DUCK HUNT FPGA GAME, A PROJECT ON
UML AND DIGITAL DESIGN

A PROJECT REPORT

Presented to the Department of Computer Engineering & Computer Science
California State University, Long Beach

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science
Option in Computer Engineering

Committee Members:
Burkhard Englert, Ph.D. (Chair)
John Tramel, M.S.
Michael Chelian, Ph.D.

College Designee:
Antonella Sciortino, Ph.D.

By Vuong D. Nguyen
B.S., 2011, California State University, Long Beach
August 2016
ABSTRACT

DUCK HUNT FPGA GAME, A PROJECT ON
UML AND DIGITAL DESIGN

By

Vuong D. Nguyen

August 2016

Field Programmable Gate Array (FPGA) is rarely associated with video games. Software video games can be made using the Unified Modeling Language (UML) and high level languages such as the Extensible Hypertext Markup Language (XHTML), Cascading Style Sheets (CSS), Javascript, and jQuery; however, FPGA video games require the building of complex hardware. The goal of this project is to create an FPGA video game by combining UML and digital design.

There are advantages to starting from the hardware level such as having more control, thus giving more freedom to create design and functional specifications. The disadvantages include creating device drivers. By using the Rational Unified Process (RUP) as the development process, a Duck Hunt FPGA Game is created that proves how software video game development is different compared to FPGA game development.
# TABLE OF CONTENTS

ABSTRACT.................................................................................................................................. ii  
LIST OF TABLES........................................................................................................................ v  
LIST OF FIGURES ...................................................................................................................... vi  

1. INTRODUCTION ............................................................................................................ 1  
2. UML AND OBJECT ORIENTED DESIGN .................................................................... 2  
3. LITERATURE REVIEW ................................................................................................. 4  
4. APPLYING UML TO DIGITAL DESIGN ...................................................................... 6  
5. WHAT IS A VIDEO GAME? .......................................................................................... 8  
6. APPARATUS AND EXPERIMENTAL SETUP ............................................................. 10  
7. BASIC ORGANIZATION ............................................................................................... 11  
8. MAIN CONTROLLER .................................................................................................... 13  
9. PRESENTATION OF THE DUCK HUNT FPGA GAME ............................................. 14  
10. RENDER PIPELINE .................................................................................................... 20  
11. COMMUNICATION BETWEEN OBJECTS .................................................................. 22  
12. GUN OBJECT .................................................................................................................. 23  
13. FLYING DUCK GAME MODULE ................................................................................. 28  
14. TURN BASED GAME ..................................................................................................... 31  
15. SCORE MANIPULATION .............................................................................................. 32  
16. SPEED OF FLYING DUCK ............................................................................................ 34  
17. EXPERIMENTAL PROCEDURE ................................................................................... 36  
18. RESULTS AND DISCUSSION ....................................................................................... 39  
19. CONCLUSION ................................................................................................................. 45
### LIST OF TABLES

1. Hardware and Software Tools ........................................................................................................ 10
2. Software Tool Requirements ........................................................................................................... 36
3. Functional Requirements ................................................................................................................. 36
4. Hardware Requirements .................................................................................................................. 37
5. Performance Requirements ............................................................................................................. 37
6. VGA Signal Combination .................................................................................................................. 44
LIST OF FIGURES

1. Use case diagram for the Duck Hunt FPGA Game ......................................................... 2
2. The top level schematic of the devices and their connections ........................................ 10
3. Relationship between finite state machine and datapath .............................................. 11
4. Top level view ................................................................................................................ 12
5. Device driver simplifies the interfacing between the FPGA and analog devices ............. 14
6. Preproduction image tile of the number 2 .................................................................. 15
7. Production image tile of the number 2 ......................................................................... 16
8. Preproduction image tile of a flying duck .................................................................... 17
9. Level 1 of the render pipeline ....................................................................................... 20
10. Level 2 of the render pipeline .................................................................................... 21
11. The PS/2 Mouse Interface, which is a device driver, shields the Main Controller from the complex details of data transmission and receipt ............................................ 24
12. Data format for the PS/2 protocol ................................................................................ 24
13. The PS/2 Mouse Device treats the center as the origin .............................................. 26
14. The VGA standard treats the top, left corner as the origin ........................................ 26
15. Internal details of flying duck datapath ....................................................................... 29
16. Player turn and score manipulation are handled by the Player Turn and Profile Database module ................................................................................................................. 32
17. Timer pipeline determines the speed of flying ducks .................................................. 34
CHAPTER 1
INTRODUCTION

The Duck Hunt Game used to be a popular video game for the Nintendo Entertainment System. The goal is to shoot down flying ducks. The game has single and multiplayer modes. Every time an active player accomplishes the goal, the screen updates the active player’s score. The challenge of shooting down flying ducks is compounded when ducks are flying at different speeds.

The goal of this project is to create the Duck Hunt Game on an FPGA prototyping board, specifically the Spartan 3 FPGA Prototyping Board. The Duck Hunt Game is chosen because it presents an opportunity to learn about FPGA, graphics, video games, and device drivers. The Duck Hunt Game is implemented on a FPGA prototyping board because the goal is to compare how FPGA video game development is different from software video game development. The FPGA game is implemented using the Verilog Hardware Description Language (Verilog HDL).

Delivering a complete and fully working Duck Hunt FPGA Game means that all development lifecycle stages have to be met. This project has to elicit requirements, parse the requirements to reveal game objects, attributes, as well as behavior, design an architecture, code the architecture, and perform the unit and integration testing.

The remainder of this report is broken down into sections for UML and object oriented design, literature review, proof that digital systems possess the characteristics of object oriented design, devices used to create the game, experimental setup, experimental procedures, results and discussion, what was learned, and errors that occurred during the development of this game.
CHAPTER 2

UML AND OBJECT ORIENTED DESIGN

According to H. Deitel and P. Deitel, UML is a “graphical language that allows people who design object-oriented software systems to use an industry-standard notation to represent them” [1]. UML is composed of several diagrams. The commonly used diagrams include use case, state, sequence, and class. Figure 1 shows the use case diagram of this project.

FIGURE 1. Use case diagram for the Duck Hunt FPGA Game.

