Individual Coding Parts

IN1900 - Hardware Project
Batch 20 – IT
Group 47
2021 - 2022

Contents

Herath P.A.U.D – 204074M

```
/*
 * CFile1.c
 *
 * Created: 11/29/2021 5:49:16 AM
 *  Author: Upeksha Herath
 */ 

#include "../defines.h"

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

/*
       To initialize the USART connection between two components
       =========================================================
       
       UCSRB register the most used bits are the RXEN and TXEN
       UCSRC and the UBRRH share same address. To select the UCSRC we have to give 1 
to URSEL else it will write in UBRRH register (because default value is 0).
       UCSZ0 and UCSZ1 sets the frame size. We have set that to 8 bits in this 
function 2nd line.
*/
void UART_init(long USART_BAUDRATE) {
       UCSRB |= (1 << RXEN) | (1 << TXEN);                                     // 
Enable USART transmission (of transmitter) and reception (of receiver)
       UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);   // Use 8-bit character 
sizes
       UBRRL = BAUD_PRESCALE;
       // Load lower 8-bits of the baud rate value
       UBRRH = (BAUD_PRESCALE >> 8);                                           // 
Load upper 8-bits
}

/*
       To receive data
       ===============
*/
unsigned char UART_RxChar() {
       while ((UCSRA & (1 << RXC)) == 0); // Wait till data is received
       return(UDR);                                  // Return the byte
}

/*
       To Transmit data
       =================
*/
void UART_TxChar(char ch) {
       while (! (UCSRA & (1<<UDRE)));     // Wait for empty transmit buffer
       UDR = ch ;
}

/*
       To send String data
       ===================
*/
void UART_SendString(char *str) {
       unsigned char j=0;         //because int allocate more memory unsigned char 
can be used to store the int values
```

```
       while (str[j]!=0) {         // Send string till null
              UART_TxChar(str[j]);
              j++;
       }
}



/*
    GPS Information extraction using ATmega16/32
       Author : Upeksha Herath
*/

#include "../defines.h"
#define BAUD 9600       //BAUDRATE = 9600

//This will initialize the GPS values according to the current GPS coordinates
void GPS_init() {
       int flag = 0;
       while (flag != 1) {
              value = UART_RxChar();
              if (value == '$') {
                     value = UART_RxChar();
                     if (value == 'G') {
                            value = UART_RxChar();
                            if (value == 'P') {
                                   value = UART_RxChar();
                                   if (value == 'G') {
                                          value = UART_RxChar();
                                          if (value == 'G') {
                                                 value = UART_RxChar();
                                                 if (value == 'A') {
                                                        value = UART_RxChar();
                                                        if (value == ',') {
                                                               value = UART_RxChar();
                                                               while (value != ',') {
                                                               value = UART_RxChar();
                                                               }
                                                               lati_value[0] =
UART_RxChar();
                                                               value = lati_value[0];
                                                               for (int i = 1; value !=
',' ; i++) {
                                                                      lati_value[i] =
UART_RxChar();
                                                                      value =
lati_value[i];
                                                               }
                                                               lati_dir =
UART_RxChar();
                                                               value = UART_RxChar();
                                                               while (value != ',') {
                                                                      value =
UART_RxChar();
                                                               }
                                                               longi_value[0] =
UART_RxChar();
                                                               value = longi_value[0];
                                                               for(int i = 1; value !=
',' ; i++) {
                                                                      longi_value[i] =
UART_RxChar();
```

```
                                                                                value =
longi_value[i];
