Mini Project Report – Team 35

Team Members: Ken Fong and Utsav Trivedi.

University of Auckland

Department - Computer and Electrical Engineering

Abstract

This course project intends to provide a gaming console experience using a FPGA. Game design and digital logic principles learnt prior are used to implement a side scroller game. Users can interact via connected man machine interfaces.

Executing logic using a design model is necessary for proper implementation of the game. We have used a FSM-D architecture as our basis for this game. Correct logic element and other FPGA resources use was learnt during this project.

Introduction

The project is aimed at designing a side scroller flappy bird themed game using digital logic and its principles using a FPGA and a man-machine interface.

The preliminary rules entail that the bird must keep staying afloat and avoid obstacles to survive. There are special pickups like food, money and medicine boxes that can replenish its health or reward the player by adding scores. The motion speed of the screen and obstacles change as the player progresses to higher levels.

The code base components provided to us were the digital to analog converter for our screen, mouse, text data, configured embedded memory block (ROM) and character. We have since added to the code base to use all the pins of the video display port, and changed the given ball block. In terms of features we have implemented a graphics user interface, finite state machine, game logic that is different to the flappy bird game and visuals that use embedded memory blocks of the FPGA.

Game Features

Brief: Our game **"Happy Copter"** is a survival game with one life. It consists of over five levels and obstacles which the player must avoid to survive. Players gain points for completing levels and the time they are alive. Since our protagonist represents a helicopter, we kept one life to show that on crashing, the helicopter is destroyed and have additional bombs falling from the sky to emulate war conditions.

Our game consists of the basic concepts of flappy bird along with other features such as:

Extra obstacle: The player will encounter a bomb (Cyan cubes) that falls vertically which adds to the challenge of the game.

Levels: Pipes move faster as the level increases.

Practice mode: The player can play for an unlimited time in the first level where the motion speed of each pipe is constant. Score is not added.

Score Mechanics: This will be based solely on time.

Single-Player mode: The player will see an increase in levels of the game as the player survives for longer.

How to Play

Provided Equipment: Deo Board and PS2 Mouse.

DEO Board Controls: The first slider switch *"**SW0**"* allows the player to choose between the real game and training mode. This is shown on the seven-segment display.***"Button2"*** allows the player to reset the game at any time.

Mouse Controls: ***"Left click"*** makes the helicopter traverse up. One can keep it pressed or click in periods to throttle up. **"*Right click"*** makes the helicopter swing in a horizontal parabola. This is useful for dodging bombs. **"*Middle mouse click"*** pauses the game or sends the player back to menu when they die.

Game Strategy

Rules: The player must throttle up or dive down to navigate through pipes and swing from left to right to avoid bombs

falling from the top. One scores points by remaining alive as long as they can and progressing to higher levels.

Final Design and Implementation

Our design consists of a control unit and data-path but most of our game logic is handled in the data-path. The control unit has full control over the data-path as it passes control signals so that it only runs processes when needed.

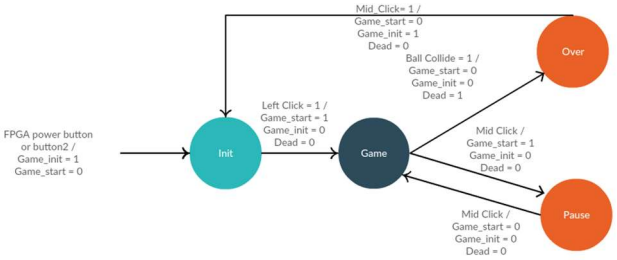

Fig 1 – State Machine Diagram

Finite State Machine (FSM): Our mealy finite state machine consists of 4 states that are "s_init, s_game, s_pause and s_dead". Each of them represents the initial menu, game, pause menu and game over screen respectively. It outputs three control signals that are "game init, game start and dead". Depending on the state and input received from the datapath, it will change state and output the appropriate control signal values to change the game settings. Our control signals are "game_init, game_start and dead". The first one initializes the game objects, the second one starts or pauses the game based on user input and the third one is sent to the text_display block to display game over text on the screen.

User Action Block: This handles inputs whenever the player presses a button on the mouse or DEO board. It passes inputs to the FSM to trigger state changes and game action signals if the player is in the game state. Eg – When the player clicks the middle mouse button to pause the game or left click to make the helicopter throttle up.

Graphics Manager: This block contains a priority encoded mux that handles the rendering order of the game objects depending on where they are currently on the screen. It accepts RGB values from the different components that are displayed and outputs the selected component's RGB values to the given "VGA Sync" component.

Collision Manager: This block receives enable signals from the helicopter, pipe and bomb to decide if a collision has occurred when they are rendered at the same place or if the helicopter goes outside the screen boundaries. It passes a status signal to the FSM to transition to the state "s_dead".

Level Manager: This block handles the level transitions in game state. It consists of a timer that changes the level and outputs a signal to the obstacles when the limit of the timer is reached. When received, it changes the speed of the pipes to be displayed for the level.

LFSR: This block consists of a left feedback shift register that sends random values within a range to the pipe to change the placement of the gap and bomb to change frequency of drop.

Text Display: This block selects the text to be displayed based on the control signals received from the FSM and increments score and level based on gameplay. It sends the required addresses of the text from the mif file to the given "char rom" block to read the specified text data at the given address.

Helicopter: This block represents our main character. The logic of this block differs the most to the other game objects as it has both horizontal and vertical motion.

