

Labyrinth Game

EDA385 - Project Report

Gustaf Gustafsson

mt08gg7@student.lth.se

Robin Lefrant

robin.lefrant.947@student.lu.se

Fredrik Lundström

mt08fi2@student.lth.se

2012-10-18

This report describes the Labyrinth-game project in the course, "Design of Embedded Systems Advanced Course" EDA385. The task was to implement a Labyrinth game in hardware and software in which the player have to take a marble from a start position to an end position without falling into the holes along the path. In the original game this is made by tilting the game board that the marble rolls on. In this project it is made by tilting an accelerometer and the marble is viewed on a VGA-screen. The system is implemented both in hardware and software using a Nexys 3 board.

TABLE OF CONTENTS

- 1. Introduction 2
- 2. Hardware..... 3
 - 2.1. Physical hardware..... 3
 - 2.2. Graphical core..... 3
 - 2.3. Accelerator communication core 4
 - 2.3.1. How to use the core 5
 - 2.3.2. How it works..... 5
 - 2.4. Other hardware cores..... 6
 - 2.5. Device occupancy and memory size requirements 6
- 3. Software 6
 - 3.1. Initialization of the software 6
 - 3.2. The main while loop 7
 - 3.2.1. Compute the new marble position 7
 - 3.2.2. Check walls..... 7
 - 3.2.3. Check holes 7
 - 3.2.4. Next level..... 8
 - 3.2.5. Update holes 8
 - 3.2.6. Pause game 8
- 4. User manual 8
 - 4.1. Connect..... 8
 - 4.2. Start 8
 - 4.3. How to play..... 9
 - 4.4. Pause the game 10
 - 4.5. How to win the game 10
- 5. Problems and lessons learned 10
- 6. Contributions 11
- 7. References..... 11

1. INTRODUCTION

The classic Labyrinth game is about to make a marble roll via a path into the final hole without making it fall into any hole along the path. To control the marble the user tilts the playfield. In this project the playfield and the marble are represented on a VGA-screen. To control the marble on the screen in a similar way as in the original game and with a similar game experience, the Nexys 3 board with a mounted accelerometer is tilted by the player, to simulate the tilting of the playfield. The accelerometer measures the acceleration of gravity that would affect a real marble on the playfield. This acceleration is measured and integrated two times to get the position that the real marble would have. In this way it is possible to control the marble on the screen because it appears to follow the basic laws of physics.

To measure the acceleration of gravity the accelerometer PmodACL from Digilent is used. It measures the acceleration in x, y and z direction, where x and y direction is parallel with the playfield. In this project only x and y is used because of the two dimensional playfield. The system communicates with the PmodACL using the SPI-bus. The communication with the PmodACL is operated by an IP-core that reads data on the SPI-bus and sends it to the Microblaze CPU upon request via the FSL-communication. The position of the marble is calculated in software running on the Microblaze CPU. The walls and holes appear on the screen if the software asks for it but are created in hardware. The software runs a loop that calculates the position of the marble and checks if the marble has run into a wall (if so, the marble bounce). A block diagram of the hardware is shown in Figure 1.

The VGA-controller is implemented in hardware using VHDL. It updates the screen with 60 Hz. To paint the walls and the marble it reads some registers that contains the position of the marble and the positions of the walls. The registers that contains the wall-positions is set by the software via the PLB-bus in the beginning of the program and when a new playfield should be shown. The position of the marble is set in the same way but is updated once every loop in the software.

In the project proposal the playfield was going to be printed from an image on an external memory. The major drawback of this method is that it becomes much more complicated to know where the walls (and the holes) are. Consequently, it's harder to make the wall-detection process. Instead we defined the walls with an algorithm in software where every wall is defined by its center x and y coordinate. This demands less memory and thus external memory is not needed. This also allows us to do the calculations more frequently because the transfer from an external memory would have been slower (note: this can actually be avoided by the use of a Direct Memory Access).

