```c
/* Filename: dcu.c
 * Authors: Kevin Oei, Koen van Vliet
 * For Pro-Q2
 * Description: Digital control unit firmware for speaker system
 * Status: -
 * Notes:    - Check for MSB in parameters. If high: error condition
 *           - Implement status LEDs
 */

 /* TO-DO: Set the DDRs correctly (optional), put the right memory address values in
 setPot(), use the right value in toggleErrorLED() */

#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include "midi.h"
#include "digipots.h"
#include "uart.c"


#define SYSFREQ 16000000
#define BAUDRATE 31250
#define RATE SYSFREQ / (2*BAUDRATE) - 1
#define RATE_L RATE%256
#define RATE_H RATE/256


char sb[64];                /* The buffer for the USART input. Incoming MIDI messages are
stored in here. */
volatile char rxcnt = 0;    /* rxcnt is used when determining the offset, which is used to
pick the right index when reading sb */
volatile char rxp = 0;      /* rxp will loop from (decimal) 0 to 63, used in determining the
index of sb that needs to be approached */


void uputc(char c);
void uputs(char str[]);
char ugetc();
void spiSend(uint8_t memcom, uint8_t data);
void setPot(int potno, uint8_t val);
void toggleErrorLED(void);


int main(void) {
    int i;

    /* Debug LEDs */
    DDRB        = 0xFF;
    DDRDIGIPOTS = 0xFF;
    DDRLEDS     = 0xFF;
    /* Setup serial comms (31250-8-n-1) NOTE: THIS IS FOR PC ONLY, HAVE TO FIX FOR ATmega */
    UCSRA = 0x00;
    UCSRB = 0x18 | (1<<7);  /* Enable receiver & transmitter and RX complete interrupt
    enable */
    UCSRC = 0x86;           /* STILL NEEDS TO BE ADJUSTED WHEN TO-BE-USED DEVICE IS KNOWN */
    UBRRH = RATE_H;
    UBRRL = RATE_L;
    /* Setup SPI bus (f/2, mode0.0) */
    SPSR = (1<<0);                      /* Double SPI speed enabled */
    SPCR = (1<<7) | (1<<6) | (1<<4);    /* Enable SPI and interrupt, select master mode */

    /* Reset digital potentiometers (255 is the max position) */
    static uint8_t potpos[6] = {255};
    /* Connect terminals */
    for (i = 0; i < sizeof(potpos); i++) {
        setPot(i,potpos[i]);
    }


    sei();

    /* Main program loop */
    while (1) {
        uint8_t cmd, cc, vv;
        char s[50]; /* Is still being used anywhere? */
        while (rxcnt == 0); /* Wait for buffer to be empty */
        cmd = (uint8_t)ugetc();
        if (cmd & 0x80) { /* Check if 0b1xxxxxxx (valid MIDI command) */
            toggleErrorLED(); /* Turn off error LED */
            while (rxcnt == 0);
            cc = ugetc(); /* Acquire controller number */
            if (~cc & 0x80) { /* Check if 0b0xxxxxxx (valid controller number value) */
                while (rxcnt == 0);
                vv = ugetc(); /* Acquire controller value */
                if (~vv & 0x80) { /* Check if 0b0xxxxxxx (valid controller value) */
                    /* Check command type */
                    switch (cmd) {
                        case CTRL_CH:   snprintf(s, sizeof(s), "Controller %d = %d", cc, vv);
                                        uputs(s);
                                        potpos[cc] = vv;
                                        setPot(cc,potpos[cc]);
                                        break;
                        /*case 'r':     for (i = 0; i < sizeof(potpos); i++) {
                                            potpos[i] = 127;
                                            setPot(i,potpos[i]);
                                        }
                                        break; */
                        default:        uputs("What?");
                    }
                }
                else {
                    toggleErrorLED(); /* Turn on error LED */
                }           }
            else {
                toggleErrorLED(); /* Turn on error LED */
            }
        }
    }
    return 0;
}

/* The messages that are sent out are 16-bit long. SPI can only send 8-bit at one time */
/* Thus, two SPI transmissions are required for a full message to be sent */
/* The message is as follows: AAAA.CCDD.DDDD.DDDD where A is memory address, C is command
and D is data. */
/* See pg.47 of DigiPot datasheet */
void spiSend(uint8_t memcom, uint8_t data) {
    SPDR = memcom;                  /* Transmit memory address and command */
    while ((SPSR & (1<<7)) == 0);   /* Wait for SPI transfer to finish. */
    SPDR = data;                    /* Transmit data (the value to be written to the
    DigiPot) */
    while ((SPSR & (1<<7)) == 0);   /* Wait for SPI transfer to finish. */
}

void setPot(int potno, uint8_t val) {
    /* Select the CS of the correct IC. Note that a low signal will 'activate' the IC. */
    /* ADDRESS MEMORY STILL NEEDS TO BE FILLED IN */
    switch (potno) {
        case 0: PORTDIGIPOTS = VOLUME_CS_PIN;
                spiSend(0x00, val);
        case 1: PORTDIGIPOTS = BALANCE_CS_PIN;
                spiSend(0x00, val);
        case 2: PORTDIGIPOTS = BASS_CUT_CS_PIN;
                spiSend(0x00, val);
        case 3: PORTDIGIPOTS = TREBLE_CUT_CS_PIN;
                spiSend(0x00, val);
        case 4: PORTDIGIPOTS = BASS_BOOST_CS_PIN;
                spiSend(0x00, val);
        case 5: PORTDIGIPOTS = TREBLE_BOOST_CS_PIN;
                spiSend(0x00, val);
        default: uputs("Invalid potmeter ID selected.");
    }
    PORTB = ~(1<<potno);

}

void toggleErrorLED(void) {
    PORTLEDS ^= 0x00; /* Toggle error LED. CORRECT VALUE TO-BE-FILLED-IN */
}

ISR (USART_RXC_vect) {
    char c;
    if (UCSRA & (1<<FE | 1<<DOR | 1<<PE)) {
        c = UDR;
        uputc('?');
    }
    else {
        c = UDR;
        /*PORTB = ~c;*/
        if (rxcnt < 64) {
            sb[rxp & 63] = c;
            rxp = (rxp + 1) & 63;
            rxcnt++;
        }
        else {
            uputc('!');
        }
    }
}
```

```
/* Temporary function for PC debugging */
void uputc(char c) {
    while (~UCSRA & 1<<UDRE);
    UDR = c;
}

/* Temporary function for PC debugging */
void uputs(char str[]) {
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        uputc(str[i] | 0x80);       /* | 0x80 is debug code*/
    }
}

char ugetc() {
    char c;
    int offset;
    offset = (rxp - rxcnt) & 63;
    c = sb[offset];
    rxcnt--;
    return c;
}
```