

DESIGNING A GAME ON FPGA USING VERILOG

LOH YI QHI

**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Engineering
(Honours) Electrical and Electronic Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

September 2020

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : Loh

Name : LOH YI QHI


ID No. : 1700781

Date : 13th September 2020

APPROVAL FOR SUBMISSION

I certify that this project report entitled "**DESIGNING A GAME ON FPGA USING VERILOG**" was prepared by **LOH YI QHI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	 _____
Supervisor	:	Dr. Khaw Mei Kum _____
Date	:	13 th September 2020 _____

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2020, LOH YI QHI. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Khaw Mei Kum for her invaluable advice, guidance and her enormous patience throughout the development of the research.

In addition, I would like to express my appreciation to the lab assistant, Mr. Teo Gee Man, who borrowed the equipment to me and helped in the completion of this project.

Lastly, I would also like to express my gratitude to my loving parents and friends who had helped and given me supports and encouragement.

ABSTRACT

As the rate of technological advancement increases with time, the market of FPGA has grown dramatically and becomes popular among the field of ASIC. Ease of creation and maintenance lead FPGA to become an attractive solution to high speed and efficient applications. From a simple interface circuit to a complex state machine, even to the extent of System on Chip (SOC), the importance of FPGA chips cannot be ignored. Several testings were carried out in this project before the start of specific technical design in order to avoid any unwanted syntax errors and equipment wiring errors.

This project mainly focused on the design and implementation of an FPGA-based Ping Pong game, consisting of both the hardware and software design working coherently. For hardware-wise, the host computer was used to program and configure the design; FPGA was implemented for operating the game, and the VGA monitor for display. At the same time, the software part included the design of the overall system, input key module, VGA display module, as well as the game control module.

The game design focused on "Double Player Mode", which simulates the real-life ping-pong game. The movement of the paddles was controlled by pressing the push buttons of FPGA. The difficulty level of the game was increased by adding the extra features, which includes the speed control of the ball and the size control of the paddles. The judgment of victory and defeat was done by comparing the accumulated points of the players. If one of the paddles unable to catch the ball and reflect it, the ball will continue to move and touch the up or down borderline of the rectangular frame, then the opponent will earn fifteen points, and the next round began. The game-winner is who first accumulated to 90 points. The game will restart again if the player inputs the RESET key.

At the end of the project, the compilation of the game design is found to be successful. The game can be displayed and functioned smoothly without any delay. Thus, this project is considered successful as the objectives were achieved.

TABLE OF CONTENTS

DECLARATION	i
APPROVAL FOR SUBMISSION	ii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF SYMBOLS / ABBREVIATIONS	xiv
LIST OF APPENDICES	xv

CHAPTER

1	INTRODUCTION	1
	1.1 General Introduction	1
	1.2 Importance of the Study	2
	1.3 Problem Formulation	3
	1.4 Aim and Objectives	4
	1.5 Scope and Limitation of the Study	5
	1.6 Contribution of the Study	5
	1.7 Outline of the Report	5
2	LITERATURE REVIEW	7
	2.1 Introduction	7
	2.2 Evolution of Programmable Logic Devices (PLDs)	9
	2.3 Strengths of FPGA	14
	2.4 Computer-Aided Design (CAD) Flow for FPGA	15
	2.5 Application of FPGA	16
	2.6 Game Development on FPGA	17
	2.6.1 Dice Game Development on FPGA	17
	2.6.2 Tetris Game Development on FPGA	22
	2.7 Summary	27
3	METHODOLOGY AND WORK PLAN	28
	3.1 Introduction	28
	3.2 Project Planning	29

	3.2.1	Scope Planning	29
	3.2.2	Schedule Planning	30
	3.3	Equipment Learning	31
	3.3.1	Block Diagram of FPGA DE2-115	31
	3.3.2	USB-Blaster	31
	3.3.3	Quartus II software	32
	3.4	Experimental Planning and Setup	33
	3.5	Preliminary Testing and Verification	33
	3.5.1	Testing of CAD Tool	34
	3.5.2	Testing of FPGA board	39
	3.5.3	Testing of VGA monitor	42
	3.6	Experiment on Game Development: Ping Pong Game	42
	3.6.1	System Overall Design	43
	3.6.2	Design of input key module	44
	3.6.3	Design of VGA module	45
	3.6.4	Design of Game Control module	49
	3.7	Summary	55
4		RESULTS AND DISCUSSION	56
	4.1	Introduction	56
	4.2	Result of Preliminary Testing	56
	4.2.1	Testing of CAD Tool	56
	4.2.2	Testing of FPGA	59
	4.2.3	Testing of VGA monitor	61
	4.3	Experiment on Game Development: Ping Pong Game	61
	4.3.1	Pin Assignment	61
	4.3.2	Compilation Result	62
	4.3.3	JTAG Configuration	63
	4.3.4	VGA Display	64
	4.3.5	Results of Game Control Module	66
	4.4	Summary	70
5		CONCLUSIONS AND RECOMMENDATIONS	71
	5.1	Conclusions	71

5.2	Recommendations for future work	72
REFERENCES		73
APPENDICES		75

LIST OF TABLES

Table 2.6.1 Function of the core program modules	24
Table 2.6.2 Game key function table	24
Table 2.6.3 Game state machine description	25
Table 3.2.1 Project tasks with the estimated duration	29
Table 3.6.1 List of Input Keys with its corresponded command	45
Table 3.6.2 Truth table to show the change of paddle color	48
Table 3.6.3 Truth table to show the change of ball color	49
Table 3.6.4 Speed control of the ball	52
Table 3.6.5 Size of the paddle according to its respective binary state	53
Table 4.2.1 Comparison of theoretical and practical result	58
Table 4.2.2 Summarized result to show the runtime behaviour of FPGA	60
Table 4.3.1 Display of paddle color according to their respective binary state	64
Table 4.3.2 Display of the ball color according to their respective binary state	65
Table 4.3.3 Display of the paddle size according to their respective binary state	67

LIST OF FIGURES

Figure 1.1.1 DE2-115 FPGA board - Top view	1
Figure 1.1.2 DE2-115 FPGA board - Bottom view	2
Figure 1.3.1 Planned experimental setup	4
Figure 2.1.1 Types of ASICs	8
Figure 2.2.1 Block Diagram of a PROM	9
Figure 2.2.2 Block Diagram of a PLA	10
Figure 2.2.3 Block Diagram of a PAL	10
Figure 2.2.4 Block Diagram of GAL	11
Figure 2.2.5 Categorization of SPLDs	11
Figure 2.2.6 Architecture of CPLD	12
Figure 2.2.7 Architecture of FPGA	13
Figure 2.6.1 Flow chart of VHDL design	19
Figure 2.6.2 Flow chart of VHDL design	20
Figure 2.6.3 System implementation flow	21
Figure 2.6.4 The display on the DE2-115 board when the user wins	21
Figure 2.6.5 The display on the DE2-115 board when the computer wins	22
Figure 2.6.6 Categorization of the main body of the Tetris game	23
Figure 2.6.7 Structure of the program module	23
Figure 2.6.8 Transition diagram of the game state.	24
Figure 2.6.9 Graphic display method	26
Figure 2.6.10 Character display method	26
Figure 2.6.11 Dividing of the area of game and feature.	26
Figure 3.1.1 Design flow of the project	28

Figure 3.2.1 Gantt chart of the project	30
Figure 3.3.1 Block diagram of DE2-115	31
Figure 3.3.2 Block Diagram of the DE2-115 Control Panel	32
Figure 3.3.3 Interface of Quartus II	32
Figure 3.4.1 Experimental Setup	33
Figure 3.5.1 CAD Flow for FPGA	34
Figure 3.5.2 Logic Circuit Design	35
Figure 3.5.3 Device Assignment	35
Figure 3.5.4 Design Entry using Verilog Code	36
Figure 3.5.5 Process of compiling the project	36
Figure 3.5.6 The result of compilation	37
Figure 3.5.7 Message window of Quartus II	37
Figure 3.5.8 Message box showing "Error"	38
Figure 3.5.9 Pin Assignment (Testing of CAD Tool)	38
Figure 3.5.10 Interface of Modelsim	39
Figure 3.5.11 Design Entry in Verilog (.v)	40
Figure 3.5.12 Design Entry in Block Design File (.bdf)	40
Figure 3.5.13 Creating PLL Megafunction using MegaWizard Plug-In Manager	41
Figure 3.5.14 Creating Multiplexer using MegaWizard Plug-In Manager	41
Figure 3.5.15 Overall Block Design File	42
Figure 3.5.16 Pin Assignment (Testing of FPGA)	42
Figure 3.6.1 Design flow of a Ping-pong game	43
Figure 3.6.2 Overall System design of Ping-pong game	44
Figure 3.6.3 Input keys of the game on DE2-115	45
Figure 3.6.4 Connections between FPGA and VGA	46

Figure 3.6.5 VGA interface with resolution 640x480	47
Figure 3.6.6 Code fragment to design the VGA display (640x480)	47
Figure 3.6.7 Resolution setting of monitor (640x480)	47
Figure 3.6.8 Code fragment to change the paddle color (Player A)	48
Figure 3.6.9 Code fragment to change the paddle color (Player B)	48
Figure 3.6.10 Code fragment to change the ball color	49
Figure 3.6.11 Code fragment to assign the command to the input key	50
Figure 3.6.12 Code fragment to indicate the initial location of the ball and paddles	50
Figure 3.6.13 Code fragment to indicate the movement of the ball and paddles	51
Figure 3.6.14 Code fragment to control the speed of the ball	52
Figure 3.6.15 Code fragment to change the paddle size (Player A)	53
Figure 3.6.16 Code fragment to change the paddle size (Player B)	53
Figure 3.6.17 Code fragment to indicate the points accumulation	54
Figure 3.6.18 Design of seven segment display module	54
Figure 3.6.19 Design of LCD module	55
Figure 4.2.1 Compilation Result in Quartus II	56
Figure 4.2.2 Compilation Result in Modelsim	56
Figure 4.2.3 Simulation Result in Modelsim	57
Figure 4.2.4 Process of loading the configuration data into FPGA (successful)	57
Figure 4.2.5 Compilation Report (Successful)	59
Figure 4.2.6 FPGA configuration (successful)	59
Figure 4.2.7 Display on VGA monitor when connected to the host computer	61
Figure 4.3.1 Pin Planner (Ping Pong Game)	62

Figure 4.3.2 Compilation Result (Successful)	62
Figure 4.3.3 Process of loading configuration data into FPGA (Successful)	63
Figure 4.3.4 Initial location of the ball and paddles	66
Figure 4.3.5 Start of the game	68
Figure 4.3.6 Player B gained 45 points	68
Figure 4.3.7 Player A gained 30 points while Player B gained 60 points	68
Figure 4.3.8 Display on the DE2-115 board when Player A wins	69
Figure 4.3.9 Display on the DE2-115 board when Player B wins	69

LIST OF SYMBOLS / ABBREVIATIONS

FPGA	Field Programmable Gate Array
SOC	System On Chip
IC	Integrated Circuit
ASIC	Application Specific Integrated Circuit
PLD	Programmable Logic Device
SPLD	Simple Programmable Logic Device
CPLD	Complex Programmable Logic Device
PAL	Programmable Array Logic
PLA	Programmable Logic Array
PROM	Programmable Read-Only Memory
MPGA	Mask-Programmable Gate Array
VGA	Video Graphic Array
AI	Artificial Intelligence
DNN	Deep Neural Network
CPU	Computer Processing Unit
GPU	Graphics Processing Unit
SOP	Sum of Product
CAD	Computer Aided Design
LED	Light Emitting Diode
ROM	Read-Only Memory
RAM	Random Access Memory
BCD	Binary Coded Decimal
I/O	Input / Output
PLL	Phase-Locked Loops
LE	Logic Element

LIST OF APPENDICES

APPENDIX A	75
APPENDIX B	84
APPENDIX C	89
APPENDIX D	90
APPENDIX E	92

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In the field of engineers, besides the required research work, the development of games is always in the mind of engineers for entertainment. With the expansion of electronics evolution, engineers can explore and add more creativity to the design of games.

As the rate of technological advancement increases with time, the market of Field Programmable Gate Array (FPGA) has grown dramatically and becomes popular among the field of Application Specific Integrated Circuit (ASIC). Ease of creation and maintenance leads FPGA to become an attractive solution to high speed and efficient applications. There are various models of FPGA available in the market, in which the two biggest manufacturers are Xilinx and Altera (Intel). The project implemented an Altera DE2-115 FPGA development board with the usage of Verilog as the hardware description language. The architecture of DE2-115 is illustrated in Figure 1.1.1 and Figure 1.1.2 for the top view and bottom view, respectively.

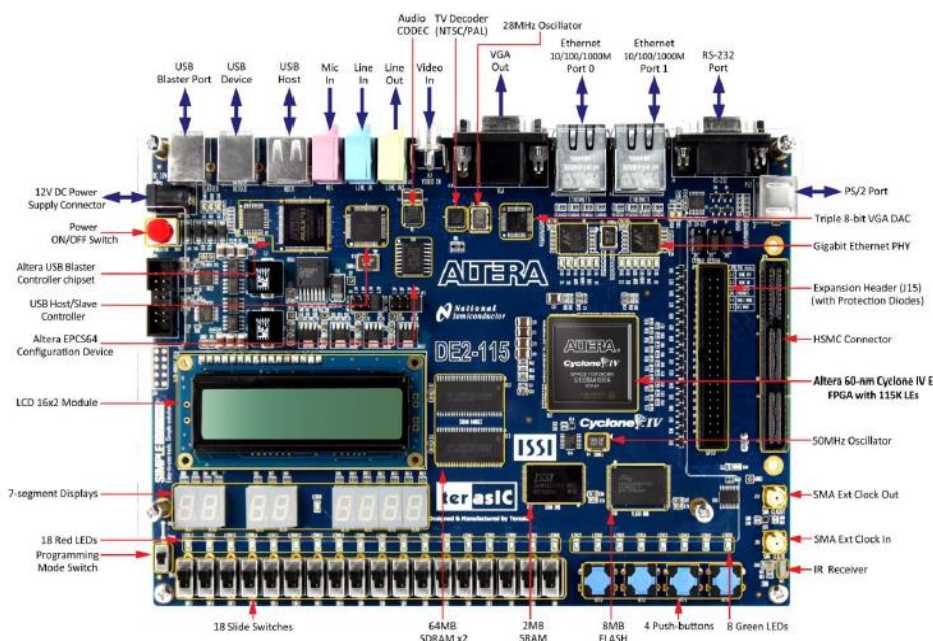


Figure 1.1.1 DE2-115 FPGA board - Top view (Altera Corporation, 2010)

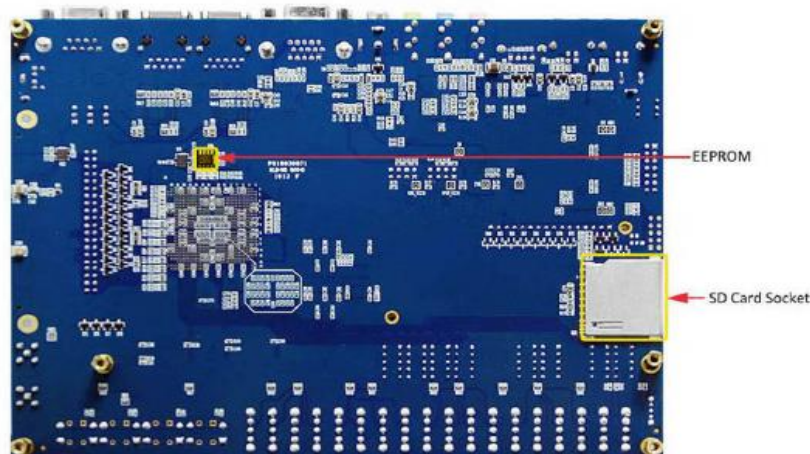


Figure 1.1.2 DE2-115 FPGA board - Bottom view (Altera Corporation, 2010)

The primary purpose of this project is to explore the world of FPGA by developing a Ping-pong game. It was one of the earliest arcade video games developed by Allan Alcorn. The game emulates the well-known ping-pong game and its rules. The game was designed to have "Double Player Mode". The system design was mainly focused on hardware and software design. The hardware part involved the host computer for design configuring and programming, FPGA for game operating as well as the VGA monitor for display. For the software-wise, the design included the design of overall system, input key module, VGA display module, and the game control module.

1.2 Importance of the Study

The main intention of this project is to explore the versatility of FPGA. One of the interesting applications of FPGA is game development. During the game design, a deeper understanding of FPGA is acquired. The main importance of this research is to introduce improvements based on previous works.

In other researchers' works, it may require lengthy scripts and loop to code the design; however, simplification could be done by shortening the scripts. This demonstrates the next importance of the study.

There are several PLDs available in the market that could be used in game development, for example, CPLD. To find out the most suitable approach and the rationale behind FPGA being chosen for most of the game design is also considered as one of the importance of the study.

Furthermore, it is vital to have improved from the previous work in terms of its game complexity. This project modified the "Single Player Mode" or so-called "Robot Mode" of Ping-pong game from other researcher's work and was enhanced to become "Double Player Mode". Also, instead of using the seven segments displays of FPGA, the improvements were made by the implementation of VGA for the game display.

The knowledge gained in this project is meant to be helpful in the future as FPGA has an enormous potential in the electronic field due to its wide range of applications including from equipment for imaging and video to computer circuitry, aerospace, automobile, and military applications, in addition to electronics for specified processing.

The world is now proceeding to the Fourth Industrial Revolution. More industries are emerging into the implementation of Artificial Intelligence (AI). The research on FPGA can be further implied in the future work, including application with high potential, AI since FPGA is gaining prominence in deep neural networks (DNN) that are applied for AI. (Touger, 2018) As the demand for AI is increasing, the demand for FPGA will be boosted accordingly. Therefore, it can be concluded that the learning of FPGA is significant, and it will be helpful for the future FPGA engineer.

1.3 Problem Formulation

This project is to remodel one of the pioneers of computer gaming history, which is the Ping-pong game. In order to make this project to be a little more complicated, the game was implemented on FPGA. The FPGA was programmed using the Verilog hardware description language.

The ping-pong game is a simple game application, but with many rooms for improvement. This project encourages the enhancement of the game from a single-player (Robot Mode) to double players (Double Mode). Also, the scoring system is one of the most critical steps to decide the winner of the game, and it has to be well-planned. Next, the graphic display of the game acts as the first image to the users, which means a well-designed graphic display is a crucial element to attract the users. Therefore, an improvement in graphic design is also

one of the expected targets for this project, for example, the color of the paddles and ball. This project was also planned to increase the difficulty of the game by allowing the player to change the size of the paddles as well as increase the speed of the ball.

The hardware-implemented in this project is the Altera DE2-115 FPGA board, as shown in Figure 1.1.1. The game can be controlled by using the pushbuttons and slide switches from the FPGA board to key in the input for game operation. Moreover, a monitor was included in the design for game display through VGA cable. The experimental setup was planned, as illustrated in Figure 1.3.1.

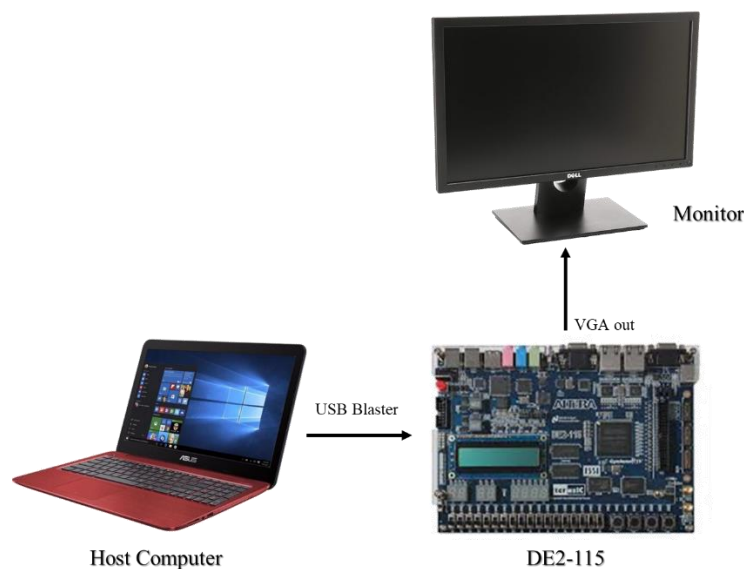


Figure 1.3.1 Planned experimental setup

1.4 Aim and Objectives

The main objectives of this project are:

1. To design a game on FPGA using Verilog.
2. To imply a relevant theory of FPGA to real-life application.
3. To highlight the benefits of FPGA by increasing the complexity of the game with creativity.

1.5 Scope and Limitation of the Study

This project mainly focused on the design of the Ping-pong game on FPGA. Nevertheless, some limitations are not covered in the scope of the project.

Firstly, the game design only covered the graphic design module, the input control module, and the game control module. However, the design does not involve the audio module, which means that the game is without music or sound.

Next, the design was specified to be controlled by the pushbuttons and slide switches of FPGA only, meaning that the other controllers will not take into consideration.

Besides, the VGA setting was set to be 60 Hz with a resolution of 640 x 480. Therefore, the resolution of the monitor also has to be set at 640 x 480 to meet the requirement. The specification and preliminary setting might limit the flexibility of the development. However, it will not bring a significant impact on the game application. The modification of code can be done to change the resolution setting.

1.6 Contribution of the Study

This project contributes to different design applications of FPGA, which is not only limited to the game design. The necessary information regarding FPGA can be found from this project, such as the evolution of PLDs, working principle, features, CAD tool, FPGA's strengths, design considerations as well as the interesting applications.

Besides that, this project introduced the Verilog HDL in the design. Throughout the project, there are limited sources of code examples. Therefore, this project could be helpful and acts as a reference and the contribution for future engineers who are interested in the FPGA design, especially for the beginners.

1.7 Outline of the Report

Chapter 1 provided a brief introduction of the project by discussing the importance of the study, problem formulation, aim, and objectives of the project, scope, and limitations of the study as well as the contribution of the study.

In Chapter 2, the literature review introduced the evolution of PLDs, highlighted the strength of FPGA, and discussed the CAD flow as well as the application of FPGA. It also described what others have done for the game development, justifying the use of specific techniques or problem-solving procedures and hence sets a benchmark for the current project.

Chapter 3 focused on the evidence of planning and organization to achieve milestones and demonstrate problem-solving skills by providing the experimental methods and solutions to achieve the objectives of the project. Various tests will be performed before carrying out the actual experiment to avoid any unnecessary works.

Chapter 4 mainly focused on demonstrating the results obtained from the experiment and discussed in detail to answer the problem stated in Chapter 1.

In Chapter 5, the overall conclusion will be provided as well as the recommendation for future work so that the project can be enhanced and improved by other researchers in the future.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The concept of the Fourth Industrial Revolution declares that the transformation of technologies is the critical element that drives society to be more digitized, organized, and enhanced in the aspect of productivity in manufacturing. One of the technologies that brought the idea of the Fourth Industrial Revolution in the production of electronics is the Integrated Circuit (IC). This technology diminished the size of the electronic products by enlarging the density of logic gates per chip. Certain types of IC are reprogrammable and can be applied in various applications, whereas some IC can only be implemented for one specific application, which is known as the Application Specific Integrated Circuit (ASIC). ASIC is a silicon chip that was created solely for a specific purpose, as opposed to a general-purpose chip that performs a wide range of functions but operates with lower efficiency (Elprocus, 2019). ASIC is said to be advantageous over general-purpose chip due to its tiny size, high-speed response, and low power consumption, which drives it to a higher chance for complex larger systems application. The uniqueness of ASIC makes it more preferred in high-level applications, which is usually employed in private data centers, public clouds, and shared devices around the world.

According to the research, ASIC can be categorized into three categories: full custom, semi-custom, and programmable. (Elprocus, 2019) Figure 2.1.1 shows the categorization of ASIC.

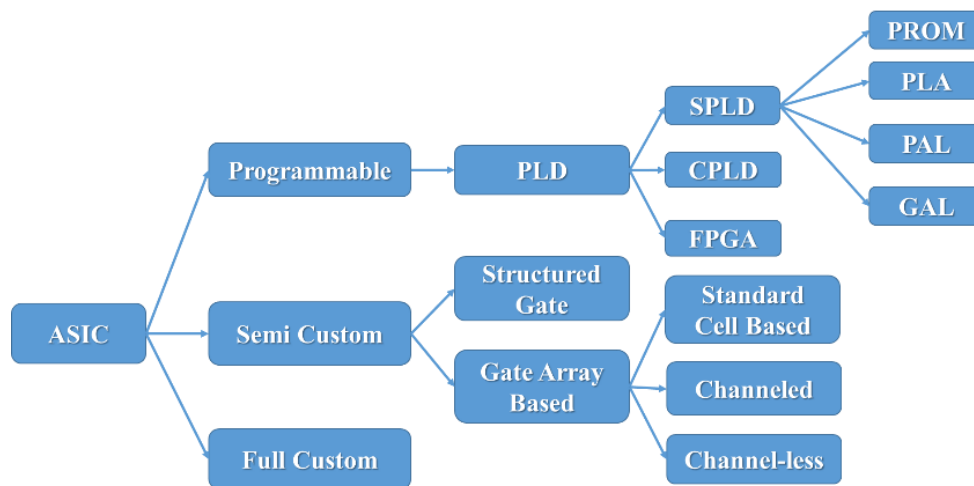


Figure 2.1.1 Types of ASICs

In full custom design, all the mask layers for interconnection are customized, while in semi-custom design, masks are partially customized, and the rest was taken from the pre-designed library. Customized design chips provide low flexibility for programming since they cannot be re-programmed and modified once they are manufactured. This problem can be substantially reduced by using Programmable ASIC, which is called a Programmable Logic Device (PLD). (Elprocus, 2019)

PLD is an integrated circuit with a large number of flip-flops and gates that can be configured with basic software to execute the logic for a specific function or perform a complex logic function. PLD provides high flexibility during the design cycle since it can be programmed or reconfigured to define the function based on the design considerations, unlike the logic gate, which has only a fixed function. The design changes can also be shown instantaneously in the working parts. In addition, lower power consumption, as well as fewer interconnections and packages significantly improve the reliability of the system and reduce the system complexity by simplifying the testing procedures. PLD offers reusable characteristics as it grants both design updating and error correction by re-programming. Since PLD is field programmable, it allows the user to program outside of the manufacturing area and thus increase the user flexibility. There are three types of PLD, namely Simple PLD (SPLD), Complex PLD (CPLD), and Field Programmable Gate Array (FPGA).

2.2 Evolution of Programmable Logic Devices (PLDs)

This topic covered the evolution of PLDs from the start of the revolution until today in terms of their type, architecture, functions, strengths, and drawbacks. According to the research of Brown, S., and Rose, J. (2002), Programmable Read-Only Memory (PROM) was the first development of a user-programmable chip that employed logic circuits, which provides the user the flexibility to program the binary information electrically. The address lines can be assigned as the inputs of logic circuits and the data lines as outputs. Figure 2.2.1 shows the block diagram of a PROM. PROM is built up of a fixed non-programmable AND array acted as a full decoder for its address inputs and a programmable OR array. PROM is used to store information at addressable locations, but it is inefficient for realizing logic circuits due to its hardware wastage issues and limited application. The limitations of PROM can be overcome in PLA and PAL.

