```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 */
```

```
51      UBRRH = RATE_H;
52      UBRRL = RATE_L;
53
54      /* Setup SPI bus (f/2, mode0.0) */
55      SPSR = (1<<0);                      /* Double SPI speed enabled */
56      SPCR = (1<<7) | (1<<6) | (1<<4);    /* Enable SPI and interrupt, select master
        mode */
57
58      /* Reset digital potentiometers (255 is the max position) */
59      static uint8_t potpos[6] = {255};
60      /* Connect terminals */
61      for (i = 0; i < sizeof(potpos); i++) {
62          setPot(i,potpos[i]);
63      }
64
65
66      sei();
67
68      /* Main program loop */
69      while (1) {
70          uint8_t cmd, cc, vv;
71          char s[50]; /* Is still being used anywhere? */
72          while (rxcnt == 0); /* Wait for buffer to be empty */
73          cmd = (uint8_t)ugetc();
74          if (cmd & 0x80) { /* Check if 0b1xxxxxxx (valid MIDI command) */
75              toggleErrorLED(); /* Turn off error LED */
76              while (rxcnt == 0);
77              cc = ugetc(); /* Acquire controller number */
78              if (~cc & 0x80) { /* Check if 0b0xxxxxxx (valid controller number value)
                */
79                  while (rxcnt == 0);
80                  vv = ugetc(); /* Acquire controller value */
81                  if (~vv & 0x80) { /* Check if 0b0xxxxxxx (valid controller value) */
82                      /* Check command type */
83                      switch (cmd) {
84                          case CTRL_CH:   snprintf(s, sizeof(s), "Controller %d = %d",
                            cc, vv);
85                                          uputs(s);
86                                          potpos[cc] = vv;
87                                          setPot(cc,potpos[cc]);
88                                          break;
89                          /*case 'r':     for (i = 0; i < sizeof(potpos); i++) {
90                                              potpos[i] = 127;
91                                              setPot(i,potpos[i]);
92                                          }
93                                          break; */
94                          default:        uputs("What?");
95                      }
96                  }
97                  else {
98                      toggleErrorLED(); /* Turn on error LED */
99                  }           }
100             else {
101                 toggleErrorLED(); /* Turn on error LED */
102             }
103         }
104     }
```

```
105         return 0;
106     }
107
108     /* The messages that are sent out are 16-bit long. SPI can only send 8-bit at one
        time */
109     /* Thus, two SPI transmissions are required for a full message to be sent */
110     /* The message is as follows: AAAA.CCDD.DDDD.DDDD where A is memory address, C is
        command and D is data. */
111     /* See pg.47 of DigiPot datasheet */
112     void spiSend(uint8_t memcom, uint8_t data) {
113         SPDR = memcom;                  /* Transmit memory address and command */
114         while ((SPSR & (1<<7)) == 0);   /* Wait for SPI transfer to finish. */
115         SPDR = data;                    /* Transmit data (the value to be written to the
            DigiPot) */
116         while ((SPSR & (1<<7)) == 0);   /* Wait for SPI transfer to finish. */
117     }
118
119     void setPot(int potno, uint8_t val) {
120         /* Select the CS of the correct IC. Note that a low signal will 'activate' the
            IC. */
121         /* ADDRESS MEMORY STILL NEEDS TO BE FILLED IN */
122         switch (potno) {
123             case 0: PORTDIGIPOTS = VOLUME_CS_PIN;
124                     spiSend(0x00, val);
125             case 1: PORTDIGIPOTS = BALANCE_CS_PIN;
126                     spiSend(0x00, val);
127             case 2: PORTDIGIPOTS = BASS_CUT_CS_PIN;
128                     spiSend(0x00, val);
129             case 3: PORTDIGIPOTS = TREBLE_CUT_CS_PIN;
130                     spiSend(0x00, val);
131             case 4: PORTDIGIPOTS = BASS_BOOST_CS_PIN;
132                     spiSend(0x00, val);
133             case 5: PORTDIGIPOTS = TREBLE_BOOST_CS_PIN;
134                     spiSend(0x00, val);
135             default: uputs("Invalid potmeter ID selected.");
136         }
137         PORTB = ~(1<<potno);
138
139     }
140
141     void toggleErrorLED(void) {
142         PORTLEDS ^= 0x00; /* Toggle error LED. CORRECT VALUE TO-BE-FILLED-IN */
143     }
144
145     ISR (USART_RXC_vect) {
146         char c;
147         if (UCSRA & (1<<FE | 1<<DOR | 1<<PE)) {
148             c = UDR;
149             uputc('?');
150         }
151         else {
152             c = UDR;
153             /*PORTB = ~c;*/
154             if (rxcnt < 64) {
155                 sb[rxp & 63] = c;
156                 rxp = (rxp + 1) & 63;
157                 rxcnt++;
```

```
158         }
159         else {
160             uputc('!');
161         }
162     }
163 }
164 
165 /* Temporary function for PC debugging */
166 void uputc(char c) {
167     while (~UCSRA & 1<<UDRE);
168     UDR = c;
169 }
170 
171 /* Temporary function for PC debugging */
172 void uputs(char str[]) {
173     int i;
174     for (i = 0; str[i] != '\0'; i++) {
175         uputc(str[i] | 0x80);       /* | 0x80 is debug code*/
176     }
177 }
178 
179 char ugetc() {
180     char c;
181     int offset;
182     offset = (rxp - rxcnt) & 63;
183     c = sb[offset];
184     rxcnt--;
185     return c;
186 }
187 
188 
189 
```