Department of Electrical and Information Technology

EDA385 Design of Embedded Systems

Advanced Course

PROJECT REPORT

5-IN-LINE GAME

Group Member: Zhonghua Wang (mas09wz1)

Ziyang Li (mas09zl1)

Hongwan Qin (mas09hq0)

Teacher: Flavius Gruian

Submission Date: 8 November 2010, Monday

Abstract

This report documents the development of a 5-in-line game running on a Digilent Nexys-2 board with Xilinx Spartan-3E FPGA (XC3S1200E, FG320). The project is mainly developed in the environment of Xilinx ISE and Xilinx Platform Studio, with a small piece of parameter extraction work done in Matlab. The system consist of a Microblaze microprocessor, some IP core provided by Xilinx and a custom VHDL IP. Keyboard and VGA monitor are two user interface of this board-based game.

Table of Content

1 Introduction

This chapter briefly introduce the hardware and software environment of our design.

1.1 5-in-line Game

This is a simple and interesting chess game played by two players on a chess board. The basic idea of the game is that two players place a piece in a grid in turn. The one who has 5 pieces placed continuously, no matter horizontally, vertically or diagonally, wins the game, shown in Figure 1.1. In our project, the user will play against the machine, which has a certain degree of intelligence.

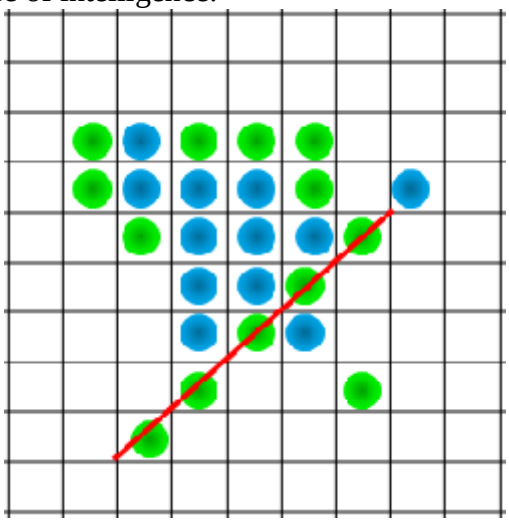

Figure 1.1: An illustration of the 5-in-line game

1.2 Hardware Platform

The 5-in-line game project was developed on Digilent Nexys-2 board with Xilinx Spartan-3E FPGA (XC3S1200E, FG320). The system frequency of this board is 50MHz.

The microprocessor included in the system is MicroBlaze, which is a embedded processor soft core optimized for implementation in Xilinx FPGA[1].

1.3 Software Environment

The project is developed in mainly in ISE and XPS of the Xilinx ISE Design Suite 12.2 - a design tool package for developing embedded systems on Xilinx FPGAs. Other software utilized in the project include Digilent Adept 2.4 System for download of the design into the FPGA, and Matlab for some parameter-extraction work.

2 Hardware implementation

This chapter introduces the hardware architecture of our project. And the implementation of the custom IP - VGA controller is explained in detail.

2.1 Overview of the hardware architecture

The hardware mainly consists of following components: MicroBlaze microprocessor, BRAM, PS/2 controller, RS232 (UART), Clock generator, PLB and FSL bus, these are soft IPs provided by XPS; besides, there is a custom VGA controller IP. The block diagram of the system architecture is shown in Figure 2.1. The PS/2 controller connected to the PLB bus is responsible for the communication between the processor and the keyboard; the RS232 controller sends data to the hyperterminal on PC (i.e. printout for

debugging purpose); and the clock generator divides the system frequency (50MHz) by two to provide the 25MHz clock frequency to the VGA controller. Our custom hardware VGA controller, seen as a IP as well, communicate with the processor through the Fast Simplex Link (FSL) bus. This bus becomes our choice instead of PLB because in our design, only a word of data need to be sent to VGA controller occasionally and memory access is not necessary. In our design, BRAM is only used for the storage of the data and instructions of the software.

