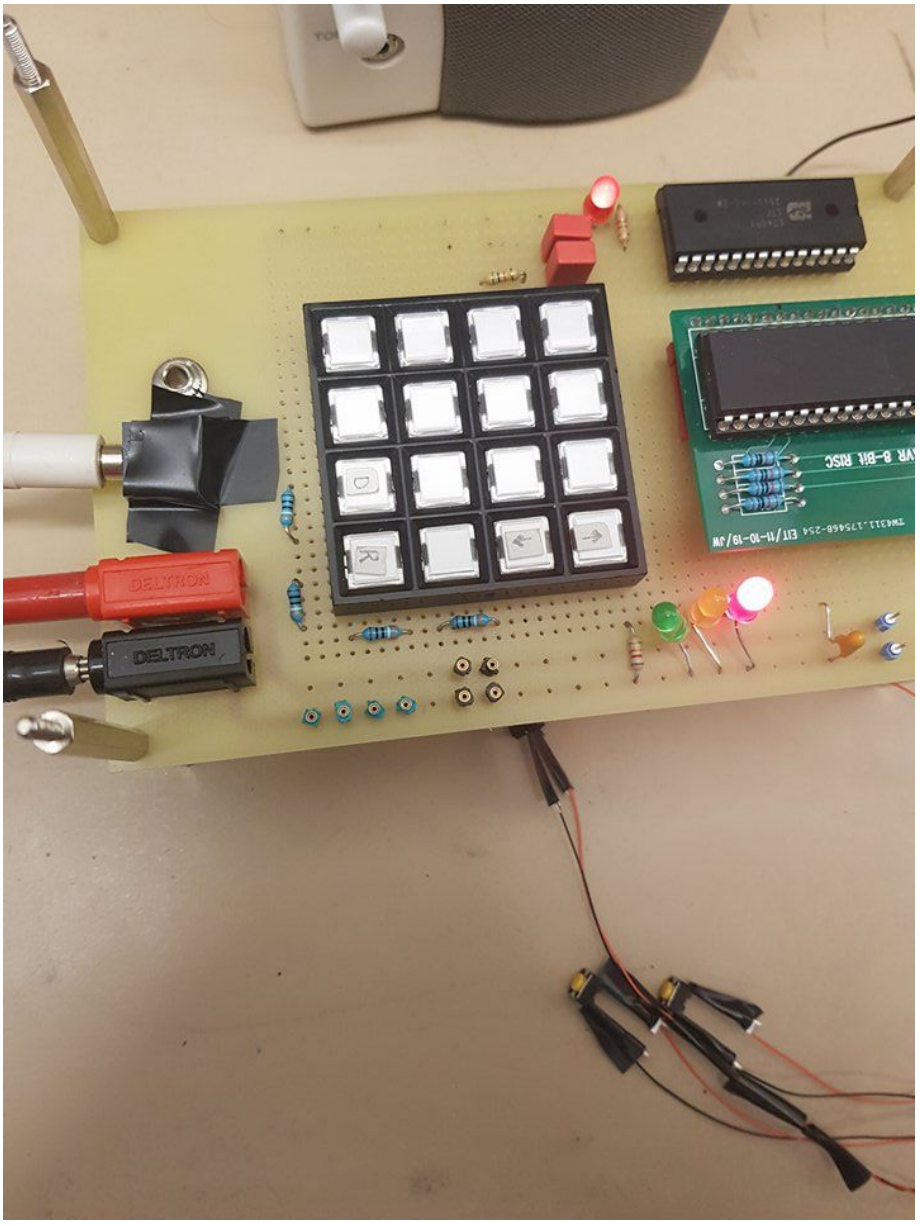
Projektrapport - Nallebjörn

Elektro- och informationsteknik, Lunds Tekniska Högskola
Projektarbete för kursen EITF11, Digitala projekt
Arbete av Johanna Wendesten,Moa Mahlberg & Herbertsson, Erik
2018-05-21



Abstract

This report presents a project creating a prototype for the sound system of a teddy bear. The purpose of the project was to simulate development work in an industrial environment. The teddy bear was required to be able to both record and play sounds stored in its memory. The functions of the teddy bear will be controlled by external stimuli, i.e. buttons.

Development of the prototype was made in three main phases: planning, connection of hardware, and coding with debugging. In the end we were able to create a working prototype of a teddy bear.

Innehållsförteckning

Inledning

Det finns en mängd produkter som innehåller chip och programmering på marknaden. En av de mest förbisedda är förmodligen nallebjörnen och liknande gosedjur för barn. Om man till exempel trycker på en nallebjörns tass kan den säga “mamma” eller liknande.

