ECE 385

Fall 2012

Final Project

Five in Row (Chess Puzzle)

Lida Zhu
ABO/Wednesday 15:00~17:50
Ying he

Objective:

We get the idea from the game Five in Row. It is a chess game with 15x15 block lines. Two players with white chess play and black chess play. Each player plays one by one in turn. Once the line has 5 same color chess, then the person with that color wins immediately.
The rule is to move the cursor. We will use keyboard to take input (W, S, A, D as moving direction of the block), VGA to show on screen. And press space to play the chess at the place. And we make two different level of difficulty, which are a 9x9 table and a 15x15 table. The two players play in turns and win until one got five same color chess in a row no matter in horizontal, parallel or tilting. And we build a competing rule for the game which is when a person wins the game; the last chess will be already played at thereto be the competitor's chess at the second run of the game.

Purpose of the Project

This is the final project of ECE 385. We are asked to use all of the things that we learned before in the previous 10 labs to build a specific circuit to implement interesting functions on the DE2 board. We use monitor to display the out puts, and the keyboard to make inputs. And our project is based on a Mealy machine which the states depend on the inputs. Most of our project is based on the moving function and the display function, so we did a lot of work on graphic and inputs.

General Description of each Circuit Element

When we do the lab project, we divide the whole circuit into several parts, which are easier to design, change and check error; we connect all of them at last. We consider some of the same idea that we did in Lab 9, which is using the keyboard to make bouncing ball moving (here we use the bouncing ball as a moving cursor to show where to play the chess). We borrow the idea of getting input from the keyboard of lab 9, and did some changes to complete the getting of input for the final project. Basically, we can divide the circuit into several parts, which are
Puzzle (moving function and graphic output)

- Chess display (moving function and graphic output, basically in the color mapper, and our chess is moving smoothly without just jump)
- Checking input(this is used to check the input to play chess and remind the previous chess play and color of the chess)In detail, the Puzzle part contains
- Getting input (keyboard and register and clock)
- Check win(detect whether there is a row in 5 same color)
- Counter (display the number of chess that played on the table by using Hex-drive and Integer converter)

In detail, the Chess input part contains chess position chess play, checking and state change. Basically we mark each intersection of the two table line (the place where the chess can be placed into different continuous numbers. For 9x9 table, we define each chess from 0 to 80, in 15x15 we define each chess in 0 to 224). And we predefined chess's coordinates and making them as constant signal in the entity. Then each time when the space button is pressed, the entity will get the output of the position of the cursor from the chess play (the function of the chess moving and chess initial condition are defined in this entity). Then we use this coordinate (two 10 bit binary numbers) to match the constant signal of each chess position. Once one matches, then it output the signal to color map which will describe in next part to ask it to display on screen. Also the color of the chess is vary once the space is been pressed. We build the white

color play and black color play in different states, once space is been pressed, the state will change to another and output the color relates to that player, and when space pressed again, the state reverse back.(black and white swapping)
The keyboard and clock part is used to get the key input from the keyboard; they are almost the same we did in lab9.
In the detail of the Chess play part contains

- Color-mapper: this is the entity that we spend a lot of time on building the graphic stuff, as it contains the checking part of whether to display the chess. As we will get the number of the chess that has been detected, then the entity take the color from the state change entity, then display a ball (we get the function from lab 9) at the place with its right color.
- Moving counter is used to make the moving of the cursor gradually without just jump, so every time when pressing the cursor the ball will move in a constant speed and stop each time when it reach an intersection of two table lines.
- State change: we build the whole system into several different states, and each state has to have specific input to go to the next state, each state is an individual number of outputs.
- VGA controller: this is the entity that used to display the things we make, which is already been given in lab 9

The "checking win" part we did is in a very "stupid" way. Since previously we decide to checking win by taking care of the chess that has been last played, and count in eight directions (up, down, left, right, upper left, upper right, down right, and down left) and count the same color of the chess. Then we add in four ways: left plus right, up plus down, upper left plus down right, upper right plus down left. For the time one of the four gets 5 then the machine halt, and it when to the win state. But finally we failed as the states massed up. So we take a more stupid way but more hard way, which is to consider all possible conditions of win in the table (a lot of logic needed, but we made it, haha).
In the Counter, there is a counter to count how many chess that each player has been played on the table, the hex drive will display the number on the DE2 board.

