***Department of Electric & Electronic Engineering,***
***Boğaziçi University***

***EE240: DIGITAL SYSTEM DESIGN***

# FPGA BASED DUNGEON ESCAPE GAME

Abdülkadir Gökce
Sefa Kayraklık

Project Advisor: Şenol Mutlu

25.05.2018

## Table of Content:

## 1 INTRODUCTION

The project is an application of FPGA (Field Programmable Gate Array) based system that is coded using VHDL (Very High Speed Integrated Circuit Hardware Description Language). Due to its programmable non-fixed hardware structure, high efficiency and performance, FPGA is an active field of current hot research topics are also implemented using FPGA based system (e.g. image processing, data analytics). In the context of the final project, we preferred implementing an adventure game in which program generates video frames and displays them on the screen of the monitor via VGA (Video Graphics Array) cable.

## 2 PROBLEM STATEMENT

The project is a Dungeon Escape Game which is actually inspired by old-style Atari games. The objective of the game is that the player is expected to leave the map without touching the obstacles and being caught by a monster. To be able to escape from the map, the player needs to pick up the key and reach the exit door. Furthermore; the game has a time counter which keeps how many seconds has elapsed. The player has to also reach the exit door before running out of time, that is in 180 seconds.

## 3 RELATED BACKGROUND

In this section, you present the appropriate background or mathematical formulation, etc. The title may be changed to reflect the contents of the Section.

## 4 DESIGN

The overall design of the project consists of 5 main parts: VGA Driver, Elapsed Time, Frame Generator, Game States and Debouncer.

### 4.1 VGA Driver

FPGA can generate a video signal to display it on a VGA monitor. The board has three bits of red, three bits of green, and two bits of blue color information as its video output. Two synchronization signals are produced in order to start and stop the beams at the right time so that a line of pixels is plotted across the monitor and the lines stack up from the top to the bottom to form an image. Each synchronization signal is composed of two periodic negative pulses. Negative pulses on the horizontal synchronization signal mark the start and end of a line and ensure that the monitor displays the pixels between the left and right edges of the visible screen area[1].[1]

VGA Driver part composed of two processes: HsyncGen and VsyncGen. These processes produce the driving signal of VGA monitor according to the timing diagram of a 640x480 screen along with some other signals for counting in other parts of the code (e.g. h_count and v_count). For example, movement of the monsters depends on the end_of_frame signal so that no bulky counter is used for registering their states.

VGA Driver is already coded and tested in Lab7 of EE240, and the same code is also used in this project.

## 4.2 Elapsed Time

FPGA can represent up to 4-bit decimal number in its 7-segment display. Using this 7-segment display; Elapsed Time component counts the time passed while the player is in the game (game_state=01). The component uses the board_clock that is 100MHz generated by FPGA. A temporary counter for the least significant digit is utilized to count 100M clock cycles. At each roll-over of this counter –which takes 1 second- the least significant digit increases, analogously the middle digit is increased by 1 when least significant digit goes from 9 to 0. The same thing applies to the most significant digit, which is increased by 1 when the middle digit goes from 9 to 0. If elapsed time reaches 180 seconds, the game is over and the program proceeds to the game over state (game_state=3).

## 4.3 Frame Generator

This component is the main part of the program. To start with, the game has 4 different game states: the start screen which presents 'DUNGEON ESCAPE' image (game_state=0), the main gameplay state where the in-game video is generated (game_state=1), the win screen which shows the 'YOU WIN' image (game_state=2) and the game over screen that displays 'GAME OVER' image (game_state=3).

### 4.3.1 Start State

Initially, the game starts at the start screen state (game_state=0) and the image 'DUNGEON ESCAPE' is presented. The only one way to reach this state is through resetting

[1] See the reference for more detail information

the game. In this state, timer does not start to count yet. Whenever a push button is pressed, a state transition through start state to gameplay state takes places (game_state 0 => 1).

### 4.3.2 Gameplay State

The next state is the actual gameplay (game_state=1) where the all magic happens. The 480x640 pixels screen is divided into 16x16 grids so that the screen can be represented with 30x40 blocks. There are 6 different objects which are constructed as 16x16 arrays, and their states are denoted by 3-bits vectors. These objects are wall (000), obstacle (010), monster (110), player (001), key (111) and background (100). The overall state information of the map about which object is present in each grid is kept in a 30*40*3 array. This array is used as main RAM (Random-access memory) block of the program on which write/read operations are performed.