I anslutning till kursen EITF11 Digitala Projekt har vi utvecklat ett digitalt system som skulle fungera likt chippet i ett gosedjur. Syftet med projektet är att illustrera industriellt utvecklingsarbete.

Kursens mål är:

- Studenten ska analysera och beskriva system av låg och medelhög komplexitet
- Studenten ska testa och felsöka en konstruktion på ett systematiskt sätt
- Studenten ska söka upp och tillgodogöra sig relevant information
- Studenten ska realisera digitala system av låg och medelhög komplexitet
- Studenten ska driva ett projekt framåt till en fungerande prototyp
- Studenten ska kunna formulera sig muntligt och skiftligt
- Studenten ska visa prov på insikt om möjligheter och begränsningar med digitala projekt

Denna rapport kommer att presentera en produktbeskrivning med kravspecifikation, teori, metod, resultat, samt diskussion angående projektet.

Kravspecifikation

I utveckling av nallebjörnen har projektet utgått från följande nyckelfunktion:

Nallebjörnens nyckelfunktion är att spela upp ljudklipp från minnet till följd av yttre stimulans.

Utefter nallebjörnens nyckelfunktion har kravspecifikationen vidareutvecklats. För att kunna lägga in ljudklipp i nallebjörnens minne lades en inspelningsfunktion till. Såväl in- och uppspelningsfunktionen ska kunna användas utan att användaren känner till nallebjörnens källkod.

- Req 1: Nallebjörnen ska kunna spela in ljudklipp från en mikrofon och lagra dem i sitt minne
- Req 2: Nallebjörnen ska kunna spela upp ljudklipp ur minnet
- Req 3: Det ska finnas knappar som kan styra Nallebjörnens olika funktioner
- Req 4: Knapparnas funktion ska vara enkla att förstå

