Reversi

Digital and analog projects - EITF40

Project Report

Group 2
Fredrik Adlercreutz - et08fa4@student.lth.se
Patrick Forsyth - dt08pf4@student.lth.se
Carl-Johan Heinze - ada10che@student.lu.se

2014-03-04

Abstract

We signed up for the course EITF40, Digital and Analogue projects, with the intention of putting previous knowledge to use to develop a product with a combination of both hardware and software. That was what was compelling about this course, to see our project grow from a few components into something we could program according to our wishes.

The actual thing we chose to program is Reversi, which is a strategy game played on an 8x8 uncheckered board.

We encountered some problems along the way, mostly related to understanding the components, but in the end we managed to put together a working product.

Table of contents

1 Introduction

1.1 The game

Reversi is a strategy game played on an 8x8 uncheckered board. Two players take turns placing markers (called disks) with their assigned color facing upwards on the board. One player uses the light side and the other player uses the dark side of the disks. A disk can only be placed where it would create a straight line between two disks of the players own color, with all of the bounded squares containing disks of the other players color. The bounded disks are then turned over to the players color which finishes the current players turn.

The game is over once the last playable square is filled. The winner is the player with most disks of their assigned color.

If a player is unable to place a disk, the turn passes back to the other player. If a position is reached where neither player can legally place a disk, the game is over and the player with the most disks in their own color wins.

1.2 Initial requirements specifications

In the first week of the course, we wrote a basic requirements specification, outlining what we wanted to be achieve during the course.

This is what we came up with:

Grupp 2 - Reversi

Requirements Specification

Functionality requirements

1. Multiplayer game of Reversi
2. Display board
3. Display current score
4. Display current selection (cursor)
5. Display current player
6. Move cursor by using 4 buttons for direction and 1 for selection
7. Sound feedback
8. Light feedback

In case of time (Optional requirements)

1. Store high-score
2. Display high-score list
3. Chess-clock functionality (Game option)

Component list

- ATMega16
- LCD
- 5 buttons
- Speaker
- Lights

As it would later turn out, this list was a bit too ambitious. This is discussed further in the Discussion section of the report.

2 Hardware

2.1 Microcontroller - Atmel ATMEGA16

The ATMEGA16 is an 16 MHz 8-bit, 40 pin microcontroller split between 4 ports, A through D. In our configuration we have configured port A for controlling the LCD-display, port B for sending data to the display, port C controls the LEDs and port D includes the buttons as well as the interrupt handling.

2.2 Debugger/Programmer - JTAG ICE mkII

The JTAG interface provides an easy way to communicate with and debug the program running on the ATMega circuit. Ports 23 - 27 had been reserved for JTAG communication. We used the JTAG controller via USB throughout the development process in order to write our program onto the microcontroller. We also used it for debugging throughout development. With the JTAG connected to a computer with AVR Studio 4, we could set breakpoints or simply step through the executing code line by line or method by method.

2.3 LCD - GDM12864C

We decided to use GDM1286C for displaying the game. It's a 128x64 LCD, which fit us nicely since we wanted the game to have an 8x8 grid, which meant we could use 8x8 pixels for each square. The display is divided into two 64x64 halves, which is selected by setting a certain bit. So to avoid any possible complications, and to reduce complexity, we decided to use the right screen half for displaying the grid and the left screen half for any additional information. For board layout purposes the display was also placed invertedly (flipped on both the vertical and horizontal axis).

The screens internal logic is constructed in a way so that each half of the screen is divided into eight parts along the x-axis meaning that when you write to the screen you will have to write to 8 pixels of the same group and light only the ones of these you want. This is controlled through the data.

2.4 Buttons and LEDs

We used 5 buttons for controlling the cursor and placing a disk. These were constructed as active low electrical switches connected to an AND gate as well as Port C on the microcontroller. The AND gate is in turn connected to an interrupt pin on the microcontroller. A press on the button would trigger an interrupt on the interrupt pin and the corresponding code on the microcontroller would then register which of the buttons had been pressed.

We also put two LEDs on our board used to indicate which players turn it is and to blink in different patterns to alert when a move is not possible and ultimately to show who won the game. The green LED is used for player one (dark disks) and the red LED is used for player two (light disks).