The player is controlled via the push buttons of FPGA board, and each push button is debounced separately. The player cannot pass through the walls; however, it loses the game if it touches any of the obstacles or monsters. The logic control of this simple physic motor is executed by checking the expected location of the player on the map controller whenever any button is pressed. Monster are controlled by the program itself, and they move along the predefined lines until hitting an obstacle. All monster can be controlled via single logic control unit since their movement is symmetric and they move the same distances.

Since the primary objective of the game is escaping the dungeon, one must pick up the key to open the exit door. Whether the key is picked up is also controlled similar to simple physic motor. If the state of new location of the player is "111" then the door will be opened. Likewise; if the player reaches the door position (28,39), then the player wins the game and program proceeds to "Win State". However, if the elapsed time exceeds 180 seconds, then game is over and program proceeds to "Game Over State".

Instead of using separate counters for each submodule of the program, output signals of the VsyncGen and the HsyncGen are modified for each design usage. control_r and control_c that are used for which grid is displayed next are obtained from v_count and c_count by dividing by 16 each signal, respectively. Similarly, grid_r and grid_c are used for indexing grid while determining the pixel color from the array blocks, and these variable are the 16 modes of  control_r and control_c, respectively.

In the gameplay state, the frame is generated by a relatively large logic controller unit. control_r and control_c loops through the main RAM and this control unit determines which block is to be displayed according to the current state of the map over that grid. grid_r and grid_c enable the program to print each block using their respective constant ROM blocks.

### 4.3.3 Win State and Game Over State

If the player could pick up the key and reach the door before the time up, program proceeds to win state and displays a “You Win” image that is stored in Game States module. Likewise, if the player hits an obstacle or monster or elapsed time exceeds 180 seconds then proceeds to game over state and displays a “Game Over” image that is also kept in Game States module.

## 4.4 Game States

The screens except for the main video frame screen are a constant 240x320-bit array that contains the information about the picture. In this array, 1s represent nonblack, 0s represent black. And conversion of an image to a 0-1 representation is done by Matlab code.

Game states module is the library part of the static screens of the game. The static screens of the game are kept on the pre-defined read only RAM  of the FPGA via IP (CORE generator& architecture wizard). It gives the corresponding bit according to frame_row, frame_column and game_state. Conversion of an image to a 0-1 representation is executed by a simple image processing Matlab code.

