```
/* Lab 4 code. This code was written by the group at 25-27B.
11/13/09
Michael O'Keefe, Vanessa Alphonse, Evan Frank
And the members of the other group, whose names I do not know.
This code is designed to implement the electronic compass, ultrasonic ranger,
LCD screen, numeric keypad, A/D battery voltage detection,
steering servo and speed control servo on the C8051 "smart car."
*/
******************************************************

#include <c8051_SDCC.h>
#include <stdio.h>
#include <stdlib.h>
#include <i2c.h>

#define ranger_addr 0xE0
#define PW_MIN 2028
#define PW_MAX 3502
#define PW_NEUT 2765

void Port_Init(void);
void PCA_Init (void);
void SMB_Init(void);
void XBR0_Init(void);
void Drive_Init(void);
void Steering_Servo(void);
void Steering_Calabrate(void);
void Compass_Calibrate(void);
void Steering_Control(unsigned int desired_heading, unsigned int
current_heading);
void PCA_ISR ( void ) interrupt 9;
unsigned int ReadCompass(void);
void wait(unsigned int);
unsigned int ReadRanger(void);
void ADC_Init();
unsigned char read_AD_input(unsigned char);
int findVoltage(void);

void Drive_Motor(unsigned int);
void Range_Update(void);
void heading_int(void);
void neutral_range_int(void);
```

```
unsigned int printcount = 0;
unsigned int PW_CENTER = 2764;  //62772;
unsigned int PW_RIGHT = 3502;   //63509;
unsigned int PW_LEFT = 2027;    //62034;
unsigned int SERVO_PW = 0;
unsigned int new_heading = 0;
unsigned int h_count = 0;
unsigned int heading = 0;
unsigned int desheading;
int compass_calibration = 0;
int tempcali;
sbit at 0xB7 Steering_Switch;
char keypad = -1;
unsigned int MOTOR_PW = 0;
unsigned char r_count = 0;
unsigned int count = 0;
bit new_range = 0;
unsigned int cmrange = 0;
unsigned char Data[2];
int adinput = 0;
unsigned int adcount = 0;
int voltage = 0;
sbit at 0xB6 SS;
unsigned char neutral_range = 45;


//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
void main(void)
{
// initialize board
Sys_Init();
putchar(' '); //the quotes in this line may not format correctly

XBR0_Init();
SMB_Init();
PCA_Init();
Drive_Init();
Port_Init();
ADC_Init();
```

```
//printf("start");

Steering_Calibrate();
Compass_Calibrate();
heading_int();
neutral_range_int();
while(1){
    if(adcount >50 && CF == 0){
        voltage = findVoltage();
        lcd_clear();
        lcd_print("Voltage: %d\n", voltage);
        lcd_print("Distance: %d\n", cmrange);
        lcd_print("Heading: %d\n", heading);
        adcount = 0;
        }


    if (SS){
            Range_Update(); //update the range
            if (cmrange <= 10)
                Drive_Motor(PW_MAX); //if range > 90, drive the motor in full
reverse
            else if (cmrange >= 90)
                Drive_Motor(PW_MIN); //if range < 10, drive the motor in full
forward
            else if ((cmrange >= (neutral_range - 5)) && (cmrange <=
(neutral_range + 5)))
                Drive_Motor(PW_NEUT); //if range is within 5 of neutral_range,
put it in neutral
            else
            {
                Drive_Motor(3502 - (18.425*(cmrange-10))); //otherwise, drive
it according to this equation
            }
        }
    else
        Drive_Motor(PW_NEUT); //if ss is not flipped, put it in neutral

    if(Steering_Switch){
    if (new_heading){ // enough overflows for a new heading
```

```
        heading = ReadCompass();
        tempcali = heading + compass_calibration;
        if (tempcali <= 0){
                heading = 3600 + tempcali;
                }
        else if(tempcali >= 3600){
                heading = 3600-tempcali;
                }
        else{
            heading = heading + compass_calibration;
        }
        new_heading = 0;
    Steering_Control(desheading,heading);
    }

}
   else {
    SERVO_PW = PW_CENTER;
    PCA0CPL0 = (0xFFFF - SERVO_PW);
    PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;

   }
}

}

void neutral_range_int()
{
    char tempval;
    bit valid = 1;
    wait(500);
    lcd_print("Neutral range default: %d\n", neutral_range);
    lcd_print("Input your own value between 30-80 cm\n");
        printf("1\n\r");
    while(valid){
        tempval = -1;
        while(tempval == -1){
            tempval = read_keypad(); //read a digit in from the keypad
                    printf(""); //helps if this is here. Not sure why.