3 Software

We wrote our code in C using AVR Studio 4, which is an IDE designed specifically for use with AVR microcontrollers. In addition to our own code we used the standard avr-lib in order to use AVR specific Ports and functions. The circuit diagram was created with PowerLogic.

The full source code is available in appendix A.

4 Execution

4.1 Planning

The first step of this project course was to decide what project to do and to write a requirement specification. During the first course week there was also a laboration (general introduction) to the ATMEGA16 microcontroller that we all attended. After some deliberation within the group, we decided that a Reversi game would be both fun to do, as well as it allowed us to use different types of interfaces, such as buttons, a display, lights, and sound.

4.2 Construction

We started out by designing a rough circuit diagram, using PowerLogic, which had to be updated several times during the development process due to finding new problems or intricacies that required either a new solution or a different approach when we started to connect the hardware on the board. After connecting the ATMEGA16 to the LCD, LEDs and buttons with their required logic we had to reiterate and reroute some of the wires, partly due to mistakes and partly due to layout optimisations. After most mistakes and rerouting of wires had been performed we applied the voltage and started out on the next part of the project: the software.

4.3 Software development

The first and probably largest trouble we had was to get the LCD to show anything at all, and only after extensive reading and eventual comprehension of the LCDs datasheet, along with a lot of trial and error, we managed to write to the top left corner of the screen and from the knowledge we acquired from this we were able to paint the 8x8 grid. This 8x8 grid is represented by a matrix, aided by a converter that converts coordinates from the matrix to actual pixel positions. Then these coordinates are marked if a player has a disk in that specific location. With every new disk that is placed every square around it is checked for an opponents disk and if that is found we search in that direction for a second disk of the active player and if that is found we flip the opponents disks between these.

5 Results

The result we have achieved is a fully operational Reversi game board for two players. The LEDs give visual feedback to show which players turn it is, and to show when a player does not have any valid moves, and ultimately blink to show who won the game. We also display the current score and indicate whose turn it is on the LCD.

