Tic-Tac-Toe

Digital and Analogue Projects, LTH

Authors:
Joakim Ahle - ada09jah@student.lu.se
Anton Botvalde - gda07abo@student.lu.se

Mentor:
Bertil Lindvall - Bertil.Lindvall@eit.lth.se

Lund, 140304

Introduction

In the Digital and Analogue project course [1], the students should build a complete embedded system from a set of components. The students start with crafting a requirements specification for the project, where the desired end-behavior of the product is defined.

When the specification is done and the mentor approved it, it's time to select components and make a block diagram, showing how the components should be connected.

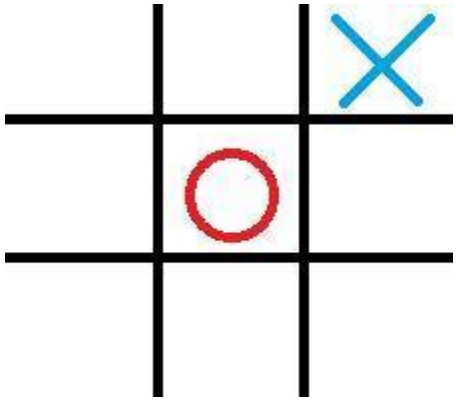
After the block diagram has been improved, the components should be connected and the program that matches the requirements sheet should be implemented.

We choose to make a Tic-Tac-Toe game, where two players can compete against each other. Each player has a direction pad that they use to position either a ring or a cross onto a 3x3 grid. The game is won when either player achieves to place three symbols in a row. If every tile has a symbol, but no player has three in a row, it's a tie.

From the very beginning, we pictured the final product as an early 80's game console.

Specification of requirements

We want to build a tic-tac-toe, featuring 9 tiles where two players want to put three of their respective symbols in a row. If one player manages that, the game is won by that player and if nine tiles are filled and no player has three in a row, it's a tie.



Tic-tac-toe board

The construction should be built with an ATMega16 microcontroller, a graphic LCD display and one steering cross for each player. There should also be a reset button and some way of showing which player's turn it is.

Hardware

The core component of the end-product is the processor, an ATMega16(L) High-performance AVR 8-bit Microcontroller [2]. It has 40 pins for connecting various components and a RISC architecture.

To play the game, each player needs five buttons. To not waste too many pins on this, a MM54C922 16-Key Encoder [3] was used. What it does is mapping up to 16 buttons to a unique four digit binary number, for later identification by the software. It's specifically designed to work on a data bus even though our project didn't need to use it. When a button is pressed, it sends a signal to an interrupt port on the processor. For controlling the internal behavior of the encoder, two oscillators have to be connected. The oscillators differ from each other with a factor of 10. This control for example the minimum time between key presses. The construction effectively handles key bounces.

The game is played on a LCD screen, the GDM12864C [4]. The screen is divided in two halves, called IC1 and IC2 respectively. For our project, it meant that we could have the game board on half, and game information on the other half. The screen is divided into 8 pages on the x-axis and one page consists of 8 pixels. To write pixels to a page, you "simply" write one data word into the corresponding address in the display data RAM. More detailed, you have to send a few control signals to the screen in the right order and timing, before the data can be placed in the RAM. Before you write data to the RAM, you have to specify the address through other instructions (one for y-coordinate and one for x-page).

The link between the computer where the code is written and the construction is a standard interface called JTAG [5]. It takes four pins on the processor to function, but helps sending debug data back to the AVR studio 4 explained under "software".

The complete system is shown on a block diagram in Appendix A.

Software

All the code in the project was written in the AVR Studio 4 environment. As well as a code editor, the program also provides debug tools and helps compile, link and transfer the executable program to the construction through JTAG.

The software itself relies on a small library from AVR, which includes the lowest level communication with the processor. For example naming of ports and delays if the program needs to use that. All code written in this project is in C.

