Project
0 to 4 player pong

Robert Foss (dt08rf1@student.lth.se)
Mikael Sahlström (dt08ms2@student.lth.se)
Mikael Nilsson (dt08mn2@student.lth.se)

It is customary to include Date and maybe Course code and name on the title page.

Contents

1 Abstract

This project implements a 0-4 player pong game on a Digilent Nexys 3 FPGA. The game is displayed via VGA and a 7 segment display. A USB-keyboard is used to control the gameplay. Uncontrolled players are controlled by an AI player. Two microblaze cpus are used, one for gameplay and one for reading keyboard input.

2 Introduction

There are spell checkers for latex, so you should not have spelling errors in your report.

This project covers a 0-4 player Pong game on a Digilent Nexys 3 FPGA.

The game is displayed via VGA using a resulotion of 640x480 pixels at 60Hz and 7 segment display.

The players paddles are controlled with an USB keyboard which internally is connected via PS/2. Each paddle can be moved in 2 directions. A player can elect to give up control of his paddle by enabling an AI player. The AI player can be enabled for any combination of actual and AI players.

Actually that was the least of the goals. The main goal should have been for you to learn more.

One goal of this project was to make a fun game, if the goal was achived is for the players to determine.

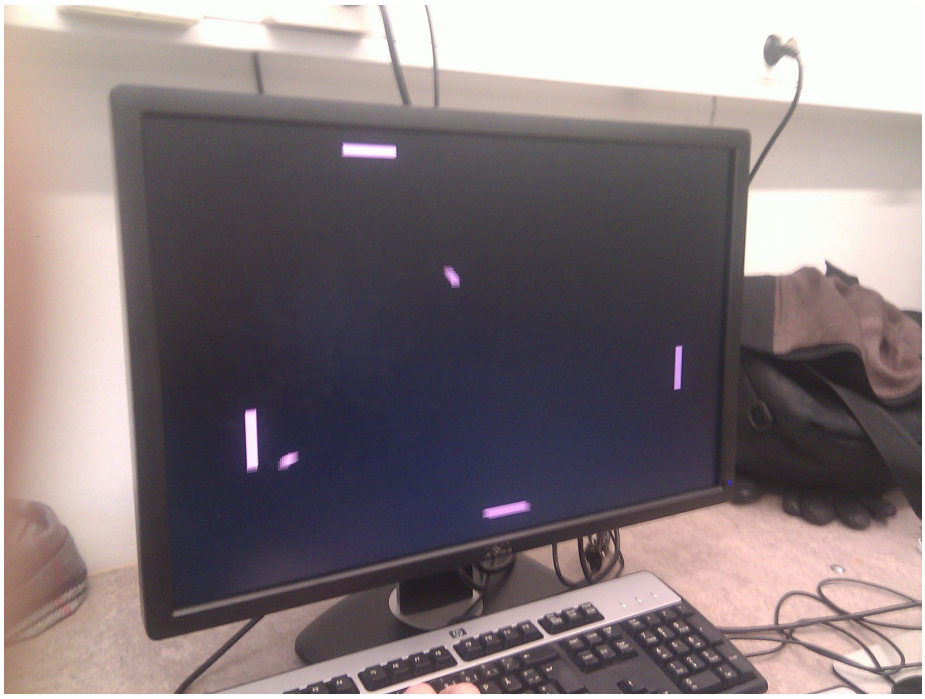

Figure 1: A photo of actual gameplay.

Nice with a photo of the system in action.

3 Hardware

The hardware consists of 5 parts. Microblaze 0 and 1, VGA controller, PS/2 controller and 7 segment controller.