Pipe: This block is the first kind of obstacle in our game. It is blue in colour and moves from right to left. It increases its speed with higher levels and it changes the gap placements based on input from the LFSR block.

Bomb: This is our second type of obstacle that moves from top to bottom of the screen and changes its frequency of dropping based on input from the LFSR block.

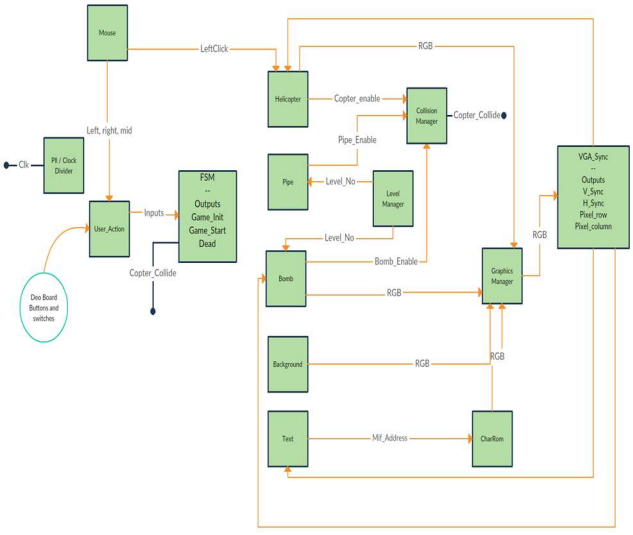

Fig 2 – Block Diagram

Results

Flow Summary	
Flow Status	Successful - Wed May 29 21:26:16 2019
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Full Version
Revision Name	305_project
Top-level Entity Name	305_project
Family	Cyclone III
Device	EP3C16F484C6
Timing Models	Final
Total logic elements	1,479 / 15,408 (10 %)
Total combinational functions	1,411 / 15,408 (9 %)
Dedicated logic registers	384 / 15,408 (2 %)
Total registers	384
Total pins	47 / 347 (14 %)
Total virtual pins	0
Total memory bits	397,888 / 516,096 (77 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	1 / 4 (25 %)

Fig 3 – Compilation Report

We used a PLL block to divide the board clock at half its frequency. This allows the game to run at 640 × 480 resolution smoothly. We chose this as we wanted to reserve logic elements for our game logic. Our main pin usage is from the seven-segment display (7 × 4) and VGA output pins (3 × 4).

Our design uses 10% of the logic elements it contains. This is mainly because we of our state action model approach. We have separated our game logic in small blocks to increase cohesion. The reason behind this was that it makes reusing logic easier and any developer could use our game blocks to make their own game. Most of our look up tables are generated in the text display block as we are specifying addresses of the given mif file using conditional signal assignments. After we changed from 1-bit to 4-bit graphics there was a significant increase in logic elements as well.

Our memory bits usage is near 80%. The reason for this is we are using embedded memory blocks to display our background image from a mif file in addition to displaying text. We decided against going lower than 12 bits as our image started distorting once we went got rid of significant portions of pixel data. This is because the M9K blocks cannot fit in memory locations the size of the operating resolution so we scaled it to fit our screen. Hence, we think the graphics improvement is justified. Our main character uses 64 bits which is less than 1% of the memory.

Slow 1200mV 85C Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	94.19 MHz	94.19 MHz	inst5|altpll_component|auto_generated|pll1|clk[0]	
2	180.96 MHz	180.96 MHz	VGA_SYNC:inst|vert_sync_out	
3	323.1 MHz	323.1 MHz	MOUSE:inst3|MOUSE_CLK_FILTER	

Fig 4 –Timing Analysis

Our worst-case performance is at 94.19 MHz and critical path is near 11ns. This restriction is due to some of our game logic being in a clocked process while some on the rising edge of the vertical sync event. We can improve our performance by using a moore state approach which should reduce logic element usage and lower our critical path as actions are performed based on state only. Since 94.19 MHz is still higher than the PLL clock it is still ideal.

Future Work

Given more time, we would have liked to improve the graphics, load sound, store high scores, implement a multiplayer mode, helicopter bullets, boss level and fuel pickups.

Boss: Our boss would be an enemy fighter-jet moving in a parabolic motion shooting projectiles.

AI: We can use the idea of AI to allow the boss to have a smarter behavior such as aiming at the player or dodging player fire.

Fuel: This pickup would replenish the copter's health making it last longer.

Helicopter: Our helicopter would have the additional feature of being able to shoot bullets.

Multiplayer mode: A second copter can be controlled via the DEO board switches and buttons. Players can compete against each other or play together against the boss.

SD Card: The FPGA provided to us has the capability to load a SD card. This would be useful to load scores from each session like a real game console.

Conclusion

In conclusion, our project was aimed at training students to make a game console on a FPGA using digital logic. The theme was a side scroller "Flappy Bird" game with freedom over implementation. We designed a survivor game with over 5 levels that allows a player to consistently strive for a new high score.

References

Terasic Technologies. (2009). *DEO User Manual*. Retrieved from https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=364&PartNo=4

Altera. *Cyclone III Device HandBook.* Retrieved from https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyc3/cyclone3_handbook.pdf

Fong, Ken. *Night Sky.jpg.*

University of Florida. *Generating .mif files from images.* Retrieved from http://www.gstitt.ece.ufl.edu/courses/eel4712/labs/lab6/lab6_mifgen.pdf

Piotrowski1994. *Praca, miffilegen.m.* Retrieved from https://github.com/Piotrowski1994/praca/blob/master/miffilegen.m

Appendix – ASM VGA Chart