2.1 Objects, Attributes, Behavior, and Interrelationship

In the physical world, there are objects. Objects have attributes and behaviors. The goal of object oriented design, as argued by H. Deitel and P. Deitel, is to model objects by looking at their attributes, behavior, and interrelationships [1]. Attributes are adjectives that describe the object, while verbs describe the behavior of an object.
2.2 Key Characteristics of Object Oriented Design

H. Deitel and P. Deitel argue that all object oriented software systems possess three characteristics [1]. They include encapsulation, information hiding, and well-defined interfaces. Attributes and behavior that are related are wrapped together into an object, which is called encapsulation. If two objects want to communicate with each other, the best approach is to use the well-defined interface. Communication by using the well-defined interface is important to object oriented design. The well-defined interface hides how the object is implemented. Object A can communicate with Object B despite Object A not knowing how Object B is implemented. Forcing all the objects to communicate by using the well-defined interface promotes information hiding. H. Deitel and P. Deitel use driving a car as an example. A human driver can control the car without having to understanding how the engine works or how the exhaust system works [1].
CHAPTER 3
LITERATURE REVIEW

When applying UML to digital design, researchers primarily focused on Extensible Markup Language (XML), Metadata Interchange (XMI), and UML profile. Z. Sun et al. used the XMI approach to design a clocked system, which was a digital down converter [2]. UML diagrams were created with the help of an UML editor, and the UML diagrams were then translated into XMI. The XMI was then inputted into a translator, which generated the source code. The SystemC source code can then be synthesized.

XMI was only used to design but not implement the clocked system. This meant that the XMI approach stayed at an abstract level and did not touch the programming constructs of SystemC.

In another scenario, G. Bazydlo et al. combined the UML language and digital design to create a logic controller [3]. The logic controller was created by using only the state diagram, an UML editor was used to create the state diagram. After the state diagram had been created, the export option of the UML editor was used to create a XMI file. The next step included the translation step. The XMI file was inputted into U2V. This tool translated the XMI file into a Hardware Description Language (HDL) implementation, specifically SystemC. The Hierarchical Concurrent Finite State Machine (HCFSMs) were used to translate the XMI into HDL description.

G. Martin and W. Muller used UML profiles to design a digital system [4]. A profile includes adding extensions or refining an existing UML metamodel, such as the class diagram. One example includes the Schedulability, Performance, and Timing Analysis (SPT) Profile. This profile is used for modeling timing and performance characteristics. For, example, the SPT
profile can be used to analyze a system, such as the Rate Monotonic Analysis (RMA) and Deadline Monotonic Analysis (DMA)

The problem with the profile approach is adoption and standardization. A profile is only useful if it is currently in the Computer-Aided Software Engineering (CASE) tool. The profile also has to be consistent across all proprietary toolsets. It will take time before commercial vendors decide to include profiles into their CASE tools.

Commercial vendors update their tools with new profiles only when a new version of the UML specification is released. For example, the SPT profile will be standardized upon the release of the UML 2 specification. Updating CASE tools with a larger set of metamodels is enough to entice commercial vendors to incorporate the smaller SPT profile.

The System on Chip (SoC) profile is dedicated to digital design and FPGA. The UML for SoC Forum (USocF) initiated the creation of this profile. This profile added a new diagram called the SoC structure diagram, which adds extensions to the UML specification.

The UML profile predicts the future of UML and digital design. Meanwhile, this project has been developed without profiles, and only basic UML diagrams are used. The basic UML diagrams have already been standardized, thus creating a lack of waiting for commercial vendors to update their CASE tools. This project is independent of any new extensions, which have not yet been standardized.

A hybrid approach is used to develop a FPGA video game. For the design of the FPGA video game, this project analyzes how and why UML can be used to design and document digital systems. For the implementation, testing, and integration, Verilog HDL is used. This project, however, does not discuss enhancing UML so that it can show the timing of digital systems.
CHAPTER 4

APPLYING UML TO DIGITAL DESIGN

UML is a design language used to design object oriented software systems. The goal of this project is to design the Duck Hunt FPGA Game using the syntax of UML. In order to design digital systems using UML syntax, this project must first prove that digital systems possess the characteristics of object oriented design.

The following sections prove that digital systems possess the characteristics of object oriented design. The proof is composed of stating how the programming constructs of Verilog HDL can implement objects, attributes, behavior, encapsulation, a well-defined interface, and information hiding.

4.1 Implementing Attributes and Behavior

Verilog HDL has the register data type. Data is loaded into the register data type; each register can be an attribute. Verilog HDL has sequential and combinational blocks, logic gates, as well as arithmetic and logical operators. These features can be combined to create a state machine. A digital system can be decomposed into two modules called finite state machine and datapath. The datapath performs computations based on the current state of the finite state machine and system inputs. Most digital systems are governed by state based behavior.

4.2 The Unit of Programming

In Verilog HDL, a module is the unit of programming. Users can instantiate Verilog HDL modules by giving the instance of the module a name and initialize the instance by using a comma separated list of values. Verilog HDL also has built in types that are used by FPGA developers to define the internal details of a module. An FPGA developer can define the internal details of a module based on how he or she desires. Thus, a module is a user defined data type.
4.3 Implementing Encapsulation, Well-Defined Interface, and Information Hiding

In Verilog HDL, registers and behavior that are related are combined into a module that implements encapsulation. Verilog modules become objects of the system. Every Verilog HDL module has one or more ports. Each port can be in one of three modes including input, output, and bidirection. Verilog HDL modules communicate by using these ports. Either data is being read from the input port when receiving data or is being placed into the output port when sending data. Internal registers cannot be directly manipulated, but can only be manipulated indirectly by using the ports. The port’s mode indicates what behaviors are possible. The ports and port mode satisfy the characteristic of a well-defined interface. As a result of hiding the internal implementation, modules can communicate without having to know how other modules are implemented. The well-defined interface implements information hiding.

4.4 Object Communication

In Verilog HDL, one module can send messages to other modules. The wire data type is used to carry messages from a source port to a destination port. Two modules must have wire connections to each other in order to communicate. A module that wants to send messages must have a wire connected to the output port while a module that wants to receive messages must have a wire connected to the input port.
CHAPTER 5

WHAT IS A VIDEO GAME?

There are many ways to define video game. S. Rogers states that a video game “is a game played on a video screen” [5]. This is a simple yet powerful definition because the word game is used in the definition. What is a game? S. Rogers defines a game as “an activity that requires at least one person, it must have rules, and it has a victory condition” [5]. The victory condition is the objective. There is also a failure condition, which occurs when a player violates the rules or victory condition.

5.1 Edutainment Versus Educational Games

There are two types of video games, edutainment and educational. M. J. Dondlinger states that an educational video games emphasizes “strategizing, hypothesis testing, or problem solving, usually with higher order thinking rather than rote memorization or simple comprehension” [6]. This is the opposite of edutainment video games, which M. J. Dondlinger argues, emphasizes skill and drill [6]. Skill and drill video games require players to “practice repetitive skills or rehearse memorized facts”.

