# nPi

## Final Design Report

M. Hunter Longshore
EEL4665/5666 IMDL

Instructors:
Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

TAs:
Mike Pridgen
Ryan Stevens
Tim Martin
Thomas Vermeer
Devin Hughes

## Table of Contents

## ABSTRACT

This document provides an overview of the design and preliminary development of a multi-agent autonomous robotic system. The context of the project is presented as well its relationship with the presented design methodology. A summary of the key functional components and the design process is also included.

## INTRODUCTION

As the scope of applications that autonomous systems are being routinely designed for broadens, so too does the autonomous system's ability to record and process information about its environment. The abstraction of a designed solution as a system removes the burden of the necessary computational capacity from a single processing unit and allows it to be distributed over multiple units that can operate cooperatively. This report outlines the design and implementation of a development platform for investigating concepts within a networked processing system, with an emphasis on developing autonomous and cooperative functionalities.

## INTEGRATED SYSTEM

Fault-tolerant operation implemented through the repetition and redundant placement of critical components within a network domain is one characteristic of networks that translates readily into the development of robust robotic systems. This concept is exemplified in the class of networked systems called swarm robotics where a redundant array of nodes is configured to perform tasks as a cooperative effort. The nature of the system puts less emphasis on the role of a single node, instead distributing the effort required to complete the task across the capacity of the network.

The networked system detailed in this report is comprised of 3 small-scale mobile nodes and a PC based host node. This node scheme was chosen to best match a likely application scenario where a base station would be used as a point of deployment and control and/or monitoring of other mobile units. In an effort to provide a more thorough introduction into the characteristics of this basic topology, networks not fitting this definition will be considered beyond the scope of the project and are reserved for future investigation.

Each mobile node implements an identical electrical and mechanical design. The electronics platform includes a 2.4GHz wireless transceiver interfaced with a programmable processing platform and a suite of 16 sensors to allow for the flexible development of node-localized and network-based applications and behaviors. The mechanical platform is a minimalistic design that reduces the complexity of node construction as well as providing lower cost of replication in high node count networks. The inherent efficiency of this implementation is that, ideally, a solution for one node will be applicable to the remaining nodes, reducing the time needed for development. The replication of hardware and software platforms also allows for the abstraction of node functionalities when considering the functional interaction of nodes within the network. A breakdown of the system hierarchy is provided in *figure 1*.

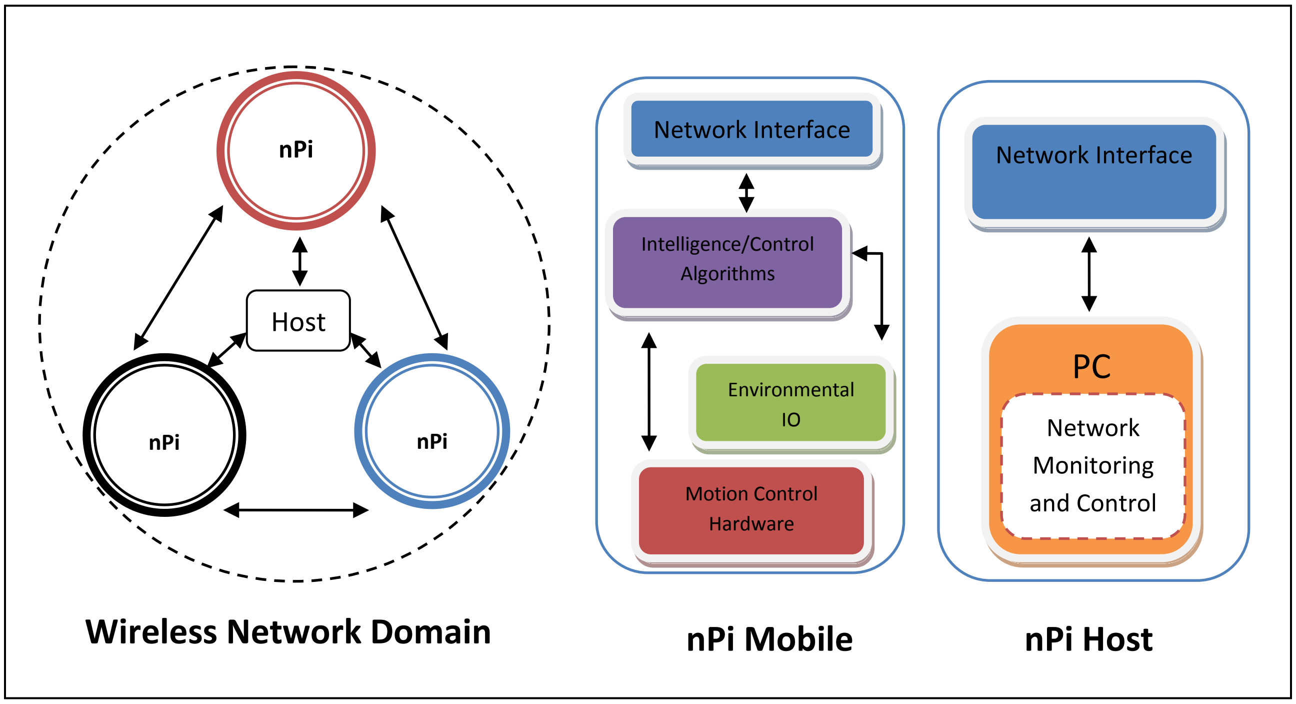

**Figure 1: System Hierarchy**

In order to allow unrestricted transfer of data between the nodes of the network, a multicast network is established where each node is able to communicate directly with every other node. This allows for flexibility in the distribution of network commands that is independent of the state of any individual node.

## MOBILE PLATFORM

The design of the nPi platform is based upon a circular geometry that implies a unidirectional orientation. The platform consists of a modular stack of circular plates, 3.14” in diameter (*figure 2*). A symmetrical mounting-hole pattern is cut into each of the stack layers to allow flexibility in the ordering of functional layers. It is conceived that new modules can be introduced into the system simply by copying the mounting pattern onto the newly designated layer, and adding the stack to the existing platform.

The base for the platform is constructed of a model aircraft grade, wooden laminate. This choice in material provides an optimal balance between manufacturability; where readily available tools can be used to make modifications, durability; as the cross layering of the laminate layers creates an extremely strong structure, and affordability; as the price of the material presents a total material cost for several nPi platforms of less than 20$.

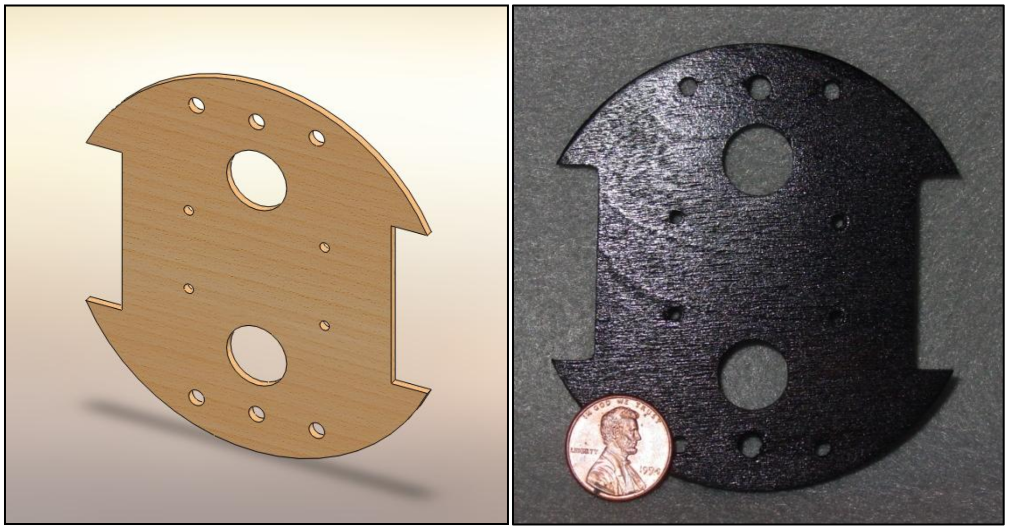

**Figure 2: nPi Modular Platform Plate**

Electrical hardware layers are designed with the same bolting pattern and circular outline. This imposes a restriction on the PCB layout where surface area must be carefully managed, especially for the 7.75 sq. in. surface of each determined by the platform geometry, which will require extensive use of top and bottom board surfaces.

The ground clearance of the nPi chassis is relatively low and relies on low-friction polymer pins to support the longitudinal axis while moving. This method was determined to be suitable for the nPi where the primary surfaces of operation will be smooth, polished areas such as tabletops, linoleum, or wood flooring.

## ACTUATION

The drivetrain design implements a two-wheel differential drive system which was chosen for ease of control and its mechanical simplicity (figure 3). Each wheel is directly coupled to a micro gearhead motor with 50:1 output ratio. The gearhead and motor assembly is rated for 6V operation producing 15 oz.in. of torque and a free-run speed of 650 rpm. Testing was performed to determine the free-run current power consumption of the motor which is presented in *figure 4*. Later current testing under load produced current consumption of 100-150% of the unloaded values, indicating that weight restrictions of the design aid in minimizing power consumption by the drivetrain.

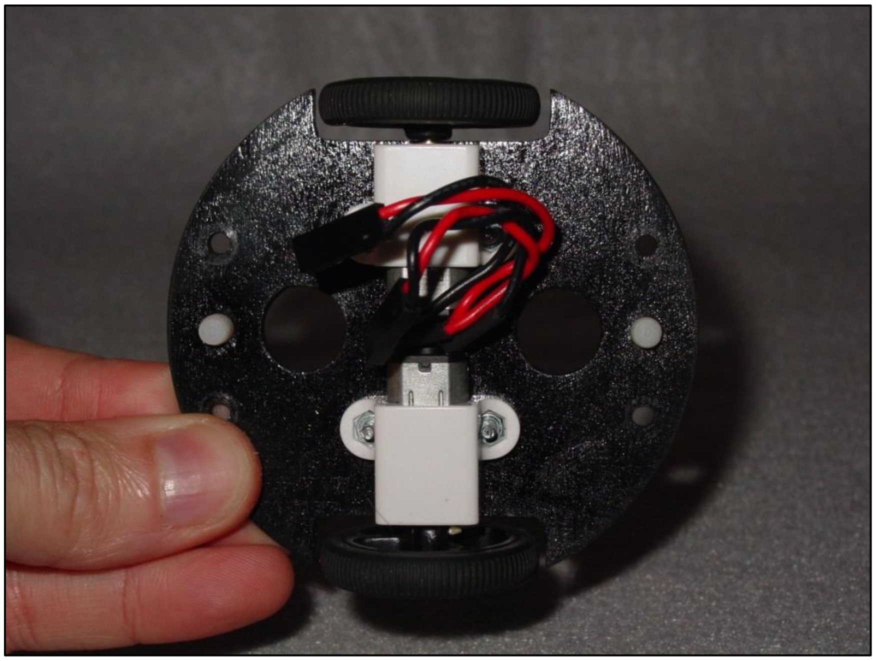

Figure 3: Mounted Differential Drive System Motors and Wheels

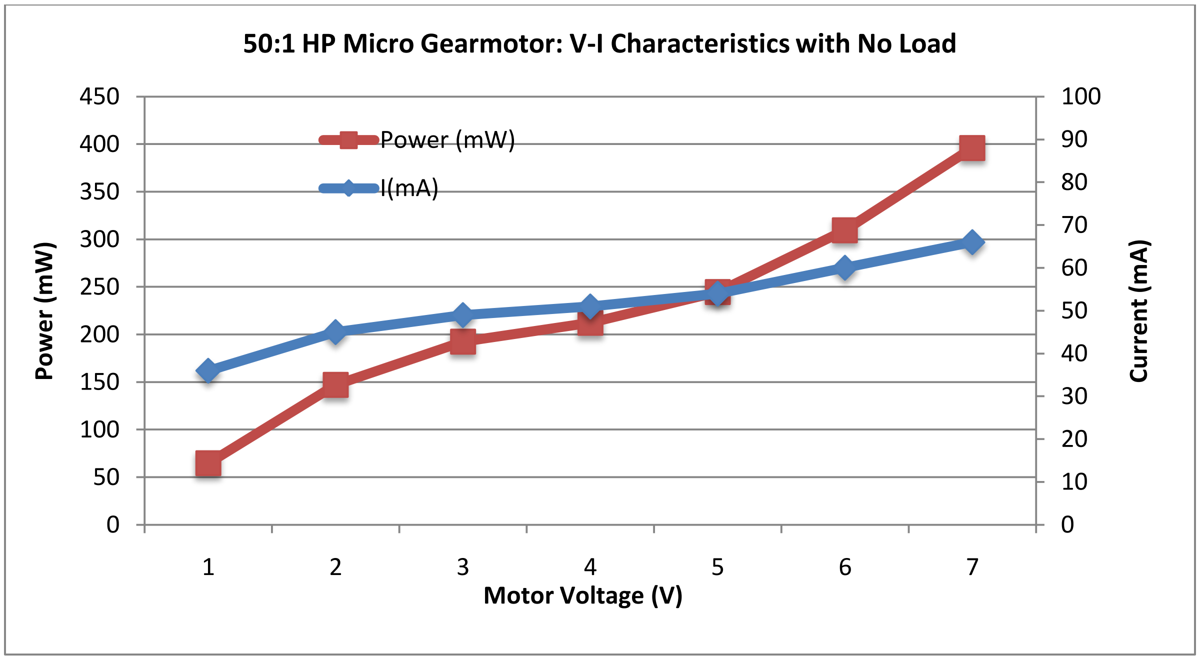

Figure 4: Drive motor power characteristics

Attached to the axle of each motor is a 32mm x 6mm polymer wheel with ribbed rubber tire (figure 3) to reduce weight as well as providing traction over a wide variety of surfaces. Each wheel couples to the gear-motor axle with a compression fitting, reducing the complexity of the assembly and providing a tool free method of attachment and detachment.

Driving the motors is a Toshiba TB6612FNG dual H-Bridge DC motor driver assembled on a Pololu Robotics breakout board with integrated bypass capacitor and power supply regulator (figure 5).

**Figure 5: Toshiba TB6612FNG Dual H-Bridge (Image source: www.Pololu.com)**

The H-bridge driver handles the drive current for both drive train motors via a 2-bit control port and PWM port for each channel. The driver specifies a 2A -peak current limit that supplies the full range tested currents for the motors. Current sensing was considered as an option to avoid the possibility of damaging the drive train through the detection of stall conditions. However, the functionality was not implemented as it does not contribute directly to the core functionality of the node. The driver also incorporates a disable pin to provide a failsafe method of turning off the motors and will be enabled extensively in the control software to extend battery life through reduced idle power consumption.

## SENSORS

The design implements a total of 16 sensors which are used to provide environmental feedback to the control module for each node. The sensor suite includes:

- (6) Sharp GP 2Y0D810Z0F 100mm Reflective IR sensors
- (6) Vishay TEPT4400 Phototransistors
- (1) HMC6352 Compass (Sparkfun breakout)
- (1) MAXBotix EZ3 Sonar
- (1) Microchip MCP9700 IC thermistor
- (1) Nordic nRF24l01+ 2.4GHz transceiver with received signal strength measurement

## Sharp GP2Y0D810Z0F

The array of six (6) Sharp GP2Y0D810Z0F 100mm IR distance detection sensors are used for determining the presence of IR reflective objects within a 100mm (short-range) peripheral distance. The array is sequenced via a 1:8 FET MUX and read in via two 3-input OR gates connected to the digital IO the processor. This scheme enables the detection of objects within each sensors field of view for estimating the position of objects and subsequent obstacle avoidance and path planning.

