TIC-TAC-TOE

Digitala System - EITA15
19/05-2023

Handledare:
Bertil Lindvall

Grupp 14:
Alexander Englesson, Lucas Vandenborre, William Croxton, André Kanakis & Fabian Wilson

Sammanfattning

Projektet genomfördes inom ramen för kursen Digitala System, EITA15, där en ATmega1284 mikrokontroller användes. Målet var att skapa ett Tic Tac Toe-spel med hjälp av programmering och 10 stycken Neopixels där 9 stycken representerade spelplanen. Spelare skulle kunna spela mot varandra och välja rutor/färger genom att använda knappsatsen. Den 10:e Neopixeln används för att markera vems tur det är.

Projektet inleddes med att inhämta information om hårdvarukomponenterna från respektive datablad och skapa ett kopplingsschema. Komponenter kopplades och löddades sedan ihop på ett kopplingsbräde. Samtidigt påbörjade programmeringen i Atmel Studio 7, där primärt las fokus på att tända enstaka Neopixels i olika färger. Detta gjordes med hjälp av kod i C samt Assembler.

Därefter utvecklades koden för att styra spelet mellan två datorstyrda spelare, kallade datorer. Både datorer valde en slumpmässig ledig plats på spelbrädet. Datorernas speldrag syntes med hjälp av deras olika färger på spelplanen, röd och blå. Därefter kopplades samt programmerades knappsatsen in för att utveckla spelet. Därefter skapades möjligheten att spela Tic Tac Toe spelare mot spelare. En spelare valde en ruta på spelplanen genom att trycka på en knapp, då ändrades färgen på den motsvarande Neopixeln för att representera spelarens markör. Vid sidan av spelplanen placerades en tionde Neopixel som visade färgen för spelaren som skulle göra nästa drag.

På knappsatsen fick användaren möjligheten att välja att spela mot en annan spelare eller låta datorn spela mot en annan dator. En resetknapp fanns också för att återställa spelplanen.

Projektet resulterade i ett fungerande Tic Tac Toe-spel med en 3x3 spelplan, kontrollerad av knappsatsen. Spelare kunde välja sin färg och sedan tända Neopixeln på respektive plats på spelplanen.

Nyckelord

NeoPixel, Knappsats, Keycoder, ATmega1284 mikrokontroller, TIC TAC TOE, C kod, Assembler kod

Abstract

As part of the Digital Systems course, EITA15, an ATmega1284 microcontroller was used to construct a project. The objective was to create a Tic Tac Toe game using programming and 10 Neopixels, with 9 of them representing the game board. Players would be able to play against each other and select squares/colours using the keypad. The 10th Neopixel was used to indicate whose turn it was.

The project began by gathering information about the hardware components from their respective datasheets and creating a wiring diagram. Following the guidance of the aforementioned diagram, components were connected and soldered on a breadboard. Simultaneously, programming started using Atmel Studio 7, initially focusing on lighting individual Neopixels in different colours. This was accomplished using C code and assembler.

Next, code was developed to control the game mode between two computer controlled players, called computers. Both computers randomly selected an available spot on the game board. Each player's moves could be observed through their respective colours on the game board, red and blue. Afterwards,gameplay was enhanced by connecting and programming the keypad. Subsequently, implementing the feature allowing players to play Tic Tac Toe against each other. A player selected a square on the game board by pressing a button, and the colour of the corresponding Neopixel changed to represent the player's marker. Adjacent to the game board, a tenth Neopixel was placed to display the colour of the player who would make the next move.

Buttons on the keypad provided the user with the option to play against another player or let the computer play against itself. A reset button was also available to reset the game board.

The project resulted in a functional Tic Tac Toe game with a 3x3 game board, controlled by the keypad. Players could choose their colour and illuminate the corresponding Neopixel on the game board.