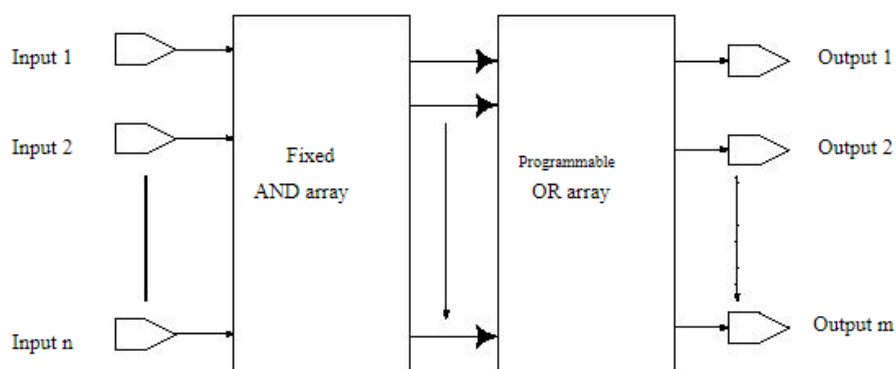


Figure 2.2.1 Block Diagram of a PROM (Brown, S. and Rose, J., 2002)

In the early 1970s, Field-Programmable Logic Array (FPLA or PLA) is developed specifically for implementing logic circuits. A PLA has two levels of logic gates, which include a programmable AND array followed by a programmable OR array, and it can be shown in Figure 2.2.2. PLA has been designed using the logic gates AND, OR, and NOT which was fabricated on the chip, making every input and its complement can be obtained toward every AND gate. PLA is well-suited for implying the logic functions in the sum of product (SOP) form by connecting the output of the AND gate to the OR gate while the OR gate output is used to generate the chip output. PLA is considered

as an adaptable device because both the AND and OR terms is allowed to have multiple inputs.

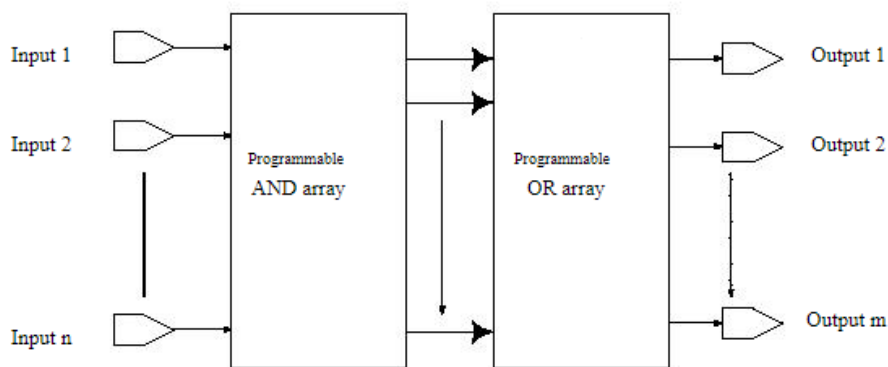


Figure 2.2.2 Block Diagram of a PLA (Brown, S. and Rose, J., 2002)

The main disadvantages of PLA were that they require high manufacturing costs and inadequate speed performance. Both drawbacks were caused by the two levels of programmable logic. The configurable logic planes directly increase the difficulty in manufacturing and hence lead to a significant increase in propagation delays. (Brown, S. and Rose, J., 2002) In order to overcome these limitations, Programmable Array Logic (PAL) devices were established. As illustrated in Figure 2.2.3, PAL is built by using a programmable AND array that feeds fixed OR gates, which consists of only a single level of programmability. PAL usually contains flip-flops connected to the outputs of the OR gate, compensating for the lack of generality incurred due to the fixed OR gates are used. PAL had a philosophical impact on the design of digital hardware and served as the foundation for some of the beginner and matured constitutions.

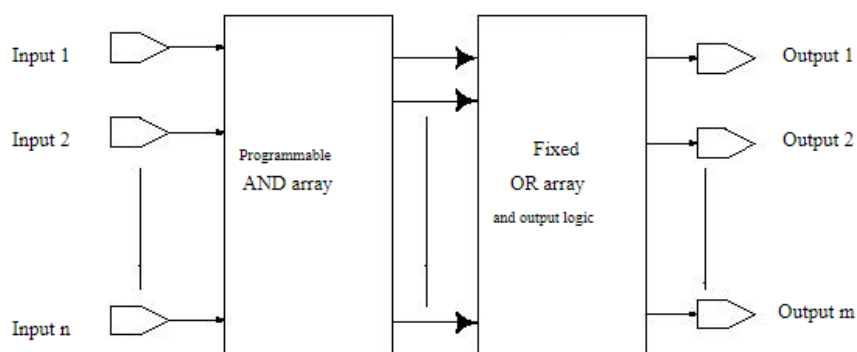


Figure 2.2.3 Block Diagram of a PAL (Brown, S. and Rose, J., 2002)

An improvement on the PAL was the Generic Array Logic (GAL). GAL comprises one AND array (reconfigurable) and one OR array (fixed) with programmable output logic, as shown below. Unlike the AND array of PAL that can only be programmed once, the reprogrammable AND array allows GAL to be programmed multiple times. As referred to the research of Brown, S. and Rose, J. (2002), Electrically Erasable CMOS is implemented in GAL rather than using fusible links and Bipolar technology. GAL uses AND arrays followed by OR arrays, and thus, it also allows the employment of the Boolean expressions SOP (sum of product).

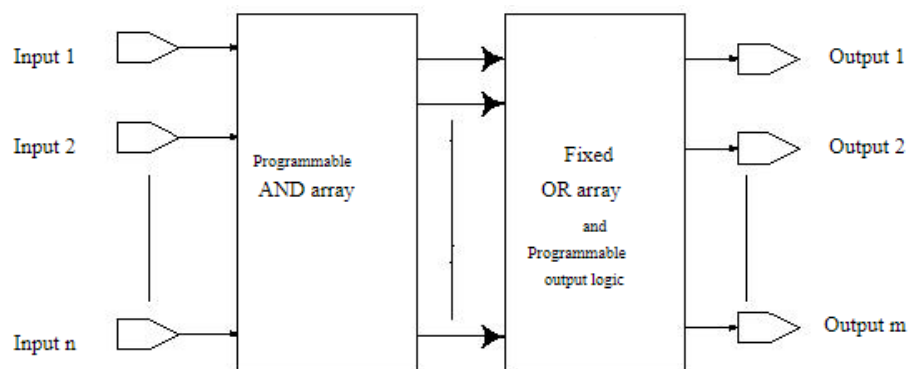


Figure 2.2.4 Block Diagram of GAL (Brown, S. and Rose, J., 2002)

The small size of PLDs, including PROMs, PALs, PLAs, and GALs are categorized as Simple PLDs (SPLDs), whose main strengths are low power consumption, more straightforward tracing process due to its smaller pin as well as high flexibility as the logic circuit designs can be altered without rewiring. The categorization of SPLDs can be shown in Figure 2.2.5.

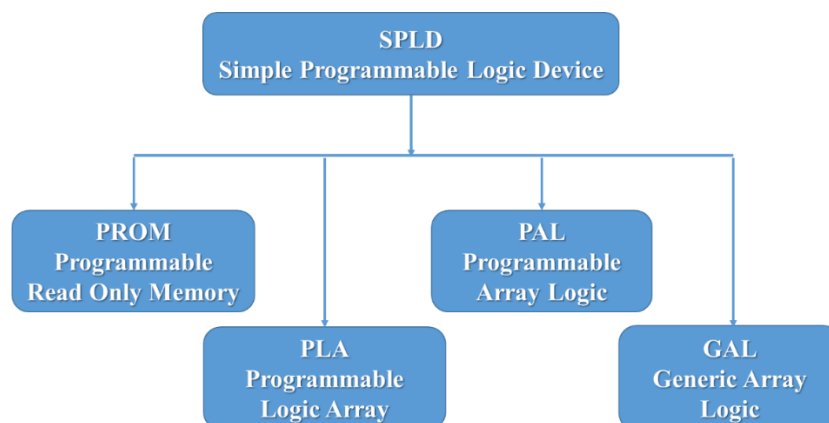


Figure 2.2.5 Categorization of SPLDs

However, the implementation of SPLD is only suitable for small digital circuits, which supports a mutual number of inputs and outputs of not more than 32. For designing a complex circuit, which requires more inputs and outputs, either combine multiple SPLDs onto a single chip or choose to implement a Complex Programmable Logic Device (CPLD).

CPLDs were first launched by Altera, three families of chips named MAX 5000, MAX 7000, and MAX 9000 were developed as mentioned by Brown, S. and Rose, J. (2002). With the recent advancements of technology, the demand for sizeable FPDs experience exponential growth. Other manufacturers start to invest in the development of devices in the CPLD category to cope with the complexity, and hence there are now many models available in the market. Figure 2.2.6 shows the architecture of CPLD that combines numerous circuit blocks (SPLD type devices) into one single chip, which is interconnected with each other by the global interconnection matrix. Each of its circuit blocks includes 8-16 macro-cells, and every logic block is responsible for a specific function. CPLD has been designed to handle knowingly higher complexity of logical functions and contributes to the logic capacity, which is equivalent to 50 standard SPLD devices. (Krazytech, 2015) Nevertheless, CPLD reached the maximum, and tough to extend these architectures to higher densities. Therefore, in order to create PLDs with very high logic capacity, another method should be implemented.

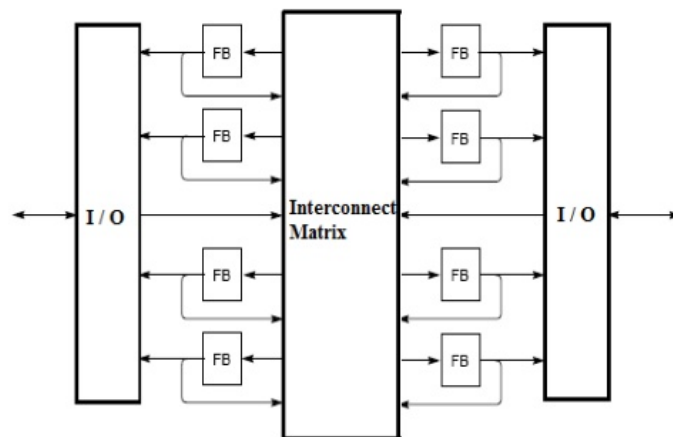


Figure 2.2.6 Architecture of CPLD (Brown, S. and Rose, J., 2002)

The highest capacity logic chips with general-purpose available nowadays are Mask-Programmable Gate Arrays (MPGA). It is structured by an array of pre-fabricated transistors, allowing the logic circuit to be customizable by joining the transistors with specific wires. The customization is actualized by specifying the metal interconnection during the chip fabrication process. It can be observed that the customization process will take a longer time in manufacturing, and hence the production cost will increase correspondingly. MPGAs are not classified under the category of field-programmable devices; however, the design of MPGA is motivated and carried to Field Programmable Gate Arrays (FPGA).

As similar to MPGAs, FPGAs consist of an array of separate circuit elements, including logic blocks and interconnected resources. However, it is different from MPGAs. The configuration of FPGA is executed through programming by the end-user. Figure 2.2.7 illustrates the architecture of a typical FPGA. CPLDs comprises of feature logic resources with a large number of inputs (AND planes), whereas FPGAs provide more narrow logic resources. (Keim, 2018) FPGA provides the advantage of having a higher ratio of flip-flops to logic resources as compared to CPLDs. Besides these reasons, FPGA is also considered as the most efficient device among all types of PLD, and its advantages will be discussed deeply in Chapter 2.3.

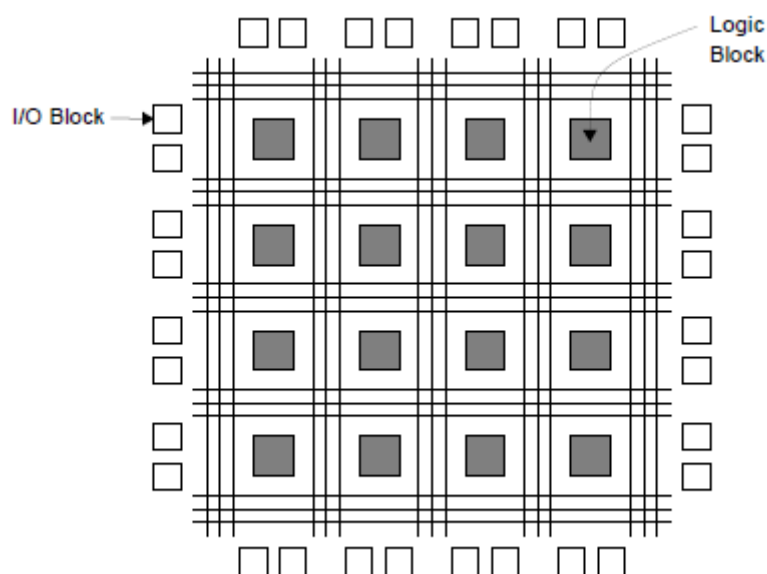


Figure 2.2.7 Architecture of FPGA (Keim, 2018)

2.3 Strengths of FPGA

Under this topic, the reason why FPGA is chosen among PLDs will be clearly stated. The strengths of FPGA makes it to be more attractive in higher speed and higher efficiency applications as compared to SPLD and CPLD.

Firstly, a candidate is said to be effective and exceptional for implementation in FPGA is the circuits that can handle a large number of logic gates and require only less number of flip-flops. This is the strong reason that is making FPGA have a large growth in the electronic market. As opposed to conventional computer chips, FPGA is fully programmable, and thus the updates, as well as the adaptations, can be carried out even after sold out to customers. FPGA allows the user to alter the design easily through re-programming, and without rewiring, even the FPGA is running. While in CPLD, it has to de-energize first, then only can re-program for the modification of the design functionality. (Center, 2017)

Next, FPGA provides long-term availability as the functionality is not from the module itself but mainly in the configuration. Therefore, it can be programmed in such a way without any modifications and adjustments on different FPGA models. (Inventions, 2017) In addition, the implementation of FPGA accelerates the development of prototypes. It is because a part of the hardware development mainly lies in the design of the IP cores. Hence, FPGA can perform time-consuming operations such as troubleshooting and commission at the same time.

Furthermore, FPGA offers the opportunity of modeling systems that are precisely tailored to the intended task, which able to work efficiently. (Inventions, 2017) The flexibility of FPGA can be clearly shown in its massively parallel data processing. FPGA is customarily used to solve complex tasks via parallelization and adaption to the application. As shown in Figure 2.2.7, the architecture of FPGA mainly consists of three parts, which include the configurable logic blocks, programmable interconnects, and programmable I/O blocks. (WatElectronics, 2019) The logic blocks are separated into smaller modules, and hence providing a significantly scalable solution through the

parallel processing of data. In short, FPGA is ideal for real-time applications and offers a significant speed advantage as compared to SPLD and CPLD. For instance, the complicated calculations can be performed in a short time.

In a nutshell, FPGA is much more capable among all the PLDs, and so it is the most suitable PLD to carry out the game developments, which consist of many different complex tasks and able to handle them at the same time. Therefore, FPGA can be more expensive as well and consume a little bit of higher idle power consumption. The uses of FPGA in other game developments will be discussed in Chapter 2.6 in detail.

2.4 Computer-Aided Design (CAD) Flow for FPGA

This topic will target the Computer-Aided Design (CAD) Flow for FPGA, as referred to as the research work of Brown, S. and Rose, J. (2002). CAD is an approach using a computer to aid in the creation, modification, and simulation. It is employed to design circuits for implementation in FPGA.

Generally, the CAD system of an FPGA would be comprised of software for tasks, such as initial design entry, logic optimizing, device fitting, system simulating, and device configuring.

Initially, the execution of design entry can be done by constructing a schematic diagram with a graphical CAD tool, which describes the design by using a text-based system in a hardware description language (HDL) such as VHDL and Verilog, or with a mix of different design entry methods. Algorithms are suggested to implement in the circuits to optimize the circuits since the initial logic entry is not always optimized.

After this, additional methods are needed for interpretation of the resulting logic equations and "fitting" them into FPGA. The "device fitter" step falls between the simulation and logic optimization, which involves three steps for the circuit implementation in FPGAs, which are mapping the basic logic gate into the logic blocks of FPGA, choosing the customized logic blocks to apply in FPGA, and wire segments allocation in FPGA for logic blocks interconnection.

Next, simulation is used to ensure the operation is running accurately, and if there are any errors detected which interrupt the process, the feedback signal is sent to the design entry step, fixing the errors. If the simulation runs successfully without error, it can be downloaded into a programming unit, which completes the configuration of an FPGA.

It is noted that the first design entry step has to be performed manually by the executor, and the rest is carried out automatically by the CAD systems. With the increased complexity, the CAD tools might require a longer time (usually more than an hour) to finish their tasks.

2.5 Application of FPGA

This topic will discuss the application of FPGA in real life. Due to the advantages of FPGA discussed in Chapter 2.3, FPGAs have gained recognition and acceptance by society, leading to rapid growth in the market due to their vast range of applications.

FPGA provides high speeds and a range of capacities, making it applicable to random logic, device controllers, filtering and encoding of communication, multiple SPLDs integration, and small to medium systems with SRAM blocks. (Touger, 2018) It is ideal for high performance, critical control applications, and can be used in digital designs.

Also, FPGA can perform different intricate designs. FPGA is usually implemented in prototyping ASIC or processor. FPGA can be reconfigured and re-programmed until the ASIC and processor design is completed and bug-free, then the final version of ASIC will be manufactured and mass-produced. For example, Intel implements FPGA in prototyping new chips due to the reason of cost-effectiveness. (Touger, 2018) After the prototyping, it can be later used for execution in gate arrays, as well as simulation of large hardware systems. All these applications can either employ only a single large FPGA or involve a quantity of FPGAs which are interconnected to each other. For instance, the hardware emulation, QuickTurn, has produced goods that comprehend many FPGAs and the required software to map and segregate circuits. (Wolff, 1990) Another interesting FPGA application is handling FPGA as custom computing

machines. Instead of using regular CPU to compile the software for execution, FPGA comprises the configurable parts to "execute" software.

FPGA is not left behind as it is gaining prominence in deep neural networks (DNN) that are implied for AI. (Touger, 2018) Operating DNN interference models consume significant processing power. Initially, Graphics Processing Units (GPUs) are often applied for accelerating the interference processing. Nevertheless, in some cases, FPGA is more preferred and outperforms GPU in interpreting massive data for machine learning.

The performance of FPGA depends on how CAD programs map the designed circuits into the chip. During the designs mapping into CPLD, the design pieces will map instinctively to the SPLD type blocks. Nevertheless, the designs that mapped with FPGA are split into block-sized logic pieces and assigned across the FPGA. Each of the logic function blocks is interconnected with each other, and the interconnection may associate various delays.

2.6 Game Development on FPGA

This project is mainly focused on game development, taking a real-world problem from the stage of hardware design. This research reviewed on two game designs, which are the Dice game (Toonsi S. Corporation, 2017) and the Tetris game (Liu, K. Corporation, 2012). Both of them differ in terms of the design complexity; Dice game is a simple game design with a lower complexity while the Tetris game is with higher design complexity. The idea and relevant information gathered from the literature review, such as improvements, design modules, system implementation flow, will be later applied in this project, which aimed to develop a Ping-pong Game on FPGA. The pros and cons of FPGA can be validated and studied during the game designing process with the implementation hardware description language and also the porting to real hardware, FPGA.

2.6.1 Dice Game Development on FPGA

The research work on Dice Game development with the implementation of FPGA is simple game design. The reason for discussing the dice game is because the basic design flow of the game and working principles of FPGA can

be learned and easy to understand. Also, the principles can later be applied to Ping-pong game development, which is explained in Chapter 3.

In this project, a complex FPGA board, Altera DE2-115 from Terasic, is implemented. The board composes a set of Cyclone-IV FPGA, SRAM, SDRAM, Flash memories, seven-segment displays, Light Emitting Diodes (LEDs), switches, pushbuttons, and LCD screen. (Altera Corporation, 2006) For projects that involve processor and simple I/O interfaces, the Altera Nios II processor can be instantiated and also the implementation of standard embedded interface drivers like Ethernet, SPI, RS-232, USB, I2C, and so on. (Altera Corporation, 2006) In addition, for a project that involves video or audio signals, the board provides basic connectors that allow graphical display and sound audition. All of these peripherals allows for developing several types of the game based on the DE2-115 board.

As mentioned in Chapter 1, VHDL is one of the hardware description languages for demonstrating digital electronic systems. It can be adopted for many applications such as creation, simulation, and hardware description. In order to synthesize the VHDL, there are various CAD tools available for the implementation of FPGA as well as ASIC. (Kleitz, 2006) A VHDL code can be distributed into two parts, which are an entity description that declares all I/O ports and their types, and architecture that declare the functionality along with all functions and operations.

To design a dice game for single player, the game flow was coded in VHDL according to the flow chart attached in Figure 2.6.1. After the system is energized and initialized, the pushbuttons, *Game1*, *Game2*, and *Game3* should be pressed to execute the game. Once the *Game1* is triggered, new numbers are generated randomly for the computer and user. The new dice numbers will be displayed on the two different seven-segment displays, which are driven by the *Computer_Score* and *User_Score* ports. Furthermore, the circuit has to decide whether the computer or user has a larger dice number to determine the winner of the game. The score for the winner (who gets a larger dice number) is incremented to one, and the score for the loser (who gets a smaller dice number) remains at zero for each round of the game. If it is a drawn game, both user and

computer's scores remain no change. Another two seven-segment displays are implemented to display the results, which are directed by the ports of *User_Total* and *Computer_Total*. This process flow has three iterations, indicating that the game has three rounds. After three rounds, the system will indicate the winner of the games by comparing the total scores earned by both the user and the computer. If the user earned a higher total score than the computer, the green LED is turned, indicating the overall game winner is the user, while the red LED is triggered if the overall winner is the computer, which has a higher total score than the user. (Toonsi S. Corporation, 2017)

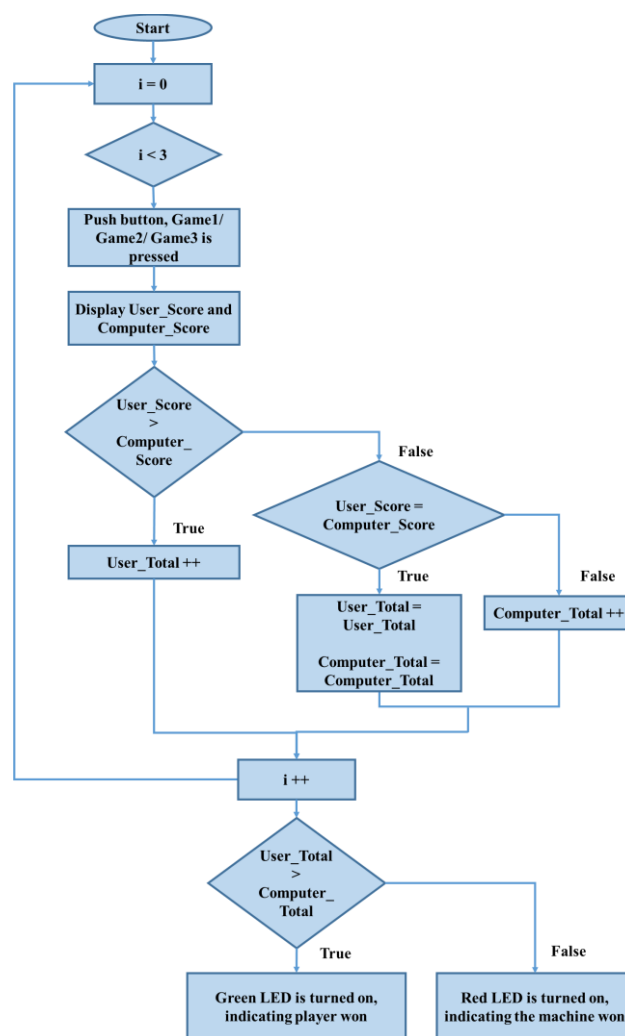


Figure 2.6.1 Flow chart of VHDL design (Toonsi S. Corporation, 2017)

The system VHDL is coded, and the entity ports description is labeled as Figure 2.6.2. It portrays that the entity "DiceGame" consists of two control ports, namely, Reset and Clk. 'Reset' is implemented to initialize the system after the system is powered on. Its control port in the system has the highest priority,

which makes it asynchronous and cannot be triggered with other commands at the same time. While 'Clk' has the second-highest priority control port, which takes over the system once 'Reset' is deserted. (Toonsi S. Corporation, 2017) The system functions when the 'Clk' is set to 1, where is the rising edge of the clock.

```
Entity DiceGame is
port(
Reset, Game1, Game2, Game3, Clk : in std_logic;
Rled, Gled: out std_logic;
User_Score: out std_logic_vector(6 downto 0);
Computer_Score: out std_logic_vector(6 downto 0);
Computer_Total: out std_logic_vector(6 downto 0);
User_Total: out std_logic_vector(6 downto 0)
);
end Dicegame;
```

Figure 2.6.2 Flow chart of VHDL design (Toonsi S. Corporation, 2017)

The system implementation is summarized in Figure 2.6.3. In this project, Altera-Quartus is used as the CAD tool for the implementation of VHDL on the DE2-115 Board. Once the dice game design was coded, the next step is creating a new project in Quartus II and add the dice game VHDL description into the new project. Before compilation and verification, the user has to select the model of the device which suits the real FPGA board used. Next, syntax verification is required to remove syntax errors, ensuring that the project can later be compiled successfully. After checking the syntax, the compilation has to be done to generate the circuit netlist. Then through the Quartus-Pin Planner, pins assignment is done by assigning the circuit ports to the physically FPGA pins on the DE2-115 board according to their respective functions as taking the DE2-115 hardware design manual as reference. Hence, the circuit is computed, and a file of binary configuration is created for the Cyclone-IV FPGA. Lastly, the final step is to configure the Cyclone-IV FPGA by completing the setup of USB-Blaster and the chosen binary configuration file. (Altera Corporation, 2006)

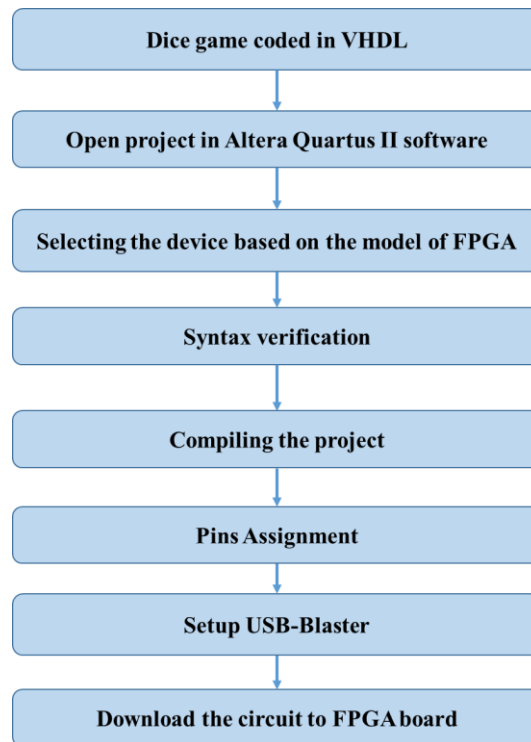


Figure 2.6.3 System implementation flow

Figure 2.6.4 indicates the user wins, while Figure 2.6.5 illustrates the computer wins. Both of the test trails proof the functionality of the developed system. It is recommended that the performance of the system can be enhanced by improving the random number generator. (Toonsi S. Corporation, 2017) Nevertheless, this approach may increase the power consumption, computational complexity, and the designation effort.

