Projektrapport - Cykellarm

LUNDS UNIVERSITET

Program: Industriell Ekonomi
Handledare: Bertil Lindvall & Christoffer Cederberg
Studenter: Jonatan Arnlund, Joakim Wahl & Anton Lindström
Datum: 2019-05-22

Innehållsförteckning

1. Inledning

Syftet med kursen Digitala projekt är att skapa en bild av hur kopplingen mellan hårdvaru- och mjukvaruutveckling fungerar för att på så sätt ge studenter en bild av hur utvecklingsarbetet kan se ut i industrin. Projektarbetet görs i grupper om 2-3 civilingenjörsstudenter och tillsammans ska de bygga hårdvaran och programmera mjukvaran till en lämplig produkt.

I den lilla studentstaden Lund i södra Sverige är det absolut vanligaste färdmedlet för studenter cykel, på grund av de generellt sett mycket korta avstånden i staden. Cykeln gör studenten flexibel, men det tillkommer också en del nackdelar. En av dessa är att cyklar med jämna mellanrum blir stulna trots kraftiga cykellås, vilket kan skapa stora och oväntade utgifter för studenter som redan lever på hårt åtstramade budgetar. Detta problem har vi försökt åtgärda i samband med kursen Digitala projekt på Lunds Tekniska Högskola, med hjälp av ett elektroniskt cykellås som slår larm om en låst cykel förflyttas från punkten den låstes på.

2. Produkt

2.a Produktbeskrivning

Detta cykellås används i kombination med ett traditionellt cykellås och försvårar arbetet att stjäla cykeln markant. Efter att cykeln har blivit låst med detta cykellås, kommer en alarmerande ljudsignal börja tjuta för att tillkalla uppmärksamhet om cykeln fysiskt förflyttas från den plats den blev låst på. Detta gör det betydligt mer komplicerat för cykeltjuven att lämna platsen med cykeln utan att dra uppmärksamhet till sig. Cykellåset aktiveras med studentens LU-kort genom RFID-läsning och använder sig därefter av en accelerometer med tre frihetsgrader för att detektera rörelser i låst tillstånd.

2.b Kravspecifikation

Den färdiga prototypen ska uppfylla följande krav:

- Prototypen skall aktiveras med hjälp av förvalda LU-kort. Enbart de förvalda LU-korten ska kunna aktivera låset.
- Då prototypen är låst, skall stora rörelser i horisontalled och vertikalled utlösa en larmsignal som indikerar cykel rör på sig utan att användaren har låst upp låset. En röd lampa ska också blinka.

- Då prototypen befinner sig i låst tillstånd, och användarens LU-kort placeras mot sensorn skall låset avaktiveras.
- Då låset är avaktiverat skall en grön lysdiod lysa för att indikera detta.

3. Hårdvara

3.b Komponenter

Processor: ATmega16 8-bit microcontroller med 40 pins. Används för att styra övriga komponenters beteende.

JTAG: JTAG ICE 3 används för att koppla processorn till en dator. Detta möjliggör programmering och debugging.

Accelerometer: ADXL 335. Används för att detektera rörelser av låset, då det befinner sig i låst läge.

LM3490 Low-Dropout Regulator: Används för att konvertera 5 V (VCC) till 3,3 V för accelerometern.