Description of each Circuit Entity

Color Mapper

Inputs: Reset, clk, state, DrawX, DrawY, chess position, chess size, chess number.
Outputs : RGB
This is one of the most significant entities in the puzzle unit. Also it is one of the most complex one. For this entity we have a lot of internal signals, first each place of chess can be played (81 places) has two signals. One is represent whether there is chess here, the other represent the color of this chess at this place. We build all our graphic display in this entity. We first consider a 9x9 table, with 81 same places of chess to play, at initial the entire place is empty, and the cursor displayed in the middle of the table. We use the center of the screen (191 to 290) along Y axis and (271 to 370) along X axis as our chess table. And we divide it into 81 identical with 30x30 pixels for each block. Then we set the table by defining each black line individually, the rest of color is brown, the back ground color is green. We have 6 different states for the 9x9 play, the first is the welcoming state, second is initial table state, next is white play state, then is black play state, finally are two state of player one win and player two win. So we build each page independently, each graphic picture on the screen are built pixel by pixel (in some place we use

range to ask the computer to give identical color for the region we pointed out). For the playing chess state, we first initialize the entire signal to be zero. The signals here we used is represented for whether there is a chess here. For example, if the number is zero, then there is no chess, if the checking number shows that here should be chess, then we set the signal to be 1, and each signal is related to each place where chess can play. Then as soon as we get the signal transfer from the checking block to tell the place it represented should display the chess, the color mapper set the signal to high and transfers the color signal to the signal represent color also. As there should be a lot of number of chess display at same time, we use “case” logic instead of “if” logic. The case logic is detecting whether the signal represent the chess display is high or low, if it is high, then detect the color signal for the place, if it is high, then display black chess, otherwise display white chess here. The RGB color also defined at this entity.

VGA Controller
It produces the timing signals for horizontal and vertical sync to display the 640 x 480 VGA simulations on a monitor. The code is provided for us and no changes were made.

Keyboard
In this code, there are four states: Start, Press, Hold, and Release. The entity will read the 33 bit input from the “key code” and determine if the key is pressed or hold. If the key is pressed and not release, it will be at the hold state and if the key is released, the state will be changed release state. Since we are reading the input of “W, A, S, D, space” when they are pressed, we only output the code when it is at the press state. This 33 bit code is output to the cursor.

Reg33bits
This is a 33 bit shift register for the “key code” from the ps2 keyboard input. It stores the data input and when a key is entered, it will register the ps2data in it. This only happens when the ps2clk is at falling edge.

512 Clock
A new clock is used to reduce the clock signal from 50MHz to roughly 100KHz.We decrease the signal by 512 and a 10 bit counter is used to check if it reached 512. The counter is incremented for every clock pulse of the 50MHz signal. Once it reaches 512, the new clock will output a high.

D-flip flop
This entity is a D-flip flop. It is used in the falling edge detector entity.

Falling edge detector
This unit detects if the ps2clk is at a falling edge. Once it’s at the falling edge, it will tell the 33 bit register to start storing the ps2data code into its register. Two D flip flops are used to detect the falling edge. The input of the first flip flop is the ps2clk and the output from the first flip flop goes to the input of the second flip flop. They first flip flop is connected to the new reduced clock and the second flip flop is connected to the 50MHz clock. Because ps2clk does not have a clean signal, the first flip flop is connected to the new clock to reduce errors when the falling edge is noisy. The clock for the second flip flop is faster in order to keep the output, data ready,

to be sensitive as we don’t want it to output a high and make the 33 bit register record any extra data.