Extractions of the code is shown in Appendix B. The program works roughly this way: A minimalistic main method calls methods to initiate variables, screen and interrupts before it enters a never ending loop calling the update method. If a button has been pressed, the update function handles it by moving tiles, eventually switching players and check for game-over. The external interrupt simply saves the data word corresponding to the pressed button, and flags that a button has been pressed. Except the external interrupt, we have an internal timer interrupt, used to control the blinking of a symbol before it's placed.

The by far largest file is the "lcd.c" file. It has a bit over 800 lines of code. Most of these are to print hard coded data to the screen, but a substantial part is the drivers used to write a word to a screen page. As explained in the "hardware section", the screen needs a few control signals in order to place the data in the RAM. to control this, we wrote a clever C style Macro that expands into the proper control signals, depending on what you want to do.

Results

In the end, we managed to build a decent tic-tac-toe game despite a lot of difficulties on the way. Most of the problems we had during the development was with the screen.

Many of the problems originate from the data sheet of the display which follows a very low standard. The very simple use case of writing one pixel to a specified position of the screen takes many hours of work to complete. The problems include, but are not limited to:

- The Display Data RAM, for some reason, stores the pixels written to it after a hard reset, Since it's virtually impossible to clear the screen until you know exactly how it works, determining the effects of the latest change in the source code is very hard.
- Parts of the data sheet seems to be written using google translate.
- Either, the coordinate system of the screen is reversed, i.e that the Y-axis is horizontal or nothing on the display or data sheet specifies what is up and down on it.
- The page partitioning of the screen makes it extremely hard to write any generic shape spanning several pages.

Due to the problem in the last bullet we concluded that, given that every LCD screen have a similar partitioning, writing something to the screen without hard coding or putting strict constraints on it, requires a lot of work. We came to the conclusion that that is done by reading the display data RAM and do bite wise OR with the new data. The paging problem would still require a serious algorithm though.

We choose to use the right half of the screen since it would have increased the complexity a lot to write a board centered over two pages, but we really liked the end result with split screens. Something we unfortunately didn't have time for was to keep the score in memory when the system is restarted. The reason for is that we would have to connect another component, an EEPROM, to the processor.

The key encoder specified that it should take care of key bounces[1] when a button was pressed. We think that it handled it really well, but it seems like a few key bounces slipped through.

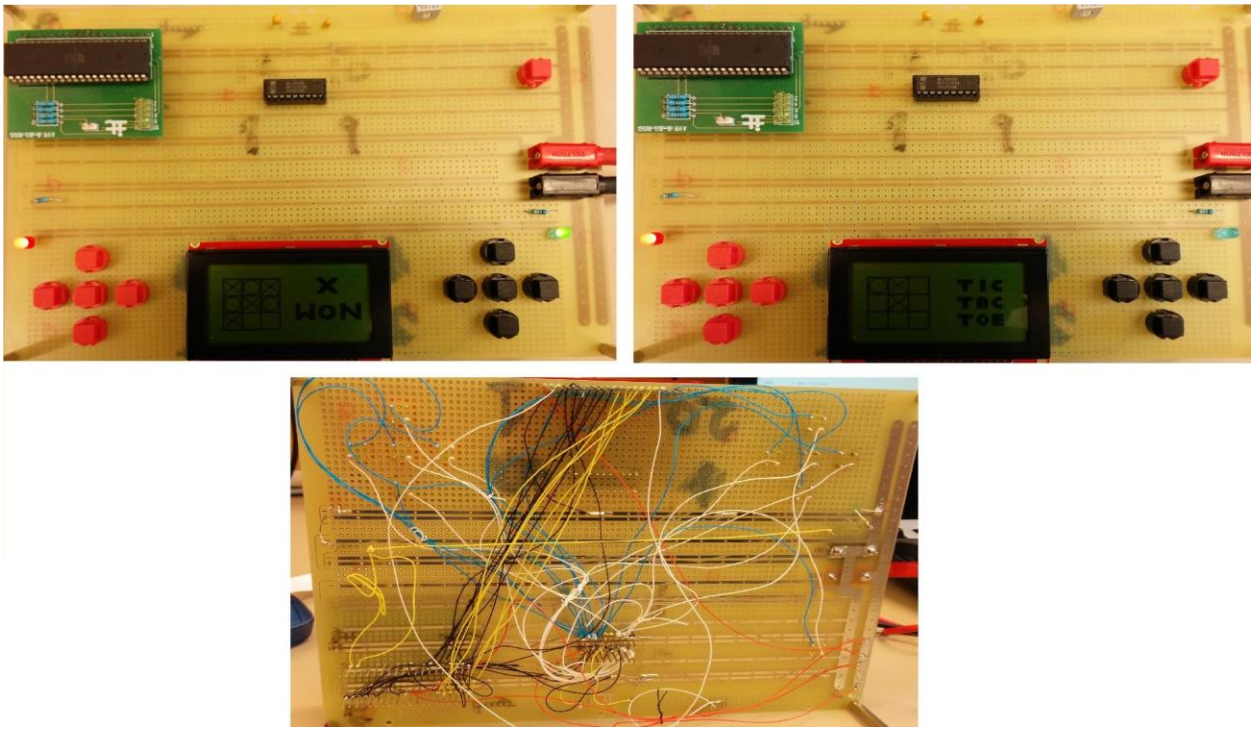
Even though the processor only had a small amount of RAM, it never affected us. This is either because we are extremely memory efficient in our program, our simply because it's not necessary to allocate a lot of memory for this kind of problem.