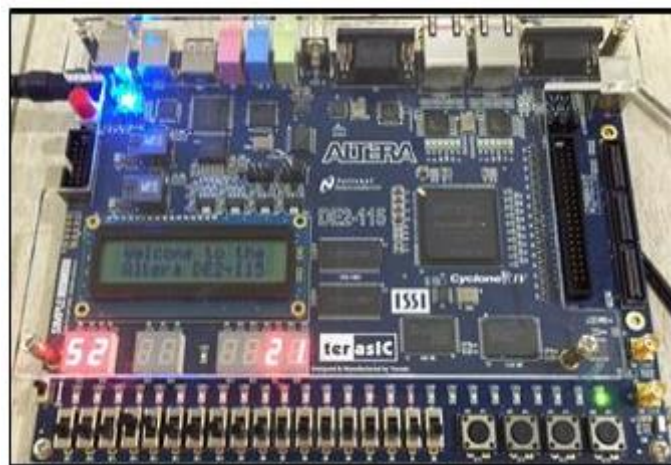


Figure 2.6.4 The display on the DE2-115 board when the user wins (Toonsi S. Corporation, 2017)

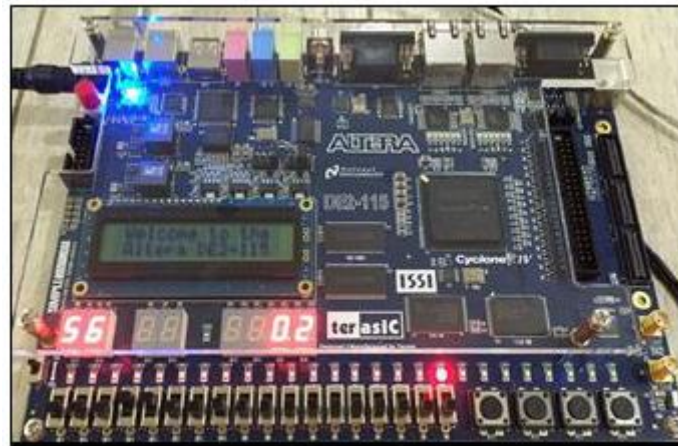


Figure 2.6.5 The display on the DE2-115 board when the computer wins
(Toonsi S. Corporation, 2017)

2.6.2 Tetris Game Development on FPGA

After discussing a simple game design (Dice Game), a more complex design of the game has to be deliberated in order to have a better understanding and gain experience in the creation of games. Tetris Game is chosen to be represented the higher complexity of the game since the design process involves more complicated steps, and the steps will be briefly highlighted under this topic.

The Tetris game is mainly built up by logic control. It is researching mainly on the development of a Tetris Game based on FPGA with the implementation of VHDL. The setting of the game allows the players to move or rotate blocks by controlling the PS/2 interface keyboard, and the game is displayed in a monitor through the VGA module.

There are several functions of the game, which are control of the movement, blocks rotation, random up-coming blocks generation, row elimination, scores accumulation, and speed acceleration. Besides that, it also composes of two modes which differ in the amount of blocks type, namely "normal mode" (seven types) and "expert mode" (11 types). The achievement of transplanting the Tetris game provides a fundamental template for the design of similar visual control systems based on FPGA. (Liu, K. Corporation, 2012)

The game consists of three core parts, as shown in the block diagram shown in Figure 2.6.6 clearly explain the related hardware and the corresponding function of these three main parts.

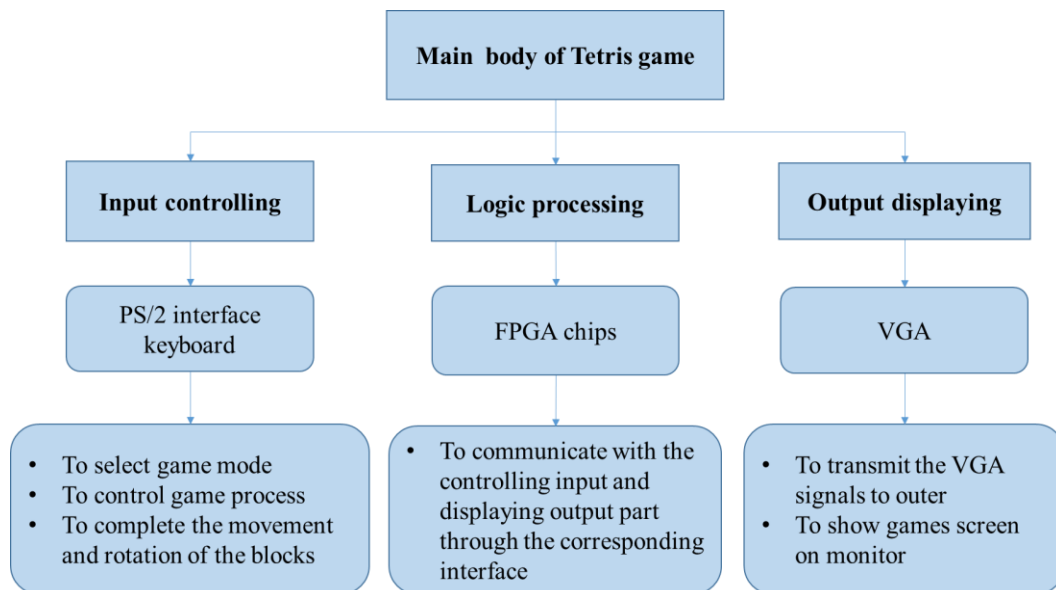


Figure 2.6.6 Categorization of the main body of the Tetris game

There are seven modules involved in the game design, where the relationship between each module structure is as shown in Figure 2.6.7. The VGA control is shared by the graphic and text display module via the multiplexing units. Also, the data exchange between the game control modules together with the graphic and text display module is completed via the storage units. (Liu, K. Corporation, 2012)

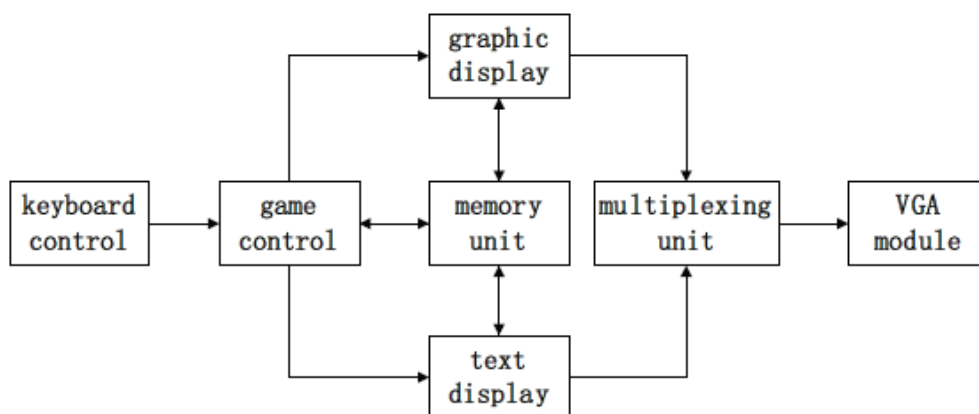


Figure 2.6.7 Structure of the program module (Liu, K. Corporation, 2012)

The main functions of the core program modules are illustrated and summarized in Table 2.6.1. The pre-set condition and description are also clearly stated.

Table 2.6.1 Function of the core program modules

Program module	Description														
Keyboard control module	<ol style="list-style-type: none"> 1. It is recognized by the identification of the keyboard and key-value filter. 2. One-way communication from the PS/2 keyboard to FPGA is done via the keyboard identification part. 3. Pre-set the input keys, and the module will only respond to the respective key. 4. The key inputs are pre-set as below: <p style="text-align: center;">Table 2.6.2 Game key function table</p> <table border="1"> <thead> <tr> <th>Keys</th><th>Function description</th></tr> </thead> <tbody> <tr> <td>F1</td><td>Stop game</td></tr> <tr> <td>F2</td><td>Normal game mode</td></tr> <tr> <td>F3</td><td>Expert game mode</td></tr> <tr> <td>A</td><td>Move blocks left</td></tr> <tr> <td>D</td><td>Move blocks right</td></tr> <tr> <td>space</td><td>Rotate blocks</td></tr> </tbody> </table> 	Keys	Function description	F1	Stop game	F2	Normal game mode	F3	Expert game mode	A	Move blocks left	D	Move blocks right	space	Rotate blocks
Keys	Function description														
F1	Stop game														
F2	Normal game mode														
F3	Expert game mode														
A	Move blocks left														
D	Move blocks right														
space	Rotate blocks														
Game control module	<ol style="list-style-type: none"> 1. The process of the game is illustrated through the "State Machine," as presented in Figure 2.6.8. 2. There are 15 states in the game, as described in Table 2.6.3. 3. The blocks have consisted of four sub-blocks for both of the game modes. All operations of blocks can be done by altering the sub-blocks coordinate. 4. The game state transition diagram is pre-set as below: <p style="text-align: center;"> </p> <p style="text-align: center;">Figure 2.6.8 Transition diagram of the game state. (Liu, K. Corporation, 2012)</p>														

5. The game state machine is described as below:

Table 2.6.3 Game state machine description (Toonsi S. Corporation, 2017)

State No.	State	State description
0000	stop	Press F1 or game over
0001	normal	Press F2 to enter normal mode
0010	master	Press F3 to enter expert mode
0011	wait	Wait for cntClk signal; then turn to "test_down" state or read command form keyboard
0101	test_down	Try to make the current active blocks fall to the next line; if there have obstacles, turn to "test_stop", or turn to "step_down".
0110	step_down	The current active blocks drop a line.
0111	test_right	Try to make the current activation block move right a column; if there are barriers, jump to "wait", or jump to "step_right".
1000	step_right	The current active block moves right a column.
1001	test_left	Try to make the current activation block move left a column; if there are barriers , jump to "wait", or jump to "step_left"
1010	step_left	The current active block moves left a column.
1011	test_rotate	Try to make the current activation block rotate as presets; if there are barriers , jump to "wait", or jump to "rotate"
1100	step_rotate	The current active block rotates as presets.
1101	test_stop	Determine whether y-coordinate is 1, if so jump to "stop", or jump to "del_rows".
1110	del_rows	Determine the rows eliminating conditions, if it meets, then execute rows elimination processing.

Graphics and text display module

1. The module allows the communication of text information and pixel graphics with dual-port block memory (BRAM).
2. The display signal is created by writing the respective memory address and feedback to the game control module by obtaining the respective address of the memory location.
3. It is mainly targeted on the display of blocks graphics and text character, as shown in Figure 2.6.9 and Figure 2.6.10, respectively.

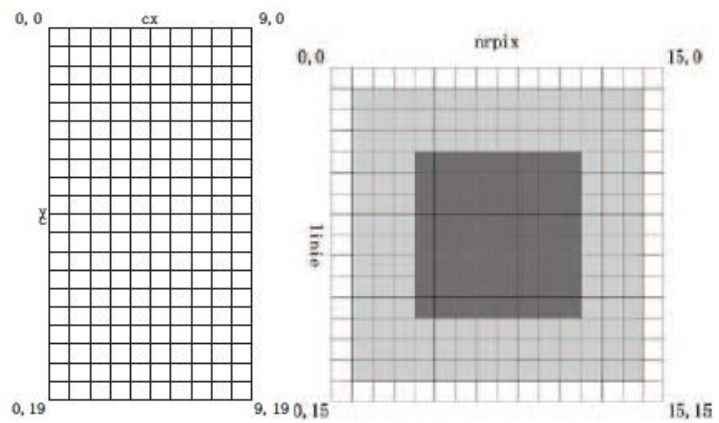


Figure 2.6.9 Graphic display method (Toonsi S. Corporation, 2017)

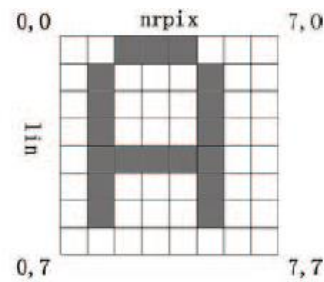


Figure 2.6.10 Character display method (Toonsi S. Corporation, 2017)

VGA module

1. VGA module is applied for coordinate and pixel settings information translation, which is obtained from the text display module and the graphic display module, then depicting the image information correctly by a monitor.
2. This module also helps in dividing the area of the game and feature on the screen.
3. HC represents the synchronizing signal of VGA Line, while VC represents the VGA Field synchronizing signal.

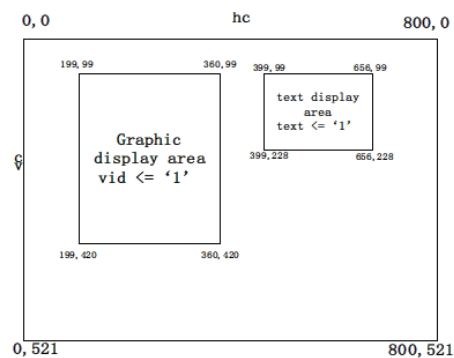


Figure 2.6.11 Dividing of the area of game and feature. (Toonsi S. Corporation, 2017)

2.7 Summary

This chapter began with the evolution of PLDs as well as an introduction to PLDs in terms of their basic technology with the programmability it provides and a short illustration of the PLD architectures. The first generation of PLD inventions such as PROM, PLA, PAL, and GAL can be grouped into the small category, namely SPLD. With the advancements of technology, the newly developed modern systems are designed with increased complexity due to the increment in market demand. In order to encounter this complexity, while maintaining the efficiency of the system in terms of power consumption and resources, CPLD and FPGA are then invented. The discussion in this chapter also highlighted the important issue and implementation flow of CAD tools for FPGA.

Furthermore, this chapter also pinpoints the application of FPGA, especially the implementation of game development. In order to learn the game design flow and basic working principles of FPGA, a classical Dice game is discussed. The dice game is a simple game development, which as a reference for further design enhancement as well as a more sophisticated game design. In order to get to know the higher complexity of the FPGA application, another game design is studied, which is the Tetris Game.

Both the Dice game and Tetris game are designed with the implementation of VHDL. The system implementation flow for both game design is similar and is further applied in the Ping-pong game. Besides the rules of the games, both of them differ in terms of their design modules such as controlling method and displaying method. In the Dice game design, the game is controlled by the push buttons of FPGA, and the game is displayed on the seven segments display of FPGA. While in the Tetris game, the game control is improved with the use of a PS/2 keyboard, and the game display is upgraded to the implementation of the VGA monitor. The enhancements from the researched works are encouraged to imply in this project by upgrading the use of internal components of FPGA to the implementation of external hardware such as VGA. The design approaches will be covered deeply in Chapter 3.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

The proposed project is to design a game on FPGA with the implementation of hardware description language. The flowchart, as shown in Figure 3.1.1, illustrates the design flow of the project.

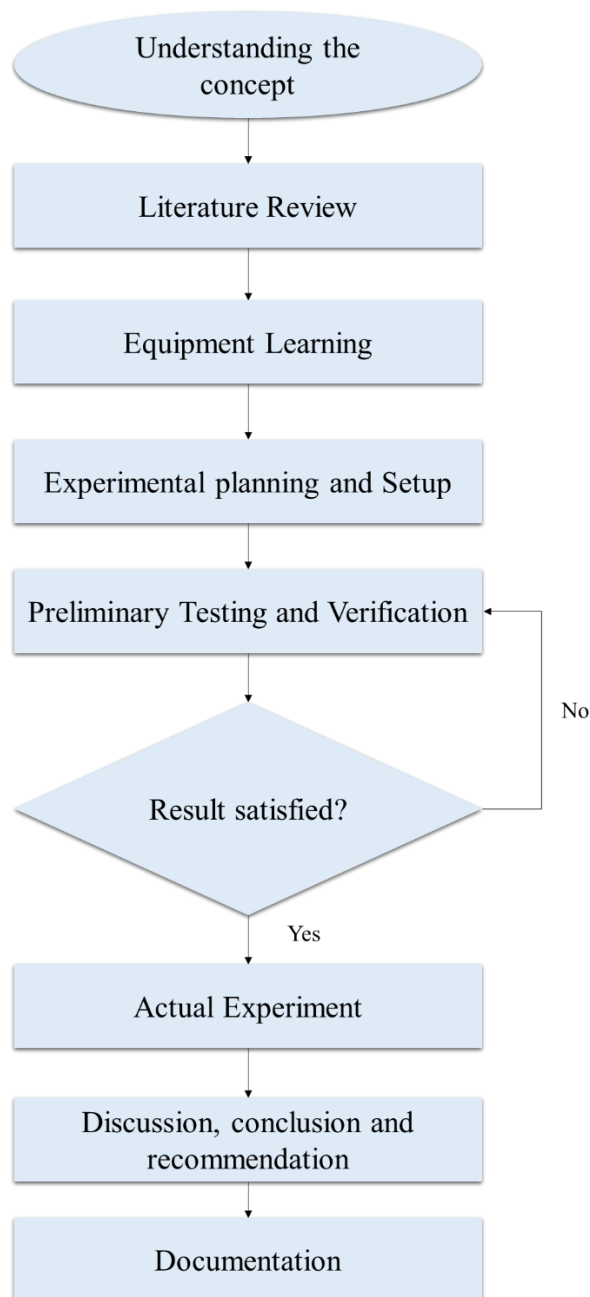


Figure 3.1.1 Design flow of the project

3.2 Project Planning

The scope and schedule of the project have to be planned and organized carefully in order to ensure that the project can be completed on time, and most crucially, that the final project outcome will meet the requirements and fulfill the objectives of the project.

3.2.1 Scope Planning

A task-based approach will be implied for project planning in order to pinpoint the required tasks. The project was separated into small tasks and activities which subscribe to the fulfillment of the project requirements and objectives, with an estimated duration being appointed to each of them, as illustrated in Table 3.2.1.

Table 3.2.1 Project tasks with the estimated duration

Phase	Activities	Description	Duration (Days)
1	A	Problem Identification	7
	B	Project Planning	7
	C	Literature review	35
	D	Equipment Learning	28
	E	Experimental Planning and Setup	7
	F	First stage of Product development & Testing	28
	G	Report Writing & Presentation	28
2	H	Project Planning	7
	I	Preliminary Testing and Verification	21
	J	Actual Experiment: Game Development	49
	K	Results discussion, conclusion and recommendation	14
	L	Report Writing & Presentation	28

3.2.2 Schedule Planning

Next, the project flow is presented in a Gantt chart, as shown in Figure 3.2.1.

Phase	No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26	W27	W28
1	M1	Problem formulation																												
	M2	Project planning																												
	M3	Literature review																												
	M4	Equipment Learning																												
	M5	Experiment Planning and Setup																												
	M6	First stage of product development & Testing																												
	M7	Report writing & presentation																												
2	M1	Project planning																												
	M2	Preliminary Testing and Verification																												
		-Testing of CAD Tool																												
		-Testing of FPGA board																												
		-Testing of VGA monitor																												
	M3	Actual Experiment: Game Development																												
		-System Overall Design																												
		-Design of input key module																												
		-Design of VGA module																												
		-Design of game control module																												
M4	Results discussion, conclusion and recommendation																													
M5	Report Writing & Presentation																													

Figure 3.2.1 Gantt chart of the project

3.3 Equipment Learning

Equipment learning is required before the specific design stage of the game, and it is crucial to ensure the design flow to be carried out smoothly.

3.3.1 Block Diagram of FPGA DE2-115

The block diagram of DE2-115 is illustrated in Figure 3.3.1 and as a reference for the further designation. The features of DE2-115 can be found from APPENDIX E.

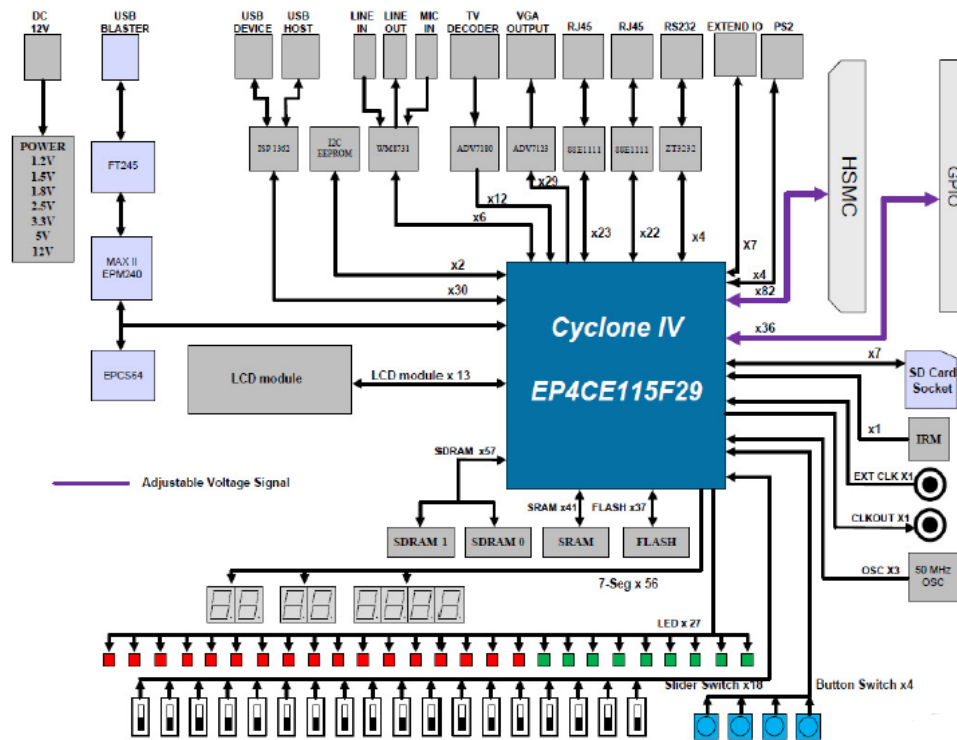


Figure 3.3.1 Block diagram of DE2-115 (Altera Corporation, 2010)

3.3.2 USB-Blaster

The Control Panel of DE2-115 relied on the Nios II SOPC system, which instantiated in the Cyclone IV E DE2-115 FPGA, with the on-chip memory being used for the software to operate. (Altera Corporation, 2010) Figure 3.3.2 shows the architecture of the Control Panel. Nios II Processor integrated into the FPGA chip supervised each I/O device. The role of the USB Blaster link is to handle the communication between the host computer and the DE2-115 board; hence the installation of Altera USB Blaster driver software is necessary. (Altera Corporation, 2010) The Nios II analyses the signal sent from the PC and actuates the corresponding actions. The USB cable provided by the host computer should be

connected to the USB Blaster connector of the DE2-115 board to complete the installation. (Altera Corporation, 2010)

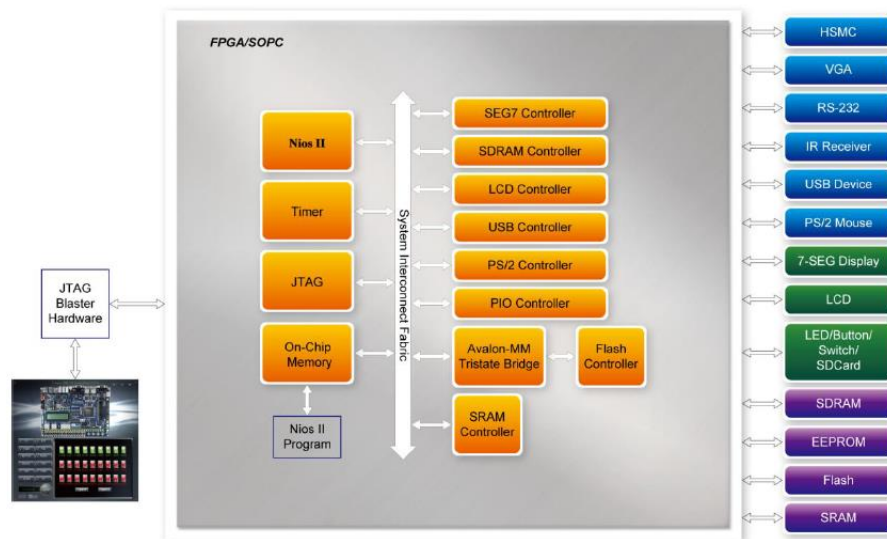


Figure 3.3.2 Block Diagram of the DE2-115 Control Panel (Altera Corporation, 2010)

3.3.3 Quartus II software

Figure 3.3.3 depicts the interface of the Quartus II software. The design entry of Quartus II can be done in different file types such as VHDL, Verilog HDL, Block Design File, Qsys System File, State Machine File and so on. Thus, it can be implemented in many different tasks, which include creation, simulation, compilation, as well as configuration.

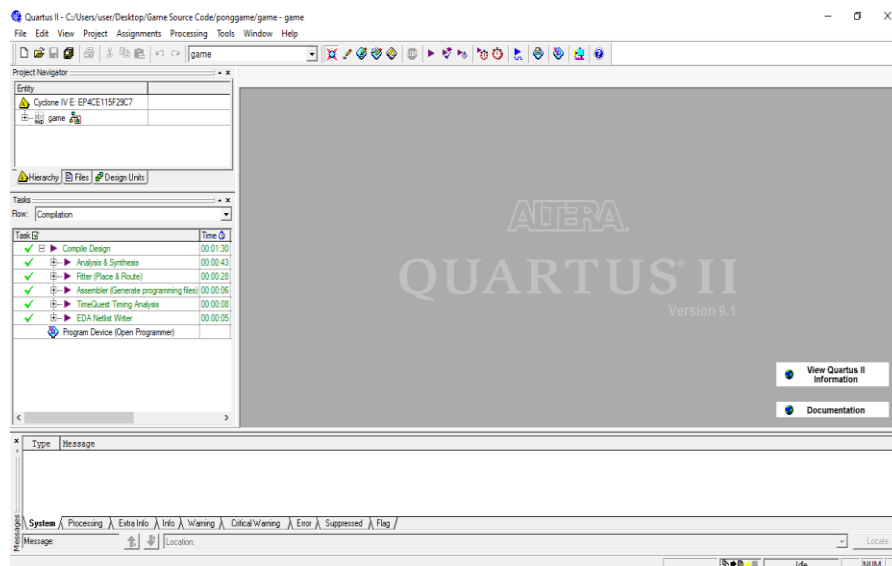


Figure 3.3.3 Interface of Quartus II

3.4 Experimental Planning and Setup

Equipment required:

Altera's DE2-115 FPGA board

Host Computer

VGA monitor

VGA cable

USB Blaster

Power Cable for DE2-115

Figure 3.4.1 shows the experimental setup in this project, which is the same as the planned experimental setup provided in Figure 1.3.1. The configuration data will be loaded from the host computer to the FPGA device and display the result on the monitor.

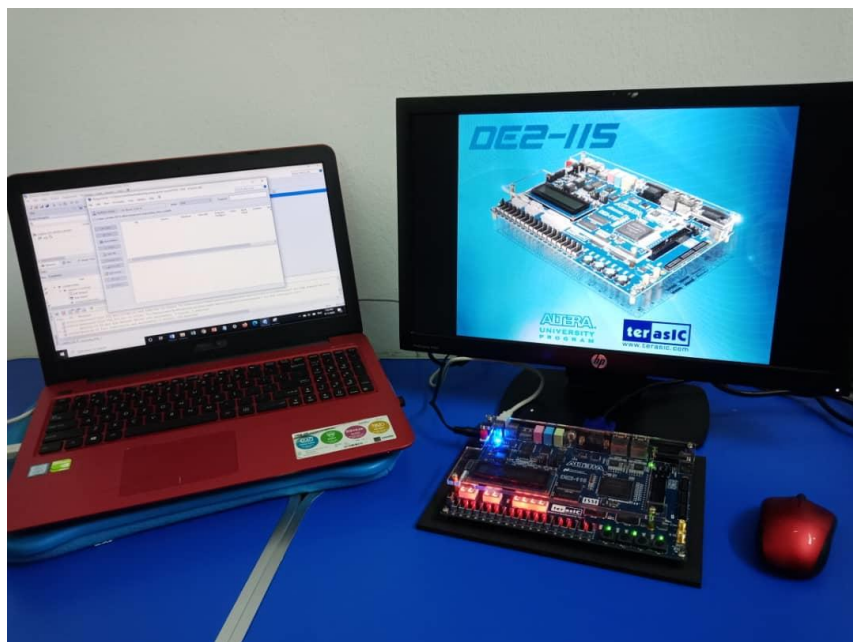


Figure 3.4.1 Experimental Setup

3.5 Preliminary Testing and Verification

General testing has to be done before carrying out a specific task to verify whether the equipment used in this project works. Three simple tests, including testing of CAD Tool, FPGA board, and VGA monitor, were carried out to determine if they can function well in order for them to be implemented in the game design.