Counter

This unit is use to detect the number of chess has been placed on the table. We did it by checking the space button.
Input: Button, color choose
Output: hex display
The logic is once the space button is pressed, then we check the color right now, if the color is black then counter 1 increment (the counter to count number of black chess), if the color signal now is low, then counter 2 increment (the counter to count number of white chess). The signal will state the same in other situations, and will be reset when reset signal has been send.

Hex-Driver

The Hex-Driver unit transfer the data output by the counter into the board and display on hex display.

State change

This entity is important, as it controls the moving between each page. And it is the control center of the whole machine.
Inputs: keyboard signal, clk
Output: state number
We are first in the welcoming page (page 0000 State welcoming), then we get through the 9x9 start page (state 1001 State nine) by press 1 on keyboard, then it automatically go to State level1 white with number of 0001 once the “w, a, s, d” is pressed. Or it will through the 15x15 start page (state 1111 State fifteen) if 2 has been pressed. When the W, S, A, D has been pressed, the state went to 0010 for state level 2 white, and cursor start to move. If we pressed the space button, it will change the state from white to black in that level, but without changing the state number output, because the two switching state is used for changing the color signal from black to white. Once the player win the game will move to the end state (state 1110 State win1 which means the white chess win; or state 1101 State win2 which means the black chess win) And the system will halt. If the player wants to play again, just press reset on the DE2 board.

Table nine and Table fifteen

This is the majority part of the project. This entity controls the moving of the cursor and output the position of the chess. We use the space as the detection whether to place the chess. We first define the cursor (which is a ball like in lab 9) of its initial position and moving maximum and minimum also its moving speed. Then we do the same W, A, S, D logic as in lab 9 to ask it to move, but without the bouncing logic. Since we want the cursor to move smoothly without jump, and moving to next block without interrupting, we send a counter enable signal to the step counter when we pressed each key. And the position of the cursor will be updated each time according to the VS clock.

Moving counter and stop detection

These two entities corporate together to make the cursor move gradually, as it contain a counter and a signal detector. Once the counter enable is sent, the counter will automatically count without interruption. The counter value will send to the stop detector each time, once the counter counts to 30 (for 15x15) or 50 for (9x9) then the stop detector will send a stop signal to counter to reset it, and that stop signal will also send to the entity of "table nine and fifteen" to ask the cursor to stop moving. Here is the logic

- Left button (A) pressed: the block on the right of empty block will move left 30 or 50 pixels
- Right button (D) pressed: the block on the left of empty block will move right 30 or 50 pixels
- Up button (W) pressed: the block on the bottom of empty block will move up 30 or 50 pixels
- Down button (S) pressed: the block on the head of empty block will move down 30 or 50 pixels

Pixel Detector for nine and fifteen

This entity is very important, as it make the signal of the coordinate of the chess into digital number. We first predefine each chess coordinate as constant internal signals. When the coordinate value send from the last entity, it will auto match the coordinate with the constant signal. Each chess position has two signals (x value and y value). Once the two values are all matched, the corresponding number will be sent to the color mapper and win detector. For example, when the chess position coordinate is (150, 90), then it matches the number 5 chess on the table, then the entity will output "0000000101" to the next entities.
Inputs: chess positionX, chess positonY
Output: chess number

