University of Colorado Boulder

Final Project : ECEN 2440

Authors:
Brandon Lewien
Jean-Christophe Owens
Adam Smrekar
Arash Yousefzadeh

Supervisor:
Alexander Fosdick

December 11th , 2017

MASTER CODE

MSP432P401R

```
/*****************************************************************************
 * @Title: main.c
 *
 * @authors Brandon Lewien, Jean-Christophe Owens, Arash Yousefzadeh, Adam Smrekar
 * @date November 30th, 2017
 * @version - ---
 *
 * Compiled using CCSv7
 *
 * Notes: LCD slows things down tremendously...
 * ***************************************************************************/

#define ADC
#define JOYSTICK
#define UART
//#define LCD
#define TIMER

#include <stdint.h>
#include <stdio.h>
#include "lcdconfig.h"
#include "msp.h"
#include "adc.h"
#include "joystick.h"
#include "uart.h"
#include "timer.h"

#define SCB_SCR_ENABLE_SLEEPONEXIT (0x00000002)
volatile uint8_t value;

void main(void){
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
    SCB->SCR &= ~SCB_SCR_ENABLE_SLEEPONEXIT;
#ifdef LCD
    lcdconfig();                    // Configure LCD
#endif
#ifdef UART
    UART_Configure();               // Configure UART
#endif
#ifdef JOYSTICK
    joystick_configure();           // Configure Joystick P1 stuff
#endif
#ifdef ADC
    ADC_init();                     // Analog to digital configuration
    ADC_addChannel(0,15,0);         // MEM0 x dir
    ADC_addChannel(4,9,0);          // MEM4 y dir
    ADC_EOS(15);                    // Enable EOS
    ADC_start();                    // Enable ADC
#endif
#ifdef TIMER
    timerA0_config();
#endif
    __enable_irq();                 // Enable Interrupts
    while(1){
#ifdef JOYSTICK
```

```
        value = joysticklocation(0,4);
#endif
#ifdef UART
        UART_sent_byte(value);
        //UART_sent_byte(0xA);
#endif
#ifdef LCD
        char text[128];
        sprintf(text,"   Quadrant: %d   ",value);
        getwordsback(text,64,64);
#endif
    }
}

WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer

    configure_GPIO(); //configures pins
    configure_SysTick();

    //configure_ADC();
    /* interrupt configuration */
    //_____________________________________
    // ADC CODE
    char i;
    LCD();
    configure_ADC();                            //configure ADC interrupts
    configure_Clocks();                         //configures clocks for UART
    NVIC_EnableIRQ(PORT5_IRQn);                 //Enables Port5 interrupts
    __enable_interrupt();

    __enable_irq();
    char input[20];
    char input1[20];
    display("Yaxis Module",64,35);
    display(".......",64,83);
    display("Mark,Ryan,JC",64,95);
              //sampling and conversion start
    //Test for screen working

    //configuration for data structure
    //configure_scooter_data(data);

    while(1) {
        distance = get_distance();
        velocity = get_velocity();
        //Convert It Son
        sprintf(distance_string, "Dist: %2.3f \n", distance); //converts distance
        sprintf(velocity_string, "Vel: %2.3f \n", velocity); //converts velocity
        __enable_irq();

        //Send It Son
        display("Escooter Final", 64, 20);
        display(distance_string,64,60);
        display(velocity_string, 64, 70);
        //Displays direction
```

```
        i = diir();
        sprintf(input,"%c",i);
        display(input,64,50);
        ADC14->CTL0 |= ADC14_CTL0_SC;
    }

}
```

```
/*
 * gpio1.h
 *
 *  Created on: Dec 5, 2017
 *      Author: Brandon, Jean-Christophe Owens
 */

#ifndef GPIO1_H_
#define GPIO1_H_

#include <stdint.h>
#include "msp.h"

void gpio_config();
void gpio_LEDconfig();

#endif /* GPIO1_H_ */
```

```
/*****************************************************************************
 * @Title: adc.c
 *
 * @author Brandon Lewien
 * @date November 3rd, 2017
 * @version 1.1--11/23/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions to configure the analog to digital converter
 * (ADC) on the MSP432P401r, sample and convert data to digital values, and
 * calculate the voltage on an analog channel.
 *
 * !Note: A detailed explanation of each function is in the adc header file.
 * Additional credit goes to Samuel Zimmer.
