porch = 20;
    parameter v_sync_pulse = 5;

    reg [10:0] hpos; // 0 to 1649
```

```
reg [9:0] vpos;  // 0 to 749

reg next_line;

always @(posedge clk)
begin
    if (hpos >= total_width-1) begin
        hpos <= 0;
        next_line <= 1;
    end else begin
        hpos <= hpos + 1;
        next_line <= 0;
    end
end

always @(posedge clk)
begin
    if (next_line) begin
        if (vpos >= total_height-1) begin
            vpos <= 0;
        end else begin
            vpos <= vpos + 1;
        end
    end else begin
        vpos <= vpos;
    end
end

// Horizontal Sync
always @*
begin
    if (hpos >= (h_back_porch + h_active + h_front_porch)) begin
        hsync <= 0;
    end else begin
        hsync <= 1;
    end
end

// Vertical Sync
```

```
    always @*
    begin
        if (vpos >= (v_back_porch + v_active + v_front_porch)) begin
            vsync <= 0;
        end else begin
            vsync <= 1;
        end
    end

    // Active
    always @*
    begin
        if (vpos >= v_back_porch && vpos < (v_back_porch + v_active)
&& hpos >= h_back_porch && hpos < (h_back_porch + h_active)) begin
            active <= 1;
        end else begin
            active <= 0;
        end
    end

    assign x = active ? (hpos - h_back_porch) : 0;
    assign y = active ? (vpos - v_back_porch) : 0;
endmodule
```

vga_gen module

```
module vga_gen
#(
   parameter RES1_LEFT = 75,
   parameter RES1_RIGHT = 95,
   parameter RES1_BOT = 720,
   parameter RES2_LEFT = 1185,
   parameter RES2_RIGHT = 1205,
   parameter RES2_BOT = 720,

   parameter BLOCK1_LEFT = 170,
   parameter BLOCK1_RIGHT = 470,
   parameter BLOCK2_LEFT = 810,
   parameter BLOCK2_RIGHT = 1110
```

```
)
(
    input [10:0] x,
    input [9:0] y,
    input active,
    input [1:0] winner, //result from game logic

    input [9:0] block1_top,   //blockcoordinates
    input [9:0] block1_bot,
    input [10:0] block1_left,
    input [10:0] block1_right,

    input [9:0] block2_top,    //blockcoordinates
    input [9:0] block2_bot,
    input [10:0] block2_left,
    input [10:0] block2_right,

    input [9:0] res1_top,       //scoring visual coordinates
    input [9:0] res1_bot,
    input [10:0] res1_left,
    input [10:0] res1_right,

    input [9:0] res2_top,       //scoring visual coordinates
    input [9:0] res2_bot,
    input [10:0] res2_left,
    input [10:0] res2_right,

    output reg [3:0] red,
    output reg [3:0] green,
    output reg [3:0] blue);


    always @*
    begin
        if (active) begin

           if(winner==0)begin
            //block1
                if (x >= BLOCK1_LEFT && x < BLOCK1_RIGHT && y >=
```

```
block1_top && y < block1_bot) begin

                        red <= 4'b1111;
                        green <= 4'b0000;
                        blue <= 4'b0000;

                end else if (x >= BLOCK2_LEFT && x < BLOCK2_RIGHT &&
y >= block2_top && y < block2_bot)begin

                        red <= 4'b0000;
                        green <= 4'b1111;
                        blue <= 4'b0000;

                end else if (x >= BLOCK1_LEFT && x < BLOCK2_RIGHT &&
y >= 620 && y < 650)begin
                        red <= 4'b0000;
                        green <= 4'b0000;
                        blue <= 4'b1111;

                end else if (x >= RES1_LEFT && x < RES1_RIGHT && y >=
res1_top)begin
                        red <= 4'b1111;
                        green <= 4'b0000;
                        blue <= 4'b0000;

                end else if (x >= RES2_LEFT && x < RES2_RIGHT && y >=
res2_top)begin
                        red <= 4'b0000;
                        green <= 4'b1111;
                        blue <= 4'b0000;

                end else begin
                        red <= 0;
                        green <= 0;
                        blue <= 0;
                end