Detector Win

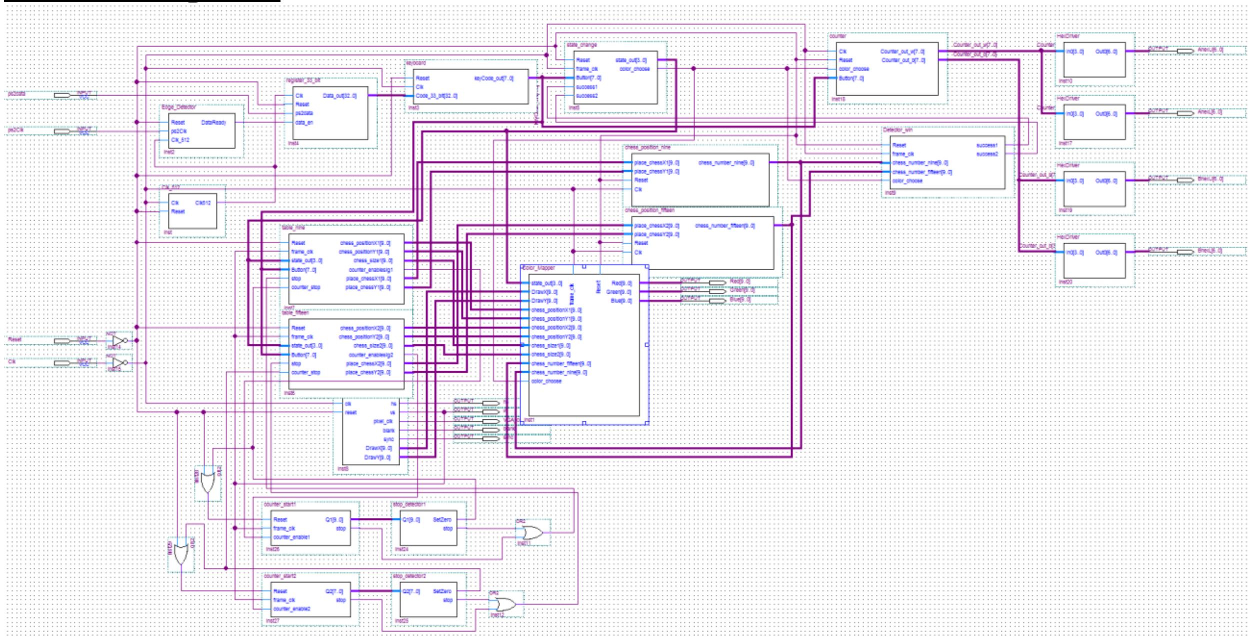
This entity used to detect whether there is a five same color of chess in a row in the horizontal or vertical or even tilting directions. This entity also gets the input from the chess detection of the number of the chess, then it will auto match the number with the internal signal. For example, if the input number is "0000000101", then the entity will set the nine05 (means the fifth chess in 9x9 table) to be high, and at the same time it will give the color value to the nine05_c (means the color signal of the fifth chess in 9x9 table). Then we did in a large typing of all possible situation of winning. For example if (nine00 and nine01 and nine02 and nine04) are all high, then it means there are 5 chess here. Then we detect there color value, like nine00_c and nine01_c and nine02_c and nine03_c and nine04_c. If all of five values are same and high, then we set black win signal to state entity, if they are all same but low, then we sent white win signal to state entity to enter the win states.

