Group 31

Andrei Purcarus (260631911)

Vlastimil Lacina (260749245)

g31_Breakout_Game

System Features

The g31_Breakout_Game logic system allows a user to play a version of Breakout, a game in which the player must move a paddle horizontally to bounce a ball upwards and break colored blocks. The player starts with 5 lives and loses one every time the ball falls without being bounced back by the paddle. When all lives are lost, the player loses. There are 7 levels that the player must complete, and the player has a score to keep track of how well they are doing. This implementation features several improvements over the basic concept:

- The ball bounces at different angles depending on where it hits the paddle. The ball will bounce to the right if it hits the right side of the paddle, and to the left if it hits the left side. In addition, the closer the ball is to the edge, the larger the angle of bounce will be with respect to a line normal to the paddle. This means hitting the ball with the side of the paddle makes it move faster horizontally than vertically.
- Three types of blocks are available: Single blocks require one hit to break, double blocks require two hits, and unbreakable blocks are invulnerable.
- Each level features a different arrangement of blocks, which may use up to 8 rows instead of the original 5 rows. In addition, the ball speeds up slightly as the level increases, and each block destroyed gives out points equal to the current level. This means higher levels get both harder and more rewarding in terms of score.
- The player can reset the game to level 1 by pressing a button.
- The game waits for user input before progressing when a major event occurs, such as completing a level, winning or losing the game, and resetting the game.
- The player can pause the game to take breaks.
- The player can enable cheat/debug mode. This creates an invisible wall below the player, preventing them from losing. In addition, the ball moves at 16 times its normal speed. This allows for quick clearing of levels for debugging purposes. However, this mode might not be able to clear a level alone due to the ball not being able to hit some blocks. In these cases, the player must use the paddle to help the ball hit these last blocks.

These improvements allow for a much more enjoyable playing experience.

Block Diagrams

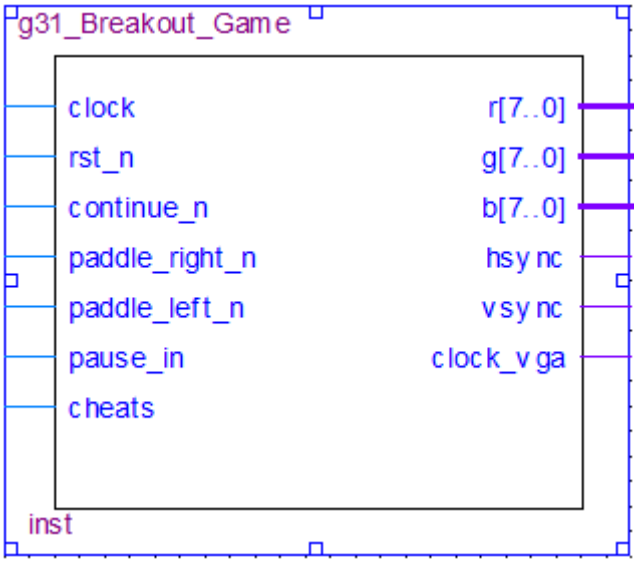

Figure 1. g31_Breakout_Game block diagram.