[1] http://en.wikipedia.org/wiki/Bounce_keys

The course has taught us a lot about building an embedded system from the ground. We wrote a list of what we think was the most important lessons we learned:

- ❏ Reading data sheets for components, and how to solve problems regarding their incompleteness.
- ❏ Excellent training in writing C code for an embedded system
- ❏ How low level interrupts are handled and used to make an efficient solution
- ❏ How an LCD display work and the problems of programming it
- ❏ How to derive a final product from a small set of requirements
- ❏ The problems with debugging an embedded system

As we wrote in the introduction, we pictured the final product as an 80's game console, and the end result could very well be just that, if it was wrapped in some shiny plastic case. We declare success!



It's not pretty, but it does the job.

References

[1] Digital and analogue projects
Course Requirements
http://kurser.lth.se/kursplaner/13_14%20eng/EITF40.html

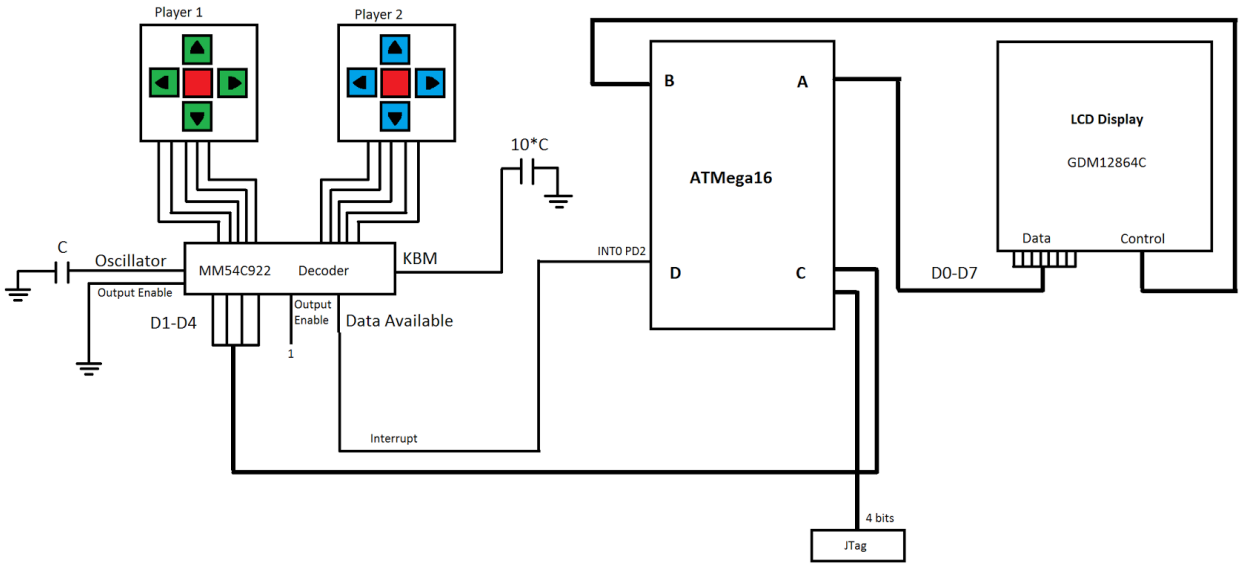
[2] ATMEL
ATMega16(L) data sheet
http://www.atmel.com/pt/br/Images/doc2466.pdf