Keywords

NeoPixel, Button Matrix/ Keypad, Key-encoder, ATmega1284 microcontroller, Tic-Tac-Toe, C code, Assembler code

Innehållsförteckning

Sammanfattning 2
Nyckelord 2
Abstract 3
Keywords 3
Innehållsförteckning 4
1. Inledning 5
    1.1 Bakgrund 5
    1.2 Syfte 5
    1.3 Kravspecifikation 5
    1.4 Problemformulering 5
2. Teknisk bakgrund 6
    2.1 Hårdvara 6
    2.2 Mjukvara 6
3. Metod 7
    3.1 Källkritik 7
4. Analys 8
5. Resultat 9
6. Slutsats 10
7. Källförteckning 11
8. Appendix 12
    8.1 Assembly.s 12
    8.2 Colors.c 13
    8.3 main.c 21
    8.4 Kopplingsschema 26

1. Inledning

1.1 Bakgrund

Digitala System, en kurs under Lunds Universitet inkluderar många moment, ett av dessa moment är ett projektarbete. Momentet utförs i grupper och studenterna ombeds bygga prototyper med hjälp av mikrokontrollern ATmega1284. Valet av projekt utfördes i koherens mellan projektgruppen och projekthandledaren. För ett lyckat projekt krävs: kopplingsschema, en prototyp av tidigare benämnda kopplingsschema, mjukvaruutveckling och kravspecifikation.

1.2 Syfte

Syftet är att få en övergripande bild av tillvägagångssättet av ett projekt, och därmed utveckla problemlösningsförmåga samt förstå den praktiska användningen av digitala system och de kunskaper införskaffade genom kursens gång.

1.3 Kravspecifikation

I detta stycke förklaras det krav som sattes på prototypen. Två datorer ska kunna spela mot varandra genom att slumpmässigt spela ett drag. Alla neopixlar ska kunna kontrolleras genom 4x4 knappsatsen av spelare för spelläget: spelare mot spelare. Spelbrädet ska kunna nollställas och växla mellan de olika spellägen.

1.4 Problemformulering

Vilka komponenter krävs för att åstadkomma önskade resultat? Vad krävs mjukvarumässigt för att två datorer ska kunna spela mot varandra? På vilket sätt kontrollerar man enklast neopixlar och dess färg?

2. Teknisk bakgrund

2.1 Hårdvara

- **Processor**: ATmega1284
- **Knappsats**: Grayhill 87-series
- **Key encoder**: MM74C922
- **J-TAG**: Atmel-ICE
- **Neopixlar**: 10 stycken WS2812B RGB LEDS
- **Kondensatorer**: 1st: 1 µF & 1st: 0,1 µF

2.2 Mjukvara

- **Atmel Studio 7**: Kod i C och Assembly
- **EAGLE**: Program för att rika kretsschema

3. Metod

En digitaliserad version av pappers-och-penna spelet Tic Tac Toe, eller tre i rad, beslutades att konstrueras av projektgruppen. Visionen var att ha en 4x4 knappsats som användes för att kontrollera ett 3x3 spelbräde som representerades av 9 stycken Neopixlar, där varje Neopixel färg representerade spelarnas drag. Därefter ritades ett kopplingsschema (*se figur A*) och en kravspecifikation skrevs för projektet. Programmet EAGLE användes för att rita kopplingsschemat, och komponenternas datablad användes för att undersöka portanslutningarna för varje hårdvarukomponent. Kommunikation inom projektgruppen skedde främst via Facebook Messenger samt vid planerade laborationer och möten på campus.

När kopplingsschemat var ritat påbörjades processen att koppla ihop alla komponenter. Mikrokontrollern kopplades först samman med en Neopixel på kretskortet genom trådning och lödning. Därefter kopplades mikrokontrollern till en JTAG, som i sin tur kopplades till datorn, och Atmel Studio 7 startades för att programmera mikrokontrollern och därmed styra Neopixeln.

Efter att ha övergått från att endast kunna styra en Neopixel fortsatte arbetet med att koppla ihop resterande komponenter. Därefter påbörjades fasen som innefattade programmering och felsökning med hjälp av Atmel Studio 7:s debugger.

Det primära fokuset låg initialt på att åstadkomma individuell belysning i önskade färger för varje enskild Neopixel och sedan övergå till att kunna styra Neopixelarna med hjälp av knappsatsen. När funktionen för att spela dator mot dator fungerade övergick fokuset till att implementera spel mellan spelare. Knappsatsen integrerades då mer i programmeringen. Därefter utvecklades en återställningsknapp som återställde spelplanen.

3.1 Källkritik

De källor som användes i projektet var databladen för de olika komponenterna. Dessa datablad betraktas som högt trovärdiga eftersom de kommer direkt från leverantören av produkterna.