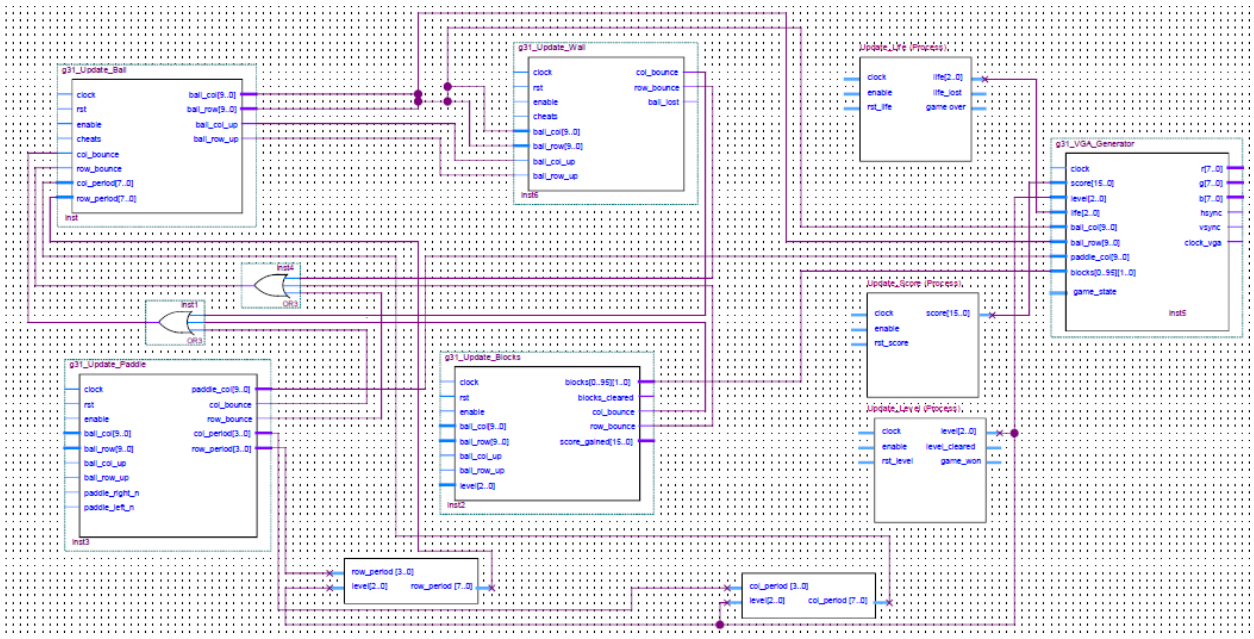

Figure 2. g31_Breakout_Game entity and process diagram.

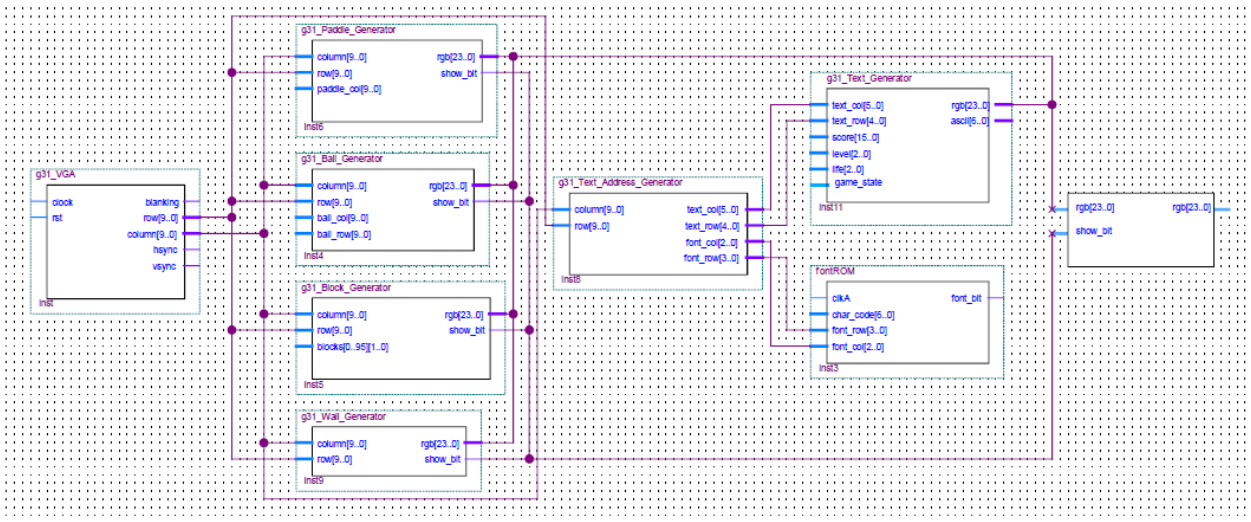

Figure 3. g31_VGA_Generator entity and process diagram.

Circuit Description

The g31_Breakout_Game circuit uses a clock and six user inputs to produce a VGA signal as output, which consists of a 24-bit RGB color, HSYNC and VSYNC signals, and an output clock signal. The user inputs are described in the next section. A block diagram of this circuit is shown in Figure 1.