*Table 1: Static Screens:*

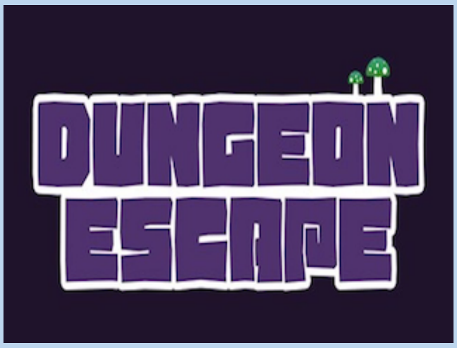
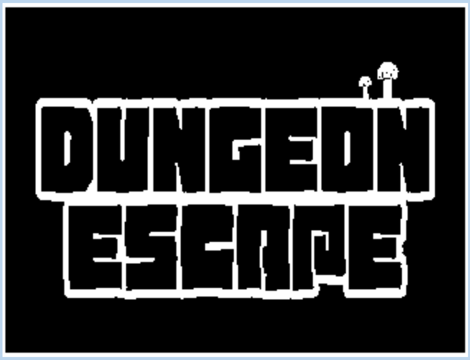
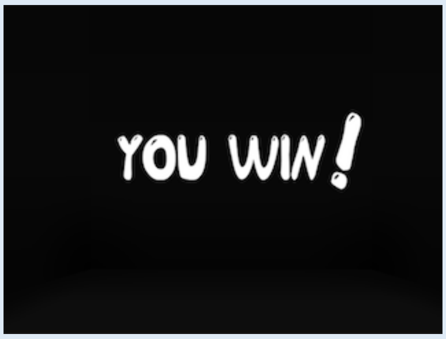
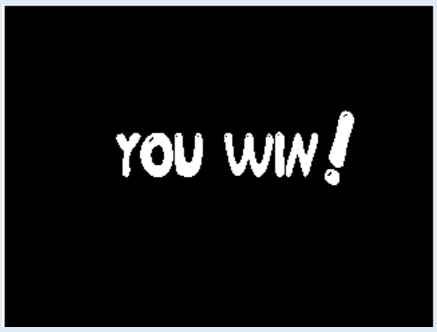
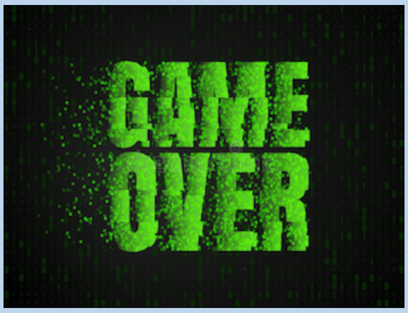
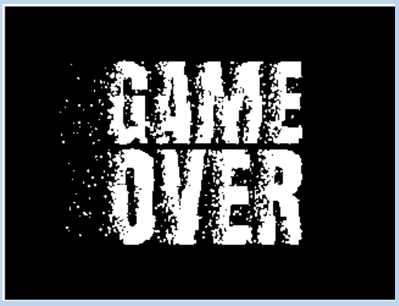
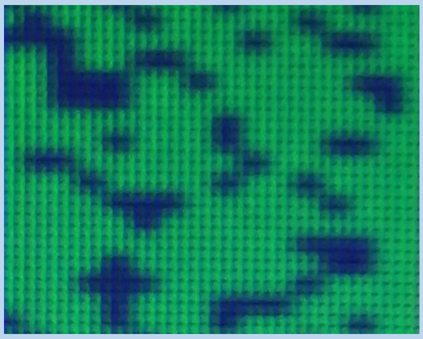
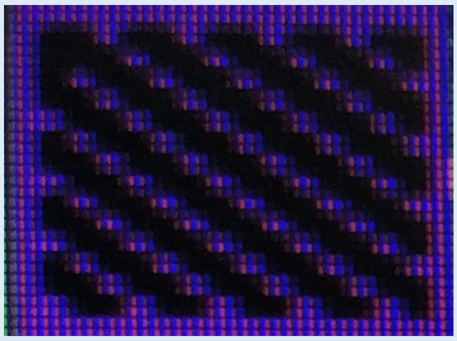