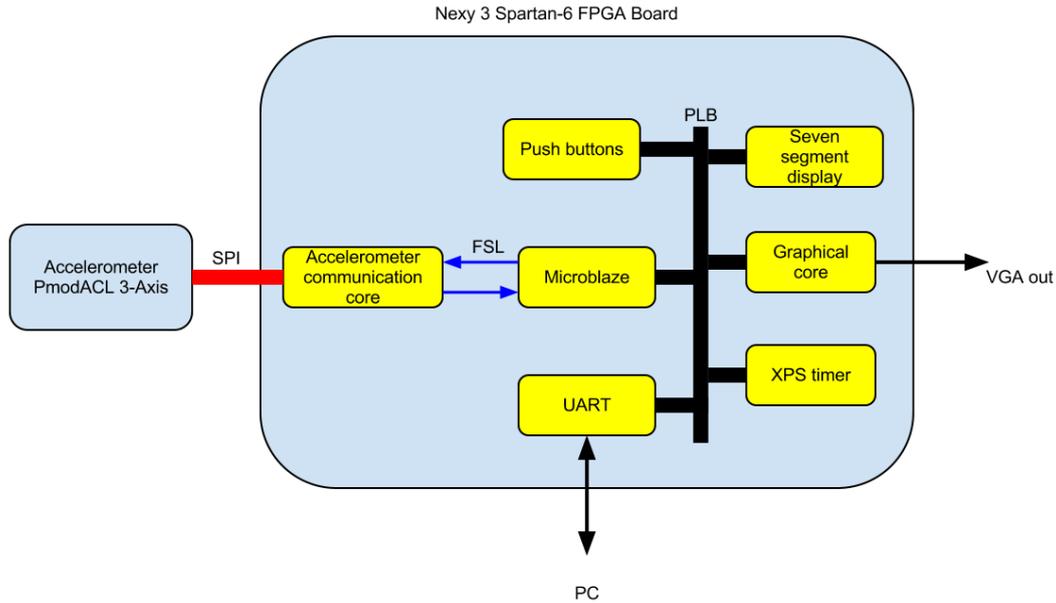


FIGURE 1 OVERVIEW OF HOW THE SYSTEM IS BUILT UP.

2. HARDWARE

The main part of the hardware is represented in the figure 1. It basically contains some push-buttons used to pause the game and change the level, the 7-segment display for the scores, the PmodACL and the graphical accelerator. All of them are described below.