                                                                        }
                                                                        longi_dir =
UART_RxChar();
                                                                        flag = 1;
                                                                } else {
                                                                        continue;
                                                                }
                                                        } else {
                                                                continue;
                                                        }
                                                } else {
                                                        continue;
                                                }
                                        } else {
                                                continue;
                                        }
                                } else {
                                        continue;
                                }
                        } else {
                                continue;
                        }
                } else {
                        continue;
                }
        }
}

//give latitude value as a string
char* get_lati_str() {
        return lati_value;   //lati_value;
}

//give longitude value as a string
char* get_longi_str() {
        return longi_value;  //longi_value;
}

//give latitude value as a double
float get_lati_float() {
        for (int i = 0; i < 15; i++) {
                if (lati_value[i] == ',') {
                        lati_value[i] = '0';
                }
        }
        float correct_lati_value = atof(lati_value) / 100;
        return correct_lati_value;
}

//give longitude value as a double
float get_longi_float() {
        for (int i = 0; i < 15; i++) {
                if (longi_value[i] == ',') {
                        longi_value[i] = '0';
                }
        }
        /*how to convert a latitude value to degrees*/
        float correct_longi_value = atof(longi_value) / 100;
        return correct_longi_value;
}
```

```
/* take the inputed GPS coordinate and value and compare with inputted*/
int angle_from_north(float lati_input, float longi_input) {
       float dy = lati_input - get_lati_float();
       float dx = cos(PI / 180 * get_lati_float()) * (longi_input - 
get_longi_float());
       float angle = dy / dx;
       int temp = angle;
       return angle;
}


/*
 * I2C.c
 *
 * Created: 5/10/2022 8:59:24 PM
 *  Author: Hansa Jayathilaka, Upeksha Herath
 */ 

#include "../defines.h"


void I2C_wait_to_process() {
       while ((TWCR & (1 << TWINT)) == 0);
}


/*************************************************************************/
/* Master                                                                */
/*************************************************************************/

void I2C_master_init() {
       TWBR = 0x62;         // Baud rate is set by calculating
       TWCR = (1 << TWEN);  // Enable I2C
       TWSR = 0x00;         // Pre-scaler set to 1
}

//Start condition
void I2C_start() {
       TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTA);      //start condition
       I2C_wait_to_process();
}

//I2C stop condition
void I2C_write(unsigned char x) {
       TWDR = x;                                        //Move value to I2C
       TWCR = (1 << TWINT) | (1 << TWEN); //Enable I2C and clear interrupt
       I2C_wait_to_process();
}

void I2C_select_slave(unsigned char address, int mode) {
       I2C_write(address + mode);
}

void I2C_stop() {
       TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
       // while((TWCR & (1<<TWSTO)) == 1);              // No need to wait
}

void I2C_master_write_buffer(unsigned char address, char* buffer, int length) {
       I2C_start();
       I2C_select_slave(address, WRITE);
	for(char* i = buffer; i < buffer + length; i++) {
		I2C_write(*i);
	}
	I2C_stop();
}

/*************************************************************************/
/* End Master                                                            */
/*************************************************************************/


/*************************************************************************/
/* Slave                                                                 */
/*************************************************************************/

void I2C_slave_init(unsigned char address) {
	TWAR = address;			// Set slave address
}

void I2C_listen(void) {
	TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);// Enable; Interrupt; 
Acknowledgment;
	I2C_wait_to_process();
}

// Read byte with NACK
unsigned char I2C_read() {
	TWCR  = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);	// Enable; Interrupt; 
Acknowledgment;