 * _________________________________WARNING_____________________________________
```

```
 * ---------------DO NOT CHANGE ANYTHING IN THIS FILE---------------
 * ----------------------------------------------------------------
 * ****************************************************************/

#include "adc.h"
#include "msp.h"

volatile uint16_t _nadc[32];
volatile uint8_t _eos;                                  // channel to end the
sequence on

void ADC_init(){
    while(REF_A->CTL0 & REF_A_CTL0_GENBUSY);            //If ref generator busy, wait
    REF_A->CTL0  = REF_A_CTL0_VSEL_0 | REF_A_CTL0_ON;   //Enable internal 1.2V ref

    ADC14->CTL0 |= ADC14_CTL0_SHT0_5                    // ADC14 sample-and-hold
time: 96 cycle sample time
                | ADC14_CTL0_SHT1_5                     // ADC14 sample-and-hold
time: 96 cycle sample time
                | ADC14_CTL0_SHP                        // SAMPCON signal is sourced
from the sampling timer
                | ADC14_CTL0_MSC                        // Sample/conversions are
performed automatically as soon as the prior conversion is completed
                | ADC14_CTL0_CONSEQ_3                   // Repeat-sequence-of-
channels
                | ADC14_CTL0_SSEL__HSMCLK               // ADC14 clock source: HSMCLK
                | ADC14_CTL0_ON;                        // Turn on the ADC14 module
    ADC14->CTL1 |= ADC14_CTL1_RES__14BIT;               // 14-bit resolution

    while(!(REF_A->CTL0 & REF_A_CTL0_GENRDY));          //Wait for ref generator to
settle
    ADC14->CTL0 |= ADC14_CTL0_ENC;                      //Enable Conversions

    NVIC_EnableIRQ(ADC14_IRQn);                         //Enable ADC int in NVIC
module
}

void ADC14_IRQHandler(){
    int i;
    for(i=0;i<=_eos;i++) {
        if(ADC14->IFGR0 & (1<<i)) {
            _nadc[i] = ADC14->MEM[i];
        }
    }
}

void ADC_EOS(uint8_t channel){
    ADC14->MCTL[channel]|=ADC14_MCTLN_EOS;
    _eos=channel;
}

void ADC_start(){
    ADC14->CTL0 |= ADC14_CTL0_SC;                        // Start conversions
}
```

```
void ADC_addChannel(uint8_t channel, uint8_t map, uint8_t vref){
    if(channel>31) return;
    if(vref!=0&&vref!=1) return;
    ADC14->CTL0&=~ADC14_CTL0_ENC;                       // Disable conversions
    ADC14->MCTL[channel]|=map|(vref<<8);                // Map MCTL[channel] to a
specific pin and set the vref
    ADC14->IER0|=(1<<channel);                          // Enable the interrupt for
that channel
    ADC14->CTL0 |= ADC14_CTL0_ENC;                      // Enable Conversions
}

uint16_t ADC_getN(uint8_t channel){
    if(channel>31){
        return 0;
    }
    return _nadc[channel];                              // Return the stored
MEM[channel] value
}

double ADC_getMV(uint8_t channel){
    if(channel>31){
        return 0;
    }
    int vref = 1200*((ADC14->MCTL[channel]&ADC14_MCTLN_VRSEL_1)>0)+3300*((ADC14-
>MCTL[channel]&ADC14_MCTLN_VRSEL_1)==0);
    return ((double)ADC_getN(channel))*(((double)vref)/(1<<14));
}
```

```
/*
 * adc.h
 *
 *  Created on: Nov 4, 2017
 *      Author: Brandon
 */

#include "msp.h"
#include <stdint.h>

#ifndef ADC_H_
#define ADC_H_
```

```
/*
 * Function: ADC14_IRQHandler
 * --------------------------
 *   IRQ Handler for when ADC has finished conversion
 */
//ADC14_IRQHandler()

/*
```

```
 * Function: ADC_init
 * ---------------------------
 *   Initializes the ADC and reference generator (1.2v)
 *
 *   returns: void
 */
void        ADC_init();
/*
 * Function: ADC_addChannel
 * ---------------------------
 *   Sets the EOS for continuous sequence mode
 *
 *   channel: the MCTL number 0-32
 *   map: A0-A31 that you want mapped to MEM[channel]
 *   vref: 0=AVCC/AVSS - 1=VREF
 *
 *   returns: void
 */
void        ADC_addChannel(uint8_t channel, uint8_t map, uint8_t vref);
/*
 * Function: ADC_getN
 * ---------------------------
 *   Fetches the NADC value for the specified channel
 *
 *   channel: the MCTL number 0-32
 *
 *   returns: MEM[channel] stored in _nadc
 */
uint16_t    ADC_getN(uint8_t channel);
/*
 * Function: ADC_getMV
 * ---------------------------
 *   Converts the NADC value for the specified channel into millivolts
 *
 *   channel: the MCTL number 0-32
 *
 *   returns: millivolts
 */
double      ADC_getMV(uint8_t channel);
/*
 * Function: ADC_EOS
 * ---------------------------
 *   Sets the EOS for continuous sequence mode
 *
 *   channel: the EOS channel you want to end sampling on
 *
 *   returns: void
 */
void        ADC_EOS(uint8_t channel);
/*
 * Function: ADC_start
 * ---------------------------
 *   Starts ADC sampling and converion. This needs to be called once if
ADC14_CTL0_MSC and ADC14_CTL0_SHP are set
 *
```

```
 *   returns: void
 */
void        ADC_start();

#endif /* ADC_H_ */
```

```
/*****************************************************************************
 * @Title: button1.c
 *
 * @author Brandon Lewien
 * @date December 6th, 2017
 * @version 1.0--12/6/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file handles all button GPIO configurations.
