LUNDS
UNIVERSITET

Larmsystem

Kurs: EITA15, Digitala System

Grupp 6

Djuro Petrovic

Fredrik Tallberg

Maximilian Swan

Olof Svensson

Handledare: Bertil Lindvall, Lars-Göran Larsson

Abstract

Burglary and theft are common security issues that have to be prevented at any property.
To prevent this, it is necessary to be able to alert property owners of any suspicious movement around their property. The present report details the design and construction of a simple prototype alarm system that detects movement and sends out a signal using a buzzer with sound. The circuit also contains a display and a keypad beside the detection sensor and the microprocessor, which serves as the base of the entire system. The alarm system can be set to armed or disarmed depending on whether the correct code was typed in. The present project has resulted in a prototype that is reliable enough to be used in a close-range environment up to 5 meters.

1. Inledning

1.1. Bakgrund

Som en avslutande del av kursen ”Digitala System” (EITA15), ingår det ett projektarbete där studenterna förväntas skapa en fungerande prototyp baserad på mikroprocessorn ATMEGA1284, varpå resultatet sedan ska dokumenteras och redovisas. Gruppen har i detta projekt valt att skapa ett larmsystem med tillhörande knappsats och display.

1.2. Syfte

Det övergripande syftet med projektet är att få färdigheter i att samverka i grupp och att applicera den kunskap som erhållits under kursens gång, samt skapa grundläggande färdigheter i att systematiskt dokumentera, planera och genomföra ingenjörsmässiga projekt. Med andra ord är syftet att tränas i grundläggande aspekter inom ingenjörsmässigt arbete. Projektet har inte framskridit helt och hållet problemfritt, men i slutändan har en fungerande prototyp producerats där samtliga av kravspecifikationens väsentliga punkter har uppfyllts.

1.3. Målformulering

Projektgruppen beslutade att larmet bör uppfylla följande grundläggande krav: Larmet måste först och främst ha fungerande sensorer som pålitligt detekterar rörelse. Användaren ska kunna interagera med larmet via en knappsats. Larmet ska ha en reset-funktion där larmet slutar ljuda och återgår till normalläge. När sensorn triggas så ska anläggningen svara med att en högtalare börjar ljuda. Ljudsignalen ska indikera för användaren att sensorn har triggats. I anläggningen ska det också finnas en display som möjliggör användarvänlig interaktion med larmets olika funktioner. Armering och desarmering av larmet ska enbart vara möjligt om användaren matar in den korrekta armeringskoden. Armeringskoden ska kunna ändras av användaren, om denne först korrekt matar in koden. Till sist ska det finnas en visuell indikator i form av en blinkande ljusdiod.

1.4. Problemformulering

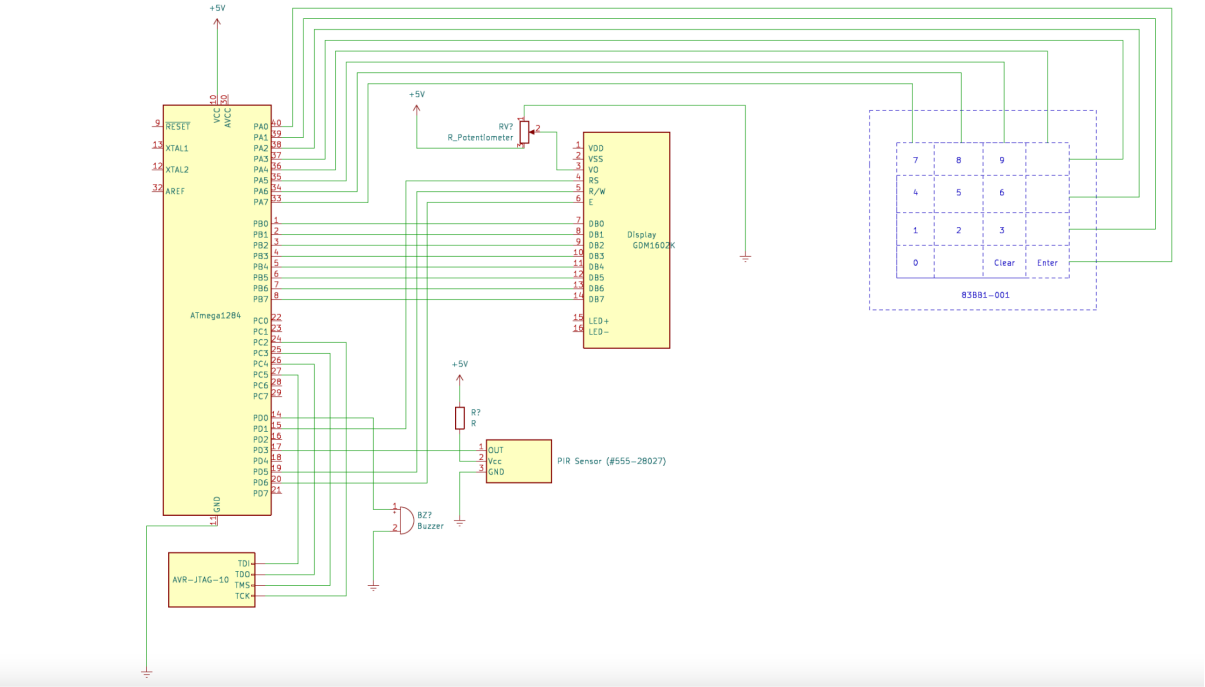
För att slutföra projektet på ett tillfredsställande sätt så krävdes att ett antal problem behövde lösas:

- Vilka komponenter passar sig bäst till larmsystemet och hur ska dessa kopplas för att underlätta eventuella ändringar samt felsökning?
- På vilket sätt ska komponenterna kommunicera med mikroprocessorn och vice versa?
- Hur ska programvaran se utformas för att möjliggöra komponenternas samverkan?
- Vilka villkor ska gälla för armering och desarmering av larmet?