	I2C_wait_to_process();
	return TWDR;
}

void I2C_slave_read_buffer(char* buffer, int length) {
	I2C_listen();
	for(char* i = buffer; i < buffer + length; i++) {
		*i = I2C_read();
	}
	I2C_listen();
}

/*************************************************************************/
/* End Slave                                                             */
/*************************************************************************/


/*
 * magnetometer.c
 *
 * Created: 5/20/2022 9:53:22 AM
 *  Author: Upeksha Herath
 */

#include "../defines.h"

/* Define declination of location from where measurement going to be done. we can get 
it from http://www.magnetic-declination.com */
#define Declination  -0.00669


void Magneto_init()		/* Magneto initialize function */
{
       I2C_select_slave(0x3C, WRITE);     /* Start and write SLA+W */

       I2C_write(0x00);     /* Write memory location address */
       I2C_write(0x70);     /* Configure register A as 8-average, 15 Hz default, 
normal measurement */
       I2C_write(0xA0);     /* Configure register B for gain */
       I2C_write(0x00);     /* Configure continuous measurement mode in mode register 
*/
       I2C_stop();          /* Stop I2C */
}


int Magneto_GetHeading()
{
       int x, y, z;
       double Heading;
       I2C_start();
       I2C_select_slave(0x3C, WRITE);

       I2C_write(0x3C);     /* Start and wait for acknowledgment */
       I2C_write(0x03);     /* Write memory location address */

       /* Read 16 bit x,y,z value (2's complement form) */
       x = (((int)I2C_read()<<8) | (int)I2C_read());
       z = (((int)I2C_read()<<8) | (int)I2C_read());
       y = (((int)I2C_read()<<8) | (int)I2C_read());
       I2C_stop();          /* Stop I2C */
       Heading = atan2((double)y,(double)x) + Declination;
       if (Heading>2*PI)    /* Due to declination check for >360 degree */
       Heading = Heading - 2*PI;
       if (Heading<0)              /* Check for sign */
       Heading = Heading + 2*PI;
       return (Heading* 180 / PI);/* Convert into angle and return */
}
```

Dissanayake D.M.B.M – 204047J

```
/*
 * joystick.c
 *
 * Created: 5/18/2022 7:53:34 PM
 *  Author: Dasuni Rathnayaka, Binari Dissanayake
 */

#include "../defines.h"


void joystick_init(void) {
       pin_mode(A0, INPUT); // Up / Down
       pin_mode(A1, INPUT); // Left / Right
       pin_mode(A2, INPUT); // Forward / Backward
}

/*
 * Get angle for camera
 */
uint8_t get_joystick_up_down() {
       return ADC_read(A0);
}

/*
 * Get turn
 */
uint8_t get_joystick_left_right() {
       return ADC_read(A1);
}

/*
 * Get forward and backward speed
 */
uint8_t get_joystick_forward_backward() {
       return ADC_read(A2);
}


/* nrf24l01_reg.h
 *
 * Created: 5/12/2022 7:53:26 AM
 *  Author: Binari Dissanayake, Dasuni Rathnayaka
 */
/**
 * Register definitions with bit definitions for the nRF24L01
 *
 */

#ifndef NRF24L01_REG_H
#define NRF24L01_REG_H


/* nRF24L01 Instruction Definitions */
#define WRITE_REG           0x20
#define RD_RX_PLOAD_W   0x60
#define RD_RX_PLOAD         0x61
#define WR_TX_PLOAD         0xA0
#define WR_ACK_PLOAD        0xA8
#define WR_NAC_TX_PLOAD 0xB0
#define FLUSH_TX            0xE1
```

```
#define FLUSH_RX            0xE2
#define REUSE_TX_PL         0xE3
#define LOCK_UNLOCK         0x50
#define NOP                 0xFF


/* nRF24L01 Register address definitions */
#define CONFIG        0x00
#define EN_AA         0x01
#define EN_RXADDR     0x02
#define SETUP_AW      0x03
#define SETUP_RETR    0x04
#define RF_CH         0x05
#define RF_SETUP      0x06
#define STATUS        0x07
#define OBSERVE_TX    0x08
#define CD            0x09
#define RX_ADDR_P0    0x0A
#define RX_ADDR_P1    0x0B
#define RX_ADDR_P2    0x0C
#define RX_ADDR_P3    0x0D
#define RX_ADDR_P4    0x0E
#define RX_ADDR_P5    0x0F
#define TX_ADDR       0x10
#define RX_PW_P0      0x11
#define RX_PW_P1      0x12
#define RX_PW_P2      0x13
#define RX_PW_P3      0x14
#define RX_PW_P4      0x15
#define RX_PW_P5      0x16
#define FIFO_STATUS   0x17
#define DYNPD         0x1C
#define FEATURE       0x1D

/********** Register bit definitions *************/
/* STATUS Reg bits */
#define STAT_MAX_RT         (1 << 4)
#define STAT_TX_DS          (1 << 5)
#define STAT_RX_DR          (1 << 6)
#define STAT_RX_P_NO (7 << 1)
#define STAT_TX_FULL (1 << 0)

/* CONFIG register bits */
#define CONFIG_RX_DR    (1 << 6)
#define CONFIG_TX_DS    (1 << 5)
#define CONFIG_MAX_RT   (1 << 4)
#define CONFIG_EN_CRC   (1 << 3)
#define CONFIG_CRCO     (1 << 2)
#define CONFIG_PWR_UP   (1 << 1)
#define CONFIG_PRIM_RX  (1 << 0)


/* RF_SETUP register bit definitions */
#define RF_CONT_WAVE (1 << 7)
#define RF_DR_LOW           (1 << 5)
#define RF_PLL_LOCK         (1 << 4)
#define RF_DR_HIGH          (1 << 3)
#define RF_PWR1             (1 << 2)
#define RF_PWR0             (1 << 1)
#define RF_LNA              (1 << 0)
```

```
