```c
//glob_header.h

#ifndef GLOB_HEADER_H_
#define GLOB_HEADER_H_

#define F_CPU            8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#endif
```

```
//main.c

/*          Knappdeklaration:
            8 är sequencer
            9 är set

            12 är record
            13 är play
            14 är erase
            15 är stop

            0-7 är ljud.
            0-3 är korta ljud och 4-7 är långa ljud
*/

#include "glob_header.h"
#include "key_encoder.h"
#include "audioChip.h"
#include "spi.h"
#include "sequencer.h"
#include "timer0.h"

volatile uint8_t keypad_status;
volatile uint8_t cnt;
volatile uint32_t time_cnt;

uint8_t repsonseRecord;

uint8_t lastResponse1;
uint8_t lastResponse2;
uint8_t lastResponse3;
uint8_t lastResponse4;
uint8_t lastResponse5;
uint8_t lastResponse6;
uint8_t lastResponse7;



int main(void)
{
            DDRA |= (1 << PA0); //initierar led-output
    keypad_init();
            keypad_interrupt();
            spi_init_master();
            sei();
            powerUp();
            audioChip_init();
            checkRdStatus();
            sequencer_init();
            timer0_init();
            timer0_start();

    while (1) {

                        if(keypad_status == 8){
                                    PORTA |= (1 << PA0);
                                    _delay_ms(50);
                                    PORTA &= ~(1 << PA0);
                                    sequencerManager();

                        }else if(keypad_status == 12 ) {//12 aka punkt är record
                                    PORTA |= (1 << PA0);
                                    recordManager();
                                    PORTA &= ~(1 << PA0);

                        }else if(keypad_status == 13){ //13 aka tom 1 är play
                                    PORTA |= (1 << PA0);
                                    _delay_ms(100);
                                    PORTA &= ~(1 << PA0);
                                    _delay_ms(100);
                                    PORTA |= (1 << PA0);
                                    _delay_ms(100);
                                    PORTA &= ~(1 << PA0);


                                    playManager();

                                    PORTA |= (1 << PA0);
                                    _delay_ms(100);
                                    PORTA &= ~(1 << PA0);
                                    _delay_ms(100);
                                    PORTA |= (1 << PA0);
                                    _delay_ms(100);
                                    PORTA &= ~(1 << PA0);


                        }else if(keypad_status == 14){
                                    PORTA |= (1 << PA0);
                                    _delay_ms(300);
                                    PORTA &= ~(1 << PA0);
                                    _delay_ms(300);
                                    PORTA |= (1 << PA0);



                                    eraseManager();


                                    PORTA &= ~(1 << PA0);
                                    _delay_ms(300);
                                    PORTA |= (1 << PA0);
                                    _delay_ms(300);
                                    PORTA &= ~(1 << PA0);


                        }else{
                                    PORTA &= ~(1 << PA0);
                        }

    }
}


ISR(INT0_vect){
            keypad_status = keypad_read();
```

```
             cnt++;
}

ISR(TIMER0_OVF_vect) {

             time_cnt++;

}
```

```
//keyEncoder.h

#ifndef KEY_ENCODER_H_
#define KEY_ENCODER_H_

#include "glob_header.h"

#define DATA_A          PB0
#define DATA_B          PB1
#define DATA_C          PB2
#define DATA_D          PB3

void keypad_init();
uint8_t keypad_read();
void keypad_interrupt();


#endif
```

```c
//keyEncoder.c

#include "key_encoder.h"

void keypad_init() {

            DDRB &= ~((1 << DATA_D) | (1 << DATA_C) | (1 << DATA_B) | (1 << DATA_A));

}

uint8_t keypad_read() {

            return PINB & 0b00001111;

}

void keypad_interrupt() {

            MCUCR |= (1 << ISC01) | (1 << ISC00);
            GICR |= (1 << INT0);

}
```

```c
//spi.h

#ifndef SPI_H_
#define SPI_H_

#include "glob_header.h"

#define ACK 0x7E

#define SS PB4
#define MOSI PB5
#define MISO PB6
#define SCK PB7

void spi_init_master(void);
uint8_t sendTwoBytes(uint8_t byte1, uint8_t byte2);
uint8_t sendThreeBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3);
uint8_t sendFourBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3, uint8_t byte4);
uint8_t sendSevenBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3, uint8_t byte4,
uint8_t byte5, uint8_t byte6, uint8_t byte7);

#endif
```

```c
