Worcester Polytechnic Institute

ECE 574: Modeling and Synthesis of Digital Systems using Verilog and VHDL

Design and implementation (in Verilog) of Pong game

by

Vladimir Vakhter

Laboratory Report

12/04/2019

Dr. Yarkin Doroz
ECE Department
Professor

Adhitya Athyur Ganesh
RBE Program
Tutor

Table of Contents

1. Introduction

This project [1] is aimed to design, implement and verify a two-player mode of Pong game in Verilog.

The project consists of the following modules:

- ***top.v*** describes the inputs and the outputs connected to the hardware resources on the board (clock 100MHz, a reset button, USB-RS232 interface, VGA port, cathodes and anodes of the seven segment indicators), as well as interconnects the instantiations of custom hardware modules and IP cores.
- ***game_logic.v*** draws the elements of the game field, describes the game by implementing a state machine.
- ***paddle_ctrl.v*** describes the movement of a paddle.
- ***borders_ctrl.v*** describes the upper and the lower borders of the game field.
- ***ball_ctrl.v*** describes the movement of the ball.
- ***seven_seg_4.vhd*** represents the 4-digit seven-segment display [2].
- ***decoder.vhd*** converts binary numbers to hex symbols to be displayed on a seven-segment indicator.
- ***vga_controller_640_60.vhd*** is the VGA Controller IP core that generates the video synch pulses for the monitor to enter 640x480@60Hz resolution state. It also provides horizontal and vertical counters for the currently displayed pixel and a blank signal that is active when the pixel is not inside the visible screen and the color outputs should be reset to 0 [3].
- ***clk_wiz_0.xci*** is the MMCM Clock Wizard IP core [4] that generates the clock of 25MHz for the instance of the VGA Controller IP core [3] and the clock of 100MHz for the instance of the MicroBlaze MCS IP core [5] and other logic in this design.
- ***microblaze_mcs_0.xci*** is the MicroBlaze MCS IP core [5] that describes a light-weight version of the Microblaze softcore (embedded) processor. In this design, it is programmed in C to read out the keys pressed on the keyboard by the players by which they control the game process.

2. Methods

In this chapter, a general overview of the approaches to solution will be provided.

This project consists of multiple independent modules. Hereinafter, these parts will be described.

2.1 Pong Game Description

The game field is shown schematically on the Fig. 1.

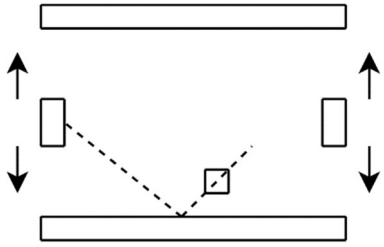

Fig. 1 – The scheme of the game field

In the implemented project, the game is displayed on a VGA screen 640x480pixels@60Hz.

There are two players that control two movable paddles from the keyboard. The possible directions of the movement are shown by arrows. The aim of each player is to hit the ball more times than the opponent: if a player misses the ball, another player gets a point. The maximum score is equal to 9. When one of the players reaches 9 points, the game ends and the score is reset. The current score in the game is reflected on the 4-digit seven segment display.

2.2 FPGA Specification

The FPGA [2] parameters presented on the Fig. 2 were set for the design project.

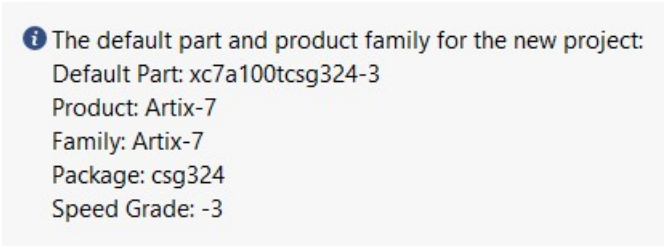

Fig. 2 – FPGA specification

2.3 Project Hierarchy

The hierarchy of the modules underlying the project is presented on the Fig. 3.