The Duck Hunt FPGA Game is a hybrid video game. The objective of the Duck Hunt FPGA Game is to shoot down flying ducks. The active player repeats this task multiple times. This makes the Duck Hunt FPGA Game an edutainment game. However, there is a bit of strategizing. Not all ducks fly at the same speed, so the active player must strategize which type of duck to shoot down first. If the active player focuses on the slow ducks, then the fast ducks will escape. However, if the active player focuses on the fast ducks, then the slow ducks will be missed. It should be noted that all ducks are of equal value. The reward of shooting down a duck is one point. A fast flying duck is of equal value as a slow flying duck.
5.2 Event Driven Programming

Video games utilize event driven programming, which J. E. Carryer argues, is needed when a system must “respond to a multitude of possible inputs” [7]. Furthermore, these inputs “may arrive at unpredictable times and in an unpredictable sequence.”

Event driven programming structures the program into events and services. J. E. Carryer states that an event “represents the occurrence of something interesting” while a service is “what you do in response to an event” [7]. The two most important characteristics of event driven programming are to continuously check for events and to have compact services. Polling is the most common approach to continuously check for events. A compact service performs the required action and returns. This allows the program to get back to checking for other events, so that none are missed.

A video game possesses the characteristics of event driven programming. There are events, services, and inputs. The inputs arrive at unpredictable times and in an unpredictable sequence. One example includes when the active player sees multiple flying ducks. A player may or may not decide to press the trigger button. The player might focus on the slow ducks for one round and the fast ducks in the next round. Thus, the timing and the sequence of the inputs are unpredictable.

Assuming that the active player sees a duck and shoots it by pressing one of the buttons of the PS/2 Mouse Device, the video game would then respond to this event by consulting with the body shot algorithm to determine the accuracy of the shot. The body shot algorithm, which is the service, executes and quickly returns so that the game can go back to checking for events.
CHAPTER 6
APPARATUS AND EXPERIMENTAL SETUP

This Duck Hunt FPGA Game was designed using UML and implemented using Verilog HDL. This hybrid approach means that both software and hardware tools are used during the development of this project. Table 1 lists the hardware components and software tools that are used to create the Duck Hunt FPGA Game. Figure 1 shows the basic setup between an HP Laptop, the Spartan 3 FPGA Prototyping Board, an Acer VGA Monitor, and a PS/2 Mouse Device.

TABLE 1. Hardware and Software Tools

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Digilent Spartan 3 FPGA with AC Adapter</td>
<td>1. Xilinx ISE Design Suite 14.6</td>
</tr>
<tr>
<td>2. HP Optical PS/2 Mouse</td>
<td>2. Digilent Adept</td>
</tr>
<tr>
<td>3. HP Elitebook 8540p Laptop</td>
<td>3. Digilent JTAG HS1 Programming Cable</td>
</tr>
<tr>
<td>4. Acer AL1716W VGA Monitor</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 2. The top level schematic of the devices and their connections.
CHAPTER 7

BASIC ORGANIZATION

The game was designed with respect to the subsystem and top level view. The subsystem view organizes modules based on a Finite State Machine with Datapath (FSMD). The top level view organizes modules and subsystems as computing blocks coordinated by a module called Main Controller.

7.1 Subsystem View

The subsystem view, which is shown in Figure 3, is composed of two types of modules. A Finite State Machine (FSM) acts as a sequencer, or controller. The Datapath (DP) is the executor and contains one or many function units. A function unit computes an output by using inputs as operands. The resulting computation could cause certain conditions to arise. Status flags are created for each condition. The DP gives the status flags to the FSM. The FSM transitions to a new state based on the current state of the status flags, system input, and current state of the FSM.

FIGURE 3. Relationship between finite state machine and datapath.

7.2 Top Level View

The top level view, which is shown in Figure 4, shows the hierarchy and overall communication scheme. The Main Controller is in charge of coordinating all modules and
subsystems. All modules and subsystems must wait for the Main Controller to assert the enable signal. Once this enable signal is asserted, modules and subsystems can resume processing. The Main Controller must wait for all the modules and subsystems to complete processing before continuing on with the next round of processing. In this case, the next round of processing is giving the idled player his or her turn.

FIGURE 4. Top level view. The rendering subsystem is not shown.
CHAPTER 8

MAIN CONTROLLER

When the active player is ready, he or she presses the enable button. The Main Controller then asserts the enable signal of each game module and subsystem, which tells the game modules and subsystems to resume processing. The Main Controller only coordinates the game modules and subsystems. As a result, the Main Controller is compact. The Main Controller is composed of four states, which include reset, wait for all modules ready, idle, and play game.
CHAPTER 9

PRESENTATION OF THE DUCK HUNT FPGA GAME

The presentation of the game is more involved because this project’s perspective is from the hardware level. Presenting the game requires creating device drivers, image tiles, background rom, assigning a game object id, setting boundaries, determining if the Video Graphics Array (VGA) scan is within the boundaries of a game object, and implementing animation.

9.1 Device Driver for VGA Monitor

The device driver for a VGA monitor, called the VGA Controller, manages timing and color. The VGA standard states that the pixel refresh rate must be 25.17 Mhz, and the frame refresh rate must be 60 Hz. The VGA port on the Spartan 3 FPGA Prototyping Board has three pins for color. The three pins represent red, green, and blue. The Spartan 3 FPGA Prototyping Board’s clock frequency is 50 Mhz. The project uses a mod-2 counter to satisfy the pixel refresh rate and a mod 833,333 counter to satisfy the frame refresh rate. Figure 5 shows the connection between the VGA monitor and the VGA Controller.

A VGA Controller also has row and column counters. The row count indicates the current location of the row refresh scan while the column count indicates the current location of the column refresh scan. The current row and column count are used to render game objects.
row counter module is called the Hsync Controller while the column counter module is called the Vsync Controller.

9.2 Image File

At the hardware level, there are no predefined image files. In order to render game objects, the FPGA developer has to manually create the image files. There are many ways to create image files, but a commonly used way is image tiles.

9.2.1 Tile Rom

An image tile, or raster graphics, is a two dimensional matrix that stores color. Each x and y coordinate pair represents a pixel, and a pattern of pixels creates an image. The preproduction image tile for number two is shown in Figure 6. The preproduction schematic is an example to illustrate the pattern that makes up number two. The image tile that is used in the game is shown in Figure 7. All game objects that make up the Graphical User Interface (GUI) of the video game have their own image tile. Image tiles are stored in a tile rom. The preproduction image tiles of a duck is shown in Figure 8.

![Image Tile Example](image.png)

FIGURE 6. Preproduction image tile of the number 2.
9.3 Game Object Square Block

Each image tile is a two dimensional matrix, which is a square or rectangular block. Every game object has a corresponding image tile. Since each image tile is a square or rectangular block, each game object has a boundary. The top and bottom boundaries represent the row boundary while the left and right boundaries represent the column boundary. The top and left boundaries represent the current position of the game object. Image tiles, as well as their square boundaries, are used to render game objects and target ducks. The following paragraphs explain how rendering works.