## Vishay TEPT4400 Phototransistors

6 phototransistors are incorporated in the design. Two are placed on the far-right and far-left sides of the top PCB to indicate emission of light oriented normal to the travel plane, while the remaining four point forward and rearward of the node to detect light oriented parallel with the travel plane. The phototransistors are placed in the common-collector configuration of figure 6. A series resistor of 22kOhms was determined to provide the best range of ambient light to darkness contrast detection with close exposure to a high-intensity LED flashlight providing near saturation (3.28V) and complete darkness providing a signal just above 0V.

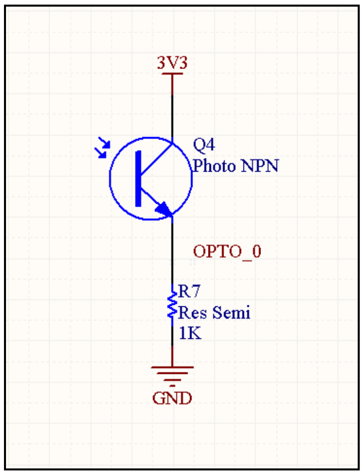

**Figure 6: Common-Collector Phototransistor**

The phototransistors are sampled using the 10-bit ADC of the processor where an iterative loop is used to determine the highest voltage AD port and the subsequent phototransistor with the highest light reading. This data is then fed into the motion controller to determine generate a directional command to move towards the higher intensity source.

### HMC6352 Compass

The compass module hardware is installed on the current platform but was not used in the latest revision of the project. For future development, the I2C based peripheral will be used to provide heading verification and contribute to the global model developed within the node. Heading data will be useful in performing mapping routines as well as aiding in more accurate path planning.

### MAXBotix LV-MaxSonar-EZ3 Sonar

The MAXBotix EZ3 sonar is an analog sensor that provides a range of 0-Vcc that is proportional to the distance of the closest object. It is used to provide long-range distance sensing which is primarily used to calculate the throttle for the motor PWM output of the control module. Given a preset minimum distance, the node will begin to decrement the throttle variable until reaching some other preset minimum throttle. This algorithm forces the node to obey a speed limit when objects are detected within a set distance to allow for more frequent proximity detection from the IR array and provide a greater chance for obstacle detection and subsequent avoidance.

### Microchip MCP9700 IC thermistor

The integrated thermistor module provides an analog signal that is proportional to the ambient temperature of the device package. The current project revision does not use the data read from the thermistor, however future revisions will more than likely use the information for detecting temperature minimums or maximums or for logging of temperatures of traversed areas (gradients).

### Nordic nRF24l01+ 2.4GHz transceiver

The RF transceiver implements a Received Power Detector (RPD) for received packets. This functionality provides a rough estimation of the received signal strength for a given channel of communication. In order to deduce the approximate range of a transmitter using this method, the received signal's attenuation is calculated using the received signal strength and the known transmitter signal strength. The calculated range is subject to significant error due to the possibility of indirect transmission (reflected reception) and temperature variations, but can be used to deduce rough estimation of range.

## BEHAVIORS

Local behaviors exhibited by each of the nodes are to avoid detected obstacles as well as monitoring of its own state for limit conditions such as low battery, loss of global communication, light detection and max/min temperature sensing. Global behaviors are controlled through the sharing of state via a node state packet transmitted between each node. The same packet will also trigger a local signal flag for enabling the active sensor elements. This sensor token will eliminate the problem of adjacent nodes interfering with each other's sensor readings.

## Experimental Layout and Results

The first iteration of testing came with the testing of the Sharp GP2Y0D810Z0F
digital IR sensor array. These proved to be effective when used in a steady state configuration. However, the multiplexing scheme that was implemented in the design to conserve power as well as processor IO introduced a transient effect in the switching of the sensor's Vcc line via the FET MUX. The effect was that the 10Hz switching frequency as well as the 15 mil traces of the nPi PCB contributed to noise and a delay in the full powering of the device. To counter this effect, a parallel bypass capacitor network consisting of a 10uF electrolytic capacitor and a 1uF ceramic capacitor were placed across Vcc and Vdd of each sensor. Once installed, the sensors began operating at ~70% of their stated range of 100mm. The reduced operating range was considered acceptable for the application and no further revisions to the circuit were made.

The remainder of component testing involved the setting up of analog sensors which included the phototransistors, the sonar and the RSD feedback of the RF module. The process for testing these components was primarily a scratch paper iterative process of plugging in scaling constants and reading variables within the MPLAB Watch window until a suitable response was achieved.

Behavioral testing is still being completed as a result of the integration of the sense packet interrupt overflows as well as inconsistent switching of the active sensor suite.

## CONCLUSION

As of the writing of this report, 3 nPi nodes sit at my desk eagerly awaiting:

1) A bug free control stack (Modularized)
2) A fully realized and exploited 2.4GHz 2Mbps wireless interface
3) A host to coordinate their behaviors
4) A few more friends

Looking back on the semester, the most glaring regret I have is the fact that I limited my effectiveness from day one by carrying too heavy of a course load for the semester. No other factor played as prominent of a role in preventing me from fully realizing this project. IMDL is a course to be taken very seriously and with great enthusiasm. No other course offered within the ECE department gives you as much freedom to benefit from your efforts, and to limit your effort is to limit your reward. To future IMDLRs: If you happen to think that you're one of the "exceptions" to the rule, and that you can carry more than everyone else and do more than anyone else, as I did....this class will humble you. I promise.

Moving forward, the platform and firmware that was developed over the semester will be used to continue my investigations into cooperative/multi-agent systems. From my experiences, the first few iterations of a new idea rarely exceed expectations. It is the continued development and accruement of new knowledge that ultimately enriches the project and pushes it to the "next level".

## Documentation

dsPIC33FJ128MC506
http://ww1.microchip.com/downloads/en/DeviceDoc/70287C.pdf

GP2Y0D810Z0F
http://sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y0d810z_e.pdf

LV-MaxSonar-EZ3
http://www.maxbotix.com/uploads/LV-MaxSonar-EZ3-Datasheet.pdf

MCP9700
http://ww1.microchip.com/downloads/en/devicedoc/21942a.pdf

Vishay TEPT4400
http://www.vishay.com/docs/81341/81341.pdf

nRF24l01+
http://www.nordicsemi.com/index_popup.cfm?obj=misc&act=download&pro=94&prop=742

## Appendices

In an effort to conserve paper, project code can be found at:
https://sites.google.com/site/npidevelopment/home/project-files