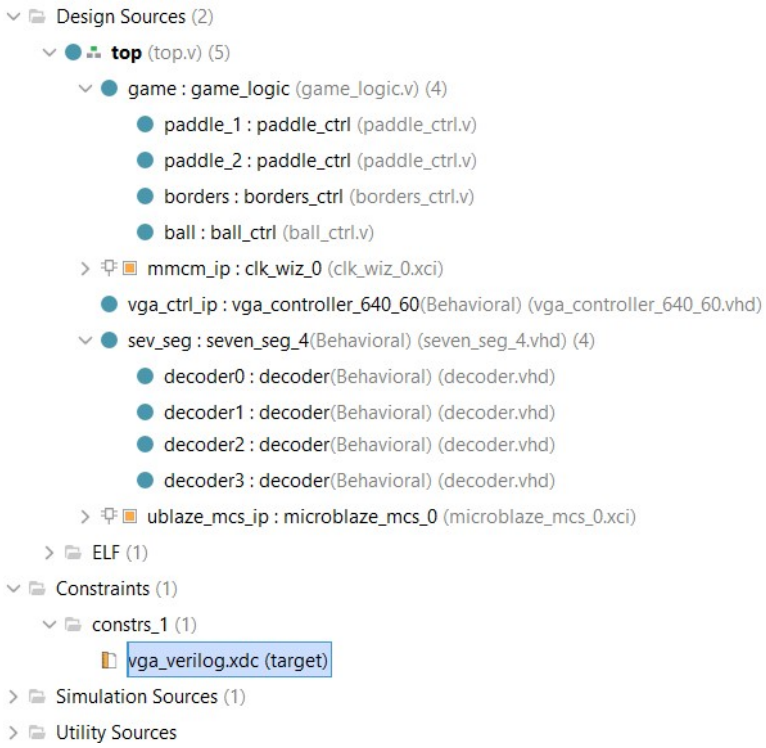

Fig. 3 – The hierarchy of the functional modules that underlie the project

Overall, the design includes 1 instance of *top.v*, 1 instance of *game_logic.v*, 1 instance of *clk_wiz_0.xci*, 1 instance of *vga_controller_640_60.vhd*, 1 instance of *seven_seg_4.vhd*, and 1 instance of *microblaze_mcs_0.xci*. In its turn, *game_logic.v* contains 2 instances of *paddle_ctrl.v*, 1 instance of *borders_ctrl.v*, and 1 instance of *ball_ctrl.v*. Also, *seven_seg_4.vhd* contains 4 instances of *decoder.vhd*. All these modules will be described in the next chapters.

2.3.1 top.v

The top-level block diagram of the game is presented on the Fig. 4.