1.5. Motivering

Gruppen har beslutat att skapa ett larmsystem eftersom projektet då går att knyta an till tidigare moment i kursen, samtidigt som det bedöms vara rimligt att genomföra inom det angivna tidsintervallet. Projektet skapar också möjligheter att självständigt samla in och bearbeta ny information kring mindre bekanta komponenter, samt få en djupare förståelse i hur dessa kan samverka med varandra. Utvecklingen av larmsystemet innefattar inte bara tolkning och hantering av rörelsedetektorns data, utan även skapandet av ett effektivt användargränssnitt, vilket kommer att ge upphov till problem som måste lösas. Detta förväntas inte bara stärka tidigare färdigheter, utan kan även ge ny kunskap kring programmering.

2. Tekniskt Bakgrund



Figur 2.1. - Kopplingsschema över de komponenter som användes inom projektet. Här syns en mikroprocessor (ATMEGA 1284), Atmel JTAG ICE, en buzzer, en rörelsedetektor (PIR Sensor #555-28027), en LCD - display (GDM1602K), samt en knappsats (83BB1-001).

2.1. Hårdvara

2.1.1. Mikroprocessor: ATMEGA 1284

8-bitars-processorn ATMEGA1284 utgjorde grundstommen till själva projektet. ATMEGA 1284 har ett programmerbart flash-minne på 128 kB, samt ett RAM - minne på 16 kB[1]. Processorn har en maximal klockfrekvens på 8 Mhz, men för detta projekt har frekvensen satts till 1 Mhz. På processorns sitter 32 valbara dubbelriktade I/O pinnar med pull-up resistorer.

2.1.2. Rörelsedetektor: PIR Sensor (#555 - 28027)

PIR-sensorn är en rörelsedetektor med en maximal räckvidd räckvidd på 9 meter. Den maximala räckvidden uppnås om sensorns försätts i ett högre känslighetsläge, men för att förenkla testning så sattes sensorn till sitt normala tillstånd, där räckvidden endast blir 5 meter [1]. Sensorn kräver en matningsspänning på 5 V, vilket genererar infraröda signaler från sensorn. När denna signal påverkas av förändringar i miljön så svarar sensorn med att skicka en logisk etta till mikroprocessorn. För att sensorn ska fungera optimalt så krävs en kalibreringstid på 40 sekunder, där den får möjlighet att “lära sig” sin statiska omgivning [1].

2.1.3. Display: GDM1602K

GDM1602K är en två-segments LCD display med 16 pinnar; 8 av dessa allokeras till datainhämtning från processorn, 3 pinnar används till att flagga för att verkställa inhämtning av kommandon, samt inläsning och utskrift av data [3]. För justering av teckenkontrast kopplades displayens jord och spänningskälla via en potentiometer.

2.1.4. Knappsats: 83BB1-001

Knappsatsen är en fritt modifierbar 4x4 knappsats med 8 pinnar, där ena hälften representerar kolumner och den andra hälften rader [4]. Utsignaler från processorn till knappsatsens kolumner bryts vid knapptryckning, denna information kan tolkas och sparas som ett bitmönster som representerar den tryckta knappens kolumn. När strömmen till en kolumn på knappsatsen bryts, så skickas en signal tillbaka till processorn, vilket gör det möjligt att koda både rader och kolumner.

2.2. Utvecklingsmiljö

2.2.1. Atmel Studio 7

Processorn och dess interaktion med övriga komponenter i larmsystemet programmerades i programspråket C med hjälp av utvecklingsmiljön Atmel Studio 7.

3. Metod

3.1. Planering och initiering av komponenter

Projektet inleddes med ett möte där den grundläggande planen och utformningen av arbetet lades upp. Följande etapp gick ut på att skapa ett kopplingsschema innehållande de komponenter som ansågs bli relevanta för projektets genomförande. Det preliminära kopplingsschemat presenterades sedan för handledarna, och när denna hade nått handledarnas krav så delades samtliga komponenter ut. Därefter kopplades kretsen ihop enligt kopplingsschemat, med speciellt fokus på kabelhantering för att underlätta och effektivisera framtida felsökning av komponenter. Det fortsatta arbetet skedde på förutbestämda tider och vid varje tillfälle deltog samtliga gruppmedlemmar. Tolkning av datablad, diskussioner kring programmeringens struktur, samt själva kodningen, delades dynamiskt upp mellan gruppmedlemmarna under varje arbetstillfälle.