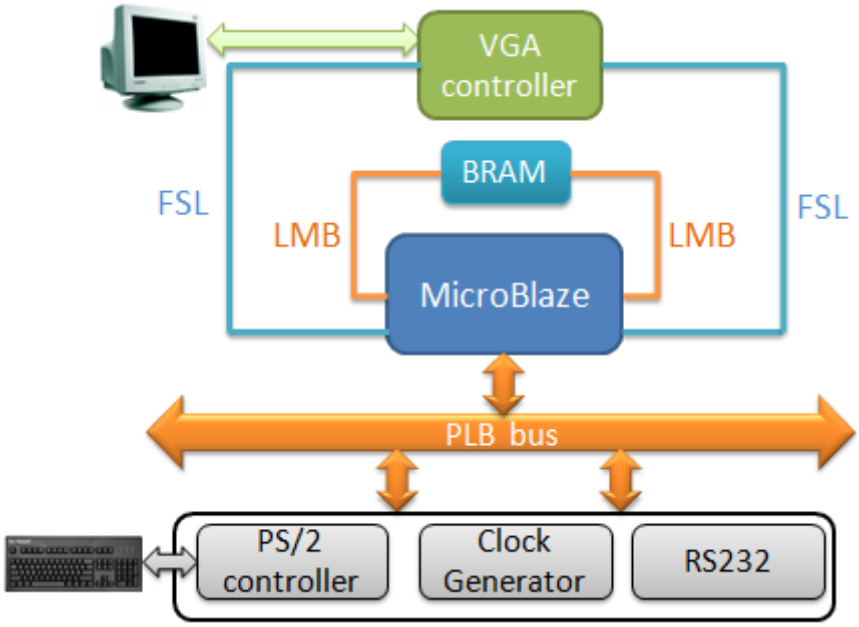

Figure 2.1: Hardware architecture of 5-in-line game project

2.2 VGA controller

The VGA port of Digilent Nexys-2 board uses 10 FPGA signals to create a VGA port with 8-bit color and two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). There are eight signal levels on the red and green VGA signals and four on blue, so 256 different colors in total can be displayed, one for each unique 8-bit input data[2]. The image on the monitor screen is obtained by constant refreshing of the frame, which is scanned from the top-left pixel to the bottom-right pixel. Both the synchronization signals and the pixel color signals should be generated by the VGA controller. In our design, the controller is divided into two modules: one for generating the synchronization and timming signals and the other for generating color signals for each pixel. The block diagram of the VGA controller is shown in Figure 2.2.

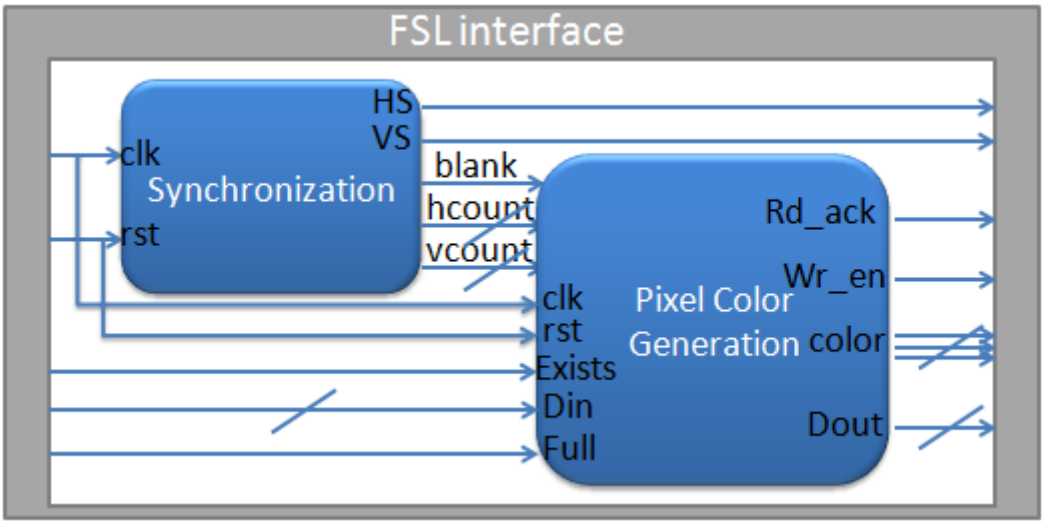

Figure 2.2: VGA controller block diagram

2.2.1 Synchronization module

The VGA controller must retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel. Therefore, it must generate the HS and VS timings signals and coordinate the pixel data output based on the pixel clock, which defines the time available to display pixel. For a 640x480 display with 60Hz refresh, the signal timings can be found in the user manual of the board[2]. The horizontal-sync counter (hcount) and vertical-sync counter (vcount) can be used to locate the colomn and the row of a pixel respectively. The pixel data is defined only if the counters correspond to a pixel in the visible area, when the blank signal is invalid.