3.5.1 Testing of CAD Tool

As discussed in Section 2.4, Computer-Aided Design (CAD) Flow for FPGA, the necessary knowledge and theory of CAD was studied. CAD is known as an approach using a computer to aid in the creation, modification, and simulation. Under this section, the assignment will firmly be focused on the study of performing the CAD tool shown in Figure 3.5.1 to complete the design. The following CAD flow for FPGA can be applied in every task using the Quartus II software. The main objective of this testing is to ensure that there is no missing steps while developing the design.

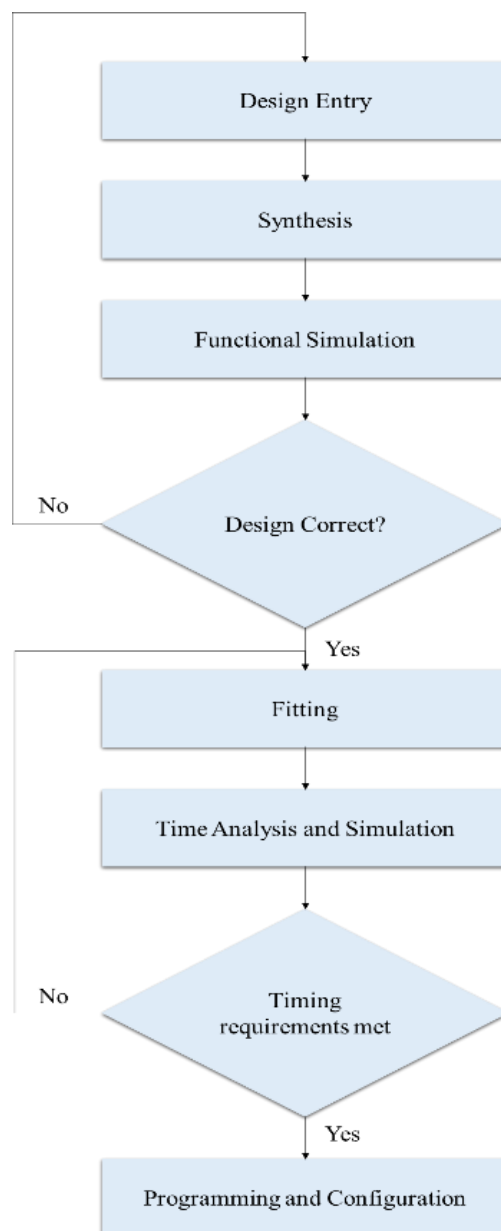


Figure 3.5.1 CAD Flow for FPGA

For this testing, a simple circuit for two-way light control, as shown in Figure 3.5.2, was designed by following the step by step stated in the CAD flow. Quartus II works on one project at a time, storing all the data for that project in a single directory. Therefore, in order to start a project, the initial step is to create a new directory to hold its files. Next, device assignments have to be done to specify the type of device in which the configured circuit will be applied. In this project, the device name was specified as EP4CE115F29C7, as presented in Figure 3.5.3.

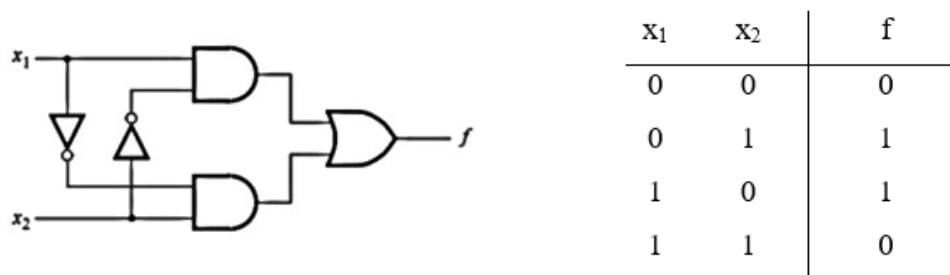


Figure 3.5.2 Logic Circuit Design

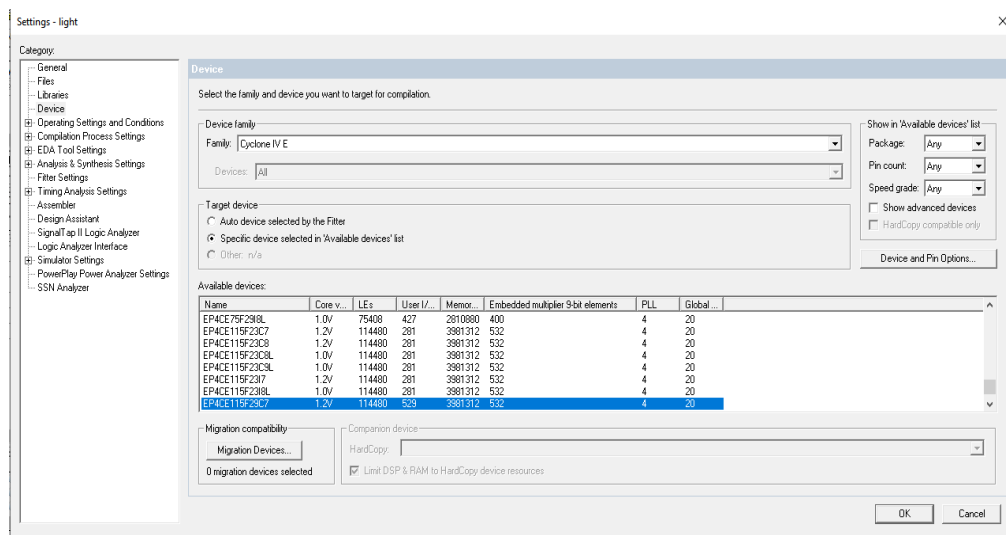


Figure 3.5.3 Device Assignment

The proposed circuit was described in Verilog code, as shown in Figure 3.5.4. One point that needs to be highlighted is that the module name must be matched with the name specified in the first step when the project was created. Otherwise, the compilation will stop, and the error occurs, showing that *"Error: top-level design entity "file_name" is undefined"*.

```

1 module light (x1, x2, f);
2   input x1, x2;
3   output f;
4   assign f = (x1 & ~x2) | (~x1 & x2);
5 endmodule

```

Figure 3.5.4 Design Entry using Verilog Code

The Verilog code in Figure 3.5.4 will be processed by Quartus II tools, including the coding analysis, circuit synthesizing, and generate an implementation of it according to the corresponded target chip. These tools are controlled under the application program called the "*Compiler*". The compilation will move through various stages and verifies the syntax. Figure 3.5.5 shows the process of compiling the project; this process might take the time of a few minutes.

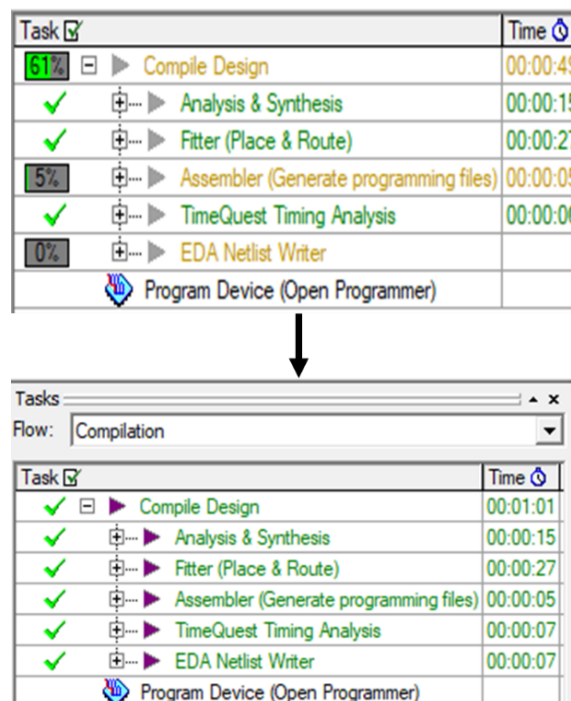


Figure 3.5.5 Process of compiling the project

Once the compilation is completed, a window will pop up and shows the result of the compilation, such as successful or unsuccessful. The example can be seen from Figure 3.5.6.

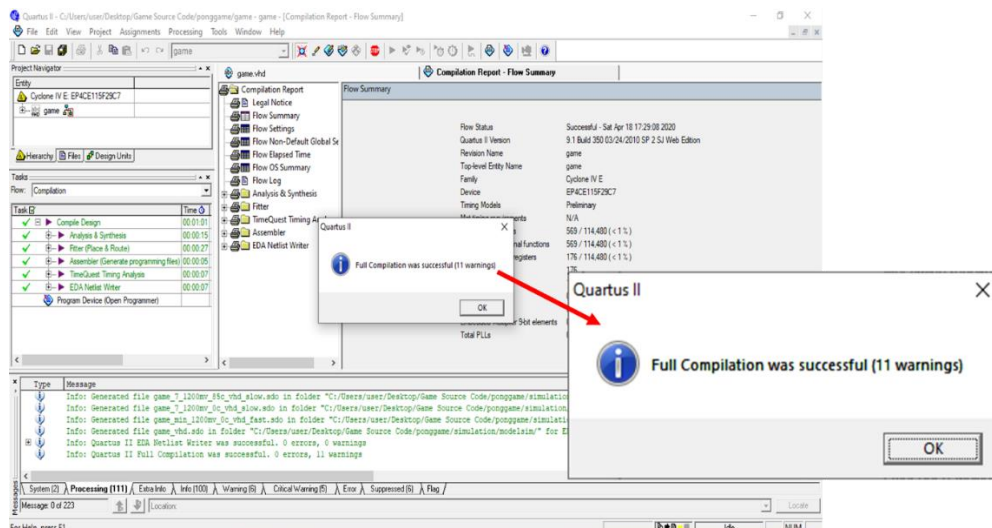


Figure 3.5.6 The result of compilation

However, Figure 3.5.6 only displays the final result. If there are any syntax errors, it may take a very long time to find out the errors or bugs. Hence, if looking for detail information on each line of code, the message window will take the responsibility to explain in detail. For example, the system warned that *"the timing characteristic of the device EP4CE115F29C7 are preliminary"*, as shown in Figure 3.5.7; while in the case of errors, the error type will be stated in the message box as shown in Figure 3.5.8.

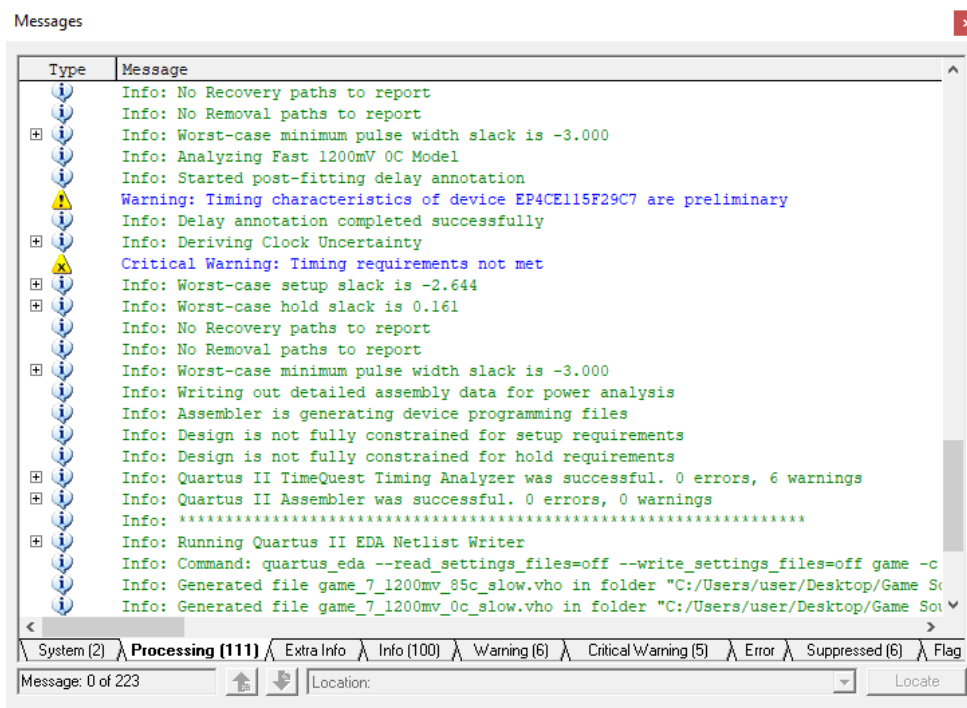


Figure 3.5.7 Message window of Quartus II

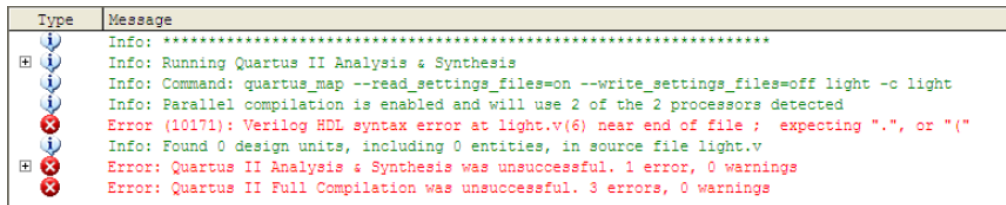


Figure 3.5.8 Message box showing "Error"

After the successful compilation, the pin assignment is required to assign the function to the targeted pins. It is the process of reflecting and connecting the software coding to the physical hardware. The Quartus II user was free to select any pins on the FPGA to serve as inputs and outputs. Nevertheless, the DE2-115 board has hardwired connections between the FPGA pins and the other components on the board. Pin assignments were made by using the Assignment Editor, as shown in Figure 3.5.9. The inputs of the designed circuit, x1, and x2, were represented by the two toggle switches on the development board, which are SW0 and SW1, respectively. As referred to the "DE2-115 User Manual" (APPENDIX D), the two inputs were assigned to the FPGA pins AB28 and AC28, respectively. And lastly, the output f was connected to the green light-emitting diode with the label of LEDG0, which was hardwired to the FPGA pin E21.

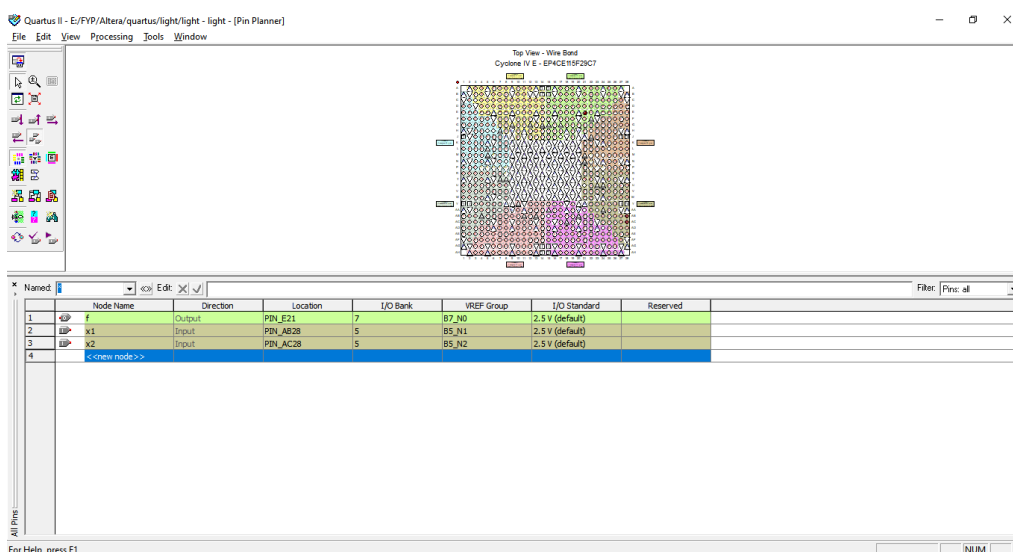


Figure 3.5.9 Pin Assignment (Testing of CAD Tool)

It is vital to verify the correctness of the design before implementing the designed circuit in the FPGA chip. Modelsim software was implemented to

simulate and verify the behavior of a designed circuit. Figure 3.5.10 shows the interface of Modelsim, and the compilation result will display in the transcript, which is placed at the bottom of the figure.

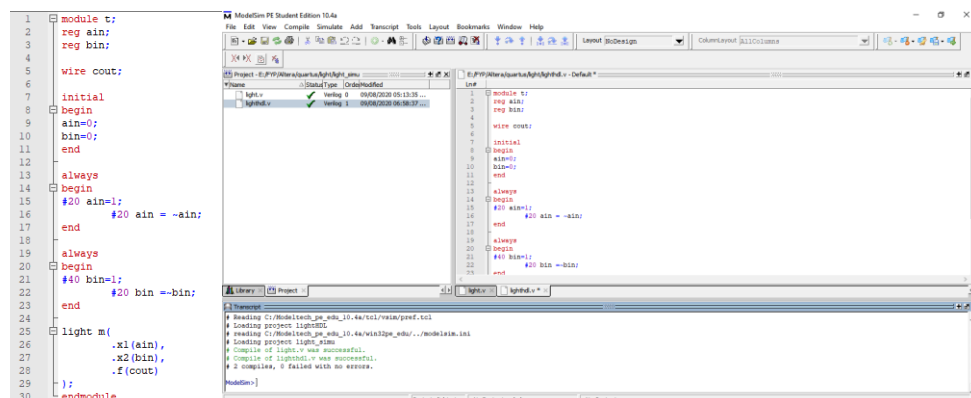


Figure 3.5.10 Interface of Modelsim

The FPGA must be programmed and configured in order to execute the designed circuit. The configuration file was created by the Quartus II Compiler's Assembler module. The configuration of Altera's DE2-115 can be done through JTAG mode. JTAG stands for Joint Test Action Group, which introduces a simple way for testing the digital circuits and loading the configured data into the development, which became an IEEE standard. The configuration data was transferred from the host computer (which runs the Quartus II software) to the board through a cable that connects a USB port on the host computer to the leftmost USB connector on the board. Hence, it is necessary to have the USB-Blaster driver installed to complete the connection. In the JTAG mode, the configuration data was loaded directly into the FPGA device. If the FPGA was configured in this manner, it would retain its configuration as long as the power remains turned on. The configuration information is lost when the power is turned off. The result of this testing will be further discussed in Section 4.2.1.

3.5.2 Testing of FPGA board

The main intention of testing the FPGA board is to ensure all the components are functioned and can be configured, outputting a corresponded design without any delays. During the test, a simple design was coded in Verilog (.v) as shown in Figure 3.5.11, to cause LEDs on the FPGA to blink at a speed that is under the control of an input key. It gives direct visual feedback to determine whether

the development board works. The design involved a 32-bit counter, a 2-input multiplexer megafunction, and a phase-locked loop (PLL) megafunction acts as the clock source. A Block Design File (.bdf) can be created by updating the file from .v to .bdf. The corresponded "*simple counter*" in the Verilog file will be generated automatically in the Block Design File, as shown in Figure 3.5.12.

```
//It has a single clock input and a 32-bit output port
module simple_counter (
    CLOCK_50,
    counter_out
);
    input CLOCK_50 ;
    output [31:0] counter_out;
    reg [31:0] counter_out;
    always @ (posedge CLOCK_50) // on positive clock edge
begin
    counter_out <= #1 counter_out + 1; // increment counter
end
endmodule // end of module counter
```

Figure 3.5.11 Design Entry in Verilog (.v)

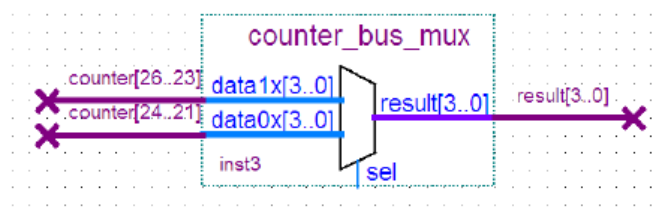


Figure 3.5.12 Design Entry in Block Design File (.bdf)

PLL Megafunction is a pre-designed module available in the LPM that the user can implement in FPGA designs. These megafunctions provided by Altera have optimized for speed, area as well as the device family. The efficiency can be boosted by employing a megafunction instead of coding the function manually. For this design, a PLL clock source was applied to drive a simple counter. A PLL used the on-board oscillator to generate a constant clock frequency acts as the input to the counter, as shown in Figure 3.5.13.

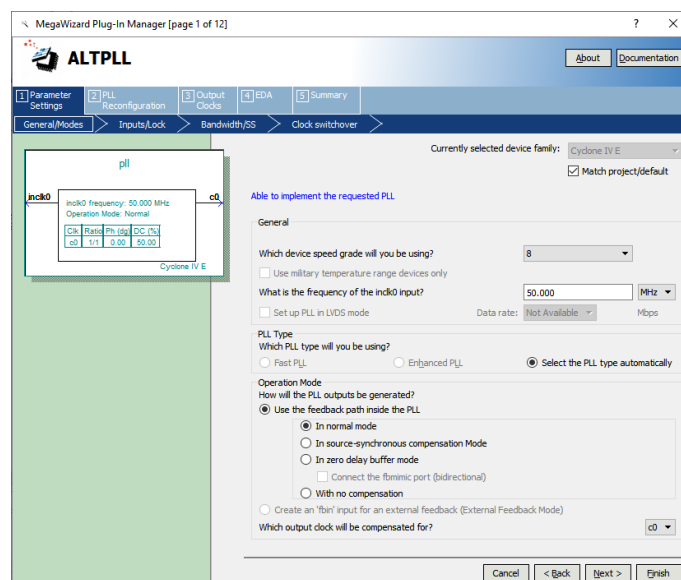


Figure 3.5.13 Creating PLL Megafunction using MegaWizard Plug-In Manager

A multiplexer was applied in this design to route the `simple_counter` output to the LED pins on the FPGA board. The MegaWizard Plug-In Manager was used to add the multiplexer, `lpm_mux`, as shown in Figure 3.5.14. The design multiplexed two alterations of the counter bus to four LEDs on the DE2-115 development board.

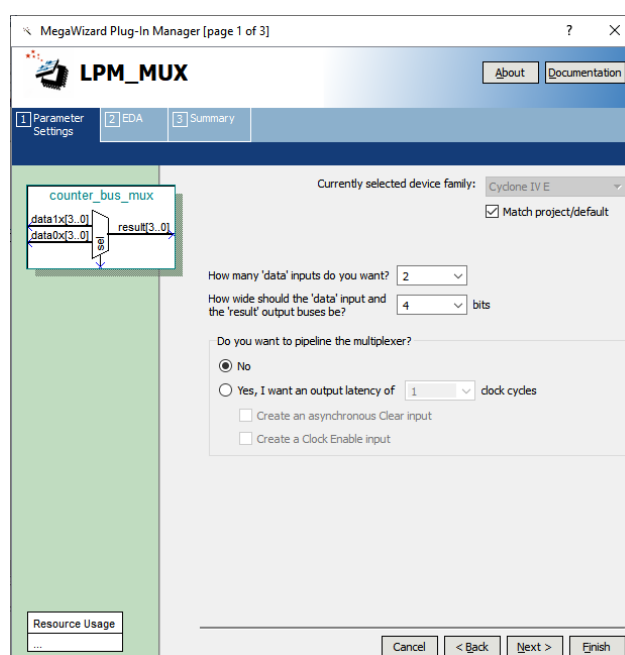


Figure 3.5.14 Creating Multiplexer using MegaWizard Plug-In Manager

All the components were connected as well as the input pins and output pins. The overall schematic diagram was drawn and presented in Figure 3.5.15.

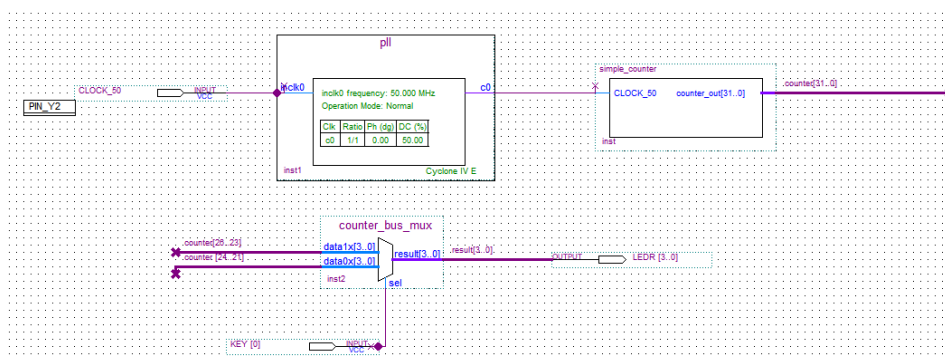


Figure 3.5.15 Overall Block Design File

The next step is conducting the pin assignment according to APPENDIX D. Figure 3.5.16 represents the pin planner of this design. The last step is to compile the project and load the configured data into the FPGA board, and the result will be further analyzed in Section 4.2.2.

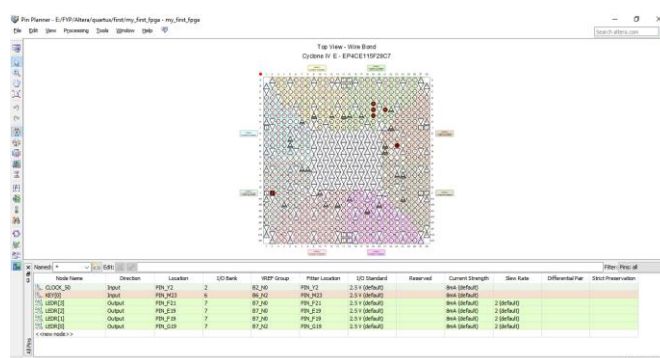


Figure 3.5.16 Pin Assignment (Testing of FPGA)

3.5.3 Testing of VGA monitor

The main aim of this test is to determine whether the VGA monitor can correctly display the design on the screen through the VGA cable. A simple test was carried out by connecting the monitor directly to the host computer through VGA cable.

3.6 Experiment on Game Development: Ping Pong Game

The game to be developed in this project is a Ping-pong game as it is one of the most common games that created using FPGA. Ping-pong game is a simple game application; however, it still has considerable space for improvement. Therefore, it is an opportunity to learn the basics and foundation from the simple game application and explore deeper when enhancing the game to a higher level

of complexity. The FPGA kit is fully utilized in developing the Ping-pong game. There are several steps to be taken to design a Ping-pong game. Figure 3.6.1 shows the design flow of the game, which includes the design of the overall system module, input key module, VGA module, and the game control module.