När den första prototypen hade kopplats färdigt så skiftades fokus till hur logiken i larmsystemet skulle utformas. Eftersom displayen kunde fungera som en tydlig källa till feedback från ATMEGA:n, så beslutades det att arbetet skulle inledas med att få LCD:n att aktiveras. Detta visade sig dock inte vara en optimal väg in i utvecklingsprocessen, vilket resulterade i att gruppen valde att istället först angripa de mest grundläggande egenskaperna hos ett larm, sensorn och ljudsignalen. Med en fungerande sensor och buzzer, så återgick arbetet till interaktionen med displayen. Efter att displayen hade integrerats så gick arbetet över till att koda knappsatsen.

3.2. Mjukvara

Med alla komponenter på plats inleddes utvecklingen av den programvara som skulle möjliggöra larmets olika funktioner. Till en början utnyttjades C:s standardbiblioteket för delay, vilket inledningsvis underlättade testandet av komponenter. Eftersom många av komponenterna krävde fördröjningar mellan pulser, så blev det snabbt tydligt att den färdiga delay-funktionen skulle hindra projektet framöver. För att alla funktioner skulle kunna köras problemfritt så krävdes det att egna delay-funktioner skapades, en för 1, en för 10, respektive en för 100 millisekunders fördröjning.

3.1. Källkritik

De huvudsakliga källorna angående de olika komponenterna var vederbörandes datablad och dessa ansågs av gruppen vara ytterst trovärdiga. I övrigt utnyttjades källor såsom olika hemsidor och forum. Trovärdigheten hos dessa källor kan ej bekräftas helt och hållet, men då dessa bidrog med information som underlättade felsökning av komponenter och tolkning av databladen, vilket resulterade i en fungerande prototyp. Därför anser vi att trovärdigheten hos sådana källor i alla fall inte hade någon negativ inverkan på projektets slutresultat.

4. Analys

4.1. Val av komponenter

För att skapa prototypen behövdes en processor som kan hantera särskilda instruktioner och möjliggöra samverkan mellan olika komponenter. Det beslutades att den skulle bestå av en sensor som kan reagera på rörelser, en högtalare för att indikera att larmet har triggats, en knappsats för att tillåta interaktion med larmsystemet och en bildskärm för att ge ett användargränssnitt att interagera utifrån. Processorn valdes baserat på gruppens tidigare erfarenhet med denna, men också på grund av dess specifikationer. En självkalibrerande infraröd sensor valdes eftersom den med enkla medel går att integrera i larmsystemet på ett tillfredsställande sätt. Bildskärmen som användes kan visa två textrader, vilket ger möjlighet att visa en rubrik på översta raden, medan den undre raden visar användarens inmatade data. Högtalaren behöver endast avge ett ljud när larmet befann sig i armerat tillstånd följt av att sensorn triggas. Den valda buzzern är avger ett starkt ljud då ström tillförs, vilket möjliggör enkel integrering i systemet.

4.2 Problemformulering

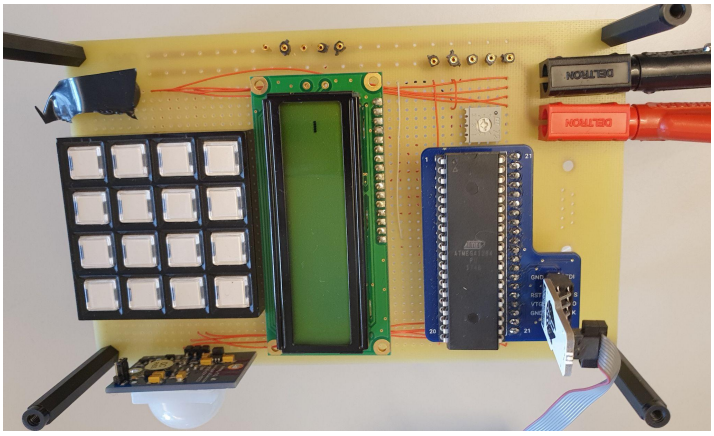
Till hårdvaran användes kablar i olika färger för att enklare kunna identifiera vart olika komponenter hade kopplats. Med detta underlättades även felsökningsprocessen, då det blev möjligt att kontrollera varje enskild anslutning med en logikprob som visar dess signal. Nyttan med detta tillvägagångssätt blev tydligt vid installationen av de olika komponenterna. Speciellt tydligt blev det när LCD-skärmen skulle initieras. Skälet till varför bildskärmen inte reagerade på insignaler var att D-portarna som den var kopplad till var förskjutna ett steg från deras angivna position i databladet. Detta kunde kunde upptäckas och åtgärdas tack vare kabelhantering och felsökning med hjälp av logikproben. Ytterligare ett problem var att knappsatsen enbart svarade på hälften av knapparna eftersom knappsatsens alla signaler skulle skickas från och till kortets A-portar. För att få samtliga knappar att svara behövdes alltså respektive sida av anslutningarna, som tidigare antogs vara rad respektive kolumn, delas upp i två mindre delar, där ena halvan representerade rader och den andra kolumner. Detta kunde enklast åtgärdas genom att använda bitmasker i koden.