 * Top button is disabled due to conflicts with LCD.
 * PORT3 IRQ Handler is also included in this file.
 * ***************************************************************************/

#include "msp.h"
#include <button1.h>
#include "uart.h"

/*
 * Function: button_config
 * ---------------------------
 *   General button configuration for TI Boosterpack MK II.
 *   Top button (disabled due to LCD) - P5.1
 *   Bottom button - P3.5
 */
void button_config(){
/*
    // Top Button
    P5->SEL0&=~BIT1;
    P5->SEL1&=~BIT1;
    P5->DIR&=~BIT1;
    P5->REN|=BIT1;
    P5->OUT|=BIT1;
    P5->IES|=BIT1;
    P5->IE|=BIT1;
*/
    // Bottom Button
    P3->SEL0&=~BIT5;
    P3->SEL1&=~BIT5;
    P3->DIR&=~BIT5;
    P3->REN|=BIT5;
    P3->OUT|=BIT5;
    P3->IES|=BIT5;
    P3->IE|=BIT5;
    P3->IFG = 0x00;
```

```
}
/*
 * Function: PORT3_IRQHandler
 * ---------------------------
 *   General PORT3 Interrupt Handler. This function
 *   primarily deals with a byte through UART to be sent
 *   via the bus in order to output feedback the slave MSP.
 *
 *   Please note that the bottom button acts funny and reads
 *   Bits 5 and 6 making 0x30 for some odd reason. A workaround
 *   just ignores that BIT6 and does whatever it needs with P3.5.
 */
void PORT3_IRQHandler(){
    if(P3->IFG & BIT5){
        if(P2->OUT & BIT6){
            P2->OUT&=~BIT6;
        }
        else{
            P2->OUT|=BIT6;
        }
        UART_sent_byte(0xB);
        int i = 0;
        for(i=0;i<100;i++);
        P3->IFG &= ~BIT5;
    }
}
```

```
/*
 * button1.h
 *
 *  Created on: Dec 5, 2017
 *      Author: Brandon
 */

#ifndef BUTTON1_H_
#define BUTTON1_H_

void button_config();

#endif /* BUTTON1_H_ */


/*****************************************************************************
 * @Title: joystick.c
 *
 * @author Brandon Lewien
 * @date November 3rd, 2017
 * @version 1.4--11/23/17
 *
 * Compiled using CCSv7
 *
```

```
 * Description:
 * This file contains configuration code for the Joystick on the MKII Boosterpack.
 *
 * ********************************************************************************/

#include <stdlib.h>
#include <stdint.h>
#include "msp.h"
#include "adc.h"
#include "joystick.h"

void joystick_configure(void){
    P4->SEL0 |= BIT4;
    P4->SEL1 |= BIT4;
    P6->SEL0 |= BIT0;
    P6->SEL1 |= BIT0;
}

int8_t joysticklocation(uint8_t channelx, uint8_t channely){
    /*
     * Default position reads for x=8310 and for y=8140
     * Channelx takes an input from main that compares values for the left and right
movement
     * Channely takes an input from main that compares values for the up and down
movement
     * Prioritizing forward movement on top, backwards and 180 turning on bottom.
     * Note that the joystick is not a complete matrix of 2^14x2^14 things (including
the negatives
     * of course). There is a weird cutoff that needed to be made specifically for
Brandon's
     * white LCD Boosterpack MKII.
     *
     * Note!!! : Other Boosterpacks will vary in values! Use Brandon's Boosterpack!!!
MSP
     * does NOT matter!