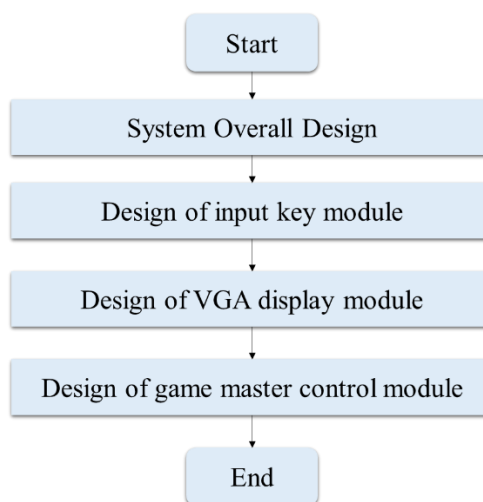


Figure 3.6.1 Design flow of a Ping-pong game

3.6.1 System Overall Design

In order to design a game on FPGA, the overall system design included the designed functions, and the defined rules have to be set preliminary before the design of the specific module.

The overall game functions and game rules were set as follows:

1. The ping pong game was designed to be played in "Double-player Mode".
2. The game interface simulated the ping-pong table with a hidden net in the middle to separate the table equally into two sides; each side consists of a paddle near the border. The upper part was assigned to Player B, whereas the bottom part was assigned to Player A. The player's scores were accumulated and displayed on the seven segment display on top of the FPGA device while the winner will be displayed in the LCD module of FPGA.

3. The ping pong ball will move in a straight line until it strikes on the left and right border of the rectangular frame or the paddle. It will be reflected and continue its motion in the opposite direction. However, when the ball is moving, if one of the paddles unable to catch the ball and bounce it, the ball will continue to move and until it touches the up or down borderline of the rectangular frame, then the opponent will earn fifteen points, and the next round began. The game-winner is who first accumulated to 90 points. The movement of the paddles was controlled by the pushbuttons of FPGA.

The Block diagram, as shown in Figure 3.6.2, indicates that the main modules of the overall system design include the VGA module, input key module, seven-segment display module, LCD module, and game control module. The game control module was the core module that used to handle the received signal from different ports and hence drive the functional modules for controlling and responding to the respective signals.

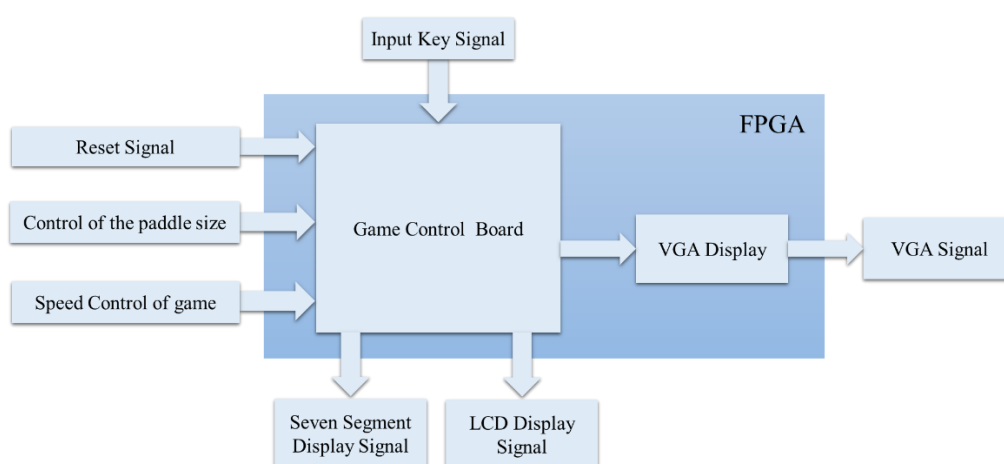


Figure 3.6.2 Overall System design of Ping-pong game

3.6.2 Design of input key module

The core function of the input key module is to receive the input scan code from the pushbuttons or slide switches of FPGA then converted to the corresponding key values. This design received four inputs from the pushbuttons and 18 inputs from the slide switches. All the input keys were summarized in Table 3.6.1. The design was coded in Verilog to assign the command to each of the respective input keys. The input keys were illustrated clearly in Figure 3.6.3.

Table 3.6.1 List of Input Keys with its corresponded command

Command	Input Key (Player A)	Input Key (Player B)
Move to Left	KEY0	KEY2
Move to Right	KEY1	KEY3
Start/Pause	SW0	
Reset	SW1	
Control the paddle size	SW4	SW2
	SW5	SW3
Speed control of the ball	SW6	
	SW7	
	SW8	
Alter the colour of paddle	SW12	SW9
	SW13	SW10
	SW14	SW11
Alter the colour of ball	SW15	
	SW16	
	SW17	
*Remarks: Other keys will be ignored by the system.		

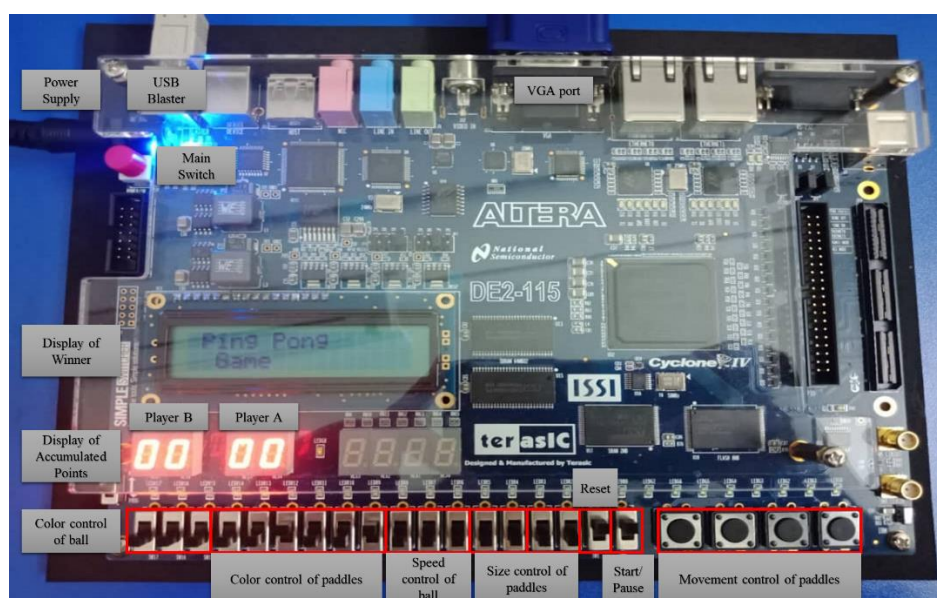


Figure 3.6.3 Input keys of the game on DE2-115

3.6.3 Design of VGA module

VGA display module mainly included the "game graphics" display function. Figure 3.6.4 demonstrates the connections between FPGA and VGA. The circuit implementation of the VGA connector are as follows:

1. Pins 13 (HS) and 14 (VS) are digital signals which are driven directly from two of the FPGA pins.

2. Pins 1, 2, and 3, which represents R, G, and B, are analog signals with a nominal value of 0.7V. Alteration of two states (0 and 1) between these three pins will output up to eight different colors.
3. The other pins (5, 6, 7, 8, and 10) are all connected to the ground.

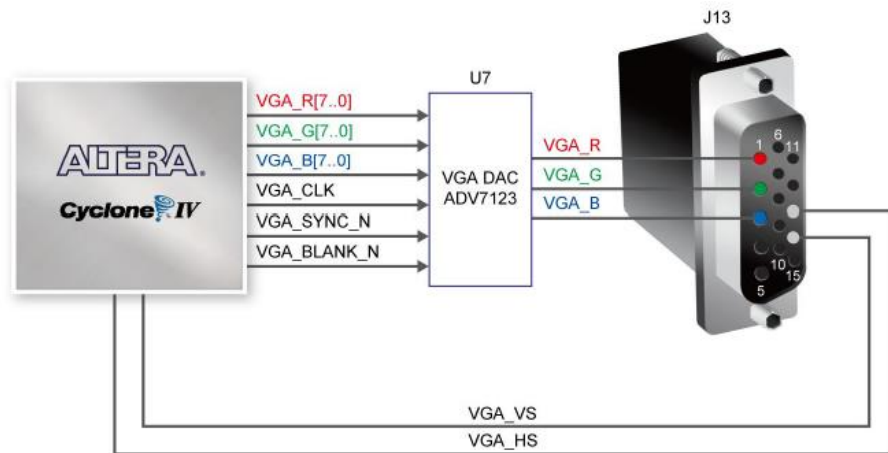


Figure 3.6.4 Connections between FPGA and VGA (Altera Corporation, 2010)

a) Resolution Setting of VGA

For this game design, the game was set to be displayed with a resolution of 640 x 480. Figure 3.6.5 portrays the VGA interface with a resolution of 640 x 480, explaining the theory of resolution, such as front porch and back porch for horizontal and vertical. This basic theory will be further used for the design in the Verilog code. Figure 3.6.6 illustrates the designed Verilog code for the VGA display. However, in order to display the selected resolution (640 x 480), one more step that has to be done is that the setting of the monitor has to be adjusted from 1600 x 900 to 640 x 480 to match the design, as depicted in Figure 3.6.7.

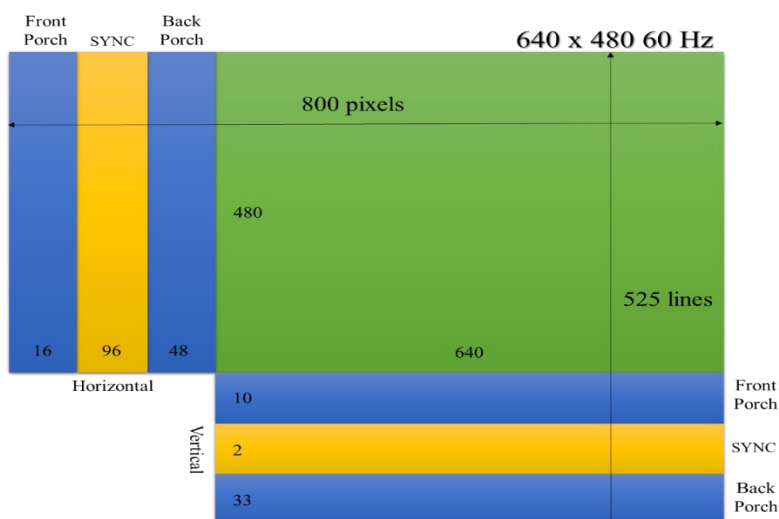


Figure 3.6.5 VGA interface with resolution 640x480

```

////////////////////////////////// VGA ////////////////////////////////////
output      oVGA_CLOCK,          // VGA Clock
output      oVGA_HS,             // VGA H_SYNC
output      oVGA_VS,             // VGA V_SYNC
output      oVGA_BLANK_N,        // VGA BLANK
output      oVGA_SYNC_N,        // VGA SYNC
output [7:0] oVGA_R,             // VGA Red[9:0]
output [7:0] oVGA_G,             // VGA Green[9:0]
output [7:0] oVGA_B,             // VGA Blue[9:0]
output [20:0] change,
output [1:0] ostate
);
// Horizontal Parameter
parameter H_FRONT = 16;
parameter H_SYNC  = 96;
parameter H_BACK  = 48;
parameter H_ACT   = 640;
parameter H_BLANK = H_FRONT + H_SYNC + H_BACK;
parameter H_TOTAL = H_FRONT + H_SYNC + H_BACK + H_ACT;

// Vertical Parameter
parameter V_FRONT = 10;
parameter V_SYNC  = 2;
parameter V_BACK  = 33;
parameter V_ACT   = 480;
parameter V_BLANK = V_FRONT + V_SYNC + V_BACK;
parameter V_TOTAL = V_FRONT + V_SYNC + V_BACK + V_ACT;

```

Figure 3.6.6 Code fragment to design the VGA display (640x480)

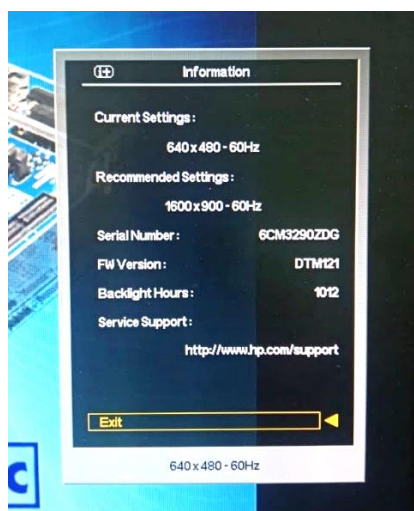


Figure 3.6.7 Resolution setting of monitor (640x480)

b) Color of Paddle

Alteration of two states (0 and 1) between R, G, and B will change the color of the paddle up to eight different colors, as summarized in Table 3.6.2. The designed Verilog codes were presented in Figure 3.6.8 and Figure 3.6.9 for Player A and B, respectively.

Table 3.6.2 Truth table to show the change of paddle color

SW14	SW13	SW12	Colour of the paddle (Player A)	SW11	SW10	SW9	Colour of the paddle (Player B)
0	0	0	Black	0	0	0	Black
0	0	1	Red	0	0	1	Red
0	1	0	Green	0	1	0	Green
0	1	1	Yellow	0	1	1	Yellow
1	0	0	Blue	1	0	0	Blue
1	0	1	Purple	1	0	1	Purple
1	1	0	Turquoise	1	1	0	Turquoise
1	1	1	White	1	1	1	White

```

always@(posedge iCLK_50)//change the paddle colour (player A)
begin
    if(iSW[12])boardB_R=8'hFF;
    else boardB_R=8'h0;
    if(iSW[13])boardB_G=8'hFF;
    else boardB_G=8'h0;
    if(iSW[14])boardB_B=8'hFF;
    else boardB_B=8'h0;
end

```

Figure 3.6.8 Code fragment to change the paddle color (Player A)

```

always@(posedge iCLK_50) //change the paddle colour (player B)
begin
    if(iSW[9])boardA_R=8'hFF;
    else boardA_R=8'h0;
    if(iSW[10])boardA_G=8'hFF;
    else boardA_G=8'h0;
    if(iSW[11])boardA_B=8'hFF;
    else boardA_B=8'h0;
end

```

Figure 3.6.9 Code fragment to change the paddle color (Player B)

c) Color of Ball

In order to change the color of the ball, the same concept of altering the two states (0 and 1) between R, G, and B was applied in this design. Table 3.6.3 summarized the truth table and its corresponded color. The designed Verilog codes were presented in Figure 3.6.10, showing the function of changing the ball color by controlling the slide switches SW15, SW16, and SW17.

Table 3.6.3 Truth table to show the change of ball color

SW17	SW16	SW15	Colour of the ball
0	0	0	Black
0	0	1	Red
0	1	0	Green
0	1	1	Yellow
1	0	0	Blue
1	0	1	Purple
1	1	0	Turquoise
1	1	1	White

```

always@(posedge iCLK_50) //colour of ball
begin
    if(iSW[15]) dateR=8'hFF;
    else dateR=8'h0;
    if(iSW[16]) dateG=8'hFF;
    else dateG=8'h0;
    if(iSW[17]) dateB=8'hFF;
    else dateB=8'h0;
end

```

Figure 3.6.10 Code fragment to change the ball color

3.6.4 Design of Game Control module

According to the block diagram of the overall system design, as shown in Figure 3.6.2, the game control module received several input signals, wherefrom input key module, reset signal, and some control signals from the external circuit. The output signals included the location of ball and paddles (VGA), game scores (seven-segment display), and the winner (LCD display). The game control module was the main module block of the game system; its main functions include:

- a. The movement control of start, pause, and stop
- b. The initial location of the ball and paddles
- c. The movement control of the ball and paddles
- d. The speed control of the ball
- e. The size control of the paddle
- f. The point's accumulation and calculation
- g. The judgment of victory and defeat

a) The movement control of start, pause, and stop

Figure 3.6.11 depicts the code that assigned the command to the respective input key of FPGA to control the movement of start, pause, and stop.

```
assign RST_N      = iSW[1];
assign right_B    = iKEY[0];
assign left_B     = iKEY[1];
assign right_A    = iKEY[2];
assign left_A     = iKEY[3];
assign oLCD_ON    = 1'b1;
assign oLCD_BLON  = 1'b0;
```

Figure 3.6.11 Code fragment to assign the command to the input key

b) The initial location of the ball and paddles

Figure 3.6.12 represents the code when fixing the initial location of the ball and paddles at the start of the game. The location of the ball and paddles were determined by the coordinate of the X-axis and Y-axis. When the RESET is pressed, the ball and paddles will revert to their original position.

```
integer originalX=320;
integer originalY=240;
integer boardAposition=320;
integer boardBposition=320;

always@(negedge iCLK_50 or negedge RST_N)
begin
if(~RST_N)
begin
counter=0;
originalX=320;
originalY=240;
directX=0;
directY=0;
scoreA=0;
scoreB=0;
state=0;
end
end
```

Figure 3.6.12 Code fragment to indicate the initial location of the ball and paddles

c) The movement control of the ball and paddle

The movement of the ball and paddles were controlled through their respective coordinates. Manually control of the paddles was done by pressing the push buttons of FPGA. Figure 3.6.13 presents the code fragment that indicated the movement of the ball and paddles.

```

if(~iSW[0])
begin
if(state==0)
begin
if(counter==(speed*50000))
begin
if(directX&&directY) //Upper left corner movement
begin
originalX=originalX-2;
originalY=originalY-1;
end
else if(~directX&&directY) //Upper right corner movement
begin
originalX=originalX+2;
originalY=originalY-1;
end
else if(directX&&~directY) //Bottom left corner movement
begin
originalX=originalX-2;
originalY=originalY+1;
end
else if(~directX&&~directY) //Bottom right corner movement
begin
originalX=originalX+2;
originalY=originalY+1;
end
if(originalX==4)directX=0; //Bounce from right
else if(originalX==636)directX=1; //Bounce from left

```

Figure 3.6.13 Code fragment to indicate the movement of the ball and paddles

d) The speed control of the ball

During the speed control of the ball, the players can adjust the speed by either slowing down or increasing it in order to increase the difficulty of the game. The higher the speed of the ball, the higher the difficulty level of the game. The speed control module was controlled by the slide switches (SW6, SW7, and SW8) as presented in Table 3.6.4, while the designed code as illustrated in Figure 3.6.14.

Table 3.6.4 Speed control of the ball

SW8	SW7	SW6	Speed of the ball
0	0	0	<div> <div>Slowest</div> <div> <div></div> <div>Increasing</div> <div></div> </div> <div>Fastest</div> </div>
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

```

always@(posedge iCLK_50) //speed control of the ball
begin
    case({iSW[8],iSW[7],iSW[6]})
        0:speed=20;
        1:speed=18;
        2:speed=16;
        3:speed=15;
        4:speed=14;
        5:speed=13;
        6:speed=12;
        7:speed=10;
        default::;
    endcase
end

```

Figure 3.6.14 Code fragment to control the speed of the ball

e) The size control of the paddle

Another task to increase the difficulty of the game is to adjust the size of the paddles. The smaller the size of the paddles, the higher the difficulty level of the game. The paddle size can be controlled by changing the binary state of the slide switches, as summarized in Table 3.6.5. The code fragment of controlling the paddle size of Player A and Player B was presented in Figure 3.6.15 and Figure 3.6.16, respectively.

Table 3.6.5 Size of the paddle according to its respective binary state

SW3	SW2	Size of the paddle (Player A)	SW5	SW4	Size of the paddle (Player B)
0	0	Largest ↓ Decreasing Smallest	0	0	Largest ↓ Decreasing Smallest
0	1		0	1	
1	0		1	0	
1	1		1	1	

```

always@(posedge iCLK_50) //change of paddle size (player A)
begin
    case({iSW[3],iSW[2]})
        0:sizeA=30;
        1:sizeA=25;
        2:sizeA=20;
        3:sizeA=15;
        default;;
    endcase
end
endmodule

```

Figure 3.6.15 Code fragment to change the paddle size (Player A)

```

always@(posedge iCLK_50) //change of paddle size (player B)
begin
    case({iSW[5],iSW[4]})
        0:sizeB=30;
        1:sizeB=25;
        2:sizeB=20;
        3:sizeB=15;
        default;;
    endcase
end

```

Figure 3.6.16 Code fragment to change the paddle size (Player B)

f) The point's accumulation and calculation

The scoring system was done by applying the method of point's accumulation with the implementation of BCD code in the total point's calculation and output it on the seven segment display. The code fragment to indicate the point's accumulation and calculation were demonstrated in Figure 3.6.17, while the design of the seven-segment display module, as illustrated in Figure 3.6.18.

```

if(((boardBposition+sizeB+2)>=originalX)&&((boardBposition-sizeB-2)<=originalX))
begin    //Bounce from paddle (player B)
directY=0;
end
else //Increase point for player A if player B did not catch the ball and reset the ball location
begin
directY=1;
originalX=320;
originalY=240;
scoreA=scoreA+15;
if(scoreA==90) state=1;
end
end
else if(originalY==474)
begin
if(((boardAposition+sizeA+2)>=originalX)&&((boardAposition-sizeA-2)<=originalX))
begin    //Bounce from paddle (player A)
directY=1;
end
else //Increase point for player B if player A did not catch the ball and reset the ball location
begin
directY=0;
originalX=320;
originalY=240;
scoreB=scoreB+15;
if(scoreB==90) state=2;
end
end
end
end

```

Figure 3.6.17 Code fragment to indicate the points accumulation

```

module SegOUT(in,out);
input [3:0]in;
output reg [6:0]out;
always@(in)
begin
case(in)
0:out=~7'b0111111;
1:out=~7'b0000110;
2:out=~7'b1011011;
3:out=~7'b1001111;
4:out=~7'b1100110;
5:out=~7'b1101101;
6:out=~7'b1111100;
7:out=~7'b0000111;
8:out=~7'b1111111;
9:out=~7'b1100111;
default out=~7'b0000000;
endcase
end
endmodule

```

Figure 3.6.18 Design of seven segment display module

g) The judgment of victory and defeat

The judgment of victory and defeat was done by comparing the accumulated points of the players. The winner will be the first player who obtained 90 points, and the result will be shown on the LCD. The designed LCD module was presented in Figure 3.6.19.

```

always@ (game_state)begin
  if(game_state==0)begin
    data[0]=8'b00100000;
    data[1]=8'b01010000;//p
    data[2]=8'b01101001;//l
    data[3]=8'b01101110;//n
    data[4]=8'b01100111;//g
    data[5]=8'b00100000;
    data[6]=8'b01010000;//p
    data[7]=8'b01101111;//o
    data[8]=8'b01101110;//n
    data[9]=8'b01100111;//g
    data[10]=8'b00100000;
    data[11]=8'b00100000;
    data[12]=8'b00100000;
    data[13]=8'b00100000;
    data[14]=8'b00100000;
    data[15]=8'b00100000;
    //
    data[16]=8'b00100000;
    data[17]=8'b00100000;
    data[18]=8'b01000111;//G
    data[19]=8'b01100001;//a
    data[20]=8'b01101101;//m
    data[21]=8'b01100101;//e
    data[22]=8'b00100000;
    data[23]=8'b00100000;
    data[24]=8'b00100000;
    data[25]=8'b00100000;
    data[26]=8'b00100000;
    data[27]=8'b00100000;
    data[28]=8'b00100000;
    data[29]=8'b00100000;
    data[30]=8'b00100000;
    data[31]=8'b00100000;
  end

  else if(game_state==1)begin
    data[0]=8'b00100000;
    data[1]=8'b01010000;//P
    data[2]=8'b01101100;//l
    data[3]=8'b01100001;//a
    data[4]=8'b01111001;//y
    data[5]=8'b01100101;//e
    data[6]=8'b01110010;//z
    data[7]=8'b00100000;//
    data[8]=8'b00100000;//
    data[9]=8'b01000001;//A
    data[10]=8'b00100000;
    data[11]=8'b00100000;
    data[12]=8'b00100000;
    data[13]=8'b00100000;
    data[14]=8'b00100000;
    data[15]=8'b00100000;
    //
    data[16]=8'b00100000;
    data[17]=8'b00100000;
    data[18]=8'b01001001;//I
    data[19]=8'b01110011;//s
    data[20]=8'b00100000;//
    data[21]=8'b01010111;//W
    data[22]=8'b01101001;//i
    data[23]=8'b01101110;//n
    data[24]=8'b01101110;//n
    data[25]=8'b01100101;//e
    data[26]=8'b01110010;//z
    data[27]=8'b00100001;//!
    data[28]=8'b00100000;
    data[29]=8'b00100000;
    data[30]=8'b00100000;
    data[31]=8'b00100000;
  end

  else if(game_state==2)begin
    data[0]=8'b00100000;
    data[1]=8'b01010000;//P
    data[2]=8'b01101100;//l
    data[3]=8'b01100001;//a
    data[4]=8'b01111001;//y
    data[5]=8'b01100101;//e
    data[6]=8'b01110010;//z
    data[7]=8'b00100000;//
    data[8]=8'b00100000;//
    data[9]=8'b01000010;//B
    data[10]=8'b00100000;
    data[11]=8'b00100000;
    data[12]=8'b00100000;
    data[13]=8'b00100000;
    data[14]=8'b00100000;
    data[15]=8'b00100000;
    //
    data[16]=8'b00100000;
    data[17]=8'b00100000;
    data[18]=8'b01001001;//I
    data[19]=8'b01110011;//s
    data[20]=8'b00100000;//
    data[21]=8'b01010111;//W
    data[22]=8'b01101001;//i
    data[23]=8'b01101110;//n
    data[24]=8'b01101110;//n
    data[25]=8'b01100101;//e
    data[26]=8'b01110010;//z
    data[27]=8'b00100001;//!
    data[28]=8'b00100000;
    data[29]=8'b00100000;
    data[30]=8'b00100000;
    data[31]=8'b00100000;
  end
endmodule

```

Figure 3.6.19 Design of LCD module

3.7 Summary

Chapter 3 is mainly targeted on how the project was being carried out, which included project planning, equipment learning, experimental setup, preliminary testing, and discuss the module design involved in the game development. A project is typically distributed into small tasks due to the ease of monitoring. Each task was given an estimated duration to complete it, and it is vital to handling the project without any delay. Hence, a task-based approach and Gantt chart was implemented to ensure that each task can be completed on time and left no one behind.

Under this topic, the development steps and ideas of the design were basically grabbed from the discussion in Chapter 2. The design modules applied in the Tetris game were further modified and applied in this project. As similar to the Tetris game, the overall design involved in this project was the VGA module, input key module as well as the game control module. Each task was explained in detail, providing the function of the module, the effect of the game, the method of design, and the communication of the module to FPGA. Besides that, the system implementation flow that was studied from Dice game was modified and implied in this research, and all the results obtained will be further discussed in Chapter 4.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

The tasks described in Chapter 3 have been thoroughly designed and conducted. In Chapter 4, all the results obtained will be discussed and analyzed in detail.

4.2 Result of Preliminary Testing

The results of testing will be analyzed and further employed in game development. If there are any problems encountered, the issues can be figured out and solved at the early stage.

4.2.1 Testing of CAD Tool

During the test, a simple circuit for two-way light control was designed in Verilog code. Both the compilation in Quartus II and Modelsim was successful, as shown in Figure 4.2.1 and Figure 4.2.2, respectively.