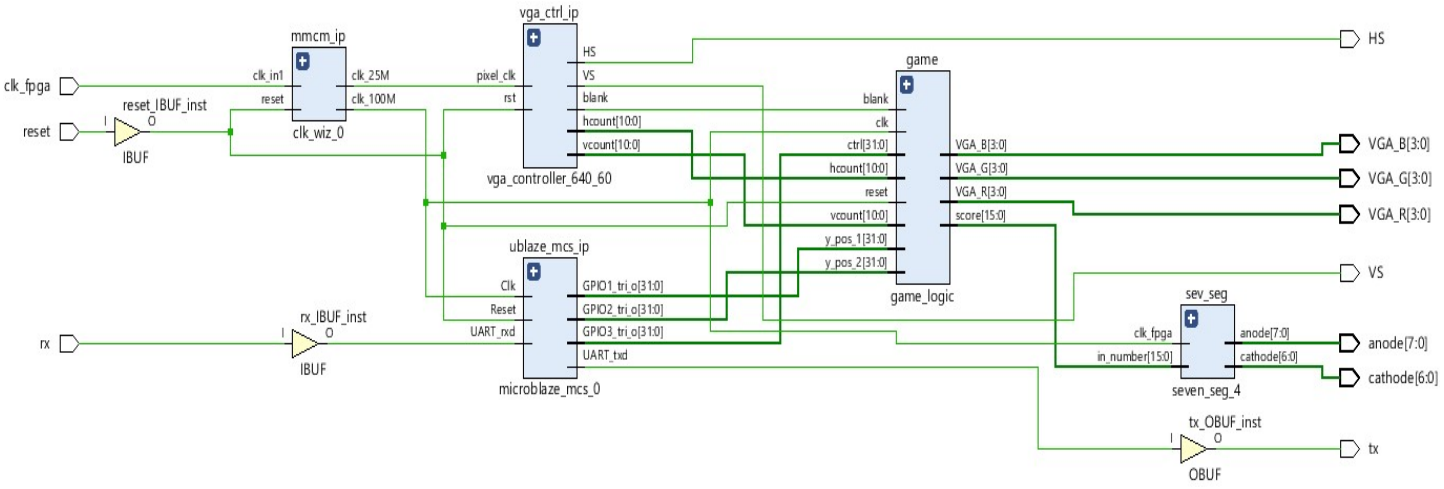

Fig. 4 – The block diagram of the project

This module interconnects all the instances of the custom blocks and the IP cores used in this design. The detailed description of the inputs and the outputs of the module is provided in Appendix A.

2.3.2 game_logic.v

The diagram of the *game_logic* entity is shown on the Fig. 5.

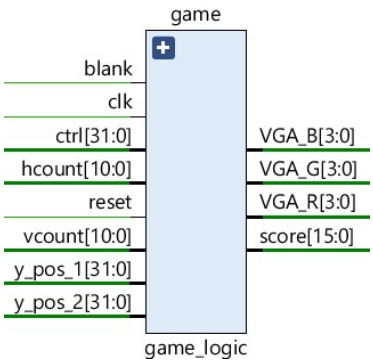

Fig. 5 – The diagram of the entity describing the game logic

The description of the inputs and the outputs of the module is provided in Appendix A.

This module draws the elements of the game field and defines the game logic via a simple state machine (Fig. ***6***).

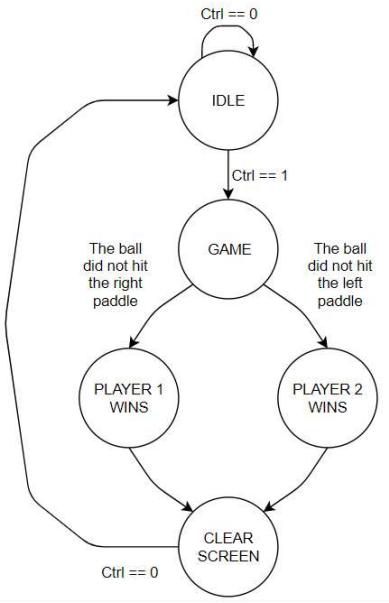

Fig. 6 – The state machine of the game logic

If the control signal is passive (0), we stay in the “IDLE” state. Otherwise, we will go to the “GAME” state where the ball is moving across the game field. If the ball’s x-position coincides with the x-position of the front edge of a paddle, but the y-position of the ball belongs to the space outside the front edge of a paddle, then a player missed a ball. In this case, we will go to one of the states “PLAYER 1 WINS” or “PLAYER 2 WINS” where we will update the score for a player. After that we will go to the “CLEAR SCREEN” state where we will wait when the control signal becomes equal to zero. After that, the consequence of the states will repeat.

2.3.3 paddle_ctrl.v

The diagram of the *paddle_ctrl* entity is shown on the Fig. 7.