5. Resultat

5.1. Koppling av kretsen

Inledningsvis kopplades prototypen upp enligt figur 1.2 där mikroprocessorn ATMEGA1284 utgjorde basen för hela systemet. Eftersom processorns I/O portar är indelade i fyra sektioner, kallade A, B C och D. Gruppen valde att utnyttja denna struktur för att underlätta konstruktion och felsökning. Den slutgiltiga kopplingen såg till stor del ut som planerat från början, men vissa justeringar blev nödvändiga allt eftersom felkopplingar upptäcktes. Till mikroprocessorns A-port kopplades det en knappsats där hälften av pinnarna representerade rader medan den andra halvan utgjorde kolumner.

Vid programmering av knappsatsen beslutade gruppen vilka knappar som skulle motsvara siffrorna 0 till 9, vilken knapp som triggar armering och desarmering, en knapp för inmatning av ny säkerhetskod och en reset-knapp som återställer systemet till startläget utan att radera sparade koden. Denna programkod finns i bilagor.



Figur 5.1. - Bild på den färdiga prototypen. Till vänster syns knappsatsen, under den sitter rörelse-sensorn. Mitterst sitter LCD-displayen. Till höger om den sitter processorn (ATMEGA1284).

Till port B på processorn kopplades det en två-raders LCD display. Pinnarna på LCD-displayen är till för datainhämtning från mikroprocessorn. Med andra ord ser dessa till att allt som matas in av en användare översätts till 8-bitars bitmönster som LCD-displayen kan ta emot och tolka som instruktioner. Kvarstående tre anslutningar, enable, read/write och reset, kopplades till port D på processorn.

De resterande pinnarna på displayen var för matningsspänning och jordning. Mellan dessa två pinnar kopplades en potentiometer för att möjliggöra kontrastjusteringar på displayen.
Pinnarna på port C på lämnades fria till JTAG:en som ser till att mikroprocessorn kan tolka och exekvera programkoden. Slutligen kopplades rörelsedetektorn och buzzern också till port D.

5.2. Programmering av larmsystemet

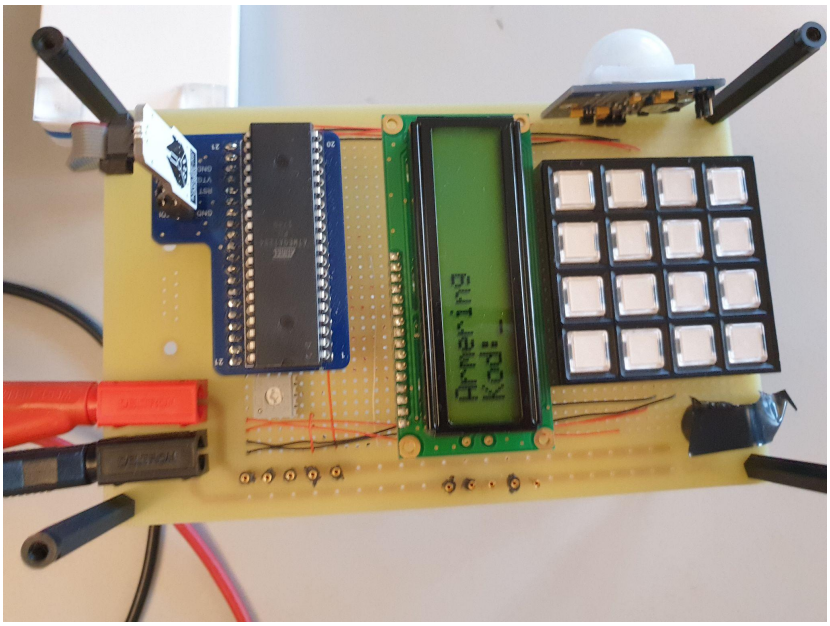
När alla komponenter hade anslutits enligt databladet blev det nödvändigt att bestämma data-riktningen för alla pinnar. Till port A på processorn har data-riktningen satts till output för de första fyra bitarna som motsvarar rader i knappsatsen. Då knappsatsen består av brytare så bryts strömmen vid knapptryck vilket gör det möjligt att identifiera knapprader. Med detta skickas även signaler tillbaka till processorn i form av ett unikt 8-bitars bitmönster för varje knapp. För att identifiera alla bitmönster som individuella knappar så skapades en C-funktion där varje knapp kan läsas av och tilldelas en karaktär.

Vidare sattes samtliga pinnar till utgångar på port B då dessa skickar operationer i form av bitmönster till LCD-skärmen. Exempel på en operation kan hittas i programkoden i metoden lcd_init() där bitmönstret 00111000 innebär att skärmen är på, cursor är på samt att cursors blinkande är av. På detta vis blev det möjligt att skapa ett simpelt användargränssnitt.