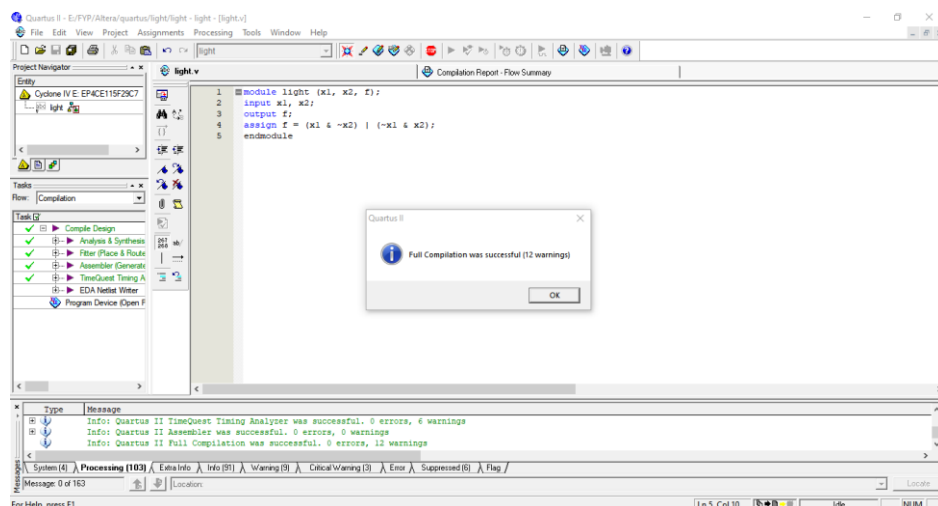


Figure 4.2.1 Compilation Result in Quartus II

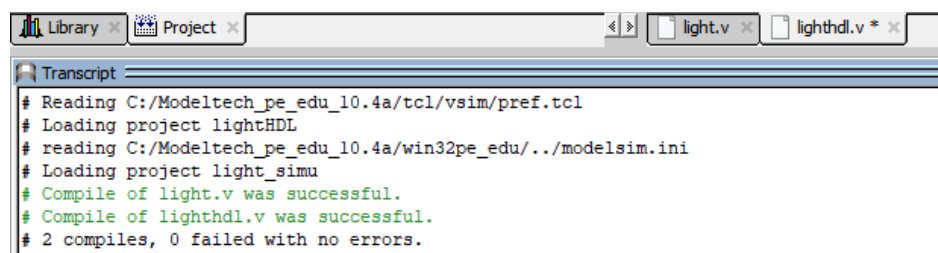


Figure 4.2.2 Compilation Result in Modelsim

In Figure 4.2.3, it can be observed that the simulation results in Modelsim were matched with the proposed truth table provided in Figure 3.5.2, which means that the accuracy of the designed circuit was verified in the software part.

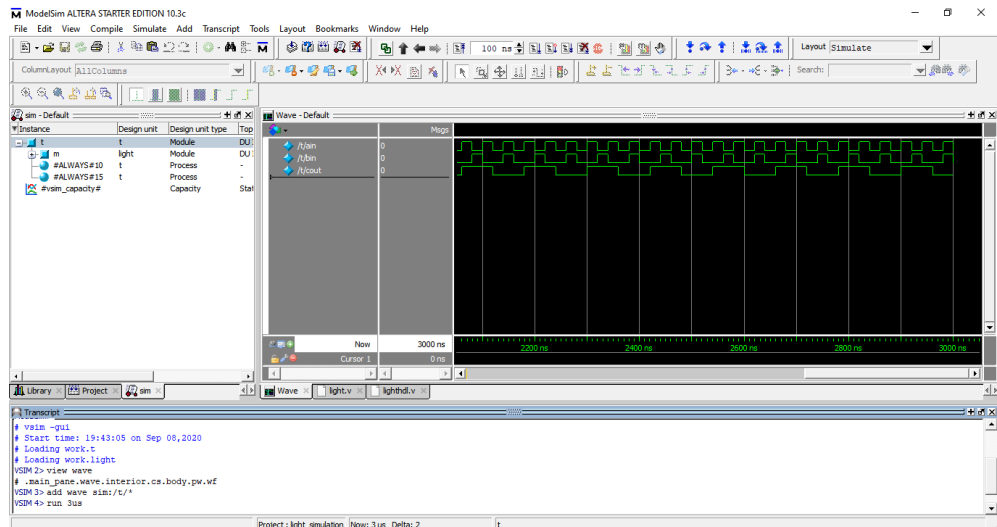


Figure 4.2.3 Simulation Result in Modelsim

While it comes to the hardware part, in order to connect both the physical hardware and simulated software, JTAG mode was used as the configuration method. The configuration data was loaded from the host computer to the board through the connection of USB blaster. Figure 4.2.4 depicted that the loading process was successful without any errors.

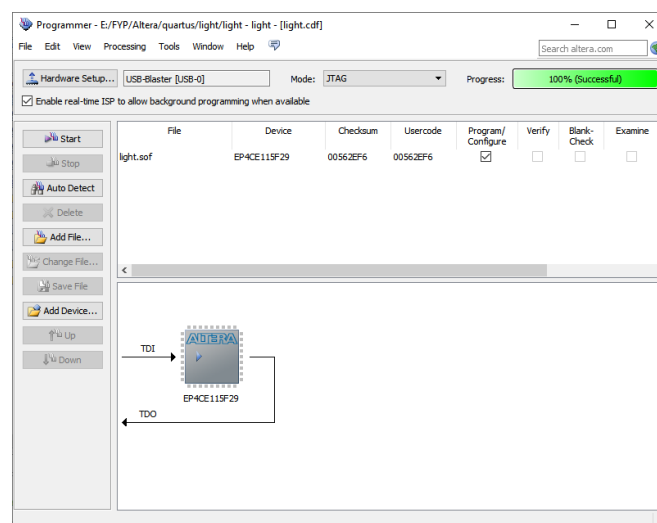


Figure 4.2.4 Process of loading the configuration data into FPGA (successful)

After the configuration, the result can be observed and verified through the visual feedback from the FPGA device. Table 4.2.1 summarized and

compared both the theoretical and practical results, and it can be observed that both the result was matched. It can be concluded that the design was successful, and the verified CAD flow can be further applied in the technical design.

Table 4.2.1 Comparison of theoretical and practical result

Theoretical Result			Practical Result
x_1	x_2	f	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

4.2.2 Testing of FPGA

This test involves both types of design entries, which included the Verilog file (.v) and Block Design File (.bdf). The extra features of PLL Megafunction and Multiplexer are doing the project to be more challenging. The knowledge learned will be further applied in the specific design. Figure 4.2.5 shows that the compilation of the design was successful.

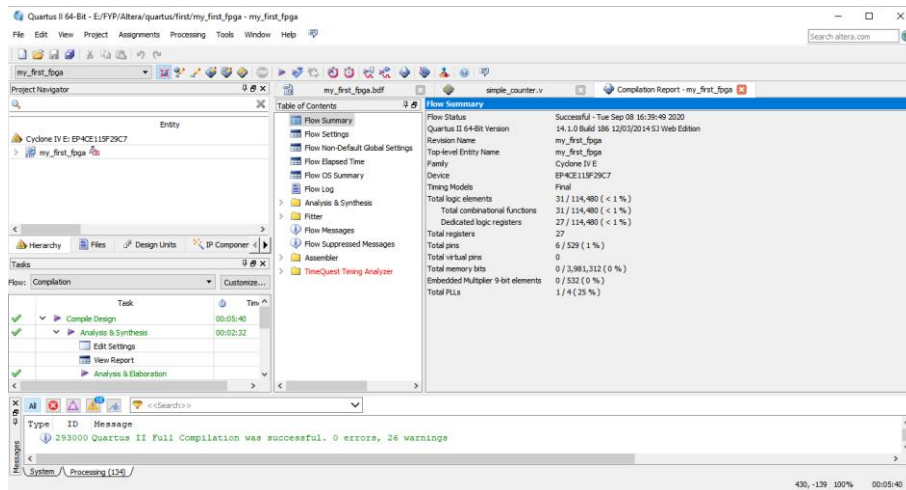


Figure 4.2.5 Compilation Report (Successful)

After compiling and verifying the design, the next step is to configure the FPGA board by transferring the configuration data into the FPGA through the USB-Blaster circuitry on the board. The compiled SRAM Object File (.sof) was loaded onto the FPGA through JTAG mode configuration, and the design was run successfully, as shown in Figure 4.2.6.

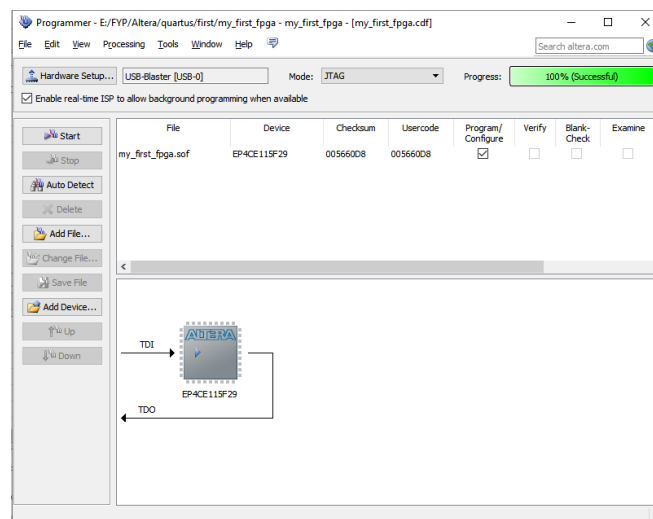
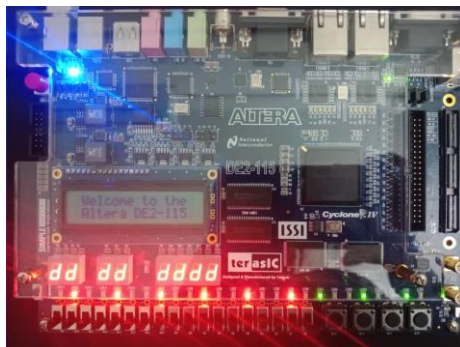


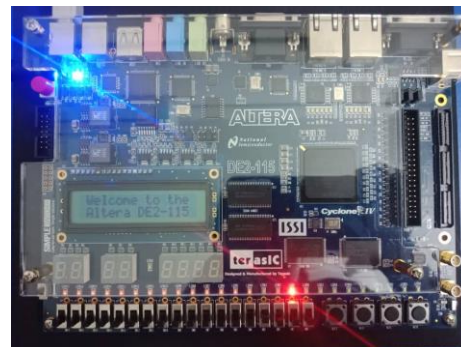
Figure 4.2.6 FPGA configuration (successful)

After verifying the design, the runtime behavior of the FPGA hardware design can be observed to make sure that it is functioning accurately. The design can be verified through observing the four LEDs in the FPGA board appear to be changing in a binary count pattern, which was controlled by the `simple_counter` bits [26..23]. The LEDs were active low; thus, when counting starts, all LEDs were turned on (the state of 0000). By pressing and holding `KEY [0]`, the LEDs advance quicker due to the implementation of the multiplex, which was handled by the counter bits [24..21]. The runtime behavior of the FPGA was captured and summarized in Table 4.2.2. However, since the LEDs are kept blinking, hence it cannot be fully snapshotted and verifying its correctness in the report. In conclusion, this testing is considered successful as the compilation and configuration were done with zero error. Besides that, the runtime behavior of FPGA was studied through the visual feedback from the LEDs on the FPGA device.

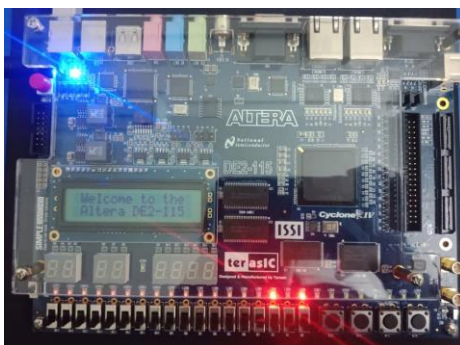
Table 4.2.2 Summarized result to show the runtime behaviour of FPGA



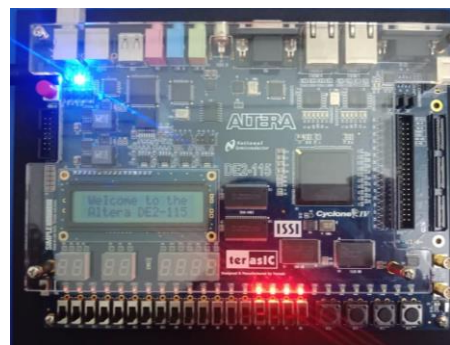
FPGA device before configuration



LEDs are advancing slowly in a binary count pattern



The LEDs advance quicker when `KEY [0]` was pressed and held



The LEDs advance at the quickest rate when `KEY [0]` was pressed and held

4.2.3 Testing of VGA monitor

The test of the VGA monitor is crucial since the display of the design is the only way to verify the design by giving the designer direct visual feedback, determining whether the design works. The test has been carried out by connecting the monitor directly to the host computer through VGA cable. From Figure 4.2.7, the result verified that the monitor could be functioned well and displayed smoothly through VGA.



Figure 4.2.7 Display on VGA monitor when connected to the host computer

4.3 Experiment on Game Development: Ping Pong Game

4.3.1 Pin Assignment

In order to control the game, the key inputs have to be set preliminary. Once the player pressed on the respective key inputs, the system will respond to it and display it on the monitor. To connect the preliminary set input from the designed code to the physical FPGA board, it is necessary to assign the FPGA pins with the respective function. (Altera Corporation, 2010)

In this design, the key inputs are assigned to FPGA pins like push buttons, clocks, slide switches, VGA outputs, and LCD. Each pin consists of their specific PIN and function. Therefore, the pin assignment was done by referring to APPENDIX D. The overall pin planner was shown in Figure 4.3.1, and the exported pin assignment in the CSV file can be found in APPENDIX B.

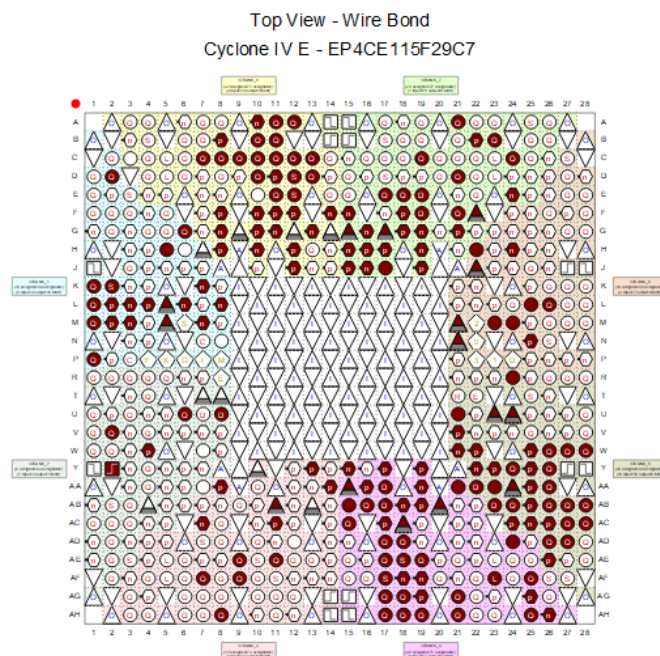


Figure 4.3.1 Pin Planner (Ping Pong Game)

4.3.2 Compilation Result

Figure 4.3.2 shows the compilation report of the designed circuit. As shown in the message box, the design was compiled successfully with zero errors.

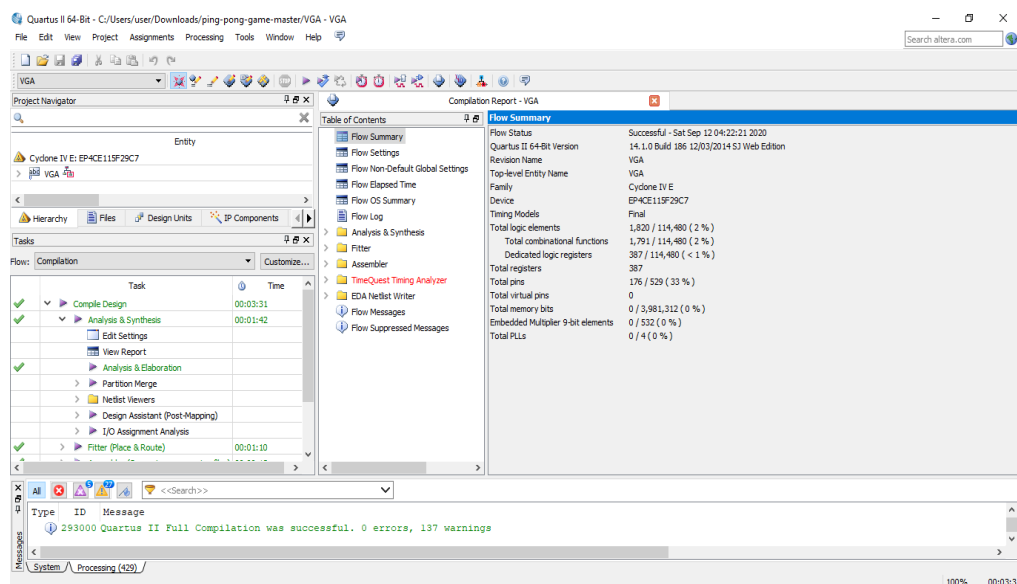


Figure 4.3.2 Compilation Result (Successful)

4.3.3 JTAG Configuration

During the loading of configuration data into the FPGA device via the JTAG configuration mode, Figure 4.3.3 shows that the loading progress was successful.

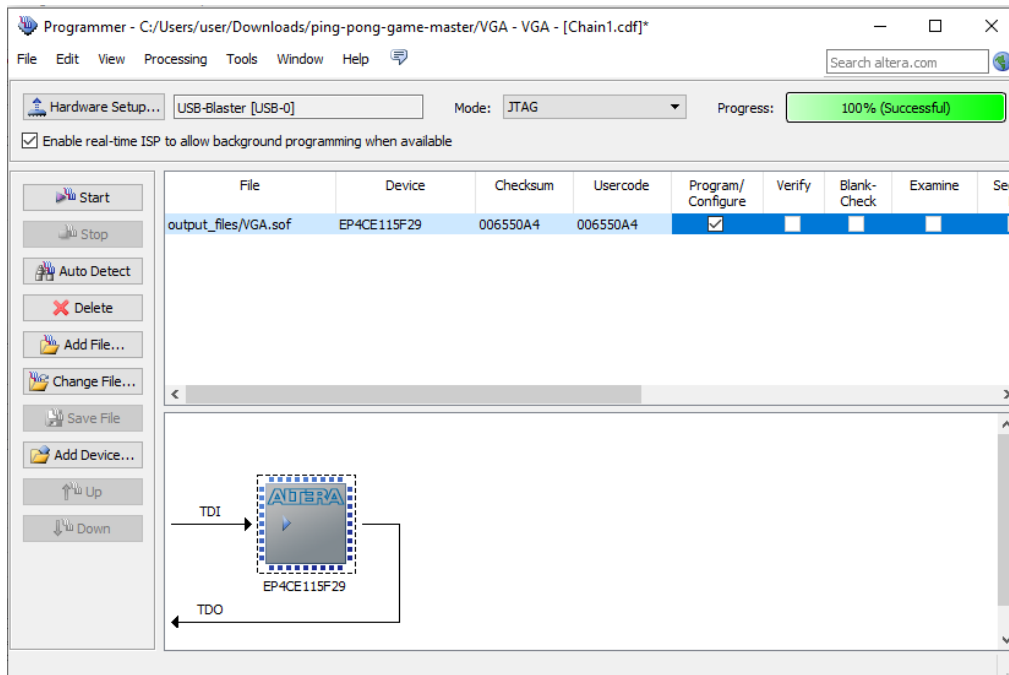


Figure 4.3.3 Process of loading configuration data into FPGA (Successful)







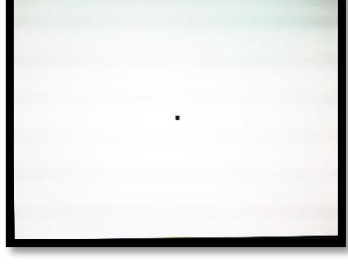

4.3.4 VGA Display

This section will show the game display on the VGA monitor. The game graphic includes the color of paddles as well as the color of the ball.

a) Color of Paddle

The color of the paddle was altered when the binary state changed. The result was shown in Table 4.3.1. The result obtained was the same as the truth table provided in Section 3.6.3.

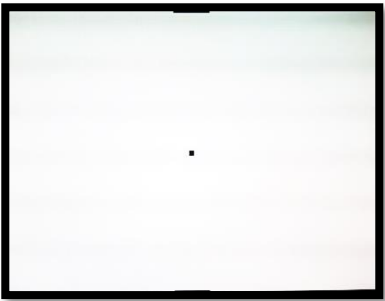
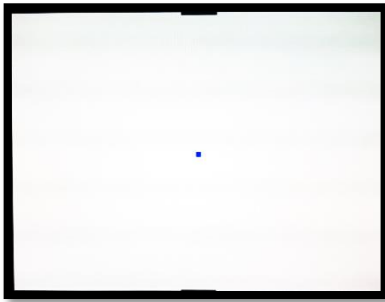
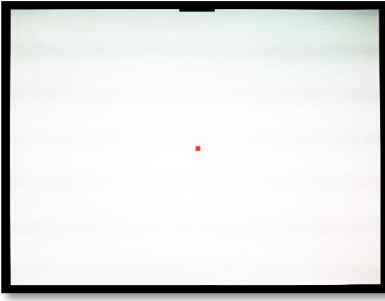


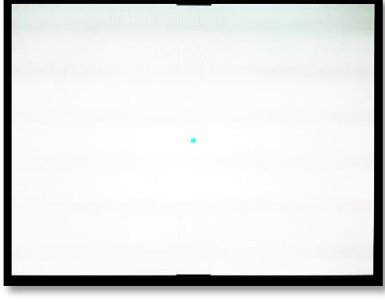


Table 4.3.1 Display of paddle color according to their respective binary state

State	Display of paddle color	State	Display of paddle color
000		100	
001		101	
010		110	
011		111	

b) Color of ball

The color of the ball was altered when the binary state changed, as illustrated in Table 4.3.2. It can be observed that the result obtained was the same as the truth table provided in Section 3.6.3.

Table 4.3.2 Display of the ball color according to their respective binary state

State	Display of ball color	State	Display of ball color
000		100	
001		101	
010		110	
011		111	

4.3.5 Results of Game Control Module

a) The initial location of the ball and paddles

The initial location of the ball and paddles were set as in Figure 4.3.4, when the RESET key is inputted, the ball and the paddles will reset back to their original position, and the game will restart again.

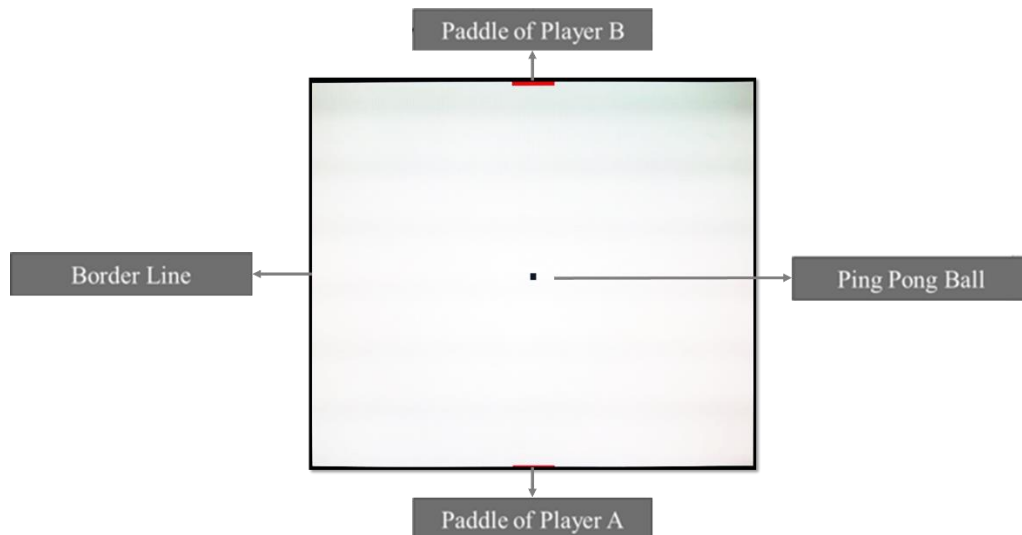
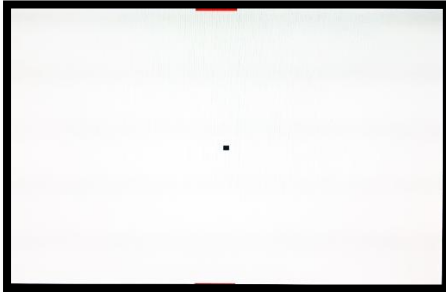
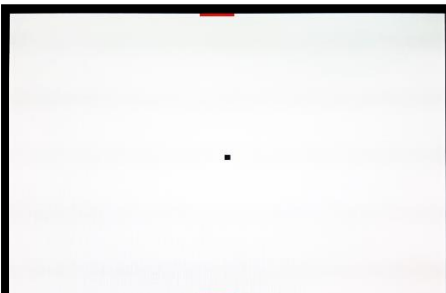
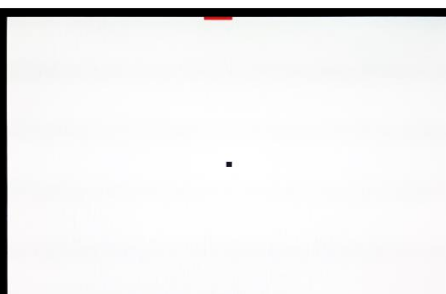
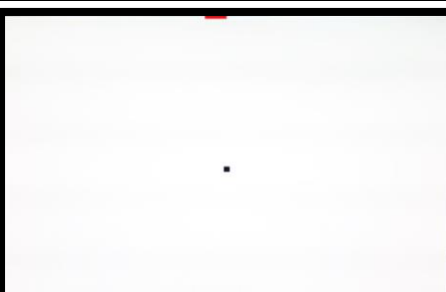


Figure 4.3.4 Initial location of the ball and paddles

b) The size control of the paddles

The paddle size can be controlled by the players to enlarge or diminish it according to their binary state, as shown in Table 4.3.3. It is found that the result obtained was the same as the truth table provided in Section 3.6.3.

Table 4.3.3 Display of the paddle size according to their respective binary state

State	Display of paddle size
00	
01	
10	
11	

c) The point's accumulation and calculation

As observed from Figure 4.3.5, the game started with 0 points for both players. The SSD on the right side is displaying the accumulated points of Player A while the left side of the SSD is displaying the accumulated points of Player B. The sample display on SSD can be found in Figure 4.3.6 and Figure 4.3.7.

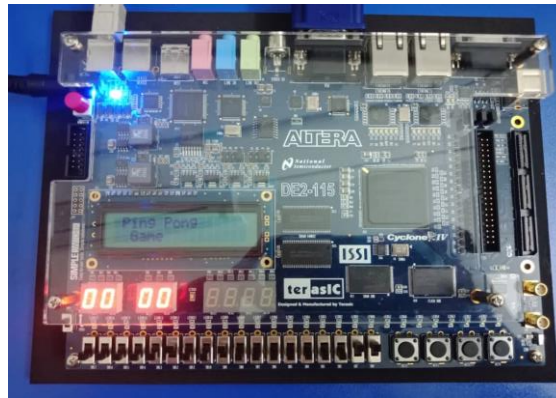


Figure 4.3.5 Start of the game

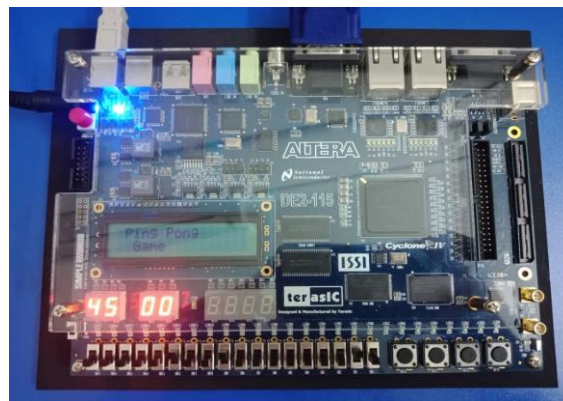


Figure 4.3.6 Player B gained 45 points



Figure 4.3.7 Player A gained 30 points while Player B gained 60 points

d) The judgment of victory and defeat

Figure 4.3.8 illustrated the condition when Player A won the game, while Figure 4.3.9 indicates the condition when Player B won the game. The judgment of victory and defeat was done by comparing the accumulated points of the players. The first player earned 90 points will be the winner of the game, and the game will end immediately. The game will start again only when the RESET key is inputted.