Function Description of the Project

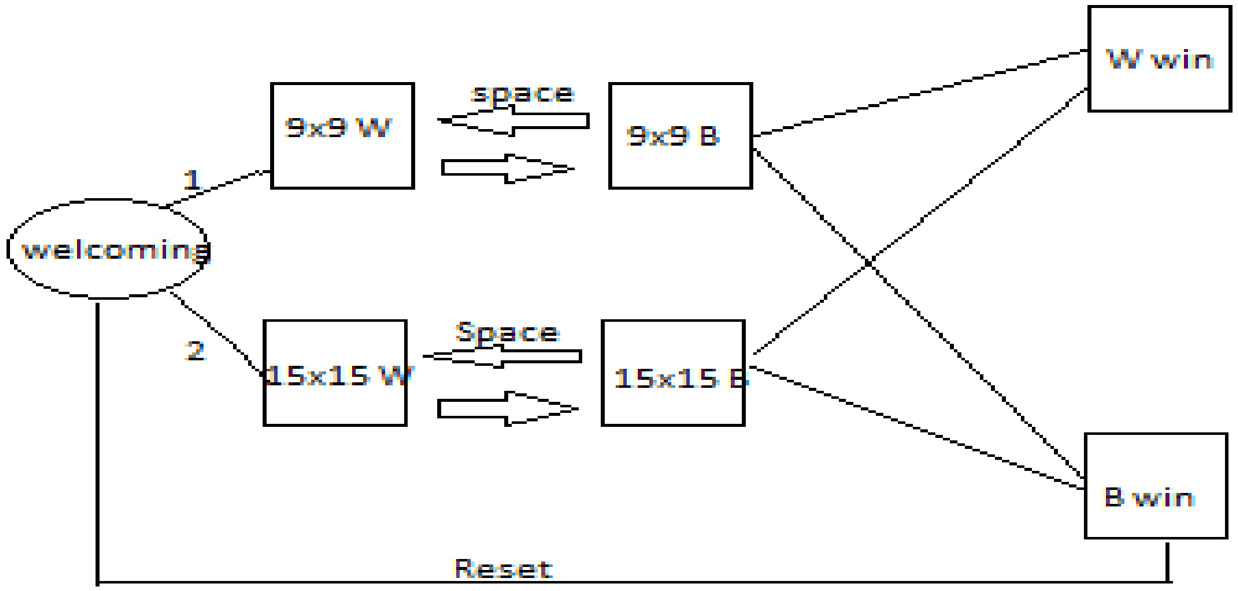
Inputs (button on the keyboard):
W: Move Up
A: Move Left
S: Move Down
D: Move Right
1: Choose 9x9 table at welcoming page
2: Choose 15x15 puzzle at welcoming page
Space: place the chess

Reset: Back to welcoming page
We start at the welcoming page when running the program or press the Reset button. Then we can choose 9x9 or 15x15 at the welcoming page by press 1 or 2 on the keyboard. Then we use W, A, S, D to move the cursor, and press space to place the chess. As soon as one row has 5 same color of chess, the machine halt. If want to play again, just press reset on the DE2 board to go back to the welcoming page.

Block Diagram



State Diagram

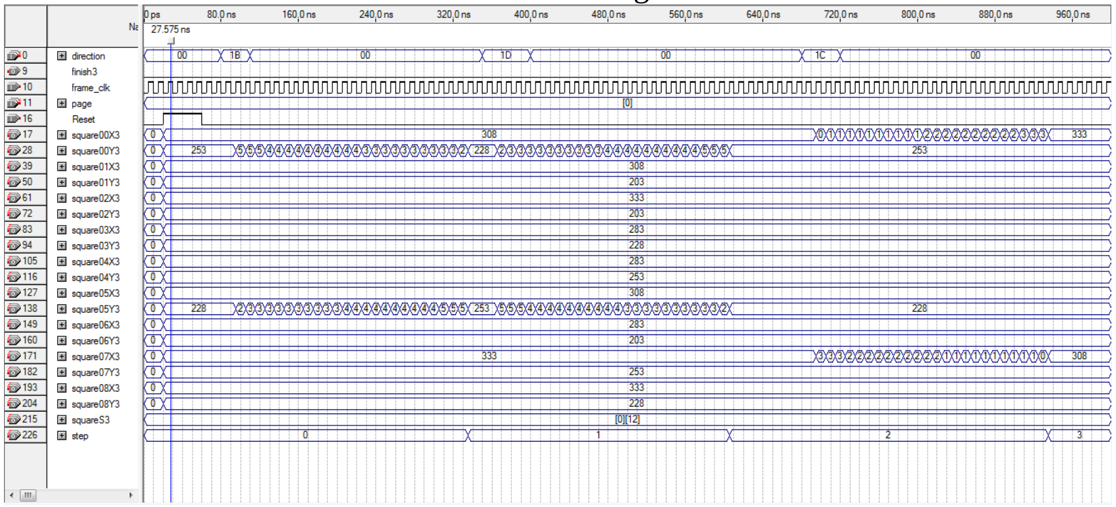


Welcoming State (0000)
Nine State (1001) for 9x9 initial
Level1_w State (0001) for 9x9 white play