Another challenge encountered in this project are game objects and image tiles. If a game object, such as a flying duck, changes position, the duck image tile must be rendered at this new position, else the active player will be confused. To account for changing positions, the FPGA developer has to associate a game object with the correct image tile.
### FIGURE 8. Preproduction image tile of a flying duck.

#### 9.4 Game Object Id

An identification number associates a game object with an image tile. Every game object is assigned an identification number, and multiple instances of the same game object have the same identification number. For example, all flying ducks have the same identification number. The identification number is an offset into the Tile Rom, which contains all of the image tiles.

The next step of development is to combine the Tile Rom, the game object boundary, and the game id. Player One is used as an example. Player One’s game object id is eleven. This id is an input into the module called Tile Rom Offset Generator. Based on the id, the output of Tile Rom Offset Generator is an address. For Player One, the game object id of eleven results in an address of 165. This address is an input into the module called Tile Pixel Calculation. The Tile Rom module is a subsystem of the module Tile Pixel Calculation. The address of 165 is the location of a blue encircled tile. This means that Player One’s gun crosshair is a blue circle.

#### 9.5 VGA Scan

The VGA Controller scans the entire VGA monitor. The scan is performed with respect to the rows of the VGA monitor. The first row is scanned, and each row is composed of columns.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For the VGA standard, a row is composed of 640 columns. Scanning a row means starting at column zero and incrementing the column count. When the VGA Controller finishes scanning a row, the row count is incremented, and the column count is reset to zero. The next row is scanned and the scanning will continue until all the rows and columns have been scanned.

**9.6 Within Boundary Calculation**

The first step in rendering a game object is to check if the VGA scan is within the boundary of a game object. Recall that all the game objects are represented by a square block, which is defined by row and column boundaries. If the VGA row and column coordinates are within the square block, the scan is within bound.

If the VGA scan is within the square block of a game object, the game object’s identification number is used as the offset to the Tile Rom module. All game objects have an identification number, and this identification number is a tile offset that is added to the rom row offset.

**9.7 Background Rom**

The game has a foreground and background. Game objects that change position make up the foreground. This means that the foreground is composed of image tiles. The image tiles are stored in a tile rom. Unlike the foreground, the background is composed of game objects that do not change position, and therefore the background is not composed of image tiles. The background has its own rom called the Background Rom. The Background Rom is not a two dimensional array. Instead, the Background Rom is a one dimensional array. Each address in the Background Rom has a color. When the addresses within the Background Rom are linked together, a pattern manifests itself. The pattern creates landscape images that are two clouds and letters indicating the current score of Player One and Two.
9.8 Animation

The VGA Controller must rescan in order to create animation. A game object such as a duck will change its current position. If the VGA Controller does not scan a second time, this change will not be rendered onto the VGA monitor. In order to render all of the changes, the VGA scan must run continuously. The VGA Controller is a free running module. In order to have animation, the positioning of the game object only needs to be updated at a frequency of 60 Hz, which is the frame refresh rate. The VGA Controller is a free running module, but internal registers are only updated at a frequency of 60 Hz.
CHAPTER 10

RENDER PIPELINE

A render pipeline simplifies the rendering of the foreground and the background game objects. The render pipeline is a two level pipeline, and each level has a multiplexer. The first level selects one game object and an image tile to render in the foreground. The first level, which is shown in Figure 9, is composed of the modules Tile Rom Offset Generator and Tile Pixel Calculation.

![Figure 9. Level 1 of the render pipeline.](image-url)
The second level, which is shown in Figure 10, selects between the foreground and the background. The chosen color information is given to the VGA Controller, which then displays the color information onto the screen. The foreground color information has priority over the background color information.

**FIGURE 10.** Level 2 of the render pipeline.
CHAPTER 11

COMMUNICATION BETWEEN OBJECTS

The Main Controller, modules, and subsystems interact using a simple communication scheme. Status flags communicate that certain conditions have occurred. There are three basic status flags including Ready, Dead, and Level Done.

Each module and subsystem individually generate these status flags. The values of the status flags are based on the result of a computation. The Ready flag is calculated by applying the bitwise AND operator of all of the individual ready flags included in the modules and subsystems. The bitwise AND operator is also used to calculate the value of the Dead and Level Done flags.
CHAPTER 12

GUN OBJECT

In order to target and shoot down flying ducks, the active player must have a virtual gun. The Gun Game Object, Personal System/2 (PS/2) Mouse Interface, and PS/2 Mouse Device, when combined, create the Gun Object module. The PS/2 Mouse Device sends movement packets to the Spartan 3 FPGA Prototyping Board. The data within the movement packets represent the button status and displacement values. Before displacement values are added to the current position of the gun, the algorithm must transform the displacement values from a math origin to a VGA standard origin.

12.1 PS/2 Mouse Interface

For this project, the PS/2 Mouse Device is the virtual gun. At the hardware level, there is not a device driver for the PS/2 Mouse Device. In order for the FPGA to communicate with the PS/2 Mouse Device, the FPGA developer has to write a device driver called the PS/2 Mouse Interface. Without the device driver, the active player must manually operate the clock and data bus of the PS/2 Mouse Device. The PS/2 Mouse Interface prevents the active player from having to learn the internal details of the PS/2 Mouse Device. The PS/2 Mouse Interface operates the clock and data bus on the active player’s behalf. The PS/2 Mouse Interface takes care of the initialization, data transmission, and the receipt on the player’s behalf. Figure 11 shows the connection between the PS/2 Mouse Device and PS/2 Mouse Interface.

The PS/2 Mouse Interface module is responsible for confirming that the PS/2 Mouse Device is in stream mode. The PS/2 Mouse Interface module must first send the reset command to the PS/2 Mouse Device, which then will put the PS/2 Mouse Device in stream mode. Next, the PS/2 Mouse Interface waits for the incoming movement packets from the PS/2 Mouse Device.
The PS/2 Mouse Interface module is composed of Tx and Rx modules. This composition gives the PS/2 Mouse Interface the ability to send commands and receive movement packets. The Tx module uses a bidirectional data bus because the Tx module must send a command and then wait for an acknowledge bit. The Tx module sends and receives data, thus, a bidirectional bus is needed. The Rx module, however, only receives data, thus, the Rx module does not need a bidirectional data bus.

![Diagram of PS/2 Mouse Interface](image)

**FIGURE 11.** The PS/2 Mouse Interface, which is a device driver, shields the Main Controller from the complex details of data transmission and receipt.

Since data transmission is serial, both modules operate on the bits differently. Each data packet, which is shown in Figure 12 and explained by A. Chapweske, has the format of one start bit, eight data bits, one odd parity bit, and one stop bit [8]. The Rx module shifts in the received bit to form a byte while the Tx module shifts out the data bit over the data bus.

