```
/******************
*          comaidsystem_functions.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
*
*********************/


/* Copyright (c) 2010 Timmy Mbaya, Brendan Davis, Joseph Cohen



  All rights reserved.


  Redistribution and use in source and binary forms, with or without

  modification, are permitted provided that the following conditions are met:


  * Redistributions of source code must retain the above copyright

    notice, this list of conditions and the following disclaimer.
```

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

//
```

```
//
//

#include "delay.h"
#include "keyboard.h"
#include "nlcd.h"
#include "comaidsystem_functions.h"

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

#include <stdio.h>

unsigned char buffcount;                  //buffer counter

 unsigned char program_mode;      //Programming Mode: control to program text to display for a specific Fn key
 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 program_default;                    //default programming control: controls the programming of default text into Fx keys
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


//Programmable char arrays in EEPROM for F-keys programmable text displays

uint8_t EEMEM F1text[18];

uint8_t EEMEM F2text[18], EEMEM F3text[18], EEMEM F4text[18], EEMEM Fotext[18], EEMEM
F5text[18], EEMEM F6text[18], EEMEM F7text[18], EEMEM F8text[18], EEMEM F9text[18], EEMEM
F10text[18], EEMEM F11text[18], EEMEM F12text[18];

uint8_t EEMEM F01text[18]; uint8_t EEMEM F02text[18];


//Buffer for strings read from and written to EEPROM

unsigned char Fntext[18];


 unsigned char shift_pressed;      //Control for shift key

 unsigned char shift_release;      //Control for shift key release (END_CODE + SHIFT scancode)


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

scode scancodes[SCODE_SIZE];



//scancodes arrays in flash memory to save RAM space

const unsigned char 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,'
'},{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,'*'},{KP_9,'('},{S
LASH,'/'},{DOT,'.'},{COMMA,','},{SEMI_COLON,';'},{RSQR_BRKT,']'}};


//shifted keys array in flash memory

const unsigned char shifted_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,')'},{D1,'!'},{D2,'@'},{D3,'#'},{D4,'$
'},{D5,'\%'},{D6,'^'},{D7,'&'},{D8,'*'},{D9,'('},{APOSTROPHE,'\"'},{HYPHEN,'_'},{EQUALS,'+'},{BACKSLASH,'|'}
,{LSQR_BRKT,'{'},{ACCENT,'~'},{SLASH,'?'},{DOT,'>'},{COMMA,'<'},{SEMI_COLON,':'},{RSQR_BRKT,'}'}};


//This array was an option for default texts but we run out of memory when we implement it. So we
instead use the individual strings after the next line.

//const char* default_Ftext[13] PROGMEM= { "EMERGENCY STOP","      STOP      ", "   SLOW  DOWN
","*TURN ON LIGHTS*", "TURN OFF H-LIGHT", "BIG TRUCK BEHIND", "TURN OFF WIPERS", "TURN
SIGNAL OFF ", "TAKE NEXT EXIT ", "TURN WIPERS ON ", "PULL OVER..SIREN", "HONK FROM LEFT ",
"HONK FROM RIGHT " };

unsigned char F1default[] PROGMEM="      STOP      ",F2default[]PROGMEM="   SLOW  DOWN
",F3default[]PROGMEM="*TURN ON LIGHTS*",F4default[]PROGMEM="TURN OFF H-
LIGHT",F5default[]PROGMEM="BIG TRUCK BEHIND",F6default[]PROGMEM="TURN OFF WIPERS
",F7default[]PROGMEM="TURN SIGNAL OFF ",F8default[]PROGMEM="TAKE  NEXT
EXIT",F9default[]PROGMEM="TURN  WIPERS  ON",F10default[]PROGMEM="PULL
OVER..SIREN",F11default[]PROGMEM="HONK  FROM  LEFT",F12default[]PROGMEM="HONK  FROM
RIGHT";


unsigned char scroll_text;


//Function to intialize array of scancode structs with ascii chars and functions to execute

void initTables()