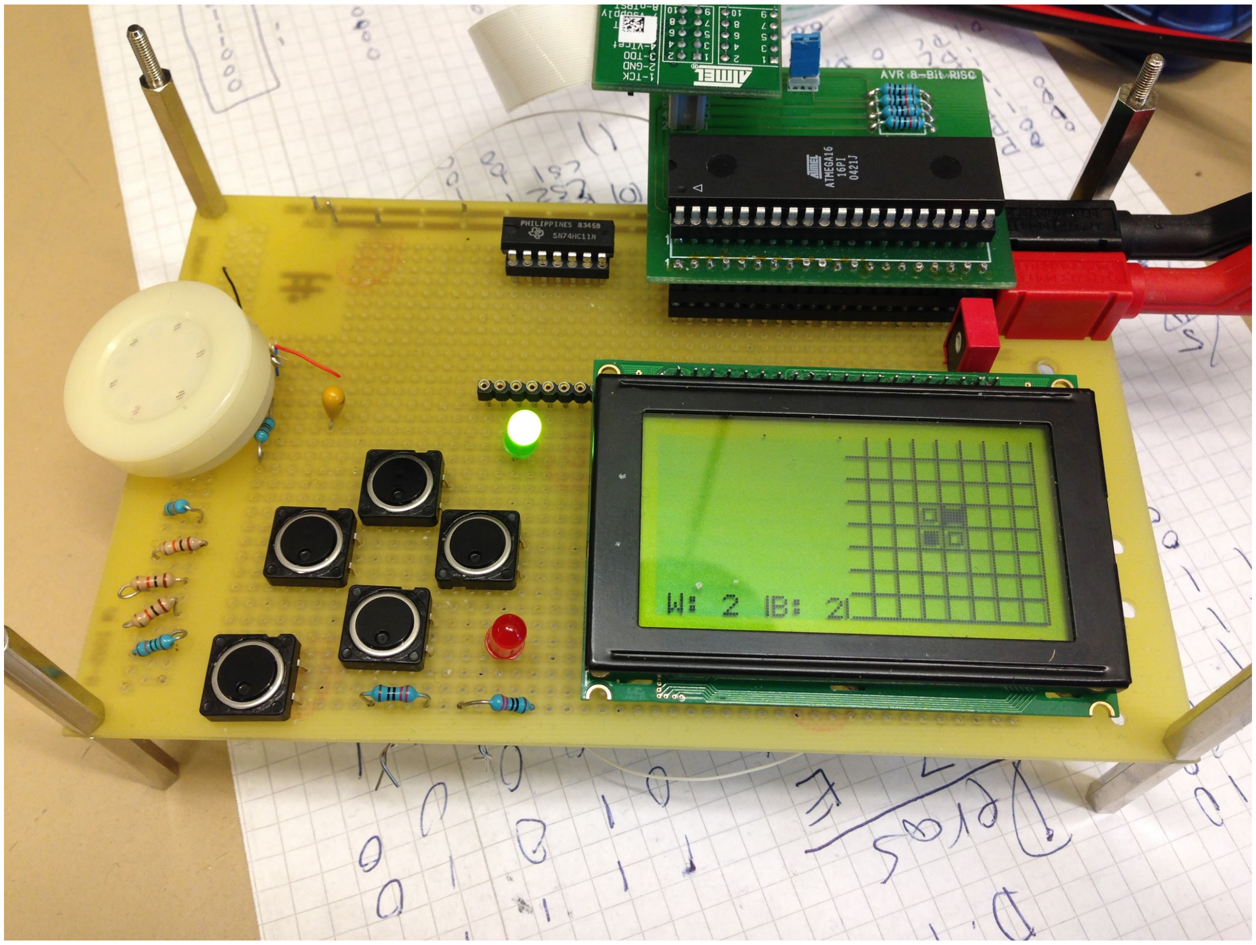


Figure 1. Our board, near the end of the course

6 Discussion

6.1 Issues

While we were not effective enough to implement all the features discussed in our requirements specification on time, we are still fairly pleased that we were able to get the game working in the end. We underestimated the difficulties of getting the hardware working correctly and did not spend enough time working during the first half of the course.

The main problem was related to fully understanding how to control the LCD. As we did not have any experience of how an LCD screen Never having used a display in this sense, the provided datasheet did not elaborate very far on how the write timing works, except for a diagram. After reading a multitude of other datasheets of similar displays, we started to get a larger comprehension of how it worked. After further experimentation we finally managed to write to the screen. This process was the most time-consuming element of the project.

6.2 Further development

The main feature we did not implement was audio feedback. We had planned to use a speaker coupled with a digital-to-analog converter to play sounds for instance when a disk was placed or a player was forced to forfeit a turn due to having no legal moves.