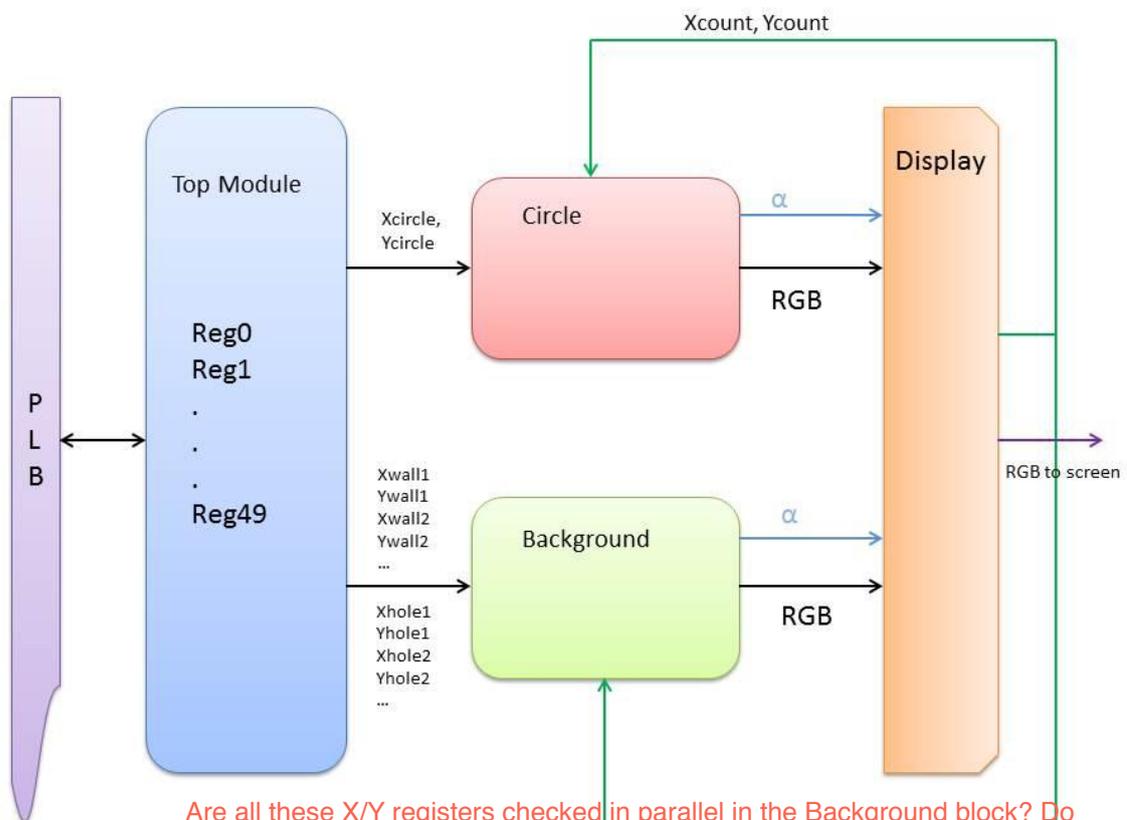
2.1. PHYSICAL HARDWARE

- Nexys 3 Spartan-6 FPGA board. [1]
- Digilent PmodACL 3-axis accelerometer. [2]
- Computer screen with VGA connection.

2.2. GRAPHICAL CORE

The need of a graphical accelerator is due to the amount of information that has to be displayed in a short amount of time. The processor alone can't afford such high data rate. We then need a dedicated hardware that will manage the data transfer to VGA output. In our game, we essentially need to display some walls and some circles (marble or holes), therefore we created some dedicated VHDL modules to make all the necessary computations for the display of the objects. Since the game is controlled by the software, we need a communication protocol between the CPU and the graphical core, this is done by the Processor Local Bus (PLB). Here is how it works, the top-level module has 50 available registers (could be more) in which both the CPU and the graphical accelerator can write. These registers contain the information about the position of the marble, the walls and the holes. The top-module instantiates two modules called *circle* and *background*. These modules receive the needed information (position of the marble for example) and will output the correct RGB (Red, Green, Blue) signals and a consistency signal α depending on the position of the counter of the VGA controller. For example, circle will always output RGB to red if we want to have a red circle but the signal will only be set if the circle is located in the region of the VGA counter. The same principle is used for the background (the walls and the holes), an RGB-color is set depending on if we want to display a hole or a wall and the signal is always set since the background is static. Finally, we have the *display* module which is the interface between the dedicated modules and the VGA screen. This last module receives

two RGB signals from the background and circle modules and will choose which one of these signals will be displayed on the screen. This choice is made with the α signals, if α -circle is set, then the RGB from the circle module will be displayed, otherwise, RGB signals from the background (the marble is always above the background) so this module can be seen as a multiplexer. This module also outputs the X_{count} and Y_{count} signals which are the pixel that is currently displayed. The previous modules need these signals to set the α signals. The use of these α signals allows us to create several modules with priority order. With this system, we can now create new modules (if we want to improve the graphical quality of the game) without changing anything in the way things work. All we have to do is to set the priority order. We use a VGA resolution of 640x480 pixels, which we can easily change, but we need to be consistent with the clock frequency so that it's fast enough for the human eye. Unfortunately, we only have 8 bits for the color which limits us to 256 different colors. A schematic of the graphical core can be seen in figure 2.



Are all these X/Y registers checked in parallel in the Background block? Do you have also dimensions for walls? For holes is enough maybe with 2 coordinates (and a default radius), but not for rectangles...

FIGURE 2 SCHEMATIC OF THE GRAPHICAL CORE

2.3. ACCELERATOR COMMUNICATION CORE

The accelerometer that is used is a PmodACL from Digilent that is directly pin compatible with the Nexys 3 Spartan-6 FPGA board. It is possible to communicate with the accelerometer by either I2C or SPI. The communication that is used in this project is SPI via the Accelerometer communication core. The main part of this core is taken from an example project from Digilent, written in VHDL for the PmodACL. It is then modified to fit into this project. A block diagram for the core is shown in Figure 3.