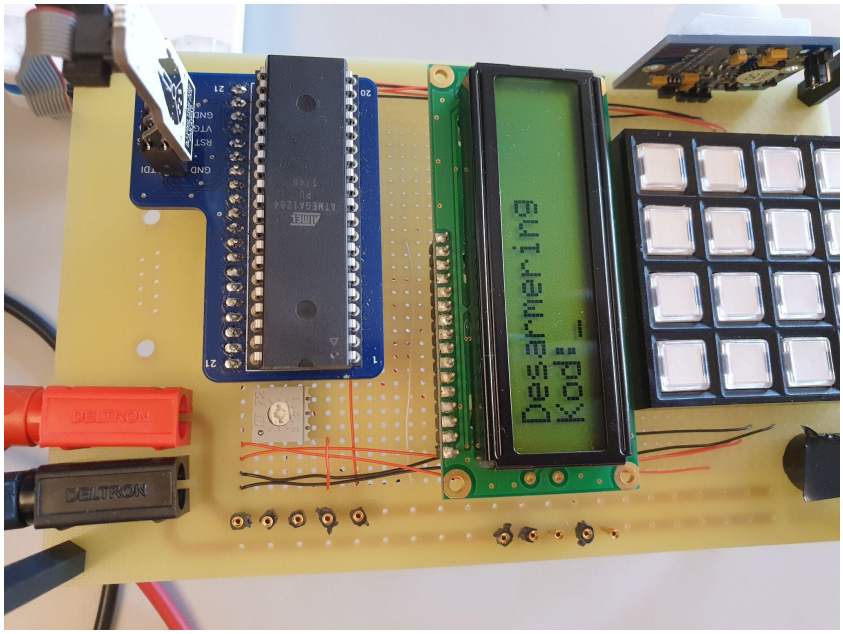
Rörelsesensorn svarar på förändringar i omgivningen genom att skicka en logisk etta till processorn varpå denna information kan hanteras på olika sätt i programkoden. Om larmet desarmeras så ignoreras insignalen så att personer kan passera sensorn. När larmet däremot armeras, svarar systemet på sensorns insignal med att aktivera buzzern. Sammanfattningsvis anses prototypen för larmsystemet uppfylla de krav som sattes upp från början.

6. Slutsats

Den slutgiltiga prototypen anser vi uppfylla samtliga tidigare uppsatta krav. Larmsystemet har en fungerande sensor, som svarar på förändringar i omgivningen med att trigga en buzzer. Larmet kan armeras genom inmatning av en lösenkod via knappsatsen, förutsatt att användaren skriver in rätt kod. Om fel kod matas in så sker ingenting. När larmet väl befinner sig i armerat läge så kan man på samma vis desarmera larmet. Då avaktiverar man sensorn och stänger av buzzern ifall den ljuder. Lösenkoden kan också ändras fritt av användaren.



Figur 6.1: Armeringsfunktionen demonstreras.



Figur 6.2: Desarmeringsfunktionen demonstreras.

6.2. Framtida utvecklingsmöjligheter

En uppenbar brist i larmsystemet är att ändring av lösenkod för närvarande inte kräver att den tidigare lösenkoden matas in rätt. Detta innebär givetvis enorma säkerhetsbrister, men detta hade enkelt kunnat åtgärdas i mån av tid. För övrigt finns det framtida möjligheter att integrera bland annat keycards som eliminerar upprepad kodinmatning, trådlös kommunikation med larmcentral och ljussignaler.

8. Källförteckning

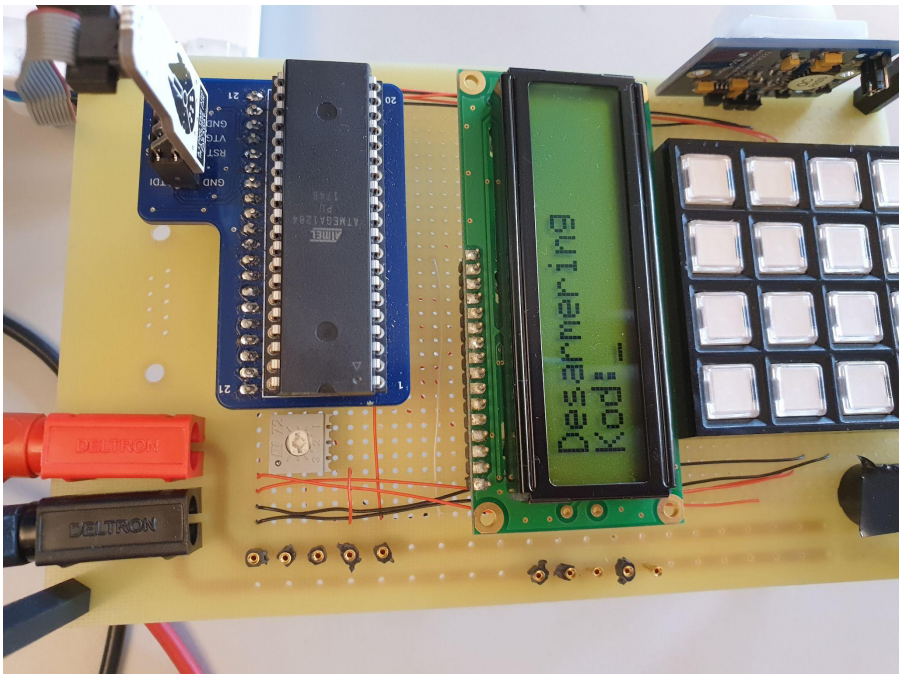
[1] *ATMEGA1284* Atmel Corporation. San Jose, USA, 2016. Tillgänglig: https://www.eit.lth.se/fileadmin/eit/courses/datablad/Processors/ATmega1284.pdf (Hämtad 2022-05-02).