2.2.2 Pixel Color Generation module

In order to minimize the memory space and data transfer through the FSL bus for the VGA display, we decided to store only the necessary information in the hardware registers. The entire area for displaying chess board is 450 pixels by 450 pixels, which is evenly divided into 15 rows and 15 colomns, with a grid consists of 30x30 pixels. Each grid has two parts: background part and foreground part. The background part includes a two-pixel wide frame and the blank area surrounding piece and cursor. The foreground is further divided into two areas: the piece area and the cursor area. There are three states of the piece area: blank (it is an unoccupied grid), green (a man's piece is placed in this grid), or blue (a machine's piece is placed in this grid). Therefore, two bits is enough to store the state of the grid, thus 15*15*2=450 bits are required to store the states of all grids. Whether the cursor should be displayed depends on the current position of the cursor. Hence, only the location and the state of the current grid are required to provided by the software through FSL once either player make a move. At the beginning of the game, for example, the man places a piece at the center of the board, then an instruction word packaged with state section ("001"– indicates it's a green piece), x coordinate section ("0111" - the center colomn 7) and y coordinate section ("0111" – the center row 7) is delivered to the module input and the registers values of the corresponding grid is modified, leading to an update of the grid displaying. An illustration of the grid displaying is shown in Figure 2.3.

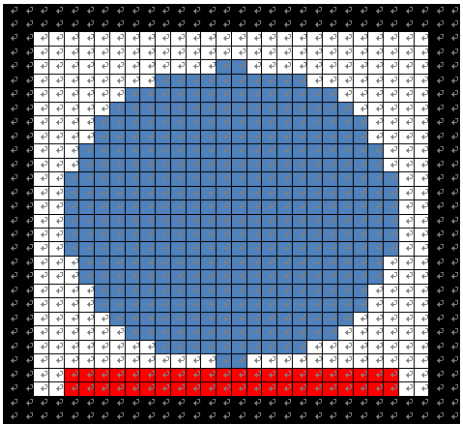

Figure 2.3: Illustration of grid displaying with pixels

The winning sign and the “clear” operation is shown on the screen when the state section of the instruction word is assigned with the predefined value. If this state section is “100”, all state-registers are reset to “00” and the piece area of all grids are cleared on the screen, with only a cursor sign left. If the state section is “011”, which means one player wins the game, then the grids of winning pieces will be assigned “11” and their piece area will appear a red circle.

The synthesis report on the device utilization of VGA controller is summarized in Table 2.1. It can be observed from the table that the hardware module consumes over five hundreds of flip flops, but no BRAM. Compared with BRAM, the data storage of flip flops is more efficient and the access time is much less.

Resources	Used/Available	Utilization
Number of Slices	1239 out of 8672	14%
Number of Slice Flip Flops	539 out of 17344	3%
Number of 4-input LUTs	1933 out of 17344	11%
Number of bonded IOBs	0 out of 250	0%
Number of BRAMs	0 out of 16	0%

Table 2.1 Device utilization summary of the VGA controller

3 Software implementation

3.1 Design overview of the software implementation

The whole software of this 5-in-line game should consider how to deal with the input data from the IP core of PS2 as well as design an algorithm for the machine. Another important task of the software is to communicate with the VGA controller which means the software should grasp when to transfer useful information to the VGA controller. Further more, the software also should implement all judges such as whether a special position can be place a piece and who win the game. The total flow of the software is as Figure 3.1 below.