4. Analys

De Neopixlar som använts har möjligheten att avläsa den signal de avger. Detta innebär att det vore möjligt att konstant avläsa värdet på Neopixlarna och på så sätt kontrollera vilka platser på brädet som är upptagna av vilka spelare. Denna lösning hade krävt en extra datakabel och en extra pin på mikrokontrollern för varje Neopixel. Istället gjorde beslutet att kontrollera brädet via mjukvara i form av en vektor i C-programmet. Med den lösningen behövs inte datariktningen på mikrokontrollerns pins ändras från insignaler till utsignaler mellan varje drag.

Genom bildandet av en metod *Led* i klassen som hanterar ljusen så blev det möjligt att använda switch-cases för att ändra färgerna på de olika Neopixlarna effektivt. Det bestämdes att identifiera färgerna som integers 0 till 3 enligt RGB. Enligt identifieringsystemet är 0 en avstängd lampa, 1 är rött, 2 är grönt, och 3 är blått. Metoden *Led* har ännu en parameter som anger vilken Neopixel det är som ska hanteras. Med en och samma metod är det nu möjligt att först fastställa vilken Neopixel det är som ska ändra färg samt till vilken färg genom att kalla på metoden *Led(int n, int c)*, där *n* anger vilken Neopixel och *c* anger vilken färg. Detta resulterade i en lång metod där det var möjligt att iterera över varje Neopixel och färg effektivt.

Koden är konstruerad på så sätt att metoden *checkWinner* itererar över varje rad och kolumn samt de två diagonalerna och returnerar sant ifall den finner tre på varandra följande av samma färg. När tre på varandra följande färger har funnits så avslutas den while-loop som håller spelet i gång. Ett problem uppstod när spelbrädet blev oavgjort. Metoden *checkWinner* kunde då aldrig hitta en vinnare utan programmet fastnade i att ändlöst leta efter en ledig ruta för datorn. Detta problem kunde åtgärdas genom att implementera en summa som adderar värdet av alla spelrutor. Detta fungerar då röd spelare alltid går först och blå spelare sist, på ett spelbräde som är oavgjort finns det då 5 röda platser och 4 blåa. Röd färg har fått ett värde på 1 och blå färg ett värde på 3. Summan av ett oavgjort spelbräde är då 17. I de fall där summan för spelbrädet överstiger 16 innebär då att spelbrädet är fullt och loopen bryts.

Det största problemet som uppstod var att efter en godtycklig tryckning på knappsatsen så behöll programmet värdet på den knapp som trycktes. Detta resulterade i att programmet kontinuerligt initierade switch-case och gjorde om samma operation tills dess att en annan knapp tryckts. Trots rådgivning med flera kunniga i övre läsår och kursassistenter så kunde inte felet identifieras. Istället gjordes beslutet att kringgå problemet genom att ha två knappar på knappsatsen som inte var kopplade till Neopixlar agera som knappar för att byta vems tur det var att spela.

5. Resultat

Projektarbetet resulterade i framtagandet av en digitaliserad version av Tic Tac Toe, där spelplanen representerades av en 3x3 matris av Neopixlar med olika färger. Den visuella representationen av spelplanen tillhandahöll en tydlig och interaktiv upplevelse för användarna. Den 10:e Neopixeln användes för att markera nästa spelares tur genom att lysa i antingen rött eller blått, beroende på vilken spelare som skulle göra nästa drag.

För att interagera med spelet användes en 4x4 knappsats. Knapparna möjliggjorde flera funktioner, inklusive att välja en plats på spelplanen, starta en dator mot dator match, återställa spelplanen samt bestämma vilken färg som skulle tilldelas spelplanen. Genom att trycka på de olika knapparna kunde spelarna aktivt delta i spelet och ta sina drag på önskade platser på spelplanen.

Denna digitaliserade Tic Tac Toe-version erbjöd en mer modern och interaktiv spelupplevelse jämfört med den traditionella pappers-och-penna versionen. Spelet var visuellt tilltalande och gav användarna möjlighet att strategiskt placera sina färger på spelplanen, samtidigt som de kunde följa spelets framsteg genom Neopixlarnas skiftande färger.

6. Slutsats

Projektet genomfördes framgångsrikt och resulterade i ett fungerande Tic Tac Toe-spel med hjälp av en ATmega1284 mikrokontroller, programmering och 10 Neopixels. Spelare kunde spela mot varandra och välja rutor/färger genom att använda knappsatsen. Projektet inkluderade också möjligheten att spela dator mot dator och en resetknapp för att återställa spelplanen.