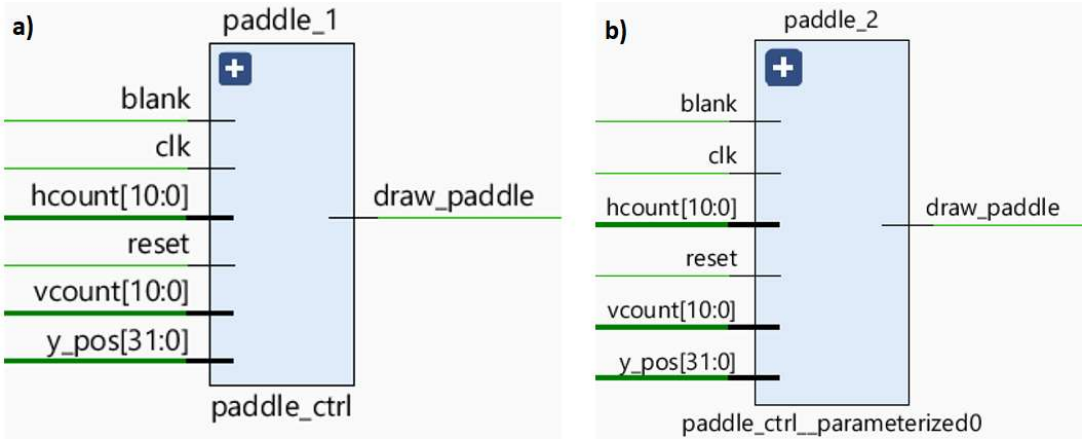

Fig. 7 – The diagram of the entity describing a paddle:
a – using the default x-position, b – using the user defined x-position

The description of the inputs and the outputs of the module is provided in Appendix A.

This module describes the movement a paddle. It uses the input signal y_pos[31:0] to update the position of the paddle. It uses the input signals hcount[10:0], vcount[10:0], and blank to form the output signal draw_paddle which is active (1) when the paddle should be drawn on the screen.

2.3.4 borders_ctrl.v

The diagram of the *borders_ctrl* entity is shown on the Fig. 8.

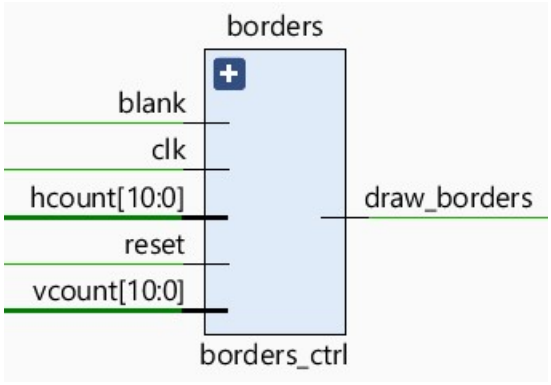

Fig. 8 – The diagram of the entity describing the borders

The description of the inputs and the outputs of the module is provided in Appendix A.

This module describes the upper and the lower borders on the game field. It uses the input signals hcount[10:0], vcount[10:0], and blank to form the output signal draw_borders which is active (1) when the borders should be drawn on the screen.

2.3.5 ball_ctrl.v

The diagram of the *ball_ctrl* entity is shown on the Fig. 9.

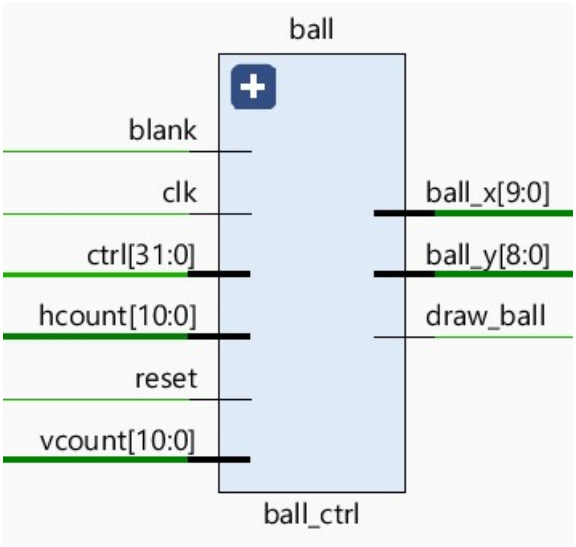

Fig. 9 – The diagram of the entity describing the ball

The description of the inputs and the outputs of the module is provided in Appendix A.