[2] *Pir Sensor (#555-28027)* Parallax Inc. Rocklin, Kalifornien, USA, 2014. Tillgänglig: https://www.eit.lth.se/fileadmin/eit/courses/datablad/Lawicell/555-28027-PIR-Sensor-Prodcut-Doc-v2.2.pdf (Hämtad 2022-05-02).

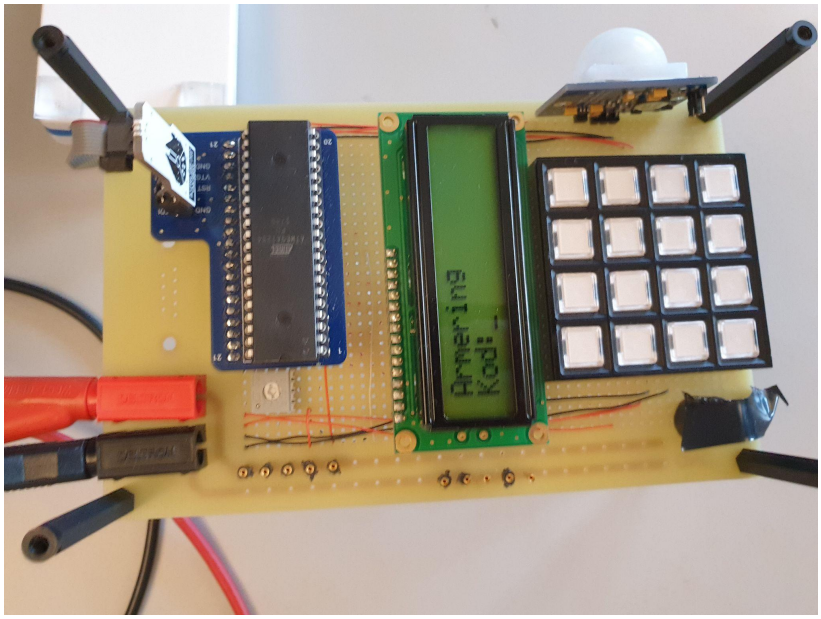
[3] *GDM1602K* Xiamen Ocular Optics CO. Xiamen, Kina, 2001. Tillgänglig: https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf (Hämtad 2022-05-02)

[4] *83BB1-001* Grayhill Inc. LaGrange, Illinois, USA, 2022. Tillgänglig: https://www.grayhill.com/documents/83-Datasheet (Hämtad 2022-05-02).