Unless this is an electronic copy, whereas the code will be attached.

```
******************************************************************************
12/6/10 NOTE:
 Sections of code containing currently unresolved bugs have been removed to prevent
dissemination of incorrect information.

This applies specifically to the algorithms pertaining to RSD calculations

DISCLAIMER:  The author provides no guarantees, warrantees, or promises, implied or
 otherwise.  By using this software you agree to indemnify the author
of any damages incurred by using it.

******************************************************************************


// nPi node stack
// Author: M. Hunter Longshore
// Date: 11/27/2010
// Version: 1.2

#include <p33Fxxxx.h>
#include <delay.h>
#include <hd44780_LCD.h>
#include <nrf24l01.h>

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//Node Preprocessor Definitions
// Uncomment one of the following defines to configure the firmware for the appropriate node:
//#define node 0  //Host
#define node 1  //Mobile Node1
//#define node 2  //Mobile Node2
//#define node 3  //Mobile Node3
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

// nRF24L01+ Definitions
#define NRF_FREQ_OFFSET          5       // 2400 + NRF_FREQ_OFFSET = nrf24l01+ carrier in
MHz

#define PIPE0_PCKT_LEN           10      // Packet length in bytes for PIPEn
```

```
#define PIPE1_PCKT_LEN              10
#define PIPE2_PCKT_LEN              10
#define PIPE3_PCKT_LEN              10
#define PIPE4_PCKT_LEN              0
#define PIPE5_PCKT_LEN              0

#define PIPE0_ADDR

// MOTOR MODE Definitions   **********************************
#define MOTION_DISABLE              0
#define MOTION_ROT_CW               1
#define MOTION_ROT_CCW              2
#define MOTION_FORWARD              3
#define MOTION_BACKWARD                     4
//**********************************************************

//#define MTRA_CTRL1                LATDbits.LATD2
//#define MTRA_CTRL2                LATDbits.LATD1
//#define MTRB_CTRL1                LATDbits.LATD5
//#define MTRB_CTRL2                LATDbits.LATD6

#define MTRS_CW                     0x0044
#define MTRS_CCW            0x0022
#define MTRS_FWD            0x0042
#define MTRS_BCK            0x0024

#define MOTOR_STDBY                 LATDbits.LATD4
#define MTRA_PWM            P1DC1
#define MTRB_PWM            P1DC2

#define THROTTLE_GAIN               10                  // Sets increment of PWM for motor
speed ramping loop
#define MIN_THROTTLE                100                 // Minimum PWM duty cycle value
to eliminate PWM deadzone

// Button Definitions  (Active Low) *************************
#define BUTTON1             PORTEbits.RE7
#define BUTTON2             PORTEbits.RE5
```

```
#define BUTTON3                 PORTEbits.RE6


// Sensor Definitions  ***********************************
#define IR_S0                   LATDbits.LATD3
#define IR_S1                   LATDbits.LATD7
#define IR_SENSE_A              PORTDbits.RD0
#define IR_SENSE_B              PORTDbits.RD11

#define SONAR_ENABLE                    LATEbits.LATE4
#define SONAR_VAL_THRESHOLD     223

// Operating Frequency constants
#define F_CY                    30000
#define         F_PWM                           24

#define TMR1_PRD                0x00C7          //Provides 100kHz/10us base interrupt period

//Label and Data Positions
#define A_MOTION_STATE_LCD_LABEL        0x00
#define A_MOTION_STATE_LCD_DATA                 0x05

#define A_MTR_PWM_LCD_LABEL             0x40
#define A_MTR_PWM_LCD_DATA              0x44

#define A_MTR_THRT_LCD_LABEL            0x40
#define A_MTR_THRT_LCD_DATA             0x45

// CONFIGURATION BITS
**************************************************************************

_FBS( RBS_NO_RAM & BSS_NO_FLASH & BWRP_WRPROTECT_OFF )          //No boot RAM, No
boot Program Memory, Write protect disabled
_FSS( RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF)                   //No secure
RAM, No secure FLASH, Secure write protect disabled
_FGS( GSS_OFF & GCP_OFF & GWRP_OFF )                            //Code protect off, write
protect off
```

```
_FOSCSEL( FNOSC_PRIPLL & IESO_OFF )                                    //External crystal with PLL,
two-speed startup disabled
_FOSC( FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_HS )                       //Clock switching and
monitoring disabled, OSC2 is used for XTAL, HS oscillator
_FWDT( FWDTEN_ON & WINDIS_OFF & WDTPRE_PR128 & WDTPOST_PS256)      //Watchdog
timer period = 31us * 128 * 256 = 1.024 sec, WDT window disabled
_FPOR( PWMPIN_OFF & HPOL_ON & LPOL_ON & FPWRT_PWR64 )            //PWM port on,
PWM polarity = Active high, Power on reset Delay = 64ms
_FICD( JTAGEN_OFF & ICS_PGD2 )                         //JTAG disabled, Programming/Debugging
using PG_2
//
***************************************************************************
********************

// FUNCTION PROTOTYPES
void Init_PWM(void);
void Init_IO(void);
void Init_ADC(void);
void Init_DMA(void);
void Init_Display(void);
void Init_SPI2(void);
void Init_TMRS(void);
void Init_Nordic(void);



void Motion_Handler(void);
void Write_Motion_State(void);
void Write_PWM(void);
void Button_Handler(void);
void Pack(void);
void unPack(void);

// Interrupt clock division counters
unsigned int intrpt_div_1ms_ph0 = 50;
unsigned int intrpt_div_10ms_ph0 = 500;
unsigned int intrpt_div_10ms_ph1 = 0;
unsigned int intrpt_div_100ms_ph0 = 7500;
unsigned int intrpt_div_100ms_ph1 = 2500;
unsigned int intrpt_div_100ms_ph2 = 1500;
unsigned int intrpt_div_100ms_ph3 = 0;

// Button debounce buffers and state registers
unsigned int Button1_Buff;
unsigned int Button1_State;
unsigned int Button2_Buff;
unsigned int Button2_State;
unsigned int Button3_Buff;
unsigned int Button3_State;
unsigned int Button_State;
unsigned int Prev_Button_State;

// Motor state register
unsigned int Motion_State;
unsigned int Prev_Motion_State;

// IR Array processing variables
unsigned int IR_Array[6];
unsigned int IR_i;
unsigned int IR_Array_Sum[2];

// Flags
unsigned int Run_State;
unsigned int Sense_Flag;

// Motor Throttle variables
unsigned int Throttle;
unsigned int Max_Throttle;
unsigned int Min_Throttle;

// ADC DMA Buffer
unsigned int DMA_BufferA[16] __attribute__((space(dma)));

//ADC conversion variables
unsigned int Sonar_Val;
unsigned int Opto_Array_val[6];
unsigned int Temp_val;

//Network Packet variables and definitions
int Assemble_Packet;
char Net_Packet[10];
#define Token_Cnt_Byte  Net_Packet[0]
#define Node1_Flag_Byte Net_Packet[1]
#define Node1_Data0_Byte Net_Packet[2]
#define Node1_Data1_Byte Net_Packet[3]
#define Node2_Flag_Byte Net_Packet[1]
#define Node2_Data0_Byte Net_Packet[2]
#define Node2_Data1_Byte Net_Packet[3]
#define Node3_Flag_Byte Net_Packet[1]
#define Node3_Data0_Byte Net_Packet[2]
#define Node3_Data1_Byte Net_Packet[3]


int main(void)
{
        _PLLPRE = 0x0;                  // PLL prescaler = 2
        _PLLDIV = 0x1E;         // PLL divisor = 32
        _PLLPOST = 0x0;         // PLL postscaler = 2

        Init_LCD();
        Init_TMRS();
        Init_IO();
        Init_PWM();
        Init_Display();
        Init_ADC();
        Init_DMA();
        Init_SPI2();
        Init_Nordic();
        delay_us(350000);     // 250ms delay for Sonar

        asm("clrwdt");

        while(1)
        {
                Write_Motion_State();
                Write_PWM();
                Button_Handler();
                Motion_Handler();
                asm("clrwdt");
        }
}


void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
{
        IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag

        intrpt_div_1ms_ph0++;
        if(intrpt_div_1ms_ph0 >= 100)
        {
                intrpt_div_1ms_ph0 = 0;

                if(BUTTON1 == 0)
                {
                        Button1_Buff++;
                        if(Button1_Buff >= 4)
                        {
                                Button1_State = 1;
                                Button3_Buff = 0;
                                Button2_Buff = 0;
                        }
                }
                else
                {
                        Button1_State = 0;
                        Button1_Buff = 0;
                }

                if(BUTTON2 == 0)
                {
                        Button2_Buff++;
                        if(Button2_Buff >= 4)
```

```
				{
					Button2_State = 1;
					Button3_Buff = 0;
					Button1_Buff = 0;
				}
			}
			else
			{
				Button2_State = 0;
				Button2_Buff = 0;
			}

			if(BUTTON3 == 0)
			{
				Button3_Buff++;
				if(Button3_Buff >= 4)
				{
					Button3_State = 1;
					Button1_Buff = 0;
					Button2_Buff = 0;
				}
			}
			else
			{
				Button3_State = 0;
				Button3_Buff = 0;
			}
		}



		Button_State = Button1_State || Button2_State || Button3_State;


		intrpt_div_10ms_ph0++;
		if(intrpt_div_10ms_ph0 >= 1000)
		{
			intrpt_div_10ms_ph0 = 0;
```

```
        DMA0CONbits.CHEN = 1;     // Turn on DMA CH0
        AD1CON1bits.ADON = 1;     // Turn on ADC1
}

intrpt_div_10ms_ph1++;
if(intrpt_div_10ms_ph1 >= 1000)
{
        intrpt_div_10ms_ph1 = 0;
        AD1CON1bits.ADON = 0;     // Turn off ADC1

        Sonar_Val = ((DMA_BufferA[0] + DMA_BufferA[8])>>1);
        Temp_val = ((DMA_BufferA[6] + DMA_BufferA[14])>>1);
        Opto_Array_val[0] = ((DMA_BufferA[7] + DMA_BufferA[15])>>1);
        Opto_Array_val[1] = ((DMA_BufferA[2] + DMA_BufferA[10])>>1);
        Opto_Array_val[2] = ((DMA_BufferA[1] + DMA_BufferA[9])>>1);
        Opto_Array_val[3] = ((DMA_BufferA[3] + DMA_BufferA[11])>>1);
        Opto_Array_val[4] = ((DMA_BufferA[4] + DMA_BufferA[12])>>1);
        Opto_Array_val[5] = ((DMA_BufferA[5] + DMA_BufferA[13])>>1);
}

intrpt_div_100ms_ph0++;
if(Sense_Flag)
{
        if(intrpt_div_100ms_ph0 >=10000)
        {
                intrpt_div_100ms_ph0 = 0;

                SONAR_ENABLE = 1;

                switch(IR_i)
                {
                        //Front-Right and Back-Left
                        case(0):
                        IR_S0 = 0;
                        IR_S0 = 0;
                        IR_S1 = 1;
                        IR_S1 = 1;
                        break;
                        //Front and Back
                        case(2):
                        IR_S0 = 1;
                        IR_S0 = 1;
                        IR_S1 = 0;
                        IR_S1 = 0;
                        break;

                        //Front-Left and Back-Right
                        case(4):
                        IR_S0 = 1;
                        IR_S0 = 1;
                        IR_S1 = 1;
                        IR_S1 = 1;
                        break;
                }
        }

        intrpt_div_100ms_ph1++;
        if(intrpt_div_100ms_ph1 >=10000)
        {
                intrpt_div_100ms_ph1 = 0;

                IR_Array[IR_i] = (!IR_SENSE_A <<(IR_i));
                IR_i++;
                IR_Array[IR_i] = (!IR_SENSE_B <<(IR_i));
                IR_i++;
                if(IR_i >= 6)
                {
                        IR_i = 0;
                        SONAR_ENABLE = 0;
                        Assemble_Packet = 1;
                }
        }
}

else
```

```
{
        IR_S0 = 0;
        IR_S0 = 0;
        IR_S1 = 0;
        IR_S1 = 0;
        SONAR_ENABLE = 0;
}

intrpt_div_100ms_ph3++;
if(intrpt_div_100ms_ph3 >=10000)
{
        intrpt_div_100ms_ph3 = 0;

        if(Motion_State != Prev_Motion_State)
        {
                LATD = (LATD & 0xFF99);             // Clear motor control bits
                MTRA_PWM = 0;                       // Stop PWM for direction change
                MTRB_PWM = 0;                       // Stop PWM for direction change
                Throttle = MIN_THROTTLE;

                switch(Motion_State)
                        {

                                case (MOTION_DISABLE):
                                MOTOR_STDBY = 0;
                                break;

                                case (MOTION_ROT_CW):
                                MOTOR_STDBY = 1;
                                LATD = (LATD | MTRS_CW);
                                break;

                                case (MOTION_ROT_CCW):
                                MOTOR_STDBY = 1;
                                LATD = (LATD | MTRS_CCW);
                                break;

                                case (MOTION_FORWARD):
```

```
            MOTOR_STDBY = 1;
            LATD = (LATD | MTRS_FWD);
            break;

            case (MOTION_BACKWARD):
            MOTOR_STDBY = 1;
            LATD = (LATD | MTRS_BCK);
            break;
        }
}

else
{
    if(Run_State ==1)
    {
        if(Sonar_Val <= SONAR_VAL_THRESHOLD)
        {
            if(Throttle > Min_Throttle) {Throttle--;}
            else Throttle = Min_Throttle;

        }

        else
        {
            if(Throttle >= Max_Throttle)
            {
                Throttle = Max_Throttle;
            }
            else
            {
                Throttle = Throttle + THROTTLE_GAIN;
            }
        }

    }
}
```

```
			Prev_Motion_State = Motion_State;
		}

	}


void Init_ADC(void)
{
		AD1CON1bits.AD12B = 0;			//  10-bit conversion mode
		AD1CON2bits.VCFG = 0b000; //  Vref = +AVdd and -AVss
		AD1CON3bits.ADRC = 0;			//  ADC clock derived from system clock
		AD1CON3bits.ADCS = 0x3F;   //  ADC Conversion clock = 64 * Tcy = 64 * 33ns = 21.3us
		AD1CON3bits.SAMC = 2;			//  2 TAD is used conversion
		AD1CON1bits.SSRC = 0b111;  //  Interval counter ends sampling and auto starts-
conversion
		AD1CON1bits.ASAM = 1;			//  Sampling begins immediately after previous
conversion is complete


		AD1PCFGLbits.PCFG0 = 0;			//  AN0 = Analog Sonar input
		AD1PCFGLbits.PCFG1 = 1;			//  AN1 = Digital
		AD1PCFGLbits.PCFG2 = 1;			//  AN2 = Digital
		AD1PCFGLbits.PCFG3 = 0;			//  AN3 = Analog OPTO_2 front-left
		AD1PCFGLbits.PCFG4 = 0;			//  AN4 = Analog OPTO_1 top-right
		AD1PCFGLbits.PCFG5 = 0;			//  AN5 = Analog OPTO_3 front-right
		AD1PCFGLbits.PCFG6 = 0;			//  AN6 = Analog OPTO_4 back-left
		AD1PCFGLbits.PCFG7 = 0;			//  AN7 = Analog OPTO_5 back-right
		AD1PCFGLbits.PCFG8 = 1;			//  AN8 = Digital
		AD1PCFGLbits.PCFG9 = 1;			//  AN9 = Digital
		AD1PCFGLbits.PCFG10 = 1;    //  AN10 = Digital
		AD1PCFGLbits.PCFG11 = 1;    //  AN11 = Digital
		AD1PCFGLbits.PCFG12 = 0;    //  AN12 = Analog Temperature
		AD1PCFGLbits.PCFG13 = 1;    //  AN13 = Digital
		AD1PCFGLbits.PCFG14 = 1;    //  AN14 = Digital
		AD1PCFGLbits.PCFG15 = 0;    //  AN15 = Analog OPTO_0 top-left

		//Set-up for Sequential sampling of 8-ports through CH0
```

```
          AD1CON2bits.CHPS = 0;                  //  Sample CH0
          AD1CON2bits.CSCNA = 1;                 //  Scan inputs for CH0
          AD1CHS0bits.CH0NA = 0;                 //  CH0 negative input is Vrefl
          AD1CON2bits.BUFM = 0;                  //  Fill buffer starting from lowest address
          AD1CON2bits.ALTS = 0;                  //  Always use Sample A channel input selects



          // Port selection for sequence conversion
          AD1CSSLbits.CSS0 = 1;
          AD1CSSLbits.CSS1 = 0;
          AD1CSSLbits.CSS2 = 0;
          AD1CSSLbits.CSS3 = 1;
          AD1CSSLbits.CSS4 = 1;
          AD1CSSLbits.CSS5 = 1;
          AD1CSSLbits.CSS6 = 1;
          AD1CSSLbits.CSS7 = 1;
          AD1CSSLbits.CSS8 = 0;
          AD1CSSLbits.CSS9 = 0;
          AD1CSSLbits.CSS10 = 0;
          AD1CSSLbits.CSS11 = 0;
          AD1CSSLbits.CSS12 = 1;
          AD1CSSLbits.CSS13 = 0;
          AD1CSSLbits.CSS14 = 0;
          AD1CSSLbits.CSS15 = 1;
}

void Init_DMA(void)
{
          AD1CON1bits.ADDMABM = 1;                       //  DMA buffer is written in order of
conversion

          AD1CON2bits.SMPI = 0xF;                        //  DMA request is made after every
conversion
          DMA0REQbits.IRQSEL = 0b0001101;                //  Set DMA CH0 to interrupt on ADC1
conversion done flag
          DMA0CONbits.AMODE = 0b10;                      //
```

```
        DMA0STA = __builtin_dmaoffset(DMA_BufferA);

        DMA0PAD = (int)&ADC1BUF0;                          //  Set DMA to read from ADC1BUF0

        DMA0CNT = 15;                                 //  15 DMA requests before completion
        DMA0CONbits.MODE = 0;                         //  DMA CH0 operating in one-shot mode
with no ping-pong;
}
void Motion_Handler(void)
{
        int i;
        int j;
        int fwd_masked_ir_array_sum;
        int IR_Motion_State;
        int Opto_Motion_State;
        int Max_Opto;

        IR_Array_Sum[0] = 0;

        for(i=0; i < 6; i++)
        {
                IR_Array_Sum[0] = IR_Array_Sum[0] + IR_Array[i];
        }

        if(Run_State == 1)
        {
                fwd_masked_ir_array_sum = (IR_Array_Sum[0] & 0b010101);

                if(!fwd_masked_ir_array_sum){IR_Motion_State = MOTION_FORWARD;}

                else
                {               switch(IR_Array_Sum[0])
                                {

                                        case(0b000001):
                                        switch(Motion_State)
                                        {
                                                case MOTION_ROT_CW ... MOTION_ROT_CCW:
```

```
                IR_Motion_State = Motion_State;
                break;