This module describes the movement of a ball. It uses the input signals hcount[10:0], vcount[10:0], blank, and ctrl[31:0] to form the output signal draw_ball which is active (1) when the ball should be drawn on the screen. The outputs ball_x[9:0] and ball_y[8:0] correspond to the current x- and y-positions of the ball and used in the instance of game_logic.v to control the location of the ball.

2.3.6 seven_seg_4.vhd

The diagram of the *seven_seg_4* entity is shown on the Fig. 10.

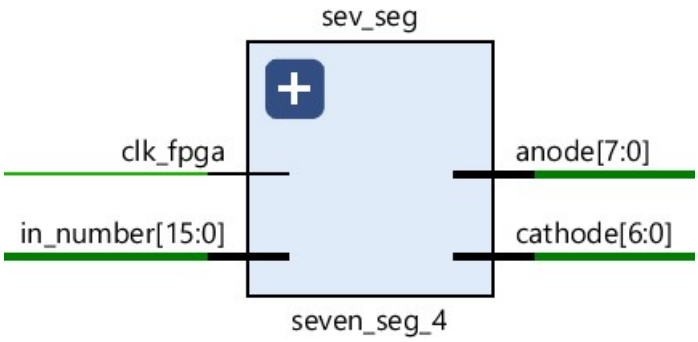

Fig. 10 – The diagram of the entity describing the 4-digit seven segment display

The description of the inputs and the outputs of the module is provided in Appendix A.

This module represents the 4-digit seven-segment display. It takes input 16-bit binary number (in_number[15:0]) and converts it into the pattern of the anodes (anode[7:0]) and cathodes (cathode[6:0]).

2.3.7 decoder.vhd

The diagram of the *decoder* entity is shown on the

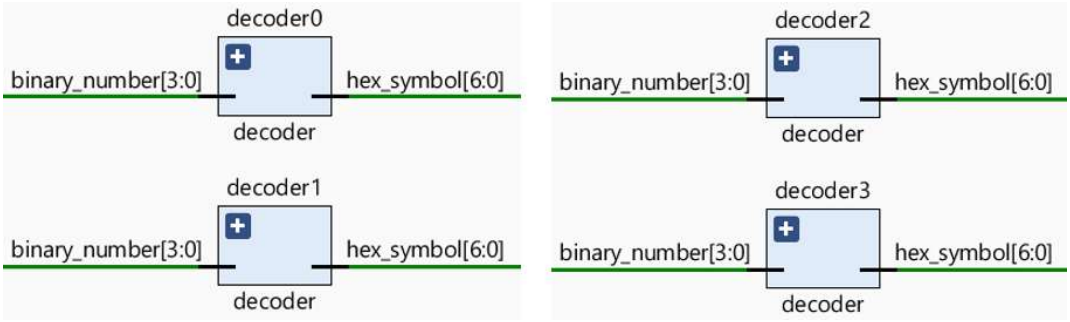

Fig. 11 – The diagram of the entity describing a converter of binary numbers into hexadecimal symbols

The description of the inputs and the outputs of the module is provided in Appendix A.

This module converts a binary number (binary_number[3:0]) to a hex symbol (hex_symbol[6:0]) to be displayed on a seven-segment indicator.

2.3.8 vga_controller_640_60.vhd

The diagram of the *vga_controller_640_60* entity is shown on the Fig. 12.

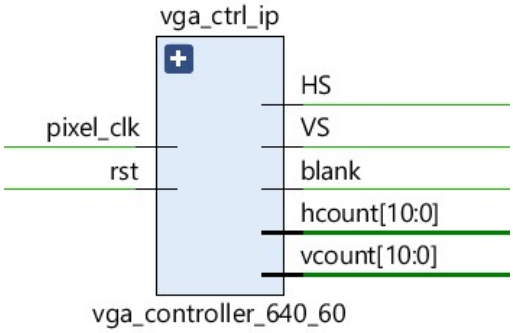

Fig. 12 – The diagram of the VGA Controller IP core

This IP core generates the video synch pulses for the VGA monitor to enter 640x480@60Hz resolution state. It also provides horizontal and vertical counters for the currently displayed pixel and a blank signal that is active when the pixel is not inside the visible screen and the color outputs should be reset to 0 [3].