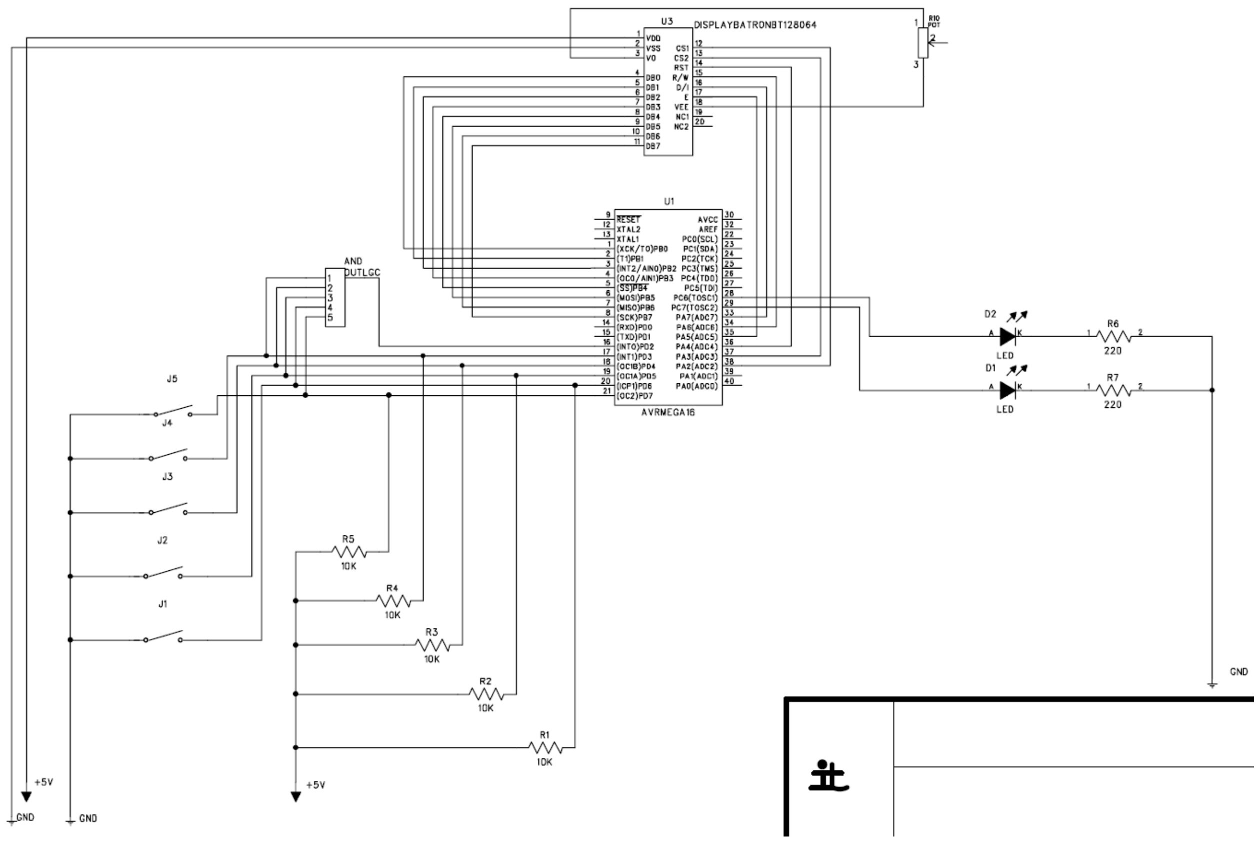
We spent the last few days trying to get the speaker working, but in the end, we ran out of time and at the moment of writing this report, the audio features are not implemented. We estimate that we could get everything in place and working correctly if we spend a few more days working on it.

The optional features what we did not implement were related to high-score functionality and a chess-clock to limit the time a player had to place a disk. This would probably also result in us having to add a menu screen. From this menu screen the chess clock feature could be activated and the high-score could be displayed. In order for us to be able to store the high-score between sessions we would also have to look into how to writing and reading from the EEPROM.

6.3 Comments and experiences

This course has been enjoyable and useful. We all feel that we have gained valuable experience. Few other courses focus on both hardware and software simultaneously, and it provides extra satisfaction to see the program you wrote being executed on hardware you yourself put together. We have gotten useful reminders of bitwise operations, soldering and circuit design. Some of us had not done much coding in C before, so writing the program also provided useful experience.