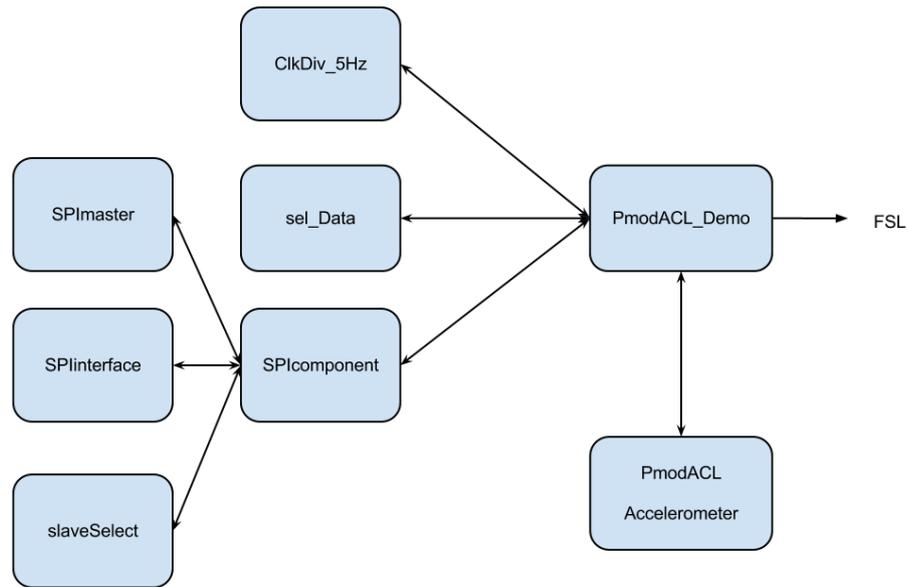


FIGURE 3 SCHEMATIC OF THE HARDWARE THAT HANDLES THE COMMUNICATION BETWEEN THE PmodACL ACCELEROMETER AND THE FSL.

2.3.1. HOW TO USE THE CORE

The accelerometer communication core uses FSL (Fast simplex link) for communication to the processor and the SPI interface to communicate with the PmodACL. The software gets the acceleration value via the FSL by sending a request via the FSL. Then the core puts the latest measured value on the FSL and the software reads it. The axis data is put on the FSL as a 32 bit vector. Each axis value is represented as a 10 bit vector. They are all put into the 32 bit vector and sent to the CPU at once. The reason why FSL is used is because the principle was known by some of the group members from the basic course.

2.3.2. HOW IT WORKS

The core consists of different entities in an hierarchical system. PmodACL_Demo is the top level module. The purpose of this module is to handle the communication with the surrounding environment. Below the top level module in the hierarchy is the modules ClkDiv_5Hz, sel_Data and SPIcomponent. The ClkDiv_5Hz is a clock divider that provides the core with a 5 Hz clock by dividing the 100 MHz system clock. The 5 Hz clock triggers the transmission and reception of data from the PmodACL. In sel_Data the values of the acceleration is put to an variable in the top level module that is put on the FSL-bus. The SPIcomponent is the module that communicates with the PmodACL via the SPI-interface. This is done with the sub-modules SPImaster, SPIinterface and slaveSelect. SPImaster have the 5 Hz clock as an input. This triggers the SPImaster to start the Transmission and Reception of data from PmodACL. The SPImaster tells the slaveSelect module to set the slave select bit in the PmodACL which tells it to start transmission and reception. Then the SPIinterface starts to receive and transmit data. When the transmission and reception is done the SPIinterface tells the SPImaster and the SPImaster tells the slaveSelect to set the slave select on PmodACL to idle mode. The received data is sent to the top level module. The data is then available to the sel_Data module that puts the data on the FSL-bus when the software sends a request.

2.4. OTHER HARDWARE CORES

Some of the complete cores from Xilinx that is used in the system:

1. Microblaze CPU: This is the CPU of the system where the software runs. [3]
2. FSL core: Two FSL-cores is used in the system to handle the communication between the CPU and the Accelerometer communication core in both directions.
3. XPS timer: The XPS-timer is used in the marble-position calculation.
4. UART: The UART has only been used to debug the system during the development.

