Acid Skate Apocalypse - Final Report EDA385

Martin Gunnarsson, dat11mgu@student.lu.se
Rickard Johansson, dat11rjo@student.lu.se
Thomas Strahl, dat11tst@student.lu.se

November 5, 2015

Abstract

In the course EDA385 an embedded system was to be designed and implemented. This report covers the design and implementation of a simple arcade game where the user can maneuver a skater to avoid obstacles. The game uses an accelerometer for user input and displays video through VGA. The design features a Microblaze CPU, a VGA controller and a accelerometer controller. During the project the design was changed and some problems were encountered, but in the end a complete game was produced.

Contents

1 Introduction

The purpose of the course *Design of Embedded Systems, Advanced Course (EDA385)[1]* is to get a better understanding of embedded system design and an insight to industry methods using both hardware and software.

In this project a simple arcade game was designed from a hardware and software perspective, then executed during the length of the course. The final design was constructed from slightly modified existing Intellectual Properties (IP:s) and ”custom” IP:s that were written by the group members.

2 Description

The system created in this project was a simple arcade game where the user can maneuver a ”skater” to avoid obstacles moving down the screen. By tilting an accelerometer to the left or right the skater can be controlled. Hitting a obstacle would end the game. After a short pause a new game would be started. The original ”setup” of the system was that the accelerometer was mounted on a skateboard without wheels. However the game could also be played by just tilting the accelerometer in the users hand.

The obstacles move at different speeds to make the game more challenging. To make the game more ”acid” the obstacles change color frequently. The skater's horizontal position is controlled and stopped just before the end of the screen so that it is not ”safe” from the obstacles at the end of the screen. The game also features so called ”power ups”, there are different kind of ”power ups”. Some of them are favorable for the player when others make the game much harder. What kind of effect the different power ups have is shown in the Appendix in subsection *10.1 Objects.*

The objective of the game is to get the highest score possible. The user gets one point per avoided obstacle and the total score is shown in the upper left corner. It is also possible to get points by collecting the ”power ups”.

The system is implemented on a *Nexys 3 Field-programmable gate array (FPGA) board* using a single core processor architecture with custom IP:s, the FPGA was a Spartan-6 (XC6LX16-CS324) FPGA [9]. The design tools used during the project were Xilinx design tools *Xilinx Platform Studio (XPS) [2]* and *ISE Project navigator [3]* for the hardware architecture and the VHDL code. For the C code *Xilinx Software Development Kit (SDK)* [6] was used.

2.1 Overview

The software part of the system is written in C. The software is executed on a Microblaze processor. The application stores the skater's position and generates ”power ups” and obstacles. An angle from the accelerometer is fetched using a *Fast Simplex Link (FSL)* channel and the position of the skater is changed. This new position of the skater is then sent to the *Video Graphics Array (VGA)* controller via a *Advanced eXtensible Interface (AXI)* bus. The VGA controller then draws the skater on the screen.

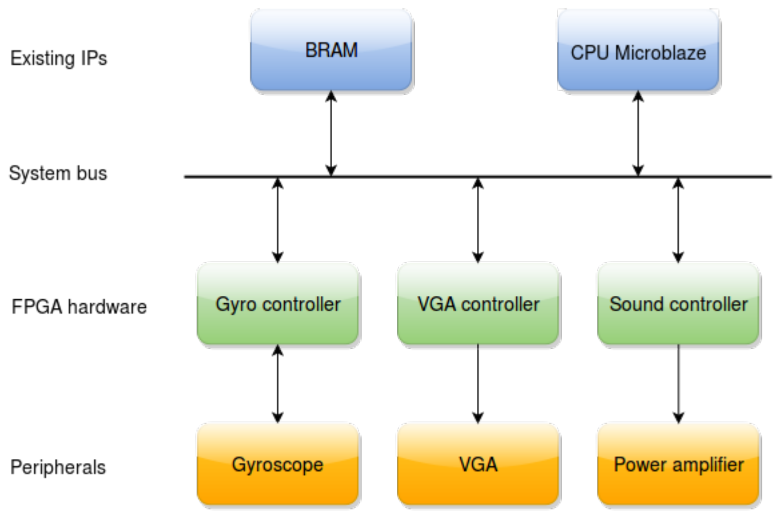


Figure 1: The initial proposal for the hardware setup of the system.

