```
/******************
*       comaidsystem.c
*
*       Communication Aid System:   Designed to assist on-road communication with deaf driver
*                       Hardware specs: Atmega168p microcontroller
*
*       Authors: Timmy Mbaya, Brendan Davis , Joseph Cohen
*
*        Under supervision from Betty O'Neil
*
*       Spring 2010 Real-Time Systems Independent Study, UMass Boston
*
*********************/
```

/* $Id: comaidsystem.c, version 1.0 2010/31/04 09:26:08 */

//

//

```
//


#include "delay.h"
#include "keyboard1.h"


#include "nlcd.h"


#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sfr_defs.h>
#include <avr/pgmspace.h>


#include <stdio.h>


#define KB_PCINT 11      //Pin change interrupt 11
#define SCODE_SIZE 0xF1   // Array size
#define NUM_BITS 11      // number of bits to receive (1 start, 1 parity, 1 stop)
#define BUFF_SIZE 24


void buffer_char(char);
void enable_pcint(int pcintnum);
void initTables(void);
void keyboard_setup(void);


 //special key functions
```

```
void end_codefn(char);

void E0fn(unsigned char);

void E1fn(char);

void f1fn(char);

void f2fn(char);

void f3fn(char);

void f4fn(char);

void f5fn(char);

void f10fn(char);

void bkspfn(char);

void deletefn(char);

void homefn(char);

void enterfn(char);

void escapefn(char);

void caplockfn(char);

void defaultfn(char);


unsigned char bitcount, buffcount;

volatile unsigned char control_mode;       //Mode control: execute command or write received char

unsigned char control_repeat;     //repeat control: when set it ignores repeated chars in control mode

unsigned char charsLeftToIgnore;  //number of chars to ignore after received scancode while in write
mode

unsigned char defaultChar;

unsigned char *buffptr; //pointer to increment into the char buffer


unsigned char buffer[BUFF_SIZE]; //buffer for received chars
```

```
scode scancodes[SCODE_SIZE];


//scancodes arrays in flash memory to save RAM space


const int regular_keys[][2] PROGMEM = {
{A,'A'},{B,'B'},{C,'C'},{D,'D'},{E,'E'},{F,'F'},{G,'G'},{H,'H'},{I,'I'},{J,'J'},{K,'K'},{L,'L'},{M,'M'},{N,'N'},{O,'O'},{P,'P'},
{Q,'Q'},{R,'R'},{S,'S'},{T,'T'},{U,'U'},{V,'V'},{W,'W'},{X,'X'},{Y,'Y'},{Z,'Z'},{D0,'0'},{D1,'1'},{D2,'2'},{D3,'3'},{D4,'4
'},{D5,'5'},{D6,'6'},{D7,'7'},{D8,'8'},{D9,'9'},{APOSTROPHE,'\''},{HYPHEN,'-
'},{EQUALS,'='},{BACKSLASH,'\\'},{SPACE,'
'},{TAB,'\t'},{LSQR_BRKT,'['},{ACCENT,'`'},{KP_SLASH,'?'},{KP_STAR,'*'},{KP_MINUS,'_'},{KP_PLUS,'+'},{KP_
DOT,'>'},{KP_0,'('},{KP_1,'!'},{KP_2,'@'},{KP_3,'#'},{KP_4,'$'},{KP_5,'\%'},{KP_6,'^'},{KP_7,'&'},{KP_8,'*'},{K
P_9,'('},{SLASH,'/'},{DOT,'.'},{COMMA,','},{SEMI_COLON,';'},{RSQR_BRKT,']'}};


const int other_keys[] PROGMEM={BKSP, END_CODE, CAPS, L_SHIFT, R_SHIFT, L_CTRL, L_GUI, L_ALT,
R_CTRL, R_GUI, R_ALT, APPS, ENTER, ESC, F1, F2, F3, F4, F5, F6, F6, F7, F8, F9, F10, F11, F12,
PRNT_SCRN, PAUSE, HOME, PG_UP, DELETE, END, PG_DN, U_ARROW, L_ARROW, R_ARROW,
D_ARROW, NUM, INSERT, EXTENDED, EXTENDED1, END_CODE};


//MAIN


int main() {

  bitcount = NUM_BITS;

  control_mode = 0;

  control_repeat = 0;

  charsLeftToIgnore = 1; //we ignore the first character

  defaultChar = '~';

  buffptr = buffer; //initializes buffer pointer
```

```
enable_pcint(KB_PCINT);  //Enable pin change interrupts from the keyboard clock

initTables();  //Initialize scancode tables



keyboard_setup();   //Sets up the keyboard after BAT test

sei();



nlcd_init(); // Initializes LCD

nlcd_string(PSTR("Comaidsystem 1.0"));
delay_ms(1000);
deletefn(defaultChar);


while(1) {
  ;
}
return 1;
```

```
}

//Function to intialize array of scancode structs with ascii chars and functions to execute
void initTables() {
  int i;

  nlcd_string(PSTR("."));

        //initalize scancode array with default scancode struct
        for (i = 0; i < SCODE_SIZE; i++) {
                scode scodestruct = {defaultChar,  ((void *)defaultfn)};
                scancodes[i] = scodestruct;