![Data format for PS/2 protocol](image)

**FIGURE 12.** Data format for the PS/2 protocol.

12.2 Gun Game Object Module

The Gun Game Object module is an instance of the Game Object module. The Game Object module has two attributes called the top and left boundary that represent the current
location of the module. Since this chapter is on the Gun Game Object, the top and left boundaries represent the current location of the gun crosshair. The top and left boundaries represent the y and x coordinates in a two dimensional matrix, respectively.

Any time the PS/2 Mouse Device senses movement, it will send movement packets to the FPGA. The PS/2 Mouse Interface module gathers the bits together into a byte. The movement packet sent by the PS/2 Mouse Device is composed of three bytes of data. The three bytes indicate the button status, x movement, and y movement. The x and y movement bytes indicate the displacement of the mouse. The frequency of transmission is one hundred samples per second. The x movement byte is connected to the column step size input port while the y movement byte is connected to the row step size input port. The row step size and column step size input ports belong to the game object module. This means that any new movement adds a displacement to the current location of the game object module. Any new movement adds a displacement to the current location of the Gun Game Object module.

12.3 Transform From Math Origin to VGA Standard Origin

Xilinx states that the PS/2 Mouse Device uses the traditional math origin, which means that the PS/2 Mouse Device treats the center as the origin [9]. In essence, coordinate (0,0) is located in the center. The value and magnitude of all movement packets sent by the PS/2 Mouse Device are with respect to this origin. For example, moving right results in a positive x displacement while moving left results in a negative x displacement. Xilinx states that the VGA standard, however, treats the top left corner as the origin [9].

These two different interpretations are problematic because of the location of the origin. Without any recalculation, the movement of the gun object is erratic. For example, moving the mouse in the downward direction means that the PS/2 Mouse Device is moving along the
negative y axis. Since the movement is along the negative y axis, the PS/2 Mouse device sends a negative number. The negative number is a signed two’s complement value. A problem occurs when a negative number becomes the displacement. The displacement is added onto the current coordinates of the Gun Game Object. Since the origin of the VGA standard is at the top left, going upward is interpreted as the negative direction while going downward is interpreted as the positive direction. Due to the different interpretations in the location of the origin, the PS/2 Mouse Device and VGA standard are opposites of each other. Figure 13 shows the PS/2 Mouse’s interpretation of the origin while Figure 14 shows the VGA standard’s interpretation of the origin.

| (-) x | (0,0) | (+) x |
| (0,0) | (-) y |

**FIGURE 13.** The PS/2 Mouse Device treats the center as the origin.

VGA Monitor

(479,639)

**FIGURE 14.** The VGA standard treats the top, left corner as the origin.
When the active player moves the PS/2 Mouse Device in the negative y direction, the gun crosshair should move in the downward direction of the VGA screen. In the VGA standard, moving in the downward direction is moving towards positive values. Thus, in order to get the desired effect, the FPGA developer must transform from the PS/2 origin to the VGA origin. This transformation is achieved by negating the value of the y movement packet, which turns the value into a two’s complement value.

Moving the PS/2 Mouse Device in the negative y direction results in the PS/2 Mouse Device sending a negative y movement value. A two’s complement operation on this negative value results in a positive value. A positive value as a displacement moves the gun cursor in the downward direction on the VGA screen, and the desired effect is implemented.
CHAPTER 13

FLYING DUCK GAME MODULE

The Flying Duck Controller and Datapath modules compose the Flying Duck module. The behavior of Datapath modules are based on the current state of the Flying Duck Controller module. This results in state based behavior. The module Get Duck Part on Crosshair is the module to be focused on. This module is within the Datapath and calculates the accuracy of the active player’s targeting. The accuracy or lack of accuracy is used to determine the next state of the Flying Duck Controller. Refer to Figure 15 when reading the following sections.

13.1 Get Duck Part on Crosshair

When the active player presses a button on the PS/2 Mouse Device, the game must determine the accuracy of the active player’s targeting. In essence, the game must determine if the active player has hit or missed a duck. Within the Datapath module is a subsystem called Get Duck Part on Crosshair. This module calculates whether the gun crosshair is positioned over a body part of a duck. The inputs of this module are gun x and y positions and the output is a code. The code indicates whether the gun is placed over a body part of a duck. If the code is a seven, the gun crosshair is not over the body part. The code would then be an input into a condition check. If the code is a seven, then the body shot wire is assigned the value of 1'b0. If not, then the body shot wire is assigned the value of 1'b1.

13.2 Trigger Status

The state of the trigger status is based on the value of the first byte received from the PS/2 Mouse Interface. If the first byte is either 8’hA, 8’hC, or 8’h9, the trigger status is assigned the value 1'b1. If not, the trigger status is assigned the value of 1'b0. If the first byte is 8’hA, the
right button has been pressed. If the first byte is 8'h9, the left button has been pressed. If the first byte is 8'hC, the middle button has been pressed.

FIGURE 15. Internal details of flying duck datapath.

13.3 Fatality and Dead State

A fatality occurs when the active player positions the gun crosshair over the body part of a flying duck and a button on the PS/2 Mouse Device is pressed while ducks are flying. A flying duck can reach the Dead state in two ways: by flying offscreen or by a fatality. A fatality means that the active player positions the gun crosshair over a body part of a duck and a button on the PS/2 Mouse Device is pressed. A flying duck can also reach the Dead state by flying offscreen.
A duck flies offscreen only if the duck flies the entire width of the screen without being killed.

The width of the entire screen is 640 pixels, as mentioned by Xilinx [9].
CHAPTER 14

TURN BASED GAME

The Duck Hunt FPGA Game is a turn based game that does not have a player priority scheme. The idled player must wait for the active player to finish. If the player turn is zero, then player one is the active player while player two is the idled player. If the player turn is one, then player two is the active player while player one is the idled player. The active player’s turn is finished when the current level is complete. The following sections discuss what conditions must be true for the level to be completed.

14.1 Level Complete

The active player’s turn is finished when the current level is complete. The current level is complete only when all the ducks are killed. A duck can reach a Dead state by either a fatality or by flying offscreen.

14.2 Calculate Level Complete

The current level is complete when all the ducks are in the Dead state. Determining the completion of a level requires the use of the bitwise AND operator. Each duck has a status flag called dead. The current value of the level done status flag is dependent upon concatenation and using the bitwise AND operator. First, the game concatenates together the dead status flags of each duck. Next, the game applies the bitwise AND operator to the concatenated value. If the resulting value is a 1'b1, all the ducks are dead and the level is complete. If there are ducks still alive, the level is not complete.
CHAPTER 15

SCORE MANIPULATION

The Duck Hunt FPGA Game is a turn based game. This means that there is one active and one idled player. When the active player shoots down a duck, only the score belonging to the active player should increment. An algorithm must make sure that the proper score is incremented. Without this algorithm, the idled player also receives credit when a duck is shot down. The modules of Figure 16 implement the algorithm.