//spi.c

#include "spi.h"

extern uint8_t lastResponse1;
extern uint8_t lastResponse2;
extern uint8_t lastResponse3;
extern uint8_t lastResponse4;
extern uint8_t lastResponse5;
extern uint8_t lastResponse6;
extern uint8_t lastResponse7;

void spi_init_master(void){

            DDRB = (1<<MOSI)|(1<<SCK)|(1<<SS);
            SPCR = (1<<SPE)|(1<<MSTR)|(1<<DORD)|(1<<SPR0)|(1<<CPOL)|(1<<CPHA);

            PORTB = (1<<SS) | (1<< SCK);
}


uint8_t sendTwoBytes(uint8_t byte1, uint8_t byte2){ //hantera svar

            PORTB &= ~(1<<SS);

            SPDR = byte1;
            while(!(SPSR & (1<<SPIF)));
            lastResponse1 = SPDR;

            SPDR = byte2;
            while(!(SPSR & (1<<SPIF)));
            lastResponse2 = SPDR;

            PORTB |= (1<<SS);
            return lastResponse1;
}

uint8_t sendThreeBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3){ //hantera svar
            PORTB &= ~(1<<SS);

            SPDR = byte1;
            while(!(SPSR & (1<<SPIF)));
            lastResponse1 = SPDR;

            SPDR = byte2;
            while(!(SPSR & (1<<SPIF)));
            lastResponse2 = SPDR;

            SPDR = byte3;
            while(!(SPSR & (1<<SPIF)));
            lastResponse3 = SPDR;

            PORTB |= (1<<SS);
            return lastResponse1;
}

uint8_t sendFourBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3, uint8_t byte4){
//hantera svar
            PORTB &= ~(1<<SS);

            SPDR = byte1;
            while(!(SPSR & (1<<SPIF)));
```

```
		lastResponse1 = SPDR;

		SPDR = byte2;
		while(!(SPSR & (1<<SPIF)));
		lastResponse2 = SPDR;

		SPDR = byte3;
		while(!(SPSR & (1<<SPIF)));
		lastResponse3 = SPDR;

		SPDR = byte4;
		while(!(SPSR & (1<<SPIF)));
		lastResponse4 = SPDR;

		PORTB |= (1<<SS);
		return lastResponse1;
}


uint8_t sendSevenBytes(uint8_t byte1, uint8_t byte2, uint8_t byte3, uint8_t byte4, 
uint8_t byte5, uint8_t byte6, uint8_t byte7){

		PORTB &= ~(1<<SS);

		SPDR = byte1;
		while(!(SPSR & (1<<SPIF)));
		lastResponse1 = SPDR;

		SPDR = byte2;
		while(!(SPSR & (1<<SPIF)));
		lastResponse2 = SPDR;

		SPDR = byte3;
		while(!(SPSR & (1<<SPIF)));
		lastResponse3 = SPDR;

		SPDR = byte4;
		while(!(SPSR & (1<<SPIF)));
		lastResponse4 = SPDR;

		SPDR = byte5;
		while(!(SPSR & (1<<SPIF)));
		lastResponse5 = SPDR;

		SPDR = byte6;
		while(!(SPSR & (1<<SPIF)));
		lastResponse6 = SPDR;

		SPDR = byte7;
		while(!(SPSR & (1<<SPIF)));
		lastResponse7 = SPDR;

		PORTB |= (1<<SS);
		return lastResponse1;
}
```

```c
//audioChip.h

#ifndef ADUIO_CHIP_H_
#define ADUIO_CHIP_H_

#include "glob_header.h"
#include "spi.h"

#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         0x80
#define SET_REC                      0x81
#define SET_ERASE        0x82

#define SET_LED                      4

uint8_t audioChip_init();
uint8_t powerUp();
uint8_t recordManager();
uint8_t recordToPosition(int position);
uint8_t playManager();
void playPosition(int position);
void checkRdStatus();
void clear_interrupt();
void calcMemoryAdresses(int position);
void eraseAllSounds();
void eraseManager();
void erasePosition(int position);

#endif
```

```
//audioChip.c

#include "audioChip.h"

extern volatile uint8_t keypad_status;
extern uint8_t repsonseRecord;

extern uint8_t lastResponse1;
extern uint8_t lastResponse2;
extern uint8_t lastResponse3;
extern uint8_t lastResponse4;
extern uint8_t lastResponse5;
extern uint8_t lastResponse6;
extern uint8_t lastResponse7;

uint8_t startAdress1;
uint8_t startAdress2;
uint8_t endAdress1;
uint8_t endAdress2;

uint8_t audioChip_init(){
            return sendThreeBytes(WR_APC2, 0b10101000, 0b00000001);

}

 uint8_t powerUp(){
            uint8_t response;
            do{
            response = sendTwoBytes(PU, 0x00);
            checkRdStatus();
            }while(lastResponse1 % 2 ==1);
            _delay_ms(50);
            clear_interrupt();
            return (response & 0b00000100) >> 2;
 }

 uint8_t recordManager(){
             int lastKeyPress = keypad_status;
             while(lastKeyPress == keypad_status); //väntar på nytt värde i keypad för
att välja recording position
             while(keypad_status != 15){

                          if(keypad_status < 8){
                                      recordToPosition(keypad_status);

                          }
             }
            keypad_status = 100;
            return 1;
 }

 uint8_t recordToPosition(int position){

             calcMemoryAdresses(position);

             clear_interrupt();
             checkRdStatus();
             if((lastResponse3 & 0b00000001) == 1){

                         uint8_t setLedErase = SET_ERASE;
                         setLedErase |= (1<<SET_LED);
                         sendSevenBytes(setLedErase, 0x00, startAdress1, startAdress2, 