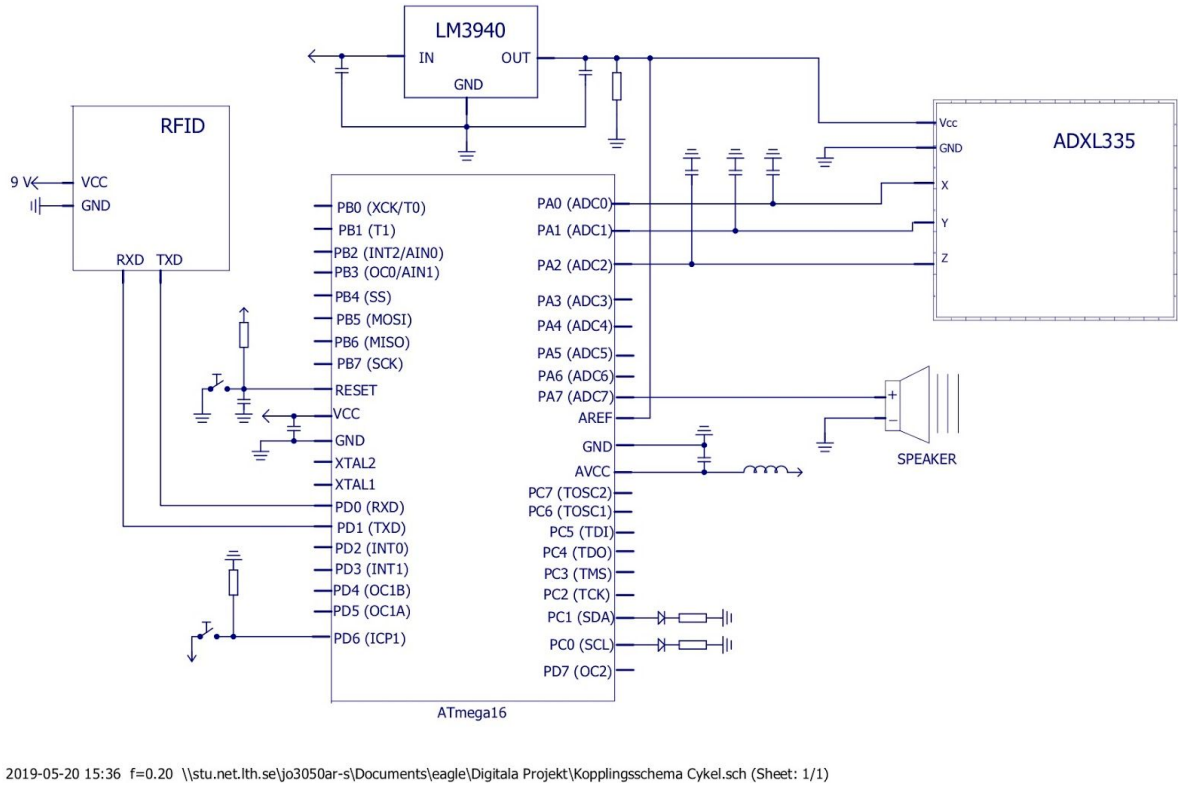
Lysdiod, grön: Används för att indikera att låset är avaktiverat.

Lysdiod, röd: Används för att indikera att låset är akt

Högtalare, KXG1205C: Används för att sätta igång larmet

Kortläsare - DF 750k användes som RFID-läsare med syftet att läsa av nycklar för att låsa upp låset. Knappsatsen används ej.

4. Kopplingsschema



figur 1. Kopplingsschema för cykellåset

5. Mjukvara

Mjukvaran för cykellåset är skrivet i C med hjälp av programmet Atmel Studio 7. JTAG har använts programmering samt debugging av processorn. För att studera källkoden hänvisas läsaren till Appendix A.

Källkoden är indelad i tre header-filer och tre c-filer. Mainfilen plockar upp interruption som kommer både från RXC och från ADC. RXC tar emot data från kortläsaren och jämför denna mot lagrad kortdata för att kontrollera om användaren är legitimerad eller ej. Är användaren legitimerad så avaktiveras larmet och en grön lampa tänds. Annars händer ingenting.

ADC tar in data från accelerometern och om larmet är aktiverat jämförs den här datan mot ett förbestämt intervall som bestämmer om larmet ska gå igång eller inte. Om någon försöker att förflytta modulen i aktiverat tillstånd kommer en röd lampa att börja blinka samtidigt som ett larmljud går igång. Om larmet är avaktiverat kommer data inte avläsas från accelerometern.