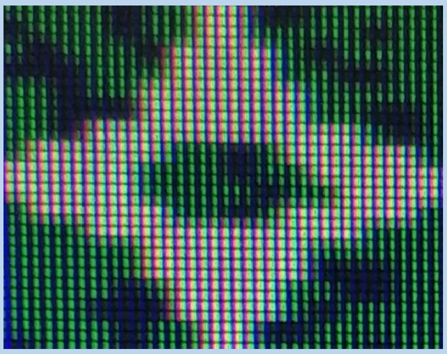
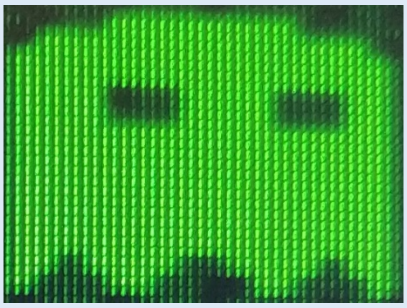
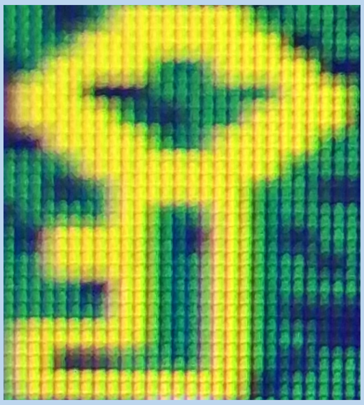
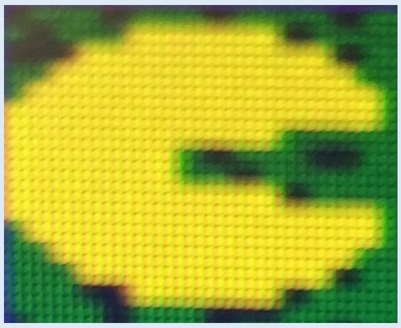
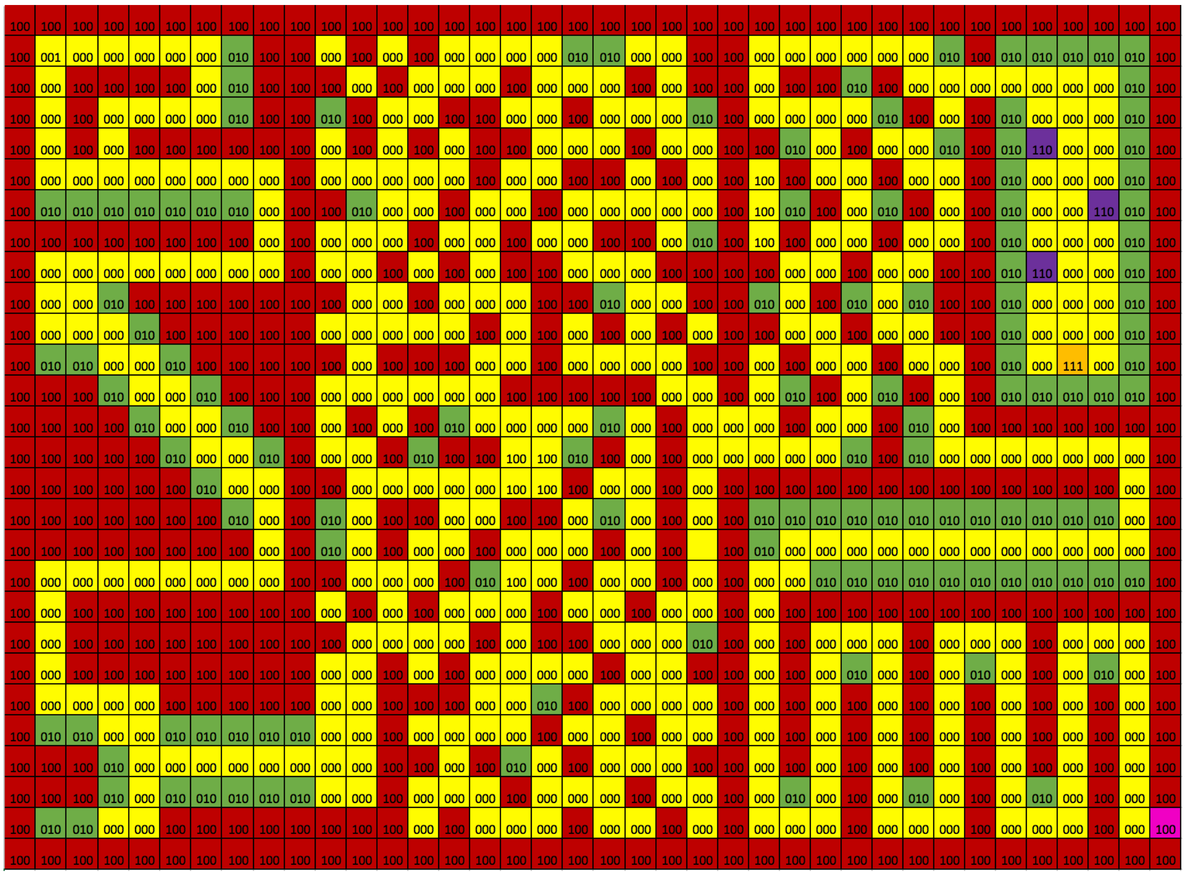
| Static Screens | Original Image | Processed Image |
|---|---|---|
| DUNGEON ESCAPE |  |  |
| YOU WIN |  |  |
| GAME OVER |  |  |