/* FIFO_STATUS register bits */
#define TX_REUSE            (1 << 6)
#define TX_FIFO_FULL        (1 << 5)
#define TX_EMPTY            (1 << 4)
#define RX_FULL             (1 << 1)
#define RX_EMPTY            (1 << 0)


/* Operation mode */
typedef enum {
    NRF_MODE_PTX = 0,
    NRF_MODE_PRX
} nrf_opmode_t;


/* Output power modes */
typedef enum {
    NRF_PWR_18DBM = 0,
    NRF_PWR_12DBM,
    NRF_PWR_6DBM,
    NRF_PWR_0DBM
} nrf_power_t;

/* data rate */
typedef enum {
       NRF_RATE_250KBPS = 0,
    NRF_RATE_1MBPS,
    NRF_RATE_2MBPS
} nrf_datarate_t;


/* pipe numbers */
typedef enum {
    NRF_PIPE0 = 0,
    NRF_PIPE1,
    NRF_PIPE2,
    NRF_PIPE3,
    NRF_PIPE4,
    NRF_PIPE5,
       NRF_TX_PIPE,
       NRF_TX_PLOAD, /* for writing tx payload */
       NRF_TX_PLOAD_NOACK, /* for tx payload with no ACK */
       NRF_RX_PLOAD, /* for reading rx payload */
    NRF_PIPE_ALL = 0xFF
} nrf_pipe_t;


#endif  // NRF24L01_REG_H
/*
 *  NRF24L01+ library header file
 *  nrf24101.h
 *
 * Created: 5/11/2022 6:53:26 PM
 *  Author: Binari Dissanayake, Dasuni Rathnayaka
 *
 *
 */

#include "../../defines.h"

#ifndef NRF24L01_H
#define NRF24L01_H
```

```
/* Public functions */


void nrf_init(nrf_opmode_t mode, uint8_t *address);
uint8_t nrf_transmit_packet(uint8_t *packet, uint8_t length);
uint8_t nrf_receive_packet(uint8_t *buf, uint8_t *length);
void nrf_set_ack_payload(uint8_t pipe, uint8_t *buf, uint8_t length);
void nrf_tx_data(uint8_t up_down, uint8_t left_right, uint8_t forward_backward,
uint8_t siren, uint8_t auto_manual_mode);
int nrf_rx_data(uint8_t *up_down, uint8_t *left_right, uint8_t *forward_backward,
uint8_t *siren, uint8_t *auto_manual_mode);


#endif  // NRF24L01_H



/*
 * ultrasonic.c
 *
 * Created: 11/1/2021 10:49:53 PM
 *  Author: Binari Dissanayake
 */

#include "../defines.h"

#define trigPin C6
#define echoPin D6

#define mux1 C4
#define mux2 C5

int TimerOverflow = 0;


void ultrazonic_init() {
        pin_mode(trigPin, OUTPUT);
        pin_mode(echoPin, INPUT_PULLUP);

        pin_mode(mux1, OUTPUT);
        pin_mode(mux2, OUTPUT);
}


double ultrazonic_distance(void) {
        char string[10];
        long count;
        double distance;

        uint8_t _TCCR1B = TCCR1B;
        uint8_t _TCCR1A = TCCR1A;
        uint8_t _TIMSK = TIMSK;

        sei();               /* Enable global interrupt */
        TIMSK = (1 << TOIE1);       /* Enable Timer1 overflow interrupts */
        TCCR1A = 0;          /* Set all bit to zero Normal operation */

        /* Give 10us trigger pulse on trig. pin to HC-SR04 */
        digital_write(C6, HIGH);
        _delay_us(10);
        digital_write(C6, LOW);
```

```
       TCNT1 = 0;    /* Clear Timer counter */
       TCCR1B = 0x41;       /* Capture on rising edge, No prescaler*/
       TIFR = 1<<ICF1;      /* Clear ICP flag (Input Capture flag) */
       TIFR = 1<<TOV1;      /* Clear Timer Overflow flag */

       /*Calculate width of Echo by Input Capture (ICP) */

       while ((TIFR & (1 << ICF1)) == 0);/* Wait for rising edge */
       TCNT1 = 0;    /* Clear Timer counter */
       TCCR1B = 0x01;       /* Capture on falling edge, No prescaler */
       TIFR = 1<<ICF1;      /* Clear ICP flag (Input Capture flag) */
       TIFR = 1<<TOV1;      /* Clear Timer Overflow flag */
       TimerOverflow = 0;/* Clear Timer overflow count */

       while ((TIFR & (1 << ICF1)) == 0);/* Wait for falling edge */
       count = ICR1 + (65535 * TimerOverflow);   /* Take count */
       /* 8MHz Timer freq, sound speed =343 m/s */
       distance = (double)count / 466.47;

       TCCR1B = _TCCR1B;
       TCCR1A = _TCCR1A;
       TIMSK = _TIMSK;
       // PWM_init();
       return distance;
}


int ultrazonic_error() {
       digital_write(mux1, LOW);
       digital_write(mux2, LOW);
       double left = ultrazonic_distance();

       digital_write(mux1, LOW);
       digital_write(mux2, HIGH);
       double right = ultrazonic_distance();

       char string[16];
       LCD_clear();
       LCD_line_1();

       dtostrf(left, 2, 2, string);/* distance to string */
       strcat(string, " cm   ");   /* Concat unit i.e.cm */
       LCD_msg(string);

       dtostrf(right, 2, 2, string);/* distance to string */
       strcat(string, " cm");      /* Concat unit i.e.cm */
       LCD_line_2();
       LCD_msg(string);

       digital_write(mux1, LOW);
       digital_write(mux2, LOW);

       int error = right - left;

       return error;
}


ISR(TIMER1_OVF_vect)
{ TimerOverflow++;   /* Increment Timer Overflow count */ }
```

Rathnayaka A.M.D.B – 204179N

```
/*
 * joystick.c
 *
 * Created: 5/18/2022 7:53:34 PM
 *  Author: Dasuni Rathnayaka, Binari Dissanayake
 */

#include "../defines.h"


void joystick_init(void) {
       pin_mode(A0, INPUT); // Up / Down
       pin_mode(A1, INPUT); // Left / Right
       pin_mode(A2, INPUT); // Forward / Backward
}

/*
 * Get angle for camera
 */