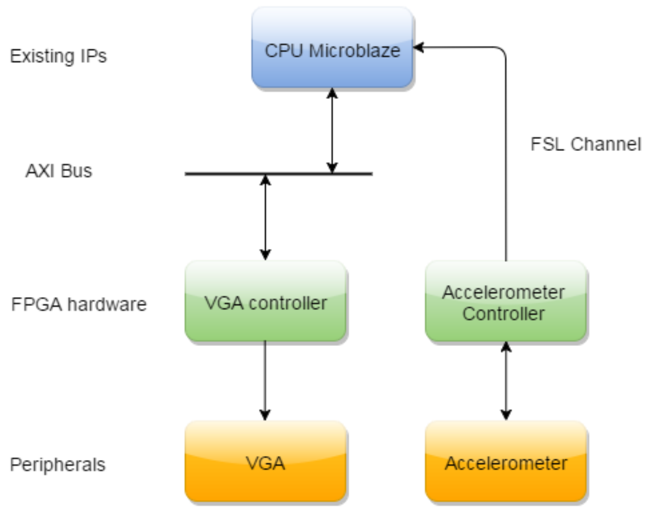


Figure 2: The actual hardware setup of the system.

In the initial proposal (*Figure 1*) it was decided to use a gyroscope to fetch an angle, but after some exploration of possibilities it was decided to use a accelerometer instead as can be seen in *Figure 2*. During the first weeks of the project many solutions to get the gyroscope to work were tried. One of them was a demo from *Digilent [7]* that was too complex for the application. The demo was also modified to fit the application better, but with no success. It was therefore decided to try the accelerometer instead. With some help from an old project from the same course from 2012 named *Labyrinth[5]*, the angle could successfully be received and it was therefore used in the final project. The original accelerometer code from the labyrinth group retrieved x-, y- and z-axis data. The application in this project only required x-axis data. The code was therefore modified to fit our needs.

The communication in the system also differs from the initial design (see *Figure 1* and *Figure 2*). The proposed design uses the AXI bus for all communication, but the current design uses FSL for the axis data from the accelerometer and AXI bus for the obstacles and skater positions. The reason for the change in design is that all of the members in the group had earlier experiences with FSL and it is also suitable for sending single digits.

3 Architecture

The core of the system is a *Microblaze[4]* CPU, an IP from *Xilinx*. It is configured for 32Mb of RAM and is clocked to 100MHz. The CPU is used to interface the other parts of the system and of course to run the software. User input is provided by the accelerometer and the game is displayed through the VGA controller. The accelerometer used is a *Digilent PmodACL accelerometer[8]* which is controlled by the accelerometer controller. The controller communicates with the accelerometer through *Serial Peripheral Interface* (SPI) and then passes on the numerical value to the CPU. The VGA controller is "custom" in the sense that it is only able to generate video for this application. The CPU gives directives to the VGA controller about where each object (e.g. skater, power up or obstacle) in the game is positioned and the color of the objects. The VGA controller uses that information to render the final image. The VGA controller is thus unable to generate anything else than blocks of different sizes.

3.1 Hardware

The FPGA is equipped with most of the hardware to begin with. The FPGA is used to synthesize the CPU, accelerometer controller and the VGA controller. The accelerometer is as previously stated an existing IP from *Digilent* that is controlled by modified code from an earlier project in this course.