        }

        //loop through entire regular_keys array
        //indexing every regular scancode and assigning char from regular keys and function to execute into scancodes structs
        for (i = 0; i < ((sizeof(regular_keys))/(sizeof(regular_keys[0]))); i++) {
                scode scodestruct = {pgm_read_byte(&regular_keys[i][1]), ((void *)buffer_char)};
                lcd_write_data(scodestruct.ascii_char);
                scancodes[pgm_read_byte(&(regular_keys[i][0]))] = scodestruct;

        }

        //indexing every scancodes and assigning every other key and function to execute into scancode array's structs
```

```
        scancodes[END_CODE].scancode_function =(void *)end_codefn;

        scancodes[EXTENDED].scancode_function = (void*)E0fn;
        scancodes[EXTENDED].ascii_char = defaultChar;

        scancodes[F1].scancode_function =(void *)f1fn;

        scancodes[F2].scancode_function =(void *)f2fn;

scancodes[F3].scancode_function =(void *)f3fn;

        scancodes[F4].scancode_function =(void *)f4fn;


        scancodes[F5].scancode_function =(void *)f5fn;

        scancodes[F10].scancode_function =(void *)f10fn;

        scancodes[ESC].scancode_function =(void *)escapefn;

        scancodes[DELETE].scancode_function =(void *)deletefn;

        scancodes[ENTER].scancode_function =(void *)enterfn;

        scancodes[BKSP].scancode_function =(void *)bkspfn;
```

```
}

//END_CODE function, ignores next char
void end_codefn(char empty) {
        charsLeftToIgnore = 1;
}

//EXTENDED E0 function, enters control mode on 0xE0 and execute commands unless control repeat is set
void E0fn(unsigned char control) {

 control_mode = 1;        //Enters control mode

 /*if (control == EXTENDED)   //Enters control mode
  { control_mode = 1; }*/

 if (control_repeat == 0){

   if (control == L_ARROW) //Notifies left turn
    {
```

```
          //lcd_write_string(PSTR(" TURN LEFT "));
    nlcd_string(PSTR("<<<<<< TURN LEFT"));
    control_repeat = 1;  //No repeat will occur until key released
    control_mode = 0;


   }
if (control == R_ARROW)  //Notifies right turn
  { //lcd_write_string(PSTR(" TURN RIGHT "));
    nlcd_string(PSTR("TURN RIGHT >>>>>"));
            control_repeat = 1;  //No repeat will occur until key released
            control_mode = 0;
   }
if (control == U_ARROW)  //Notifies go ahead
  { //lcd_write_string(PSTR(" GO STRAIGHT "));
                  nlcd_string(PSTR("^^GO  STRAIGHT^^"));
            control_repeat = 1;  //No repeat will occur until key released
            control_mode = 0;
   }
if (control == D_ARROW)  //Noties U turn
  {   //lcd_write_string(PSTR(" TURN AROUND "));
      nlcd_string(PSTR("  TURN AROUND!  "));
            control_repeat = 1;  //No repeat will occur until key released
            control_mode = 0;
   }
```

```
  if (control == PG_UP)  //Signals YES
   {  //lcd_write_string(PSTR(" YES "));
      nlcd_string(PSTR("      YES      "));
            control_repeat = 1;  //No repeat will occur until key released
            control_mode = 0;
    }

   if (control == PG_DN)  //Signals NO
   {  //lcd_write_string(PSTR(" NO "));
      nlcd_string(PSTR("       NO      "));
            control_repeat = 1;  //No repeat will occur until key released
            control_mode = 0;
    }




//control_repeat = 1;     //No repeat will occur until key released
}

if (control == DELETE)  //if DELETE, clears lcd screen
 { lcd_clear_and_home();
          control_mode = 0;
  control_repeat = 0;
  }
```

```
  if (control == END_CODE)  //if END_CODE, ignores next char, exit control mode and clear
control_repeat
   { charsLeftToIgnore = 1;
            control_mode = 0;
     control_repeat = 0;
    }
}

void E1fn(char empty) {
 ;
}

void bkspfn(char empty) {
 nlcd_backspace();
}

//Delete function: resets LCD screen
void deletefn(char empty) {
 //lcd_clear_and_home();
 nlcd_clear();  //Clears screen with new LCD.