{

  int i;
  //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)};

    scancodes[pgm_read_byte(&(regular_keys[i][0]))] = scodestruct;

  }

    //indexing special key scancodes and assigning them ascii chars and functions 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 *)Fkeys_Function; //f1fn;
```

```
scancodes[F1].ascii_char =(unsigned char) 1;

scancodes[F2].scancode_function =(void *)Fkeys_Function; //f2fn;
scancodes[F2].ascii_char =(unsigned char) 2;

scancodes[F3].scancode_function =(void *)Fkeys_Function; //f3fn;
scancodes[F3].ascii_char =(unsigned char) 3;

scancodes[F4].scancode_function =(void *)Fkeys_Function; //f4fn;
scancodes[F4].ascii_char =(unsigned char) 4;

scancodes[F5].scancode_function =(void *)Fkeys_Function; //f5fn;
scancodes[F5].ascii_char =(unsigned char) 5;

scancodes[F6].scancode_function =(void *)Fkeys_Function; //f6fn;
scancodes[F6].ascii_char =(unsigned char) 6;

scancodes[F7].scancode_function =(void *)Fkeys_Function; //f7fn;
scancodes[F7].ascii_char =(unsigned char) 7;

scancodes[F8].scancode_function =(void *)Fkeys_Function; //f8fn;
scancodes[F8].ascii_char =(unsigned char) 8;

scancodes[F9].scancode_function =(void *)Fkeys_Function; //f9fn;
scancodes[F9].ascii_char =(unsigned char) 9;
```

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

    scancodes[F10].ascii_char =(unsigned char) 10;


    scancodes[F11].scancode_function =(void *)Fkeys_Function; //f11fn;

    scancodes[F11].ascii_char =(unsigned char) 11;


    scancodes[F12].scancode_function =(void *)Fkeys_Function; //f12fn;

    scancodes[F12].ascii_char =(unsigned char) 12;


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


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


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


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


    scancodes[L_CTRL].scancode_function =(void *)l_ctrlfn;


    scancodes[L_SHIFT].scancode_function = (void *)shiftfn;


    scancodes[R_SHIFT].scancode_function = (void *)shiftfn;

  }
```

```
//Function to initialize states of system global variables
void init_sysvarStates(void)
{

  buffcount = 1;

  program_mode = 0; //program_modeF1=1; program_modeF2=1; program_modeF3=1;
program_modeF4=1; program_modeF5=1; program_modeF6=1; program_modeF7=1;
program_modeF8=1; program_modeF9=1; program_modeF10=1; program_modeF11=1;
program_modeF12=1;
  control_mode = 0;
  control_repeat = 0;
  shift_pressed = 0;
  shift_release = 1;
  charsLeftToIgnore = 1;        //we ignore the first character
  defaultChar = '~';
  buffer[0]='\0';              //initialize buffer's first slot with NUll terminator
  buffptr = buffer;            //initializes buffer pointer
  scroll_text = 0;
  program_default = -1;        //initialize three-state control variable to -1


}

//Function to process received scancode within PCINT ISR
void process_scancode(unsigned char char_data)
{
  if (control_mode == 0) {             //continue if not in control mode


     if (charsLeftToIgnore > 0)

       charsLeftToIgnore--;


     else {


      if (shift_pressed == 0)

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

      else

       (scancodes[L_SHIFT].scancode_function)(char_data);    //if SHIFT has been pressed before, only
this executes


      if (program_mode == 1)           //If we are in programming mode, we also buffer the char for
programming

       programfn(char_data);

     }


    }

    else

      E0fn(char_data);                  //If in control mode, skip previous 10 lines and just call E0 function

}


//END_CODE function, ignores next char

void end_codefn(unsigned 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
       {
         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
       {
         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
 {
  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
 {
     nlcd_string(PSTR("  TURN AROUND!  "));
   control_repeat = 1;  //No repeat will occur until key released
   control_mode = 0;
   }


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


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