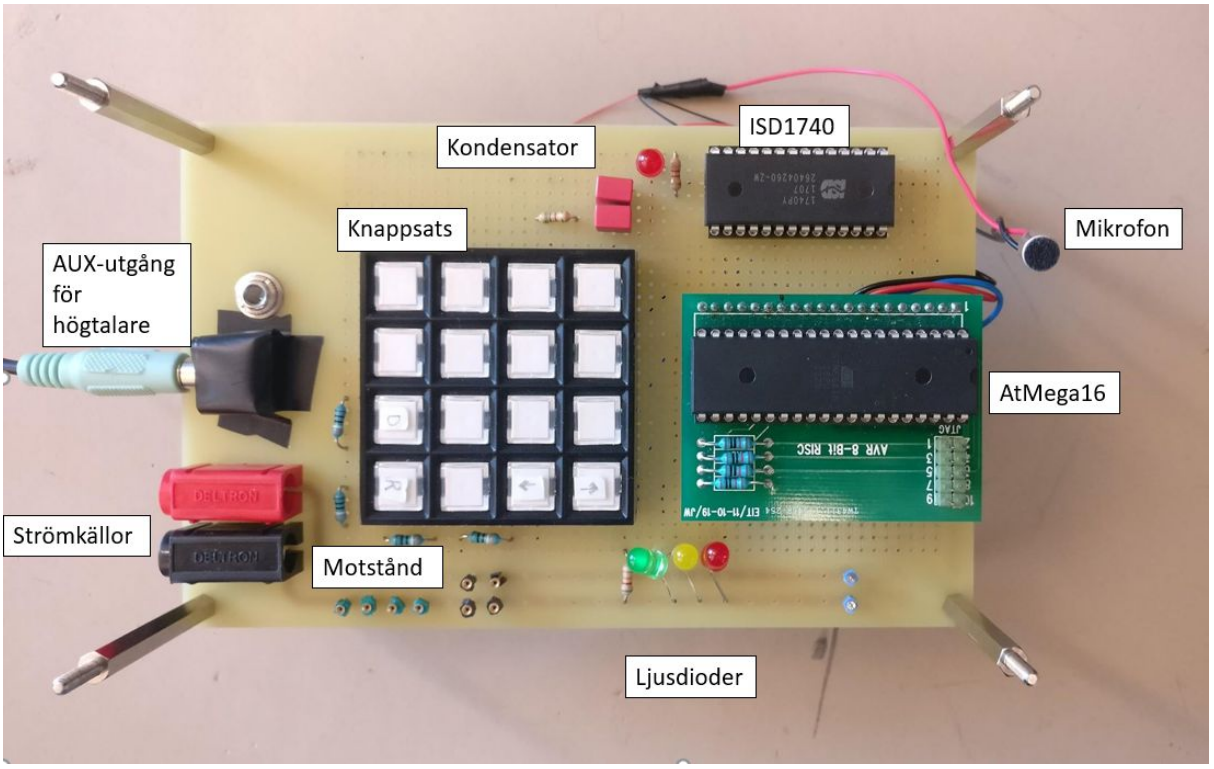
Kvalitetskrav

- QReq 1: Nallebjörnen ska kunna spela upp flera olika ljudklipp

Teori

Hårdvara

Det finns flera större och mindre komponenter i nallebjörnens prototyp. Dessa visas i figur 1. Se bilaga 1 för ett detaljerat kopplingsschema.



Figur 1: Prototyp med namngivna komponenter

Processor

Den processor som har använts är AVR AtMega16. Processorn är 8-bit och har 40 pinnar, varav 32 kan användas som portar. PB4-PB7 är reserverade att användas för Serial Peripheral Interface (SPI), med vilket processorn kommunicerar med ISD1740.

ISD1740

ISD 1740 är en modul som kan spela in och lagra flera olika ljudklipp samt spela upp dem. Modulen är flexibel i vilken ljudkvalitet den spelar in med, vilket styrs av ett motstånd på port 20.

Då ljudklipp spelas in läggs de i en lista i ISD:s cirkulära minne. Minnet har tre interna pointers, en pointer pekar alltid på nästa tomma plats i minnet och används vid inspelning medan två pekar på samma existerande ljudklipp och används vid radering samt uppspelning.

Då SPI-interface används har man ytterligare en pointer som spelar upp på specifika platser. Denna funktion utnyttjas via tre externa knappar.

Knappsats

En knappsats styr vilka funktioner nallebjörnen utför. Utöver dessa tillkommer tre knappar fastsatta i nallebjörnens händer och mage som spelar upp tre olika ljud. Knappsatsens knappar är:

- Power up, sätter på nallebjörnen
- Power down, stänger av nallebjörnen
- Reset, återställer ISD:s pointers
- Rec 1-3, som placerar ljudklipp i olika platser i minnet
- Play, spelar upp det ljudklipp ISD:s pointer pekar på
- Forward, ändrar vilket ljudklipp pointern pekar på
- Erase all, raderar alla ljudklipp ur nallebjörnens minne

Ljusdioder

Tre ljusdioder lyser i olika färger beroende på nedtryckt knapp. De röda ljusdioderna representerar de funktioner som ISD:n använder sig av. De blinkar då ljudklipp spelas upp och lyser då ljudklipp spelas in. Ljusdioderna användes mycket som stöd vid kodning.

Motstånd

Motsånd reglerar strömmen i nallebjörnen. Dessa kopplas till knappsatsen samt olika delar av ISD:n utefter dess design sheet.

JTAG

JTAG kopplar och exekverar kod från en dator till processorn.

Kondensatorer

Kondensatorer kopplades till mikrofonen.

Högtalare

Nallebjörnen kopplas till en högtalare genom en AUX-sladd. Dessa är datorhögtalare som är kopplad till en strömkälla.

Mikrofon

En mikrofon används för att spela in ljud.

Mjukvara

Källkoden är utvecklad i Atmel Studio 7 i programmeringsspråket C. Den är bifogad i bilaga 2.

Genomförande

Projektarbetet bestod av tre faser: planering, koppling av hårdvara samt kodning och felsökning. Eftersom medlemmarna initialt hade väldigt begränsade kunskaper om ellära fick vissa moment göras om flera gånger då misstag hade skett och våra färdigheter utvecklades.

Planering

I planeringsfasen behövde gruppen komma överens om vad som skulle skapas, skapa ett kopplingsschema och diskutera upplägg av arbetet. Kopplingsschemat blev kontrollerat av handledare innan gruppen fick tillåtelse att påbörja nästa fas.