2.3.9 clk_wiz_0.xci

The diagram of the *clk_wiz_0* entity is shown on the Fig. 13.

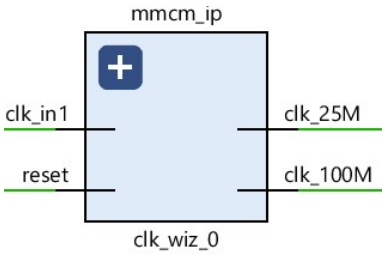

Fig. 13 – The diagram of the MMCM Clock Wizard IP core

This IP core was used to generate the clock of 25MHz for the instance of the VGA Controller IP core [3] and the clock of 100MHz for the instance of the MicroBlaze MCS IP core [5] and other logic in this design.

2.3.10 microblaze_mcs_0.xci

The diagram of the *microblaze_mcs_0* entity is shown on the Fig. 14.

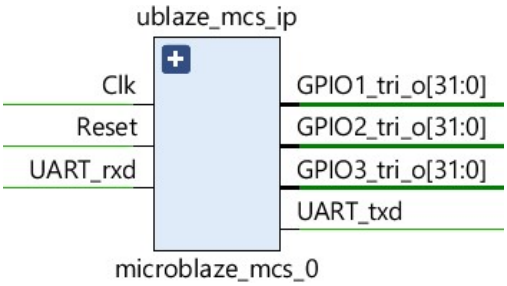

Fig. 14 – The diagram of the MicroBlaze MCS IP core

This IP core describes a light-weight version of the Microblaze softcore (embedded) processor. In this design, it is programmed in C to read out the keys pressed on the keyboard by the players: the button 'w' moves the left paddle 16 pixels up, the button 's' moves the left platform 16 pixel down. The button 'i' moves the right paddle 16 pixels up, the button 'k' moves the right platform 16 pixel down. The button ‘r’ is used to run/restart a game.

MicroBlaze IP parameters are presented in Table 1.

Table 1 – Parameters of MCS

Parameter	**Value**
Input Clock Frequency	100 MHz
Memory Size	32 KB
Optimization	AREA
UART	enable receiver – yes, enable transmitter – yes, baud rate = 9600, #data bits = 8
GPO	Channel 1, 2 (the y-positions of the paddles), Channel 3 (the control of the game: run/restart)
GPI	-

3. Output Graphical and Text Data

In this chapter, the output graphical and text data produced by the implemented modules are shown.

On the **Fig. *15***, the graphical interface of the game is presented. It includes the upper and the lower borders of the game field, two paddles, and the ball.

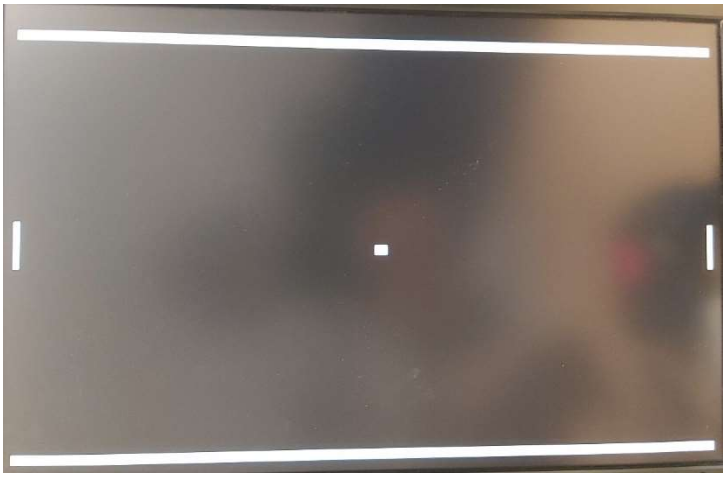

Fig. 15 – The graphical interface of the game

The 4-digit seven segment display reflects the current score in the game (**Fig. *16***).