[3] Texas Instruments
MM54C922 16-Key Encoder data sheet
http://www.ti.com/lit/ds/symlink/mm54c922.pdf

[4] XIAMEN OCULAR OPTICS CO
GDM12864HLCM data sheet
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/GDM12864H.pdf

[5] JTAG
http://www.jtag.com/

Appendix A - Block diagram



Appendix B - C code

Main.c:

```
int blink_count = -5;
bool refresh = false;

void init_interrupts() {
	cli();

	GICR = (1 << INT0);
	MCUCR = (3 << ISC00);		//Edge

	sei();
}

void timer1_init() {
  TCCR1B |= (1 << CS11);
  TCNT1 = 0;
  TIMSK |= (1 << TOIE1);

  sei();
}

ISR(INT0_vect) {

	changed = true;
	DDRD = 0x00;

	button = (PIND & 0x78) >> 3;
}

ISR(TIMER1_OVF_vect) {
	cli();
	blink_count++;

	if (blink_count >= 5)
		blink_count = -5;

  refresh = true;

	sei();
}

int main() {
	board_t board;
	init_interrupts();
	timer1_init();
	init_lcd();

	start_game(&board);

	while(true) {
		if(refresh) {
			update(&board);
			refresh = false;
		}
	}

	return 0;
}
```