The implementation of the circuit is partially shown in Figure 2. Not shown is a start-up circuit which generates two reset pulses when the system boots. This ensures that the system resets itself at the start of the game, so the user can start playing immediately. The core of the circuit consists of the g31_Update_Ball, g31_Update_Wall, g31_Update_Blocks, and g31_Update_Paddle entities. The ball position is stored in g31_Update_Ball, which takes as an input the col_bounce and row_bounce bits. These bits tell the circuit to reverse the direction of the ball, and they are set by the other three entities through an OR gate. This entity also takes as input the relative period for the col and row counters from the output of g31_Update_Paddle. That output is multiplied by 16 – level/2 to increase the speed with increasing levels. The g31_Update_Wall entity takes the ball position and direction as inputs, and checks if the ball should bounce off a wall or fall. It outputs the appropriate bounce signals and a ball_lost signal to indicate that the ball has fallen. The g31_Update_Blocks entity keeps track of the positions and types of the blocks. It takes as input the current level, which it uses to load the block configuration for that level from a ROM when it resets. It should be noted that the blocks are stored as an array of 2-bit std_logic_vectors, to allow for 4 types of blocks (none, single, double, and unbreakable). This entity also takes as input the current ball position and direction and checks if any blocks have been hit. If so, it updates the block array and outputs the appropriate bounce signals. It also outputs the score that has been gained this clock cycle from destroying blocks. In the current implementation, one point is awarded per block. It also outputs blocks_cleared, which signals that all blocks have been destroyed and the game can progress. The g31_Update_Paddle entity keeps track of the col position of the paddle. It takes as inputs the ball position and direction, and checks if it has hit the paddle. If so, it computes the position where it has hit, and outputs the appropriate bounce signals, as well as the relative periods for the col and row direction to control the angle of the bounce. This entity also takes an input the asynchronous paddle_right_n and paddle_left_n signals, which it latches until the next paddle position update. It uses these latched signals to move the paddle left and right.

Out of this core come three signal which progress the game, and several other signals which describe the positions of the game elements. The positions are input to the g31_VGA_Generator entity, which is described later. The other three signals, score_gained, blocks_cleared, and ball_lost, are used to update the game state with the Update_Score, Update_Level, and Update_Life process blocks. The Update_Score process takes score_gained and adds it to the current score. The Update_Level process takes blocks_cleared and either updates the level and signals level_cleared or signals that the game has been won through the game_won signal. The Update_Life process takes ball_lost and either subtracts one from the number of lives and signals life_lost or signals game_over when no lives are left.

The various reset signals are generated through an OR gate with inputs the signals which should reset that component. For example, the rst_blocks signal is generated when rst, game_won, game_over, or level_cleared is signaled. It should be noted that game_won and game_over are both delayed using a register to preserve the game state for the player to observe. Parallel to the reset signals is an enable signal which goes low when a reset is triggered, stopping the game from progressing until the player presses the continue button. An additional process not shown keeps track of the game state. It outputs a game_state signal to the g31_VGA_Generator to indicate if the state is playing, paused, reset, level_cleared, game_over, and game_won.

The g31_VGA_Generator circuit component diagram is shown in Figure 3. This circuit contains the status bar generation circuitry described in previous labs. However, the g31_Text_Generator circuit has been supplemented with a game_state input, which it uses to output a message to the player below the blocks and above the paddle. The RGB signal and font_bit are input to a multiplexer to select a VGA output. Several other entities are used to generate additional RGB and show_bit signals for the multiplexer. The g31_Ball_Generator, g31_Wall_Generator, and g31_Paddle_Generator entities check the column and row and output the appropriate color to display. The g31_Block_Generator entity does the same, but outputs a different color depending on the row of the block, and the type of block.

The VHDL description files are included with this report.

User Interface

The user can operate the game with two slide switches and four push-buttons. Their operation is described below.

Push-Buttons:

KEY0 – Move the paddle right.

KEY1 – Move the paddle left.

KEY2 – Continue the game if it has been paused.