A Circuit diagram



B Source code

```
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>

// LCD <=> AVR connections:
//
// D:   PORTB
// D/I: PA7
// R/W: PA6
// E:   PA5
// RS:  PA4
// CS2: PA3
// CS1: PA2
//


#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
```

```
#define Y_MIN 0x40
#define Y_MAX 0x3F
#define _NOP()  asm volatile("nop\n");
#define X_MIN 0xB8
#define X_MAX 0xBF

unsigned short int xPos;
unsigned short int yPos;
unsigned int allMarkers[8][8];
unsigned int aPos;
unsigned int bPos;
unsigned short int drawX = 0xB8;
unsigned short int drawY = 0x42;
unsigned short int player;
unsigned short int skipped;
unsigned short int score1;
unsigned short int score2;

void wait()
{
        int i = 0;
        while(i < 32)
        {
                i++;
        }
}

void toggle()
{
        PORTA = PORTA | 0x20; //Or med "Toggle hög" (1 0 0 0 0 0 0 0 0) => E hög
        PORTA = PORTA & 0xDF; //And med inverterad "Toggle hög" (0 1 1 1 1 1 1 1 1) => E låg //7F
        PORTA = PORTA | 0x20; //Or med "Toggle hög" (Se första steget) => E hög
}

void displayOn()
{
        PORTB = 0x3F;
        PORTA = 0x3C; //9C
        toggle();
}

void displayOff()
{
        PORTB = 0x3E;
        PORTA = 0x3C; //9C
        toggle();
}

void convertGridToCord(int a,int b) {// konverterar en koordinat till pixelkoordinater
        drawX = 0xB8+(0x01*a);
        drawY = 0x42+(0x08*b);
}

void convertCordToGrid(int x, int y){
        aPos = x - 0xB8;
        bPos = (y - 0x42)/0x08;
}

void clearDisplay()
{
        for(int k = 0xB8; k <= (0xB8 + 0x07); k++)
        {
        PORTA = 0x3C; //Markera båda skärmhalvorna //var 9C
        PORTB = k; //Markera den aktuella byten
        wait();
        toggle();
        PORTB = 0x40; //Markera längst till höger
        wait();
        toggle();
        PORTA = 0xBC; //Markera för utskrift //var DC
        PORTB = 0x00; //Rensa markerade pixlar
        for(int i = 0; i < 64; i++)
                {
//              wait();
                toggle();
                }
        }
}

void clearScoreDisplay()
{
        for(int k = 0; k <= 16; k++)
        {
        PORTA = 0x38; //Markera båda skärmhalvorna //var 9C
        PORTB = k; //Markera den aktuella byten
        wait();
        toggle();
        PORTB = 0x40; //Markera längst till höger
        wait();
        toggle();
        PORTA = 0xB8; //Markera för utskrift //var DC
        PORTB = 0x00; //Rensa markerade pixlar
        for(int i = 0; i < 64; i++)
                {
//              wait();
                toggle();
                }
        }
}



void write(int x, int y, int data, int displayScreen)
{
        if(displayScreen == 1)
        {
                PORTA = 0x34; //34
        } else if (displayScreen == 2) {
                PORTA = 0x38; //98
        }
        //wait();
        //toggle();
        PORTB = y; //y
        wait();
        toggle();
        PORTB = x ; //x
        wait();
        toggle();

        if(displayScreen == 1)
        {
                PORTA = 0xB4; //D4
        } else if (displayScreen == 2){
                PORTA = 0xB8; //D8
        }
        PORTB = data;
//      wait();
//      toggle();

        PORTA = 0x30; //90
        wait();
        toggle();
}

void drawCursor() {
        int x=xPos;
        int y=yPos;
        for(int i=0x00;i<=0x06;i+=0x01){
                if(i == 0x00 || i == 0x06)
                        write(x,y-0x01+i,0xFF,1);
        }



}
void drawBlackMarker(int x, int y) {

        for (int i = 0; i < 5; i++) {
                write(x,y+i,0xBE,1);
        }
}

void drawWhiteMarker(int x, int y) {

        for(int i=0x00;i<=0x04;i+=0x01){
                //write(x,y,0xBE,1); Svart bricka (Y)
                if(i == 0x00 || i == 0x04){
                        write(x,y+i,0xBE,1);
                } else {
                        write(x,y+i,0xA2,1);
                }
        }
}


void drawChar(int x, int y, int num) {
        switch(num) {
                case 0:
                        write(x,y,0x00, 2);
                        write(x,y,0x7C, 2);
                        write(x,y,0x8A, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0xA2, 2);
                        write(x,y,0x7C, 2);
                        break;
                case 1:
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0xFE, 2);
                        write(x,y,0x40, 2);
                        write(x,y,0x20, 2);
                        write(x,y,0x00, 2);
                        break;
                case 2:
                        write(x,y,0x00, 2);
                        write(x,y,0x62, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x8A, 2);
                        write(x,y,0x86, 2);
                        write(x,y,0x42, 2);
                        break;
                case 3:
                        write(x,y,0x00, 2);
                        write(x,y,0x6C, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x82, 2);
                        write(x,y,0x44, 2);
                        break;
                case 4:
                        write(x,y,0x00, 2);
                        write(x,y,0x08, 2);
                        write(x,y,0xFE, 2);
                        write(x,y,0x48, 2);
                        write(x,y,0x28, 2);
                        write(x,y,0x18, 2);
                        break;
                case 5:
                        write(x,y,0x00, 2);