*Table 2: Objects Representations:*

| Objects | Bit Representations | VGA Display |
|---|---|---|
| **Background**<br>**‘000’** | ("1000100010000100",<br>"0100010000010000",<br>"1110000001001100",<br>"0010011000000000",<br>"0011100100000110",<br>"0011100000000010",<br>"0000000010000000",<br>"0000100010001100",<br>"0110000001000000",<br>"0001000010010000",<br>"0000111000001000",<br>"0000010000000000",<br>"0000000000011100",<br>"0000100000001100",<br>"0001110000010000",<br>"0000100011100000"); |  |
| **Wall**<br>**‘100’** | ("1111111111111111",<br>"1001001001001001",<br>"1000100100100101",<br>"1010010010010011",<br>"1001001001001001",<br>"1100100100100101",<br>"1010010010010011",<br>"1001001001001001",<br>"1100100100100101",<br>"1010010010010011",<br>"1001001001001001",<br>"1100100100100101",<br>"1010010010010011",<br>"1001001001001001",<br>"1100100100100101",<br>"1111111111111111"); |  |
| **Obstacle**<br>**‘010’** | ("0000000110000000",<br>"0000001111000000",<br>"0000001111000000",<br>"0000011111100000"<br>"0000011111100000",<br>"0001111111111000",<br>"0111110000111110",<br>"1111100000011111",<br>"1111100000001111",<br>"0111110000111110",<br>"0001111111111000",<br>"0000011111100000",<br>"0000011111100000",<br>"0000001111000000",<br>"0000001111000000",<br>"0000000110000000"); |  |

| | | |
|---|---|---|
| **Monster** **‘110’** | ( "0000000000000000", "0000111111110000", "0111111111111110", "1111111111111111", "1111111111111111", "1111000111100011", "1111000111100011", "1111111111111111", "1111111111111111", "1111111111111111", "1111111111111111", "1111111111111111", "1111111111111111", "1111111111111111", "1101111001111011", "1000110000110001"); |  |
| **Key** **‘111’** | ("0000001111000000", "0000011111100000", "0000111001110000", "0001100000011000", "0001110000111000", "0000111001110000", "0000011111100000", "0000001111000000", "0000001001000000", "0000111001000000", "0000111001000000", "0000001001000000", "0000001001000000", "0001111001000000", "0001000001000000", "0001111111000000"); |  |
| **Player** **‘001’** | ("0000000000000000", "0000011111100000", "0001111111111000", "0011111111111100", "0111111111111110", "1111111111111110", "1111111111100000", "1111111000000000", "1111111000000000", "1111111111100000", "1111111111111110", "0111111111111110", "0011111111111100", "0001111111111000", "0000011111100000", "0000000000000000"); |  |



| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 001 | 000 | 000 | 000 | 000 | 000 | 010 | 100 | 100 | 000 | 100 | 000 | 100 | 000 | 000 | 000 | 000 | 010 | 010 | 000 | 000 | 100 | 100 | 000 | 000 | 000 | 000 | 000 | 000 | 010 | 100 | 010 | 010 | 010 | 010 | 010 | 100 |
| 100 | 000 | 100 | 100 | 100 | 100 | 000 | 010 | 100 | 100 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 100 | 100 | 000 | 100 | 100 | 010 | 100 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 010 | 100 |
| 100 | 000 | 100 | 000 | 000 | 000 | 000 | 010 | 100 | 100 | 010 | 100 | 000 | 000 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 000 | 010 | 100 | 000 | 000 | 000 | 000 | 010 | 100 | 000 | 100 | 010 | 000 | 000 | 000 | 010 | 100 |
| 100 | 000 | 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 010 | 000 | 100 | 000 | 000 | 010 | 100 | 010 | 110 | 000 | 000 | 010 | 100 |
| 100 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 000 | 100 | 000 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 010 | 000 | 000 | 000 | 010 | 100 |
| 100 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 000 | 100 | 100 | 010 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 000 | 000 | 000 | 100 | 100 | 010 | 100 | 000 | 010 | 100 | 000 | 100 | 010 | 000 | 000 | 110 | 010 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 000 | 010 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 010 | 000 | 000 | 000 | 010 | 100 |
| 100 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 100 | 000 | 100 | 100 | 000 | 000 | 000 | 100 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 010 | 110 | 000 | 000 | 010 | 100 |
| 100 | 000 | 000 | 010 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 100 | 010 | 000 | 000 | 100 | 100 | 010 | 000 | 100 | 010 | 000 | 010 | 100 | 100 | 010 | 000 | 000 | 000 | 010 | 100 |
| 100 | 000 | 000 | 000 | 010 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 000 | 000 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 010 | 000 | 000 | 000 | 010 | 100 |
| 100 | 010 | 010 | 000 | 000 | 010 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 000 | 000 | 000 | 100 | 100 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 010 | 000 | 111 | 000 | 010 | 100 |
| 100 | 100 | 100 | 010 | 000 | 000 | 010 | 100 | 100 | 100 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 010 | 100 | 000 | 010 | 100 | 000 | 100 | 010 | 010 | 010 | 010 | 010 | 100 |
| 100 | 100 | 100 | 100 | 010 | 000 | 000 | 010 | 100 | 100 | 000 | 100 | 000 | 100 | 010 | 000 | 000 | 000 | 000 | 010 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 010 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 100 | 100 | 100 | 100 | 010 | 000 | 000 | 010 | 100 | 000 | 000 | 100 | 010 | 100 | 100 | 100 | 100 | 010 | 100 | 000 | 100 | 000 | 000 | 000 | 000 | 000 | 010 | 100 | 010 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 010 | 000 | 000 | 100 | 100 | 000 | 000 | 000 | 000 | 000 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 010 | 000 | 100 | 010 | 000 | 100 | 100 | 000 | 000 | 100 | 100 | 000 | 010 | 000 | 100 | 000 | 100 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 000 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 010 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 100 | | 100 | 010 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 |
| 100 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 100 | 000 | 000 | 000 | 100 | 010 | 100 | 000 | 100 | 000 | 000 | 100 | 000 | 100 | 000 | 000 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 010 | 100 |
| 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 000 | 000 | 100 | 000 | 100 | 100 | 000 | 000 | 000 | 010 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 |
| 100 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 100 | 000 | 100 | 000 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 100 | 000 | 100 | 000 | 010 | 000 | 100 | 000 | 010 | 000 | 100 | 000 | 010 | 000 | 100 |
| 100 | 000 | 000 | 000 | 000 | 100 | 100 | 100 | 100 | 100 | 000 | 000 | 100 | 100 | 100 | 000 | 000 | 010 | 100 | 000 | 000 | 000 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 |
| 100 | 010 | 010 | 000 | 000 | 010 | 010 | 010 | 010 | 010 | 000 | 000 | 100 | 000 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 |
| 100 | 100 | 100 | 010 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 100 | 000 | 100 | 010 | 000 | 100 | 000 | 000 | 000 | 100 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 | 000 | 100 |
| 100 | 100 | 100 | 010 | 000 | 010 | 010 | 010 | 010 | 010 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 010 | 000 | 100 | 000 | 010 | 000 | 100 | 000 | 010 | 000 | 100 | 000 | 100 |
| 100 | 010 | 010 | 000 | 000 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 100 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 000 | 000 | 100 | 000 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

*Figure 1: Map Controller (green->obstacle, red->wall, yellow->background, purple->monster, orange->key, pink->door)*

### 4.5 Debouncer

Debouncer part is needed for the push buttons of the FPGA because these buttons cannot produce stable pulses as they are pressed. Namely, to get a clean signal from push buttons, debouncing should be done for the push buttons.

In the algorithm, instead of giving the slowed clock to the Debouncer as Lab6 of EE240, the pixel_clock (25MHz) is given to the Debouncer because it is wanted that each stable signal should correspond only one clock sized signal for one press, and periodic clock sized signals for long presses.

## 5 RESULTS

When the project is synthesized, at first the error of exceeding the memory of the FPGA is occurred so the synthesis tool is not able to map the project on the FPGA. This error emerges since the synthesis tool can't optimize the mapcontrol array due to the presence of a lot of 1s in the array. The problem is overcome by redefining the objects' names (as the current names) with less 1s, so that the synthesis tool can eliminate the 0s in the array. Although it eliminates the 0s in the mapcontrol, the ratio of the used memory of the FPGA is 98, it almost reaches its limit.

## 6 CONCLUSION

The project is a FPGA based system that is coded in VHDL language, one of Hardware Description Languages. In the project, video frames are generated and displayed on a VGA monitor. And the project has 5 main parts:

- VGA Driver that provides synchronous negative pulses to drive the VGA monitor,
- Elapsed Time that counts and shows the game time in seconds at the 7-segment display,
- Frame Generator that produces all the video frames of the game,
- Game States that provide the library for the static screens,
- Debouncer that enables the push buttons to generate stable one-clock sized signals.