                case(MOTION_FORWARD):
                IR_Motion_State = MOTION_ROT_CCW;
                break;

                case(MOTION_BACKWARD):
                IR_Motion_State = MOTION_BACKWARD;
                break;
        }
        break;

        case(0b000011):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b000100):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CW;
                break;
        }
        break;
```

```
case(0b000101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD:
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b000110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case(MOTION_FORWARD):
        IR_Motion_State = MOTION_ROT_CW;
        break;

        case(MOTION_BACKWARD):
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b000111):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;
        case(MOTION_FORWARD):
        IR_Motion_State = MOTION_ROT_CW;
        break;

        case(MOTION_BACKWARD):
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b001001):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b001011):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b001100):
switch(Motion_State)
```

```
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b001101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b001110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b001111):
```

```
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b010000):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b010001):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;
```

```
case(0b010010):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b010011):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b010101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;
```

```
case(0b010110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b010111):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b011000):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
```

```
break;

case(0b011001):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b011010):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b011011):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
```

```
}
break;

case(0b011100):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b011101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b011110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b011111):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b100001):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b100011):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
```

```
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b100100):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b100101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b100110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;
```

```
                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b100111):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b101001):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b101011):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;
                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b101100):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CW;
                break;
        }
        break;

        case(0b101101):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
                break;

                case MOTION_FORWARD ... MOTION_BACKWARD :
                IR_Motion_State = MOTION_ROT_CCW;
                break;
        }
        break;

        case(0b101110):
        switch(Motion_State)
        {
                case MOTION_ROT_CW ... MOTION_ROT_CCW :
                IR_Motion_State = Motion_State;
```

```
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CW;
            break;
}
break;

case(0b101111):
switch(Motion_State)
{
            case MOTION_ROT_CW ... MOTION_ROT_CCW :
            IR_Motion_State = Motion_State;
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CCW;
            break;
}
break;

case(0b110000):
switch(Motion_State)
{
            case MOTION_ROT_CW ... MOTION_ROT_CCW :
            IR_Motion_State = Motion_State;
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CW;
            break;
}
break;


case(0b110001):
switch(Motion_State)
{
```

```
            case MOTION_ROT_CW ... MOTION_ROT_CCW :
            IR_Motion_State = Motion_State;
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CCW;
            break;
}
break;

case(0b110010):
switch(Motion_State)
{
            case MOTION_ROT_CW ... MOTION_ROT_CCW :
            IR_Motion_State = Motion_State;
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CW;
            break;
}
break;

case(0b110011):
switch(Motion_State)
{
            case MOTION_ROT_CW ... MOTION_ROT_CCW :
            IR_Motion_State = Motion_State;
            break;

            case MOTION_FORWARD ... MOTION_BACKWARD :
            IR_Motion_State = MOTION_ROT_CCW;
            break;
}
break;

case(0b110100):
switch(Motion_State)
```

```
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b110101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b110110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b110111):
```

```
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b111000):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b111001):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;
```

```
case(0b111010):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

case(0b111011):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b111100):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;
```

```
case(0b111101):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
}
break;

case(0b111110):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CW;
        break;
}
break;

//All sensors detect
case(0b111111):
switch(Motion_State)
{
        case MOTION_ROT_CW ... MOTION_ROT_CCW :
        IR_Motion_State = Motion_State;
        break;

        case MOTION_FORWARD ... MOTION_BACKWARD :
        IR_Motion_State = MOTION_ROT_CCW;
        break;
```

```
                    }
                    break;
          }
     }

for(i=0; i<6; i++)
{
     for(j=(i+1); j<6; j++)
     {
          if(Opto_Array_val[i] <= Opto_Array_val[j]){j=6;}
          if( j == 5){Max_Opto = i;}
     }
}

switch(Max_Opto)
{
     case 0 :
     Opto_Motion_State = MOTION_ROT_CCW;
     break;

     case 1 :
     Opto_Motion_State = MOTION_ROT_CW;
     break;

     case 2 :
     Opto_Motion_State = MOTION_ROT_CCW;
     break;

     case 3 :
     Opto_Motion_State = MOTION_ROT_CW;
     break;

     case 4 :
     Opto_Motion_State = MOTION_ROT_CCW;
     break;

     case 5 :
     Opto_Motion_State = MOTION_ROT_CW;
```

```
				break;
			}

			if((IR_Motion_State == MOTION_FORWARD) || (IR_Motion_State ==
MOTION_BACKWARD)){Motion_State = Opto_Motion_State;}
			else{Motion_State = IR_Motion_State;}

			MTRA_PWM = Throttle;
			MTRB_PWM = Throttle;
		}

}



void Button_Handler(void)
{
		if(Button_State != Prev_Button_State)
		{
			if(Button1_State)
			{
				if(Max_Throttle >= (2*PTPER)) Max_Throttle = (2*PTPER);
				else Max_Throttle = Max_Throttle + 20;
			}

			if(Button2_State)
			{
				if(Max_Throttle == 0) Max_Throttle = 0;
				else Max_Throttle = Max_Throttle - 20;
			}

			if(Button3_State)
			{
				if(Run_State == 0)
				{
					Run_State = 1;
				}

				else
```

```
            {
                Run_State = 0;
                Motion_State = MOTION_DISABLE;
            }
        }


        Prev_Button_State = Button_State;
    }
}
void Init_Display()
{
    WriteLCD_DDR(A_MOTION_STATE_LCD_LABEL);
    WriteLCD_String("Mode:");
    WriteLCD_DDR(A_MTR_THRT_LCD_LABEL);
    WriteLCD_String("MaxT:--%");
}

void Init_IO(void)
{
    TRISD = 0xFF00;
    TRISB = 0xFFFF;
    TRISE = 0xFFE5;
}



void Init_Nordic(void)
{

//   Nordic Initialization
************************************************************
//
**************************************************************************
*********
//
//   Pipe Configuration:
************************************************************
```

```
//    pipe | 5-byte address   | description
//   -------------------------------------
//        P0  | x61676f6E79      | Host <-> All Nodes
//        P1  | x6D75736531      | Node0 <-> Node1
//        P2  | x6D75736534      | Node1 <-> Node2
//        P3  | x6D75736538      | Node2 <-> Node0
//
//   Shared Packet
Structure:**************************************************************
//
//   Byte |   Desc.
//  ---------------------------
//    0   |  Token_Cnt    Increments after each node handles the packet.
//        |                          For n nodes, the byte rolls over to zero after Token_Cnt = n
//        |
//    1   |  Node1_Flag_Byte              Flag byte for Node1
//    2   |  Node1_DAT0_Byte              DAT0 byte for Node1
//    3   |  Node1_DAT1_Byte              DAT1 byte for Node1
//        |
//    4   |  Node2_Flag_Byte              Flag byte for Node2
//    5   |  Node2_DAT0_Byte              DAT0 byte for Node2
//    6   |  Node2_DAT1_Byte              DAT1 byte for Node2
//        |
//    7   |  Node3_Flag_Byte              Flag byte for Node3
//    8   |  Node3_DAT0_Byte              DAT0 byte for Node3
//    9   |  Node3_DAT1_Byte              DAT1 byte for Node3
//
//
//        void nrf24l01_initialize(unsigned char config,
//                                 unsigned char opt_rx_standby_mode,
//                                 unsigned char en_aa,
//                                 unsigned char en_rxaddr,
//                                 unsigned char setup_aw,
//                                 unsigned char setup_retr,
//                                 unsigned char rf_ch,
//                                 unsigned char rf_setup,
//                                 unsigned char * rx_addr_p0,
//                                 unsigned char * rx_addr_p1,
//                              unsigned char rx_addr_p2,
//                              unsigned char rx_addr_p3,
//                              unsigned char rx_addr_p4,
//                              unsigned char rx_addr_p5,
//                              unsigned char * tx_addr,
//                              unsigned char rx_pw_p0,
//                              unsigned char rx_pw_p1,
//                              unsigned char rx_pw_p2,
//                              unsigned char rx_pw_p3,
//                              unsigned char rx_pw_p4,
//                              unsigned char rx_pw_p5);

        nrf24l01_initialize(    (nrf24l01_CONFIG_DEFAULT_VAL | nrf24l01_CONFIG_PWR_UP |
nrf24l01_CONFIG_EN_CRC | nrf24l01_CONFIG_CRCO),

                        #if node==0
                                true,           //Primary RX
                        #elif node==1
                                false,          //Primary TX
                        #elif node==2
                                true,           //Primary RX
                        #elif node==3
                                true,           //Primary RX
                        #endif
                                (nrf24l01_EN_AA_ENAA_P0 | nrf24l01_EN_AA_ENAA_P1 |
nrf24l01_EN_AA_ENAA_P2 | nrf24l01_EN_AA_ENAA_P3),
                                (nrf24l01_EN_RXADDR_ERX_P0 | nrf24l01_EN_RXADDR_ERX_P1
| nrf24l01_EN_RXADDR_ERX_P2 | nrf24l01_EN_RXADDR_ERX_P3),
                                nrf24l01_SETUP_AW_5BYTES,
                                nrf24l01_SETUP_RETR_DEFAULT_VAL,
                                NRF_FREQ_OFFSET,
                                nrf24l01_RF_SETUP_DEFAULT_VAL,
                                NULL,
                                NULL,
                                NULL,
                                NULL,
                                nrf24l01_RX_ADDR_P4_DEFAULT_VAL,
                                nrf24l01_RX_ADDR_P5_DEFAULT_VAL,
                                        NULL,
                                        PIPE0_PCKT_LEN,
                                        PIPE1_PCKT_LEN,
                                        PIPE2_PCKT_LEN,
                                        PIPE3_PCKT_LEN,
                                        PIPE4_PCKT_LEN,
                                        PIPE5_PCKT_LEN);
}

void Init_PWM(void)
{
          TRISE = 0xFFF5;                                       // PWM RE1,RE3 enabled
          PTPER = (F_CY/F_PWM) - 1;                   // Compute Period based on CPU
speed and PWM frequency

          OVDCON = 0x0000;                            // Disable all PWM outputs.

          DTCON1bits.DTAPS = 0b11;                    // 8 TCY dead A time prescale = .4us
          DTCON1bits.DTA = 2;                         // Dead time A = 8Tcy * 2 = .8us
          PWMCON1 = 0x0030;                           // Enable PWM output mode/non-
complimentary for PWM channel 1H and 2H
          MTRA_PWM = 0;                                         // Initialize MTRA with 0
PWM duty cycle
          MTRB_PWM = 0;                                         // Initialize MTRB with 0
PWM duty cycle

          PWMCON2 = 0x0002;                           // Output overides are synchronous
with PWM time base.

          P1TCON = 0x8000;                            // Start PWM in free-run mode

          P1OVDCONbits.POVD1H = 1;
          P1OVDCONbits.POVD2H = 1;                    // Output for PWM1,PWM3 are
controlled by PWM module
}

void Init_SPI2()
{
          IFS2bits.SPI2IF = 0;            // Clear the Interrupt Flag
          IEC2bits.SPI2IE = 0;            // Disable the Interrupt

          SPI2CON1bits.MODE16 = 0;    // Set data mode to 8-bits

          SPI2CON1bits.DISSCK = 0;        // Internal SPI clock is enabled
          SPI2CON1bits.DISSDO = 0;        // SDO2 pin is controlled by the module
          SPI2CON1bits.CKE = 1;                   // MOSI data changes on the falling edge of SCK
          SPI1CON1bits.SMP = 0;                   // Data sampled at middle of clock pulse
          SPI2CON1bits.CKP = 0;                   // SCK is active-high
          SPI2CON1bits.SSEN = 0;                  // Slave select is a port pin.

          SPI2CON1bits.PPRE = 0b10;   // Primary prescaler = 4:1
          SPI2CON1bits.SPRE = 0b100;  // Secondary prescaler = 4:1
                                      // Fspi = (FCY = 40MHz)/((2*PPRE = 2*64) * (2*SPRE =
2*2))
                                      // Fspi frequency = 2.5MHz

          SPI2CON1bits.MSTEN = 1;                 // SPI2 is a MASTER
          SPI2BUF = 0;                    // Clear the SPI2 Transmit Buffer
          SPI2STATbits.SPIEN = 1;                 // Enable the SPI2 module
}

void Init_TMRS()
{
          //TMR1
          T1CONbits.TON = 0;    // Disable Timer
          T1CONbits.TCS = 0;    // Select internal instruction cycle clock
          T1CONbits.TGATE = 0;            // Disable Gated Timer mode
          T1CONbits.TCKPS = 0;            // Select 1:1 Prescaler
          TMR1 = 0;             // Clear timer register
          PR1 = TMR1_PRD;       // Load the period value

          IPC0bits.T1IP = 0x05; // Set Timer 1 Interrupt Priority Level
          IFS0bits.T1IF = 0;    // Clear Timer 1 Interrupt Flag
          IEC0bits.T1IE = 1;    // Enable Timer 1 interrupt

          T1CONbits.TON = 1;    // Enable Timer
```

```
}

void Write_Motion_State(void)
{
	WriteLCD_DDR(A_MOTION_STATE_LCD_DATA);

	switch(Motion_State)
	{
		case MOTION_DISABLE:
		WriteLCD_String("DSB");
		break;
		case MOTION_ROT_CW:
		WriteLCD_String("CW ");
		break;
		case MOTION_ROT_CCW:
		WriteLCD_String("CCW");
		break;
		case MOTION_FORWARD:
		WriteLCD_String("FWD");
		break;
		case MOTION_BACKWARD:
		WriteLCD_String("BCK");
		break;
	}

}

void Write_PWM(void)
{
	int PWM_10;
	int PWM_01;
	int duty_cycle;

	duty_cycle = (Max_Throttle * .06);

	PWM_10 = ((((duty_cycle%100 - duty_cycle%10)/10)) + 48);
	PWM_01 = (duty_cycle%10 + 48);
```

```
        WriteLCD_DDR(A_MTR_THRT_LCD_DATA);
        WriteLCD_Data(PWM_10);
        WriteLCD_Data(PWM_01);
}



/*****************************************************************
File Name:              hd44780_LCD.C
Description:        Functions for writing to an hd44780 compatible LCD
*****************************************************************/
#include <p33fxxxx.h>
#include <hd44780_LCD.h>
#include <delay.h>


//Initialize LCD for 4-bit / 2-line mode
void Init_LCD(void)
{
        Delay_cycles(Delay_80ms);
        WriteLCD_CMD(FOUR_BIT);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(FOUR_BIT);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(FOUR_BIT);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(FOUR_BIT);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(DON);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(CLEAR);
        Delay_cycles(Delay_5_5ms);
        WriteLCD_CMD(ENTIRE_SHIFT_OFF);
        Delay_cycles(Delay_290us);
        WriteLCD_CMD(CURSOR_OFF);
        Delay_cycles(Delay_290us);
}
```

```
//Write command to LCD
void WriteLCD_CMD(char cmd)
{
        TRIS_E = 0;
        TRIS_RW = 0;
        TRIS_RS = 0;

        RW_PIN = 0;
        RS_PIN = 0;

        TRIS_DATA_PIN_7 = 0;
        TRIS_DATA_PIN_6 = 0;
        TRIS_DATA_PIN_5 = 0;
        TRIS_DATA_PIN_4 = 0;

        DATA_PIN_7 = (unsigned int)((cmd & 0x80)>>7);
        DATA_PIN_6 = (unsigned int)((cmd & 0x40)>>6);