2.5. DEVICE OCCUPANCY AND MEMORY SIZE REQUIREMENTS

The device utilization for the FPGA is shown in Figure 4. We use 5078 of the 18224 available slice registers which means that 27% are used. We also see that the number of slice LUT's is 7214 of 9112 available and thus 79% of these are used. 27896 Bytes of the 32kB available BRAM is used. The UART takes around 3kB of this memory and this feature can be removed if we run out of memory.

Device Utilization Summary (actual values)			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	5,078	18,224	27%
Number used as Flip Flops	5,072		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	6		
Number of Slice LUTs	7,214	9,112	79%
Number used as logic	6,990	9,112	76%
Number using O6 output only	4,491		
Number using O5 output only	158		
Number using O5 and O6	2,341		
Number used as ROM	0		
Number used as Memory	164	2,176	7%
Number used as Dual Port RAM	64		
Number using O6 output only	0		
Number using O5 output only	0		
Number using O5 and O6	64		
Number used as Single Port RAM	0		
Number used as Shift Register	100		
Number using O6 output only	29		
Number using O5 output only	1		
Number using O5 and O6	70		
Number used exclusively as route-thrus	60		
Number with same-slice register load	42		
Number with same-slice carry load	18		
Number with other load	0		
Number of occupied Slices	2,235	2,278	98%

FIGURE 4 EXCERPTS OF XILINX DEVICE UTILIZATION REPORT

3. SOFTWARE

The software handles most of the game logic. Briefly it requests information from the accelerometer core, then calculates the new position of the marble and then sends it to the graphical core. When the marble gets into the final hole the player scores up and the playfield switches.

3.1. INITIALIZATION OF THE SOFTWARE

Before the main while loop is executed some initializing must be done. First a short UART message is sent for debugging reasons, to see if the UART and the program run well. Then the start position of

the marble will be set and every wall and hole will be initialized and the positions of these will be sent to the graphical core to be drawn on the screen. The timer will then be started and used in the position calculations that use integration. When the game starts it will be paused and can be resumed by pressing button A8.

3.2. THE MAIN WHILE LOOP

All program logic is called in this while loop. During the while loop a lot of functions are called that handles different parts of the game logic.

3.2.1. COMPUTE THE NEW MARBLE POSITION

This function starts by reading the timer and reset it. The time is then used to make an integration of the acceleration to get the velocity and then an integration of the velocity to get the position. This is made in both x and y direction. The mathematical formulas are the following:

$$v_x = \int a_x dt$$

$$v_y = \int a_y dt$$

$$s_x = \int v_x dt$$

$$s_y = \int v_y dt$$

How do you actually compute the integrals? What are the bounds for these?

To be able to do these computations, the values of the acceleration in the different directions are requested from the accelerometer communication core via the FSL communication. When the new positions are calculated, a check is made to see if the position is outside the playfield (the screen). If the position is outside the playfield, the marble has reached a wall and should bounce. The velocity is then reversed and decreased to simulate the loss of kinetic energy in a bounce. Then the new marble position is moved inside the playfield to not get stuck in the wall.

3.2.2. CHECK WALLS

This function checks if the new calculated position is inside a wall. If it is, the marble will bounce. All walls have the same dimensions and they are internally represented by the x and y coordinates of the center and a variable telling if the wall is vertical or horizontal. All walls are stored in a matrix and in every loop the matrix is traversed to see if the marble should bounce. Three different cases are taken into account; if the marble should bounce against one of the long sides of the walls, one of the short sides or against a corner. The detection of bounces to the sides of the walls is rather simple, just check if the new calculated marble coordinates is inside the wall. If the coordinates are inside a wall the coordinates are moved outside and the direction of the velocity is reversed and decreased as mentioned above. To avoid the time-consuming computations that a square root operation would require the marble is internally represented as 8 points which are used to check if the marble is hitting a corner. If the marble has hit a corner the bounce looks like the bounce against a wall side.