```
    control_mode = 0;
   }


  //the following controls the setting of the default programming variable,program_default
  if (control == HOME)
   {

     if (program_mode == 1)
      {
        program_default = 1;

       }

    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
 {
  nlcd_wipe();       //Wipes LCD screen and buffer
  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(unsigned char empty) {
  ;
}

void bkspfn(unsigned char empty) {
  nlcd_char(BACKSPACE);
}

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

//Left CONTROL function: enbales programming of Fx keys
void l_ctrlfn(unsigned char a_char) {
   program_mode = 1;  //enters programming mode
}

void homefn(unsigned char empty) {
  ;
}

//Enter function: not used at the moment
void enterfn(unsigned char empty) {
  ;
}

//Escape function: Notifies need for emergency stop
void escapefn(unsigned char empty) {
  nlcd_string(PSTR("EMERGENCY STOP!!"));
  nlcd_flash(5);
}

void caplockfn(unsigned char empty) {
  ;
}

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

```
}


//Function called by all Fx keys: its argument determines which arguments to pass to FxFunction

void Fkeys_Function(unsigned char Fx_char)

{

  switch(Fx_char){


    case (unsigned char)1:  FxFunction(F01text, F1default/*"      STOP      "*/);

            break;


    case (unsigned char)2:  FxFunction(F02text, F2default/*"   SLOW DOWN   "*/);

            break;


    case (unsigned char)3:  FxFunction(F3text, F3default/*"TURN ON H-LIGHTS"*/);

            break;


    case (unsigned char)4:  FxFunction(F4text, F4default/*"TURN H-LIGHTS OFF"*/);

            break;


    case (unsigned char)5:  FxFunction(F5text, F5default/*"BIG TRUCK BEHIND"*/ );

            break;


    case (unsigned char)6:  FxFunction(F6text, F6default/*"TURN OFF WIPERS "*/);

            break;
  case (unsigned char)7:  FxFunction(F7text, F7default/*"TURN WIPERS ON "*/);
        break;

  case (unsigned char)8:  FxFunction(F8text, F8default/*"TAKE NEXT EXIT "*/);
        break;

  case (unsigned char)9:  FxFunction(F9text, F9default/*"TURN SIGNAL OFF "*/);
        break;

  case (unsigned char)10: FxFunction(F10text,F10default/*"PULL OVER..SIREN"*/);
        break;

  case (unsigned char)11: FxFunction(F11text,F11default/*"HONK FROM LEFT "*/);
        break;

  case (unsigned char)12: FxFunction(F12text,F12default/*"HONK FROM RIGHT "*/);
        break;
  }

}


//called within Fkeys_Function based on Fx_char
void FxFunction(uint8_t* Fxtext, unsigned char* Default_Fxstring)
{
   if (program_mode == 1)                   //if we are in programming mode
    {

      program(Fxtext, Default_Fxstring);     //program

      program_mode = 0;
    }


  else {                                  //otherwise read text from eeprom into Fntext[] and send to LCD


     //Read string from EEPROM
     read_eeprom_string(Fxtext);

     nlcd_vstring(Fntext);
  }

}

//Function to handle the shift key
void shiftfn(unsigned char char_received)
{

  if (shift_pressed == 0) {             //if the shift key was just pressed
    shift_pressed = 1;              //toggle states

    shift_release = 0;

  }


  else if (char_received == END_CODE)   //if END_CODE was received, be ready to exit shift mode

    shift_release = 1;


  else {

     if ( (shift_release == 1) && ((char_received == L_SHIFT) || (char_received == R_SHIFT)))

       { shift_pressed = 0; }


     else if( (shift_release == 1) && ((char_received != L_SHIFT) && (char_received != R_SHIFT)))

       { shift_release = 0;}       //if the code after END_CODE is not SHIFT, stay in shift mode




     else {                        //if in full shift mode scan shifted_keys array for key match and print