Hårdvarukoppling

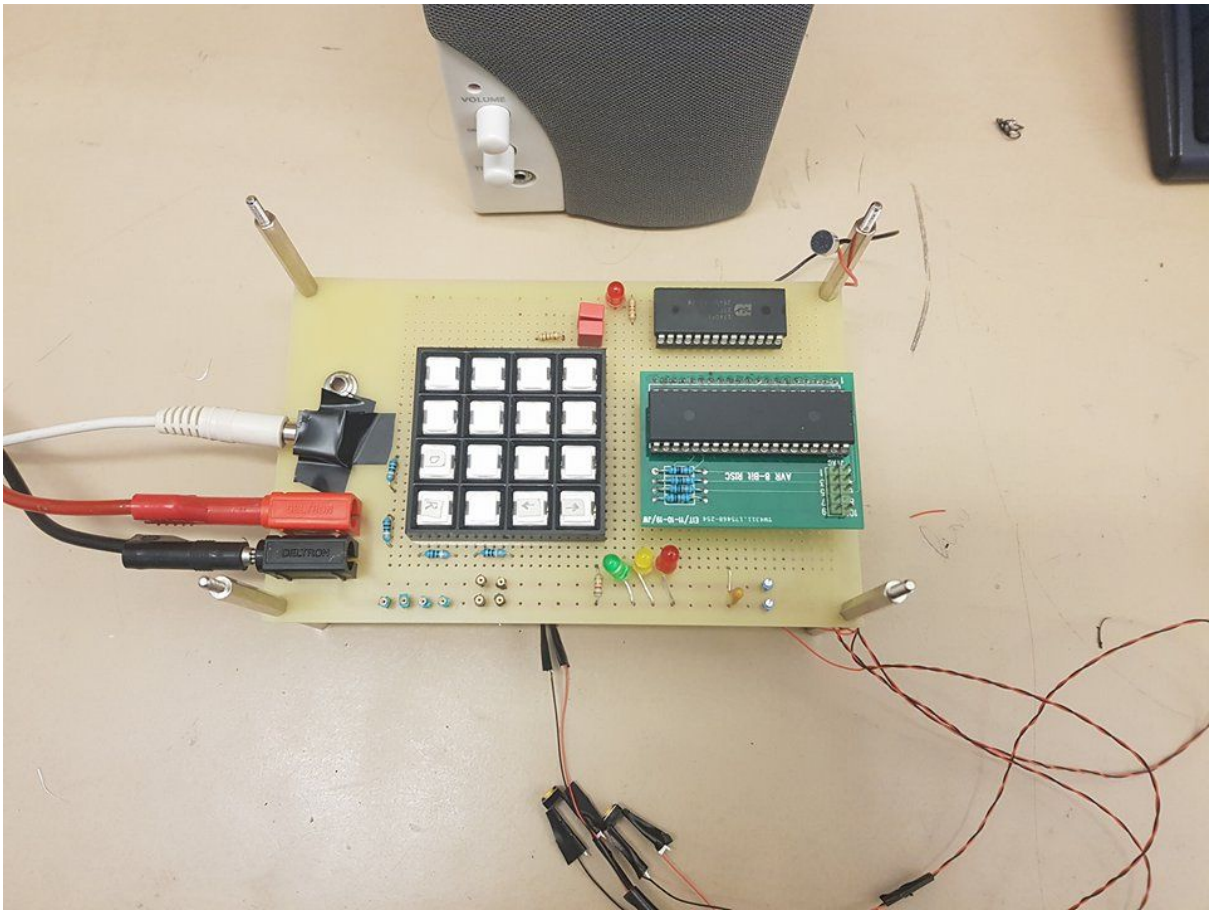
I projektets andra fas gavs tillgång till en verktygslåda att koppla ihop nallebjörnens komponenter med. Komponenterna löddes ihop på en platta och kontakter kopplades samman med koppartråd.