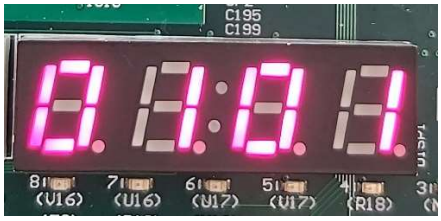

Fig. 16 – The 4-digit seven segment display reflects the current score in the game (here, 1:1)

The current y-coordinates of the paddles and the information about the developer are shown on the PuTTY terminal (Fig. 17).

```
Current date: 12/4/2019, Developer: Vladimir Vakhter
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 216, y_player_2 = 216
y_player_1 = 200, y_player_2 = 216
y_player_1 = 184, y_player_2 = 216
y_player_1 = 168, y_player_2 = 216
```

Fig. 17 – The console output shows the current y-coordinates of the paddles

4. FPGA Resource Usage

In this chapter, the information on the FPGA resource usage is provided.

The design utilizes 1.77% of lookup tables (LUT), 0.96% of LUTRAM (“distributed RAM”), 0.83% of flip-flops (FF), 5.93% of BRAM (“block RAM”), 15.71% of Input/Output Blocks (IO), and 16.67% (1/6) of MMCM (clock manager) (Fig. 18).

Resource	Utilization	Available	Utilization %
LUT	1122	63400	1.77
LUTRAM	182	19000	0.96
FF	1056	126800	0.83
BRAM	8	135	5.93
IO	33	210	15.71
MMCM	1	6	16.67

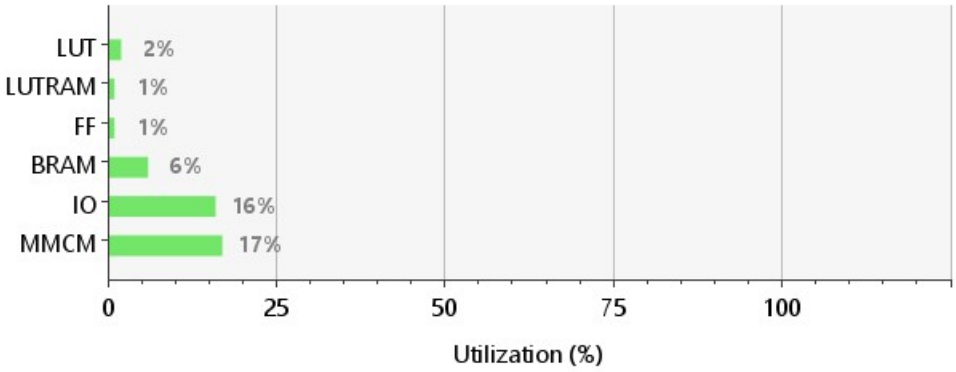

Fig. 18 – Utilization of resources by the design

The more detailed information on the resource distribution among the modules is presented on the Fig. 19.

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
N top	1122	1056	32	8	33	3	1
game (game_logic)	257	81	0	0	0	0	0
ball (ball_ctrl)	107	59	0	0	0	0	0
borders (borders_ctrl)	0	1	0	0	0	0	0
paddle_1 (paddle_ctrl)	20	1	0	0	0	0	0
paddle_2 (paddle_ctrl__parameter	20	1	0	0	0	0	0
mmcm_ip (clk_wiz_0)	0	0	0	0	0	3	1
sev_seg (seven_seg_4)	9	20	0	0	0	0	0
ublaze_mcs_ip (microblaze_mcs_0)	776	930	32	8	0	0	0
vga_ctrl_ip (vga_controller_640_60)	92	25	0	0	0	0	0

Fig. 19 - Utilization of resources by modules

5. Warning messages

In this chapter, the information on the warning messages will be provided.