LCD.c:

```
static void set_x(unsigned char cs, unsigned char offset);
static void set_y(unsigned char cs, unsigned char offset);

void status_read() {
	PORTA &= 0x7F;
	DDRA = 0x7F; // Sätt datariktning till ingång på bit 7

	RS_L;
	RW_H;

	while(PINA & 0x80); // While busy

	DDRA = 0xFF; // Reset datariktning
}

void init_lcd() {
	DDRA = 0xFF;
	PORTA = 0xFF;

	DDRB = 0xFF;
	PORTB = 0x27;		//E = RS = RW = RES = CS1 = CS2 = 1

	PORTA = 0x3F;		//starta båda skärmarna

	status_read();

	PORTB = 0x2C;		//E = RS = RW = RES = 1, 0 rest

	PORTA = 0x00;
}



unsigned char read_data(unsigned char cs) {
	status_read();

	if(cs == 1)
		SELECT_CS1;
	else
		SELECT_CS2;

	RS_H;
	RW_H;
	E_H;

	PORTA = 0x00;
	DDRA = 0x00;

	E_L;
	unsigned char data = PINA;

	return data;
}

void write_data(unsigned char cs, unsigned char data) {
	status_read();

	if (cs == 1)
		SELECT_CS1;
	else
		SELECT_CS2;
```

```
	RS_H;
	RW_L;
	E_H;

	PORTA = data;
	E_L;
}

void clear_lcd() {
	set_y(1, 0);
	set_y(2, 0);

	for (unsigned char x = 0; x < 8; ++x) {
		set_x(1, x);
		set_x(2, x);

		for (unsigned char y = 0; y < 64; ++y) {
			write_data(1, 0x00);
			write_data(2, 0x00);
		}

	}
}

void clear_cell(cell_t* cell) {

	set_x(2, cell->x_page);
	set_y(2, cell->y);

	for(unsigned int i = 0; i < 15; i++) {

		write_data(2, 0x01);

	}

	set_x(2, cell->x_page+1);
	set_y(2, cell->y);

	for(unsigned int i = 0; i < 15; i++) {
		write_data(2, 0x00);
	}
}

void draw_cross(cell_t* cell) {

	set_x(2, cell->x_page);
	set_y(2, cell->y);

	unsigned char data = 2;

	for (unsigned int i = 0; i < 7; i++) {

		data--;
		data <<= 1;
		data++;

		write_data(2, data);

	}

	write_data(2, data);
	write_data(2, data);

	for (unsigned int i = 8; i < 14; i++) {

		data--;
```

```
            data >>= 1;
            data++;

            write_data(2, data);


        }

        set_x(2, cell->x_page+1);
        set_y(2, cell->y);

        data = 0x80;

        for (unsigned int i = 0; i < 7; i++) {

            data--;
            data >>= 1;
            data++;

            write_data(2, data);

        }

        write_data(2, data);
        write_data(2, data);
        data = 1;

        for (unsigned int i = 8; i < 14; i++) {

            data <<= 1;

            write_data(2, data);

        }

}

void draw_circle(cell_t* cell) {

        set_x(2, cell->x_page);
        set_y(2, cell->y);

        write_data(2, 0xC1);
        write_data(2, 0x31);
        write_data(2, 0x09);
        write_data(2, 0x05);
        write_data(2, 0x05);
        write_data(2, 0x03);
        write_data(2, 0x03);
        write_data(2, 0x03);
        write_data(2, 0x03);
        write_data(2, 0x03);
        write_data(2, 0x05);
        write_data(2, 0x05);
        write_data(2, 0x09);
        write_data(2, 0x31);
        write_data(2, 0xC1);

        set_x(2, cell->x_page+1);
        set_y(2, cell->y);

        write_data(2, 0x07);
        write_data(2, 0x18);
        write_data(2, 0x20);
        write_data(2, 0x40);
        write_data(2, 0x40);
```

```
        write_data(2, 0x80);
        write_data(2, 0x80);
        write_data(2, 0x80);
        write_data(2, 0x80);
        write_data(2, 0x80);
        write_data(2, 0x40);
        write_data(2, 0x40);
        write_data(2, 0x20);
        write_data(2, 0x18);
        write_data(2, 0x07);
}

void redraw_cell(board_t* board) {

        if(board->previous == -1) {

                if(board->cells[board->current].s == BLINK_CROSS && blink_count <= 0)  {
                        draw_cross(&board->cells[board->current]);
                        return;
                } else
                        clear_cell(&board->cells[board->current]);


                if(board->cells[board->current].s == BLINK_CIRCLE && blink_count <= 0) {
                        draw_circle(&board->cells[board->current]);
                        return;
                } else
                        clear_cell(&board->cells[board->current]);

        } else {

                if(board->cells[board->previous].s == EMPTY)