        DATA_PIN_5 = (unsigned int)((cmd & 0x20)>>5);
        DATA_PIN_4 = (unsigned int)((cmd & 0x10)>>4);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        DATA_PIN_7 = (unsigned int)((cmd & 0x08)>>3);
        DATA_PIN_6 = (unsigned int)((cmd & 0x04)>>2);
        DATA_PIN_5 = (unsigned int)((cmd & 0x02)>>1);
        DATA_PIN_4 = (unsigned int)(cmd & 0x01);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        Delay_cycles(Delay_290us);
}
```

```
//Write data to LCD
void WriteLCD_Data(char data)
{
        TRIS_E = 0;
        TRIS_RW = 0;
        TRIS_RS = 0;

        RW_PIN = 0;  /* enable write */
        RS_PIN = 1;  /* select Data Reg */

        TRIS_DATA_PIN_7 = 0;
        TRIS_DATA_PIN_6 = 0;
        TRIS_DATA_PIN_5 = 0;
        TRIS_DATA_PIN_4 = 0;

        DATA_PIN_7 = (unsigned int)((data & 0x80)>>7);
        DATA_PIN_6 = (unsigned int)((data & 0x40)>>6);
        DATA_PIN_5 = (unsigned int)((data & 0x20)>>5);
        DATA_PIN_4 = (unsigned int)((data & 0x10)>>4);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        DATA_PIN_7 = (unsigned int)((data & 0x08)>>3);
        DATA_PIN_6 = (unsigned int)((data & 0x04)>>2);
        DATA_PIN_5 = (unsigned int)((data & 0x02)>>1);
        DATA_PIN_4 = (unsigned int)(data & 0x01);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        RS_PIN = 0;

        Delay_cycles(Delay_290us);
}
```

```
//Set DDR address in LCD for next write
void WriteLCD_DDR(char address)
{
        TRIS_E = 0;
        TRIS_RW = 0;
        TRIS_RS = 0;

        RW_PIN = 0;
        RS_PIN = 0;

        TRIS_DATA_PIN_7 = 0;
        TRIS_DATA_PIN_6 = 0;
        TRIS_DATA_PIN_5 = 0;
        TRIS_DATA_PIN_4 = 0;

        DATA_PIN_7 = 1;
        DATA_PIN_6 = (unsigned int)((address & 0x40)>>6);
        DATA_PIN_5 = (unsigned int)((address & 0x20)>>5);
        DATA_PIN_4 = (unsigned int)((address & 0x10)>>4);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        DATA_PIN_7 = (unsigned int)((address & 0x08)>>3);
        DATA_PIN_6 = (unsigned int)((address & 0x04)>>2);
        DATA_PIN_5 = (unsigned int)((address & 0x02)>>1);
        DATA_PIN_4 = (unsigned int)(address & 0x01);

        E_PIN = 1;
        Delay_cycles(Delay_880ns);
        E_PIN = 0;

        Delay_cycles(Delay_290us);
}
```

```
//Write character string to LCD
void WriteLCD_String(char *buffer)
{
   while(*buffer != '\0')
   {
      WriteLCD_Data(*buffer);
      buffer++;
   }
}
```

```
/*****************************************************************************
*
*
* File: nrf24l01.c
*
* Copyright S. Brennen Ball, 2006-2007
*
* The author provides no guarantees, warantees, or promises, implied or
*       otherwise.  By using this software you agree to indemnify the author
*       of any damages incurred by using it.
*
*****************************************************************************/

#include "nrf24l01.h"

//Arguments except opt_rx_standby_mode fill the actual register they are named
//  after. Registers that do not need to be initialized are not included here.
//The argument opt_rx_active_mode is only used if the user is initializing the
//  24L01 as a receiver.  If the argument is false, the receiver will remain in
//  standby mode and not monitor for packets.  If the argument is true, the CE
//  pin will be set and the 24L01 will monitor for packets.  In TX mode, the value
//  of this argument is insignificant.
//If the user wants to leave any 1-byte register in its default state, simply put
//  as that register's argument nrf24l01_<reg>_DEFAULT_VAL, where <reg> is the register
//  name.
//If the user wants to leave any of the 5-byte registers RX_ADDR_P0, RX_ADDR_P1, or
//  TX_ADDR in its default state, simply put NULL in the argument for that address value.
void nrf24l01_initialize(unsigned char config,
                         unsigned char opt_rx_active_mode,
                         unsigned char en_aa,
                         unsigned char en_rxaddr,
                         unsigned char setup_aw,
                         unsigned char setup_retr,
                         unsigned char rf_ch,
                         unsigned char rf_setup,
                         unsigned char * rx_addr_p0,
                         unsigned char * rx_addr_p1,
                         unsigned char rx_addr_p2,
                         unsigned char rx_addr_p3,
                         unsigned char rx_addr_p4,
                         unsigned char rx_addr_p5,
                         unsigned char * tx_addr,
                         unsigned char rx_pw_p0,
                         unsigned char rx_pw_p1,
                         unsigned char rx_pw_p2,
                         unsigned char rx_pw_p3,
                         unsigned char rx_pw_p4,
                         unsigned char rx_pw_p5)
{
        unsigned char data[5];

        data[0] = en_aa;
        nrf24l01_write_register(nrf24l01_EN_AA, data, 1);

        data[0] = en_rxaddr;
        nrf24l01_write_register(nrf24l01_EN_RXADDR, data, 1);

        data[0] = setup_aw;
        nrf24l01_write_register(nrf24l01_SETUP_AW, data, 1);

        data[0] = setup_retr;
        nrf24l01_write_register(nrf24l01_SETUP_RETR, data, 1);

        data[0] = rf_ch;
        nrf24l01_write_register(nrf24l01_RF_CH, data, 1);

        data[0] = rf_setup;
        nrf24l01_write_register(nrf24l01_RF_SETUP, data, 1);
```

```
if(rx_addr_p0 != NULL)
	nrf24l01_set_rx_addr(rx_addr_p0, 5, 0);
else
{
	data[0] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;
	data[1] = nrf24l01_RX_ADDR_P0_B1_DEFAULT_VAL;
	data[2] = nrf24l01_RX_ADDR_P0_B2_DEFAULT_VAL;
	data[3] = nrf24l01_RX_ADDR_P0_B3_DEFAULT_VAL;
	data[4] = nrf24l01_RX_ADDR_P0_B4_DEFAULT_VAL;

	nrf24l01_set_rx_addr(data, 5, 0);
}

if(rx_addr_p1 != NULL)
	nrf24l01_set_rx_addr(rx_addr_p1, 5, 1);
else
{
	data[0] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;
	data[1] = nrf24l01_RX_ADDR_P1_B1_DEFAULT_VAL;
	data[2] = nrf24l01_RX_ADDR_P1_B2_DEFAULT_VAL;
	data[3] = nrf24l01_RX_ADDR_P1_B3_DEFAULT_VAL;
	data[4] = nrf24l01_RX_ADDR_P1_B4_DEFAULT_VAL;

	nrf24l01_set_rx_addr(data, 5, 1);
}

data[0] = rx_addr_p2;
nrf24l01_set_rx_addr(data, 1, 2);

data[0] = rx_addr_p3;
nrf24l01_set_rx_addr(data, 1, 3);

data[0] = rx_addr_p4;
nrf24l01_set_rx_addr(data, 1, 4);

data[0] = rx_addr_p5;
nrf24l01_set_rx_addr(data, 1, 5);
```

```
        if(tx_addr != NULL)
                nrf24l01_set_tx_addr(tx_addr, 5);
        else
        {
#if node==0
                data[0] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;
                data[1] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;
                data[2] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;
                data[3] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;
                data[4] = nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL;

#elif node==1
                data[0] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;
                data[1] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;
                data[2] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;
                data[3] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;
                data[4] = nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL;

#elif node==2
                data[0] = nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL;
                data[1] = nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL;
                data[2] = nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL;
                data[3] = nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL;
                data[4] = nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL;

#elif node==3
                data[0] = nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL;
                data[1] = nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL;
                data[2] = nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL;
                data[3] = nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL;
                data[4] = nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL;
#endif



                nrf24l01_set_tx_addr(data, 5);
        }

        data[0] = rx_pw_p0;
```

```
	nrf24l01_write_register(nrf24l01_RX_PW_P0, data, 1);

	data[0] = rx_pw_p1;
	nrf24l01_write_register(nrf24l01_RX_PW_P1, data, 1);

	data[0] = rx_pw_p2;
	nrf24l01_write_register(nrf24l01_RX_PW_P2, data, 1);

	data[0] = rx_pw_p3;
	nrf24l01_write_register(nrf24l01_RX_PW_P3, data, 1);

	data[0] = rx_pw_p4;
	nrf24l01_write_register(nrf24l01_RX_PW_P4, data, 1);

	data[0] = rx_pw_p5;
	nrf24l01_write_register(nrf24l01_RX_PW_P5, data, 1);

	if((config & nrf24l01_CONFIG_PWR_UP) != 0)
		nrf24l01_power_up_param(opt_rx_active_mode, config);
	else
		nrf24l01_power_down_param(config);
}

//initializes the 24L01 to all default values except the PWR_UP and PRIM_RX bits
//this function also disables the auto-ack feature on the chip (EN_AA register is 0)
//bool rx is true if the device should be a receiver and false if it should be
//  a transmitter.
//unsigned char payload_width is the payload width for pipe 0.  All other pipes
//  are left in their default (disabled) state.
//bool enable_auto_ack controls the auto ack feature on pipe 0.  If true, auto-ack will
//  be enabled.  If false, auto-ack is disabled.
void nrf24l01_initialize_debug(bool rx, unsigned char p0_payload_width, bool
enable_auto_ack)
{
	unsigned char config;
	unsigned char en_aa;

	config = nrf24l01_CONFIG_DEFAULT_VAL | nrf24l01_CONFIG_PWR_UP;
	if(enable_auto_ack != false)
		en_aa = nrf24l01_EN_AA_ENAA_P0;
	else
		en_aa = nrf24l01_EN_AA_ENAA_NONE;

	if(rx == true)
		config = config | nrf24l01_CONFIG_PRIM_RX;

	nrf24l01_initialize(config,
			true,
			en_aa,
			nrf24l01_EN_RXADDR_DEFAULT_VAL,
			nrf24l01_SETUP_AW_DEFAULT_VAL,
			nrf24l01_SETUP_RETR_DEFAULT_VAL,
			nrf24l01_RF_CH_DEFAULT_VAL,
			nrf24l01_RF_SETUP_DEFAULT_VAL,
			NULL,
			NULL,
			nrf24l01_RX_ADDR_P2_DEFAULT_VAL,
			nrf24l01_RX_ADDR_P3_DEFAULT_VAL,
			nrf24l01_RX_ADDR_P4_DEFAULT_VAL,
			nrf24l01_RX_ADDR_P5_DEFAULT_VAL,
			NULL,
			p0_payload_width,
			nrf24l01_RX_PW_P1_DEFAULT_VAL,
			nrf24l01_RX_PW_P2_DEFAULT_VAL,
			nrf24l01_RX_PW_P3_DEFAULT_VAL,
			nrf24l01_RX_PW_P4_DEFAULT_VAL,
			nrf24l01_RX_PW_P5_DEFAULT_VAL);
}

//initializes only the CONFIG register and pipe 0's payload width
//the primary purpose of this function is to allow users with microcontrollers with
//  extremely small program memories to still be able to init their 24L01.  This code
//  should have a smaller footprint than the above init functions.
//when using this method, the 24L01 MUST have its default configuration loaded
//  in all registers to work.  It is recommended that the device be reset or
//  have its power cycled immediately before this code is run.
//in normal circumstances, the user should use nrf24l01_initialize() rather than this
//  function, since this function does not set all of the register values.
void nrf24l01_initialize_debug_lite(bool rx, unsigned char p0_payload_width)
{
	unsigned char config;

	config = nrf24l01_CONFIG_DEFAULT_VAL;

	if(rx != false)
		config |= nrf24l01_CONFIG_PRIM_RX;

	nrf24l01_write_register(nrf24l01_RX_PW_P0, &p0_payload_width, 1);
	nrf24l01_power_up_param(true, config);
}

//powers up the 24L01 with all necessary delays
//this function takes the existing contents of the CONFIG register and sets the PWR_UP
//the argument rx_active_mode is only used if the user is setting up the
//  24L01 as a receiver.  If the argument is false, the receiver will remain in
//  standby mode and not monitor for packets.  If the argument is true, the CE
//  pin will be set and the 24L01 will monitor for packets.  In TX mode, the value
//  of this argument is insignificant.
//note: if the read value of the CONFIG register already has the PWR_UP bit set, this function
//  exits in order to not make an unecessary register write.
void nrf24l01_power_up(bool rx_active_mode)
{
	unsigned char config;

	nrf24l01_read_register(nrf24l01_CONFIG, &config, 1);

	if((config & nrf24l01_CONFIG_PWR_UP) != 0)
		return;

	config |= nrf24l01_CONFIG_PWR_UP;

	nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);
	delay_us(1500);

	if((config & nrf24l01_CONFIG_PRIM_RX) == 0)
		nrf24l01_clear_ce();
	else
	{
		if(rx_active_mode != false)
			nrf24l01_set_ce();
		else
			nrf24l01_clear_ce();
	}
}

//powers up the 24L01 with all necessary delays
//this function allows the user to set the contents of the CONFIG register, but the function
//  sets the PWR_UP bit in the CONFIG register, so the user does not need to.
//the argument rx_active_mode is only used if the user is setting up the
//  24L01 as a receiver.  If the argument is false, the receiver will remain in
//  standby mode and not monitor for packets.  If the argument is true, the CE
//  pin will be set and the 24L01 will monitor for packets.  In TX mode, the value
//  of this argument is insignificant.
void nrf24l01_power_up_param(bool rx_active_mode, unsigned char config)
{
	unsigned char test, test2;

	config |= nrf24l01_CONFIG_PWR_UP;

	nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);

	delay_us(1500);

	if((config & nrf24l01_CONFIG_PRIM_RX) == 0)
		nrf24l01_clear_ce();
	else
	{
		if(rx_active_mode != false)
			nrf24l01_set_ce();
		else
```

```
				nrf24l01_clear_ce();
		}
}

//powers down the 24L01
//this function takes the existing contents of the CONFIG register and simply
//  clears the PWR_UP bit in the CONFIG register.
//note: if the read value of the CONFIG register already has the PWR_UP bit cleared, this
//  function exits in order to not make an unecessary register write.
void nrf24l01_power_down()
{
		unsigned char config;

		nrf24l01_read_register(nrf24l01_CONFIG, &config, 1);

		if((config & nrf24l01_CONFIG_PWR_UP) == 0)
			return;

		config &= (~nrf24l01_CONFIG_PWR_UP);

		nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);

		nrf24l01_clear_ce();
}

//powers down the 24L01
//this function allows the user to set the contents of the CONFIG register, but the function
//  clears the PWR_UP bit in the CONFIG register, so the user does not need to.
void nrf24l01_power_down_param(unsigned char config)
{
		config &= (~nrf24l01_CONFIG_PWR_UP);

		nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);

		nrf24l01_clear_ce();
}
```

```
//sets up the 24L01 as a receiver with all necessary delays
//this function takes the existing contents of the CONFIG register and sets the PRIM_RX
//  bit in the CONFIG register.
//if the argument rx_active_mode is false, the receiver will remain in standby mode
//  and not monitor for packets.  If the argument is true, the CE pin will be set
//  and the 24L01 will monitor for packets.
//note: if the read value of the CONFIG register already has the PRIM_RX bit set, this function
//  exits in order to not make an unecessary register write.
void nrf24l01_set_as_rx(bool rx_active_mode)
{
	unsigned char config;
	unsigned char status;

	status = nrf24l01_read_register(0, &config, 1);

	if((config & nrf24l01_CONFIG_PRIM_RX) != 0)
		return;

	config |= nrf24l01_CONFIG_PRIM_RX;

	nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);

	if(rx_active_mode != false)
		nrf24l01_set_ce();
	else
		nrf24l01_clear_ce();
}

//sets up the 24L01 as a receiver with all necessary delays
//this function allows the user to set the contents of the CONFIG register, but the function
//  sets the PRIM_RX bit in the CONFIG register, so the user does not need to.
//if the argument rx_active_mode is false, the receiver will remain in standby mode
//  and not monitor for packets.  If the argument is true, the CE pin will be set
//  and the 24L01 will monitor for packets.
void nrf24l01_set_as_rx_param(bool rx_active_mode, unsigned char config)
{
	config |= nrf24l01_CONFIG_PRIM_RX;
	if((config & nrf24l01_CONFIG_PWR_UP) != 0)
		nrf24l01_power_up_param(rx_active_mode, config);
	else
		nrf24l01_power_down_param(config);
}

//takes a 24L01 that is already in RX standby mode and puts it in active RX mode
void nrf24l01_rx_standby_to_active()
{
	nrf24l01_set_ce();
}

//takes a 24L01 that is already in active RX mode and puts it in RX standy mode
void nrf24l01_rx_active_to_standby()
{
	nrf24l01_clear_ce();
}

//sets up the 24L01 as a transmitter
//this function takes the existing contents of the CONFIG register and simply
//  clears the PRIM_RX bit in the CONFIG register.
//note: if the read value of the CONFIG register already has the PRIM_RX bit cleared, this
//  function exits in order to not make an unecessary register write.
void nrf24l01_set_as_tx()
{
	unsigned char config;

	nrf24l01_read_register(nrf24l01_CONFIG, &config, 1);

	if((config & nrf24l01_CONFIG_PRIM_RX) == 0)
		return;

	config &= (~nrf24l01_CONFIG_PRIM_RX);

	nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);

	nrf24l01_clear_ce();
}
```

```
//sets up the 24L01 as a transmitter
//this function allows the user to set the contents of the CONFIG register, but the function
//  clears the PRIM_RX bit in the CONFIG register, so the user does not need to.
void nrf24l01_set_as_tx_param(unsigned char config)
{
	config &= ~(nrf24l01_CONFIG_PRIM_RX);

	if((config & nrf24l01_CONFIG_PWR_UP) != 0)
		nrf24l01_power_up_param(false, config);
	else
		nrf24l01_power_down_param(config);
}

//executes the W_REGISTER SPI operation
//unsigned char regnumber indicates the register number assigned by the nrf24l01