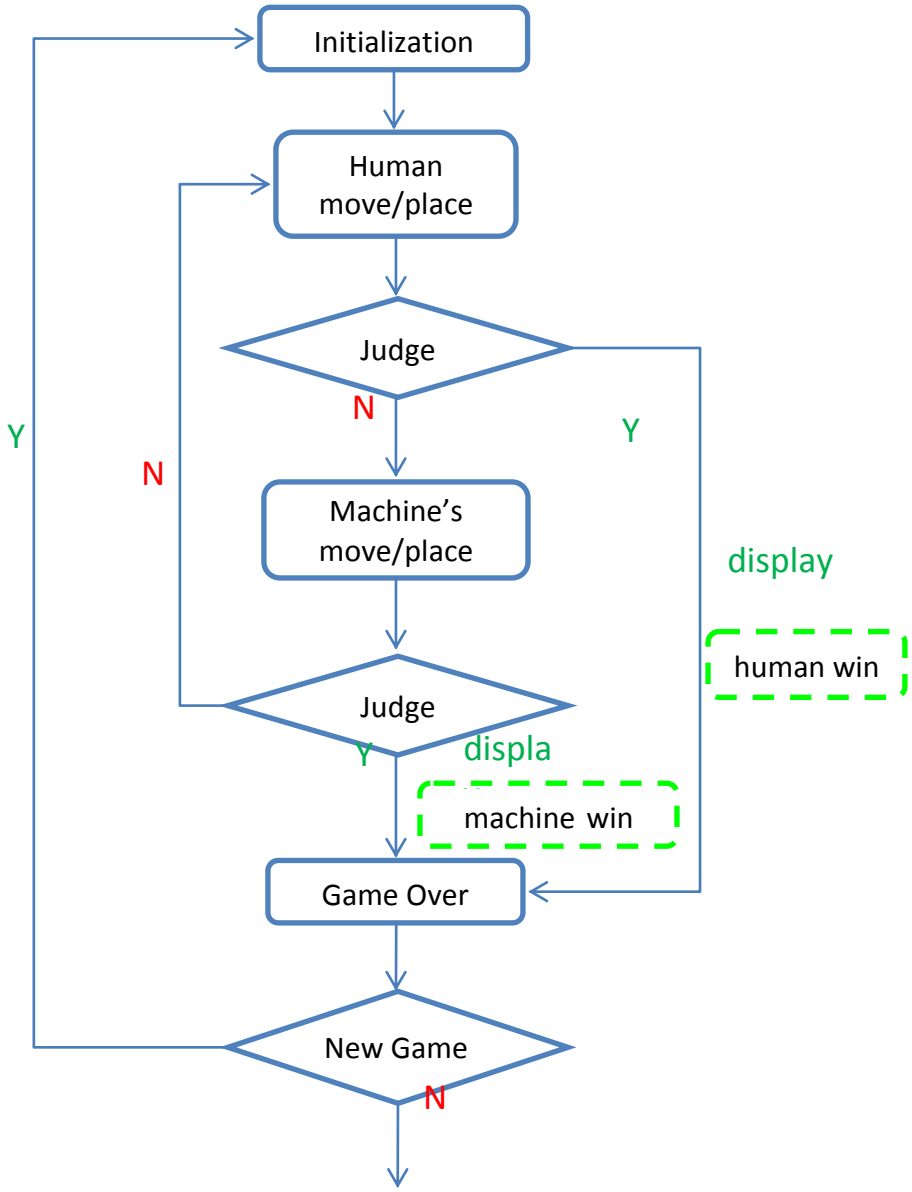

Figure 3.1 The total flow of the software for the 5-in-line game

In Figure 3.1, there are two tests of which the first one is to judge if man has won or not and the second one is to judge machine won or not. In the module of 'human move/place' the software will receive the input data from the keyboard and judge is that a valid signal, then moves the cursor or places a piece as the input control signal. The part of 'machine' move/place' is an compound algorithm for the machine including calculating which grid is the necessary position to place a piece and transferring the corresponding display information to the VGA controller. No matter who win the game it will wait for an instruction of starting a new game.

3.2 Detail of the software implementation

3.2.1 Storage organization

Since the display function is implemented by the hardware of VGA controller the software can use a relatively small memory to store all the constants and variables. We choose the BRAM as this small memory because it's convenient storage and access characteristic. On the other hand in hardware we define one group of 32 bits signal transferred from the FSL bus as one unit information for a grid. The four least significant

bits are defined as the ‘Y’ coordinates information and the upper four bits are defined as the ‘X’ coordinates, then another higher three bits are defined as the color information. So each time the valid information transferred to the VGA controller is ‘color+16×X+Y’. For example if the color of green is defined as ‘001’ in hardware we should transfer ‘256+16×X+Y’ to the VGA controller.

3.2.2 Receive signal from the PS2 controller

We use the IP core of PS2 controller as a hardware module to handle communication between keyboard and CPU. To use this PS2 IP core we will apply the following three function provided by the EDK suite[3].