                        clear_cell(&board->cells[board->previous]);

                if(board->cells[board->previous].s == CROSS)
                        draw_cross(&board->cells[board->previous]);

                if(board->cells[board->previous].s == CIRCLE)
                        draw_circle(&board->cells[board->previous]);

                if(board->cells[board->current].s == BLINK_CROSS && blink_count <= 0)
                        draw_cross(&board->cells[board->current]);
                else if(board->cells[board->current].s == BLINK_CROSS)
                        clear_cell(&board->cells[board->current]);

                if(board->cells[board->current].s == BLINK_CIRCLE && blink_count <= 0)
                        draw_circle(&board->cells[board->current]);
                else if(board->cells[board->current].s == BLINK_CIRCLE)
                        clear_cell(&board->cells[board->current]);
        }
}

void draw_board() {

        for (unsigned char x = 1; x <= 7; ++x) {
                set_x(2, x);
                set_y(2, 8);
                for (unsigned char y = 8; y <= 56; ++y) {

                        if(x == 7) {
                                write_data(2, 0x01);
                        } else {
                                if (y == 8 || y == 24 || y == 40 || y == 56) {
                                        write_data(2, 0xFF);
                                        continue;
                                }
```

```
                        if (x == 1)
                                write_data(2, 0x01);
                        else if (x == 3)
                                write_data(2, 0x01);
                        else if (x == 5)
                                write_data(2, 0x01);
                        else
                                write_data(2, 0x00);
                }

        }
    }
}

static void set_x(unsigned char cs, unsigned char offset) {
    status_read();

    if (cs == 1)
        SELECT_CS1;
    else
        SELECT_CS2;

    RS_L;
    RW_L;
    E_H;

    PORTA = 0xB8 + offset;      //X-address for cs1
    E_L;
}

static void set_y(unsigned char cs, unsigned char offset) {
    status_read();

    if (cs == 1)
        SELECT_CS1;
    else
        SELECT_CS2;

    RS_L;
    RW_L;
    E_H;

    PORTA = 0x40 + offset; // Y-address for cs
    E_L;
}

void draw_start_text() {

    // Övre halva T
    set_x(1, 1);
    set_y(1, 15);
    write_data(1, 0x78);
    write_data(1, 0x78);
    write_data(1, 0x78);
    write_data(1, 0x78);
    write_data(1, 0xF8);
    write_data(1, 0xF8);
    write_data(1, 0xF8);
    write_data(1, 0xF8);
    write_data(1, 0x78);
    write_data(1, 0x78);
    write_data(1, 0x78);
    write_data(1, 0x78);

    //Övre halva I
    set_y(1, 33);
```

```
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);

//Övre halva C
set_y(1, 42);
write_data(1, 0xC0);
write_data(1, 0xE0);
write_data(1, 0xF0);
write_data(1, 0xF8);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x38);

//Undre halva T
set_x(1, 2);
set_y(1, 19);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);

//undre halva I
set_y(1, 33);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);

//undre halva C
set_y(1, 42);
write_data(1, 0x07);
write_data(1, 0x0F);
write_data(1, 0x1F);
write_data(1, 0x3F);
write_data(1, 0x3C);
write_data(1, 0x3C);
write_data(1, 0x3C);
write_data(1, 0x38);

//övre halva T
set_x(1, 3);
set_y(1, 15);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);

//Övre halva A
set_y(1, 30);
write_data(1, 0xE0);
write_data(1, 0xF0);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0x38);
write_data(1, 0xF8);
```

```
write_data(1, 0xF8);
write_data(1, 0xF0);
write_data(1, 0xE0);

//Övre halva C
set_y(1, 42);
write_data(1, 0xC0);
write_data(1, 0xE0);
write_data(1, 0xF0);
write_data(1, 0xF8);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x38);


//Undre halva T
set_x(1, 4);
set_y(1, 19);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);