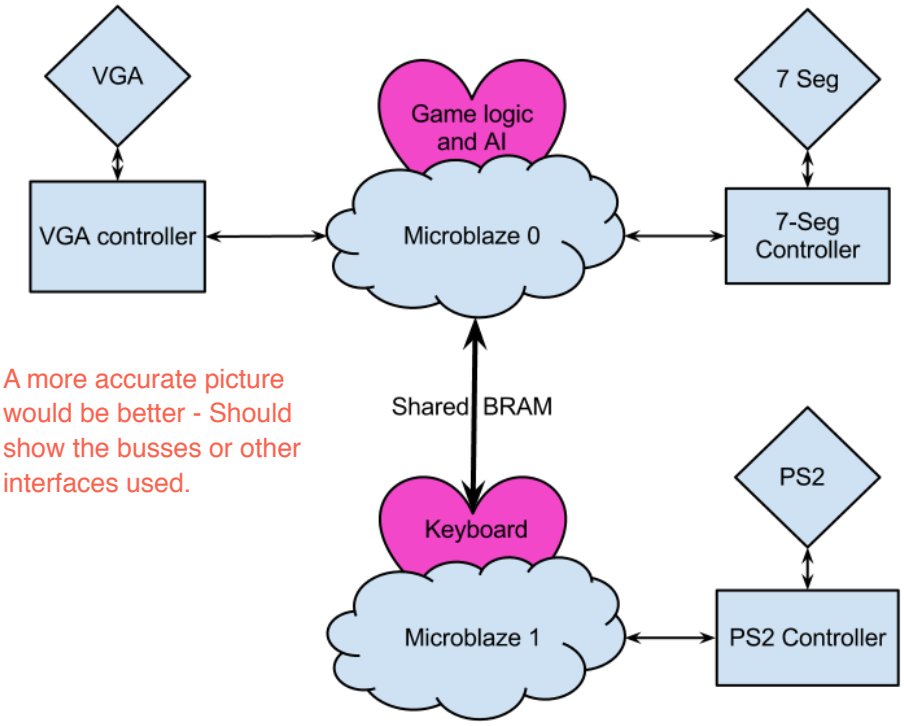

Figure 2: Hardware and communication paths.

3.1 Microblaze 0 and 1

Microblaze 0 handles gameplay logic and updating displays (VGA and 7seg). Microblaze 1 is responsible for reading and parsing PS/2 input from the keyboard. The reason for the division of labour between the two cpu cores is that the main loop of the game is longer than the minimum amount of time a keystroke will be available for polling from the PS/2 interface. Hence the PS/2 polling was split into a second cpu core due to ease of implementation.

Microblaze 0 communicates with the VGA controller via FSL, since the amount of data needed by VGA controller to accurately depict the current game-state is only 76 bits, see section 8.

Communication between microblaze 0 and 1 is handled via shared bram, which describes which keys are currently active. The shared memory is only read by cpu 0 but read and written by cpu 1.

3.2 VGA controller

The VGA controller is split into two parts. One that handles the VGA specific things (VGA monitor controller in figure 3). The other part handles the communication with the mircoblaze and logic for when to draw, called the logic part (VGA I/O, game state and combinatorial net in figure 3).

The microblaze sends the coordinates of the paddles and balls to the VGA logic part which also recives the pixel that currently (or soon) is going to be

The size of paddles could be easily made adjustable, making the gameplay more fun.

drawn. The logic part then calculates if that pixel is part of either the paddles or the balls and if it is, the logic part sends paint to the VGA part and that pixel will be painted. This means that the forms and sizes are hardcoded in VHDL.

The VGA part runs on a 25 MHz clock while the logic part runs on a 50 Mhz clock (same as the microblazes). The choice to run the VGA part on 25 Mhz was to make the vertical- and horizontal-sync easy to handle.

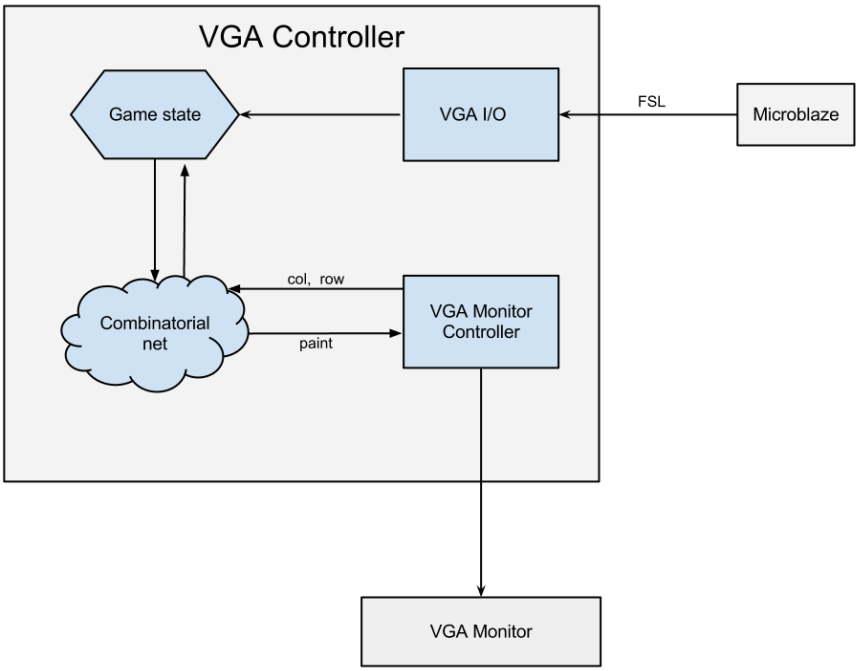

Figure 3: Internal design of VGA controller.