There were not any warnings related to synthesis and implementation observed. The only warnings that appeared were referred to Out-of-Context Module Runs, i.e. MMCM and Microblaze (IP cores). But, these warnings did not affect the correctness of the work of the hardware design.

The configuration bank voltage select (CFGBVS) was set equal to VCCO and the configuration voltage (CONFIG_VOLTAGE) was set equal to 3.3V by adding the following lines to the constraints (.xdc) files:

- set_property CFGBVS VCCO [current_design]
- set_property CONFIG_VOLTAGE 3.3 [current_design]

It allowed avoiding the warning "[DRC CFGBVS-1] Missing CFGBVS and CONFIG_VOLTAGE".

6. Conclusions

During this project, the Pong game was developed in Verilog and tested.

The design of this game demanded the application of a broad range of approaches:

- Decomposition of a complex project into multiple smaller independent parts in order to easier design, test, debug, and reuse these specified modules.
- Implementation of standard elements such as state machines and counters.
- Utilization of IP cores in order to accelerate the development cycle.
- Particularly, the use of a softcore embedded processor, Microblaze MCS.
- Programming in C.

The following lessons learned from the project:

- Complex projects should be divided in smaller ones. It accelerates and simplify the design process.
- Smaller independent modules may be reused in the next projects.
- Before implementation, the fundamental parts of the design should be developed in a form of block diagrams, state machines, flow charts, etc. This approach accelerates the design flow.
- All outputs, if-else and case statements should have default values in order to avoid latches in combinational logic.
- Both in an HDL code and a C code, it is a good practice to implement symbolic names for numerical values. It makes the code more readable (because of the lack of "magic numbers"), easier to debug, and more flexible in terms of changing the values.
- Code should be commented or "self-commented" when the name of a variable shows its purpose. It also helps to reuse code or verify it by another developer.

References

[1] D. Y. Doroz, "Design and implementation (in Verilog) of a memory interface along with a Microblaze embedded processor," [Online]. Available: http://users.wpi.edu/~ydoroz/ece574/assignments/ece574_2019_proj3.pdf. [Accessed 1 Dec. 2019].

[2] "Nexys A7 FPGA Board Reference Manual," Digilent, [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual. [Accessed 7 Oct. 2019].

[3] "VGA Controller," Digilent, [Online]. Available: http://users.wpi.edu/~ydoroz/ece574/tutorials/vga_controller_640_60.vhd. [Accessed 7 Oct. 2019].

[4] J. Duckworth, "MMCM Tutorial," [Online]. Available: http://users.wpi.edu/~ydoroz/ece574/tutorials/MMCM%20Vivado%20example%20VHDL.pdf. [Accessed 7 Oct. 2019].

[5] J. Duckworth, "Microblaze MCS Tutorial," [Online]. Available: http://users.wpi.edu/~ydoroz/ece574/tutorials/Microblaze%20MCS%20Tutorial%20Vivado%20v3.pdf. [Accessed 10 Nov. 2019].

[6] "Master XDC Files," Digilent, [Online]. Available: https://github.com/Digilent/digilent-xdc/. [Accessed 7 Oct. 2019].

[7] "Nexys A7 Schematic," Digilent, [Online]. Available: https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7-sch.pdf. [Accessed 7 Oct. 2019].

[8] Xilinx, "DS865 Product Specification for Microblaze Micro Controller System," [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/ds865_microblaze_mcs.pdf. [Accessed 10 Nov. 2019].

[9] D. Y. Doroz, "Design and implementation (in Verilog) of a memory interface along with a Microblaze embedded processor," [Online]. Available: http://users.wpi.edu/~ydoroz/ece574/assignments/ece574_2019_proj2_micro_with_arith.pdf. [Accessed 10 Nov. 2019].