//undre halva A
set_y(1, 30);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x07);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);
write_data(1, 0x3F);

//undre halva C
set_y(1, 42);
write_data(1, 0x07);
write_data(1, 0x0F);
write_data(1, 0x1F);
write_data(1, 0x3F);
write_data(1, 0x3C);
write_data(1, 0x3C);
write_data(1, 0x3C);
write_data(1, 0x38);

// Övre halva T
set_x(1, 5);
set_y(1, 15);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0xF8);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);
write_data(1, 0x78);

//Övre halva O
set_y(1, 30);
```

```
        write_data(1, 0xC0);
        write_data(1, 0xE0);
        write_data(1, 0xF0);
        write_data(1, 0x38);
        write_data(1, 0x38);
        write_data(1, 0x38);
        write_data(1, 0xF0);
        write_data(1, 0xE0);
        write_data(1, 0xC0);

        // ÖVre halva E
        set_y(1, 42);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);

        //Undre halva T
        set_x(1, 6);
        set_y(1, 19);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);


        //Undre halva O
        set_y(1, 30);
        write_data(1, 0x07);
        write_data(1, 0x0F);
        write_data(1, 0x1F);
        write_data(1, 0x38);
        write_data(1, 0x38);
        write_data(1, 0x38);
        write_data(1, 0x1F);
        write_data(1, 0x0F);
        write_data(1, 0x07);


        // Undre halva E
        set_y(1, 42);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3B);
        write_data(1, 0x3B);
        write_data(1, 0x3B);
        write_data(1, 0x3B);
        write_data(1, 0x3B);
}

void draw_winner(int player) {

        set_y(1, 0);

        for (unsigned char x = 0; x < 8; ++x) {
                set_x(1, x);

                for (unsigned char y = 0; y < 64; ++y) {
                        write_data(1, 0x00);
```

```
        }
}

if(player != 0) {

        if(player == 1) {       //CROSS WON

        //CROSS OVER
        set_x(1, 1);
        set_y(1, 24);
        write_data(1, 0x03);
        write_data(1, 0x07);
        write_data(1, 0x0F);
        write_data(1, 0x1F);
        write_data(1, 0x3F);
        write_data(1, 0x7E);
        write_data(1, 0xFC);
        write_data(1, 0xF8);
        write_data(1, 0xF0);
        write_data(1, 0xF0);
        write_data(1, 0xF8);
        write_data(1, 0xFC);
        write_data(1, 0x7E);
        write_data(1, 0x3F);
        write_data(1, 0x1F);
        write_data(1, 0x0F);
        write_data(1, 0x07);
        write_data(1, 0x03);


        // CROSS UNDER
        set_x(1, 2);
        set_y(1, 24);
        write_data(1, 0x60);
        write_data(1, 0x70);
        write_data(1, 0x78);
        write_data(1, 0x7C);
        write_data(1, 0x7E);
        write_data(1, 0x3F);
        write_data(1, 0x1F);
        write_data(1, 0x0F);
        write_data(1, 0x07);
        write_data(1, 0x07);
        write_data(1, 0x0F);
        write_data(1, 0x1F);
        write_data(1, 0x3F);
        write_data(1, 0x7E);
        write_data(1, 0x7C);
        write_data(1, 0x78);
        write_data(1, 0x70);
        write_data(1, 0x60);

        } else {//CIRCLE WON

                set_x(1, 1);
                set_y(1, 25);

                // Övre halvan av cirkeln
                write_data(1, 0xE0);
                write_data(1, 0xF0);
                write_data(1, 0x1C);
                write_data(1, 0x0C);
                write_data(1, 0x07);
                write_data(1, 0x03);
                write_data(1, 0x03);
                write_data(1, 0x03);
```

```
        write_data(1, 0x03);
        write_data(1, 0x03);
        write_data(1, 0x03);
        write_data(1, 0x07);
        write_data(1, 0x0C);
        write_data(1, 0x1C);
        write_data(1, 0xF0);
        write_data(1, 0xE0);


        // Undre halvan av cirkeln
        set_x(1, 2);
        set_y(1, 25);
        write_data(1, 0x07);
        write_data(1, 0x0F);
        write_data(1, 0x38);
        write_data(1, 0x30);
        write_data(1, 0x60);