3.1.1 Accelerometer

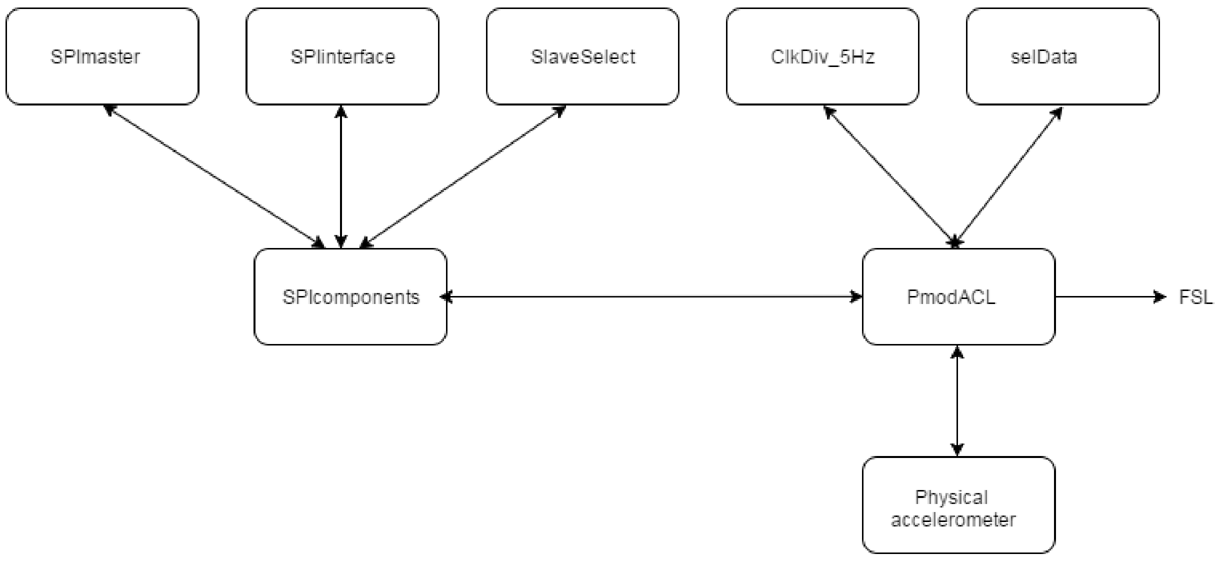


Figure 3: How the hardware modules and components are connected. The picture illustrates how accelerometer data is fetched from the physical accelerometer and then sent to the CPU via FSL.

The physical accelerometer from *Digilent* has support for three axes; x, y and z. When axis data is sent over FSL to the game logic, only the x axis is extracted and used for calculations. *Figure 3* shows how the VHDL code is connected and how axis data is sent to the CPU via FSL.

3.1.2 VGA

The VGA controller (*Figure 4*) is written by the group members and is implemented in VHDL. It uses the AXI bus to write values from the CPU to the screen. Furthermore the controller has a 100MHz clock input and a reset, it has outputs connected to the digital to analog converter that is connected to the VGA connector port. The color channels red, blue and green are represented as vectors with three bits for red and green, and two for blue. Eight bit colors enable 256 colors that can be displayed.

The VGA controller takes the 100MHz clock input and scales it down to 25MHz to be used as pixel clock. The pixel clock is used together with the horizontal and vertical synchronization signals to generate horizontal count and vertical count. The signals horizontal and vertical count are used to determine which pixel that is currently set to what color and sent to the output. A blanking logic signal is used to determine if a pixel is inside the visible area.