     */
    int8_t location = 0;
    int16_t xaxis = ADC_getN(channelx);
    int16_t yaxis = ADC_getN(channely);

    if ((xaxis < 13000) && (yaxis >= 15000) && (xaxis > 3000)){
        //This is for full straight up
        location = 4;
    }
    else if ((xaxis < 10000) && (yaxis < 15000) && (xaxis> 6000) && (yaxis > 9000)){
        //This is for half straight up
        location = 3;
    }
    else if((xaxis > 12000) && (yaxis > 13000)){
        //This is for full quadrant 1
        location = 2;
    }
    else if((xaxis > 10000) && (yaxis > 10000) && (xaxis < 16000) && (yaxis <
15000)){
        //This is for half quadrant 1
```

```
        location = 1;
    }
    else if((xaxis < 3000) && (yaxis > 12000)){
        //This is for full quadrant 2
        location = 6;
    }
    else if((xaxis < 8000) && (xaxis > 1000) && (yaxis > 8000) && (yaxis < 15000)){
        //This is for half quadrant 2
        location = 5;
    }
    else if((xaxis < 1000) && (yaxis < 13000) && (yaxis > 3000)){
        //Full left
        location = 7;
    }
    else if((xaxis > 15500) && (yaxis < 13000) && (yaxis > 3000)){
        //Full right
        location = 8;
    }
    else if ((xaxis < 11000) && (yaxis < 3000) && (xaxis > 7000)){
        //This is for full straight down
        location = 9;
    }
    else{
        location = 0;
    }
    return location;
}


/*
 * joystick.h
 *
 *  Created on: Nov 4, 2017
 *      Author: Brandon
 */

#ifndef JOYSTICK_H_
#define JOYSTICK_H_

#include "msp.h"
#include <math.h>
#include <stdlib.h>
#include <stdint.h>

extern volatile uint16_t _nadc[32];
void joystick_configure(void);
int8_t joysticklocation(uint8_t channelx, uint8_t channely);

#endif /* JOYSTICK_H_ */
```

```c
/*****************************************************************************
 * @Title: timer.c
 *
 * @authors Brandon Lewien
 * @date November 23rd, 2017
 * @version 1.2--11/29/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions to configure Pulse-Width Modulation Timers.
 * Two timers are being used with various pin-outs for different PWM
 * configurations. Refer to gpio1.c for specific pin outputs. Additional PWM
 * pins can be set over there.
 * ***************************************************************************/

#include "msp.h"
#include "timer.h"
#include "uart.h"

#define TIMERINTERRUPTS

/*
 * Function: timerA0_config
 * ---------------------------
 *   Timer A0_0 general configuration. This function is
 *   created explicitly for the heartbeat timer.
 */
void timerA0_config(){
    TIMER_A0->R = 0;                            // Reset count
    TIMER_A0->CTL |= TIMER_A_CTL_TASSEL_2 |     // SMCLK
                     TIMER_A_CTL_ID_3 |         // Divider = 8
                     TIMER_A_CTL_IE |           // Timer A0 Interrupt Enable
                     TIMER_A_CTL_MC__UP;        // Up Mode
    TIMER_A0->CCR[0] = 200;                     // Frequency
#ifdef TIMERINTERRUPTS
    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE;     // Enable CCTL interrupts
    NVIC_EnableIRQ(TA0_0_IRQn);
    NVIC_SetPriority(TA0_0_IRQn,2);
#endif
}
/*
 * Function: TA0_0_IRQHandler
 * ---------------------------
 *   Timer A0_0 Interrupt Handler. This function sends a byte
 *   of 0xA every 200ms based off of the configuration above.