specification.
//  For regnumber values, see section titled "register definitions" in nrf24l01.h.
//unsigned char * data should be of size 1 for all register writes except for RX_ADDR_P0,
RX_ADDR_P1,
//	and TX_ADDR.  The size of data should be set according to the user-specified size of the
address
//  length for the register the address is being sent to.
//unsigned int len is always the size of unsigned char * data.  For example, if data is declared as
//  data[6], len should equal 6.
//returns the value of the STATUS register
unsigned char nrf24l01_write_register(unsigned char regnumber, unsigned char * data,
unsigned int len)
{
	return nrf24l01_execute_command(nrf24l01_W_REGISTER | (regnumber &
nrf24l01_W_REGISTER_DATA), data, len, false);
}

//executes the R_REGISTER SPI operation
//unsigned char regnumber indicates the register number assigned by the nrf24l01
specification.
//  For regnumber values, see section titled "register definitions" in nrf24l01.h.
```

```
//unsigned char * data should be of size 1 for all register writes except for RX_ADDR_P0,
RX_ADDR_P1,
//	and TX_ADDR.  The size of data should be set according to the user-specified size of the
address
//  length for the register the address is being read from.
//unsigned int len is always the size of unsigned char * data.  For example, if data is declared as
//  data[6], len = 6.
//returns the value of the STATUS register
unsigned char nrf24l01_read_register(unsigned char regnumber, unsigned char * data,
unsigned int len)
{
	return nrf24l01_execute_command(regnumber & nrf24l01_R_REGISTER_DATA, data,
len, true);
}

//executes the W_TX_PAYLOAD operation
//unsigned char * data is the actual payload to be sent to the nrf24l01.
//unsigned int len is the length of the payload being sent (this should be sized
//	according to the payload length specified by the receiving nrf24l01).
//if bool transmit is true, the nrf24l01 immediately transmits the data in the payload.
//	if false, the user must use the nrf24l01_transmit() function to send the payload.
//returns the value of the STATUS register
unsigned char nrf24l01_write_tx_payload(unsigned char * data, unsigned int len, bool transmit)
{
	unsigned char status;

	status = nrf24l01_execute_command(nrf24l01_W_TX_PAYLOAD, data, len, false);

	if(transmit == true)
		nrf24l01_transmit();

	return status;
}

//executes the R_RX_PAYLOAD instruction
//unsigned char * data is the actual payload that has been received by the nrf24l01.
//	The user must size data according to the payload width specified to the nrf24l01.
//	This variable is filled by this function, so individual byte values need not be
```

```
//	initialized by the user.
//unsigned int len is the length of the payload being clocked out of the nrf24l01 (this
//	should be sized according to the payload length specified to the nrf24l01).
//returns the value of the STATUS register
unsigned char nrf24l01_read_rx_payload(unsigned char * data, unsigned int len)
{
	unsigned char status;

	nrf24l01_clear_ce();
	status = nrf24l01_execute_command(nrf24l01_R_RX_PAYLOAD, data, len, true);
	nrf24l01_set_ce();

	return status;
}

//executes the FLUSH_TX SPI operation
//this funciton empties the contents of the TX FIFO
//returns the value of the STATUS register
unsigned char nrf24l01_flush_tx()
{
	return nrf24l01_execute_command(nrf24l01_FLUSH_TX, NULL, 0, true);
}

//executes the FLUSH_RX SPI operation
//this funciton empties the contents of the RX FIFO
//returns the value of the STATUS register
unsigned char nrf24l01_flush_rx()
{
	return nrf24l01_execute_command(nrf24l01_FLUSH_RX, NULL, 0, true);
}

//executes the REUSE_TX_PL SPI operation
//this funciton allows the user to constantly send a packet repeatedly when issued.
//returns the value of the STATUS register
unsigned char nrf24l01_reuse_tx_pl()
{
	return nrf24l01_execute_command(nrf24l01_REUSE_TX_PL, NULL, 0, true);
}
```

```
//executes the FLUSH_TX SPI operation
//this funciton does nothing
//returns the value of the STATUS register
unsigned char nrf24l01_nop()
{
	return nrf24l01_execute_command(nrf24l01_NOP, NULL, 0, true);
}

//transmits the current tx payload
void nrf24l01_transmit()
{
	nrf24l01_set_ce();
	delay_us(10);
	nrf24l01_clear_ce();
}

//clears the pin on the host microcontroller that is attached to the 24l01's CE pin
void nrf24l01_clear_ce()
{
	nrf24l01_CE_IOREGISTER &= ~nrf24l01_CE_PINMASK;
}

//sets the pin on the host microcontroller that is attached to the 24l01's CE pin
void nrf24l01_set_ce()
{
	nrf24l01_CE_IOREGISTER |= nrf24l01_CE_PINMASK;
}

//returns true if CE is high, false if not
bool nrf24l01_ce_pin_active()
{
	if((nrf24l01_CE_IOREGISTER & nrf24l01_CE_PINMASK) != 0)
		return true;
	else
		return false;
}
```

```
//sets the pin on the host microcontroller that is attached to the 24l01's CSN pin
void nrf24l01_clear_csn()
{
	nrf24l01_CSN_IOREGISTER &= ~nrf24l01_CSN_PINMASK;
}

//clears the pin on the host microcontroller that is attached to the 24l01's CSN pin
void nrf24l01_set_csn()
{
	nrf24l01_CSN_IOREGISTER |= nrf24l01_CSN_PINMASK;
}

//returns true if CSN is high, false if not
bool nrf24l01_csn_pin_active()
{
	if((nrf24l01_CSN_IOREGISTER & nrf24l01_CSN_PINMASK) != 0)
		return true;
	else
		return false;
}

//sets the TX address in the TX_ADDR register
//unsigned char * address is the actual address to be used.  It should be sized
//	according to the tx_addr length specified to the nrf24l01.
//unsigned int len is the length of the address.  Its value should be specified
//	according to the tx_addr length specified to the nrf24l01.
void nrf24l01_set_tx_addr(unsigned char * address, unsigned int len)
{
	nrf24l01_write_register(nrf24l01_TX_ADDR, address, len);
}

//sets the RX address in the RX_ADDR register that is offset by rxpipenum
//unsigned char * address is the actual address to be used.  It should be sized
//	according to the rx_addr length that is being filled.
//unsigned int len is the length of the address.  Its value should be specified
//	according to the rx_addr length specified to the nrf24l01.
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
```

```
//  does nothing.
void nrf24l01_set_rx_addr(unsigned char * address, unsigned int len, unsigned char rxpipenum)
{
	if(rxpipenum > 5)
		return;

	nrf24l01_write_register(nrf24l01_RX_ADDR_P0 + rxpipenum, address, len);
}

//sets the RX payload width on the pipe offset by rxpipenum
//unsigned char payloadwidth is the length of the payload for the pipe referenced inFDelay
//  rxpipenum.  It must be less than or equal to 32.  If an invalid payload width is
//  specified, the function does nothing.
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  does nothing.
void nrf24l01_set_rx_pw(unsigned char payloadwidth, unsigned char rxpipenum)
{
	if((rxpipenum > 5) || (payloadwidth > 32))
		return;

	nrf24l01_write_register(nrf24l01_RX_PW_P0 + rxpipenum, &payloadwidth, 1);
}

//gets the RX payload width on the pipe offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  does nothing.
unsigned char nrf24l01_get_rx_pw(unsigned char rxpipenum)
{
	unsigned char data;

	if((rxpipenum > 5))
		return 0;

	nrf24l01_read_register(nrf24l01_RX_PW_P0 + rxpipenum, &data, 1);

	return data;
}

//returns the value of the CONFIG register
unsigned char nrf24l01_get_config()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_CONFIG, &data, 1);

	return data;
}

//sets the value of the CONFIG register
void nrf24l01_set_config(unsigned char config)
{
	nrf24l01_write_register(nrf24l01_CONFIG, &config, 1);
}

//returns the current RF channel in RF_CH register
unsigned char nrf24l01_get_rf_ch()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_RF_CH, &data, 1);

	return data;
}

//unsigned char channel is the channel to be changed to.
void nrf24l01_set_rf_ch(unsigned char channel)
{
	unsigned char data;

	data = channel & ~nrf24l01_RF_CH_RESERVED;

	nrf24l01_write_register(nrf24l01_RF_CH, &data, 1);
}
```

```
//returns the value of the OBSERVE_TX register
unsigned char nrf24l01_get_observe_tx()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_OBSERVE_TX, &data, 1);

	return data;
}

//returns the current PLOS_CNT value in OBSERVE_TX register
unsigned char nrf24l01_get_plos_cnt()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_OBSERVE_TX, &data, 1);

	return ((data & nrf24l01_OBSERVE_TX_PLOS_CNT) >> 4);
}

//clears the PLOS_CNT field of the OBSERVE_TX register
//this function makes a read of the current value of RF_CH and
//  simply writes it back to the register, clearing PLOS_CNT
void nrf24l01_clear_plos_cnt()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_RF_CH, &data, 1);
	nrf24l01_write_register(nrf24l01_RF_CH, &data, 1);
}

//clears the PLOS_CNT field of the OBSERVE_TX register
//this function allows the user to set the RF_CH register by using
//  the argument in the function during the PLOS_CNT clearing process
void nrf24l01_clear_plos_cnt_param(unsigned char rf_ch)
{
	nrf24l01_write_register(nrf24l01_RF_CH, &rf_ch, 1);
}
```

```
//returns the current ARC_CNT value in OBSERVE_TX register
unsigned char nrf24l01_get_arc_cnt()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_OBSERVE_TX, &data, 1);

	return (data & nrf24l01_OBSERVE_TX_ARC_CNT);
}

//returns true if auto-ack is enabled on the pipe that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  returns false.
bool nrf24l01_aa_enabled(unsigned char rxpipenum)
{
	unsigned char data;

	if(rxpipenum > 5)
		return false;

	nrf24l01_read_register(nrf24l01_EN_AA, &data, 1);

	return (data & (0x01 << rxpipenum));
}

//enables auto-ack is enabled on the pipe that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	does nothing.
void nrf24l01_aa_enable(unsigned char rxpipenum)
{
	unsigned char data;

	if(rxpipenum > 5)
		return;

	nrf24l01_read_register(nrf24l01_EN_AA, &data, 1);
	if((data & (0x01 << rxpipenum)) != 0)
		return;

	data |= 0x01 << rxpipenum;

	nrf24l01_write_register(nrf24l01_EN_AA, &data, 1);
}

//disables auto-ack is enabled on the pipe that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	does nothing.
void nrf24l01_aa_disable(unsigned char rxpipenum)
{
	unsigned char data;

	if(rxpipenum > 5)
		return;

	nrf24l01_read_register(nrf24l01_EN_AA, &data, 1);

	if((data & (0x01 << rxpipenum)) == 0)
		return;

	data &= ~(0x01 << rxpipenum);

	nrf24l01_write_register(nrf24l01_EN_AA, &data, 1);
}

//returns true if the pipe is enabled that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  returns false.
bool nrf24l01_rx_pipe_enabled(unsigned char rxpipenum)
{
	unsigned char data;

	if((rxpipenum > 5))
```

```
		return false;

	nrf24l01_read_register(nrf24l01_EN_RXADDR, &data, 1);

	return (data & (0x01 << rxpipenum));
}

//enables the pipe that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  does nothing.
void nrf24l01_rx_pipe_enable(unsigned char rxpipenum)
{
	unsigned char data;

	if(rxpipenum > 5)
		return;

	nrf24l01_read_register(nrf24l01_EN_RXADDR, &data, 1);

	if((data & (0x01 << rxpipenum)) != 0)
		return;

	data |= 0x01 << rxpipenum;

	nrf24l01_write_register(nrf24l01_EN_RXADDR, &data, 1);
}

//disables the pipe that is offset by rxpipenum
//unsigned char rxpipenum is the pipe number (zero to five) whose address is being
//	specified.  If an invalid address (greater than five) is supplied, the function
//  does nothing.
void nrf24l01_rx_pipe_disable(unsigned char rxpipenum)
{
	unsigned char data;

	if(rxpipenum > 5)
		return;
	nrf24l01_read_register(nrf24l01_EN_RXADDR, &data, 1);

	if((data & (0x01 << rxpipenum)) == 0)
		return;

	data &= ~(0x01 << rxpipenum);

	nrf24l01_write_register(nrf24l01_EN_RXADDR, &data, 1);
}

//returns the status of the CD register (true if carrier detect [CD] is
//  active, false if not)
bool nrf24l01_cd_active()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_CD, &data, 1);

	return data;
}

//returns the value of the FIFO_STATUS register
unsigned char nrf24l01_get_fifo_status()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return data;
}

//return the value of the status register
unsigned char nrf24l01_get_status()
{
	return nrf24l01_nop();
}
```

```
//returns true if TX_REUSE bit in FIFO_STATUS register is set, false otherwise
bool nrf24l01_fifo_tx_reuse()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return (bool)(data & nrf24l01_FIFO_STATUS_TX_REUSE);
}

//returns true if TX_FULL bit in FIFO_STATUS register is set, false otherwise
bool nrf24l01_fifo_tx_full()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return (bool)(data & nrf24l01_FIFO_STATUS_TX_FULL);
}

//returns true if TX_EMPTY bit in FIFO_STATUS register is set, false otherwise
bool nrf24l01_fifo_tx_empty()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return (bool)(data & nrf24l01_FIFO_STATUS_TX_EMPTY);
}

//returns true if RX_FULL bit in FIFO_STATUS register is set, false otherwise
bool nrf24l01_fifo_rx_full()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return (bool)(data & nrf24l01_FIFO_STATUS_RX_FULL);
}

//returns true if RX_EMPTYE bit in FIFO_STATUS register is set, false otherwise
bool nrf24l01_fifo_rx_empty()
{
	unsigned char data;

	nrf24l01_read_register(nrf24l01_FIFO_STATUS, &data, 1);

	return (bool)(data & nrf24l01_FIFO_STATUS_RX_EMPTY);
}

//returns true if IRQ pin is low, false otherwise
bool nrf24l01_irq_pin_active()
{
	if((nrf24l01_IRQ_IOREGISTER & nrf24l01_IRQ_PINMASK) != 0)
		return false;
	else
		return true;
}

//returns true if RX_DR interrupt is active, false otherwise
bool nrf24l01_irq_rx_dr_active()
{
	return (nrf24l01_get_status() & nrf24l01_STATUS_RX_DR);
}

//returns true if TX_DS interrupt is active, false otherwise
bool nrf24l01_irq_tx_ds_active()
{
	return (nrf24l01_get_status() & nrf24l01_STATUS_TX_DS);
}

//returns true if MAX_RT interrupt is active, false otherwise
bool nrf24l01_irq_max_rt_active()
{
	return (nrf24l01_get_status() & nrf24l01_STATUS_MAX_RT);
}
```

```
//clear all interrupts in the status register
void nrf24l01_irq_clear_all()
{
	unsigned char data = nrf24l01_STATUS_RX_DR | nrf24l01_STATUS_TX_DS |
nrf24l01_STATUS_MAX_RT;

	nrf24l01_write_register(nrf24l01_STATUS, &data, 1);
}

//clears only the RX_DR interrupt
void nrf24l01_irq_clear_rx_dr()
{
	unsigned char data = nrf24l01_STATUS_RX_DR;

	nrf24l01_write_register(nrf24l01_STATUS, &data, 1);
}

//clears only the TX_DS interrupt
void nrf24l01_irq_clear_tx_ds()
{
	unsigned char data = nrf24l01_STATUS_TX_DS;

	nrf24l01_write_register(nrf24l01_STATUS, &data, 1);
}

//clears only the MAX_RT interrupt
void nrf24l01_irq_clear_max_rt()
{
	unsigned char data = nrf24l01_STATUS_MAX_RT;

	nrf24l01_write_register(nrf24l01_STATUS, &data, 1);
}

//returns the current pipe in the 24L01's STATUS register
unsigned char nrf24l01_get_rx_pipe()
{
	return nrf24l01_get_rx_pipe_from_status(nrf24l01_get_status());
}

unsigned char nrf24l01_get_rx_pipe_from_status(unsigned char status)
{
	return ((status & 0xE) >> 1);
}

//flush both fifos and clear interrupts
void nrf24l01_clear_flush()
{
	nrf24l01_flush_rx();
	nrf24l01_flush_tx();
	nrf24l01_irq_clear_all();
}

//unsigned char * data must be at least 35 bytes long
void nrf24l01_get_all_registers(unsigned char * data)
{
	unsigned int outer;
	unsigned int inner;
	unsigned int dataloc = 0;
	unsigned char buffer[5];

	for(outer = 0; outer <= 0x17; outer++)
	{
		nrf24l01_read_register(outer, buffer, 5);

		for(inner = 0; inner < 5; inner++)
		{
			if(inner >= 1 && (outer != 0x0A && outer != 0x0B && outer != 0x10))
				break;

			data[dataloc] = buffer[inner];
			dataloc++;
		}
	}
}
```

```
//low-level spi send function for library use
//the user should not call this function directly, but rather use one of the 8 SPI data instructions
unsigned char nrf24l01_execute_command(unsigned char instruction, unsigned char * data,
unsigned int len, bool copydata)
{
	unsigned char status;

	nrf24l01_clear_csn();

	status = instruction;
	nrf24l01_spi_send_read(&status, 1, true);
	nrf24l01_spi_send_read(data, len, copydata);

	nrf24l01_set_csn();

	return status;
}

//low-level spi send function for library use
//the user should not call this function directly, but rather use one of the 8 SPI data instructions
void nrf24l01_spi_send_read(unsigned char * data, unsigned int len, bool copydata)
{
	unsigned int count;
	unsigned char tempbyte;

	for(count = 0; count < len; count++)
	{
		if(copydata != false)
			data[count] = spi_send_read_byte(data[count]);
		else
		{
			tempbyte = data[count];
			spi_send_read_byte(tempbyte);
		}
	}
}

unsigned char spi_send_read_byte(unsigned char byte)
```

```
{
        SPI2STATbits.SPIROV = 0;

        SPI2BUF = byte;                                 // Transmit Read command
        while(SPI2STATbits.SPITBF)                  // Wait for command to be transmitted
                {asm("clrwdt");}

        SPI2BUF = 0xAA;
        while(!SPI2STATbits.SPIRBF)                 // Wait for buffer to fill up
                {asm("clrwdt");}

        return  SPI2BUF;
}



/****************************************************************
File Name:              delay.C
Description:        Functions for creating a cycle based delay
****************************************************************/
#include <p33fxxxx.h>

void Delay_cycles(long cycles)
{
        long i;
        for(i=0; i<cycles; i++)
        asm("clrwdt");
}

void Delay_us(long u_secs)
{
        long i;
        for(i=0; i<u_secs; i++)
        Delay_cycles(5);
}
```