endAdress1, endAdress2, 0x00);


                         _delay_ms(100); //väntar för att kunna ta emot nästa kommando
                         clear_interrupt();

                         uint8_t ledSetRec = SET_REC;
                         ledSetRec |= (1<<SET_LED);
                         sendSevenBytes(ledSetRec, 0x00, startAdress1, startAdress2, 
endAdress1, endAdress2, 0x00);

                         if((lastResponse1 & 0b00000001) == 1){
                                       PORTA &= ~(1 << PA0);
                                       _delay_ms(500);
                                       PORTA |= (1 << PA0);

                          }
              }
             while(keypad_status != 15);
             sendTwoBytes(STOP, 0x00);

              keypad_status = 100;

}

uint8_t playManager(){
             int lastKeyPress = keypad_status;
             while(lastKeyPress == keypad_status);

             while(keypad_status != 15){

                         if(keypad_status < 8){
                         playPosition(keypad_status);
                         keypad_status = 100;

                         }
             }

             sendTwoBytes(STOP, 0x00);
             keypad_status = 100;
             return 1;
}

void playPosition(int position){
             calcMemoryAdresses(position);
             clear_interrupt();
             sendTwoBytes(STOP, 0x00);

             checkRdStatus();
             _delay_ms(30);

             if((lastResponse3 & 0b00000001) == 1){

                         uint8_t ledSetPlay = SET_PLAY;
                         ledSetPlay |= (1<<SET_LED);
                         sendSevenBytes(ledSetPlay, 0x00, startAdress1, startAdress2, 
endAdress1, endAdress2, 0x00);

                         if((lastResponse1 & 0b00000001) == 1){
                                      PORTA |= (1 << PA0);
                                      _delay_ms(500);
```

```
            PORTA &= ~(1 << PA0);

        }
    }

    return;
}

void eraseManager(){
    keypad_status = 100;

    while(keypad_status != 15){

        if(keypad_status < 8){//knappar 0-8 är ljud atm
            erasePosition(keypad_status);
        }else if(keypad_status ==14){
            eraseAllSounds();
            return;
        }
    }

    sendTwoBytes(STOP, 0x00);
    keypad_status = 100;
    return;
}

void erasePosition(int position){
    PORTA &= ~(1 << PA0);
    _delay_ms(100);
    PORTA |= (1 << PA0);
    _delay_ms(100);
    PORTA &= ~(1 << PA0);
    _delay_ms(100);
    PORTA |= (1 << PA0);

    calcMemoryAdresses(position);
    clear_interrupt();
    checkRdStatus();
    if((lastResponse3 & 0b00000001) == 1){
        uint8_t ledSetErase = SET_ERASE;
        ledSetErase |= (1<<SET_LED);
        sendSevenBytes(ledSetErase, 0x00, startAdress1, startAdress2,
endAdress1, endAdress2, 0x00);

        if((lastResponse1 & 0b00000001) == 1){

            PORTA |= (1 << PA0);
            _delay_ms(500);
            PORTA &= ~(1 << PA0);

        }
    }

    keypad_status = 100;
    return;
}

void eraseAllSounds(){
    PORTA &= ~(1 << PA0);
    _delay_ms(100);
    PORTA |= (1 << PA0);
    _delay_ms(100);
```

```
        PORTA &= ~(1 << PA0);
        _delay_ms(100);
        PORTA |= (1 << PA0);
        _delay_ms(100);
        PORTA &= ~(1 << PA0);
        _delay_ms(100);
        PORTA |= (1 << PA0);
        _delay_ms(100);
        PORTA &= ~(1 << PA0);
        _delay_ms(100);
        PORTA |= (1 << PA0);


        keypad_status = 100;
        while(keypad_status != 15){
                if(keypad_status == 14){
                        sendTwoBytes(G_ERASE, 0x00);

                        PORTA &= ~(1 << PA0);
                        _delay_ms(100);
                        PORTA |= (1 << PA0);
                        _delay_ms(100);
                        PORTA &= ~(1 << PA0);
                        _delay_ms(100);
                        PORTA |= (1 << PA0);
                        _delay_ms(100);
                        PORTA &= ~(1 << PA0);
                        _delay_ms(100);
                        PORTA |= (1 << PA0);
                        _delay_ms(100);
                        PORTA &= ~(1 << PA0);
                        _delay_ms(100);
                        PORTA |= (1 << PA0);
                        _delay_ms(100);
                        PORTA &= ~(1 << PA0);
                        _delay_ms(100);
                        PORTA |= (1 << PA0);

                        keypad_status = 100;
                        return;
                }
        }
        PORTA &= ~(1 << PA0);
        keypad_status = 100;
        return;
}

void checkRdStatus(){
        sendThreeBytes(RD_STATUS, 0x00,0x00);
}