6. Metod

6.a Planering

Under kursens första vecka bestämdes vilka studenter gruppen skulle bestå av. Därefter började arbetet med att välja vad som skulle tillverkas. Alternativen var många, men det skulle också vara möjligt att implementera under den givna tiden, kompetensen och kunskapen hos studenterna samt att komponenterna skulle finnas tillgängliga. Under andra veckan under laboration 1 på kursen, presenterades idén för kursens handläggare som ansåg att idén var genomförbar, med några modifieringar på ursprungsidén. Initialt var tanken att cykellåset skulle innehålla en GPS, vilken skulle göra det möjligt för användaren att spåra sin cykel vid stöld. Denna idé skulle dock bli för tidskrävande och istället bestämdes att cykellåset skulle innehålla ett tjutande larm som aktiveras vid stöld istället. En första kravspecifikation skapades också för att precisera vad prototypen skulle kunna åstadkomma.

Därefter konstruerades ett översiktligt tidsschema över vad som skulle ske under de närmaste veckorna, för att på så sätt få en helhetsbild över projektet och dess beståndsdelar. Med tidsplanen satt började arbetet med att ta fram ett förslag till kopplingsschema för prototypen, utefter den satta kravspecifikationen. Kopplingsschemat visades sedan för handledaren som efter modifieringar kunde ge ett initialt godkännande av schemat. Noterbart är att detta kopplingsschema behövde modifieras ett fåtal gånger till, till den slutgiltiga versionen som ses i figur 1. Kopplingsschemat togs fram med hjälp av mjukvaran *Eagle*. Därefter delades verktyg ut och det var dags att börja bygga.

6.b Montering

Med samtliga komponenter i ägo började monteringen steg för steg. För gruppen var det viktigt att se till att fel i monteringen märktes tidigt och inte i slutskedet, för att lättare kunna felsöka prototypen. Därför testades varje komponents funktionalitet löpande innan nästa komponent monterades. I de situationer där det var möjligt skrevs också programmeringskoden för den specifika komponentens funktionalitet parallellt för att underlätta processen så mycket som möjligt, men också för att skapa en djupare förståelse för de olika delarna i prototypen.

Det första steget i monteringen var att montera processorn och koppla den till både jord och spänningskällan. Detta var en del i processen som tog relativt lång tid, även om det i efterhand kanske ses som en av de enklare delarna av arbetet. Detta kan förklaras med att gruppen hade en väldigt sparsam erfarenhet av området sedan tidigare, vilket gör att det krävdes en del tid för att lära sig att hantera de olika verktygen.

Efter att processorn var inkopplad skulle de båda lysdioderna monteras och kopplas till processorn. Detta gjordes tillsammans med en av knapparna, och för att bekanta sig med hur koden fungerar. Då kunde vi nämligen programmera så att lamporna började lysa då man tryckte på knappen, även om detta inte var det slutgiltiga användningsområdet för prototypen. Sedan monterades högtalaren till processorn, och knappen programmerades om så att en knapptryckning nu istället skulle generera alarmljudetfrån högtalaren.

Därefter kopplades accelerometern till processorn. Då accelerometern krävde en spänning på 3,3 volt, och spänningen från VCC annars var 5 volt, krävdes något för att sänka spänningen in i accelerometern. För detta ändamål användes LM3490 Low-Dropout Regulatorn, som sänker spänningen innan den når accelerometern, till just 3,3 volt. Under denna process kortslöts kretsen flertalet gånger, på grund av missförstånd i kopplingen, men efter mycket om och men löstes problemet.

6.c Programmering

Från början skrevs ett separat program för att kontrollera att ljud och ljus var korrekt inkopplat, samt att processorn fungerade. När detta verifierats påbörjades det riktiga kodarbetet.

Kodandet utgick från start med en main-metod. Det första som implementerades var accelerometern och detta gjordes genom filerna adc.c och adc.h. Dess funktionalitet är att skriva till ADCSRA så att vi kan ta emot signaler på ad-portarna och att skifta mellan att läsa av x,y och z genom att skriva bitar till ADMUX.

Därefter implementerades alla larm-funktioner, dvs. en funktion som gör så att larmet piper och blinkar och en funktion för att slå på larmet och en för att slå av det.

Därefter implementerades uart.h och uart.c med syfte att kontrollera och styra kortläsaren. Kortläsaren initieras genom att skriva bitar till UCSRB, UBRRL och UCSRC.

Nästa steg var att skriva en metod i main-klassen för att jämföra inkommande kortdata mot sparade kortnummer. Kortnumret togs fram genom att läsa av output från kortläsaren i output-fönstret i Atmel.