- ✓XPs2_LookupConfig(KEYBOARD_ID)
- ✓XPs2_CfgInitialize(&Ps2Inst, ConfigPtr, ConfigPtr->BaseAddress)
- ✓XPs2_Recv(&Ps2Inst, &RxBuffer, 1)

Where the variable ‘Ps2Inst’ is defined as XPS2 pattern which means it is a PS2 driver instance. ConfigPtr is a point variable of XPs2_Config pattern which is used to point the base address of the PS2 controller allocated automatically. The first two functions are used to do initializing of the PS2 controller and the last function is used to receive data from the key board and store it in the RXBuffer variable.

To enhance reliability we should firstly detect the break code from the key board which step indicates that a certain key is releasing. Then we receive the following scan code as desired data. In this game we choose ‘UP’,’DOWN’,’LEFT’,’RIGHT’ key to control which direction to move the cursor and choose ‘ENTER’ key to try to place a piece on a certain position. Another key of ‘ESC’ is applied to clear the whole chessboard and begin a new game. So after receiving a desired data the software will detect is it a valid signal. In other words only when the software detect that the received scan code is one of those direction control and place and clear signals it will continue to implement, otherwise the software will wait for the next valid input signal.

3.2.3 Human move/place

If the valid input signal from the keyboard is direction information the software will change the current coordinate of the cursor according to the selected direction and put the new coordinate to the VGA controller. If the valid input signal is place instruction the software will judge if there is an existed piece at the position of the current cursor. If the position of the current cursor is empty software will change the value of the grid array in the BRAM and simultaneously transfer the new 32 bits signal including color and index information to the VGA controller. If the input valid signal is clear software will call the ‘clear’ function which is appointed to clear the whole grid array and transfer empty color information 225 times to the VGA controller which is used to clear all the grids.

3.2.4 Judge win or not

After placing a piece to the chess board the software will right away judge whether man win or not. The judgment can be composed to a function and the flow is as the following Figure 3.2.

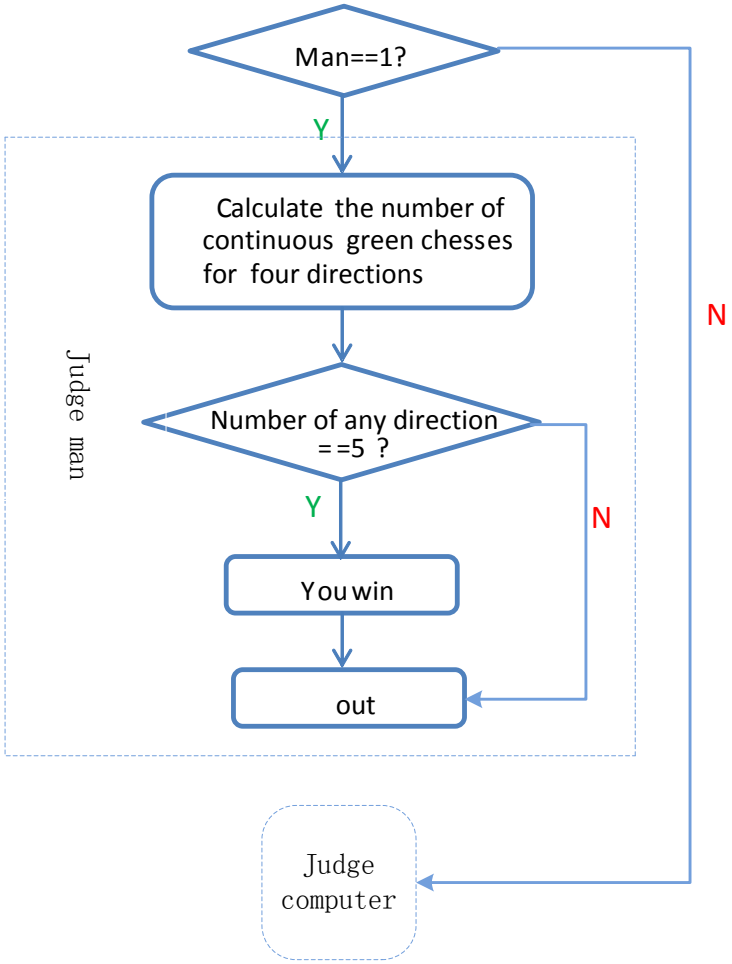

Figure 3.2 The judgment of win or not

In Figure3.2 variable ‘man’ is a real parameter variable which indicating judging human or machine’s turn. If now judge human turn the software will calculate the number of continuous green pieces from the recently-placed piece. The calculation will be implemented four times corresponding four directions that include from right to left, from down right to up left, from down to up and from down left to up right. The calculation directions are showed in Figure 3.3.
If the number equals to five then the software will call the function of ‘you win’ in which it will transfer the coordinate information of these five continuous pieces and red color to the VGA controller. If the variable ‘man’ is 0, the software will implement another calculating in which the color of the continuous five pieces should be changed to blue.