3.3 PS/2 Controller

The PS/2 controller consists of the xps_ps2 module from Xilinx Platform Studio. The Nexys 3 board uses the microcontroller PIC24FJ192 to abstract away the USB HID interface and present a PS/2 keyboard interface. The xps_ps2 module reads ps2 data and clock and via PLB communicates scan-codes which represent key events.

Like mentioned earlier the PS/2 interface is polled by cpu 1 to retrieve new key events. As mentioned ...

3.4 7 segment display

The 7 segment display consists of the digilent_sevseg_disp module from the Digilent Nexys 3 EDK. The module is connected directly to the hardware via the anode and cathode logic vectors. The module reads data via PLB from cpu 0.

It's improper to say that the module reads data, since the module is a slave on the PLB. The CPU writes data to the module!

3.5 Device Utilization summary

Table 1 shows resource utilization. Utilization could easily be lowered by moving away from a two core design, for our usecase high utilization is not problematic.

	Used	Available	Utilization
Occupied Slices	2,240	2,278	98%
Slice Registers	5,390	18,244	29%
Slice LUTs	6,809	9,112	74%

Table 1: FPGA resource utilization.

4 Software

Software will keep the game state (paddle positions, AI movement for each enabled AI and ball positions) updated and send current positions to the VGA controller. A life counter is displayed on the 7 segment display with each segment representing the number of lives left for a player. The last players with any lives left is the winner.

4.1 AI

Each paddle can be controlled by a so called AI. If a paddle is controlled by the AI, a function is called with parameters containing the current game state (poistions of paddles and balls) and which paddle it is supposed to control. This function will return either 1, -1 or 0 depending och which direction the AI wants the paddle to move. The direction is deterimined by predicting where each ball will hit by checking the movment speed on the x-axis and on the y-axis. The AI will only react to balls moving (on any axis) towards the paddle. The prediction does only cover the next hit, not where the ball will go after a hit. The AI is not agressive and only tries to prevent the ball from hitting a wall.

5 Installation and controls

Included in our archive is the download.bit file. Use Digilent adept or a similiar tool to download the game onto a Nexys 3 FPGA.

The game is controlled via Q/A, T/Y, O/P, PgUp/PgDn and 1/2/3/4 for AI toggling.

6 Lessons and conclusions

The first iteration of this project was not a game but rather an ethernet switch. Due to limitations in the tooling and time considerations we were forced to scrap this idea and replace it with something a bit more feasible.

An earlier iteration of the game used a custom PS/2 module which proved to be fairly hard to integrate due to non-digital characteristics of the internal PS/2 interface combined with poor diagnostic support in the tooling.

An alternative solution to using polling would be to use interrupts for PS/2 communication. The reason for not using interrupts is pragmatism, it is easier

and faster to implement a second cpu core and polling than it is to implement well functioning interrupts.

Well, yes, but then you have the shared memory issue (not to mention the size), which for more complex data would be a problem.

7 Contributions

The VGA controller was written by Mikael and Mikael. The game logic was written by Mikael N. The AI was written by Mikael S. The seven segment display and PS2 keyboard was written by Robert. Software and hardware integration was done by all of us.

A list of references would not hurt.

8 Appendix: A

```
Paddle0, y -- 9  bits     // West
Paddle1, y -- 9  bits     // Eeast
Paddle2, x -- 10 bits     // North
Paddle3, x -- 10 bits     // South
Ball0, x   -- 10 bits
Ball0, y   -- 9  bits
Ball1, x   -- 10 bits
Ball1, y   -- 9  bits
--------------------
Sum        -- 76 bits
 0               16             31
+--------------------------------+
|Paddle0y|Paddle1y|Paddle2,x|    |
+--------------------------------+
|Paddle3,x|Ball0, x |Ball0, y|   |
+--------------------------------+
|Ball1, x |Ball1, y|             |
+--------------------------------+
```

Bitfields for CPU→VGA communication.