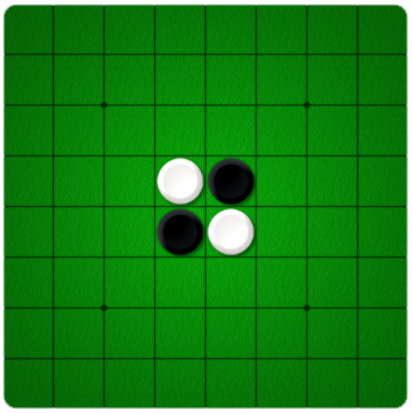
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

EMBEDDED SYSTEMS

PROJECT - AN OTHELLO GAME

REPORT OF MILESTONE 2

10 May 2019
Professor: A. Dollas



Working Group LAB41140573
Vasilakopoulos Panagiotis A.M:2015030048
Maragkaki Maria A.M:2015030153

PROJECT PURPOSE

The purpose of this Project is the implementation of the Othello game where it can play a game against another human player or against another STK500. The Othello game is played on a 8x8 chessboard with cells of the same color. On the horizontal axis the columns are numbered from 1 to 8 and on the vertical axis the lines named from A to H. There are two-colored checkers, black and white. Each player choose one color. At the start of the game 4 checkers are placed in the middle of the chessboard. Two white in the positions D4 and E5 and two black in the positions D5 and E4. The first player places a checker near of the opponent's checker. If the opponent's checkers are closed into the first's player checkers, then they change color. The goal is at the end of the game one of the players to have the most checkers on his color. If the number of checkers is the same then the game is tie. The game ends when all of the squares have a checker or when there are no valid moves.

PURPOSE OF MILESTONE 2

The purpose of the second milestone is to expand the requests of the first milestone where they were, the support of the instruction set, the implementation of the SRAM in memory and the implementation of lighting a LED. This milestone is intended to create a player who is playing without strategy. So requests of this milestone are the following:

- Create a player who finds the legal moves and plays one of them
- The player detects if the opponent played an illegal move and he indicates it
- The player detects if the opponent exceeded the allowable time

The block diagram of the system is presented below.

MEMORY ARRANGEMENT

In comparison with the previous design in Milestone 1, memory alignment has been upgraded. Now, each block of memory contains four positions of a chessboard. Specifically, one block of memory is divided (which is 1 Byte) into two half-Bytes. The most-significant half-Byte contains the colour of the specific checker (setted bit means black and cleared bit means white). The position of each bit represents the corresponding column. The least significant half-Byte contains the corresponding enable-bits, which are showing if the appropriate cell is empty or full. Each line of the table is matched with two positions of memory because each row must have 8 columns. This design reduces the space requirements because now we are using only 16 Bytes for one chessboard, as opposed to the initial alignment, where 64 Bytes will be wasted for the same table. This gives the

advantage to run bigger amount of chess boards. The memory block memory diagram is presented below.

	1	2	3	4	5	6	7	8
A	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
B	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
C	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
D	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
E	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
F	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
G	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8
H	1	2	3	4	E1	E2	E3	E4
	5	6	7	8	E5	E6	E7	E8

Image 1: Memory Diagram

MEMORY INTERFACE

First of all in order to have access in a cell of the chessboard, for example D5, must be transformed the line and the column into the appropriate index of the memory. At start, it is calculated the initial index, subtracting from the ascii code the 65 decimal (which is the ascii code of 'A'). Then this number is shifted logically left one time, which equals with multiplication with 2. This is crucial because according to the design, each row begins in even index numbers (e.g A => index: 0, B => index: 2, C => index: 4). After, if the requested column is greater than 4, the next memory position must be scanned, so the index is incremented by 1 and the requested bit position equals with column - 4. The data of the index are stored into a 8-bit buffer. In order to enter a checker in the table, if the checker is black, the requested bit and the appropriate bit (which is bit + 4) are setted, else only the appropriate enable-bit is asserted in the buffer and then the buffer is saved in the same memory location. This process takes place in the function "setChecker". Respectively,

if a checker must be read, the same process takes place with the difference that only the requested bit and the enable-bit are checked. This is done in the function “readCell”.

FIND THE LEGAL MOVES

The first thing that the program needs to do is to initialize the chessboard. Specifically to place two black checkers in the cells D5 and E4 and two white checkers in cells D4 and E5. This happens with a functions who takes as arguments the line the column and the color of the checker that will be placed.