uint8_t get_joystick_up_down() {
       return ADC_read(A0);
}

/*
 * Get turn
 */
uint8_t get_joystick_left_right() {
       return ADC_read(A1);
}

/*
 * Get forward and backward speed
 */
uint8_t get_joystick_forward_backward() {
       return ADC_read(A2);
}



/* nrf24101_reg.h
 *
 * Created: 5/12/2022 7:53:26 AM
 *  Author: Binari Dissanayake, Dasuni Rathnayaka
 */
/**
 * Register definitions with bit definitions for the nRF24L01
 *
 */

#ifndef NRF24L01_REG_H
#define NRF24L01_REG_H


/* nRF24L01 Instruction Definitions */
#define WRITE_REG           0x20
#define RD_RX_PLOAD_W    0x60
#define RD_RX_PLOAD         0x61
#define WR_TX_PLOAD         0xA0
#define WR_ACK_PLOAD        0xA8
```

```
#define WR_NAC_TX_PLOAD 0xB0
#define FLUSH_TX            0xE1
#define FLUSH_RX            0xE2
#define REUSE_TX_PL         0xE3
#define LOCK_UNLOCK         0x50
#define NOP                 0xFF


/* nRF24L01 Register address definitions */
#define CONFIG        0x00
#define EN_AA         0x01
#define EN_RXADDR     0x02
#define SETUP_AW      0x03
#define SETUP_RETR    0x04
#define RF_CH         0x05
#define RF_SETUP      0x06
#define STATUS        0x07
#define OBSERVE_TX    0x08
#define CD            0x09
#define RX_ADDR_P0    0x0A
#define RX_ADDR_P1    0x0B
#define RX_ADDR_P2    0x0C
#define RX_ADDR_P3    0x0D
#define RX_ADDR_P4    0x0E
#define RX_ADDR_P5    0x0F
#define TX_ADDR       0x10
#define RX_PW_P0      0x11
#define RX_PW_P1      0x12
#define RX_PW_P2      0x13
#define RX_PW_P3      0x14
#define RX_PW_P4      0x15
#define RX_PW_P5      0x16
#define FIFO_STATUS   0x17
#define DYNPD         0x1C
#define FEATURE       0x1D

/********** Register bit definitions **************/
/* STATUS Reg bits */
#define STAT_MAX_RT         (1 << 4)
#define STAT_TX_DS          (1 << 5)
#define STAT_RX_DR          (1 << 6)
#define STAT_RX_P_NO (7 << 1)
#define STAT_TX_FULL (1 << 0)

/* CONFIG register bits */
#define CONFIG_RX_DR    (1 << 6)
#define CONFIG_TX_DS    (1 << 5)
#define CONFIG_MAX_RT   (1 << 4)
#define CONFIG_EN_CRC   (1 << 3)
#define CONFIG_CRCO     (1 << 2)
#define CONFIG_PWR_UP   (1 << 1)
#define CONFIG_PRIM_RX  (1 << 0)


/* RF_SETUP register bit definitions */
#define RF_CONT_WAVE (1 << 7)
#define RF_DR_LOW           (1 << 5)
#define RF_PLL_LOCK         (1 << 4)
#define RF_DR_HIGH          (1 << 3)
#define RF_PWR1             (1 << 2)
#define RF_PWR0             (1 << 1)
#define RF_LNA              (1 << 0)
```

```
/* FIFO_STATUS register bits */
#define TX_REUSE            (1 << 6)
#define TX_FIFO_FULL        (1 << 5)
#define TX_EMPTY            (1 << 4)
#define RX_FULL             (1 << 1)
#define RX_EMPTY            (1 << 0)


/* Operation mode */
typedef enum {
    NRF_MODE_PTX = 0,
    NRF_MODE_PRX
} nrf_opmode_t;


/* Output power modes */
typedef enum {
    NRF_PWR_18DBM = 0,
    NRF_PWR_12DBM,
    NRF_PWR_6DBM,
    NRF_PWR_0DBM
} nrf_power_t;

/* data rate */
typedef enum {
       NRF_RATE_250KBPS = 0,
    NRF_RATE_1MBPS,
    NRF_RATE_2MBPS
} nrf_datarate_t;


/* pipe numbers */
typedef enum {
    NRF_PIPE0 = 0,
    NRF_PIPE1,
    NRF_PIPE2,
    NRF_PIPE3,
    NRF_PIPE4,
    NRF_PIPE5,
       NRF_TX_PIPE,
       NRF_TX_PLOAD, /* for writing tx payload */
       NRF_TX_PLOAD_NOACK, /* for tx payload with no ACK */
       NRF_RX_PLOAD, /* for reading rx payload */
    NRF_PIPE_ALL = 0xFF
} nrf_pipe_t;


#endif  // NRF24L01_REG_H
/*
 *  NRF24L01+ library header file
 *  nrf24101.h
 *
 * Created: 5/11/2022 6:53:26 PM
 *  Author: Binari Dissanayake, Dasuni Rathnayaka
 *
 *
 */

#include "../../defines.h"
```

```
#ifndef NRF24L01_H
#define NRF24L01_H


/* Public functions */


void nrf_init(nrf_opmode_t mode, uint8_t *address);
uint8_t nrf_transmit_packet(uint8_t *packet, uint8_t length);
uint8_t nrf_receive_packet(uint8_t *buf, uint8_t *length);
void nrf_set_ack_payload(uint8_t pipe, uint8_t *buf, uint8_t length);
void nrf_tx_data(uint8_t up_down, uint8_t left_right, uint8_t forward_backward,
uint8_t siren, uint8_t auto_manual_mode);
int nrf_rx_data(uint8_t *up_down, uint8_t *left_right, uint8_t *forward_backward,
uint8_t *siren, uint8_t *auto_manual_mode);


#endif  // NRF24L01_H



/*
 * keypad.c
 *
 * Created: 12/4/2021 3:50:40 PM
 *  Author: Dasuni Rathnayaka
 */

#include "../defines.h"

#define KEY_PRT      PORTA
#define KEY_DDR             DDRA
#define KEY_PIN             PINA

unsigned char keypad[4][4] = {
       {'7','4','1',' '},
       {'8','5','2','0'},
       {'9','6','3','='},
       {'/','*','-','+'},
};

unsigned char colloc, rowloc;


char key_char() {
       while(1) {
              KEY_DDR = 0xF0;           /* set port direction as input-output */
              KEY_PRT = 0xFF;

              do
              {
                     KEY_PRT &= 0x0F;      /* mask PORT for column read only */
                     asm("NOP");
                     colloc = (KEY_PIN & 0x0F); /* read status of column */