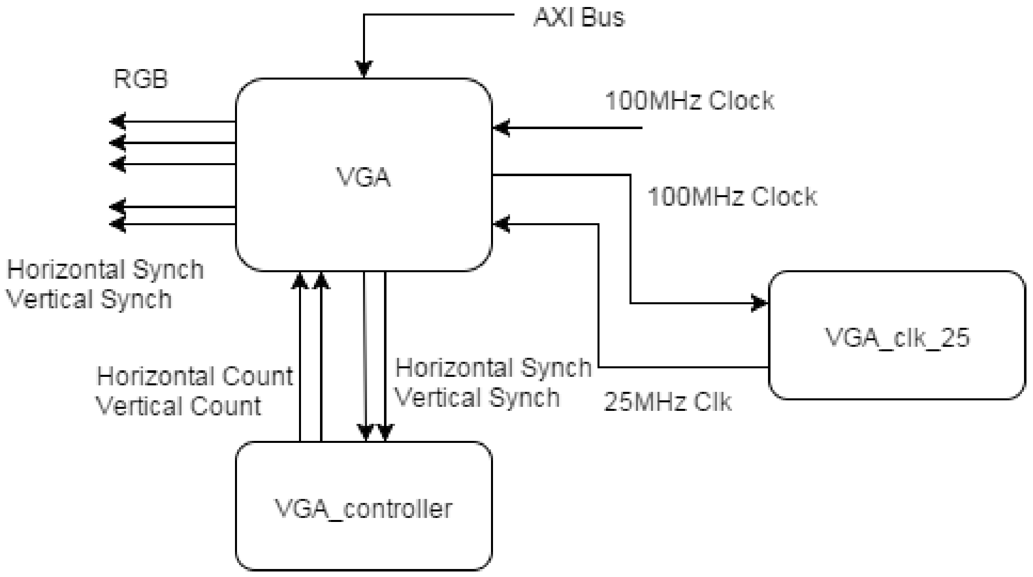


Figure 4: The internals of the VGA controller and selected outputs and inputs.

3.1.3 VGA Controller

The objects have a static order in which they are painted. Overlapping objects can easily be managed, if a pixel is not covered by an object then it is assigned the background color. The coloring logic is organized into a rather large conditional statement. The controller begins to check if the skater is going to be painted and then checks for ”power ups” and obstacles. The condition for each object on the screen is that the horizontal count must be larger than the x-position for the object and smaller than the sum of the x-position and the object width. Then the vertical condition is checked, if the vertical counter is larger than the y-position of the object and smaller than the y-position plus the object height. If these conditions are true, the pixel is within the object's boundaries and will be given the object's color. All the objects are checked in a specific order to allow simple overlapping without any troubles. The final color for the pixel is sent to the RGB output.

3.2 Software

The game loop uses several functions to run the game itself that are listed below and will be clarified. The CPU periodically polls data from the accelerometer controller and each game loop iteration updates the position and color of the obstacles. The skater's position is also updated with data from the accelerometer. Then the skater and obstacles are sent to the registers for the VGA controller. A ”power up” is in essence an obstacle but if the skater collides with it the player receives a potential bonus. Every game loop iteration the possible collision between obstacles and ”power ups” and the skater is checked. If a collision is detected the game ends. After a short time a new game starts.

3.2.1 Game loop

The game loop that runs in the main function is constructed of two while loops, the outer one is an infinite loop and the inner one is controlled by a bool variable. If the skater hits an obstacle then the bool variable is changed to ”false” which causes the inner loop to stop its iteration. After that the bool variable will hold its value for some time until it is set to ”true”. At this point the game will be reinitialized and the inner loop will start iterating again. The fundemental structure of the gameloop can be see in the pseudocode below.

```
while(true) {
        while(game_active) {
                move_obstacles();
                .
                .
                .
                get_accelerometer_angle();
                move_skater();
                .
                .
                .
                collision_detection();
    }
}
```

3.2.2 Initializing the game

When the game starts all the obstacles are initialized as inactive, meaning that no obstacles will be displayed on the screen. The background color is set to its default color. All obstacles vertical speed is randomized and their color is set to the background color so that they will not be visible. The obstacles x positions are randomized so that they will be spread out on the screen. The skater’s horizontal position is centered and set to a color.