KEY3 – Reset the game to its initial state.

Slide switches:

SW9 – Pauses and un-pauses the game.

SW8 – Turns cheat/debug mode on and off.

Discussions

The system was tested by programming the Cyclone V FPGA with the game and confirming the correct functionality of the system. The VGA output was connected to a monitor and several screenshots were taken to confirm the correct operation of the system. Figure 4 shows the state of the game when the game starts up or when the reset key is pressed. The game is frozen until the continue key is pressed. Figure 5 shows a screenshot of level 3. It shows that the game is paused after the completion of a level and that the blocks are arranged differently with every level. Figure 6 shows a screenshot of level 5. It shows single blocks, double blocks, and unbreakable blocks all in the same level. Figures 7 and 8 show screenshots of level 7, both at the start and during operation of the game. Figure 8 shows that the player has paused the game. Figure 9 shows a screenshot of the completed game. It shows that the final score is 1680, as expected. It also shows that the game stops for the player to admire his completion of the game. Finally, Figure 10 shows a screenshot of the player losing the game. It shows that the game pauses in its final state for the player to contemplate their loss and learn from their mistakes.

Figure 4. Screenshot of the game at start-up.

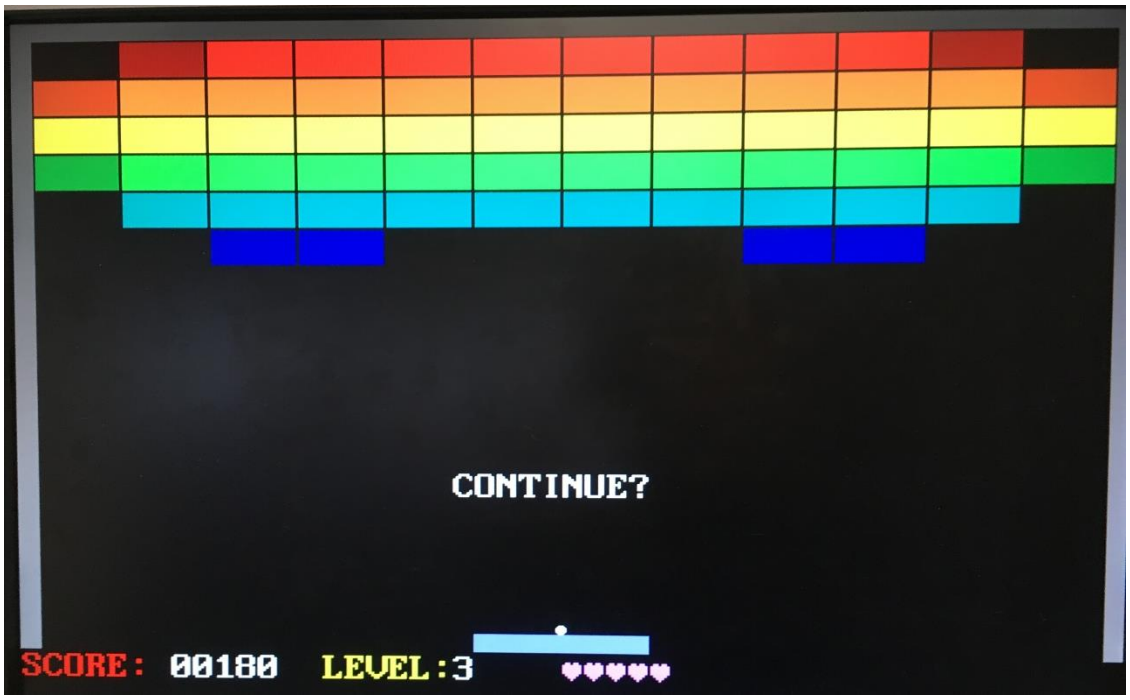

Figure 5. Screenshot of the game at level 3.