3.2.3. CHECK HOLES

The check holes function traverse an array containing all holes positions and check if the position of the marble is close enough to fall into any of the holes. If the marble falls into the last hole, the player got five points and the level is changed (a new playfield is visible). This is done by setting new values to the registers containing information about holes and walls positions. If the player falls into wrong hole the player got one point subtracted from the scores. It is not possible to get less than zero points.

3.2.4. NEXT LEVEL

This function will be used if the marble fell into the red hole or if button C4 is pressed in paused mode. A call of this function will change the playfield and the marble position will be set at the start position.

3.2.5. UPDATE HOLES

This function updates the visible holes on the playfield. Only a few holes are visible at the same time, the holes that are closest to the marble and the final hole (the one to win the game). This function ranks the holes with respect to their distances to the marble. The limited number of visible holes is due to the fact that the game is already hard to play.

3.2.6. PAUSE GAME

This function polls if button A8 (pushbutton) on Nexys 3 is pressed. If it is pressed the function does a “busy wait” until the button is released. This is to avoid resuming the game. The pause function is also a “busy wait” because there is nothing else that should happen but wait for the button to be pressed again to resume the game. When the game is paused, it is possible to change the playfield by pressing button C4.

When all functions above are called the next state of the marble is calculated and it can be shown at the display. The new position of the marble will then be transferred to the registers in the graphical core. The seven-segment display will also be updated with the score information. When this is done the while-loop restarts.

4. USER MANUAL

4.1. CONNECT

First thing to do is to connect the PmodACL three axis accelerometer to the Nexys 3 board. This should be done using the 12-pins connection with the first pin named JB1 at the Nexys board. This first pin should be connected to pin 1 at the PmodACL board 12-pins connection. The second thing to connect is a VGA cable between a computer screen and the VGA connection at the Nexys board. For best player experience the screen should be able to update at 60Hz, show an 8 bit color image in the resolution of 640*480 pixels with the aspect ratio 4:3. The third thing to connect is a USB cable between a computer and the USB connection marked USB PROG. An optional connection that can be made is another USB cable between the UART connection on the Nexys board and a computer. This is used to show a nice little message when the game starts. The UART settings in the computer must be to 9600bits/s. All connections connected except the UART can be seen in figure 6.

4.2. START

The first thing to do is to upload the bit-file “download.bit” to the Nexys 3 FPGA board; this can be done by using Digilent Adept. When the bit-file is uploaded the game is paused but ready to start. To resume the game, press the button A8. If you later want to pause the game just press button A8 again. To restart the game, press button B8. For button configuration, see figure 6.