The main objective of the project is to gain experiences about the hardware language coding, VHDL, and to be able to control a VGA monitor via FPGA based system. Actually, it is observed that hardware language coding is high level of coding which one should consider all logical operations as hardware.

## 7 REFERENCES

1. Şenol Mutlu and H. Işıl Bozma, "EE 240 DIGITAL SYSTEM DESIGN LAB MANUAL", *Laboratory Manual*, pp. 79–83, 2015.

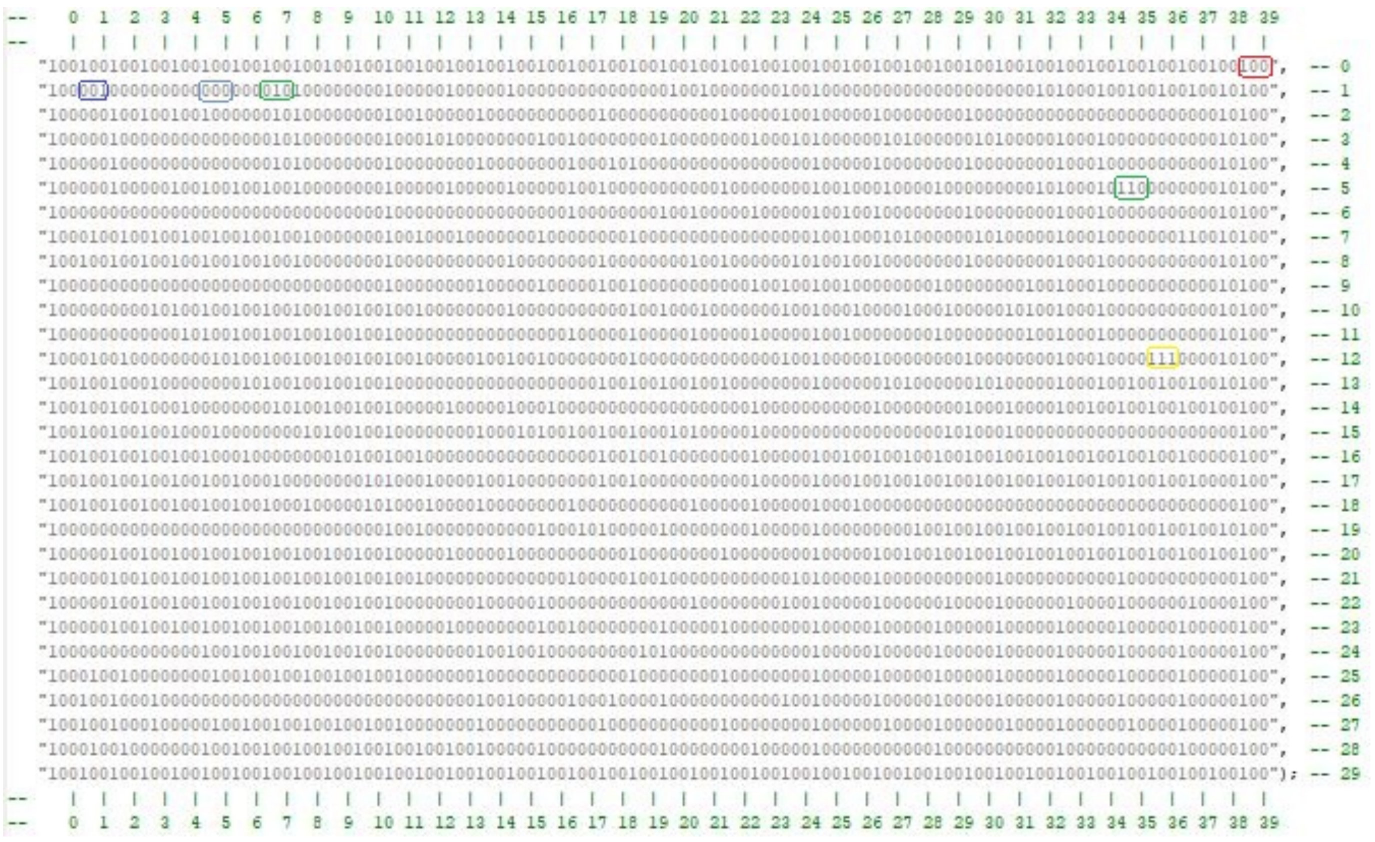
# APPENDIX

## A READ ME

In generating the bit file of the project, an error will occur if the user's operating system is windows 10 because Xilinx ISE Design Suite 14.7 is not properly working in some cases. To avoid this error, you should use a virtual machine or Windows 7.