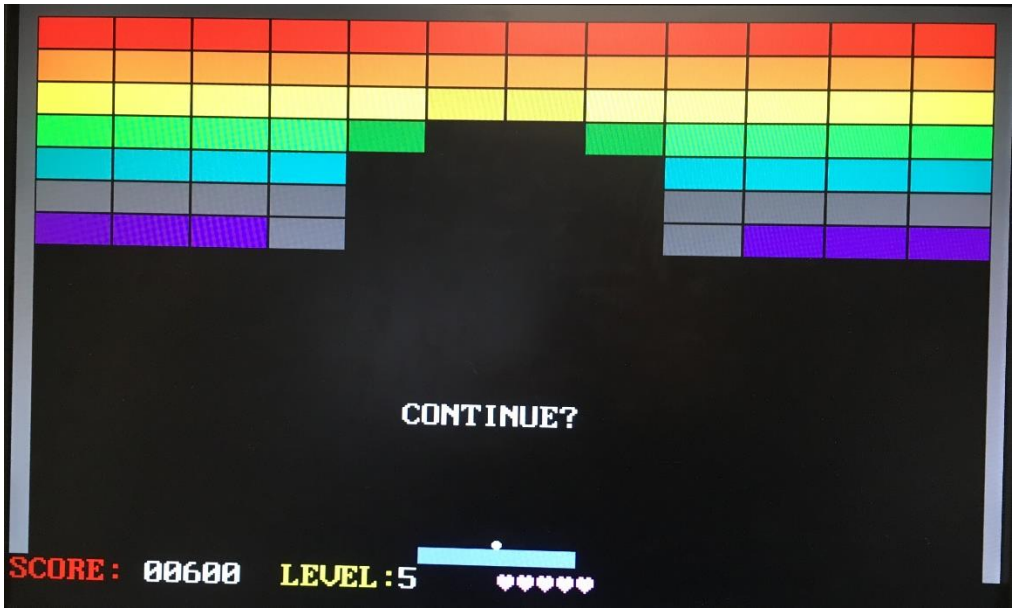

Figure 6. Screenshot of the game at level 5.

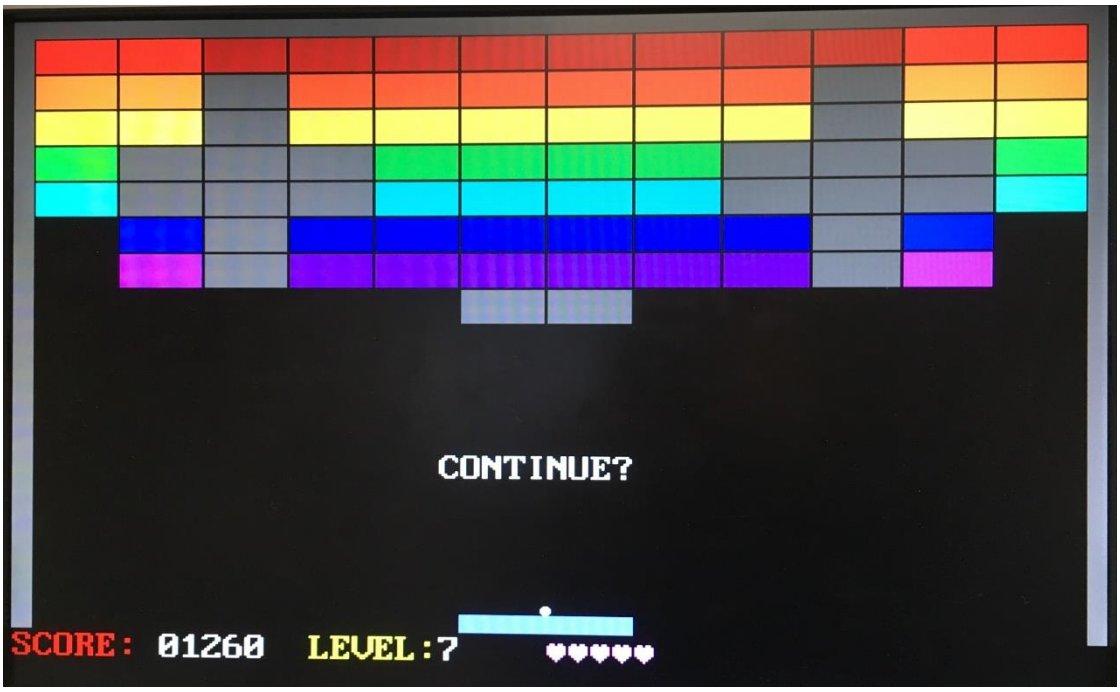

Figure 7. Screenshot of the game at level 7.

