**IN1900 - Hardware Project**

**Batch 20 – IT**

**Group 47**

**2021 - 2022**

**Individual Coding Parts**

**Contents**

[Herath P.A.U.D – 204074M 2](#_Toc105579876)

[Dissanayake D.M.B.M – 204047J 8](#_Toc105579877)

[Rathnayaka A.M.D.B – 204179N 13](#_Toc105579878)

[Pathirana S.P.S.N – 204150T 19](#_Toc105579879)

[Jayathilaka P.H.P – 204087F 22](#_Toc105579880)

# 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);

}

/\* 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

/\*

\* 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

/\*

\* 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();

}