Level1_b State (0001) for 9x9 black play
Level2_w State (0010) for 15x15 white play
Level2_b State (0010) for 15x15 black play
Win1 State (1110) for white chess win
Win2 State (1101) for black chess win

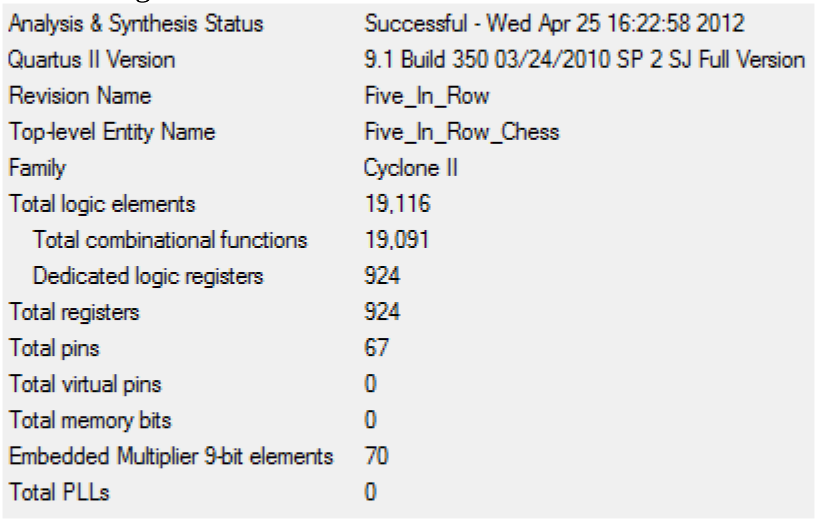
Simulation Diagram

cursor moving simulation



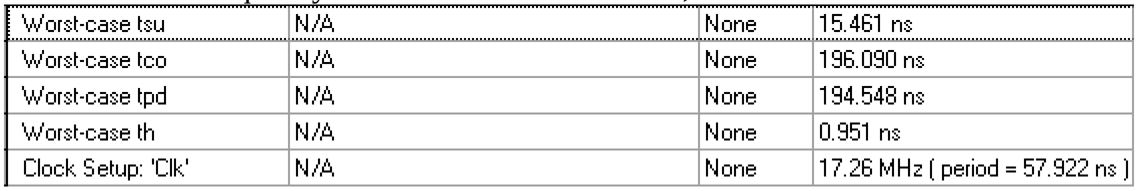
Post-Lab

From the design statistics, our design required 19116 logic elements and 924 total registers, which is shown as following.



Analysis & Synthesis Status	Successful - Wed Apr 25 16:22:58 2012
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Full Version
Revision Name	Five_In_Row
Top-level Entity Name	Five_In_Row_Chess
Family	Cyclone II
Total logic elements	19,116
Total combinational functions	19,091
Dedicated logic registers	924
Total registers	924
Total pins	67
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	70
Total PLLs	0

The maximum frequency of the circuit is 17.26 MHz, as shown below.



Worst-case tsu	N/A	None	15.461 ns
Worst-case tco	N/A	None	196.090 ns
Worst-case tpd	N/A	None	194.548 ns
Worst-case th	N/A	None	0.951 ns
Clock Setup: 'Clk'	N/A	None	17.26 MHz (period = 57.922 ns)

Problem in Project

Our back ground picture is set as green. And the table is set as if logic. The problem is sometimes there are some unexpected lines of color appear randomly (like glitch). Maybe it is because we define the color of background (which is green) is doing by range define, so sometimes the color will be recover by some other definitions. It is better to define every pixel for a fixed color, in order to avoid this happen. But it is too much work load, and the project file will be too huge if we defined every pixel by fixed color at each state.