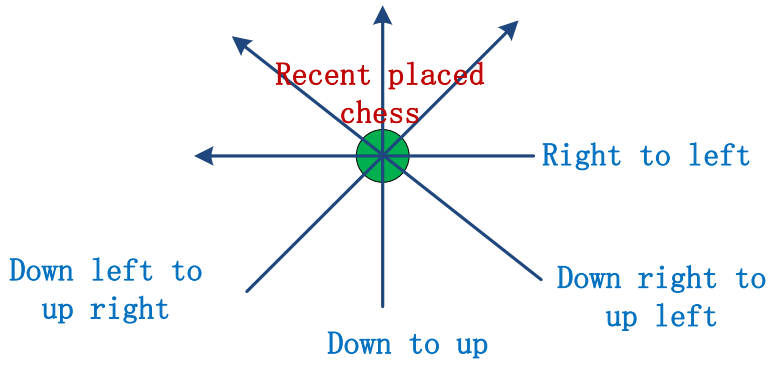

Figure 3.3 The four directions of calculating continuous five green pieces

3.2.5 Defense move of the machine

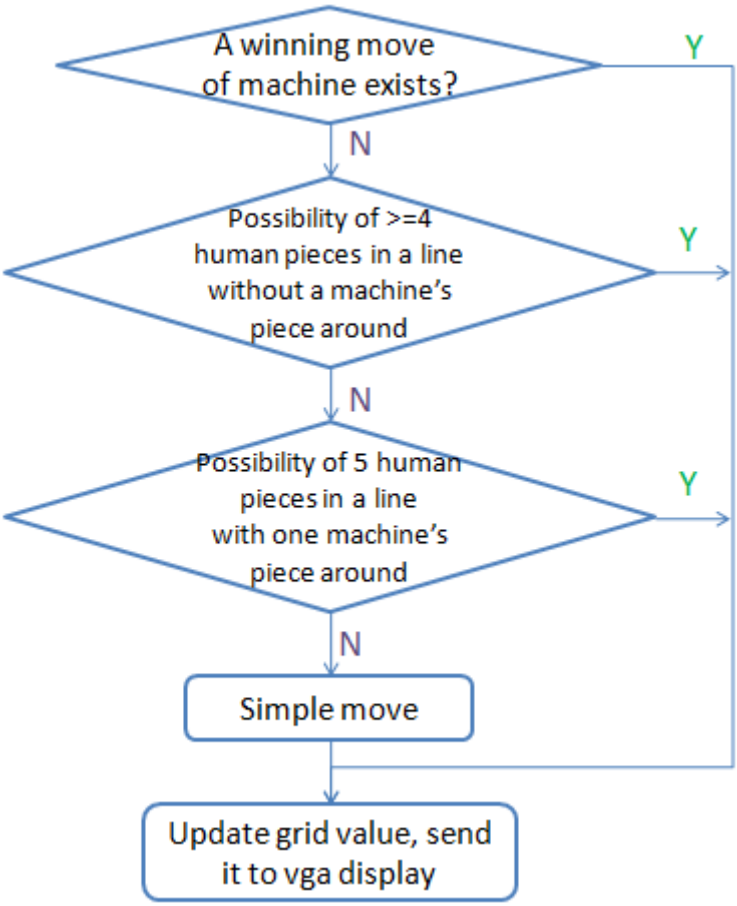

Figure 3.4: Checking procedure of machine's defense move

As shown in Figure 3.4, when it's machine's turn, the machine first check if there is a winning move available for him (i.e. there are four continuous pieces in a line, and a grid next to them is not yet occupied). This check is done by: first, calling the function "*judge_all_direction*" with parameter entry of *man=0* indicating the check operation is for machine's pieces; second, check if either pair of indexes of four directions (right-left, down-up, downright-upleft, downleft-upright) has a sum of four; third, if the previous condition is satisfied then check if there is a blank grid on either side of these four pieces. If there is an straightforward opportunity to win, then the machine make that move regardless of the situation of human pieces for the obvious reason.