       int i = 0;

       for ( ; (pgm_read_byte(&shifted_keys[i][0]) != char_received) &&
(pgm_read_byte(&shifted_keys[i][0]) != NULL); i++)

       ; // Do nothing


       if (pgm_read_byte(&shifted_keys[i][0]) == char_received) {

       nlcd_char((unsigned char)pgm_read_byte(&shifted_keys[i][1]));

       if (program_mode == 1)

         programfn(char_received);

      }
```

```
    }



  }

}


//buffer_char function: Buffers received and decoded characters. But actually just prints the char to LCD

void buffer_char(unsigned char thechar)

{

 nlcd_char(thechar);

}




//Function to buffer received chars for programming

void programfn(unsigned char the_char)

{

  if(the_char == ENTER)

    {

      *buffptr = '\0';

      //nlcd_string(PSTR("Press Fn Key:"));

    }


  else if(the_char == BKSP)

    {

      *(--buffptr) = '\0';      //deletes last buffered character
```

```
   buffcount--;

  }

 else {
  if (buffcount < BUFF_SIZE) {     //buffers the received char
   if ( (scancodes[the_char].ascii_char) == defaultChar)
    ;
   else {
    *buffptr = (scancodes[the_char].ascii_char);
    buffptr++;
    *buffptr = '\0'; //for safety always place Null terminator after inserting new char
    buffcount++;
   }
  }
  if (buffptr >= buffer + BUFF_SIZE)
    buffptr = buffer;
 }



}



//Function to program either received chars or default text into F-key eeprom addresses
```

```
void program (unsigned char *keytext, unsigned char* default_text)

{

  uint8_t charbuf; uint8_t i =0;

  unsigned char* textptr = keytext;


 if(program_default != 1)            //continue if no default programming signal received

  {

  buffptr = buffer;

  while((*buffptr)!= '\0'){

     *textptr = *buffptr;


     charbuf = (uint8_t)(*textptr);


     eeprom_write_byte((uint8_t *)& keytext[i], charbuf);  //write each char from RAM buffer to EEPROM array


     textptr++;

     buffptr++; i++;

   }

   *textptr = '\0';


   charbuf = (uint8_t)(*textptr);


   eeprom_write_byte((uint8_t *)& keytext[i], charbuf);   //Write NULL Terminator('\0') from RAM to EEPROM string
```

```
    delay_ms(4000); //wait for EEPROM to settle


    buffptr = buffer;


    buffcount = 1;
  }


  else if (program_default == 1)         //if default programming signal was received, only program from
default_text
  {
    buffptr = default_text;


   while (  (pgm_read_byte(&default_text[i])) != '\0' )
       {
         *textptr = pgm_read_byte(&default_text[i]);              //copy each default char from Flash to RAM


            charbuf = (uint8_t) (*textptr);


           eeprom_write_byte((uint8_t *) &keytext[i], charbuf) ;   //write each char from RAM buffer to
EEPROM array


            buffptr++;
          textptr++; i++;
       }
     *textptr = '\0';
     charbuf = (uint8_t)(*textptr);
   eeprom_write_byte((uint8_t *) &keytext[i], charbuf  ) ;     //Write NULL Terminator('\0') from RAM to EEPROM string

    delay_ms(4000); //wait for EEPROM to settle

   program_default = -1;                                  //reset (un-assert) default programming signal

   buffptr = buffer;                                      //reassign buffptr to buffer for future use
  }
}


//Function to read contents from eeprom addresses into RAM buffer (Fntext)
void read_eeprom_string(const uint8_t * Ftext)
{
  uint8_t  ramchar; uint8_t i = 0;

  unsigned char* textbuffer = Fntext;

    //Read string from EEPROM
    while((ramchar= eeprom_read_byte((const uint8_t*)&Ftext[i])) != '\0'){       //read from eeprom to RAM
      *textbuffer = ramchar;
      textbuffer++;
      i++;
    }

    *textbuffer = '\0';



}
```