        write_data(1, 0xC0);
        write_data(1, 0xC0);
        write_data(1, 0xC0);
        write_data(1, 0xC0);
        write_data(1, 0xC0);
        write_data(1, 0xC0);
        write_data(1, 0x60);
        write_data(1, 0x30);
        write_data(1, 0x38);
        write_data(1, 0x0F);
        write_data(1, 0x07);
}

//Övre halva W
set_x(1, 4);
set_y(1, 6);

write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
set_y(1,18);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);

//Övre halva O
set_y(1, 25);
write_data(1, 0xC0);
write_data(1, 0xF0);
write_data(1, 0xF8);
write_data(1, 0xFC);
write_data(1, 0x7C);
write_data(1, 0x3C);
write_data(1, 0x1C);
write_data(1, 0x1C);
write_data(1, 0x1C);
write_data(1, 0x3C);
write_data(1, 0x7C);
write_data(1, 0xFC);
write_data(1, 0xF8);
write_data(1, 0xF0);
write_data(1, 0xC0);

// Övre halva N
set_y(1, 44);
write_data(1, 0xFF);
```

```
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0x7C);
write_data(1, 0xF8);
write_data(1, 0xF0);
write_data(1, 0xE0);
write_data(1, 0xC0);
write_data(1, 0x80);
write_data(1, 0x00);
write_data(1, 0x00);
write_data(1, 0x00);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);

set_x(1, 5);
set_y(1, 6);

// Undre halva W
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0x7C);
write_data(1, 0x3E);
write_data(1, 0x1F);
write_data(1, 0x0F);
write_data(1, 0x0F);
write_data(1, 0x1F);
write_data(1, 0x3E);
write_data(1, 0x7C);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);

//Undre halva O
set_y(1, 25);
write_data(1, 0x07);
write_data(1, 0x1F);
write_data(1, 0x3F);
write_data(1, 0x7F);
write_data(1, 0x7C);
write_data(1, 0xF8);
write_data(1, 0xF0);
write_data(1, 0xF0);
write_data(1, 0xF0);
write_data(1, 0xF8);
write_data(1, 0x7C);
write_data(1, 0x7F);
write_data(1, 0x3F);
write_data(1, 0x1F);
write_data(1, 0x07);

// Undre halva N
set_y(1, 44);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0xFF);
write_data(1, 0x00);
write_data(1, 0x00);
write_data(1, 0x01);
write_data(1, 0x03);
```

```
        write_data(1, 0x07);
        write_data(1, 0x0F);
        write_data(1, 0x1F);
        write_data(1, 0x3E);
        write_data(1, 0x7C);
        write_data(1, 0xFF);
        write_data(1, 0xFF);
        write_data(1, 0xFF);
        write_data(1, 0xFF);

} else {//TIE

        // Övre halva T
        set_x(1, 3);
        set_y(1, 15);
        write_data(1, 0x78);
        write_data(1, 0x78);
        write_data(1, 0x78);
        write_data(1, 0x78);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0x78);
        write_data(1, 0x78);
        write_data(1, 0x78);
        write_data(1, 0x78);

        //Övre halva I
        set_y(1, 33);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);

        // ÖVre halva E
        set_y(1, 42);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xF8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);
        write_data(1, 0xB8);

        //Undre halva T
        set_x(1, 4);
        set_y(1, 19);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);

        //undre halva I
        set_y(1, 33);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
        write_data(1, 0x3F);

        // Undre halva E
        set_y(1, 42);
        write_data(1, 0x3F);
        write_data(1, 0x3F);
```

```
                write_data(1, 0x3F);
                write_data(1, 0x3F);
                write_data(1, 0x3B);
                write_data(1, 0x3B);
                write_data(1, 0x3B);
                write_data(1, 0x3B);
                write_data(1, 0x3B);
        }
}
```

Macro.h:

```
#ifndef MACRO_H
#define MACRO_H

#include <avr/io.h>

//PORTB => XX E RS RW CS1 CS2

#define RW_H (PORTB |= 0x08)
#define RW_L (PORTB &= ~0x08)

#define E_H (PORTB |= 0x20)
#define E_L (PORTB &= ~0x20)

#define SELECT_CS1 (PORTB |= 0x02, PORTB &= ~0x01)
#define CS1_L (PORTB &= ~0x02)

#define SELECT_CS2 (PORTB |= 0x01, PORTB &= ~0x02)
#define CS2_L (PORTB &= ~0x01)

#define RS_H (PORTB |= 0x10)
#define RS_L (PORTB &= ~0x10)

#endif
```