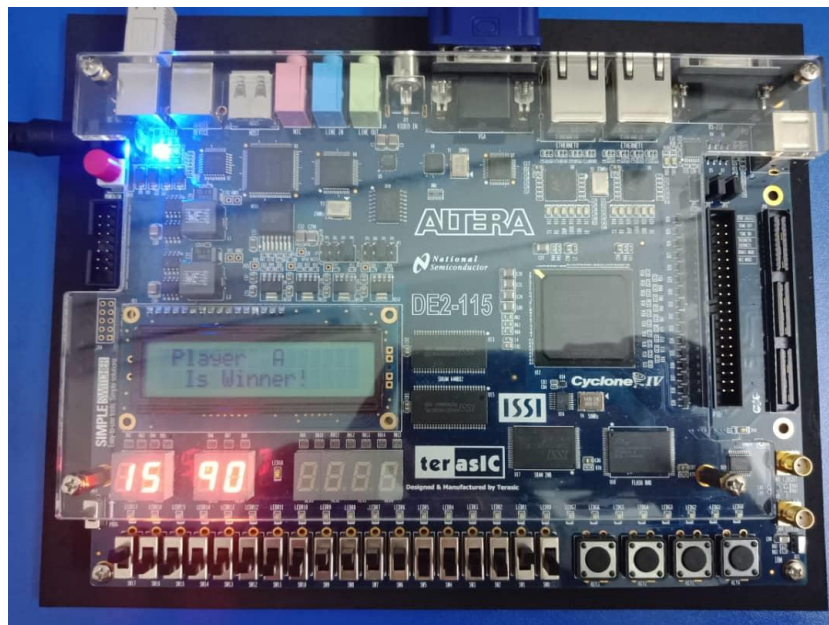


Figure 4.3.8 Display on the DE2-115 board when Player A wins

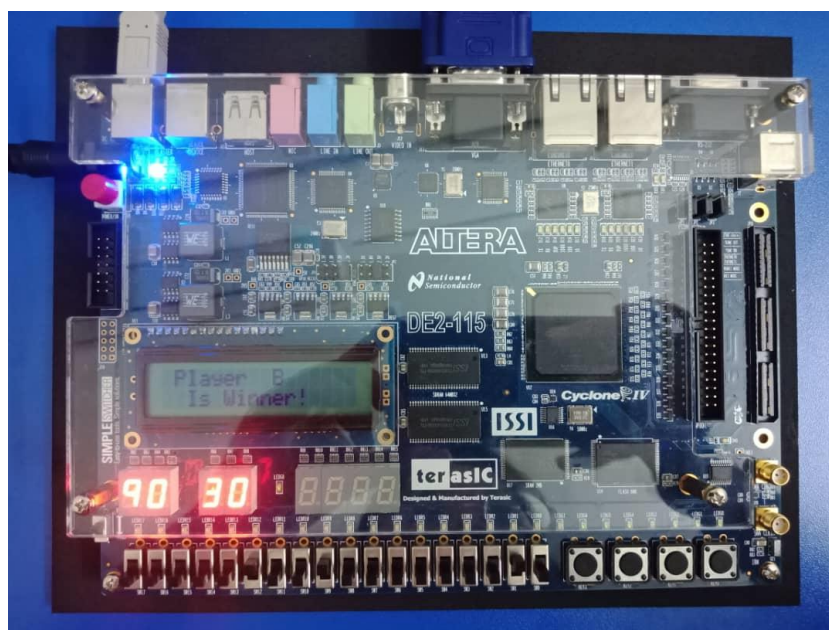


Figure 4.3.9 Display on the DE2-115 board when Player B wins

4.4 Summary

This chapter mainly focused on the discussion of the result obtained during the three preliminary testing as well as game development. For the testing of the CAD tool, FPGA, as well as the VGA monitor, the theoretical result and practical result were matched and considered successful. After conducting the test, if there were any problems encountered, the issues can be discovered and solved during the early stage of the project.

During the development of the FPGA-based ping-pong game, the game display on the VGA monitor proved that the practical results were fully matching with the theoretical results. Firstly, the initial location of the ball and paddles was set according to the XY-coordinates. The ball and the paddles were able to be back to their initial location when the RESET key was inputted. Next, the display results were verified by comparing the game display to the truth table; for example, the color of the ball and paddles. The color of the ball and paddles can be altered by changing the binary state of the slide switches. Lastly, the point's accumulation system and the judicial system were designed successfully as the FPGA can calculate and compare the points, decide the winner, and then display on the SSD and LCD.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The project design comes with hardware and software design, where both works coherently to introduce the development of ping pong game based on FPGA. The project included the design of input keys, VGA display, location and movement of the ball and paddles, principle of ball de-bouncing, speed control of the ball, size control of the paddle, point's accumulation and calculation as well as the judgment of victory and defeat.

Throughout the game development, the relevant theory of FPGA was implied in the game application, and hence a more in-depth understanding of FPGA and Verilog language has been acquired, for instance, the features, standard, pins connection, as well as the CAD flow of FPGA, were studied.

Besides that, in order to highlight the benefits of FPGA, the difficulty level of the game was increased by adding the extra features, where the players can control the speed of the ball and the size of the paddles. More benefits of FPGA have been discovered and emphasized in this project as compared to other ASICs.

As observed from the experimental results obtained, the compilation of the game design is found to be successful. The game can be displayed and functioned smoothly without any delay. In conclusion, this project is considered successful as all the objectives were achieved.

5.2 Recommendations for future work

In this project, a complete framework for the design of an FPGA-based ping pong game using Verilog has been built. However, there are several things to be accomplished in the future.

First, the inputting method in this project was done by the input keys from the development board. Instead of using the internal push buttons and slide switches, it is recommended to improve it to be controlled by an external keyboard, for example, a PS/2 keyboard.

Besides, the display of points and winners was done by using the seven segment display and LCD module, respectively. It is encouraged to enhance it to be displayed on the VGA monitor, which includes the "game characters" display functions to display 26 capital letters of English characters (A to Z) and 10 Roman numerals (0 to 9) on the VGA monitor.

Lastly, it is suggested to involve the design of the audio module in order to enhance the users' experience.

REFERENCES

- Altera Corporation, 2010. *U. Manual, "Altera DE2 Board"*, s.l.: Terasic Technologies.
- Altera Corporation, 2006. *U. Manual, "Altera DE2 Board"*. s.l.:s.n.
- Brown, S. and Rose, J., 2002. *Architecture of FPGAs and CPLDs: A Tutorial*, Toronto: EECG.
- Center, N. L. H., 2017. *CPLD vs FPGA: Differences between them and which one to use?*. [Online]
Available at: <https://numato.com/kb/cpld-vs-fpga-differences-one-use/>
[Accessed 25 March 2020].
- Elprocus, 2019. *Introduction to Application Specific Integrated Circuit (ASIC)*. [Online]
Available at: <https://www.elprocus.com/application-specific-integrated-circuits/>
[Accessed 25 March 2020].
- England, T. T., n.d. *THE BASIC RULES OF TABLE TENNIS*. [Online]
Available at: https://www.keepintheloop.uk/rules-of-table-tennis/?doing_wp_cron=1587803200.2764790058135986328125
[Accessed 02 April 2020].
- fpga4fun, n.d. *Pong Game*. [Online]
Available at: <https://www.fpga4fun.com/PongGame.html>
[Accessed 2 April 2020].
- Inventions, A., 2017. *FPGA advantages*. [Online]
Available at: <https://www.af-inventions.de/en/services/fpga-development/fpga-advantages.html>
[Accessed 18 April 2020].
- Keim, R., 2018. *What Is an FPGA? An Introduction to Programmable Logic*. [Online]
Available at: <https://www.allaboutcircuits.com/technical-articles/what-is-an-fpga-introduction-to-programmable-logic-fpga-vs-microcontroller/>
[Accessed 25 March 2020].

Kleitz, W., 2006. *Digital Electronics with VHDL*. Quartus II Version ed. s.l.:Pearson Prentice Hall.

Krazytech, 2015. *Programmable Logic Devices*. [Online] Available at: <https://krazytech.com/technical-papers/programmable-logic-devices-pld>

[Accessed 25 March 2020].

Liu, K. Corporation, 2012. Tetris game design based on the FPGA. *IEEE*, pp. 2925-2928.

Toonsi S. Corporation, 2017. VHDL Based Circuits Design and Synthesis on FPGA: A Dice Game Example. *2017 IEEE 2nd International Conference on Signal and Image Processing*, pp. 418-422.

Touger, E., 2018. *What Is an FPGA and Why Is It a Big Deal?*. [Online] Available at: <https://www.prowesscorp.com/what-is-fpga/> [Accessed 19 April 2020].

WatElectronics, 2019. *Basic FPGA Architecture and its Applications*. [Online] Available at: <https://www.watelectronics.com/fpga-architecture-applications/> [Accessed 18 April 2020].

Wolff, H., 1990. *Electronics*. [Online] Available at: <https://researchers.dellmed.utexas.edu/en/publications/how-quickturn-is-filling-a-gap>

[Accessed 25 March 2020].

Zhang, G. and Xie, M., 2010. Design of Visual Based-FPGA Ping-pang Game with. *IEEE*, pp. 31-34.

APPENDICES

APPENDIX A

Designed Verilog Code for FPGA-based Ping-Pong Game

```
`include "LCD.v"
module VGA (
    ////////////////////////////////////////////////// Clock Input ///////////////////////////////////
    input          iCLK_28,          // 28.63636 MHz
    input          iCLK_50,          // 50 MHz
    input          iCLK_50_2,        // 50 MHz
    input          iCLK_50_3,        // 50 MHz
    input          iCLK_50_4,        // 50 MHz
    input          iEXT_CLOCK,       // External Clock
    ////////////////////////////////////////////////// Push Button ///////////////////////////////////
    input [3:0]    iKEY,             // Pushbutton[3:0]
    ////////////////////////////////////////////////// DPDT Switch ///////////////////////////////////
    input [17:0]   iSW,             // Toggle Switch[17:0]
    ////////////////////////////////////////////////// 7-SEG Dispaly ///////////////////////////////////
    output [6:0]   oHEX0_D,          // Seven Segment Digit 0
    output [6:0]   oHEX1_D,          // Seven Segment Digit 1
    output [6:0]   oHEX2_D,          // Seven Segment Digit 2
    output [6:0]   oHEX3_D,          // Seven Segment Digit 3
    output [6:0]   oHEX4_D,          // Seven Segment Digit 4
    output [6:0]   oHEX5_D,          // Seven Segment Digit 5
    output [6:0]   oHEX6_D,          // Seven Segment Digit 6
    output [6:0]   oHEX7_D,          // Seven Segment Digit 7
    ////////////////////////////////////////////////// LED ///////////////////////////////////
    output [8:0]   oLEDG,            // LED Green[8:0]
    output [17:0]  oLEDR,            // LED Red[17:0]
    //////////////////////////////////////////////////LCD//////////////////////////////////////
    inout  [7:0]   LCD_D,            // LCD Data bus 8 bits
    output          oLCD_ON,         // LCD Power ON/OFF
    output          oLCD_BLON,       // LCD Back Light ON/OFF
    output          oLCD_RW,         // LCD Read/Write Select, 0 = Write, 1 = Read
    output          oLCD_EN,         // LCD Enable
    output          oLCD_RS,         // LCD Command/Data Select, 0 = Command, 1 = Data

    ////////////////////////////////////////////////// VGA ///////////////////////////////////
    output          oVGA_CLOCK,      // VGA Clock
    output          oVGA_HS,         // VGA H_SYNC
    output          oVGA_VS,         // VGA V_SYNC
    output          oVGA_BLANK_N,    // VGA BLANK
    output          oVGA_SYNC_N,     // VGA SYNC
    output [7:0]    oVGA_R,          // VGA Red[9:0]
    output [7:0]    oVGA_G,          // VGA Green[9:0]
    output [7:0]    oVGA_B,          // VGA Blue[9:0]
    output [20:0]   change,
    output [1:0]    ostate
);
// Horizontal Parameter
parameter H_FRONT = 16;
parameter H_SYNC  = 96;
parameter H_BACK  = 48;
parameter H_ACT   = 640;
parameter H_BLANK = H_FRONT + H_SYNC + H_BACK;
parameter H_TOTAL = H_FRONT + H_SYNC + H_BACK + H_ACT;

// Vertical Parameter
parameter V_FRONT = 11;
parameter V_SYNC  = 2;
parameter V_BACK  = 32;
parameter V_ACT   = 480;
parameter V_BLANK = V_FRONT + V_SYNC + V_BACK;
parameter V_TOTAL = V_FRONT + V_SYNC + V_BACK + V_ACT;
```

```

wire CLK_25;
wire CLK_move;
wire CLK_to_DAC;
wire RST_N;
wire left_one;
wire right_one;
wire left_two;
wire right_two;
PLL pll0 (
    .inclk0(iCLK_50),
    .c0(CLK_25)
);
PLL #(500000)pll1(
    .inclk0(iCLK_50),
    .c0(CLK_move)
);
LCD lcdout(iCLK_50,RST_N,{oLCD_EN,oLCD_RS,oLCD_RW,LCD_D},ostate);
// Select DAC clock
assign CLK_to_DAC = CLK_25;
assign oVGA_SYNC_N = 1'b0; // This pin is unused.
assign oVGA_BLANK_N = ~((H_Cont<H_BLANK)||(V_Cont<V_BLANK));
assign oVGA_CLOCK = ~CLK_to_DAC; // Invert internal clock to output clock
assign RST_N = iSW[1]; // Set reset signal is KEY[0]
assign right_B = iKEY[0];
assign left_B = iKEY[1];
assign right_A = iKEY[2];
assign left_A = iKEY[3];
assign oLCD_ON = 1'b1;
assign oLCD_BLON = 1'b0;
reg [10:0] H_Cont;
reg [10:0] V_Cont;
reg [7:0] vga_r;
reg [7:0] vga_g;
reg [7:0] vga_b;
reg vga_hs;
reg vga_vs;
reg [10:0] X;
reg [10:0] Y;
reg [6:0] scoreA;
reg [6:0] scoreB;
reg [1:0] state;
integer originalX=320;
integer originalY=240;
integer boardAposition=320;
integer boardBposition=320;
reg directX,directY;
reg [35:0] counter;
integer countboard=0;
reg [7:0] dateR,dateG,dateB;
reg [7:0] boardA_R,boardA_G,boardA_B;
reg [7:0] boardB_R,boardB_G,boardB_B;
reg [5:0] sizeA,sizeB;
reg [4:0] speed;
assign ostate = state;
assign oVGA_R = vga_r;
assign oVGA_G = vga_g;
assign oVGA_B = vga_b;
assign oVGA_HS = vga_hs;
assign oVGA_VS = vga_vs;
SegOUT SEG0(10,oHEX0_D);
SegOUT SEG1(10,oHEX1_D);
SegOUT SEG2(10,oHEX2_D);
SegOUT SEG3(10,oHEX3_D);
SegOUT SEG4(scoreA%10,oHEX4_D);
SegOUT SEG5(scoreA%100/10,oHEX5_D);
SegOUT SEG6(scoreB%10,oHEX6_D);
SegOUT SEG7(scoreB%100/10,oHEX7_D);

```

```

// Horizontal Generator: Refer to the pixel clock
always@(posedge CLK_to_DAC, negedge RST_N) begin
    if(!RST_N) begin
        H_Count <= 0;
        vga_hs <= 1;
        X <= 0;
    end
    else begin
        if (H_Count < H_TOTAL)
            H_Count <= H_Count+1'b1;
        else
            H_Count <= 0;
        // Horizontal Sync
        if(H_Count == H_FRONT-1) // Front porch end
            vga_hs <= 1'b0;
        if(H_Count == H_FRONT + H_SYNC -1) // Sync pulse end
            vga_hs <= 1'b1;
        // Current X
        if(H_Count >= H_BLANK)
            X <= H_Count-H_BLANK;
        else
            X <= 0;
    end
end

// Vertical Generator: Refer to the horizontal sync
always@(posedge oVGA_HS, negedge RST_N) begin
    if(!RST_N) begin
        V_Count <= 0;
        vga_vs <= 1;
        Y <= 0;
    end
    else begin
        if (V_Count<V_TOTAL)
            V_Count <= V_Count + 1'b1;
        else
            V_Count <= 0;
        // Vertical Sync
        if (V_Count == V_FRONT-1) // Front porch end
            vga_vs <= 1'b0;
        if (V_Count == V_FRONT + V_SYNC-1) // Sync pulse end
            vga_vs <= 1'b1;
        // Current Y
        if (V_Count >= V_BLANK)
            Y <= V_Count-V_BLANK;
        else
            Y <= 0;
    end
end

// Pattern Generator
always@(posedge CLK_to_DAC, negedge RST_N) begin
    if(!RST_N) begin
        vga_r <= 0;
        vga_g <= 0;
        vga_b <= 0;
    end
    else
        begin
            {vga_r,vga_g,vga_b} <= (Y>=(originalY-4)&&Y<=(originalY+4)&&X>=(originalX-4)&&(X<=(originalX+4))) ? {dateR,dateG,dateB} :
                (Y<480&&Y>=476&&((boardAposition+sizeA)>=X)&&((boardAposition-sizeA)<=X)) ? {boardA_R,boardA_G,boardA_B} :
                (Y<4&&Y>=0&&((boardBposition+sizeB)>=X)&&((boardBposition-sizeB)<=X)) ? {boardB_R,boardB_G,boardB_B} :
                24'hFFFFFF;
        end
    end
end

```

```

always@(negedge iCLK_50 or negedge RST_N)
begin
if (~RST_N)
begin
counter=0;
originalX=320;
originalY=240;
directX=0;
directY=0;
scoreA=0;
scoreB=0;
state=0;
end
else
begin
if (~iSW[0])
begin
if (state==0)
begin
if (counter==(speed*50000))
begin
if (directX&&directY) //Upper left corner movement
begin
originalX=originalX-2;
originalY=originalY-1;
end
else if (~directX&&directY) //Upper right corner movement
begin
originalX=originalX+2;
originalY=originalY-1;
end
else if (directX&&~directY) //Bottom left corner movement
begin
originalX=originalX-2;
originalY=originalY+1;
end

```

```

else if (~directX&&~directY) //Bottom right corner movement
begin
originalX=originalX+2;
originalY=originalY+1;
end
if (originalX==4) directX=0; //Bounce from right
else if (originalX==636) directX=1; //Bounce from left
if (originalY==6)
begin
if (((boardBposition+sizeB+2)>=originalX)&&((boardBposition-sizeB-2)<=originalX))
begin //Bounce from paddle (player B)
directY=0;
end
else //Increase point for player A if player B did not catch the ball and reset the ball location
begin
directY=1;
originalX=320;
originalY=240;
scoreA=scoreA+15;
if (scoreA==90) state=1;
end
end
else if (originalY==474)
begin
if (((boardAposition+sizeA+2)>=originalX)&&((boardAposition-sizeA-2)<=originalX))
begin //Bounce from paddle (player A)
directY=1;
end
else //Increase point for player B if player A did not catch the ball and reset the ball location
begin
directY=0;
originalX=320;
originalY=240;
scoreB=scoreB+15;
if (scoreB==90) state=2;
end
end
end
end

```

```

        counter=0;
    end
    else counter=counter+1;
    end
    else
    begin
        state=3;
    end
    end
end
end
always@(negedge iCLK_50)
begin
    if(state==0)
    begin
        if(countboard==500000)
        begin
            if(~left_A)boardAposition<=boardAposition-2;
            if(~right_A)boardAposition<=boardAposition+2;
            if(~left_B)boardBposition<=boardBposition-2;
            if(~right_B)boardBposition<=boardBposition+2;
            countboard=0;
        end
        else
        begin
            countboard=countboard+1;
            if(boardAposition<sizeA)boardAposition<=sizeA;
            if(boardAposition>(640-sizeA))boardAposition<=640-sizeA;
            if(boardBposition>(640-sizeB))boardBposition<=640-sizeB;
            if(boardBposition<sizeB)boardBposition<=sizeB;
        end
    end
end
always@(posedge iCLK_50) //colour of ball
begin
    if(iSW[15])dateR=8'hFF;
    else dateR=8'h0;
    if(iSW[16])dateG=8'hFF;
    else dateG=8'h0;
    if(iSW[17])dateB=8'hFF;
    else dateB=8'h0;
end
always@(posedge iCLK_50)//paddle for player A
begin
    if(iSW[12])boardB_R=8'hFF;
    else boardB_R=8'h0;
    if(iSW[13])boardB_G=8'hFF;
    else boardB_G=8'h0;
    if(iSW[14])boardB_B=8'hFF;
    else boardB_B=8'h0;
end
always@(posedge iCLK_50) //paddle for player B
begin
    if(iSW[9])boardA_R=8'hFF;
    else boardA_R=8'h0;
    if(iSW[10])boardA_G=8'hFF;
    else boardA_G=8'h0;
    if(iSW[11])boardA_B=8'hFF;
    else boardA_B=8'h0;
end
end

```

```

always@(posedge iCLK_50) //speed control of the ball
begin
    case({iSW[8],iSW[7],iSW[6]})
    0:speed=20;
    1:speed=18;
    2:speed=16;
    3:speed=15;
    4:speed=14;
    5:speed=13;
    6:speed=12;
    7:speed=10;
    default;;
    endcase
end
always@(posedge iCLK_50) //paddle size for player B
begin
    case({iSW[5],iSW[4]})
    0:sizeB=30;
    1:sizeB=25;
    2:sizeB=20;
    3:sizeB=15;
    default;;
    endcase
end
always@(posedge iCLK_50) //paddle size for player A
begin
    case({iSW[3],iSW[2]})
    0:sizeA=30;
    1:sizeA=25;
    2:sizeA=20;
    3:sizeA=15;
    default;;
    endcase
end
endmodule

module PLL(inclk0,c0);
input inclk0;
output reg c0;
parameter target=1;
reg [30:0]count;
always@(posedge inclk0)
begin
    if(count==target) count=0;
    else count=count+1;
    if(count<=target/2)
    begin
        c0=0;
    end
    else
    begin
        c0=1;
    end
end
end
endmodule

```



```

module SegOUT(in,out);
input [3:0]in;
output reg [6:0]out;
always@(in)
begin
    case(in)
        0:out=~7'b0111111;
        1:out=~7'b0000110;
        2:out=~7'b1011011;
        3:out=~7'b1001111;
        4:out=~7'b1100110;
        5:out=~7'b1101101;
        6:out=~7'b1111100;
        7:out=~7'b0000111;
        8:out=~7'b1111111;
        9:out=~7'b1100111;
        default out=~7'b0000000;
    endcase
end
endmodule

module LCD (input Clk,
            input reset,
            output reg [10:0]n_LCD_DATA,
            input [1:0]game_state
            );
reg [10:0]LCD_DATA;
reg [30:0]count;
reg [5:0]state;
reg [7:0]data[32:0];
always@ (posedge Clk or negedge reset)
begin
    if(~reset)
    begin
        count=30'd0;
        state=6'd0;
        n_LCD_DATA=11'b0;
    end
    else
    begin
        if(game_state==1)state=0;
        if(game_state==2)state=0;
        if(count==25000)
        begin
            n_LCD_DATA=11'b0;
            if(state<40)state=state+1;
            count=0;
            LCD_DATA=11'b0;
        end
        else count=count+1;
    end
    n_LCD_DATA=LCD_DATA;
end

```

```

case(state)
    6'd0:LCD_DATA=11'b000000000000;
    6'd1:LCD_DATA=11'b10000111000;
    6'd2:LCD_DATA=11'b10000001110;
    6'd3:LCD_DATA=11'b100000000001;
    6'd5:LCD_DATA=11'b10000000110;
    6'd6:LCD_DATA=11'b10010000000;
    6'd7:LCD_DATA={3'b110,data[0]};
    6'd8:LCD_DATA={3'b110,data[1]};
    6'd9:LCD_DATA={3'b110,data[2]};
    6'd10:LCD_DATA={3'b110,data[3]};
    6'd11:LCD_DATA={3'b110,data[4]};
    6'd12:LCD_DATA={3'b110,data[5]};
    6'd13:LCD_DATA={3'b110,data[6]};
    6'd14:LCD_DATA={3'b110,data[7]};
    6'd15:LCD_DATA={3'b110,data[8]};
    6'd16:LCD_DATA={3'b110,data[9]};
    6'd17:LCD_DATA={3'b110,data[10]};
    6'd18:LCD_DATA={3'b110,data[11]};
    6'd19:LCD_DATA={3'b110,data[12]};
    6'd20:LCD_DATA={3'b110,data[13]};
    6'd21:LCD_DATA={3'b110,data[14]};
    6'd22:LCD_DATA={3'b110,data[15]};
    6'd23:LCD_DATA=11'b10011000000;
    6'd24:LCD_DATA={3'b110,data[16]};
    6'd25:LCD_DATA={3'b110,data[17]};
    6'd26:LCD_DATA={3'b110,data[18]};
    6'd27:LCD_DATA={3'b110,data[19]};
    6'd28:LCD_DATA={3'b110,data[20]};
    6'd29:LCD_DATA={3'b110,data[21]};
    6'd30:LCD_DATA={3'b110,data[22]};
    6'd31:LCD_DATA={3'b110,data[23]};
    6'd32:LCD_DATA={3'b110,data[24]};
    6'd33:LCD_DATA={3'b110,data[25]};
    6'd34:LCD_DATA={3'b110,data[26]};
    6'd35:LCD_DATA={3'b110,data[27]};
    6'd36:LCD_DATA={3'b110,data[28]};
    6'd37:LCD_DATA={3'b110,data[29]};
    6'd38:LCD_DATA={3'b110,data[30]};
    6'd39:LCD_DATA={3'b110,data[31]};
    default:LCD_DATA=11'b0;
endcase
end

```

```

always@ (game_state)begin
  if(game_state==0)begin
    data[0]=8'b00100000;
    data[1]=8'b01010000;//p
    data[2]=8'b01101001;//i
    data[3]=8'b01101110;//n
    data[4]=8'b01100111;//g
    data[5]=8'b00100000;
    data[6]=8'b01010000;//p
    data[7]=8'b01101111;//o
    data[8]=8'b01101110;//n
    data[9]=8'b01100111;//g
    data[10]=8'b00100000;
    data[11]=8'b00100000;
    data[12]=8'b00100000;
    data[13]=8'b00100000;
    data[14]=8'b00100000;
    data[15]=8'b00100000;
    //
    data[16]=8'b00100000;
    data[17]=8'b00100000;
    data[18]=8'b01000111;//G
    data[19]=8'b01100001;//a
    data[20]=8'b01101101;//m
    data[21]=8'b01100101;//e
    data[22]=8'b00100000;
    data[23]=8'b00100000;
    data[24]=8'b00100000;
    data[25]=8'b00100000;
    data[26]=8'b00100000;
    data[27]=8'b00100000;
    data[28]=8'b00100000;
    data[29]=8'b00100000;
    data[30]=8'b00100000;
    data[31]=8'b00100000;
  end
end