Kodning och felsökning

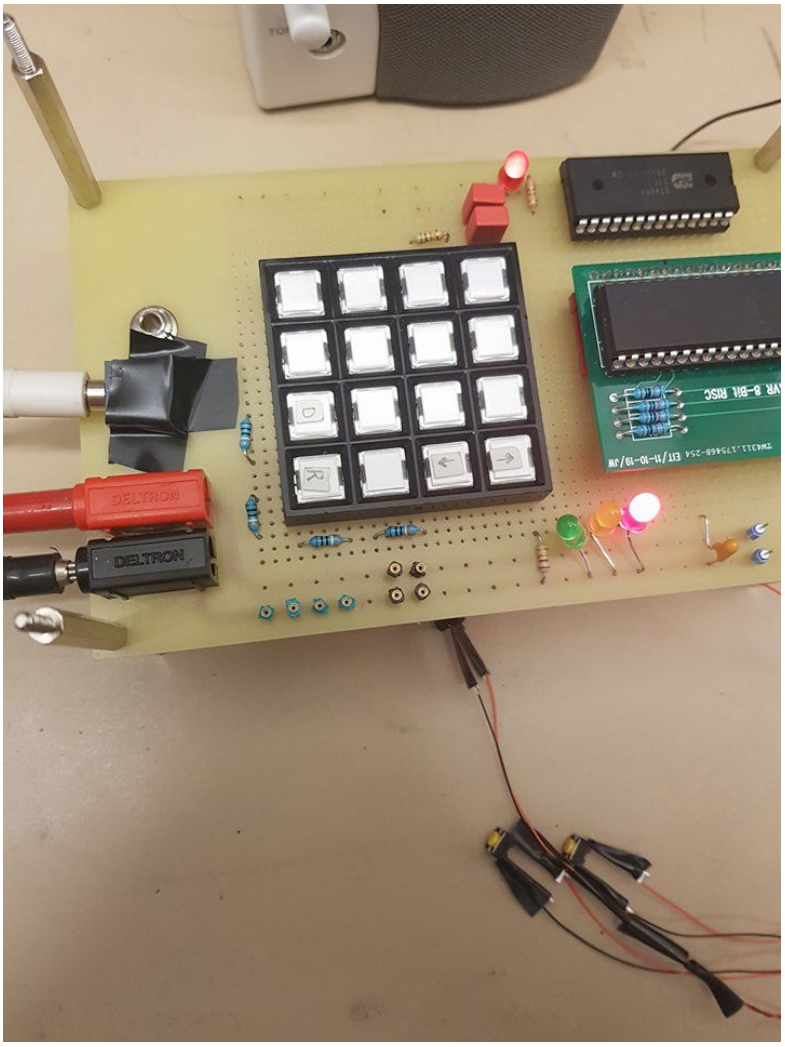
Då nallebjörnen hade kopplats skulle kodning ske. Denna skedde i Atmel Studio 7 i programmeringsspråket C. En stor del av kodningsarbetet bestod av felsökning och med stor hjälp av debuggern. För att säkra att olika moduler i nallebjörnen fungerade kontrollerades mycket kommunikation även med en oscillator.

Resultat

Resultatet av projektet var en prototyp som uppfyller kravspecifikationen. Den kan spela in och spela upp ljud och har ett minne som räcker för sammanlagt ca 50 sekunder ljud. Dock är inte mikrofonen tillräckligt avancerad för att kunna spela in röster utan ljudmeddelanden blir knackningar. Se figur 2 för den kompletta uppsättningen.



Figur 2: Det färdiga systemet ovanifrån



Figur 3: Vid inspelning lyser två röda lampor

Diskussion och slutsatser

Genom projektet har gruppmedlemmarna gjort något de inte tidigare haft någon erfarenhet av och är nöjda med resultatet och sin personliga utveckling. Det går enkelt att notera att arbetet har uppfyllt kursens mål med god precision. En stor andel tid har i synnerhet lagts på att söka och tillgodogöra sig information, att felsöka konstruktionen samt att realisera och driva projektet framåt.

Att notera från projektet är att utvecklingen av prototypen inte skedde linjärt. Flertalet gånger behövde vi gå tillbaka och koppla om vår konstruktion efter att kodningen inte fungerade, vilket resulterade i att vi behövde göra ett nytt kopplingsschema och således var tillbaka i första fasen. Med det lyckade slutgiltiga resultatet i hand är vi därför stolta och kan ta med oss som lärdom att den viktigaste utvecklingen inte skedde i konstruktionen utan snarare i våra egna kunskaper.