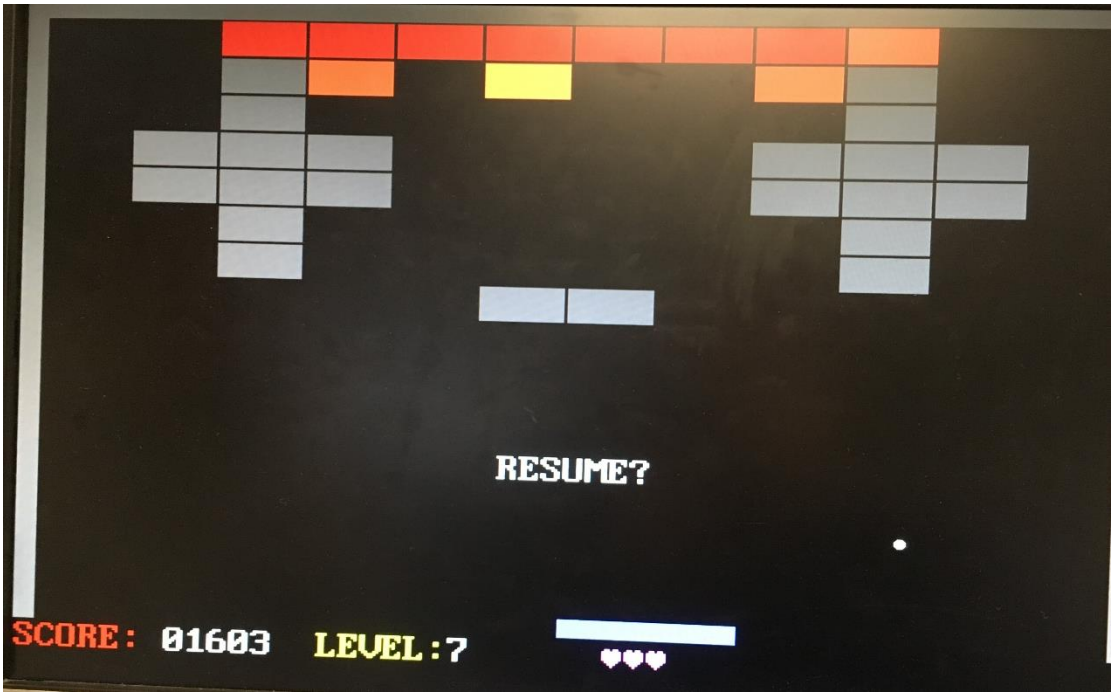

Figure 8. Screenshot of the player pausing the game at level 7.