void clear_interrupt(){
        sendTwoBytes(CLR_INT, 0x00);

}

void calcMemoryAdresses(int position){ //4 första korta, 4 sista långa
switch(position){
        case 0:
                startAdress1 = 17;
                startAdress2 = 0x00;
                endAdress1 = 32;
```

```
                        endAdress2 = 0x00;
                        break;
            case 1:
                        startAdress1 = 33;
                        startAdress2 = 0x00;
                        endAdress1 = 48;
                        endAdress2 = 0x00;
                        break;
            case 2:
                        startAdress1 = 49;
                        startAdress2 = 0x00;
                        endAdress1 = 64;
                        endAdress2 = 0x00;
                        break;
            case 3:
                        startAdress1 = 65;
                        startAdress2 = 0x00;
                        endAdress1 = 80;
                        endAdress2 = 0x00;
                        break;
            case 4:
                        startAdress1 = 81;
                        startAdress2 = 0x00;
                        endAdress1 = 145;
                        endAdress2 = 0x00;
                        break;
            case 5:
                        startAdress1 = 146;
                        startAdress2 = 0x00;
                        endAdress1 = 210;
                        endAdress2 = 0x00;
                        break;
            case 6:
                        startAdress1 = 211;
                        startAdress2 = 0;
                        endAdress1 = 19;
                        endAdress2 = 1;
                        break;
            case 7:
                        startAdress1 = 20;
                        startAdress2 = 1;
                        endAdress1 = 79;
                        endAdress2 = 1;
                        break;
            }

}
```

```
//sequencer.h

#ifndef SEQUENCER_H_
#define SEQUENCER_H_

#include "glob_header.h"
#include "audioChip.h"

void sequencer_init(void);
void sequencerManager(void);
void setSequence(void);
void setPosition(uint8_t position);
void playSequence();

#endif
```

```c
//sequencer.c

/*
En åttondelsnot är 37 counts lång

*/

#include "sequencer.h"

extern uint8_t startAdress1;
extern uint8_t startAdress2;
extern uint8_t endAdress1;
extern uint8_t endAdress2;
extern volatile uint8_t keypad_status;
extern volatile uint8_t time_cnt;

uint8_t sequence[8];

void sequencer_init(){
        for(int i =0; i<8; i++){
                sequence[i] = 10;
        }
        return;
}

void sequencerManager(){
        keypad_status = 100;
        while(keypad_status != 15){
                if(keypad_status == 13){
                        playSequence();
                }else if(keypad_status == 9){
                        setSequence();
                } else if (keypad_status == 14) {
                        for(int i = 0; i < 8 ; i++) {
                                sequence[i] = 10;
                        }
                }
        }

}

void setSequence(){
        PORTA |= (1 << PA0);
        keypad_status = 100;
        while (keypad_status != 15){
                if (keypad_status <8){
                        setPosition(keypad_status);
                }
        }
        keypad_status = 100;
        PORTA &= ~(1 << PA0);
        return;
}

void setPosition(uint8_t position){
        keypad_status = 100;
        while(keypad_status != 15){
                if (keypad_status <8){
                        sequence[position] = keypad_status;
                        keypad_status = 100;
                        return;
                }
```

```
            }
            keypad_status = 100;
            return;
}
void playSequence(){
            PORTA |= (1 << PA0);
            time_cnt = 0;
            while (keypad_status != 15){
                        for (int i= 0; i<8; i++){
                                    while((time_cnt < 37));
                                                if (sequence[i] < 8){

            playPosition(sequence[i]);
                                                }
                                    time_cnt = 0;
                        }
            }
            sendTwoBytes(STOP, 0x00);
            keypad_status = 100;
            PORTA &= ~(1 << PA0);
            return;
}
```

```c
//timer0.h

#ifndef TIMER0_H_
#define TIMER0_H_

#include "glob_header.h"

void timer0_init(void);
void timer0_start(void);
void timer0_stop(void);

#endif
```

```c
//timer0.c

#include "timer0.h"

void timer0_init(){

            TIMSK |= (1 << TOIE0);

}

void timer0_start() {

            TCCR0 |= (1<<CS02);

}

void timer0_stop() {

            TCCR0 &= ~(1<<CS02);
}
```