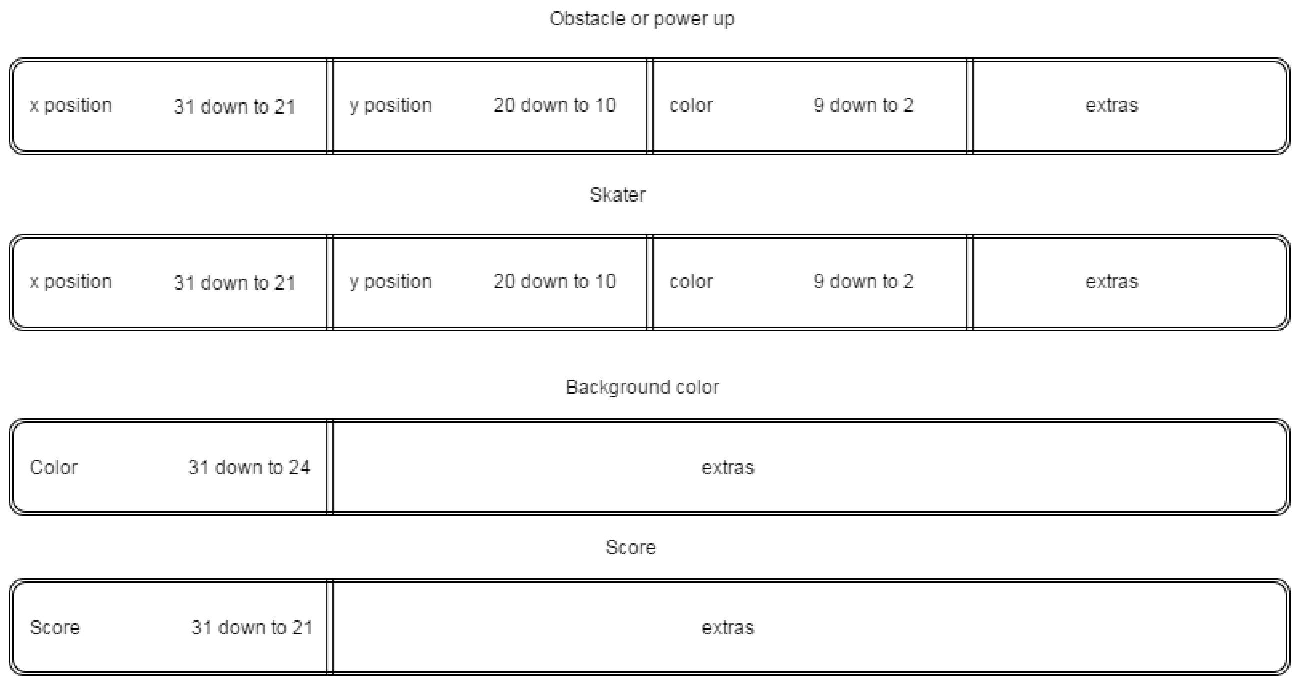


Figure 5: How the data is located in the registers.

Data in registers

Register	Data
0	Skater
1	Obstacle 1
2	Obstacle 2
3	Obstacle 3
4	Obstacle 4
5	Obstacle 5
6	Obstacle 6
7	Obstacle 7
8	Obstacle 8
9	Obstacle 9
10	Obstacle 10
11	Score
12	Power up 1
13	Power up 2
14	Background

Table 1: Illustrating which registers the different game objects is stored to. A game object is always written to a specific register.

The background color and all objects are sent to a specific register corresponding to them according to the documentation in table 1.

3.2.3 Collision detection

The obstacles and "power ups" are stored in arrays. In every iteration of the game loop, the position of the skater is checked to see if it is within a obstacle or "power up". This is done by checking all the corners of the square that represents the skater.

3.2.4 Skater

The skater's speed is determined by how much the accelerometer is tilted. In every loop iteration in the main function, the skater is moved accordingly. The more the accelerometer is tilted towards one direction the faster the skater moves on the screen. If the skater moves to the boundaries of the game field it is stopped and prevented to move further in that direction.

3.2.5 Obstacles

During the initialization of the game all obstacles are assigned a vertical speed. The obstacles y position is incremented with the number of pixels that correspond to the vertical speed. The obstacles color is also randomized to achieve flashing obstacles. The y position of the obstacles is also checked if they are on the game field or not. If they have reached the bottom of the screen, then the y position is set to zero. Finally the obstacles are written to the corresponding register.