## B USERS' MANUAL

Dungeon Escape is a simple user-controlled game. The control of the player is obtained via the push buttons of FPGA. If the user presses any button at the start screen, the game will begin, and the timing will start also. The user should pick the key without touching any obstacles and monsters to open the door until the required time is not elapsed.



```
--    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
--    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
    "100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100",   -- 0
    "100001000000000000000010100000000100000100000100000000000000000100100000000100100000000000000000000010100010010010010100",   -- 1
    "100000100100100000000010100000000000100100000000100000000000100000000100000100100000000100000000000000000000000000010100",   -- 2
    "100000100000000000000010100000000000100010100000000100100000000000100000000100000100000010100000010100000100100000000000010100",   -- 3
    "100000100000000000000010100000000000100000000100000000100010100000000000000000100000100000000100000000100100000000010100",   -- 4
    "100000100000100100100100000000100000100000100000100100000000000000100000000000100100010000100000010100010110000000010100",   -- 5
    "100000000000000000000000000000000100000000000000000100000000100100000100000100100100000000100000000010100000000000010100",   -- 6
    "100010010010010010010010000000100100010000000100000000100000000000000000000000100100010100000000010100000000110010100",   -- 7
    "100100100100100100100100000000000100000000000000100000000000100100000000010100100000000000100000000100010000000000010100",   -- 8
    "100000000000000000000000000000000100000000100000100000100100000000000100100100000000000100000000100100010000000000010100",   -- 9
    "100000000010100100100100100100100100000000100000000000100100010000000100100010000010000100000010100100010000000000010100",   -- 10
    "100000000000010100100100100100100100000000000000000000100000100000100000100100000000100000000000100100010000000000010100",   -- 11
    "100010010000000000010100100100100100100000100100000000000000000000000100000100000100000000100000000000100000111000010100",   -- 12
    "100100010000000000010100100100100000000000000000000000100100100100000000000100000000010100000000010100000100100100010100",   -- 13
    "100100100010000000000010100100100000100000100010000000000000000000100000000000000100000000010000100000100100100100100100",   -- 14
    "100100100100010000000000010100100100000000000100010100100100010100000100000000000000000000010100000000000000000000000100",   -- 15
    "100100100100100010000000000010100100000000000000000000100100100100000000000100000000100100100100100100100100100100000100",   -- 16
    "100100100100100100010000000000010100100100000000100000000100000000000100000100100100100100100100100100100100100100000100",   -- 17
    "100100100100100100100010000010000010100000100000000000100000000000100000000100000100000000000000000000000000000000000100",   -- 18
    "100000000000000000000000000000000100100000000000000010000100000000100000100000000100100100100100100100100100100100010100",   -- 19
    "100000100100100100100100100100100100000000100000000100000000000000100000000000100000000100100100100100100100100100100100",   -- 20
    "100000100100100100100100100100100000000000000000000100000100100000000000000000010100000000000000000100000000000000000100",   -- 21
    "100000100100100100100100100100100000000100000000000000100000000000100000100000000100000100000000100000100000000100000100",   -- 22
    "100000100100100100100100100100100000000100000000100000000100000000000100000100000100000100000100000100000100000100000100",   -- 23
    "100000000000100100100100100100100000000000100100000000000010100000000000000000100000100000100000100000100000100000000100",   -- 24
    "100010010000000000100100100100100100000000000000000000100000000000100000000100000100000100000100000100000100000100000100",   -- 25
    "100100100010000000000000000000000000000000000000000000100100000000010000000100100000100000100000100000100000100000000100",   -- 26
    "100100100010000010010010010010010010000000000100000000000000100000000000100000000100000100000000100000100000000100000100",   -- 27
    "100010010000000000010010010010010010010010010010000000100000000000000000100000000100000000000000000100000000000000000100",   -- 28
    "100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100100"); -- 29
--    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
--    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
```