              } while(colloc != 0x0F);

              do
              {
                     do
                     {
                            _delay_ms(20);             /* 20ms key debounce time */
                            colloc = (KEY_PIN & 0x0F); /* read status of column */
```

```
                        } while(colloc == 0x0F);        /* check for any key press */

                        _delay_ms (40);                   /* 20 ms key debounce time */
                        colloc = (KEY_PIN & 0x0F);
                } while(colloc == 0x0F);

                /* now check for rows */
                KEY_PRT = 0xEF;            /* check for pressed key in 1st row */
                asm("NOP");
                colloc = (KEY_PIN & 0x0F);
                if(colloc != 0x0F)
                {
                        rowloc = 0;
                        break;
                }

                KEY_PRT = 0xDF;             /* check for pressed key in 2nd row */
                asm("NOP");
                colloc = (KEY_PIN & 0x0F);
                if(colloc != 0x0F)
                {
                        rowloc = 1;
                        break;
                }

                KEY_PRT = 0xBF;             /* check for pressed key in 3rd row */
                asm("NOP");
                colloc = (KEY_PIN & 0x0F);
                if(colloc != 0x0F)
                {
                        rowloc = 2;
                        break;
                }

                KEY_PRT = 0x7F;             /* check for pressed key in 4th row */
                asm("NOP");
                colloc = (KEY_PIN & 0x0F);
                if(colloc != 0x0F)
                {
                        rowloc = 3;
                        break;
                }
        }

        if(colloc == 0x0E)
                return(keypad[rowloc][0]);
        else if(colloc == 0x0D)
                return(keypad[rowloc][1]);
        else if(colloc == 0x0B)
                return(keypad[rowloc][2]);
        else
                return(keypad[rowloc][3]);
}


void key_string(char buffer[], int buff) {
        UART_TxChar('\n');
        for(int i = 0; i < buff; i++) {
                char temp = key_char();
                LCD_char(temp);
                UART_TxChar(temp);
                if (temp == '=') {
```

```
            buffer[i] = '\0';
            break;
        }
        buffer[i] = temp;
    }
}
```

Pathirana S.P.S.N – 204150T

```
/*
 * PWM.c
 *
 * Created: 11/1/2021 10:09:46 PM
 *  Author: Pathirana S.P.S.N
 */

#include "../defines.h"


/*
 * Initialize PWN settings
 *
 * Parameter
 *     - None
 * Return
 *     - None
 */
void PWM_init(void) {
       // Force compare match
       TCCR0 = (1 << FOC0);
       // Timer0 mode selection bit; Normal, CTC, PWM-Phase correct, [Fast PWM]
       TCCR0 |= (1 << WGM00) | (1 << WGM01);
       // Output Mode; Disconnected, Reserved, [Non-inverted], Inverted (This is only
for Fast PWM)
       TCCR0 |= (1 << COM01) | (0 << COM00);
       // Clock Source Select; no pre-scaling
       TCCR0 |= (0 << CS02) | (1 << CS01) | (1 << CS00);
       // Enable Overflow interrupt
       TIMSK |= (1 << TOIE0);

       TCCR1A = (1 << COM1A1) | (0 << COM1A0); // A - Compare match mode, Non-Inverted
Mode.
       TCCR1A |= (1 << COM1B1) | (0 << COM1B0); // B - Compare match mode, Non-
Inverted Mode.
       TCCR1A |= (1 << WGM11) | (0 << WGM10); // Fast PWM mode
       TCCR1B = (1 << WGM13);
       TCCR1B |= (1 << WGM12); // Compare mode
       TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10); // No pre-scaler
       ICR1 = 255;

       // Force compare match
       TCCR2 = (1 << FOC2);
       // Timer0 mode selection bit; Normal, CTC, PWM-Phase correct, [Fast PWM]
       TCCR2 |= (1 << WGM20) | (1 << WGM21);
       // Output Mode; Disconnected, Reserved, [Non-inverted], Inverted (This is only
for Fast PWM)
       TCCR2 |= (1 << COM21) | (0 << COM20);
       // Clock Source Select; no pre-scaling
       TCCR2 |= (0 << CS22) | (1 << CS21) | (1 << CS20);
}


/*
 * Analog output from pin
 *
 * Parameter
 *     - pin (string) - Input pin eg: B3, D4, D5, D7
 *     - level (int) - Value between 0 - 255
 * Return
 *     - (int) - 0 if no errors.
 */
int PWM_write(Pin pin, int dutyCyle) {
       if (pin.port == 'B' && pin.pin == 3) {
              OCR0 = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 4) {
              OCR1B = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 5) {
              OCR1A = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 7) {
              OCR2 = dutyCyle;
              } else {
              return -1;
       }
       return 0;
}


/*
 * Analog output from register
 *
 * Parameter
 *     - regi (string) - Register eg: OCR0, OCR1B, OCR1A, OCR2
 *     - level (int) - Value between 0 - 255
 * Return
 *     - (int) - 0 if no errors.