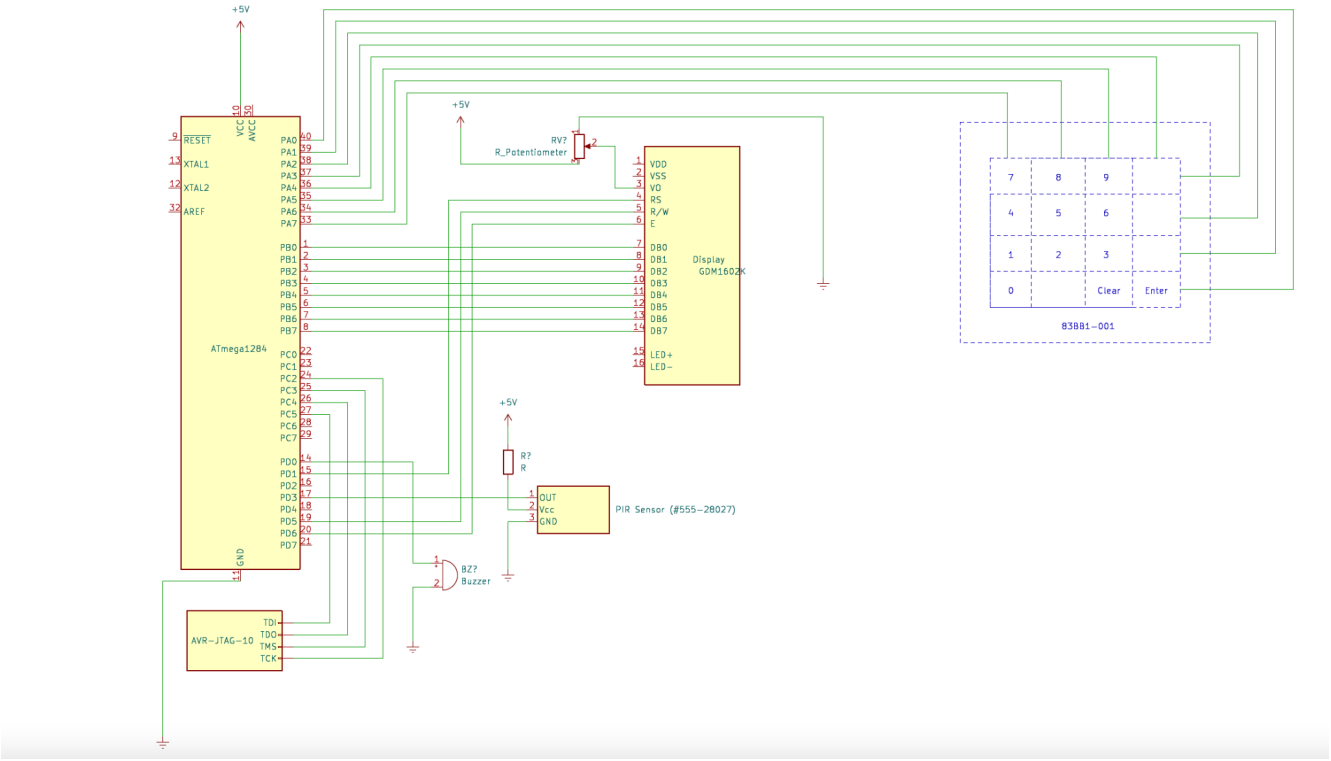
9. Bilagor



Figur 6.2: Desarmering



Figur 6.1: Armering



Figur 2.1: Kopplingsschema

Källkod

```
#define F_CPU 1000000UL
#define LCD_dat PORTB
#define rs PORTD1 // rs pin 1
#define en PORTD5 // en pin 5
#define buzzer PORTD0

#include <avr/io.h>
#include <avr/delay.h>
#include <string.h>
#include <stdio.h>

char keypressed = 0;
uint8_t keyPressCode;

volatile char charData;
volatile char arm[] = "Armering                                    Kod:";
volatile char ratt[] = "Ratt kod";
volatile char fel[] = "Fel kod";
volatile char disarm[] = "Desarmering                                Kod:";
volatile char ratt_d[] = "Desarmerad";
char ange_ny_kod[] = "Ange ny kod:                                ";

char alarm_toggle_bit = 0;
char sensor_toggle_bit = 0;
char ready_bit = 2;
char pin_bit;

volatile char saved_code[] = "1234";
volatile char input_code[5];
int codeCharCounter = 0;
int counter;

/* Delays */
void delay_1ms()
{
      asm volatile (
      "    ldi  r18, 2 \n"
      "    ldi  r19, 75\n"
      "1:  dec  r19    \n"
      "    brne 1b     \n"
      "    dec  r18    \n"
      "    brne 1b     \n"
      "    rjmp 1f     \n"
      "1:   \n"
      );
```

```
}

void delay_10ms()
{
      asm volatile (
      "    ldi  r18, 13\n"
      "    ldi  r19, 252     \n"
      "1:  dec  r19     \n"
      "    brne 1b      \n"
      "    dec  r18     \n"
      "    brne 1b      \n"
      "    nop    \n"
      );

}

void delay_100ms()
{
      asm volatile (
      "    ldi  r18, 130     \n"
      "    ldi  r19, 222     \n"
      "1:  dec  r19     \n"
      "    brne 1b      \n"
      "    dec  r18     \n"
      "    brne 1b      \n"
      "    nop    \n"
      );

}

/* Handles LCD functions */
void lcd_cmd(unsigned char ch)
{
      LCD_dat = ch;
      PORTD &= 0b11111101;//(0 << rs);
      PORTD |= (1 << en);
      delay_1ms();
      PORTD &= 0b11011111;//(0 << en);
}

/* Handles single character output to the LCD */
void lcd_data(unsigned char ch)
{
      LCD_dat = ch;
      PORTD |= (1 << rs);
      PORTD |= (1 << en);
```

```
	PORTD &= 0b11011111;//(0 << en);  // alla bitar = 0, finns risk att
PD0 påverkas, se över
	//PORTD |= (0 << rs);
}

/* Initializes the LCD */
void lcd_init()
{
	lcd_cmd(0x38); // (8 bitar, 1 rad, 5x8 punkter)
	delay_100ms();
	lcd_cmd(0x0e); // (Display on, Cursor on, Cursor blink OFF)
	delay_100ms();
	lcd_cmd(0x01); // (Clear display)
	delay_100ms();
}

/* Assigns value to keyPressCode corresponding to the column and row
hexadecimal identifiers */
void keypadscan()
{
	DDRA = 0x0f;
	PORTA = 0xf0;
	if(PINA == 0b11110000) return
	delay_1ms();
	keyPressCode = PINA;
	DDRA = 0b11110000;
	PORTA = 0b00001111;
	delay_1ms();
	//x = PINA; // Använde dessa som korta delays innan, verkar behövas
i metoden
	keyPressCode |= PINA;
}

/* Translates value of keyPressCode to assigned character and prints text
to the display, and otherwise assigns tasks to buttons */
void key_read()
{
	if(keyPressCode == 0x77)
	{
		lcd_data('1');
		delay_100ms();
		delay_100ms();
		charData = '1';

	}
	if(keyPressCode == 0xb7)
	{
```

```
        lcd_data('2');
        delay_100ms();
        delay_100ms();
        charData = '2';
    }
    if(keyPressCode == 0xe7)
    {

        lcd_data('3');
        delay_100ms();
        delay_100ms();
        charData = '3';
    }
    if(keyPressCode == 0xd7)
    {

        lcd_data('4');
        delay_100ms();
        delay_100ms();
        charData = '4';
    }
    if(keyPressCode == 0x7b)
    {

        lcd_data('5');
        delay_100ms();
        delay_100ms();
        charData = '5';
    }
    if(keyPressCode == 0xbb)
    {

        lcd_data('6');
        delay_100ms();
        delay_100ms();
        charData = '6';
    }
    if(keyPressCode == 0xeb)
    {

        lcd_data('7');
        delay_100ms();
        delay_100ms();
        charData = '7';
    }
    if(keyPressCode == 0xdb)
```

```
{

      lcd_data('8');
      delay_100ms();
      delay_100ms();
      charData = '8';
}
if(keyPressCode == 0x7d)
{

      lcd_data('9');
      delay_100ms();
      delay_100ms();
      charData = '9';
}
if(keyPressCode == 0xbd)
{

      lcd_data('0');
      delay_100ms();
      delay_100ms();
      charData = '0';
}
if(keyPressCode == 0xde)
{

      lcd_cmd(0x01);
      delay_100ms();
      delay_100ms();
}
if(keyPressCode == 0xdd)
{
      charData = 'x';

}