Referenslista

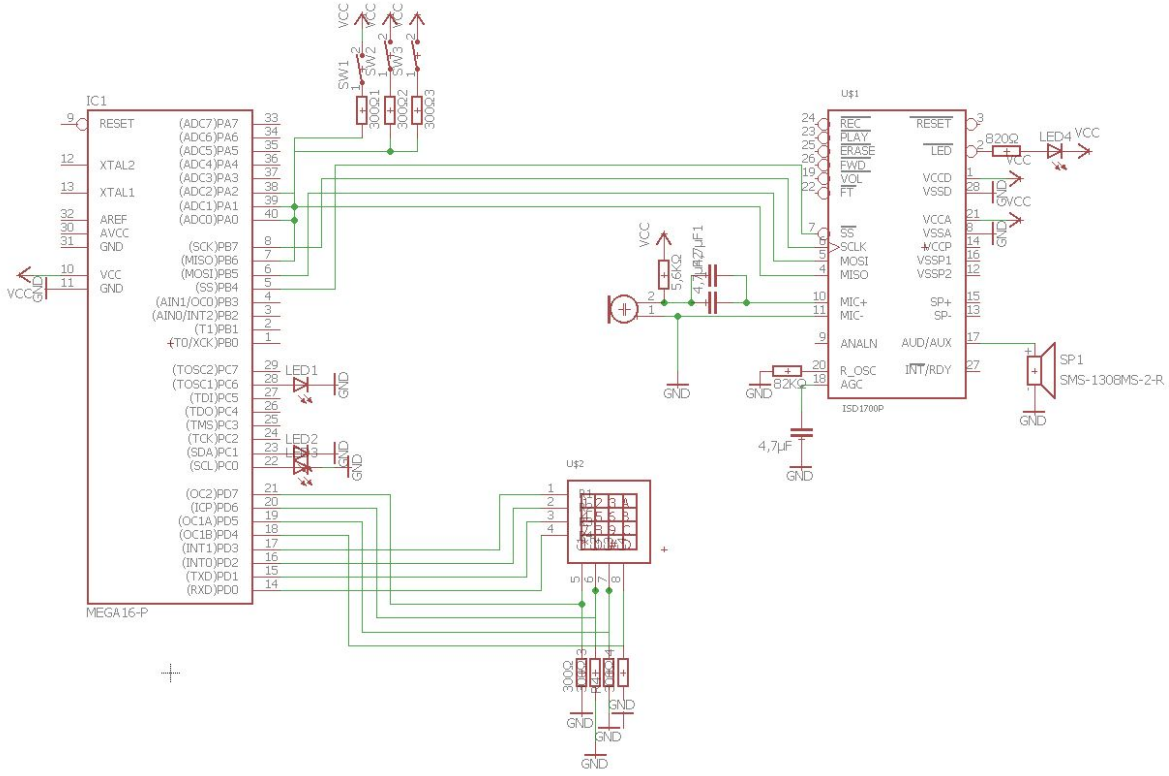
Datablad för AVR AtMega16
https://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf
Hämtad 2018-05-13

Datablad för ISD17xx
http://www.microtechnica.tv/support/manual/ISD1700_Design_Guide.pdf Hämtad 2018-05-13

Kurshemsida EITF11 - Digitala projekt https://www.eit.lth.se/kurs/eitf11 Hämtad 2018-05-16

The SPI of the AVR http://maxembedded.com/2013/11/the-spi-of-the-avr/ Hämtad 2018-05-17