If there is not straightforward opportunity available, the machine start to check if there will be a winning move or, more likely, a "dangerous" move that will lead to a winning move on the next round for the man. There are many possible situations of human pieces, which can be categorized according to the existance of the blank grid and machine's pieces. As a line is checked from one side to the other (for the horizontal case check in order of incremental x coordinate; for other cases in order of decremental y coordinate), the number of human pieces (*chess_cnt*), together with the existance of machine's piece (*f_against*) and the existance (*f_blank*) and position (*blank_pos*) of the blank grid are recorded. The *blank_pos* always updates to the location of the latest blank encountered. The continuous blank grids will reset *chess_cnt* and *f_against*; and the existance of machine's piece will reset *chess_cnt* and *f_blank*. Assume there is/are blank grid(s) and no machine's piece before (*f_against=0*), if the number of human pieces accumulate to over three, the machine will place a piece in the blank grid. Assume there is/are blank grid(s) and machine's chess exists, either the condition that *chess_cnt* has accumulated to

four or the condition that *chess_cnt*=3 and the next grid is blank will result in the placement of machine's piece in the blank grid. Similar situations are not listed here. An illustration of the situations mentioned above is shown in Figure 3.5. Based on the location (coordinates) of the newly placed human piece, the conditions are checked in four directions. Once a situation is satisfied, the function *yellow_update* will be called to accomplish the following tasks: update the status value of the grid (in the array *grid*) corresponding to the placed piece; calculate the instruction word for the VGA controller and sent it out through *fsl*.

Figure 3.5: Illustration of different situations of defense

3.2.6 Offense move of the machine

If none situation in four directions is satisfied, then the machine begins to make an offensive move. As the possibility of a winning move has been checked before the defense, the machine just need to check if a critical move can be made, which will bring victory in the next round. The algorithm is similar to the one mentioned above except that the flag and counter are for different type of pieces. For example, assume there is/are blank grid(s) and no human piece before (*f_against=0*), if the number of machine's pieces accumulate to over four, then the machine will place a piece in the blank grid. Since the algorithms for the move of defense and of offense are almost identical, it is realized by a single function with parameter entries which provide the coordinates of the piece (coordinates of human and machine's piece for defense and offense purpose respectively) and the mode (defense or offense). Similarly, an illustration of the situations checked for offense of the machine is shown in Figure 3.6.

Figure 3.6: Illustration of different situations of offense

4 Conclusion

This project was completed in fulfillment of the course requirement of EDA385, Design of Embedded Systems – Advanced Course. It provided a very good opportunity to obtain both the theoretical knowledge and practical experience. It helps us to get more famaliar with the development flow of embedded systems. The result of our project meet both the requirement of the course and our expectation. A competitive game between human user and the machine is realized with keyboard and VGA interface. The system is implemented with a relatively low device utilization and high speed. Our design environment is shown in the following figure:

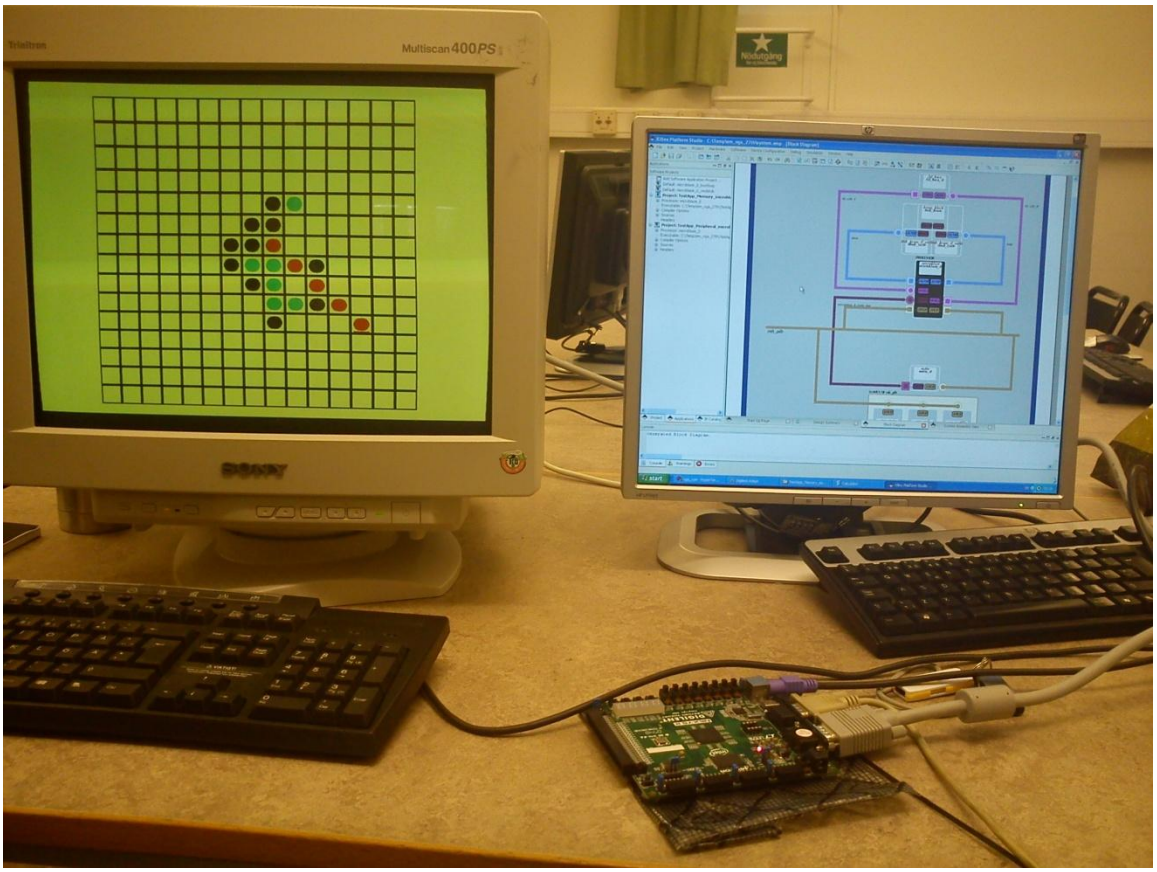

Figure 4.1: Design environment of our project

4.1 Lesson learned

Through this project, we obtain more in-depth knowledge of the embedded system: how its hardware architecture is built, how the communication between processor and other components can be built through the bus interface, how the HW/SW partitioning is done. More importantly, we've got a big picture of how an embedded system is constructed and how it works. Specifically, we faced the memory limitation problem for VGA controller at the beginning, but after we study into the feature of our system, we come up with the solution: : only store the information of the chess board, avoid the memory access and the transmisstion of the bits of all pixels between SW and HW. Another interesting question is: provided the fact that the machine's move is designed by human, can machine win? The answer is: yes, it could win. The reason is: given a comparable degree of intellegence, human usually depends on his/her feeling and can make mistakes; however, machine calculates, it will never make mistakes.

4.2 Further work

- Better visualization

In this project we display chess pieces in single color. A better visualization means we can apply some complex colors with the cost of a more complex display module. If we use SRAM we can also store some pictures in advance such as showing the human's and the machine's turn and call these pictures when needed.

- More intelligent machine – advanced algorithm

So far we have designed a relatively smart algorithm for the machine but there are still many improvement could be done. For example now the machine's offense is just based

on the existed continuous three or four chesses but a more intelligent algorithm can implement offense in richer way.

- Communication for a multi-board game

Our project is only designed to implement a competitive game between human user and machine. An small improvement can be done to introduce a multi-board game.

4.3 Installation and user manual

1. Connect the CRT monitor and the keyboard to the Nexys-2 board through VGA interface and PS/2 interface, respectively;
2. Start Xilinx XPS 12.2, open the project file in XPS; download the bit stream onto the board using Digilent Adept;
3. After the downloading is succeeded, there will be a chess board, in the center of which a cursor appears. Move the cursor using four direction key and place a piece using the Enter key, as shown in Figure 4.2. The Esc key is used for a restart of the game.

Figure 4.2 Useful keys for the game

4.4 Contribution

Hardware:
Zhonghua Wang and Ziyang Li

Software:
Zhonghua Wang and Ziyang Li

Integration and Testing:
Everyone

Report:
Everyone

5 Reference

[1] MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 12.2.
http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/mb_ref_guide.pdf
[2] Digilent Nexys2 Board Reference Manual.
http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf
[3] Driver documentation of Xilinx soft IPs.
"Xilinx\sw\driver\doc\"