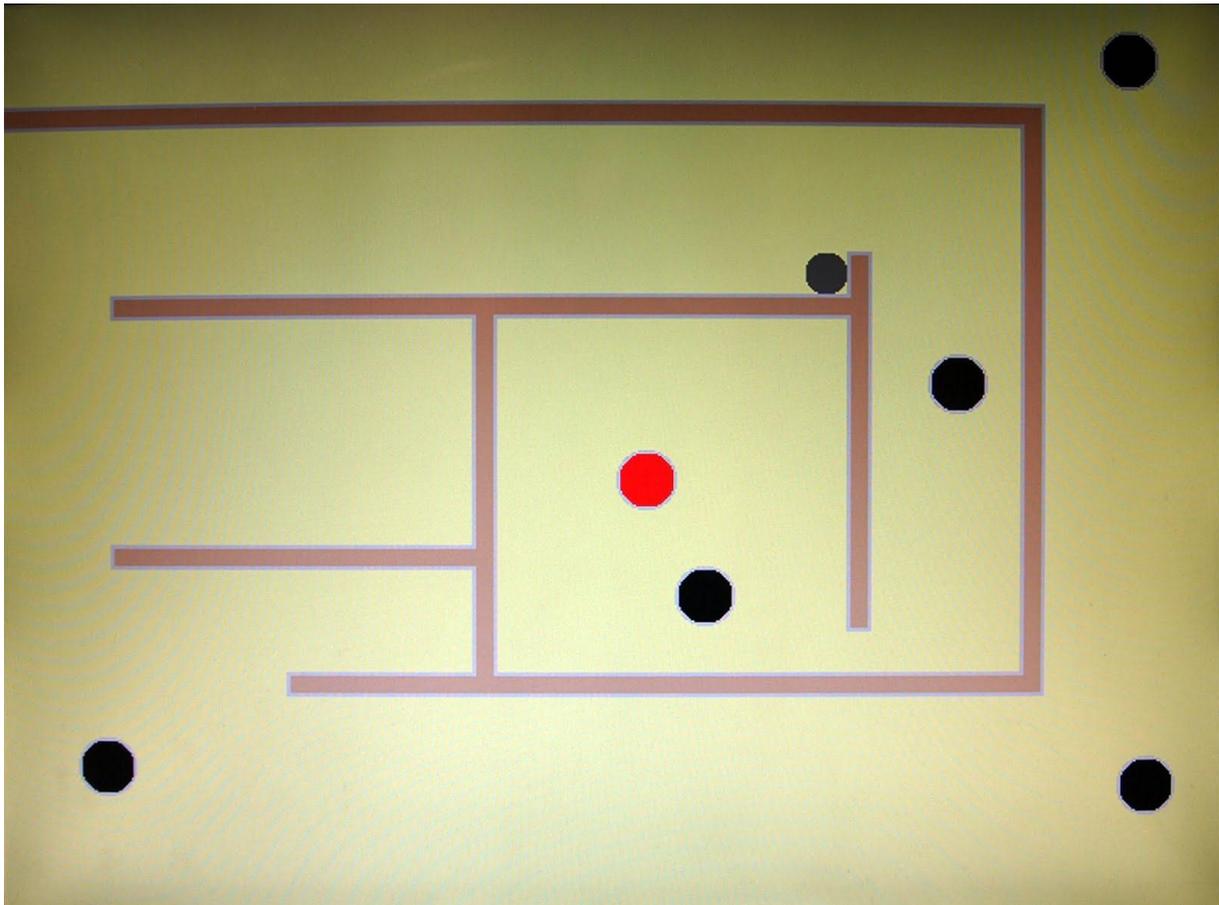


FIGURE 5 SCREENSHOT OF ONE PLAYFIELD.

Nice with a picture of the game. Are the holes of different sizes? How is the size stored in the registers of the VGA controller?

4.3. HOW TO PLAY

The goal of the game is to get the moving marble into the red hole, see figure 5. The marble is moved by tilting the Nexys board with the connected PmodACL accelerometer. The marble accelerates faster if the board is tilted more. The black holes should be avoided. If the marble falls into a black hole you lose one point. Your points balance can't be lower than zero. There are also brown walls that prevent the marble to go where you want; it just bounces if it hits them. When the marble reaches the red hole you get 5 points and another playfield is visible too. The score is visible at the seven segment display on the Nexys board, see figure 6.