3.2.6 Power ups

When the game is starting, the "power ups" are assigned a random type and constant speed. Depending on the type assigned, the power up gets a color specified in the Appendix, subsection 10.1.

The "power ups" are moved in a similar way to obstacles and when they reach the bottom of the screen they reappear at the top of the screen after they are assigned a new type and color.

3.2.7 Increment score

Every time an obstacle reaches the end of the screen the player gets one point. This is done in the function move obstacles at the same time as the obstacles y position is checked.

The player can also get points by collecting "power ups". The different "power up" types give different number of points, as specified in the Appendix under subsection *10.1 Objects*.

3.2.8 Send objects and score

When an object is to be sent to the VGA controller, a 32-bit integer is assigned the correct values that are sent to the registers. The data is bit shifted to end up in the correct position in the integer. The data sent is x position, y position and color. The way data is shifted is illustrated in figure 5. One object is sent at a time. Each object has it's own register and will be sent to the same register every time.

The score is also sent in the same manner, but the score is first divided in to three parts by dividing by ten and storing the rest three times. This is done so that the score can be represented in decimal radix on the screen.

4 User manual

4.1 Download

To start with, the compressed project must be downloaded from the course page [1] and extracted. The software is located in the "acid_workspace" folder. The file that is needed to run the system on the Nexys 3 board is located in the "acid_skate_apocalypce_hw_platform" folder and is called download.bit. Locate this file and have it ready for later.

4.2 Connect the system

Connect a USB cable to the Nexys 3 board for power and programming of the FPGA. Make sure the power switch on the Nexys board is turned off. The micro USB should be connected to the board and the USB 2.0 to the computer. Connect the VGA cable to the screen and the board.

Now its time to connect the accelerometer to the board. It is to be connected to the 12-pin connector called JB and can be connected with a cable or straight in to the 12-pin connector. It is important that the accelerometer is connected to the JB connection or the system will not respond to user input. Finally mount the accelerometer to the skateboard.

When everything is connected its time to turn on the screen and set it to VGA input, so the game can be displayed. Now the board can be turned on. If the red light is turned on then the board is on.

4.3 Program the board

To program the board use Adept with the file download.bit from the location found in section *4.0.9 Download*. When Adept has transferred the download.bit file correctly to the board, you are ready to play. Now you can use the accelerometer to move the skater, by hand or connecting the accelerometer to a skateboard and moving the skater that way.

5 Problems and solutions

5.1 Tools

In general we had a lot of problems with the design tools. The biggest problem was creating new IP:s and connecting them correctly. It took us a very long time to get the hardware up and running so that we could start testing the software. To be efficient we wrote the software at the same time as the hardware to avoid wasting time.

When writing the VHDL code we had to export the design each time we changed something to be able to test it. The exporting took around five minutes and was therefore a very time consuming part of the work.

To have some kind of software configure management we made new versions of the system along the project by renaming the folder containing the system. This created problems when exporting a new version in XPS because it is dependent on the directory where the design was last exported. Sometimes the projected was exported into a old version and sometimes a old project was exported. This was very confusing in the beginning but we solve the problem by having only the newest version of the system in the directory being used. In the future a real configuration management tool might be used. We thought of that in the beginning but decided not to because some files have "total" search paths to other files and we thought that could be problematic on different computers.

5.2 Gyro

As explained earlier the gyroscope controller caused some problems and delayed the project a little. The cause for this was twofold. First the Gyroscope controller IP that we initially tried to use was too advanced for our needs and it was quite complex to modify. Secondly we realized that we could work with an accelerometer instead. We chose to take a simpler IP from a previous iteration of the course. This accelerometer controller was easier to configure for our needs.

5.3 AXI bus

The AXI bus was not the real problem, but the registers that were used for the VGA controller proved a challenge. The problem is related to the tools and how ports, signals and so on are connected. We had trouble setting up the first shared register between the CPU and the VGA controller. But after that the rest of the shared registers were rather easy to add.