Genom att kombinera kunskap om hårdvarukomponenterna, kopplingsscheman samt programmering i C och Assembler lyckades projektgruppen skapa en prototyp som uppfyllde de angivna kraven och målen för projektet. Genom att använda metoder som att tända och ändra färgerna på Neopixels samt att kontrollera spelet via knappsatsen kunde spelare interagera med spelet på ett intuitivt sätt.

Sammanfattningsvis visar projektet på förmågan att kombinera hårdvarukunskaper, kopplingsscheman och programmering för att skapa ett fungerande digitalt system. Det ger också en inblick i problemlösningsförmåga och praktisk tillämpning av digitala system. Projektet har uppfyllt de angivna målen och kraven och har visat att det är möjligt att skapa ett spel som Tic Tac Toe med hjälp av mikrokontrollern ATmega1284 och Neopixels.

7. Källförteckning

- Datablad Key Encoder: MM74C922 • MM74C923 16-Key Encoder • 20-Key Encoder
  Hämtad: 05/04 - 2023
- Datablad knappsats: 87 Datasheet
  Hämtad: 05/04 - 2023
- Datablad Neopixel: WS2812B LED WORLDSEMI CO.,LIMITED
  Hämtad: 05/04 - 2023
- Datablad ATmega1284: ATmega1284
  Hämtad: 05/04 - 2023

8. Appendix

8.1 Assembly.s

```
.global send_one
.global send_zero
.global send_ret
.global read_buttons

#define PORTD 0x0B
#define DDRD 0x0A
#define PIND 0x09
#define PINA 0x20
#define DDRA 0x01
#define PORTB 0x05
#define DDRB 0x04

send_one:
    ldi r19, 0b11111111
    out PORTB, r19
    out PORTD, r19
    nop
    nop
    ldi r18, 0b00000000
    nop
    out PORTB, r18
    out PORTD, r18

    ret

send_zero:
    ldi r18, 0b11111111
    out PORTB, r18
    out PORTD, r18
    ldi r18, 0b00000000
    out PORTB, r18
    out PORTD, r18

    ret

send_ret:
    ldi r18,0b00000000
    out PORTB, r18
    out PORTD, r18
```

```
    ldi r19, 20
         loop1:
              ldi r18, 10
              loop2:
                   dec r18
                   brne loop2
              dec r19
              brne loop1

    ret
```

8.2 Colors.c

```
/*
 * main.c
 *
 * Created: 2023-05-15 14:32:18
 * Author : an8615ka-s
 */
extern void send_one();
extern void send_zero();
extern void send_ret();

#define F_CPU 8000000
#include <util/delay.h>
#include <avr/io.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <stdbool.h>

void green_led() {
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();

  send_zero();
  send_zero();
  send_zero();
```

```
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_ret();
}
void blue_led() {
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
```

```
  send_ret();
}
void red_led() {
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();
  send_one();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_ret();
}
void noll_led() {
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
```

```
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();
  send_zero();

  send_ret();
}
void ret() {

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();

  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
  send_ret();
```

```
}

void Led(int n, int c) {
      switch (n)
      {
            case 0:
            DDRB = 0b00000000;
            DDRD = 0b01000000;
            switch (c)
            {
                  case 0: //Ingen färg
                  noll_led();
                  break;
                  case 1: //Röd
                  red_led();
                  break;
                  case 2: //Grön
                  green_led();
                  break;
                  case 3: //Blå
                  blue_led();
                  break;
            }
            _delay_ms(500);
            break;
            case 1:
            DDRB = 0b00000000;
            DDRD = 0b00100000;
            switch (c)
            {
                  case 0: //Ingen färg
                  noll_led();
                  break;
                  case 1: //Röd
                  red_led();
                  break;
                  case 2: //Grön
                  green_led();
                  break;
                  case 3: //Blå
                  blue_led();
                  break;
            }
            _delay_ms(500);
            break;
```

```
case 2:
DDRB = 0b00000000;
DDRD = 0b00010000;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 3:
DDRB = 0b00000000;
DDRD = 0b00001000;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 4:
DDRB = 0b00000000;
DDRD = 0b00000100;
switch (c)
{
```

```
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 5:
DDRB = 0b00000000;
DDRD = 0b00000010;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 6:
DDRB = 0b00000000;
DDRD = 0b00000001;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
```

```
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 7:
DDRB = 0b10000000;
DDRD = 0b00000000;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
      blue_led();
      break;
}
_delay_ms(500);
break;
case 8:
DDRB = 0b01000000;
DDRD = 0b00000000;
switch (c)
{
      case 0: //Ingen färg
      noll_led();
      break;
      case 1: //Röd
      red_led();
      break;
      case 2: //Grön
      green_led();
      break;
      case 3: //Blå
```

```
                blue_led();
                break;
            }
            _delay_ms(500);
            break;
            case 9:
            DDRB = 0b00100000;
            DDRD = 0b00000000;
            switch (c)
            {
                case 0: //Ingen färg
                noll_led();
                break;
                case 1: //Röd
                red_led();
                break;
                case 2: //Grön
                green_led();
                break;
                case 3: //Blå
                blue_led();
                break;
            }
            _delay_ms(500);
            break;
            default:
            break;
      }
}
```