Bilaga 1. Kopplingsschema



Bilaga 2. Källkod

```
/*
 * SPI_test.c
 *
 * Created: 2018-05-07 16:10:36
 * Author : Moa Mahlberg, Erik Herbertsson, Johanna Wendesten
 */

#define F_CPU                8000000UL

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

//SPI
#define INT         PB3
#define SS          PB4
#define MOSI PB5
#define MISO PB6
#define SCK         PB7

//LED
#define red         PC6
#define yellow      PC1
#define green       PC0

#define PU          0x01
#define STOP        0x02
#define RESET       0x03
#define CLR_INT             0x04
#define RD_STATUS   0x05
#define RD_PLAY_PTR 0x06
#define PD          0x07
#define RD_REC_PTR  0x08
#define DEVID       0x09
#define PLAY        0x40
#define REC         0x41
#define ERASE       0x42
#define G_ERASE             0x43
#define RD_APC  0x44
#define WR_APC1             0x45
#define WR_APC2             0x65
#define WR_NVCFG            0x46
#define LD_NVCFG            0x47
#define FWD         0x48
#define CHK_MEM             0x49
#define EXTCLK              0x4A
#define SET_PLAY            0x49
#define SET_REC             0x81
#define SET_ERASE   0x82
```

```
void init_spi();
void send_byte_spi(uint8_t byte);
void send_two_bytes(uint8_t byte0, uint8_t byte1);
void runButtons();

void isd_clear_int();
void isd_rec();
void isd_reset();
void isd_power_up();
void isd_read_status() ;
void isd_reset();
void isd_play();
void isd_power_down();
void isd_FWD();
void isd_G_erase();
void isd_set_play1();
void isd_set_play2();
void isd_set_play3();
void isd_set_rec1();
void isd_set_rec2();
void isd_set_rec3();

volatile uint16_t spi_two_bytes;
volatile uint8_t dummy;
volatile uint8_t scrap;
volatile uint8_t first;
volatile uint8_t second;
volatile uint8_t third;
volatile uint8_t fourth;
volatile uint8_t crnt_byte;
volatile uint8_t spi_count;
volatile uint16_t spi_two_bytes1;
volatile uint16_t led_byte;

uint8_t nbr_of_bytes;


int main(void)
{
	//Initiate SPI-function and start interruptions
	init_spi();

	//Set up buttons and LED
	DDRD = 0b00001111;
	DDRC = (1<<red)|(1<<yellow)|(1<<green);
	isd_power_up();
	isd_set_APC();

	while (1) {

		runButtons();

	}
}
```

```
void isd_power_up(){
        nbr_of_bytes = 3;
        send_two_bytes(PU, 0x00);

        dummy++;

}

void isd_RD_APC(){
        nbr_of_bytes = 4;
        send_four_bytes(RD_APC, 0x00, 0x00, 0x00);

        dummy++;

}
void isd_G_erase(){
        nbr_of_bytes = 3;
        send_two_bytes(G_ERASE, 0x00);

        dummy++;

}

void isd_FWD(){
        send_two_bytes(FWD, 0x00);
}

void isd_set_APC(){
        nbr_of_bytes = 4;
        send_three_bytes(WR_APC1, 0b11000000, 0b00000100);

        dummy++;

}

void isd_clear_int(){
        nbr_of_bytes = 3;
        send_two_bytes(CLR_INT, 0x00);

        dummy++;

}

void isd_read_status() {

        nbr_of_bytes = 3;
        send_three_bytes(RD_STATUS, 0x00, 0x00);

}

void isd_reset(){

        nbr_of_bytes = 3;
```

```
        send_two_bytes(RESET, 0x00);

        dummy++;

}

void isd_stop(){

        nbr_of_bytes = 3;
        send_two_bytes(STOP, 0x00);

        dummy++;

}

void isd_play() {

        nbr_of_bytes = 3;
        led_byte= PLAY + 0x10;
        send_two_bytes(led_byte, 0x00);

}

void isd_set_play1() {

        nbr_of_bytes = 5;
        led_byte= SET_PLAY + 0x10;
        send_five_bytes(led_byte, 0x01, 0x00, 0x00, 0x00);

}
void isd_set_play2() {

        nbr_of_bytes = 5;
        led_byte= SET_PLAY + 0x10;
        send_five_bytes(led_byte, 0x02, 0x00, 0x00, 0x00);

}

void isd_set_play3() {

        nbr_of_bytes = 5;
        led_byte= SET_PLAY + 0x10;
        send_five_bytes(led_byte, 0x03, 0x00, 0x00, 0x00);

}

void isd_set_rec1() {

        nbr_of_bytes = 5;
        led_byte= SET_REC + 0x10;
        send_five_bytes(led_byte, 0x01, 0x00, 0x00, 0x00);

}
```

```
void isd_set_rec2() {

        nbr_of_bytes = 5;
        led_byte= SET_REC + 0x10;
        send_five_bytes(led_byte, 0x02, 0x00, 0x00, 0x00);

}

void isd_set_rec3() {

        nbr_of_bytes = 5;
        led_byte= SET_REC + 0x10;
        send_five_bytes(led_byte, 0x03, 0x00, 0x00, 0x00);

}

void isd_rec() {

        nbr_of_bytes = 3;
        led_byte= REC + 0x10;
        send_two_bytes(led_byte, 0x00);

}
void isd_power_down() {

        nbr_of_bytes = 2;
        send_two_bytes(PD, 0x00);

}
void init_spi() {

        DDRB = (1<<SCK)|(1<<MOSI)|(1<<SS);
        PORTB= (1<<SCK)|(1<<SS);

        // Enable SPI, Set as Master
        // Prescaler: Fosc/16
        // |(1<<CPOL)|(1<<CPHA) Borttaget då vi tror på noll!
        //|(1<<CPOL)|(1<<CPHA);
        SPCR = (1<<SPE)|(1<<MSTR)|(1<<DORD)|(1<<SPR0)|(1<<SPIE)|(1<<CPOL)|(1<<CPHA);
        //Klockan är rätt då vi prövat allt annat!



}

void send_byte_spi(uint8_t byte) {


        //PORTB &= ~(1<<SS);
        // Load data into the buffer
        SPDR = byte;