                        write(x,y,0x9C, 2);
                        write(x,y,0xA2, 2);
                        write(x,y,0xA2, 2);
                        write(x,y,0xA2, 2);
                        write(x,y,0xE4, 2);
                        break;
                case 6:
                        write(x,y,0x00, 2);
                        write(x,y,0x4C, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x7C, 2);
                        break;
                case 7:
                        write(x,y,0x00, 2);
                        write(x,y,0xE0, 2);
                        write(x,y,0x90, 2);
                        write(x,y,0x8E, 2);
                        write(x,y,0x80, 2);
                        write(x,y,0x80, 2);
                        break;
                case 8:
                        write(x,y,0x00, 2);
                        write(x,y,0x6C, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x6C, 2);
                        break;
                case 9:
                        write(x,y,0x00, 2);
                        write(x,y,0x7C, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x64, 2);
                        break;
                case 10:        //10 = B
                        write(x,y,0x00, 2);
                        write(x,y,0x6C, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0x92, 2);
                        write(x,y,0xFE, 2);
                        break;

                case 11:        //11 = W
                        write(x,y,0x00, 2);
                        write(x,y,0xFE, 2);
                        write(x,y,0x04, 2);
                        write(x,y,0x18, 2);
                        write(x,y,0x04, 2);
                        write(x,y,0xFE, 2);
                        break;

                case 12:        //12 = Colon-sign
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x6C, 2); //6C
                        write(x,y,0x6C, 2); //6C
                        write(x,y,0x00, 2);
                        break;

                case 13:        //13 = Space
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        break;

                case 14:        //14 = | marker (Active player)
                        write(x,y,0x00, 2);
                        write(x,y,0x7C, 2);
                        break;

                case 15:        //15 = small-space (Not active player)
                        write(x,y,0x00, 2);
                        write(x,y,0x00, 2);
                        break;
                }



/*
                //Conversion chart to flip char-segments
                        3E -> 7C
                        42 -> 42
                        61 -> 86
                        51 -> 8A
                        49 -> 92
                        46 -> 62
                        00 -> 00
                        04 -> 20
                        02 -> 40
                        7F -> FE
                        18 -> 18
                        14 -> 28
                        12 -> 48
                        36 -> 6C
                        41 -> 82
                        22 -> 44
                        39 -> 9C
                        45 -> A2
                        27 -> E4
                        32 -> 4C
                        07 -> E0
                        09 -> 90
                        01 -> 80
                        71 -> 8E
                        26 -> 64
                        10 -> 80
*/
```

```
}

void drawScore() {
        clearScoreDisplay();
        int b1 = score1/10;
        int b2 = score1%10;
        int w1 = score2/10;
        int w2 = score2%10;

        //PLAYER ONE TEXT///////////////////////

        if (player==1) {
                drawChar(16,0,14); //|
                } else {
                drawChar(16,0,15); //small-space
        }

        drawChar(16,0,b2);
        if (b1 >= 1) {
                drawChar(16,0,b1);
                } else {
                drawChar(16,0,13); //space
        }

        drawChar(16,0,12); //colon-sign
        drawChar(16,0,10); //B

        if (player==1) {
                drawChar(16,0,14); //|
                } else {
                drawChar(16,0,15); //small-space
        }
        /////////////////////////////////////////

        drawChar(16,0,13); //Space

        //PLAYER TWO TEXT///////////////////////

        if (player==2) {
                drawChar(16,0,14); //active-marker
                } else {
                drawChar(16,0,15); //small-space
        }


        drawChar(16,0,w2);
        if (w1 >= 1) {
                drawChar(16,0,w1);
                } else {
                drawChar(16,0,13); //space
        }
        drawChar(16,0,12); //colon-sign
        drawChar(16,0,11); //W
        if (player==2) {
                drawChar(16,0,14); //active-marker
                } else {
                drawChar(16,-7,15); //small-space
        }
        /////////////////////////////////////////

}


void drawAllMarkers() {
        score1 = 0;
        score2 = 0;
        for (int k = 0; k <= 7; k++) {
                for (int l = 0; l <= 7; l++) {

                        if (allMarkers[k][l] == 1) {
                                convertGridToCord(k,l);
                                score1++;
                                drawBlackMarker(drawX, drawY);
                                //      converter(k,l);