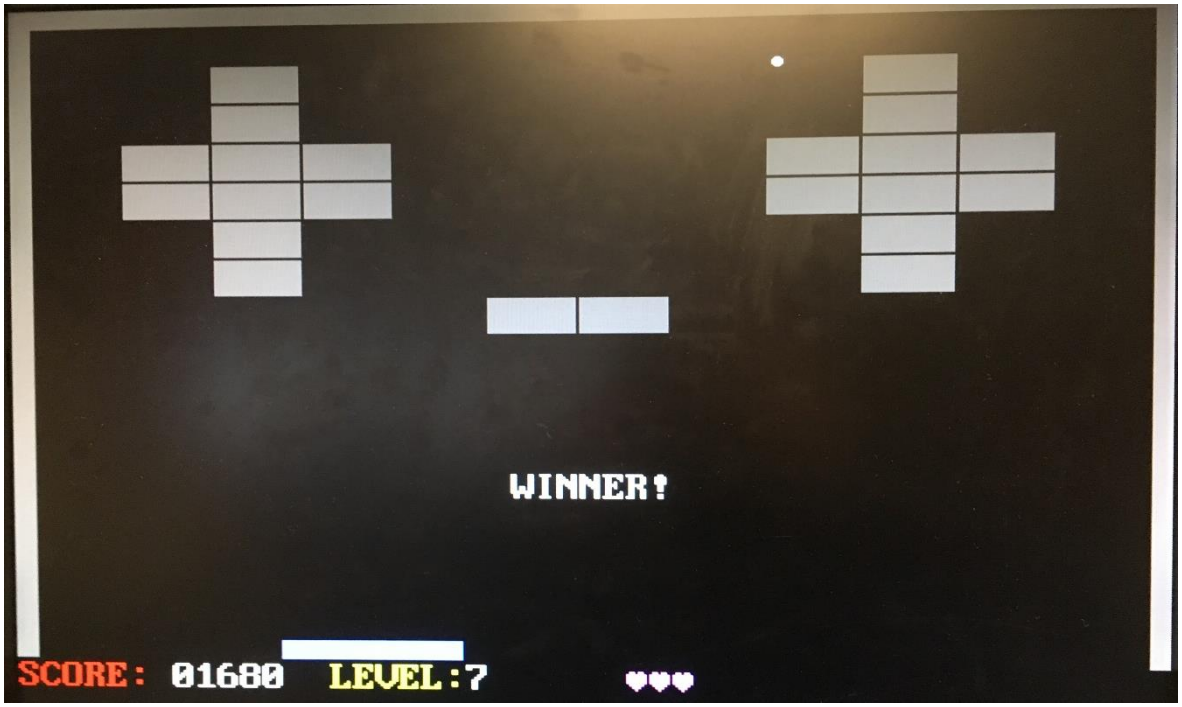

Figure 9. Screenshot of the player completing the game.

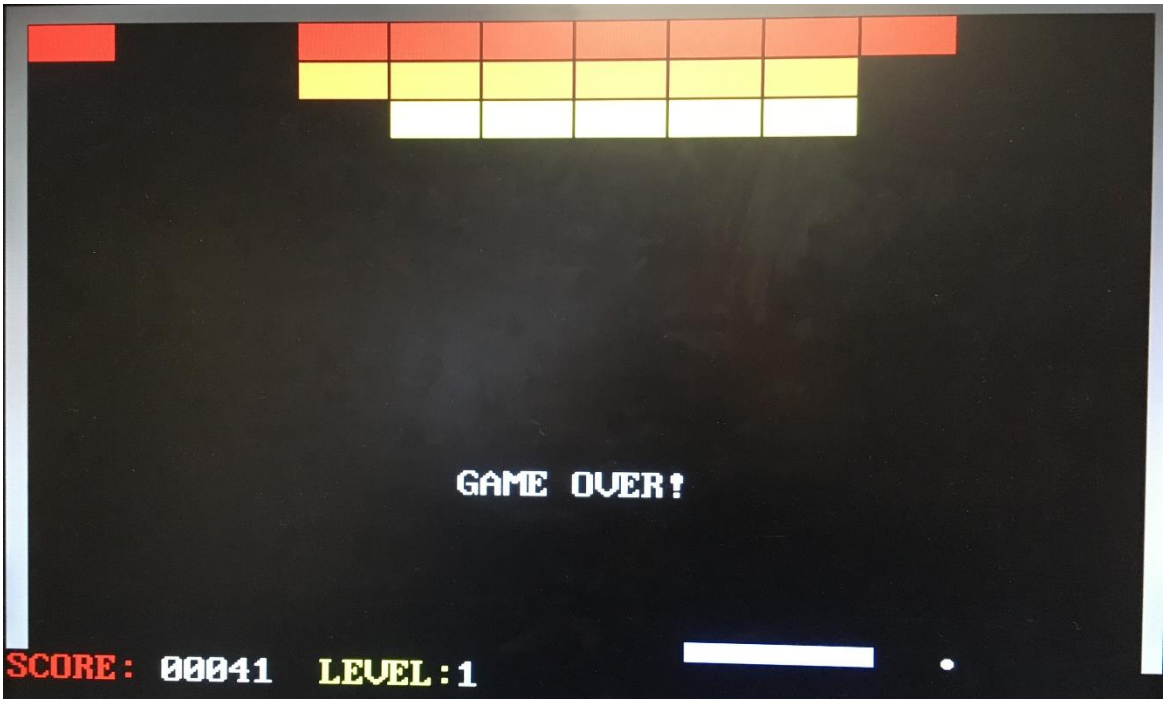

Figure 10. Screenshot of the player losing all their lives and the game.