5.4 External connections

The external connections caused some trouble in the beginning of the project, but after some time and some attempts we figured out how to map external ports properly.

6 Improvements and Future work

The logic in the controller for all x-, y- and z-axis is left in even if is not used. This functionality can be used if we in the future want to add more features. For example the feature to be able to jump over obstacles.

The graphics could also use an update. The simple 2D graphics without much of textures and visual effects is looking rather dull. A major overhaul of the VGA controller could add sprites and some support for a dynamic background. A sprite for the "skater" that actually looks like a skater would be a good first step.

7 Lessons Learned

Start early with the project! In this course this was actually not solely the groups fault. The design tools needed to implement and develop the project were not installed on the school computers resulting in a weeks delay to begin with. This lead to some work on the weekends and long days to get back on track. A common problem in many projects is communication in the team. We solved this by working at the same time in the same computer lab. This limited problems and delays from misunderstandings and gave every member in the group a good understanding in how the system works. We also decided and documented the interfaces between the CPU and VGA controller. This proved to be very useful. Whenever something was unclear we could check the documentation to see what had been decided. We have last but not least gained valuable skills and knowledge on how to design embedded systems with the Xilinx Design Tools. As computer science students we have also gained valuable insight in hardware design. With this knowledge we will approach our next embedded system project with more confidence and the ability to avoid issues before they arise.

8 Contributions

We divided the workload to be able to work in parallel. It is not efficient for everybody to sit by one computer and work together. But some times we worked together if something did not work as expected. The way the work was divided can be seen in table 2.

Contributions

Moment	contributed
VGA controller	Martin
Report	Martin, Rickard, Thomas
Accelerometer controller	Rickard, Thomas
Software design	Rickard
Programming the game loop	Rickard, Thomas
Hardware design	Thomas
Presentation	Martin

Table 2: Contributions from group members during different moments of the project.

References

[1] http://cs.lth.se/eda385, accessed 25-09-2015

[2] http://www.xilinx.com/tools/xps.htm, accessed 25-09-2015

[3] http://www.xilinx.com/tools/projnav.htm, accessed 25-09-2015

[4] http://www.xilinx.com/tools/microblaze.htm, accessed 25-09-2015

[5] http://cs.lth.se/eda385/archive/projects-2012/, accessed 25-09-2015

[6] http://www.xilinx.com/tools/sdk.htm, accessed 07-10-2015

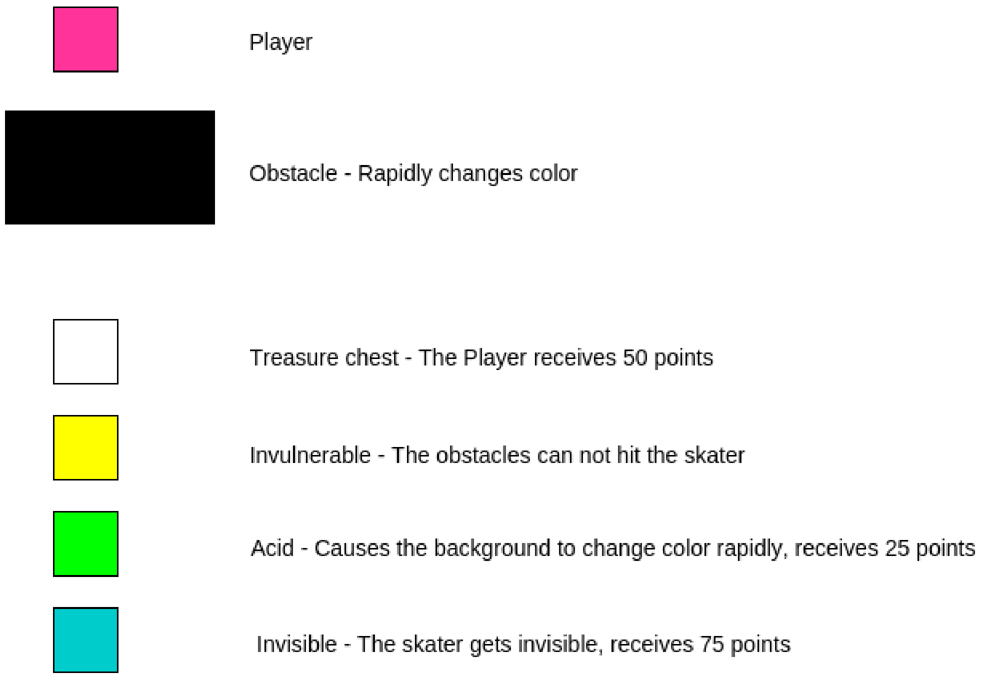
[7] http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-GYRO, accessed 07-10-2015