        }
        neutral_range = (tempval - 48) * 10; //convert ascii to correct decimal
value (tens place)
```

```
        printf(""); //helps if this is here. Not sure why.
        while (tempval != -1){
            tempval = read_keypad(); //debouncing
        }

        printf(""); //helps if this is here. Not sure why.
        while(tempval == -1){
            tempval = read_keypad();
            printf(""); //helps if this is here. Not sure why.
        }
        neutral_range += tempval - 48; //convert ascii to correct decimal value
(ones place)
        printf("Neutral range is %d\r\n", neutral_range);

        if(neutral_range <= 80 && neutral_range >= 30){
            valid = 0; //check if the neutral range input is in the right range
            }
        else {
            lcd_clear();
            lcd_print("Invalid input; try again (30-80 cm is valid)\n\r");
            while (tempval != -1){
            tempval = read_keypad(); //debouncing
            }
        }

        }
}

int findVoltage(void){
    float advolt;
    adinput = read_AD_input(7); //read the voltage on pin 1.7 and convert it to
an unsigned char
    advolt = adinput;
    advolt = 15 *(advolt / 256); //do some math, get a float out between
0-15(V)
    return advolt;
}

//-------------------------------------------------------------------------
// Port_Init
//-------------------------------------------------------------------------
//
```

```
// Set up ports for input and output
//
void Port_Init(){
    P1MDOUT = 0xFF; //set output pin for CEX0 in push-pull mode

    P3MDOUT &= ~0xC0; //set Port 3, pin 6 and P3.7 to open-drain mode (input)
    P3 |= 0xC0; //Write a logic high to P3.6 and P3.7


    P1MDIN &= ~0x80; //Set Port 1, Pin 7 to analog input
    P1MDOUT &= ~0x80; //Set Port 1, Pin 7 to open drain mode (input)
    P1 |= 0x80; //Set Port 1, Pin 7 to logic high

}

//-----------------------------------------------------------------------------
// XBR0_Init
//-----------------------------------------------------------------------------
//
// Set up the crossbar
//
void XBR0_Init(){
XBR0 = 0x27;    //configure crossbar with UART, SPI, SMBus, and CEX channels
}

//-----------------------------------------------------------------------------
// PCA_Init
//-----------------------------------------------------------------------------
//
// Set up Programmable Counter Array
//
void PCA_Init(void)
{
         PCA0MD = 0x81;             // SYSCLK/12, enable CF interrupts, suspend
when idle
         PCA0CPM0 = 0xC2;           // 16 bit, enable compare, enable PWM
         PCA0CPM2 = 0xC2;
         EIE1 |= 0x08;              // enable PCA interrupts
         PCA0CN |= 0x40;            // enable PCA
         EA = 1;                    // enable all interrupts

}
```

```
//-----------------------------------------------------------------------------
// PCA_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for Programmable Counter Array Overflow Interrupt
//
void PCA_ISR (void) interrupt 9
{
         if (CF){
                 PCA0L = 0xFF;     // low byte of start count
                 PCA0H = 0x6F;     // high byte of start count
                 CF = 0;           // Very important - clear interrupt flag
         }
         else PCA0CN &= 0xC0; // all other type 9 interrupts

         h_count++;
         if(h_count>=2){
            new_heading=1; // 2 overflows is about 40 ms
            h_count = 0;
         }
         count++;
        r_count++;
        if(r_count>=4)
        {
            new_range=1;  // 4 overflows is about 80 ms
            r_count = 0;
        }

        adcount++;
}

void Steering_Calibrate(void){
    char input = 'r';
    //print beginning message
    printf("Embedded Control Steering Calibration\r\n");
    //set initial value for steering (set to center)
    SERVO_PW = PW_CENTER;
    printf("\r\nCalibrate center!\r\n");
    SERVO_PW = PW_CENTER;
    printf("Press 'l' to calibrate left and 'r' to calibrate right. Press 'y'