Resource Utilization

The summary of FPGA resources used by the circuit is shown in Table 1. It should be noted that only 6% of the total available logic was used to implement the system.

Table 1. FPGA Resource Utilization

Device model	5CSEMA5F31C6
Logic utilization (in ALMs)	1,954/ 32,070 (6%)
Total registers	1310
Total pins	34/457 (7%)
Total block memory bits	73,728/4,065,280
Total RAM Blocks	8/397 (2%)
Total DSP Blocks	3/87 (3%)

Timing Analysis

The summary of the TimeQuest timing analysis is shown in Table 2. It should be noted that all slack values are positive and that Fmax is greater than 50MHz, indicating that the system has no timing problems.

Table 2. TimeQuest Timing Analysis

Fast Model Hold Slack	0.130 ns
Slow Model Setup Slack	0.696 ns
Slow Model Fmax	51.8 MHz

Conclusions

Four areas of the design were the causes of problems. The first was the g31_Update_Blocks entity. This entity computes 4 boundary coordinates for the next position of the ball, and checks each one for a block. Initially, this was all done sequentially in the same process block. However, this caused problems with the setup slack times due to how long it took. To solve the problem, some intermediate computations were taken out of the process block and done in parallel. Then, these computations were stored in a register, thus cutting the setup path, and reducing the time delay between registers. This made the operation of checking for blocks a two clock cycle process, and the rest of the entity was updated to reflect that.

A second problem came from system start-up. Initially, the system would start executing from the moment the boot ended, which meant that the ball would start bouncing before the VGA monitor could show it. A reset pulse generator was added to reset the system, but one pulse was not enough. The system still had start-up issues with only one. Finally, a second pulse was added and the system now starts up correctly.

A third problem came from the asynchronous nature of the button inputs. The reset button would sometimes cause the game to pause but not reset. To fix this, the reset button was latched and made to reinitialize the start-up circuitry, which generates two reset pulses. The other buttons were also latched for smoother operation and to reduce switch bounce.

A fourth problem came from the different timings of the core system. After the blocks had been updated, the block update circuitry would try to update it again, before the ball has had time to update. In addition, block updating takes two clock cycles and ball updating takes one. This creates the problem that multiple updates would be executed on the same inputs. To fix this, delays and waits were added to each component as needed to ensure only one update would take place. This is one area where further improvement is possible. By using a data path-controller architecture, it would be possible to add a start signal to each component and use a controller to direct them, thus ensuring multiple updates are never a problem without having to make the components know about each other's delays.

The design could also be improved by adding new features. The addition of power-ups would make the game a lot more enjoyable. These could increase or decrease the paddle length, or speed the ball up or down. In addition, a different scoring system could be used depending on the color of the block. This could be made better by the expansion of the blocks array to also include color, points, and power-up information.

Appendix

Grade Sheet for Lab #5

Fall 2016.

Group Number: 31

Group Member Name: Andrei Purcarus Student Number: 260631911

Group Member Name: [illegible] Student Number: [illegible]

Marks			TA Signatures
2	1.	VHDL Description of the walls and moving ball	[illegible]
2	2.	Demonstration of the walls and moving ball	[illegible]
2	3.	VHDL description of the blocks	Ali
2	4.	Demonstration of the blocks and block breaking	Ali
2	5.	VHDL for the moving paddle	Ali
2	6.	Demonstration of the moving paddle	Ali
2	7.	Demonstration of the complete breakout game	Ali

Each part should be demonstrated to one of the TAs who will then give a grade and sign the grade sheet. Grades for each part will be either 0, 1, or 2. A mark of 2 will be given if everything is done correctly. A grade of 1 will be given if there are significant problems, but an attempt was made. A grade of 0 will be given for parts that were not done at all, or for which there is no TA signature.