}

void homefn(char empty) {
 ;
}
```

```
//Enter function: sets lcd cursor to second line
void enterfn(char empty) {
  //lcd_line_two();
  ;
}

//Escape function: Notifies need for emergency stop
void escapefn(char empty) {
  //lcd_write_string(PSTR(" EMERGENCY STOP!! "));
  nlcd_string(PSTR("EMERGENCY STOP!!"));
}

void caplockfn(char empty) {
  ;
}

//Default function: does nothing
void defaultfn(char key) {
  ;
}

//F1 function: Notifies stop
void f1fn(char empty) {
  //lcd_write_string(PSTR(" STOP "));
```

```
  nlcd_string(PSTR("      STOP      "));
}

//F2 function: Notifies need to slow down
void f2fn(char empty) {
  //lcd_write_string(PSTR(" SLOW DOWN "));
  nlcd_string(PSTR("   SLOW  DOWN   "));
}

//F3 function: Notifies to turn on head lights
void f3fn(char empty) {
  //lcd_write_string(PSTR("TURN ON H-LIGHTS"));
  nlcd_string(PSTR("*TURN ON LIGHTS*"));
}

//F4 function: Notifies to turn off head lights
void f4fn(char empty) {
  //lcd_write_string(PSTR("TURN OFF H-LIGHT"));
  nlcd_string(PSTR("TURN OFF H-LIGHT"));
}

//F5 function: Notifies flat tire
void f5fn(char empty) {
  //lcd_write_string(PSTR(" FLAT TIRE "));
  nlcd_string(PSTR("   FLAT  TIRE   "));
```

```
}

//F10 function: Notifies need to pull over due to police ar ambulance emergency siren
void f10fn(char empty) {
  //lcd_write_string(PSTR(" PULL OVER...(SIREN) "));
  nlcd_string(PSTR("PULL OVER..SIREN"));
}

//buffer_char function: Buffers received and decoded characters
void buffer_char(char thechar) {
 //lcd_write_data(thechar);
 nlcd_write(thechar);

  //lcd_write_int16(thechar);
  /*if (buffcount < BUFF_SIZE) {  //buffers the received char
    *buffptr = thechar;
    buffptr++;
    buffcount++;
    if (buffptr >= buffer + BUFF_SIZE)
      buffptr = buffer;
  } */
}

//Interrupt Service Routine for pin change interrupts from PCINT8-14
```

```
ISR(PCINT1_vect) {
  static unsigned char char_data = 0;

  //if negative edge, ignore start, parity and stop bits and read bit
  if ( (PINC & (1 << PINC3)) == 0 ) {
    if (bitcount < NUM_BITS && bitcount > 2) {
      char_data = (char_data >> 1);
      if (PINC & 0x4)
        char_data = char_data | 0x80;
    }

          //if we received a byte
    if (--bitcount == 0) {
                    delay_ms(1);
      //lcd_write_int16(char_data);
      bitcount = NUM_BITS;
            if (  control_mode == 0 ){
                    if (charsLeftToIgnore > 0)
                            charsLeftToIgnore--;
                    else{

                    (scancodes[char_data].scancode_function)(scancodes[char_data].ascii_char);



                    }
```

```
    }
        else{
          E0fn(char_data);
    }
   }
  }
}

//Enables pin change interrupts
void enable_pcint(int pcintnum) {
  lcd_write_string(PSTR("."));
  nlcd_string(PSTR("."));

  if ((pcintnum >= 0) && (pcintnum < 8)) {
    ;
  }

  if ((pcintnum >= 8) && (pcintnum <= 14)) {
    PCICR  |= 0x2;  //Enables pin change interrupts from PCINT8-14
    PCMSK1 |= 0x8;  //Unmasks pin change interrupt from PCINT11 only

    DDRC  &= ~(1 << DDC3);   //Sets PINC3 as input for data
    PORTC |= (1 << PORTC3); //Sets pull-up on PINC3
    MCUCR |= (1 << PUD);    //Completes Tri-state (Hi-Z) DDxn:0 PORTxn:1 PUD:1 (MCUCR)
```

```
   DDRC  &= ~(1 << DDC2);   //Sets PINC2 as input for data
   PORTC |= (1 << PORTC2); //Sets pull-up on PINC2
   MCUCR |= (1 << PUD);    //Completes Tri-state (Hi-Z) DDxn:0 PORTxn:1 PUD:1 (MCUCR)
  }

  if ((pcintnum >= 16) && (pcintnum <= 23)) {
    ;
  }
}


//Sets up the keyboard fater BAT test, clears keyboard buffer and resets
void keyboard_setup(void)
{

  nlcd_string(PSTR("."));

  //Set up for output
  DDRC  |= (1 << DDC2);
  PORTC |= (1 << PORTC2);

  PINC = 0xFF;  //Tell keyboard to reset.

  delay_ms(1000);
```

```
  //Set up for input again
  DDRC  &= ~(1 << DDC2);
  PORTC |= (1 << PORTC2);
}
```