7. Diskussion

Sammanfattningsvis har denna kurs varit mycket lärorik och underhållande. Stora delar av kursen har bestått av många timmar med att försöka komma över hinder som för tillfället kändes helt olösbart. I och med att samtliga i gruppen innan denna kurs hade väldigt begränsade förkunskaper om både elektronik, men också denna typ av programmering, så hade vi inte så mycket verktyg att använda oss av för att fundera ut lösningar. Dock allteftersom att problemen har lösts, ofta med hjälp av handledare, så har vi byggt på vår kunskapsbas och allt eftersom har de till synes olösliga problemen blivit allt färre. Det ska också tilläggas att tillfällena då vi stött på motgångar och inte vetat hur vi ska gå tillväga, har varit de perioder då vi faktiskt har lärt oss som mest. Det finns också en stor tillfredsställelse i att lösa ett problem som man har känts omöjligt initialt.

Det svåraste under projekten har varit att förstå kopplingen mellan mjukvara och hårdvara. Detta har berott på att det blev en hög tröskel för samtliga i gruppen att kliva över med både ett nytt programmeringsspråk samt hårdvaran som vi inte har arbetat med sedan tidigare. Denna tröskel gjorde att det var mycket tid i början som lades på förståelse och misstag, vilket var frustrerade till viss del, men man kändes alltid nära lösningen, vilket ändå motiverade en att fortsätta.

Sammanfattningsvis är vi mycket nöjda med produkten. När det kommer till funktionaliteten så gör produkten i stora drag det vi tänkte från början, men några få förändringar för saker som skulle ta för lång tid att implementera. För att ta det ett steg längre skulle nästa steg vara att lägga några tankar på designen och få produkten mindre samt mer kompakt, detta har nämligen varit av låg prioritet då funktionaliteten har stått i fokus. Ett annan utveckling hade varit att försöka få produkten att bli vattentät, då det svenska klimatet ibland bjuder på mängder med regn. I det stora hela har kursen varit extremt lärorik, och samtliga i gruppen känner att en en bra grund har byggts för framtiden.