8.3 main.c

```
/*
 * main.c
 *
 * Created: 2023-05-15 14:13:20
 * Author : an8615ka-s
 */

#define F_CPU 8000000
#define SIZE 3

#include <util/delay.h>
#include <avr/io.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <time.h>
#include <stdbool.h>
#include <unistd.h>
extern void Led(int n, int c);

void initPins() {                    // INITIERAR ALLA PINS OCH DDR
     PORTB |= 0b11100000;
     PORTD |= 0b00000000;
     DDRB |= 0b11100000;
     DDRD |= 0xFF;
}
void resetBoard(int board[]) {       // STÄNGA AV ALLA LAMPOR!
     for (int i=0; i < 10; i++) {
          Led(i,0);
     }
}
void sendColors(int board[]) {       // SKICKAR FÄRGER TILL LAMPOR 1
-> 9
     for (int i = 0; i < 9; i++) {
          Led(i+1, i);
          _delay_ms(50);
     }
}

int main(void)
{
     initPins();
     int gameBoard[] = {0,0,0,0,0,0,0,0,0};
     resetBoard(gameBoard);
     Led(0,2);       // VISAR ATT DEN ÄR REDO FÖR KEYPRESS!
     int keypad = 100;
     int player = 1;
    while (1)
    {
          DDRB |= 0b11100000;
          keypad = 100;
          if (PINB != keypad) {
               DDRB |= 0b11100000;
               keypad = PINB;
               if (keypad != 100) {
                    switch(keypad) {
                         case 0:
                         Led(7,player);
                         break;
```

```
                        case 1:
                        Led(4,player);
                        break;
                        case 2:
                        Led(1,player);
                        break;
                        case 8:
                        Led(9,player);
                        break;
                        case 9:
                        Led(6,player);
                        break;
                        case 10:
                        Led(3,player);
                        break;
                        case 12:
                        Led(8,player);
                        break;
                        case 13:
                        Led(5,player);
                        break;
                        case 14:
                        Led(2,player);
                        break;
                        case 11: ;      // BOT VS BOT;
                        int choice = -1;
                        int turn = 1;
                        Led(0,1);

                        while(checkWinner(gameBoard) == 0){
                              int sum = 0;
        // Sum-funktionen är till för att stoppa när brädet är
oavgjort.
                              for (int i = 0; i < 9; i ++) {
                                    sum += gameBoard[i];
                              }
                              if (sum > 16) {
                                    break;
                              }
                              Led(0, turn);
                              do {
                                    choice = rand() % 9;
                              } while (gameBoard[choice] > 0);
                              _delay_ms(100);
                              gameBoard[choice] = turn;
```

```
                                    Led(choice+1, turn);
                                    if (turn == 3) {
                                          turn = 1;
                                    } else if (turn == 1) {
                                          turn = 3;
                                    }
                              }
                              break;
                              case 15:
      // Resettar värdet på gameBoard och släcker alla lampor.
                              for (int i = 0; i < 9; i ++) {
                                    gameBoard[i] = 0;
                              }
                              resetBoard(gameBoard);
                              break;
                              case 4:
                              player = 1;
                              Led(0,player);
                              break;
                              case 5:
                              player = 3;
                              Led(0,player);
                              case 3:
                              srand(1337);
                              _delay_ms(300);
                              break;
                              case 7:
                              srand(420);
                              _delay_ms(300);
                              break;
                              case 6:
                              srand(6969);
                              _delay_ms(300);
                              break;
                              default:
                              break;
                              }
                        }
                  }
      }
}


int checkWinner(int board[]) {
      // Rader och kolonner
```

```
     for (int i = 0; i < SIZE; i++) {
          if (board[i*SIZE] != 0 && board[i*SIZE] == board[i*SIZE+1]
&& board[i*SIZE+1] == board[i*SIZE+2])
               return board[i*SIZE];
          if (board[i] != 0 && board[i] == board[i+SIZE] &&
board[i+SIZE] == board[i+2*SIZE])
               return board[i];
     }
     // Diagonaler
     if (board[0] != 0 && board[0] == board[4] && board[4] ==
board[8])
          return board[0];
     if (board[2] != 0 && board[2] == board[4] && board[4] ==
board[6])
          return board[2];

     return 0;
}
```