```

```

else if(game_state==1)begin
  data[0]=8'b00100000;
  data[1]=8'b01010000;//P
  data[2]=8'b01101100;//l
  data[3]=8'b01100001;//a
  data[4]=8'b01111001;//y
  data[5]=8'b01100101;//e
  data[6]=8'b01110010;//r
  data[7]=8'b00100000;//
  data[8]=8'b00100000;//
  data[9]=8'b01000001;//A
  data[10]=8'b00100000;
  data[11]=8'b00100000;
  data[12]=8'b00100000;
  data[13]=8'b00100000;
  data[14]=8'b00100000;
  data[15]=8'b00100000;
  //
  data[16]=8'b00100000;
  data[17]=8'b00100000;
  data[18]=8'b01001001;//I
  data[19]=8'b01110011;//s
  data[20]=8'b00100000;//
  data[21]=8'b01010111;//W
  data[22]=8'b01101001;//i
  data[23]=8'b01101110;//n
  data[24]=8'b01101110;//n
  data[25]=8'b01100101;//e
  data[26]=8'b01110010;//r
  data[27]=8'b00100001;//!
  data[28]=8'b00100000;
  data[29]=8'b00100000;
  data[30]=8'b00100000;
  data[31]=8'b00100000;
end

```

```

else if(game_state==2)begin
  data[0]=8'b00100000;
  data[1]=8'b01010000;//P
  data[2]=8'b01101100;//l
  data[3]=8'b01100001;//a
  data[4]=8'b01111001;//y
  data[5]=8'b01100101;//e
  data[6]=8'b01110010;//r
  data[7]=8'b00100000;//
  data[8]=8'b00100000;//
  data[9]=8'b01000010;//B
  data[10]=8'b00100000;
  data[11]=8'b00100000;
  data[12]=8'b00100000;
  data[13]=8'b00100000;
  data[14]=8'b00100000;
  data[15]=8'b00100000;
  //
  data[16]=8'b00100000;
  data[17]=8'b00100000;
  data[18]=8'b01001001;//I
  data[19]=8'b01110011;//s
  data[20]=8'b00100000;//
  data[21]=8'b01010111;//W
  data[22]=8'b01101001;//i
  data[23]=8'b01101110;//n
  data[24]=8'b01101110;//n
  data[25]=8'b01100101;//e
  data[26]=8'b01110010;//r
  data[27]=8'b00100001;//!
  data[28]=8'b00100000;
  data[29]=8'b00100000;
  data[30]=8'b00100000;
  data[31]=8'b00100000;
end
end
endmodule

```

APPENDIX B

Pin Assignment for FPGA-based Ping-Pong Game

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
LCD_D[7]	Bidir	PIN_M5	1	B1_N2	PIN_M5
LCD_D[6]	Bidir	PIN_M3	1	B1_N1	PIN_M3
LCD_D[5]	Bidir	PIN_K2	1	B1_N1	PIN_K2
LCD_D[4]	Bidir	PIN_K1	1	B1_N1	PIN_K1
LCD_D[3]	Bidir	PIN_K7	1	B1_N1	PIN_K7
LCD_D[2]	Bidir	PIN_L2	1	B1_N2	PIN_L2
LCD_D[1]	Bidir	PIN_L1	1	B1_N2	PIN_L1
LCD_D[0]	Bidir	PIN_L3	1	B1_N1	PIN_L3
PS2_KBCLK	Bidir	PIN_G6	1	B1_N0	PIN_G6
PS2_KBDAT	Bidir	PIN_H5	1	B1_N1	PIN_H5
PS2_MSCLK	Bidir	PIN_AG25	4	B4_N1	PIN_AG25
PS2_MSDAT	Bidir	PIN_W4	2	B2_N2	PIN_W4
change[20]	Output	PIN_D2	1	B1_N0	PIN_D2
change[19]	Output	PIN_U8	2	B2_N1	PIN_U8
change[18]	Output	PIN_N25	6	B6_N2	PIN_N25
change[17]	Output	PIN_B23	7	B7_N0	PIN_B23
change[16]	Output	PIN_AE26	5	B5_N2	PIN_AE26
change[15]	Output	PIN_Y15	3	B3_N0	PIN_Y15
change[14]	Output	PIN_C24	7	B7_N0	PIN_C24
change[13]	Output	PIN_AH25	4	B4_N1	PIN_AH25
change[12]	Output	PIN_V2	2	B2_N0	PIN_V2
change[11]	Output	PIN_AE9	3	B3_N1	PIN_AE9
change[10]	Output	PIN_Y17	4	B4_N0	PIN_Y17
change[9]	Output	PIN_AF7	3	B3_N1	PIN_AF7
change[8]	Output	PIN_D21	7	B7_N0	PIN_D21
change[7]	Output	PIN_G21	7	B7_N1	PIN_G21
change[6]	Output	PIN_U6	2	B2_N1	PIN_U6
change[5]	Output	PIN_AH8	3	B3_N1	PIN_AH8
change[4]	Output	PIN_AB11	3	B3_N1	PIN_AB11
change[3]	Output	PIN_E24	7	B7_N1	PIN_E24
change[2]	Output	PIN_A21	7	B7_N1	PIN_A21
change[1]	Output	PIN_Y13	3	B3_N0	PIN_Y13
change[0]	Output	PIN_P1	1	B1_N2	PIN_P1
iCLK_28	Input	PIN_C19	7	B7_N1	PIN_C19
iCLK_50	Input	PIN_Y2	2	B2_N0	PIN_Y2
iCLK_50_2	Input	PIN_C7	8	B8_N2	PIN_C7
iCLK_50_3	Input	PIN_AC11	3	B3_N0	PIN_AC11
iCLK_50_4	Input	PIN_F14	8	B8_N0	PIN_F14
iEXT_CLOCK	Input	PIN_D13	8	B8_N0	PIN_D13

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
iKEY[3]	Input	PIN_R24	5	B5_N0	PIN_R24
iKEY[2]	Input	PIN_N21	6	B6_N2	PIN_N21
iKEY[1]	Input	PIN_M21	6	B6_N1	PIN_M21
iKEY[0]	Input	PIN_M23	6	B6_N2	PIN_M23
iSW[17]	Input	PIN_Y23	5	B5_N2	PIN_Y23
iSW[16]	Input	PIN_Y24	5	B5_N2	PIN_Y24
iSW[15]	Input	PIN_AA22	5	B5_N2	PIN_AA22
iSW[14]	Input	PIN_AA23	5	B5_N2	PIN_AA23
iSW[13]	Input	PIN_AA24	5	B5_N2	PIN_AA24
iSW[12]	Input	PIN_AB23	5	B5_N2	PIN_AB23
iSW[11]	Input	PIN_AB24	5	B5_N2	PIN_AB24
iSW[10]	Input	PIN_AC24	5	B5_N2	PIN_AC24
iSW[9]	Input	PIN_AB25	5	B5_N1	PIN_AB25
iSW[8]	Input	PIN_AC25	5	B5_N2	PIN_AC25
iSW[7]	Input	PIN_AB26	5	B5_N1	PIN_AB26
iSW[6]	Input	PIN_AD26	5	B5_N2	PIN_AD26
iSW[5]	Input	PIN_AC26	5	B5_N2	PIN_AC26
iSW[4]	Input	PIN_AB27	5	B5_N1	PIN_AB27
iSW[3]	Input	PIN_AD27	5	B5_N2	PIN_AD27
iSW[2]	Input	PIN_AC27	5	B5_N2	PIN_AC27
iSW[1]	Input	PIN_AC28	5	B5_N2	PIN_AC28
iSW[0]	Input	PIN_AB28	5	B5_N1	PIN_AB28
oHEX0_D[6]	Output	PIN_H22	6	B6_N0	PIN_H22
oHEX0_D[5]	Output	PIN_J22	6	B6_N0	PIN_J22
oHEX0_D[4]	Output	PIN_L25	6	B6_N1	PIN_L25
oHEX0_D[3]	Output	PIN_L26	6	B6_N1	PIN_L26
oHEX0_D[2]	Output	PIN_E17	7	B7_N2	PIN_E17
oHEX0_D[1]	Output	PIN_F22	7	B7_N0	PIN_F22
oHEX0_D[0]	Output	PIN_G18	7	B7_N2	PIN_G18
oHEX1_D[6]	Output	PIN_U24	5	B5_N0	PIN_U24
oHEX1_D[5]	Output	PIN_U23	5	B5_N1	PIN_U23
oHEX1_D[4]	Output	PIN_W25	5	B5_N1	PIN_W25
oHEX1_D[3]	Output	PIN_W22	5	B5_N0	PIN_W22
oHEX1_D[2]	Output	PIN_W21	5	B5_N1	PIN_W21
oHEX1_D[1]	Output	PIN_Y22	5	B5_N0	PIN_Y22
oHEX1_D[0]	Output	PIN_M24	6	B6_N2	PIN_M24
oHEX2_D[6]	Output	PIN_W28	5	B5_N1	PIN_W28
oHEX2_D[5]	Output	PIN_W27	5	B5_N1	PIN_W27
oHEX2_D[4]	Output	PIN_Y26	5	B5_N1	PIN_Y26
oHEX2_D[3]	Output	PIN_W26	5	B5_N1	PIN_W26
oHEX2_D[2]	Output	PIN_Y25	5	B5_N1	PIN_Y25
oHEX2_D[1]	Output	PIN_AA26	5	B5_N1	PIN_AA26
oHEX2_D[0]	Output	PIN_AA25	5	B5_N1	PIN_AA25

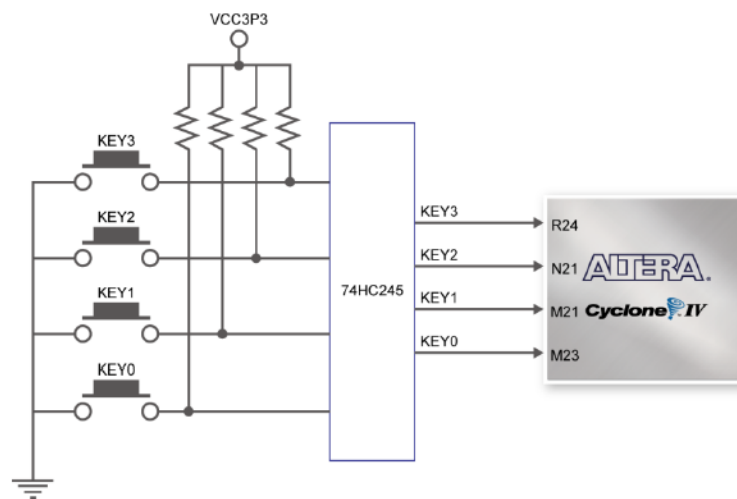
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
oHEX3_D[6]	Output	PIN_Y19	4	B4_N0	PIN_Y19
oHEX3_D[5]	Output	PIN_AF23	4	B4_N0	PIN_AF23
oHEX3_D[4]	Output	PIN_AD24	4	B4_N0	PIN_AD24
oHEX3_D[3]	Output	PIN_AA21	4	B4_N0	PIN_AA21
oHEX3_D[2]	Output	PIN_AB20	4	B4_N0	PIN_AB20
oHEX3_D[1]	Output	PIN_U21	5	B5_N0	PIN_U21
oHEX3_D[0]	Output	PIN_V21	5	B5_N1	PIN_V21
oHEX4_D[6]	Output	PIN_AE18	4	B4_N2	PIN_AE18
oHEX4_D[5]	Output	PIN_AF19	4	B4_N1	PIN_AF19
oHEX4_D[4]	Output	PIN_AE19	4	B4_N1	PIN_AE19
oHEX4_D[3]	Output	PIN_AH21	4	B4_N2	PIN_AH21
oHEX4_D[2]	Output	PIN_AG21	4	B4_N2	PIN_AG21
oHEX4_D[1]	Output	PIN_AA19	4	B4_N0	PIN_AA19
oHEX4_D[0]	Output	PIN_AB19	4	B4_N0	PIN_AB19
oHEX5_D[6]	Output	PIN_AH18	4	B4_N2	PIN_AH18
oHEX5_D[5]	Output	PIN_AF18	4	B4_N1	PIN_AF18
oHEX5_D[4]	Output	PIN_AG19	4	B4_N2	PIN_AG19
oHEX5_D[3]	Output	PIN_AH19	4	B4_N2	PIN_AH19
oHEX5_D[2]	Output	PIN_AB18	4	B4_N0	PIN_AB18
oHEX5_D[1]	Output	PIN_AC18	4	B4_N1	PIN_AC18
oHEX5_D[0]	Output	PIN_AD18	4	B4_N1	PIN_AD18
oHEX6_D[6]	Output	PIN_AC17	4	B4_N2	PIN_AC17
oHEX6_D[5]	Output	PIN_AA15	4	B4_N2	PIN_AA15
oHEX6_D[4]	Output	PIN_AB15	4	B4_N2	PIN_AB15
oHEX6_D[3]	Output	PIN_AB17	4	B4_N1	PIN_AB17
oHEX6_D[2]	Output	PIN_AA16	4	B4_N2	PIN_AA16
oHEX6_D[1]	Output	PIN_AB16	4	B4_N2	PIN_AB16
oHEX6_D[0]	Output	PIN_AA17	4	B4_N1	PIN_AA17
oHEX7_D[6]	Output	PIN_AA14	3	B3_N0	PIN_AA14
oHEX7_D[5]	Output	PIN_AG18	4	B4_N2	PIN_AG18
oHEX7_D[4]	Output	PIN_AF17	4	B4_N2	PIN_AF17
oHEX7_D[3]	Output	PIN_AH17	4	B4_N2	PIN_AH17
oHEX7_D[2]	Output	PIN_AG17	4	B4_N2	PIN_AG17
oHEX7_D[1]	Output	PIN_AE17	4	B4_N2	PIN_AE17
oHEX7_D[0]	Output	PIN_AD17	4	B4_N2	PIN_AD17
oLCD_BLON	Output	PIN_L6	1	B1_N2	PIN_L6
oLCD_EN	Output	PIN_L4	1	B1_N1	PIN_L4
oLCD_ON	Output	PIN_L5	1	B1_N1	PIN_L5
oLCD_RS	Output	PIN_M2	1	B1_N2	PIN_M2
oLCD_RW	Output	PIN_M1	1	B1_N2	PIN_M1

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
oLEDG[8]	Output	PIN_AF25	4	B4_N1	PIN_AF25
oLEDG[7]	Output	PIN_H24	6	B6_N0	PIN_H24
oLEDG[6]	Output	PIN_AH26	4	B4_N0	PIN_AH26
oLEDG[5]	Output	PIN_AE11	3	B3_N1	PIN_AE11
oLEDG[4]	Output	PIN_AA8	3	B3_N1	PIN_AA8
oLEDG[3]	Output	PIN_AF9	3	B3_N1	PIN_AF9
oLEDG[2]	Output	PIN_AC7	3	B3_N2	PIN_AC7
oLEDG[1]	Output	PIN_B22	7	B7_N1	PIN_B22
oLEDG[0]	Output	PIN_AC15	4	B4_N2	PIN_AC15
oLEDR[17]	Output	PIN_H15	7	B7_N2	PIN_H15
oLEDR[16]	Output	PIN_G16	7	B7_N2	PIN_G16
oLEDR[15]	Output	PIN_G15	7	B7_N2	PIN_G15
oLEDR[14]	Output	PIN_F15	7	B7_N2	PIN_F15
oLEDR[13]	Output	PIN_H17	7	B7_N2	PIN_H17
oLEDR[12]	Output	PIN_J16	7	B7_N2	PIN_J16
oLEDR[11]	Output	PIN_H16	7	B7_N2	PIN_H16
oLEDR[10]	Output	PIN_J15	7	B7_N2	PIN_J15
oLEDR[9]	Output	PIN_G17	7	B7_N1	PIN_G17
oLEDR[8]	Output	PIN_J17	7	B7_N2	PIN_J17
oLEDR[7]	Output	PIN_H19	7	B7_N2	PIN_H19
oLEDR[6]	Output	PIN_J19	7	B7_N2	PIN_J19
oLEDR[5]	Output	PIN_E18	7	B7_N1	PIN_E18
oLEDR[4]	Output	PIN_F18	7	B7_N1	PIN_F18
oLEDR[3]	Output	PIN_F21	7	B7_N0	PIN_F21
oLEDR[2]	Output	PIN_E19	7	B7_N0	PIN_E19
oLEDR[1]	Output	PIN_F19	7	B7_N0	PIN_F19
oLEDR[0]	Output	PIN_G19	7	B7_N2	PIN_G19
oVGA_B[7]	Output	PIN_D12	8	B8_N0	PIN_D12
oVGA_B[6]	Output	PIN_D11	8	B8_N1	PIN_D11
oVGA_B[5]	Output	PIN_C12	8	B8_N0	PIN_C12
oVGA_B[4]	Output	PIN_A11	8	B8_N0	PIN_A11
oVGA_B[3]	Output	PIN_B11	8	B8_N0	PIN_B11
oVGA_B[2]	Output	PIN_C11	8	B8_N1	PIN_C11
oVGA_B[1]	Output	PIN_A10	8	B8_N0	PIN_A10
oVGA_B[0]	Output	PIN_B10	8	B8_N0	PIN_B10
oVGA_BLANK	Output	PIN_F11	8	B8_N1	PIN_F11
oVGA_CLOCK	Output	PIN_A12	8	B8_N0	PIN_A12

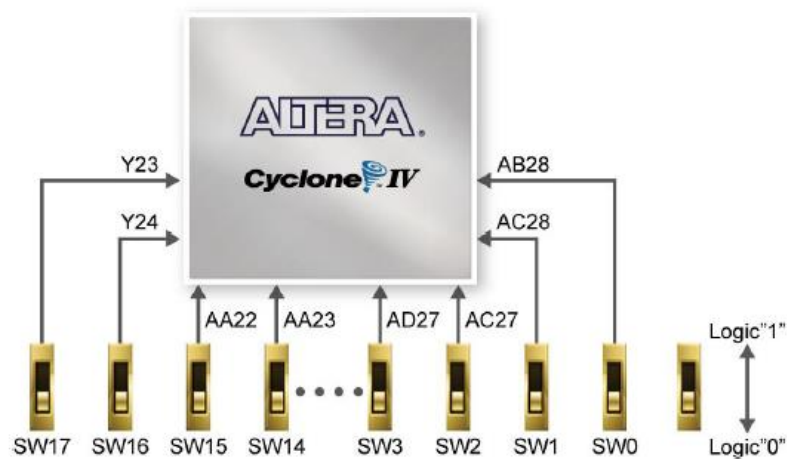
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
oVGA_G[7]	Output	PIN_C9	8	B8_N1	PIN_C9
oVGA_G[6]	Output	PIN_F10	8	B8_N1	PIN_F10
oVGA_G[5]	Output	PIN_B8	8	B8_N1	PIN_B8
oVGA_G[4]	Output	PIN_C8	8	B8_N1	PIN_C8
oVGA_G[3]	Output	PIN_H12	8	B8_N1	PIN_H12
oVGA_G[2]	Output	PIN_F8	8	B8_N2	PIN_F8
oVGA_G[1]	Output	PIN_G11	8	B8_N1	PIN_G11
oVGA_G[0]	Output	PIN_G8	8	B8_N2	PIN_G8
oVGA_HS	Output	PIN_G13	8	B8_N0	PIN_G13
oVGA_R[7]	Output	PIN_H10	8	B8_N1	PIN_H10
oVGA_R[6]	Output	PIN_H8	8	B8_N2	PIN_H8
oVGA_R[5]	Output	PIN_J12	8	B8_N0	PIN_J12
oVGA_R[4]	Output	PIN_G10	8	B8_N1	PIN_G10
oVGA_R[3]	Output	PIN_F12	8	B8_N1	PIN_F12
oVGA_R[2]	Output	PIN_D10	8	B8_N1	PIN_D10
oVGA_R[1]	Output	PIN_E11	8	B8_N1	PIN_E11
oVGA_R[0]	Output	PIN_E12	8	B8_N1	PIN_E12
oVGA_SYNC_N	Output	PIN_C10	8	B8_N0	PIN_C10
oVGA_VS	Output	PIN_C13	8	B8_N0	PIN_C13
ostate[1]	Output	PIN_L8	1	B1_N2	PIN_L8
ostate[0]	Output	PIN_M7	1	B1_N2	PIN_M7

APPENDIX C

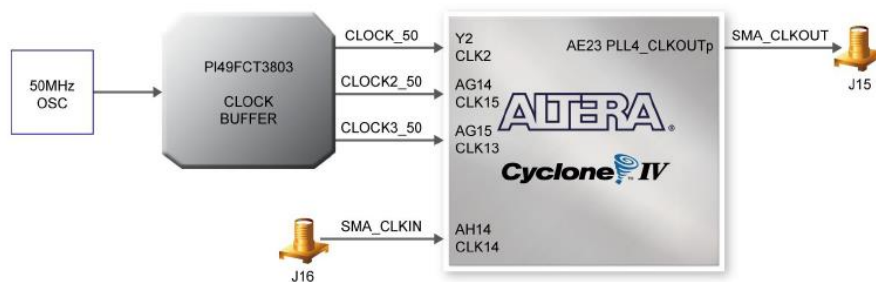
Connections between the pushbutton and Cyclone IV E FPGA



Connections between the slide switches and Cyclone IV E FPGA



Block diagram of the clock distribution



APPENDIX D

Pin Assignments for Push Buttons

Signal Name	FPGA Pin No.	Description	I/O Standard
KEY[0]	PIN_M23	Push-button[0]	Depending on JP7
KEY[1]	PIN_M21	Push-button[1]	Depending on JP7
KEY[2]	PIN_N21	Push-button[2]	Depending on JP7
KEY[3]	PIN_R24	Push-button[3]	Depending on JP7

Pin Assignments for Slide Switches

Signal Name	FPGA Pin No.	Description	I/O Standard
SW[0]	PIN_AB28	Slide Switch[0]	Depending on JP7
SW[1]	PIN_AC28	Slide Switch[1]	Depending on JP7
SW[2]	PIN_AC27	Slide Switch[2]	Depending on JP7
SW[3]	PIN_AD27	Slide Switch[3]	Depending on JP7
SW[4]	PIN_AB27	Slide Switch[4]	Depending on JP7
SW[5]	PIN_AC26	Slide Switch[5]	Depending on JP7
SW[6]	PIN_AD26	Slide Switch[6]	Depending on JP7
SW[7]	PIN_AB26	Slide Switch[7]	Depending on JP7
SW[8]	PIN_AC25	Slide Switch[8]	Depending on JP7
SW[9]	PIN_AB25	Slide Switch[9]	Depending on JP7
SW[10]	PIN_AC24	Slide Switch[10]	Depending on JP7
SW[11]	PIN_AB24	Slide Switch[11]	Depending on JP7
SW[12]	PIN_AB23	Slide Switch[12]	Depending on JP7
SW[13]	PIN_AA24	Slide Switch[13]	Depending on JP7
SW[14]	PIN_AA23	Slide Switch[14]	Depending on JP7
SW[15]	PIN_AA22	Slide Switch[15]	Depending on JP7
SW[16]	PIN_Y24	Slide Switch[16]	Depending on JP7
SW[17]	PIN_Y23	Slide Switch[17]	Depending on JP7

Pin Assignments for Clock Inputs Signal

Name	FPGA Pin No.	Description	I/O Standard
CLOCK_50	PIN_Y2	50 MHz clock input	3.3V
CLOCK2_50	PIN_AG14	50 MHz clock input	3.3V
CLOCK3_50	PIN_AG15	50 MHz clock input	Depending on JP6
SMA_CLKOUT	PIN_AE23	External (SMA) clock output	Depending on JP6
SMA_CLKIN	PIN_AH14	External (SMA) clock input	3.3V

Pin Assignments for ADV7123 Signal

Name	FPGA Pin No.	Description	I/O Standard
VGA_R[0]	PIN_E12	VGA Red[0]	3.3V
VGA_R[1]	PIN_E11	VGA Red[1]	3.3V
VGA_R[2]	PIN_D10	VGA Red[2]	3.3V
VGA_R[3]	PIN_F12	VGA Red[3]	3.3V
VGA_R[4]	PIN_G10	VGA Red[4]	3.3V
VGA_R[5]	PIN_J12	VGA Red[5]	3.3V
VGA_R[6]	PIN_H8	VGA Red[6]	3.3V
VGA_R[7]	PIN_H10	VGA Red[7]	3.3V
VGA_G[0]	PIN_G8	VGA Green[0]	3.3V
VGA_G[1]	PIN_G11	VGA Green[1]	3.3V
VGA_G[2]	PIN_F8	VGA Green[2]	3.3V
VGA_G[3]	PIN_H12	VGA Green[3]	3.3V
VGA_G[4]	PIN_C8	VGA Green[4]	3.3V
VGA_G[5]	PIN_B8	VGA Green[5]	3.3V
VGA_G[6]	PIN_F10	VGA Green[6]	3.3V
VGA_G[7]	PIN_C9	VGA Green[7]	3.3V
VGA_B[0]	PIN_B10	VGA Blue[0]	3.3V
VGA_B[1]	PIN_A10	VGA Blue[1]	3.3V
VGA_B[2]	PIN_C11	VGA Blue[2]	3.3V
VGA_B[3]	PIN_B11	VGA Blue[3]	3.3V
VGA_B[4]	PIN_A11	VGA Blue[4]	3.3V
VGA_B[5]	PIN_C12	VGA Blue[5]	3.3V
VGA_B[6]	PIN_D11	VGA Blue[6]	3.3V
VGA_B[7]	PIN_D12	VGA Blue[7]	3.3V
VGA_CLK	PIN_A12	VGA Clock	3.3V
VGA_BLANK_N	PIN_F11	VGA BLANK	3.3V
VGA_HS	PIN_G13	VGA H_SYNC	3.3V
VGA_VS	PIN_C13	VGA V_SYNC	3.3V
VGA_SYNC_N	PIN_C10	VGA SYNC	3.3V

APPENDIX E

Features of DE2-115

Features	Characteristics
FPGA Device	Cyclone IV EP4CE115F29 device 114,480 LEs 432 M9K memory blocks 3,888 Kbits embedded memory 4 PLLs
FPGA Configuration	JTAG and AS mode configuration EPCS64 serial configuration device On-board USB Blaster circuitry
Memory Device	128MB (32Mx32bit) SDRAM 2MB (1Mx16) SRAM 8MB (4Mx16) Flash with 8-bit mode 32Kb EEPROM
SD Card Socket	Provides SPI and 4-bit SD mode for SD Card access
Connectors	Two Ethernet 10/100/1000 Mbps ports High Speed Mezzanine Card (HSMC) Configurable I/O standards (voltage levels:3.3/2.5/1.8/1.5V) USB type A and B -Provide host and device controllers compliant with USB 2.0 -Support data transfer at full-speed and low-speed -PC driver available 40-pin expansion port -Configurable I/O standards (voltage levels:3.3/2.5/1.8/1.5V) VGA-out connector -VGA DAC (high speed triple DACs) DB9 serial connector for RS-232 port with flow control PS/2 mouse/keyboard
Clock	Three 50MHz oscillator clock inputs SMA connectors (external clock input/output)
Audio	24-bit encoder/decoder (CODEC) Line-in, line-out, and microphone-in jacks
Display	16x2 LCD module
Switches and indicators	18 slide switches and 4 push-buttons switches 18 red and 9 green LEDs Eight 7-segment displays
Other features	Infrared remote-control receiver module TV decoder (NTSC/PAL/SECAM) and TV-in connector