10. Appendix A: Källkod

```
/*
 * glob_header.h
 *
 * Created: 2019-05-08 18:08:00
 *  Author: jo3050ar-s
 */


#ifndef GLOB_HEADER_H_
#define GLOB_HEADER_H_

#define F_CPU 8000000UL
#define LED1 0
#define LED2 1
#define BTN1 6
#define SPK1 7

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


#endif /* GLOB_HEADER_H_ */

/*
 * adc.h
 *
 * Created: 2019-05-08 18:06:33
 *  Author: jo3050ar-s
 */


#ifndef ADC_H_
#define ADC_H_

#include "glob_header.h"

void adc_init();
int adc_start();

#endif /* ADC_H_ */

/*
 * adc.c
```

```
 *
 * Created: 2019-05-08 18:06:20
 *  Author: jo3050ar-s
 */

#include "adc.h"
#include "glob_header.h"

uint8_t adc_input;


void adc_init() {

	ADCSRA |= (1 << ADEN) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1);

}

int adc_start() {
			switch(adc_input) {
			case 0:
			ADMUX &=~ (1<<MUX1);
			adc_input++;
			break;
			case 1:
			ADMUX |= (1<<MUX0);
			adc_input++;
			break;
			case 2:
			ADMUX &=~ (1<<MUX0);
			ADMUX |= (1<<MUX1);
			adc_input = 0;
			break;
		}
		ADCSRA |= (1 << ADSC);


		return adc_input;
}

/*
 * uart.h
 *
 * Created: 2019-05-14 18:31:37
 *  Author: jo3050ar-s
 */
```

```
#ifndef UART_H_
#define UART_H_

#include "glob_header.h"

void uart_init(void);
void uart_transmit(uint8_t data);
unsigned char uart_recieve (void);

#endif /* UART_H_ */
/*
 * uart.c
 *
 * Created: 2019-05-14 18:31:59
 *  Author: jo3050ar-s
 */


#include "uart.h"


void uart_init() {

	UCSRB |= (1 << RXCIE) | (1 << RXEN) | (1 << TXEN);
	UBRRL  =  51;
	UCSRC|=(1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);




}

void uart_transmit(uint8_t data) {
	/* Wait for empty transmit buffer */
	while(!( UCSRA & (1<<UDRE)));
	/* Put data into buffer, sends the data */
	UDR = data;
}

unsigned char uart_recieve (void) {
	while(!(UCSRA) & (1<<RXC)); //wait while data is being received
	return UDR; //return 8-bit data
}

/*
```

```
 * GccApplication3.c
 *
 * Created: 2019-04-17 14:37:42
 * Author : anton
 */


#include "glob_header.h"
#include "adc.h"
#include "uart.h"
#include <avr/interrupt.h>
volatile uint16_t X_LIMIT_UPPER = 545;
volatile uint16_t X_LIMIT_LOWER = 485;
volatile uint16_t Y_LIMIT_UPPER = 545;
volatile uint16_t Y_LIMIT_LOWER = 485;
volatile uint16_t Z_LIMIT_UPPER = 660;
volatile uint16_t Z_LIMIT_LOWER = 590;

volatile uint16_t adc_res;

volatile uint8_t rx_data[5];
volatile uint8_t ANTONS_KORT[4] = {29,26,44,20};
volatile uint8_t JOCKES_KORT[4] = {92,221,89,16};

volatile uint8_t rx_cnt;
volatile unsigned char rx_8bit;
int currentCardID;
int dataLength;
int alarm_fired;
uint8_t b_state;
uint8_t adc_input2;
uint8_t alarm_armed;


void alarm_arm() {
	alarm_armed = 1;
	PORTC &=~ (1<<LED2);
}
void alarm() {

	while(alarm_fired == 1) {

		PORTC |=(1<<LED1);
		PORTA |=(1<<SPK1);
		_delay_ms(250);
		PORTC &= ~(1<<LED1);
```

```
                PORTA &=~ (1<<SPK1);
                _delay_ms(250);
        }
}

int main (void) {
        uart_init();
        adc_init();
        sei();
        adc_start();

        DDRC |= (1<<LED1) | (1<<LED2);
        DDRA |= (1<<SPK1);
        DDRC |= (0<<BTN1);
        alarm_arm();
        while(1){
                if(alarm_fired==1) {
                        alarm();
                }
        }
}



void movement () {

        if(adc_res != 0) {
                switch(adc_input2) {
                        case 0:
                        adc_input2 ++;
                        if(adc_res < X_LIMIT_LOWER || adc_res > X_LIMIT_UPPER) {
                                alarm_fired=1;
                        }
                        break;
                        case 1:
                        adc_input2 ++;
                        if(adc_res <Y_LIMIT_LOWER || adc_res > Y_LIMIT_UPPER) {
                                alarm_fired=1;
                        }
                        break;
                        case 2:
                        adc_input2 = 0;
                        if(adc_res < Z_LIMIT_LOWER || adc_res > Z_LIMIT_UPPER) {
                                alarm_fired=1;
                        }
                        break;
```

```
                }
        }
}



void alarm_disarm() {
        alarm_armed = 0;
        PORTC |= (1<<LED2);
        alarm_fired = 0;
        adc_input2 = 0;
}

void ID_check(uint8_t card[]) {
        for(int i = 0; i<4; i++) {
                if(!(rx_data[i] == card[i])) {
                        break;
                } else if(i == 3) {
                        if(alarm_armed == 1) {
                                alarm_disarm();
                        } else {
                                alarm_arm();
                        }
                }
        }


}

ISR(USART_RXC_vect) {
        rx_data[rx_cnt] = UDR;
        rx_cnt++;
        if(rx_cnt==4) {
                rx_cnt=0;
                ID_check(ANTONS_KORT);
                ID_check(JOCKES_KORT);
                rx_data[0] = 0;
                rx_data[1] = 0;
                rx_data[2] = 0;
                rx_data[3] = 0;
        }

}

ISR(ADC_vect) {
        adc_res = ADC;
```

```
        if(alarm_armed == 1) {
                movement(adc_res);
        }

        adc_start();
}
```