        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
```

```
        //PORTB |= (1<<SS);
        first=SPDR;
}

void send_two_bytes(uint8_t byte0, uint8_t byte1) {

        PORTB &= ~(1<<SS);
        //Load data into the buffer
        SPDR = byte0;


        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        first=SPDR;

        SPDR = byte1;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        second=SPDR;
        PORTB |= (1<<SS);

}

void send_three_bytes(uint8_t byte0, uint8_t byte1, uint8_t byte2) {

        PORTB &= ~(1<<SS);
        //Load data into the buffer
        SPDR = byte0;

        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        first=SPDR;

        SPDR = byte1;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        second=SPDR;

        SPDR = byte2;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        third=SPDR;
        PORTB |= (1<<SS);
}

void send_four_bytes(uint8_t byte0, uint8_t byte1, uint8_t byte2, uint8_t byte3) {

        PORTB &= ~(1<<SS);
        //Load data into the buffer
        SPDR = byte0;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        first=SPDR;
```

```
        SPDR = byte1;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        second=SPDR;

        SPDR = byte2;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        third=SPDR;

        SPDR = byte3;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        fourth=SPDR;

        PORTB |= (1<<SS);
}

void send_five_bytes(uint8_t byte0, uint8_t byte1, uint8_t byte2, uint8_t byte3, uint8_t byte4) {

        PORTB &= ~(1<<SS);
        //Load data into the buffer
        SPDR = byte0;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        first=SPDR;

        SPDR = byte1;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        second=SPDR;

        SPDR = byte2;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        third=SPDR;

        SPDR = byte3;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        fourth=SPDR;

        SPDR = byte4;
        //Wait until transmission complete
        while(!(SPSR & (1<<SPIF) ));
        scrap=SPDR;

        PORTB |= (1<<SS);
}



void runButtons(){
```

```
PORTD=0b00000001;
if (PIND==0b00100001) // THE START BUTTON (1,1)
{

        //PU
        isd_power_up();
        PORTC = (1<<red);
        _delay_ms(1000);
        PORTC = (1<<yellow);
        _delay_ms(1000);
        PORTC = (1<<green);
        _delay_ms(1000);
        PORTC=0x00;

        //reset when starting
        isd_reset();
        isd_set_APC();

}
if (PIND==0b00010001) // RESET (2,1)
{
        isd_reset();
        PORTC = (1<<red);
        _delay_ms(100);
        PORTC=0x00;
        _delay_ms(100);
        PORTC = (1<<red);
        _delay_ms(100);
        PORTC=0x00;
        _delay_ms(100);
        PORTC = (1<<red);
        PORTC=0x00;
        isd_set_APC();
}
if (PIND==0b10000001) // POWER DOWN (3,1)
{
        //PD
        isd_power_down();

        PORTC = (1<<red);
        _delay_ms(800);
        PORTC=0x00;
}
if (PIND==0b01000001) // STOP (4,1)
{
        //Stop
        isd_stop();

        PORTC = (1<<yellow);
        _delay_ms(100);
        PORTC=0x00;

}
PORTD=0b00000010;
```

```
if (PIND==0b00100010) // RECORD (1,2)
{
        PORTC = (1<<red);

        isd_rec();
        _delay_ms(10000);
        // Stop recording
        isd_stop();
        PORTC=0x00;
}
if (PIND==0b00010010) // PLAY (2,2)
{


        isd_play();

        PORTC = (1<<green);
        _delay_ms(100);
        PORTC=0x00;
}

if (PIND==0b10000010) // Rec 1 (3,2)
{
        isd_set_rec1();

        PORTC=0b010000011;
        _delay_ms(5000);
        PORTC=0x00;
}

if (PIND==0b01000010) // Rec 2 (4,2)
{
        isd_set_rec2();
        PORTC=0b0100011;
        _delay_ms(100);
        PORTC=0x00;
}

PORTD=0b00000100;
if (PIND==0b00100100) // Rec 3 (1,3)
{
        isd_set_rec3;

}

if (PIND==0b00010100) //(2,3)
{
        PORTC = (1<<red);

        isd_rec();
        _delay_ms(10000);
        // Stop recording
        isd_stop();
        PORTC=0x00;
```

```
}
if (PIND==0b10000100) //(3,3)
{

}
if (PIND==0b01000100) //(4,3)
{

}
PORTD=0b00001000;
if (PIND==0b00101000) //(1,4)
{

}
if (PIND==0b00011000) //(2,4)
{

}
if (PIND==0b10001000) //(3,4)
{
        PORTC = (1<<green) | (1<<yellow);
        isd_FWD();
        _delay_ms(500);
        PORTC=0x00;

}

if (PIND==0b01001000) //(4,4)
{
        PORTC= (1<<red)|(1<<yellow)|(1<<green);
        _delay_ms(500);
        isd_G_erase();
        PORTC=0x00;

}

if(PINA = 0b00000001) // Nalle spela 1
{
        PORTC = (1<<green);
        isd_set_play1();

}

if(PINA = 0b00000010) // Nalle spela 2
{
        PORTC = (1<<yellow);
        isd_set_play2();

}

if(PINA = 0b00000100) // Nalle spela 3
{
        PORTC = (1<<red);
```

```
                isd_set_play3();
            }

    }
```