FIGURE 16. Player turn and score manipulation are handled by the Player Turn and Profile Database module.

15.1 Score Buffer

Each player has a score buffer. The score buffer is a d flip flop with a load enable signal. If Player One is the active player, then the load enable signal for Player One’s buffer is 1'b1, while the load enable signal for Player Two’s buffer is 1'b0. When Player Two is the active player, the opposite is true. In essence, when Player Two is the active player, the load enable signal for Player Two’s buffer is 1 ’b1, while the load enable signal for Player One’s buffer is
1'b0. This means that there is a condition check that determines the value of the load enable signal for each score buffer.
The objective of the Duck Hunt FPGA Game is to shoot down flying ducks. In order to have the perception of a duck in motion, there must be a delay followed by a movement. This sequence of delay and then movement gives the impression that a duck is flying across the screen. A timer pipeline, which is shown in Figure 17, is used to implement the delay and movement sequence. The first level of the pipeline is the Base Timer while the second level is the Derived Timer. The third level is the flying duck.

16.1 Base Timer

The Base Timer has an initial count of zero and an end count of one thousand. The Base Timer is a free running timer, which means that the incr port has a value of 1'b1. When the Base Timer reaches the end count, the limit reached flag is set to 1'b1.

16.2 Derived Timer

The limit reached flag of the Base Timer is connected to the incr input port of the Derived Timer. When the Derived Timer reaches its end count, its limit reached flag is set to
1'b1. The limit reached flag of the Derived Timer is then connected to the move input port of a duck.
CHAPTER 17

EXPERIMENTAL PROCEDURE

The RUP is used because it emphasizes project management and development in iterations. The following paragraphs discuss the steps taken to develop the Duck Hunt FPGA Game.

17.1 Requirements Elicitation

Requirements are elicited by playing the Duck Hunt Game through online sites. The Duck Hunt Game has functional, hardware, software, and performance requirements. One example of a functional requirement is a reset game. The game must use a PS/2 Mouse Device as the gun, which is an example of a hardware requirement. The software requirements include the Xilinx ISE Design Suite 14.6 as the development suite. The VGA Controller must meet the timing specification of the VGA pixel refresh rate, which is 25.17 Mhz. This rate is an example of a performance requirement. The complete set of requirements can be found in Tables 2, 3, 4, and 5.

TABLE 2. Software Tool Requirements

<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The game shall be developed using the Xilinx ISE Design Suite 14.6</td>
</tr>
<tr>
<td>2</td>
<td>The Xilinx ISE Design Suite 14.6 shall be used to generate a bit file of the game</td>
</tr>
<tr>
<td>3</td>
<td>The bit file shall be downloaded onto the Spartan 3 FPGA Prototyping board using Digilent Adept</td>
</tr>
<tr>
<td>4</td>
<td>Visual Paradigm shall be used to create UML diagrams</td>
</tr>
</tbody>
</table>

TABLE 3. Functional Requirements

<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset Game</td>
</tr>
<tr>
<td>2</td>
<td>Resume Game</td>
</tr>
<tr>
<td>3</td>
<td>Pull Trigger</td>
</tr>
<tr>
<td>4</td>
<td>Move Gun Left or Right</td>
</tr>
<tr>
<td>5</td>
<td>Move Gun Up or Down</td>
</tr>
</tbody>
</table>
### TABLE 4. Hardware Requirements

<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The bit file of the game must be downloaded onto a Spartan 3 FPGA Prototyping Board</td>
</tr>
<tr>
<td>2</td>
<td>The game shall use the PS/2 Mouse Device for targeting</td>
</tr>
<tr>
<td>3</td>
<td>The game shall use the buttons of the PS/2 Mouse Device as trigger</td>
</tr>
<tr>
<td>4</td>
<td>The game shall be displayed onto a VGA monitor.</td>
</tr>
<tr>
<td>5</td>
<td>The bit file shall be downloaded onto the Spartan 3 FPGA Prototyping Board via a Digilent JTAG HS1 Programming Cable</td>
</tr>
</tbody>
</table>

### TABLE 5. Performance Requirements

<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The game shall have a frame refresh rate of 60 Hz</td>
</tr>
<tr>
<td>2</td>
<td>The game shall have a pixel refresh rate of 25 Mhz</td>
</tr>
</tbody>
</table>

#### 17.2 Use Cases

The next step is to write use cases. Most UML templates have two sections that are valuable to software and FPGA developers. These two sections are the main and alternate sequence sections, which describe the interaction between the active player and the Duck Hunt FPGA Game.

#### 17.3 Parsing

Parsing for nouns, adjectives, and verbs follows. The main and alternate sequence sections of a use case encode information about the video game. This information can be decoded by parsing the main and alternate sequence for nouns, adjectives, and verbs. Nouns become game objects, adjectives become attributes, and verbs become behavior. Parsing the main and alternate sequence sections reveal all game objects, attributes, and behavior needed to create the game.
17.4 Implementation and Unit Testing

Each game module or subsystem goes through white box and black box testing. Testbenches are written in order to perform unit testing for each game module and subsystem. The testbench instantiates the module, declares wires and registers, creates a running clock, and gives predefined input values for the instance. For black box testing, the actual output is checked to match the expected output. For white box testing, the actual state transition is checked to match the expected state transition.

17.5 Integration Testing

After unit testing, the next steps are integration and integration testing. The top level system is assembled incrementally. Rather than integrating a module and its behavior, integration is performed with respect to the use cases of the Duck Hunt FPGA Game. Recall that use cases are the functional requirements of the game.

The first use case that is integrated was the base use case, which is the Reset use case. The game is fully assembled when all use cases have been implemented.
CHAPTER 18
RESULTS AND DISCUSSION

There are many ways to design, document, and implement an FPGA video game. This project combines UML and digital design to create the Duck Hunt FPGA video game. The following sections discuss what is learned from developing the video game using this hybrid approach.

18.1 Object Oriented Digital Systems

The programming constructs of Verilog HDL can directly implement the characteristics of object oriented design. The resulting digital system is an object oriented digital system. An object oriented digital system is a digital system that possesses the characteristics of object oriented design. Since the resulting digital system possesses the characteristics of object oriented design, UML can be used to design digital systems.

18.2 FPGA Game Development is Different from Software Game Development

The biggest difference between FPGA video game development and software video game development is perspective. When developing FPGA video games, the perspective is at the hardware level. When developing software video games, the perspective is above hardware and more abstract.

At the hardware level, the game is a blank canvas. Thus, device drivers and image files have to be created. An example of this includes the VGA monitor. When the VGA monitor is connected to the VGA port of the Spartan 3 FPGA Prototyping Board, the VGA monitor displays No signal. The VGA monitor does not work because there is no device driver at the hardware level. A device driver must be created to control the VGA monitor. Software game development, such as Monopoly Empire, does not have such a requirement.
18.3 Control Versus Productivity

FPGA video game developers have more control while software video game developers are more productive. Designers and FPGA programmers should always start by looking for preexisting hardware solutions. However, if the preexisting hardware solutions do not meet design, timing, or functional specifications the designer and FPGA programmer can create their own hardware. Unfortunately, making custom hardware might mean having to create device drivers that is an extra step, which is a consequence of having control.