                        //      for (int i = 0; i < 5; i++) {
                                        //write(k,l+i,0xBE,1);
                        //              write(drawX,drawY-i,0xBE,1);
                                //}
                        }
                        else if (allMarkers[k][l] == 2) {
                                convertGridToCord(k,l);
                                score2++;
                                drawWhiteMarker(drawX,drawY);
                        //      for (int i = 0; i< 5; i++) {
                        //              if (i == 0 || i == 4) {
                        //                      write(k,l-1, 0xFF, 1);
                        //              } else {
                        //                      write(k,l-1, 0xC1, 1);
                        //              }
                        //      }
        //              }
                        }
                }
        }
}


void redraw()
{
        //clearDisplay();
        int i;

        for (i=X_MIN; i <= X_MAX ; i+=0x01) {

                int y = Y_MIN;
                int j;
                for (j = 0; j<64; j++){
                        if((j % 8) == 0){
                                write(i, y, 0xFF, 1);
                        } else {
                                write(i, y, 0x80, 1);
                        }
                        y += 0x01;
                }

        }
        drawAllMarkers();
        drawCursor(xPos,yPos);

//      PORTC=0x40;

}

void switchPlayer(){
        if(player == 1){
                player = 2;
        //      PORTC =0x80;
                PORTC=(0xC0 ^ PORTC);//SPEAKERTEST
        } else {
                player = 1;
                //PORTC =0x40;
                PORTC=(0xC0 ^ PORTC);//SPEAKERTEST
        }
}

int movePossible() {
        int opponent = (player == 1)? 2 : 1;
        for (int a = 0 ; a <= 7 ; a++) {
                for (int b  = 0 ; b <= 7 ; b++) {
                        if (allMarkers[a][b] == 0) {
                                int deltaA;
                                int deltaB;
                                for (deltaA = -1; deltaA <= 1; deltaA++) {
                                        for(deltaB = -1; deltaB <= 1; deltaB++){

                                                if (a+deltaA > 7 || a+deltaA < 0 ||  b+deltaB > 7 ||
b+deltaB < 0 || (deltaA == 0 && deltaB == 0)){
                                                        continue;
                                                }

                                                if (allMarkers[a+deltaA][b+deltaB] == opponent){

                                                        int nextA = a+deltaA;
                                                        int nextB = b+deltaB;

                                                        for(;;){

                                                                nextA += deltaA;
                                                                nextB += deltaB;

                                                                if(nextA > 7 || nextA < 0 || nextB >
7 || nextB < 0){
                                                                        break;
                                                                        // Out-of-bounds, not valid
                                                                }

                                                                if(allMarkers[nextA][nextB] == 0){
                                                                        break;
                                                                        // Empty, not valid
                                                                }
                                                                if(allMarkers[nextA][nextB] ==
player){
                                                                        return 1;
                                                                        // Valid move found
                                                                }
                                                                break;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
        return 0;
}



void placeBrick() { //kollar om draget är giltligt och placerar då ut bricka
        int movesPerformed = 0;
        int opponent = (player == 1)? 2 : 1;
        int a = aPos;
        int b = bPos;
        if (allMarkers[a][b] == 0) {
```

```
		int deltaA;
		int deltaB;
		for (deltaA = -1; deltaA <= 1; deltaA++) {
			for(deltaB = -1; deltaB <= 1; deltaB++){


				if (a+deltaA > 7 || a+deltaA < 0 ||  b+deltaB > 7 || b+deltaB < 0 ||
(deltaA == 0 && deltaB == 0)){
					continue;
				}


				if (allMarkers[a+deltaA][b+deltaB] == opponent){

					int nextA = a+deltaA;
					int nextB = b+deltaB;

					for(;;){

						nextA += deltaA;
						nextB += deltaB;

						if(nextA > 7 || nextA < 0 || nextB > 7 || nextB < 0){
							break;
							// Out-of-bounds, not valid
						}

						if(allMarkers[nextA][nextB] == 0){
							break;
							// Empty, not valid
						}
						if(allMarkers[nextA][nextB] == player){

while(allMarkers[nextA-=deltaA][nextB-=deltaB] == opponent){
								allMarkers[nextA][nextB] = player;
								movesPerformed++;
							}
							break;
							// Valid move found, traverses back and flips
the disks
						}



					}
				}
			}
		}
	}

	if(movesPerformed != 0){
		allMarkers[aPos][bPos] = player;
		skipped = 0;