The second thing is to scan the chessboard with a function who takes as argument the color of the checker which is desirable and find the legal moves. This function is implementing a for loop for all the ram table and find the index with adding the index of the current repetition shifted by 1 (divided by 2) and the number 65 which is the letter A. The shifting happens because every letter in ram is placed in two rows. After finding the index and pumping the particular line, a mask is applied to find in which of the four cells has placed a checker. Afterwards a check is in place to find the line in which it is located and then a function is called which checks if this specific cell is available for this color. This function takes as arguments the row, the column and the colors of the players and look in all directions, right, left, up , down, main and secondary diagonal up and main and secondary diagonal down to find the available cell. If it founds the cell it stores into a table of 8 rows (8-Byte memory space) in which every row is a letter and every bit of the row is the column, so the function stores in the right row and the right bit the valid bit. The same logic is followed to be found the valid cells of the opponent. Finally the player scans the valid table and plays the first valid move he founds by calling the function which is described at the beginning.

CHANGING COLOUR

According to the rules of the othello game, when the move of the player has trapped checkers of the opponent, then the colour of the trapped checkers is changed into the player’s colour. This process takes place in the function “TurnOtherCheckers”. This function takes as arguments the last move that had taken place in the game, the opponent’s colour and the player’s colour. It searches left, right, up, down, main and secondary diagonal up-down from the last move and if it finds trapped opponent's checkers, then inverts the colour.

FIND THE ILLEGAL MOVES

In this phase the program takes the row and the column that opponent plays and then it searches in the enemy's valid table , if this row and column belongs to the available moves. If it doesn't then it sends to the PC's screen the message IL<CR> from the instruction set.

MEASURING TIME

In comparison with the Milestone 1, a crystal oscillator is used at 10 Mhz for clock source. For this reason, the 16-bit timer1 must have completed 39062 steps with a prescaler => clk / 256, in order to achieve one second time-interrupt.
In each command, when the AVR sends the "OK", then the timer starts counting. In the case, where the AVR receives "OK" then timer is disabled and this is feasible by clearing the prescaler. Afterwards, the amount of time of the opponent's move. If this amount is greater than the setted time limit, then the opponent has exceeded the allowable time and the AVR sends illegal Time "IT<CR>" .

BLOCK DIAGRAM

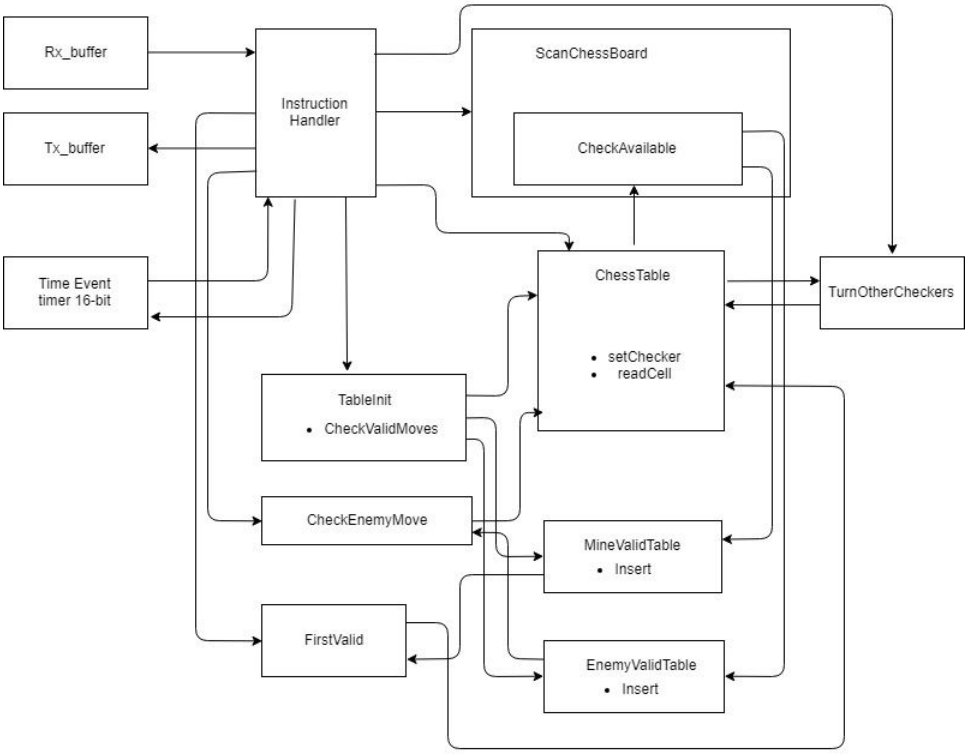


Image 2: System Block Diagram