## HEADER FILES

```
//*********************************************************************
//        File: hd44780_LCD.h
//        Author: M. H. Longshore
//        Description: Define the HW and software interface for a 4-bit mode
//            HD44780 compatible LCD driver
//        Date: 11/05/2010

#ifndef _HD44780_LCD_H
#define _HD44780_LCD_H

#include <delay.h>

// LCD DEFINITIONS     ****************************************

//Data Pins
#define DATA_PIN_7          LATBbits.LATB9
#define DATA_PIN_6          LATBbits.LATB8
#define DATA_PIN_5          LATBbits.LATB11
#define DATA_PIN_4          LATBbits.LATB10

#define READ_PIN_7          PORTBbits.RB9
#define READ_PIN_6          PORTBbits.RB8
#define READ_PIN_5          PORTBbits.RB11
#define READ_PIN_4          PORTBbits.RB10


#define TRIS_DATA_PIN_7          TRISBbits.TRISB9
#define TRIS_DATA_PIN_6          TRISBbits.TRISB8
#define TRIS_DATA_PIN_5          TRISBbits.TRISB11
```

```
#define TRIS_DATA_PIN_4                 TRISBbits.TRISB10


#define E_PIN                   LATDbits.LATD8
#define RW_PIN                  LATDbits.LATD9
#define RS_PIN                  LATBbits.LATB14

#define TRIS_E                  TRISDbits.TRISD8
#define TRIS_RW                 TRISDbits.TRISD9
#define TRIS_RS                 TRISBbits.TRISB14


// Commands
#define DON                     0x0F    //Turn the display on
#define DOFF                    0x0B    //Turn the display off
#define CURSOR_ON               0x0F    //Turn display on with cursor on
#define CURSOR_OFF              0x0C    //Turn display on with cursor off
#define BLINK_ON                0x0F    //Turn cursor blink on
#define BLINK_OFF               0x0E    //Turn cursor blink off
#define CLEAR               0x01        //Clear the display
#define RETURN_CURSOR_HOME      0x02    //Set cursor to the home position
#define INCR_MODE               0x07    //Set increment address after write
#define DECR_MODE               0x05    //Set decrement address after write
#define ENTIRE_SHIFT_OFF        0x06    //Turn screen shift off
#define ENTIRE_SHIFT_ON                 0x07    //Turn screen shift on
#define SHIFT_CUR_LEFT          0x13    //Cursor shifts left
#define SHIFT_CUR_RIGHT                 0x17    //Cursor shifts right
#define SHIFT_DISP_LEFT         0x1B   //Shift display left
#define SHIFT_DISP_RIGHT                0x1F    //Shift display right
#define FOUR_BIT                0x2F    //Four-bit mode
#define EIGHT_BIT               0x3F    //Eight-bit mode
#define SINGLE_LINE             0x37    //Single line mode
#define TWO_LINE                0x3F    //Double line mode

//Delay Definitions for Fcy = 80MHz
#define Delay_80ms              300000
#define Delay_5_5ms                     20000
#define Delay_290us             2000
```

```
#define Delay_880ns             4

//***** Function Prototypes *************
//Setup the LCD for 4-bit mode
void Init_LCD(void) __attribute__ ((section (".libperi")));
//Write command to the LCD
void WriteLCD_CMD(char) __attribute__ ((section (".libperi")));
//Write data to LCD
void WriteLCD_Data(char) __attribute__ ((section (".libperi")));
//Set LCD address pointer
void WriteLCD_DDR(char) __attribute__ ((section (".libperi")));
//Write a string of characters to the LCD
void WriteLCD_String(char *) __attribute__ ((section (".libperi")));

#endif
```

```
/****************************************************************************
*
*
* File: nrf24l01.h
*
* Copyright S. Brennen Ball, 2006-2007
*
* The author provides no guarantees, warantees, or promises, implied or
*        otherwise.  By using this software you agree to indemnify the author
*        of any damages incurred by using it.
*
*****************************************************************************/

#ifndef NRF24L01_H_
#define NRF24L01_H_

#include <stddef.h>

#ifndef bool
#define bool unsigned char
#endif
#ifndef false
#define false 0
#endif
#ifndef true
#define true !false
#endif

/////////////////////////////////////////////////////////////////////////////////
// SPI function requirements
//
// The user must define a function to send one byte of data and also return the
//   resulting byte of data data through the SPI port. The function used here
//   has the function prototype
```

```
//
//            unsigned char spi_send_read_byte(unsigned char byte);
//
// This function should take the argument unsigned char byte and send it through
//   the SPI port to the 24L01.  Then, it should wait until the 24L01 has returned
//   its response over SPI.  This received byte should be the return value of the
//   function.
//
// You should also change the include file name below to whatever the name of your
//   SPI include file is.
////////////////////////////////////////////////////////////////////////////////

#define spi_send_read_byte(byte)    spi2_send_read_byte(byte)


////////////////////////////////////////////////////////////////////////////////
// Delay function requirements
//
// The user must define a function that delays for the specified number of
//         microseconds. This function needs to be as precise as possible, and the use
//   of a timer module within your microcontroller is highly recommended. The
//   function used here has the prototype
//
//            void delay_us(unsigned int microseconds);
//
// You should also change the include file name below to whatever the name of your
//   delay include file is.
////////////////////////////////////////////////////////////////////////////////
#include <delay.h>
#define delay_us(microseconds)                  Delay_us(microseconds)


////////////////////////////////////////////////////////////////////////////////
// IO pin definitions
//
// Below you will find several definitions and includes.  The first is an #include
//   for your microcontroller's include file to allow you to use register names
//   rather than numbers.  The next three are to allow you to control the pins on
//   the 24L01 that aren't automatically handled by SPI.  These are CE, CSN, and
//   IRQ.
//
// The general format of these defines is a define for the IO register the pin is
//   attached to.  The second define is a mask for the pin.  For example, say that
//   your CE pin is tied to an IO port with the register name IOPORT1. Also, let's
//   say that the IO port is 8-bits wide, and you have attached the pin to pin 0 of
//   the port.  Then your define would look like this:
//
//   #define nrf24l01_CE_IOREGISTER          IOPORT1
//   #define nrf24l01_CE_PINMASK        0x01
//
// If you have defines in your include file for individual IO pins, you could use
//   this define in this file, as well.  Using the previous example, assume that in
//   your microcontroller's include file, pin 0 of IOPORT1 has a define like this
//
//   #define IOPORT1_PIN0     0x01
//
// Then, you could make your defines for the CE pin in this file look like this:
//
//   #define nrf24l01_CE_IOREGISTER          IOPORT1
//   #define nrf24l01_CE_PINMASK        IOPORT1_PIN0
//
// You should also change the include file name below to whatever the name of your
//   processor's register definition include file is.
/////////////////////////////////////////////////////////////////////////////////
#include <p33fxxxx.h>

//defines for uC pins CE pin is connected to
//This is used so that the routines can send TX payload data and
//properly initialize the nrf24l01 in TX and RX states.
//Change these definitions (and then recompile) to suit your particular application.
#define nrf24l01_CE_IOREGISTER          PORTF
#define nrf24l01_CE_PINMASK             0x0001
//#define nrf24l01_CE                   LATFbits.LATF0

//defines for uC pins CSN pin is connected to
//This is used so that the routines can send properly operate the SPI interface
```

```
// on the nrf24l01.
//Change these definitions (and then recompile) to suit your particular application.
#define nrf24l01_CSN_IOREGISTER             PORTG
#define nrf24l01_CSN_PINMASK                0x0200
//#define nrf24l01_CSN                      LATGbits.LATG9

//defines for uC pins IRQ pin is connected to
//This is used so that the routines can poll for IRQ or create an ISR.
//Change these definitions (and then recompile) to suit your particular application.
#define nrf24l01_IRQ_IOREGISTER             PORTF
#define nrf24l01_IRQ_PINMASK                0x0002
//#define nrf24l01_IRQ                      PORTFbits.RF1


////////////////////////////////////////////////////////////////////////////////////
// SPI commands
//
// The following are defines for all of the commands and data masks on the SPI
//   interface.
////////////////////////////////////////////////////////////////////////////////////
//SPI command defines
#define nrf24l01_R_REGISTER                 0x00
#define nrf24l01_W_REGISTER                 0x20
#define nrf24l01_R_RX_PAYLOAD               0x61
#define nrf24l01_W_TX_PAYLOAD               0xA0
#define nrf24l01_FLUSH_TX           0xE1
#define nrf24l01_FLUSH_RX           0xE2
#define nrf24l01_REUSE_TX_PL                0xE3
#define nrf24l01_NOP                        0xFF

//SPI command data mask defines
#define nrf24l01_R_REGISTER_DATA0x1F
#define nrf24l01_W_REGISTER_DATA            0x1F

////////////////////////////////////////////////////////////////////////////////////
// Register definitions
//
// Below are the defines for each register's address in the 24L01.
```

```
////////////////////////////////////////////////////////////////////////////////
#define nrf24l01_CONFIG              0x00
#define nrf24l01_EN_AA               0x01
#define nrf24l01_EN_RXADDR           0x02
#define nrf24l01_SETUP_AW        0x03
#define nrf24l01_SETUP_RETR          0x04
#define nrf24l01_RF_CH               0x05
#define nrf24l01_RF_SETUP        0x06
#define nrf24l01_STATUS              0x07
#define nrf24l01_OBSERVE_TX          0x08
#define nrf24l01_CD              0x09
#define nrf24l01_RX_ADDR_P0          0x0A
#define nrf24l01_RX_ADDR_P1          0x0B
#define nrf24l01_RX_ADDR_P2          0x0C
#define nrf24l01_RX_ADDR_P3          0x0D
#define nrf24l01_RX_ADDR_P4          0x0E
#define nrf24l01_RX_ADDR_P5          0x0F
#define nrf24l01_TX_ADDR         0x10
#define nrf24l01_RX_PW_P0        0x11
#define nrf24l01_RX_PW_P1        0x12
#define nrf24l01_RX_PW_P2        0x13
#define nrf24l01_RX_PW_P3        0x14
#define nrf24l01_RX_PW_P4        0x15
#define nrf24l01_RX_PW_P5        0x16
#define nrf24l01_FIFO_STATUS         0x17

////////////////////////////////////////////////////////////////////////////////
// Default register values
//
// Below are the defines for each register's default value in the 24L01. Multi-byte
//   registers use notation B<X>, where "B" represents "byte" and <X> is the byte
//   number.
////////////////////////////////////////////////////////////////////////////////
#define nrf24l01_CONFIG_DEFAULT_VAL          0x08
#define nrf24l01_EN_AA_DEFAULT_VAL           0x3F
#define nrf24l01_EN_RXADDR_DEFAULT_VAL           0x03
#define nrf24l01_SETUP_AW_DEFAULT_VAL            0x03
#define nrf24l01_SETUP_RETR_DEFAULT_VAL          0x03
```

```
#define nrf24l01_RF_CH_DEFAULT_VAL                0x02
#define nrf24l01_RF_SETUP_DEFAULT_VAL             0x0F
#define nrf24l01_STATUS_DEFAULT_VAL               0x0E
#define nrf24l01_OBSERVE_TX_DEFAULT_VAL                   0x00
#define nrf24l01_CD_DEFAULT_VAL                   0x00

#define nrf24l01_RX_ADDR_P0_DEFAULT_VAL                   0xC0
#define nrf24l01_RX_ADDR_P0_B0_DEFAULT_VAL  0x61
#define nrf24l01_RX_ADDR_P0_B1_DEFAULT_VAL  0x67
#define nrf24l01_RX_ADDR_P0_B2_DEFAULT_VAL  0x6F
#define nrf24l01_RX_ADDR_P0_B3_DEFAULT_VAL  0x6E
#define nrf24l01_RX_ADDR_P0_B4_DEFAULT_VAL  0x79

#define nrf24l01_RX_ADDR_P1_DEFAULT_VAL                   0xC1
#define nrf24l01_RX_ADDR_P1_B0_DEFAULT_VAL  0x6D
#define nrf24l01_RX_ADDR_P1_B1_DEFAULT_VAL  0x75
#define nrf24l01_RX_ADDR_P1_B2_DEFAULT_VAL  0x73
#define nrf24l01_RX_ADDR_P1_B3_DEFAULT_VAL  0x65
#define nrf24l01_RX_ADDR_P1_B4_DEFAULT_VAL  0x31

#define nrf24l01_RX_ADDR_P2_DEFAULT_VAL                   0xC2
#define nrf24l01_RX_ADDR_P2_B0_DEFAULT_VAL  0x6D
#define nrf24l01_RX_ADDR_P2_B1_DEFAULT_VAL  0x75
#define nrf24l01_RX_ADDR_P2_B2_DEFAULT_VAL  0x73
#define nrf24l01_RX_ADDR_P2_B3_DEFAULT_VAL  0x65
#define nrf24l01_RX_ADDR_P2_B4_DEFAULT_VAL  0x34

#define nrf24l01_RX_ADDR_P3_DEFAULT_VAL                   0xC3
#define nrf24l01_RX_ADDR_P3_B0_DEFAULT_VAL  0x6D
#define nrf24l01_RX_ADDR_P3_B1_DEFAULT_VAL  0x75
#define nrf24l01_RX_ADDR_P3_B2_DEFAULT_VAL  0x73
#define nrf24l01_RX_ADDR_P3_B3_DEFAULT_VAL  0x65
#define nrf24l01_RX_ADDR_P3_B4_DEFAULT_VAL  0x34

#define nrf24l01_RX_ADDR_P4_DEFAULT_VAL                   0xC5

#define nrf24l01_RX_ADDR_P5_DEFAULT_VAL                   0xC6
```

```
#define nrf24l01_TX_ADDR_B0_DEFAULT_VAL                 0xE7
#define nrf24l01_TX_ADDR_B1_DEFAULT_VAL                 0xE7
#define nrf24l01_TX_ADDR_B2_DEFAULT_VAL                 0xE7