		switchPlayer();
	}
	drawAllMarkers();
}


void removeCursor() {

	int x = xPos;
	int y = yPos;
	for(int i = 0x00; i <= 0x06; i+= 0x01) {
		if (i == 0x00 || i == 0x06) {
```

```
                        write(x,y-0x01+i,0x80,1);
                }
        }

}

void newGame() {
        xPos = 0xBC;
        yPos = 0x5A;
        aPos = 4;
        bPos = 3;

        memset(allMarkers,0, sizeof allMarkers);//resets allMarkers to zeros
        // Wait a little while the display starts up
        wait();


        //      SREG = 0xFF;

        //PORTC=0x40; //LED GREEN

        PORTC=(0x40 | PORTC);//SPEAKERTEST

        displayOn();
        clearDisplay();
        allMarkers[3][4] = 1;
        allMarkers[4][3] = 1;
        allMarkers[3][3] = 2;
        allMarkers[4][4] = 2;
        player = 1;
        skipped = 0;
/*      allMarkers[5][4] = 1;
        allMarkers[5][3] = 1;
        allMarkers[6][3] = 1;
        allMarkers[6][4] = 1;
        allMarkers[2][3] = 1;
        allMarkers[2][3] = 1;
        allMarkers[2][4] = 1;

*/
        //drawAllMarkers();
        redraw();
}

void checkWinner() {
        if (score1 > score2) {
                for(int i = 0 ; i < 5 ; i++) {
                        PORTC=0x00;
                        _delay_ms(300);
                        PORTC=0x40;
                        _delay_ms(300);
                }
        } else if (score2 > score1) {
                for(int i = 0 ; i < 5 ; i++) {
                        PORTC=0x00;
                        _delay_ms(300);
                        PORTC=0x80;
                        _delay_ms(300);
                }
        } else {
                for(int i = 0 ; i < 5 ; i++) {
                        PORTC=0x00;
                        _delay_ms(300);
                        PORTC=0xC0;
                        _delay_ms(300);
                }
        }

}

//void cursor(left,right,up,down) {}

ISR(INT0_vect) {

        unsigned short int input = PIND;

        _delay_ms(150);
        _NOP();
        switch(input) {//enter FUNKAR
                case 0xBB:
                        //PORTC = (0x01 | PORTC);
                        placeBrick();
                        drawScore();
                        if(!movePossible()) { //kolla nu movePossible för nästa spelare
                                //skipped++;
                                //if (skipped == 2) {
                                //      PORTC=0xC0;
                        //              _delay_ms(3000);
                        //              checkWinner();
                //                      newGame();
                //              } else {
                                        PORTC=0xC0;
                                        _delay_ms(1000);
                                        switchPlayer();
                                        if(!movePossible()){
                                                PORTC=0xC0;
                                                _delay_ms(3000);
                                                checkWinner();
                                                newGame();
                                        }
                                        drawScore();

                        }
                        break;

                case 0xF3://right Funkar
                        if (yPos <= 0x42) {
                        break;
                        }
                        else
                                removeCursor();
                                yPos-= 0x08;
                                bPos--;
                                drawCursor();
                                //redraw();
                                //cursor(1,0,0,0);
                        break;

                case 0xDB://left FUNKAR
                        if (yPos >= 0x7A) {
                                break;
                        }
                        else
                                removeCursor();
                                yPos+=0x08;
                                bPos++;
                                drawCursor();
                                //redraw();
                        //cursor(0,0,0,1):
                        break;
                case 0xEB://down FUNKAR
                        if (xPos <= 0xB8) {
                                break;
                        }
                        else
                                removeCursor();
                                xPos -= 0x01;
                                aPos--;
                        //      redraw();
                                drawCursor();
                        //cursor(0,1,0,0);
                        break;

                case 0x7B://up
                        if (xPos >= 0xBF) {
                                break;
                        }
                        else
                                removeCursor();
                                xPos += 0x01;
                                aPos++;
                        //      redraw();
                                drawCursor();
                        //cursor(0,0,1,0);
                        break;
        }

}


int main(void) {
        //INIT
        DDRA = 0xFF;
        DDRB = 0xFF;

        DDRD = 0x00;
        DDRC=0xFF;

        GICR = 0x40;
        MCUCR = 0x02;
        newGame();
        drawScore();

        sei();
        while(1);

}
```