 */
int PWM_write_reg(void *regi, int dutyCyle) {
       volatile uint8_t *_regi = regi;
       *_regi = dutyCyle;
       return 0;
}


/*
 * servo.c
 *
 * Created: 5/21/2022 8:55:11 PM
 *  Author: Sadini Pathirana
 */

#include "../defines.h"


/*
 * -90 - 14
 *  90 - 31
 */

void servo_init() {
       pin_mode(B3, OUTPUT);
}


void servo_write(int angle) {
       float val = (31 - 14) / 255 * angle + 14;
       PWM_write(B3, val);
}
```

//siren

```
// Start siren when remote siren button pressed
       if (siren == 1)
              digital_write(B0, HIGH);
       else
              digital_write(B0, LOW);

/*
 * Digital output
 *
 * Parameter
 *      - pin (string) - Input pin eg: A1, B4
 *      - level (int) - 1 for high value, 0 for low value
 * Return
 *      - (int) - 0 if no errors.
 */
int digital_write(Pin pin, int level) {
       volatile uint8_t *regi = select_register(pin.port, &PORTA, &PORTB, &PORTC,
&PORTD); // Select PORT register according to pin

       if (level == 1) { // Check weather high or low
              *regi |= 1 << pin.pin; // Output high value
       } else {
              *regi &= ~(1 << pin.pin); // Output low value
       }
       return 0;
}
```

Jayathilaka P.H.P – 204087F

```c
/*
 * PWM.c
 *
 * Created: 11/1/2021 10:09:46 PM
 *  Author: Hansa Jayathilaka
 */

#include "../defines.h"


/*
 * Initialize PWN settings
 *
 * Parameter
 *      - None
 * Return
 *      - None
 */
void PWM_init(void) {
	// Force compare match
	TCCR0 = (1 << FOC0);
	// Timer0 mode selection bit; Normal, CTC, PWM-Phase correct, [Fast PWM]
	TCCR0 |= (1 << WGM00) | (1 << WGM01);
	// Output Mode; Disconnected, Reserved, [Non-inverted], Inverted (This is only for Fast PWM)
	TCCR0 |= (1 << COM01) | (0 << COM00);
	// Clock Source Select; no pre-scaling
	TCCR0 |= (0 << CS02) | (1 << CS01) | (1 << CS00);
	// Enable Overflow interrupt
	TIMSK |= (1 << TOIE0);

	TCCR1A = (1 << COM1A1) | (0 << COM1A0); // A - Compare match mode, Non-Inverted Mode.
	TCCR1A |= (1 << COM1B1) | (0 << COM1B0); // B - Compare match mode, Non-Inverted Mode.
	TCCR1A |= (1 << WGM11) | (0 << WGM10); // Fast PWM mode
	TCCR1B = (1 << WGM13);
	TCCR1B |= (1 << WGM12); // Compare mode
	TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10); // No pre-scaler
	ICR1 = 255;

	// Force compare match
	TCCR2 = (1 << FOC2);
	// Timer0 mode selection bit; Normal, CTC, PWM-Phase correct, [Fast PWM]
	TCCR2 |= (1 << WGM20) | (1 << WGM21);
	// Output Mode; Disconnected, Reserved, [Non-inverted], Inverted (This is only for Fast PWM)
	TCCR2 |= (1 << COM21) | (0 << COM20);
	// Clock Source Select; no pre-scaling
	TCCR2 |= (0 << CS22) | (1 << CS21) | (1 << CS20);
}


/*
 * Analog output from pin
 *
 * Parameter
 *      - pin (string) - Input pin eg: B3, D4, D5, D7
 *      - level (int) - Value between 0 - 255
 * Return
 *      - (int) - 0 if no errors.
 */
int PWM_write(Pin pin, int dutyCyle) {
       if (pin.port == 'B' && pin.pin == 3) {
              OCR0 = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 4) {
              OCR1B = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 5) {
              OCR1A = dutyCyle;
              } else if (pin.port == 'D' && pin.pin == 7) {
              OCR2 = dutyCyle;
              } else {
              return -1;
       }
       return 0;
}


/*
 * Analog output from register
 *
 * Parameter
 *      - regi (string) - Register eg: OCR0, OCR1B, OCR1A, OCR2
 *      - level (int) - Value between 0 - 255
 * Return
 *      - (int) - 0 if no errors.