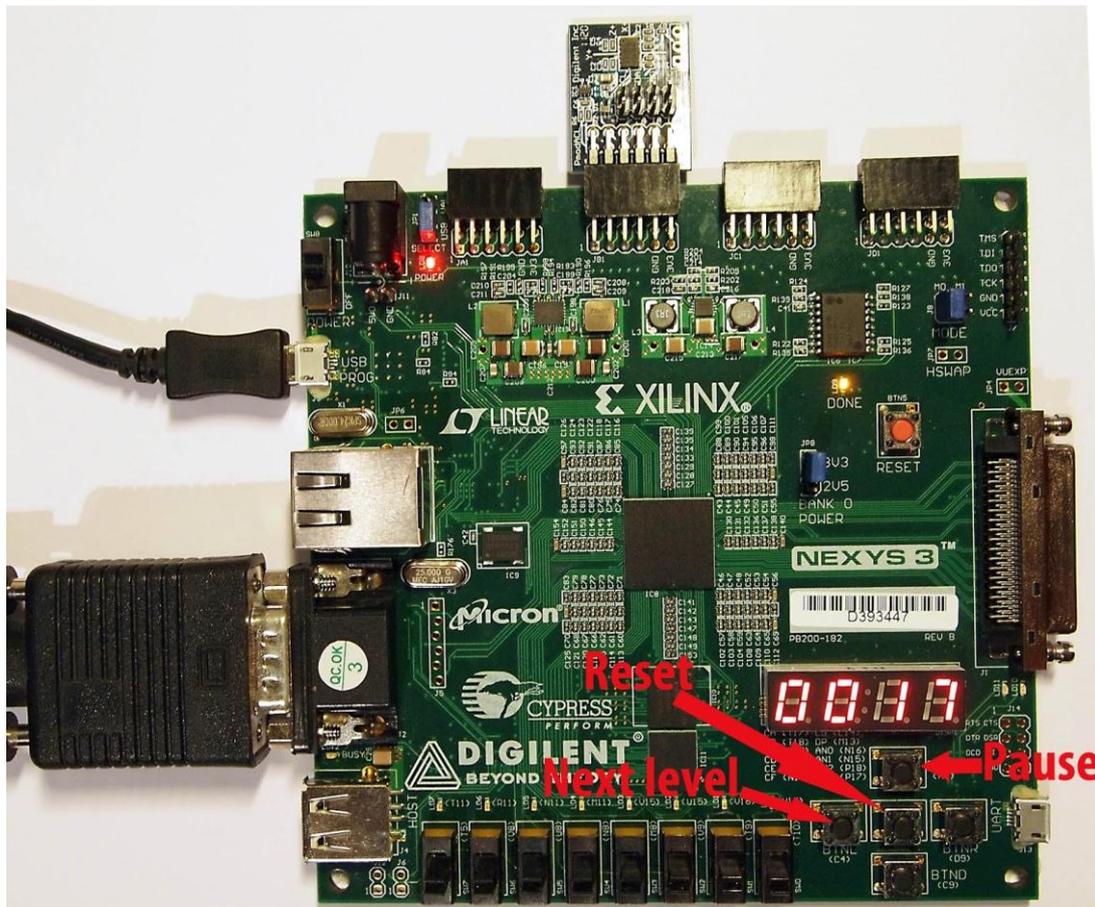


FIGURE 6 THE NEXYS 3 BOARD WITH VGA-SCREEN, PMODA1 AND USB-PROGRAMMING CABLE CONNECTED.

4.4. PAUSE THE GAME

As mentioned above the game can be paused by pressing button A8. When the game is paused, it is possible to change playfield by pressing button C4. To resume the game, press button A8 again.

4.5. HOW TO WIN THE GAME

The goal of the game, the player determines himself. For one player it can be to just finish the available levels. For another player the goal can be to achieve as much scores as possible. This is a game of possibilities.

5. PROBLEMS AND LESSONS LEARNED

It has been a project with many difficulties but we have learnt a lot from it. During the project it has been obvious that it takes a lot of time to connect different cores to each other and to make them talk to each other. We also think it has been hard to find good c documentations to communicate with different cores but on the other hand there were good examples in the Xilinx/Digilent installation. When we were building the hardware part of the project a lot of time was spent on waiting for the hardware to synthesize and therefore it is very important to make everything correct before synthesizing. One thing to think about is to clean the hardware in XPS before synthesizing when a new core has been imported to XPS. We can also save a lot of time by using pre built cores instead of writing them by ourselves from scratch. But on the other hand, it brings much more understanding to build it from scratch. There are still some known issues in the program. The corner bounce is not optimal and in some situations it can appear strange. This problem is caused of the fact that we do a simplified model of how the marble bounce to a corner. To make it bounce as physical marble it would

require very time consuming calculations. Another issue is that the pause-function sometimes doesn't work. These are problems that should be investigated if the game would be further developed.

6. CONTRIBUTIONS

- Acceleration communication core(modifications) and its communication with the FSL: Fredrik and Gustaf
- Graphical core: Robin
- Game logic in software: Fredrik and Gustaf
- Hole detection logic: Fredrik
- Position calculation: Fredrik
- Moving hole logic: Gustaf
- Wall detection logic: Gustaf
- Software communication with graphical core: Robin
- Report: Fredrik, Gustaf and Robin
- Presentations: Robin and Fredrik

7. REFERENCES

[1] Nexys 3 <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3>

[2] PmodACL <http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-ACL>

[3] Microblaze <http://www.xilinx.com/tools/microblaze.htm>

A good report and a nice result. Well done.