#define nrf24l01_TX_ADDR_B3_DEFAULT_VAL                 0xE7
#define nrf24l01_TX_ADDR_B4_DEFAULT_VAL                 0xE7

#define nrf24l01_RX_PW_P0_DEFAULT_VAL           0x00
#define nrf24l01_RX_PW_P1_DEFAULT_VAL           0x00
#define nrf24l01_RX_PW_P2_DEFAULT_VAL           0x00
#define nrf24l01_RX_PW_P3_DEFAULT_VAL           0x00
#define nrf24l01_RX_PW_P4_DEFAULT_VAL           0x00
#define nrf24l01_RX_PW_P5_DEFAULT_VAL           0x00

#define nrf24l01_FIFO_STATUS_DEFAULT_VAL        0x11

////////////////////////////////////////////////////////////////////////////////////
// Register bitwise definitions
//
// Below are the defines for each register's bitwise fields in the 24L01.
////////////////////////////////////////////////////////////////////////////////////
//CONFIG register bitwise definitions
#define nrf24l01_CONFIG_RESERVED                0x80
#define         nrf24l01_CONFIG_MASK_RX_DR              0x40
#define         nrf24l01_CONFIG_MASK_TX_DS              0x20
#define         nrf24l01_CONFIG_MASK_MAX_RT             0x10
#define         nrf24l01_CONFIG_EN_CRC                  0x08
#define         nrf24l01_CONFIG_CRCO                    0x04
#define         nrf24l01_CONFIG_PWR_UP                  0x02
#define         nrf24l01_CONFIG_PRIM_RX                 0x01

//EN_AA register bitwise definitions
#define nrf24l01_EN_AA_RESERVED                 0xC0
#define nrf24l01_EN_AA_ENAA_ALL                 0x3F
#define nrf24l01_EN_AA_ENAA_P5                  0x20
#define nrf24l01_EN_AA_ENAA_P4                  0x10
#define nrf24l01_EN_AA_ENAA_P3                  0x08
#define nrf24l01_EN_AA_ENAA_P2                  0x04
#define nrf24l01_EN_AA_ENAA_P1                  0x02
```

```
#define nrf24l01_EN_AA_ENAA_P0                  0x01
#define nrf24l01_EN_AA_ENAA_NONE                0x00

//EN_RXADDR register bitwise definitions
#define nrf24l01_EN_RXADDR_RESERVED             0xC0
#define nrf24l01_EN_RXADDR_ERX_ALL              0x3F
#define nrf24l01_EN_RXADDR_ERX_P5               0x20
#define nrf24l01_EN_RXADDR_ERX_P4               0x10
#define nrf24l01_EN_RXADDR_ERX_P3               0x08
#define nrf24l01_EN_RXADDR_ERX_P2               0x04
#define nrf24l01_EN_RXADDR_ERX_P1               0x02
#define nrf24l01_EN_RXADDR_ERX_P0               0x01
#define nrf24l01_EN_RXADDR_ERX_NONE             0x00

//SETUP_AW register bitwise definitions
#define nrf24l01_SETUP_AW_RESERVED              0xFC
#define nrf24l01_SETUP_AW               0x03
#define nrf24l01_SETUP_AW_5BYTES                0x03
#define nrf24l01_SETUP_AW_4BYTES                0x02
#define nrf24l01_SETUP_AW_3BYTES                0x01
#define nrf24l01_SETUP_AW_ILLEGAL               0x00

//SETUP_RETR register bitwise definitions
#define nrf24l01_SETUP_RETR_ARD                 0xF0
#define nrf24l01_SETUP_RETR_ARD_4000            0xF0
#define nrf24l01_SETUP_RETR_ARD_3750            0xE0
#define nrf24l01_SETUP_RETR_ARD_3500            0xD0
#define nrf24l01_SETUP_RETR_ARD_3250            0xC0
#define nrf24l01_SETUP_RETR_ARD_3000            0xB0
#define nrf24l01_SETUP_RETR_ARD_2750            0xA0
#define nrf24l01_SETUP_RETR_ARD_2500            0x90
#define nrf24l01_SETUP_RETR_ARD_2250            0x80
#define nrf24l01_SETUP_RETR_ARD_2000            0x70
#define nrf24l01_SETUP_RETR_ARD_1750            0x60
#define nrf24l01_SETUP_RETR_ARD_1500            0x50
#define nrf24l01_SETUP_RETR_ARD_1250            0x40
#define nrf24l01_SETUP_RETR_ARD_1000            0x30
#define nrf24l01_SETUP_RETR_ARD_750             0x20
```

```
#define nrf24l01_SETUP_RETR_ARD_500              0x10
#define nrf24l01_SETUP_RETR_ARD_250              0x00
#define nrf24l01_SETUP_RETR_ARC                  0x0F
#define nrf24l01_SETUP_RETR_ARC_15               0x0F
#define nrf24l01_SETUP_RETR_ARC_14               0x0E
#define nrf24l01_SETUP_RETR_ARC_13               0x0D
#define nrf24l01_SETUP_RETR_ARC_12               0x0C
#define nrf24l01_SETUP_RETR_ARC_11               0x0B
#define nrf24l01_SETUP_RETR_ARC_10               0x0A
#define nrf24l01_SETUP_RETR_ARC_9                0x09
#define nrf24l01_SETUP_RETR_ARC_8                0x08
#define nrf24l01_SETUP_RETR_ARC_7                0x07
#define nrf24l01_SETUP_RETR_ARC_6                0x06
#define nrf24l01_SETUP_RETR_ARC_5                0x05
#define nrf24l01_SETUP_RETR_ARC_4                0x04
#define nrf24l01_SETUP_RETR_ARC_3                0x03
#define nrf24l01_SETUP_RETR_ARC_2                0x02
#define nrf24l01_SETUP_RETR_ARC_1                0x01
#define nrf24l01_SETUP_RETR_ARC_0                0x00

//RF_CH register bitwise definitions
#define nrf24l01_RF_CH_RESERVED                  0x80

//RF_SETUP register bitwise definitions
#define nrf24l01_RF_SETUP_RESERVED               0xE0
#define nrf24l01_RF_SETUP_PLL_LOCK               0x10
#define nrf24l01_RF_SETUP_RF_DR                  0x08
#define nrf24l01_RF_SETUP_RF_PWR                 0x06
#define nrf24l01_RF_SETUP_RF_PWR_0               0x06
#define nrf24l01_RF_SETUP_RF_PWR_6               0x04
#define nrf24l01_RF_SETUP_RF_PWR_12              0x02
#define nrf24l01_RF_SETUP_RF_PWR_18              0x00
#define nrf24l01_RF_SETUP_LNA_HCURR              0x01

//STATUS register bitwise definitions
#define nrf24l01_STATUS_RESERVED                 0x80
#define nrf24l01_STATUS_RX_DR                    0x40
#define nrf24l01_STATUS_TX_DS                    0x20
```

```
#define nrf24l01_STATUS_MAX_RT                         0x10
#define nrf24l01_STATUS_RX_P_NO                        0x0E
#define nrf24l01_STATUS_RX_P_NO_RX_FIFO_NOT_EMPTY          0x0E
#define nrf24l01_STATUS_RX_P_NO_UNUSED                      0x0C
#define nrf24l01_STATUS_RX_P_NO_5                      0x0A
#define nrf24l01_STATUS_RX_P_NO_4                      0x08
#define nrf24l01_STATUS_RX_P_NO_3                      0x06
#define nrf24l01_STATUS_RX_P_NO_2                      0x04
#define nrf24l01_STATUS_RX_P_NO_1                      0x02
#define nrf24l01_STATUS_RX_P_NO_0                      0x00
#define nrf24l01_STATUS_TX_FULL                        0x01

//OBSERVE_TX register bitwise definitions
#define nrf24l01_OBSERVE_TX_PLOS_CNT                   0xF0
#define nrf24l01_OBSERVE_TX_ARC_CNT                    0x0F

//CD register bitwise definitions
#define nrf24l01_CD_RESERVED                           0xFE
#define nrf24l01_CD_CD                                 0x01

//RX_PW_P0 register bitwise definitions
#define nrf24l01_RX_PW_P0_RESERVED                     0xC0

//RX_PW_P0 register bitwise definitions
#define nrf24l01_RX_PW_P0_RESERVED                     0xC0

//RX_PW_P1 register bitwise definitions
#define nrf24l01_RX_PW_P1_RESERVED                     0xC0

//RX_PW_P2 register bitwise definitions
#define nrf24l01_RX_PW_P2_RESERVED                     0xC0

//RX_PW_P3 register bitwise definitions
#define nrf24l01_RX_PW_P3_RESERVED                     0xC0

//RX_PW_P4 register bitwise definitions
#define nrf24l01_RX_PW_P4_RESERVED                     0xC0
```

```
//RX_PW_P5 register bitwise definitions
#define nrf24l01_RX_PW_P5_RESERVED                0xC0

//FIFO_STATUS register bitwise definitions
#define nrf24l01_FIFO_STATUS_RESERVED             0x8C
#define nrf24l01_FIFO_STATUS_TX_REUSE             0x40
#define nrf24l01_FIFO_STATUS_TX_FULL              0x20
#define nrf24l01_FIFO_STATUS_TX_EMPTY             0x10
#define nrf24l01_FIFO_STATUS_RX_FULL              0x02
#define nrf24l01_FIFO_STATUS_RX_EMPTY             0x01

//////////////////////////////////////////////////////////////////////////////////
// Function declarations
//
// Below are all function definitions contained in the library.  Please see
//   nrf24l01.c for comments regarding the usage of each function.
//////////////////////////////////////////////////////////////////////////////////
//initialization functions
void nrf24l01_initialize(unsigned char config,
                         unsigned char opt_rx_standby_mode,
                         unsigned char en_aa,
                         unsigned char en_rxaddr,
                         unsigned char setup_aw,
                         unsigned char setup_retr,
                         unsigned char rf_ch,
                         unsigned char rf_setup,
                         unsigned char * rx_addr_p0,
                         unsigned char * rx_addr_p1,
                         unsigned char rx_addr_p2,
                         unsigned char rx_addr_p3,
                         unsigned char rx_addr_p4,
                         unsigned char rx_addr_p5,
                         unsigned char * tx_addr,
                         unsigned char rx_pw_p0,
                         unsigned char rx_pw_p1,
                         unsigned char rx_pw_p2,
                         unsigned char rx_pw_p3,
                         unsigned char rx_pw_p4,
                                        unsigned char rx_pw_p5);

void nrf24l01_initialize_debug(bool rx, unsigned char p0_payload_width, bool
enable_auto_ack);
void nrf24l01_initialize_debug_lite(bool rx, unsigned char p0_payload_width);

//power-up, power-down functions
void nrf24l01_power_up(bool rx_active_mode);
void nrf24l01_power_up_param(bool rx_active_mode, unsigned char config);
void nrf24l01_power_down(void);
void nrf24l01_power_down_param(unsigned char config);

//SPI commands defined by the spec
//for regnumber values, see section above titled "register definitions"
//all functions return the STATUS register
unsigned char nrf24l01_write_register(unsigned char regnumber, unsigned char * data,
unsigned int len);
unsigned char nrf24l01_read_register(unsigned char regnumber, unsigned char * data,
unsigned int len);
unsigned char nrf24l01_write_tx_payload(unsigned char * data, unsigned int len, bool
transmit);
unsigned char nrf24l01_read_rx_payload(unsigned char * data, unsigned int len);
unsigned char nrf24l01_flush_tx(void);
unsigned char nrf24l01_flush_rx(void);
unsigned char nrf24l01_reuse_tx_pl(void);
unsigned char nrf24l01_nop(void);

//RX/TX setting functions
void nrf24l01_set_as_rx(bool rx_active_mode);
void nrf24l01_set_as_rx_param(bool rx_active_mode, unsigned char config);
void nrf24l01_rx_standby_to_active(void);
void nrf24l01_rx_active_to_standby(void);
void nrf24l01_set_as_tx(void);
void nrf24l01_set_as_tx_param(unsigned char config);

//register-oriented get/set functions for commonly-used registers during operation
unsigned char nrf24l01_get_config(void);
void nrf24l01_set_config(unsigned char config);
```

```
unsigned char nrf24l01_get_rf_ch(void);
void nrf24l01_set_rf_ch(unsigned char channel);
unsigned char nrf24l01_get_status(void);
unsigned char nrf24l01_get_observe_tx(void);
void nrf24l01_set_rx_addr(unsigned char * address, unsigned int len, unsigned char
rxpipenum);
void nrf24l01_set_tx_addr(unsigned char * address, unsigned int len);
void nrf24l01_set_rx_pw(unsigned char payloadwidth, unsigned char rxpipenum);
unsigned char nrf24l01_get_rx_pw(unsigned char rxpipenum);
unsigned char nrf24l01_get_fifo_status(void);

//auto-ack and pipe-related functions
bool nrf24l01_aa_enabled(unsigned char rxpipenum);
void nrf24l01_aa_enable(unsigned char rxpipenum);
void nrf24l01_aa_disable(unsigned char rxpipenum);
bool nrf24l01_rx_pipe_enabled(unsigned char rxpipenum);
void nrf24l01_rx_pipe_enable(unsigned char rxpipenum);
void nrf24l01_rx_pipe_disable(unsigned char rxpipenum);
unsigned char nrf24l01_get_plos_cnt(void);
void nrf24l01_clear_plos_cnt(void);
void nrf24l01_clear_plos_cnt_param(unsigned char rf_ch);
unsigned char nrf24l01_get_arc_cnt(void);

//utility functions
bool nrf24l01_cd_active(void);
void nrf24l01_clear_flush(void);
unsigned char nrf24l01_get_rx_pipe(void);
unsigned char nrf24l01_get_rx_pipe_from_status(unsigned char status);
void nrf24l01_get_all_registers(unsigned char * data);

//interrupt check/clear functions
bool nrf24l01_irq_pin_active(void);
bool nrf24l01_irq_rx_dr_active(void);
bool nrf24l01_irq_tx_ds_active(void);
bool nrf24l01_irq_max_rt_active(void);
void nrf24l01_irq_clear_all(void);
void nrf24l01_irq_clear_rx_dr(void);
void nrf24l01_irq_clear_tx_ds(void);
void nrf24l01_irq_clear_max_rt(void);

//FIFO_STATUS check functions
bool nrf24l01_fifo_tx_reuse(void);
bool nrf24l01_fifo_tx_full(void);
bool nrf24l01_fifo_tx_empty(void);
bool nrf24l01_fifo_rx_full(void);
bool nrf24l01_fifo_rx_empty(void);

//IO interface-related functions
void nrf24l01_transmit(void);
void nrf24l01_clear_ce(void);
void nrf24l01_set_ce(void);
void nrf24l01_clear_csn(void);
void nrf24l01_set_csn(void);
bool nrf24l01_ce_pin_active(void);
bool nrf24l01_csn_pin_active(void);

//low-level functions for library use only
unsigned char nrf24l01_execute_command(unsigned char instruction, unsigned char * data,
unsigned int len, bool copydata);
void nrf24l01_spi_send_read(unsigned char * data, unsigned int len, bool copydata);
unsigned char spi_send_read_byte(unsigned char byte);

#endif /*NRF24L01_H_*/
```