At the software level, device drivers have already been written. The Application Programming Interface (API) already exists at the software level. The API prevents software programmers from having to learn about the hardware. Rather than learning about the hardware and then writing code to control the hardware, software programmers need only to learn the API. Having the API lowers the learning curve, which results in productivity. Software programmers can focus on game development and avoid the extra step of writing device drivers, and thus increasing productivity.

Referring back to the VGA monitor example in Section 18.2, this project is developed from the hardware perspective. This means that the first task is to create a device driver for the VGA monitor. When compared to the software approach, having to create the hardware manually means that FPGA developers are one step behind in the development process.

18.4 No Super Loop

Super loop or polling is not needed when monitoring for events because Verilog HDL has parallel and sequential behavior. After the synthesis step, modules become hardware. This is similar to having multiple processing units in the same FPGA system. Multiple processing units are also synchronized to a common clock. Having a common clock allows parallel behavior
because all processing units load new data with respect to this common clock. For this project, all processing units load new data on the positive edge of the common clock. The parallel behavior means that a super loop is not needed for the Duck Hunt FPGA Game. The next step includes event checking and servicing. Distributed event checking using a finite state machine with datapath is the solution for both event checking and servicing.

18.5 Distributed Event Checking and Servicing Using Finite State Machine with Datapath

A distributed finite state machine with datapath checks for and services all events. A distributed finite state machine with datapath is implemented by using the programming constructs of Verilog HDL. There are two important key points about distributed finite state machines with datapath. First, event checking responsibilities are distributed to all modules and subsystems. Second, the finite state machine within each module and subsystem only checks for events that it is interested in. Finally, using the programming constructs of Verilog HDL makes event checking more efficient.

One must pay special attention to the sensitivity list of a combinational block. A combinational block can be used to calculate the next state of a finite state machine. A combinational block is composed of a sensitivity list and a computation block. The computation block executes whenever any member of the sensitivity list changes state. Taking advantage of this characteristic results in the sensitivity list being used as the event checker. An event occurs whenever at least one member of the sensitivity list changes state.

The Flying Duck Controller has this capability. Initially, the Flying Duck Controller enters the Reset state and then transitions to the Idle state. When the Flying Duck Controller is in the Idle state, it is only interested in the enable signal. Thus, the Flying Duck Controller only checks the state of the enable signal. This is why the enable signal must be part of the sensitivity list.
list. When an active player is ready to play the game, he or she presses the enable button. Pressing the enable button causes the enable signal to change state from 1'b0 to 1'b1. Since the enable signal is part of the sensitivity list, this state change is considered an event. Since an event has occurred, the computation block executes and calculates the next state transition. In the Idle state, the next state is Go To Respawn Position. The Flying Duck Controller then transitions from the Idle state to the Go To Respawn Position state.

The datapath computes based on the current state of the finite state machine. The Flying Duck Game module is part of the datapath module. The Flying Duck Game module sees that the Flying Duck Controller is in state Go To Respawn Position. The Flying Duck Game module responds by putting itself into the respawn position. The service, or event handler, is the Flying Duck Game module putting itself into the respawn position.

To recap, the event is the enable signal changing state from 1'b0 to 1'b1. The Flying Duck Controller then transitions from the Idle state to the Go To Respawn Position state. This causes the Flying Duck Game module to put itself into the respawn position.

When implementing event driven programming, the previous paragraph’s example shows that distributed finite state machine with datapath implemented using the programming constructs of Verilog HDL is a viable alternative approach to a main loop and polling.

18.6 Structural Hazard

The Duck Hunt FPGA Game has parallel computation but not parallel rendering. Digital systems usually have parallel behavior because of a common clock and multiple function units. However, the Duck Hunt FPGA Game only uses one VGA monitor; thus, only one device driver is needed. This is why there is only one VGA Controller module in the top level design. There are many game objects that have to be rendered but only one VGA Controller is used. This
causes a structural hazard. Since only one VGA Controller is used, only one game object can be rendered at any time.

The solution is to create a render pipeline. In this case, the pipeline is a digital pipeline where all stages are synchronized to a common clock. This means that each stage performs a different computation at each positive edge of the common clock. The output of one stage is passed onto the next stage. Despite all the modules and subsystems having parallel behavior, game objects are rendered in a serial fashion.

18.7 Uneven Pixel Rate Calculation

The clock frequency of the Spartan 3 FPGA Prototyping Board is 50 Mhz. The VGA specification requires a pixel refresh rate of 25.17 Mhz. Dividing 50 Mhz over 25 Mhz give a result of two. This project can implement a two count by using double flops. However, the remaining .17 Mhz are difficult to implement. Thus, the pixel refresh rate for the VGA Controller module is 25 Mhz and not 25.17 Mhz. Since the timing is not precise, there are several occasions where the VGA monitor displays No signal.

18.8 Solution to Uneven Calculation

The solution to an uneven pixel rate calculation is to mix clocked and unclocked elements. In this case, the element is a wire data type. An element that is stored into a synchronous buffer is called a clocked element. The solution requires manipulating hsync, vsync, and rgb in. All three elements are outputs from the Spartan 3 FPGA Prototyping Board to the VGA monitor. Mixing clocked and unclocked elements results in a several system state that works. The combinations are listed in Table 6.
TABLE 6. VGA Signal Combination

<table>
<thead>
<tr>
<th>Combination #</th>
<th>Buffered</th>
<th>Unbuffered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hsync</td>
<td>rgb in</td>
</tr>
<tr>
<td>2</td>
<td>hsync</td>
<td>rgb in</td>
</tr>
<tr>
<td>3</td>
<td>hsync</td>
<td>vsync rgb in</td>
</tr>
</tbody>
</table>

Note: The first combination is the version that generated the official bit file.

18.9 Solution Lifetime

Due to the uneven timing, the solutions in Table 6 have only an average lifetime of three uses. This means that the VGA monitor works for an average of three tries. After the third try, the VGA monitor displays No signal. Once the lifetime is up, the next configuration is used.

Despite the VGA monitor not working, the rest of the system works. Provisions for diverse testing are made in case all three configurations do not work. The provision is to use the seven segment display to test the system.

The seven segment display shows information about the current state of one of the four duck, movement packets of the PS/2 Mouse Device, current state of the Main Controller, current score, and current player turn.