[8] http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-ACL, accessed 08-10-2015

[9] http://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html, accessed 13-10-2015

9 Appendix

9.1 Objects



Player

Obstacle - Rapidly changes color

Treasure chest - The Player receives 50 points

Invulnerable - The obstacles can not hit the skater

Acid - Causes the background to change color rapidly, receives 25 points

Invisible - The skater gets invisible, receives 75 points

Figure 6: Decription of the different objects in the game.

9.2 Game

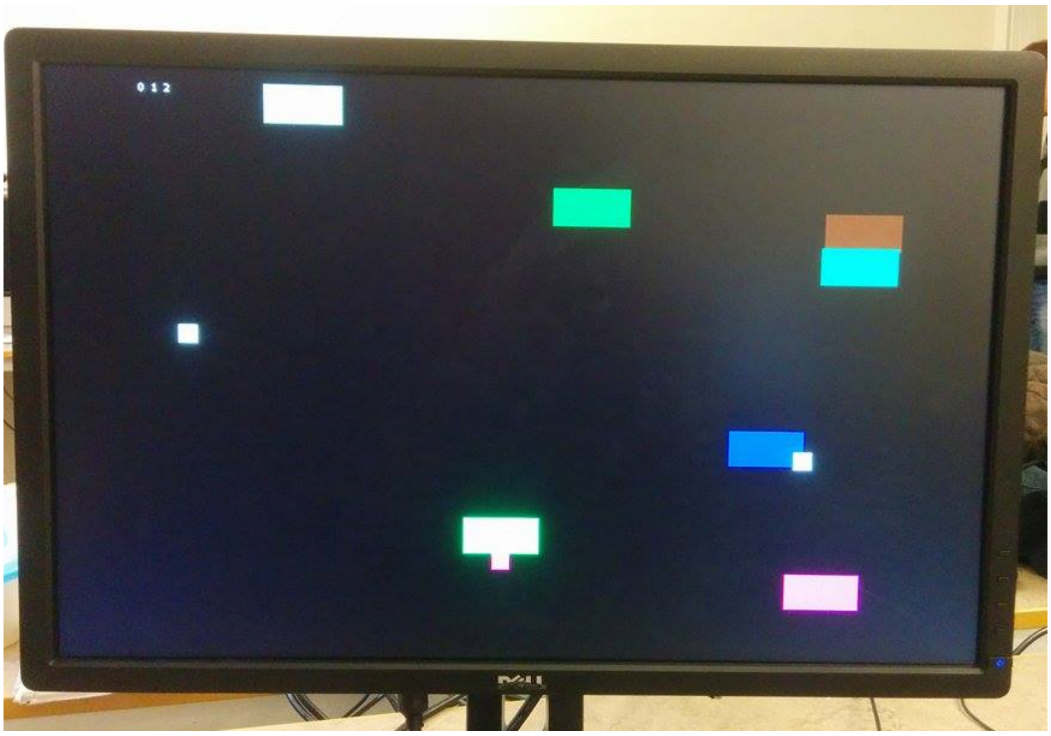


Figure 7: A screenshot of the game, the is visible in the center bottom of the screen.