/* Arms the alarm */
if(keyPressCode == 0xee)
{
      delay_100ms();
      delay_100ms();
      charData = 'a';
}
/* Changes arming and disarming password */
if(keyPressCode == 0x7e)
{
      lcd_cmd(0x01);
      delay_100ms();
```

```
            for (int n = 0; n < strlen(ange_ny_kod); n++) // meddelandet
skrivs ut
            {
                  lcd_data(ange_ny_kod[n]);
            }
            int i = 0;
            delay_100ms();
            keyPressCode = 0x00;
            while(i<4)
            {
                  keypadscan();
                  if(keyPressCode != 0x00)
                  {
                        key_read();
                        saved_code[i] = charData;
                        i++;
                        delay_100ms();
                  }
                  delay_100ms();
            }
            lcd_cmd(0x01);
            delay_100ms();
      }
      keyPressCode = 0x00;
}
/* Checks if user input matches working password and prints relevant text
to LCD */
void code_check()
{
      lcd_cmd(0x01);
      delay_100ms();
      /* Notifies user of arming and disarming status when entering
password */
      if(sensor_toggle_bit == 1)
      {
            for (int n = 0; n < strlen(disarm); n++) // meddelandet skrivs
ut
            {
                  lcd_data(disarm[n]);
            }
      } else {
            for (int n = 0; n < strlen(arm); n++) // meddelandet skrivs ut
            {
                  lcd_data(arm[n]);
            }
      }
```

```
	/* Automatically stores input password when input reaches original password length */
	for(int n = 0; n < strlen(saved_code); n++) // kod matas in, sparas i string
	{
		while(1)
		{
			keypadscan();
			if(keyPressCode != 0x00)
			{
				key_read();
				input_code[n] = charData;
				break;
			}
		}
	}
	/* Compares input password to true password and prints match status */
	for(int n = 0; n < strlen(saved_code); n++)
	{
		if(input_code[n] != saved_code[n])
		{
			lcd_cmd(0x01);
			delay_100ms();
			for (int n = 0; n < strlen(fel); n++) // "fel kod" skrivs ut
			{
				lcd_data(fel[n]);
			}
		} else if(n + 1 == strlen(saved_code) && input_code[n] == saved_code[n])
		{
			lcd_cmd(0x01);
			delay_100ms();
			for (int n = 0; n < strlen(ratt); n++) // rätt kod
			{
				lcd_data(ratt[n]);
			}
			if(sensor_toggle_bit == 1)
			{
				sensor_toggle_bit = 0;
				buzzer_toggle();
				} else {
				sensor_toggle_bit = 1;
				ready_bit = 1;
			}
			delay_100ms();
```

```
            }
      }
      pin_bit = PIND;
      delay_100ms();
}
/* Switches the alarm buzzer on or off depending on previous state */
void buzzer_toggle()
{
      if (alarm_toggle_bit == 1)
      {
            PORTD &= ~(1 << buzzer);
            alarm_toggle_bit = 0;
            delay_100ms();
            } else {
            PORTD |= 0b00000001;
            alarm_toggle_bit = 1;
            delay_100ms();
      }
}

void lcd_cmd(unsigned char ch);
void lcd_data(unsigned char ch);

int main(void)
{
      DDRD = 0b11110111;
      DDRB = 0b11111111;
      PORTD = 0x00;
      DDRA = 0x0f;
      lcd_init();

      while(1)
      {

            keypadscan();
            if(keyPressCode != 0x00)
            {
                  key_read();
                  if(charData == 'a') // armeringsknappen
                  {
                        code_check(); // kod matas in, jämförs med gammal
kod

                  }
                  if(charData == 'x')
                  {
                        buzzer_toggle();
```

```
                        charData = 'z';
                }
        }
        /* Allows the buzzer to activate when the device is armed */
        if(ready_bit == 1 && sensor_toggle_bit == 1)
        {
                delay_10ms();
                pin_bit = PIND;
                delay_10ms();
                while(pin_bit == 0b00001010 || pin_bit == 0b00001011)
                {
                        pin_bit = PIND;
                        delay_10ms();

                }
                while(pin_bit == 0b00000010 || pin_bit == 0b00001011)
                {
                        pin_bit = PIND;
                        delay_10ms();
                }
                while(pin_bit == 0b00001010 || pin_bit == 0b00001011)
                {
                        buzzer_toggle();
                        ready_bit = 0;
                        break;
                }

        }
    }
}
```