            end else if (winner==1)begin
```

```
    if (x >= 0 && x < 1280 && y >= 0 && y < 720) begin

            red <= 4'b1111;
            green <= 4'b0000;
            blue <= 4'b0000;

     end else begin
            red <= 0;
            green <= 0;
            blue <= 0;
     end


end else if (winner==2)begin

    if (x >= 0 && x < 1280 && y >= 0 && y < 720) begin

            red <= 4'b0000;
            green <= 4'b1111;
            blue <= 4'b0000;
     end else begin
            red <= 0;
            green <= 0;
            blue <= 0;
     end

end else begin
    if (x >= 0 && x < 1280 && y >= 0 && y < 720) begin

            red <= 4'b1111;
            green <= 4'b1111;
            blue <= 4'b0000;

     end else begin
            red <= 0;
            green <= 0;
            blue <= 0;
     end
```

```
            end



        end else begin
            red <= 0;
            green <= 0;
            blue <= 0;
        end
    end
endmodule
```

topmodule

```
module topmodule(
    input clk_100MHz,
    input reset,
    input userin1,
    input userin2,
    output wire hsync,
    output wire vsync,
    output wire [3:0] red,
    output wire [3:0] green,
    output wire [3:0] blue
    );

    wire clk_74_25MHz;
    wire clk_slow;
    wire active;
    wire [10:0] xpos;
    wire [9:0] ypos;

    wire [9:0] block1_top;
    wire [9:0] block1_bot;
    wire [10:0] block1_left;
    wire [10:0] block1_right;
```

```
wire [9:0] block2_top;
wire [9:0] block2_bot;
wire [10:0] block2_left;
wire [10:0] block2_right;

wire [9:0] res1_top;
wire [9:0] res1_bot;
wire [10:0] res1_left;
wire [10:0] res1_right;

wire [9:0] res2_top;
wire [9:0] res2_bot;
wire [10:0] res2_left;
wire [10:0] res2_right;

wire [4:0] result1;
wire [4:0] result2;
wire [1:0] winner;


vga_gen vg(.x(xpos),
           .y(ypos),
           .active(active),
           .winner(winner),

           .block1_top(block1_top),
           .block1_bot(block1_bot),
           .block1_left(block1_left),
           .block1_right(block1_right),

           .block2_top(block2_top),
           .block2_bot(block2_bot),
           .block2_left(block2_left),
           .block2_right(block2_right),

           .res1_top(res1_top),
           .res1_bot(res1_bot),
           .res1_left(res1_left),
```

```
            .res1_right(res1_right),

            .res2_top(res2_top),
            .res2_bot(res2_bot),
            .res2_left(res2_left),
            .res2_right(res2_right),

            .red(red),
            .green(green),
            .blue(blue));

winner wn(.clk(clk_74_25MHz),
          .reset(reset),
          .result1(result1),
          .result2(result2),
          .winner(winner));


linecheck lc(.clk(clk_slow),  // originally 74.25MHz
             .reset(reset),
             .userin1(userin1),
             .userin2(userin2),
             .block1_bot(block1_bot),
             .block2_bot(block2_bot),
             .result1(result1),
             .result2(result2),

             .res1_top(res1_top),
             .res1_bot(res1_bot),
             .res1_left(res1_left),
             .res1_right(res1_right),

             .res2_top(res2_top),
             .res2_bot(res2_bot),
             .res2_left(res2_left),
             .res2_right(res2_right));

blockmaster bm(.clk_blk(clk_slow),
               .reset(reset),
```

```
                    .userin1(userin1),
                    .userin2(userin2),
                    .block1_top(block1_top),
                    .block1_bot(block1_bot),
                    .block1_left(block1_left),
                    .block1_right(block1_right),

                    .block2_top(block2_top),
                    .block2_bot(block2_bot),
                    .block2_left(block2_left),
                    .block2_right(block2_right));

    vga_timing vt(.clk(clk_74_25MHz),
                  .vsync(vsync),
                  .hsync(hsync),
                  .active(active),
                  .x(xpos),
                  .y(ypos));

    clk_wiz_0 cw(.clk_in1(clk_100MHz),
                 .clk_out1(clk_74_25MHz));

    clk_blk cb(.clk_in(clk_100MHz),
               .clk_slow(clk_slow));
endmodule
```