to confirm.\r\n");
```

```
    while (input != 'y'){
        input = getchar();
        if(input == 'r'){
                SERVO_PW = SERVO_PW + 2; //increase the steering pulsewidth by
10
                PW_CENTER = SERVO_PW;
        }
        else if(input == 'l'){
                SERVO_PW = SERVO_PW - 2; //decrease the steering pulsewidth by
10
                PW_CENTER = SERVO_PW;
        }
        PCA0CPL0 = (0xFFFF - SERVO_PW);
        PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;
    }
    input = 'x';
    printf("\r\nCalibrate Left!\r\n");
    SERVO_PW = PW_LEFT;
    printf("Press 'l' to calibrate left and 'r' to calibrate right. Press 'y'
to confirm.\r\n");
    while (input != 'y'){
        input = getchar();
        if(input == 'r'){
                SERVO_PW = SERVO_PW + 4; //increase the steering pulsewidth by
10
                PW_LEFT = SERVO_PW;
        }
        else if(input == 'l'){
                SERVO_PW = SERVO_PW - 4; //decrease the steering pulsewidth by
10
                PW_LEFT = SERVO_PW;
        }
        PCA0CPL0 = (0xFFFF - SERVO_PW);
        PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;
    }

    input = 'x';
    printf("\r\nCalibrate Right!\r\n");
    SERVO_PW = PW_RIGHT;
    printf("Press 'l' to calibrate left and 'r' to calibrate right. Press 'y'
to confirm.");
    while (input != 'y'){
```

```
        input = getchar();
        if(input == 'r'){
                SERVO_PW = SERVO_PW + 4; //increase the steering pulsewidth by
10
                PW_RIGHT = SERVO_PW;
        }
        else if(input == 'l'){
                SERVO_PW = SERVO_PW - 4; //decrease the steering pulsewidth by
10
                PW_RIGHT = SERVO_PW;
        }
        PCA0CPL0 = (0xFFFF - SERVO_PW);
        PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;
    }
    printf("\r\nCalibration Complete!\r\n");
}

unsigned int ReadCompass(){
    unsigned char addr = 0xC0;                          // the address of the
sensor, 0xC0 for the compass
    unsigned char Data_com[2];                           // Data is an array
with a length of 2
    unsigned int heading;                               // the heading returned
in degrees between 0 and 3599.

    i2c_read_data(addr,2,Data_com,2);                    // read two byte,
starting at reg 2

    heading =(((unsigned int)Data_com[0] << 8) | Data_com[1]);  //combine the
two values
                                                        //heading has units of
1/10 of a degree
    return heading;                                     // the heading returned
in degrees between 0 and 3599.
    }

void SMB_Init(void){
    SMB0CR=0x93;
    ENSMB=1;
}

void Compass_Calibrate(void){
```

```
    char input = 'r';
    printf("Press 'u' to calibrate up and 'd' to calibrate down.\r\n");
    printf("A headding of due north should print out a 0.\r\n");
    printf("Press 'y' to confirm.\r\n");

    while (input != 'y'){
        input = getchar();
        if(input == 'u'){
                compass_calibration = compass_calibration + 5;
        }
        else if(input == 'd'){
                compass_calibration = compass_calibration - 5;
        }
        if (new_heading){ // enough overflows for a new heading
            heading = ReadCompass();

            if (heading+compass_calibration < 0){
                tempcali = heading + compass_calibration;
                heading = 3600 + tempcali;
                }
            else if(heading + compass_calibration > 3600){
                tempcali = heading + compass_calibration;
                heading = 3600- tempcali;
                }
            else
            heading = heading + compass_calibration;
            printf("Heading is, %d\r\n", heading); // print heading
            new_heading = 0;
        }
        }
    printf("\r\nCalibration Complete!\r\n");
}

void Steering_Control(unsigned int desired_heading, unsigned int
current_heading){
    int constant = 1.4;
    int steering_error;