 */
int PWM_write_reg(void *regi, int dutyCyle) {
       volatile uint8_t *_regi = regi;
       *_regi = dutyCyle;
       return 0;
}


/*
 * motor.c
 *
 * Created: 11/30/2021 4:54:44 PM
 *  Author: Hansa Jayathilaka
 */

#include "../defines.h"

#define PWM0A   D4    // Left Forward
#define PWM0B   D5   // Right Forward
#define DIRA    C3    // Left Backward
#define DIRB    C7   // Right Backward

void motor_init() {
       pin_mode(PWM0A, OUTPUT);
       pin_mode(PWM0B, OUTPUT);
       pin_mode(DIRA, OUTPUT);
       pin_mode(DIRB, OUTPUT);
}

void setM2Speed(int speed) {
       unsigned char reverse = 0;

       if (speed < 0) {
              speed = -speed; // make speed a positive quantity
              reverse = 1;    // preserve the direction
       }
```

```
        if (speed > 0xFF)
        speed = 0xFF;

        if (reverse) {
                digital_write(DIRB, HIGH);
                PWM_write(PWM0B, 0xFF - speed);
        }
        else { // forward
                digital_write(DIRB, LOW);
                PWM_write(PWM0B, speed);
        }
}

void setM1Speed(int speed) {
        unsigned char reverse = 0;

        if (speed < 0) {
                speed = -speed; // make speed a positive quantity
                reverse = 1;    // preserve the direction
        }

        if (speed > 0xFF)
        speed = 0xFF;

        if (reverse) {
                digital_write(DIRA, HIGH);
                PWM_write(PWM0A, 0xFF - speed);
        }
        else { // forward
                digital_write(DIRA, LOW);
                PWM_write(PWM0A, speed);
        }
}

void drive(int m1Speed, int m2Speed) {
        setM1Speed(m1Speed);
        setM2Speed(m2Speed);
}


/*
 * display.c
 *
 * Created: 10/27/2021 4:00:14 PM
 *  Author: Hansa Jayathilaka
 */

#include "../defines.h"

#define LCD_ADDRESS 0x70


void toggle() {
        I2C_write(TWDR | 0x02);                  // Set enable pin 1;  Latching data in
to LCD data register using High to Low signal
        I2C_write(TWDR & ~0x02);          // Set enable pin 0;
}

void LCD_cmd_hf(char val) {
        I2C_write(TWDR & ~0x01);          // Set RS pin to 0; Selecting register as
Command register
```

```
       I2C_write(TWDR & 0x0F);                   // Clearing the Higher 4 bits
       I2C_write(TWDR | (val & 0xF0));    //----Masking higher 4 bits and sending to 
LCD
       toggle();
}

void LCD_cmd(char val) {
       I2C_write(TWDR & ~0x01);           //rs = 0; ----Selecting register as command 
register
       I2C_write(TWDR & 0x0F);                   //----clearing the Higher 4 bits
       I2C_write(TWDR | (val & 0xF0));    //----Masking higher 4 bits and sending to 
LCD
       toggle();

       I2C_write(TWDR & 0x0F);                                 //----clearing the 
Higher 4 bits
       I2C_write(TWDR | ((val & 0x0F) << 4));    //----Masking lower 4 bits and sending 
to LCD
       toggle();
}

void LCD_dwr(char val) {
       I2C_write(TWDR | 0x01);                                 //rs = 1; ----Selecting 
register as command register
       I2C_write(TWDR & 0x0F);                              //----clearing the Higher 4 
bits
       I2C_write(TWDR | (val & 0xF0));                  //----Masking higher 4 bits and 
sending to LCD
       toggle();

       I2C_write(TWDR & 0x0F);                                 //----clearing the 
Higher 4 bits
       I2C_write(TWDR | ((val & 0x0F) << 4));    //----Masking lower 4 bits and sending 
to LCD
       toggle();
}

void LCD_init() {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       LCD_cmd_hf(0x30);       //-----Sequence for initializing LCD
       LCD_cmd_hf(0x30);       //-----     "            "              "         
"
       LCD_cmd_hf(0x20);       //-----     "            "              "         
"
       LCD_cmd(0x28);          //-----Selecting 16 x 2 LCD in 4Bit mode
       LCD_cmd(0x0C);          //-----Display ON Cursor OFF
       LCD_cmd(0x01);          //-----Clear display
       LCD_cmd(0x06);          //-----Cursor Auto Increment
       LCD_cmd(0x80);          //-----1st line 1st location of LCD

       I2C_stop();
}

void delay(int ms) {
       int i,j;
       for(i=0;i<=ms;i++)
       for(j=0;j<=120;j++);
}

void LCD_msg(char *c) {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       while(*c != 0)      //----Wait till all String are passed to LCD
       LCD_dwr(*c++);             //----Send the String to LCD

       I2C_stop();
}

void LCD_rig_sh() {
       LCD_cmd(0x1C);      //----Command for right Shift
       delay(400);
}

void LCD_lef_sh() {
       LCD_cmd(0x18);      //----Command for Left Shift
       delay(200);
}

void LCD_clear_msg(char* c) {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       LCD_cmd(0x01);
       LCD_cmd(0x80);

       while(*c != 0)      //----Wait till all String are passed to LCD
              LCD_dwr(*c++);      //----Send the String to LCD

       I2C_stop();
}

void LCD_clear() {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       LCD_cmd(0x01);
       LCD_cmd(0x80);

       I2C_stop();
}

void LCD_line_1() {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       LCD_cmd(0x80);

       I2C_stop();
}

void LCD_line_2() {
       I2C_start();
       I2C_select_slave(LCD_ADDRESS, WRITE);

       LCD_cmd(0xC0);

       I2C_stop();
}
```