8.4 Kopplingsschema

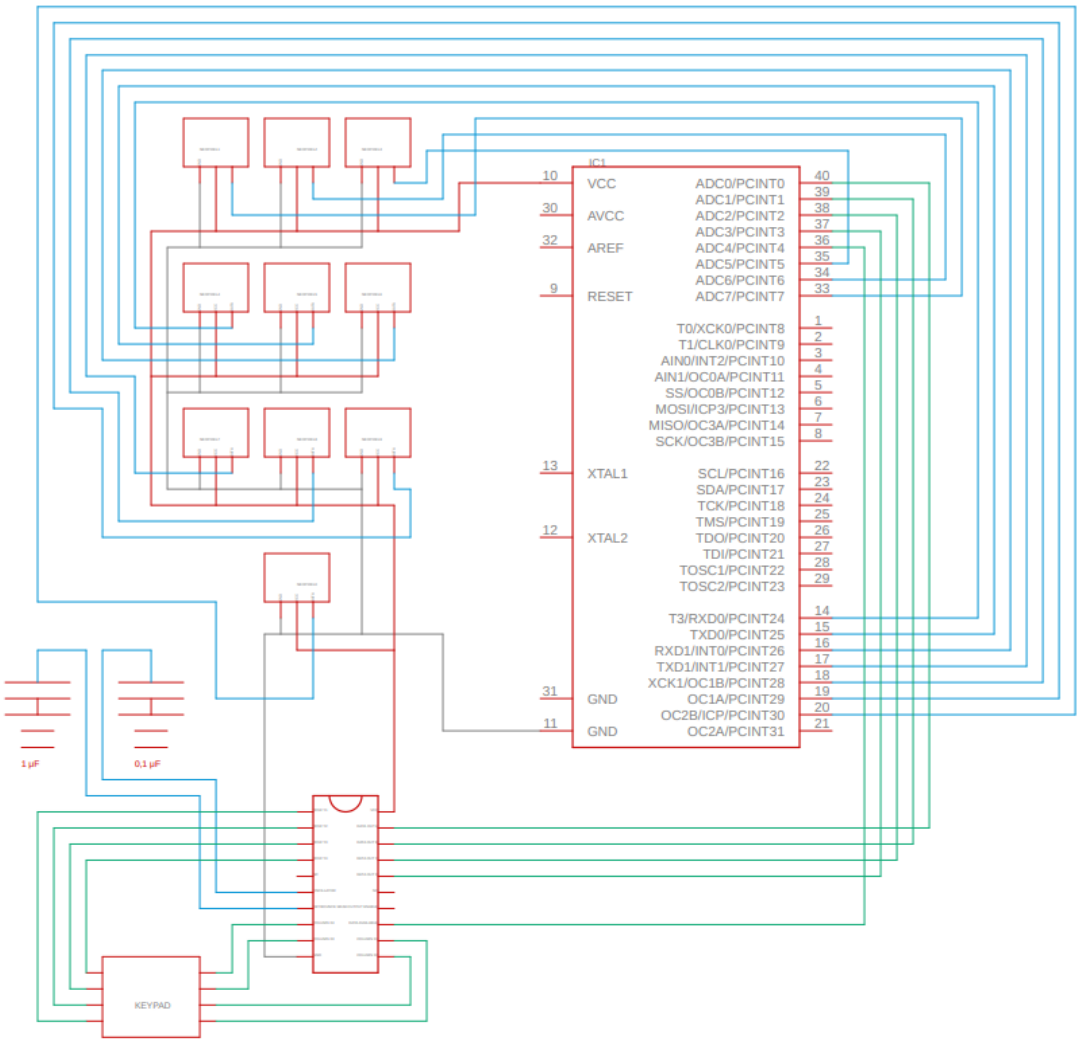

Fig A. Kretsschema