18.10 Effect of Errors on Project

The uneven calculation does not stop the project but makes it tedious to test. The uneven calculation means that several combinations of bit files are generated. Occasionally, only one module does not work, which is the VGA monitor while the rest of the modules and subsystems do work. Tracing the state information using the seven segment display is proof that the rest of the system works.
CHAPTER 19
CONCLUSION

UML is the design language while Verilog HDL is the implementation language in this project. FPGA developers have the extra step of creating device drivers and image tiles before creating game play. The programming constructs of Verilog HDL are able to create device drivers, image tiles, and game play. The programming constructs of Verilog HDL can also implement the characteristics of object oriented design. The resulting digital system is an object oriented digital system, and the reward is that UML can be used to design digital systems.

Completing this project proves that a hybrid approach can be used to develop digital systems.

The base behavior of the game, such as game play, the render pipeline, and device drivers are the most difficult to develop. Now that the base behavior is complete, the next step is to augment the behavior. Currently, there is only one level, a static background, and no audio. The next step is to augment the behavior with new levels, a scrolling background, and an audio.

19.1 Levels

Currently, the game only has one level. One way to augment the game is to add more levels. There are two basic ways to increase the level of difficulty. One way is to make the ducks fly faster; another way is to populate the screen with more flying ducks. As the active player progresses from one level to the next, there will be more ducks on the screen and the ducks will fly faster.

19.2 Background Scrolling

Another great feature for the development is a scrolling background. The first version of the Duck Hunt FPGA Game has a static background. The game would be more interactive if the
background was dynamic. A scrolling background can be used to signify to the active player that the player has progressed to the next level.

19.3 Audio

Currently, the game is silent and does not have an audio subsystem. Audio is important to video games because audio can be used to signal a condition. There are several conditions in the Duck Hunt FPGA Game. An audio signal can alert the active player about a fatality, a level done, a miss hit, and a trigger pull. Audio also situates the mood for the game. For example, the audio transitions from slow music to fast music when the active player reaches a difficult level.
APPENDIX A

USE CASES
### Use Case #1: Reset Game

**Summary**
The active player presses the reset button and waits for the game to finish initializing.

**Actors**
The active player

**Preconditions**
The Spartan 3 FPGA Prototyping Board, VGA Monitor PS/2 Mouse Device are turned on. The bit file has been successfully downloaded.

**Description of main sequence**

1. Player Turn modules reset current count to 0
2. Player Profile Database sets player 1 and player 2 score to 0
3. Base and derive timer set reset their respective counts to 0
4. Flying Duck puts its self into the starting position
5. Gun cross hair is put into the staring position

**Description of alternate sequence**
None

**Post Condition**
The Duck Hunt FPGA Game is fully initialized

---

### Use Case #2: Play Game

**Summary**
The active player presses the enable button to player the game.

**Actors**
The active player

**Preconditions**
The game has finished initialization and is idled

**Description of main sequence**

1. The Base Timer starts to increment upwards from the initial count
2. The appropriate player score buffer is active
3. The PS/2 Mouse Interface checks for any received movement packets

**Description of alternate sequence**

1a. The active player presses the "Reset" button. Go to the "Reset Game" use case

**Post Condition**
The game is in progress
<table>
<thead>
<tr>
<th>#</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>Pull Trigger</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>The active player pulls the trigger to shoot down a flying duck</td>
</tr>
<tr>
<td>Actors</td>
<td></td>
<td>The active player</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
<td>The game is one and the current level is not complete</td>
</tr>
<tr>
<td>Description of main sequence</td>
<td>1</td>
<td>Receive a status bit from Get Duck Part on Crosshair</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Check trigger status</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Increment the current score</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Load the current score into the active player's score buffer</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Check if all ducks are dead</td>
</tr>
<tr>
<td>Description of alternate sequence</td>
<td>1a.</td>
<td>The gun crosshair was not over the body part of a flying duck</td>
</tr>
<tr>
<td></td>
<td>5a.</td>
<td>Not all ducks are dead, the current level continues</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>The active player presses the reset button. Go to &quot;Reset Game&quot; use case.</td>
</tr>
<tr>
<td>Post Condition</td>
<td></td>
<td>The active player's current score is incremented by 1</td>
</tr>
<tr>
<td>#</td>
<td>4</td>
<td>Move Gun Left or Right</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Summary</td>
<td>The active player moves the gun left or right to target a flying duck</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>The active player</td>
<td></td>
</tr>
<tr>
<td>Preconditions</td>
<td>The game is on and the current level is not complete.</td>
<td></td>
</tr>
<tr>
<td>Description of main sequence</td>
<td>1</td>
<td>The PS/2 Mouse Device sends movement packets</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The PS/2 Mouse Interface receives the movement packets</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Byte 1, the x displacement value, is routed to the column offset port of the Gun Game Object module</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>The column offset value is added onto the current column value of the Gun Game Object</td>
</tr>
<tr>
<td>Description of alternate sequence</td>
<td>5</td>
<td>The active player presses the reset button. Go to &quot;Reset Game&quot; use case.</td>
</tr>
<tr>
<td>Post Condition</td>
<td>The gun crosshair is displayed at its new position.</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Move Gun Up or Down</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The active player moves the gun up or down to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>target a flying duck</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Summary</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The game is on and the current level is not</td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Actors</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The active player</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Preconditions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Description of main sequence</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The PS/2 Mouse Device sends movement packets</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The PS/2 Mouse Interface receives the movement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>packets</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Byte 2, the y displacement value, is transformed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>into a 2’s complement value</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>The 2's complement value is routed to the row</td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset port of the Gun Game Object module</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>The row offset value is added onto the current</td>
<td></td>
</tr>
<tr>
<td></td>
<td>row value of the Gun Game Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Description of alternate sequence</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>The active player presses the reset button. Go</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to &quot;Reset Game&quot; use case.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Post Condition</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The gun crosshair is displayed at its new</td>
<td></td>
</tr>
<tr>
<td></td>
<td>position.</td>
<td></td>
</tr>
</tbody>
</table>
Main Controller State Transition

Flying Duck Control Unit State Transition
Player Turn Control Unit State Transition

- Reset
  - Load Player Turn Data To Init Ram
    - Init Cell 0
      - Init Cell 1
        - Filler State
  - Load Player Turn
    - Get Player Profile
      - Load Into Current Score
        - Idle
          - Increment To Next Player
            - Wait For Level Done
              - Store Into Database
              - Level done
PS/2 Mouse Interface

1. Reset
2. Fail
   - !correct
3. Load
4. Check Ack Code
   - correct
   - !rx done
5. Wait For Rx Done
6. Send Command To Mouse
   - !tx done
7. Wait For Sample Rate Time Up
   - time up
8. Done
   - Yes
   - No
9. Check If Receive All Packet
10. Incr Packet Count
    - rx done
11. Wait For Rx Done
    - rx done
APPENDIX C

CLASS DIAGRAM
REFERENCES