    int steering_control;

    if(desired_heading > 1800){
          desired_heading = desired_heading - 3600;
          }
```

```
    steering_error = current_heading - desired_heading;

    if (steering_error < 1800){
         steering_control = 0-steering_error;
         }
    else {
         steering_control = 3600-steering_error;
        }

    if(steering_control > 0){
            //printf("%d\r\n",steering_control);
            SERVO_PW = PW_CENTER + constant*steering_control;
    }
    else if(steering_control < 0){
            //printf("  %d\r\n",steering_control);
            SERVO_PW = PW_CENTER + constant*steering_control;
    }
    else {
        SERVO_PW = PW_CENTER;
    }



    if (SERVO_PW > PW_RIGHT)
        SERVO_PW = PW_RIGHT;
    else if (SERVO_PW < PW_LEFT)
        SERVO_PW = PW_LEFT;


    /*
    printcount++;
        if(printcount > 5){
            printf("Heading is:          %d\r\n", current_heading); // print
heading
            printf("Desired heading is:  %d\r\n", desired_heading);
            printf("Pulse Width is:      %d\r\n\r\n", SERVO_PW);
            printf("Range = %d, motor pulsewidth = %d\r\n", cmrange, MOTOR_PW);
            printcount = 0;
        }
    */
    PCA0CPL0 = (0xFFFF - SERVO_PW);
    PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;
```

```
}

void ADC_Init(void)
{
    REF0CN = 0x03;                      /* Set Vref to use internal reference
voltage (2.4 V) */
    ADC1CN = 0x80;                      /* Enable A/D converter (ADC1) */
    ADC1CF |= 0x01;                     /* Set A/D converter gain to 1 */
}

unsigned char read_AD_input(unsigned char n)
{
    AMX1SL = n;                         /* Set P1.n as the analog input for ADC1
*/
    ADC1CN = ADC1CN & ~0x20;            /* Clear the ìConversion Completedî flag
*/
    ADC1CN = ADC1CN | 0x10;             /* Initiate A/D conversion */
    while ((ADC1CN & 0x20) == 0x00);    /* Wait for conversion to complete */
    return ADC1;                        /* Return digital value in ADC1 register
*/
}

void Drive_Init(void)
{
    //set initial value
    MOTOR_PW = PW_NEUT;

    PCA0CPL2 = 0xFFFF - MOTOR_PW; //set low byte of motor CCM PW register
    PCA0CPH2 = (0xFFFF - MOTOR_PW) >> 8; //set high byte
    wait(1000); //make sure the motor sits in neutral for a second
}

void wait(unsigned int waitTime)
{
    count = 0; //reset count
    while ((count * 20) <= waitTime)
    {
        printf(""); //this is necessary. Not sure why.
    }
    count = 0;
}
```

```
void Range_Update(void)
{
    if (new_range)
    {
        new_range = 0; //reset the new_range flag
        cmrange = ReadRanger(); //get the range back from the ranger
        Data[0] = 0x51;  //write 0x51 to reg 0 of the ranger:
        i2c_write_data(ranger_addr, 0,  Data, 1) ;  // write one byte of data
to reg 0 at addr
    }
}


void Drive_Motor(unsigned int motorval)
{
    MOTOR_PW = motorval; //set the motor_pw to whatever value was passed in
    PCA0CPL2 = 0xFFFF - MOTOR_PW; //set low byte
    PCA0CPH2 = (0xFFFF - MOTOR_PW) >> 8; //set high byte

}

unsigned int ReadRanger(void)
{
    unsigned int range =0;
    i2c_read_data(ranger_addr, 2, Data, 2);  // read two bytes, starting at reg
2
    range = (((unsigned int)Data[0] << 8) | Data[1]); //concatenate the two
bytes.
    return range;
}

void heading_int(void){
    printf("press keypad to enter heading\n\r");
    printf("2 - North\n\r");
    printf("6 - East\n\r");
    printf("8 - South\n\r");
    printf("4 - West\r\n\n\r");

    while (keypad == -1){
       keypad = read_keypad();
       }
```

```
    if(keypad == '6'){
        desheading = 900;
        printf("East selected\n\r");
        }
    else if(keypad == '8'){
        desheading = 1800;
        printf("South selected\n\r");
        }
    else if(keypad == '4'){
        desheading = 2700;
        printf("West selected\n\r");
        }
    else{
        desheading = 0;
        printf("North selected\n\r");
        }
}
```