 */

void TA0_0_IRQHandler(void){
    if((TIMER_A0->CCTL[0] & TIMER_A_CCTLN_CCIFG) == TIMER_A_CCTLN_CCIFG)
    {
        UART_sent_byte(0xA);
        TIMER_A0->CCTL[0] &= ~(TIMER_A_CCTLN_CCIFG);
    }
```

```
}

/*
 * timer.h
 *
 *  Created on: Nov 3, 2017
 *      Author: Brandon
 */

#ifndef TIMER_H_
#define TIMER_H_

void timerA0_config();

#endif /* TIMER_H_ */
```

```
/*****************************************************************************
 * @Title: uart.c
 *
 * @authors Adam Smrekar, Brandon Lewien
 * @date November 30th, 2017
 * @version - ---
 *
 * Compiled using CCSv7
 *
 * ***************************************************************************/

#include "uart.h"
#include "msp.h"

volatile uint8_t FEEDBACK[100];
volatile uint8_t COUNTER = 0;

#define UCA2

void UART_Configure(){
#ifdef UCA2
    UCA2CTLW0 |= UCSWRST;                   // Put eUSCI in reset
/* Set Pins */
    P3SEL0 |= (BIT2 | BIT3);                // TX & RX Primary mode
    P3SEL1 &= ~(BIT2 | BIT3);

/* Select Frame parameters and clock source */
    UCA2CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK;
    UCA2CTLW0 &= ~EUSCI_A_CTLW0_PEN         // Parity disabled
              &  ~EUSCI_A_CTLW0_MODE0       // Set to uart mode
              &  ~EUSCI_A_CTLW0_MODE1
              &  ~EUSCI_A_CTLW0_MSB         // LSB first
              &  ~EUSCI_A_CTLW0_SEVENBIT    // 8 bit character length
              &  ~EUSCI_A_CTLW0_SPB;        // One stop bit one start bit is default
/* Baud Rate == 115200 */
    UCA2MCTLW = 0xB5A1;
    UCA2BR0 = 0x01;                         // Set Baud Rate
```

```
    UCA2BR1 = 0x00;
    UCA2CTLW0 &= ~UCSWRST;                  // Initialize eUSCI
  //UCA2IE |= EUSCI_A_IE_RXIE;              // Enable interrupt for RX receive
    NVIC_EnableIRQ(EUSCIA2_IRQn);           // Enable IRQ for UART
#endif
}
/* Send a byte of data */
void UART_sent_byte(uint8_t tx_data){
    while(EUSCI_A_IFG_TXIFG & ~UCA2IFG);    // While there is a Transmit flag
    EUSCI_A2->TXBUF = tx_data;              // TX is the data that you want to
transmit
}

/* Send multiple bytes of data */
void UART_sent_n(uint8_t * data, uint32_t length){
    volatile int i = 0;                     // Initialize counter
    for(i; i < length; i++){
        char test = data[i];
        UART_sent_byte(data[i]);            // Loop through data and send
    }
}

/*
 * Function: Bluetooth_config
 * ---------------------------
 *   This function uses the same AT commands that are used to
 *   configure the HC-05 with an Arduino. Since the AT commands are
 *   sent over UART, we took that received data over UART and used
 *   it in this function to configure the bluetooth modules. In
 *   order to send these commands, this function needs to be called
 *   while bluetooth modules are in AT command mode meaning the En
 *   pins are high and there should be a slow consistent blinking LED.
 */
void Bluetooth_config(){
    //UCA2IE |= EUSCI_A_IE_TXIE;              // Enable interrupt for TX send
    UCA2IE |= EUSCI_A_IE_RXIE;              // Enable interrupt for RX receive
    UART_sent_n("AT",2);
    UART_sent_n("AT+UART=115200",15);
    UART_sent_n("AT+ROLE=1",10);
    UART_sent_n("AT+CMODE=0",11);
    UART_sent_n("AT+BIND=14:3:60d17",19);
}

void EUSCIA2_IRQHandler(void){
    if (EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG){
        FEEDBACK[COUNTER] = EUSCI_A2->RXBUF;
        COUNTER++;
    }
}
```

```c
/*
 * uart.h
 *
 *  Created on: Nov 25, 2017
 *      Author: Adam
 */

#include "msp.h"

#ifndef UART_H_
#define UART_H_

void UART_Configure();                              /* UART Configuration */
void UART_sent_n(uint8_t * data, uint32_t length); /* Send multiple bytes of data */
void UART_sent_byte(uint8_t data);                  /* Send a byte of data */
void Bluetooth_config();

#endif /* UART_H_ */
```

SLAVE CODE

MSP432P401R

```
/*******************************************************************************
 * @Title: main_s.c
 *
 * @authors Brandon Lewien, Jean-Christophe Owens, Arash Yousefzadeh, Adam Smrekar
 * @date November 30th, 2017
 * @version - ---
 *
 * Compiled using CCSv7
 *
 * Please read within timer.c the pwm function for more details about the errors
 * that occurred with changing of speeds with the speed controller.
 *
 * WARNING! DO NOT EDIT AND RECOMPILE CODE TO MSPS!
 * WARNING! DO NOT EDIT AND RECOMPILE CODE TO MSPS!
 * ******************************************************************************/
#define TIMER
#define UART
#define GPIO
#define PWM

#include "msp.h"
#include "gpio1.h"
#include "timer.h"
#include "uart.h"
#include "bluetooth.h"
#include "buffer.h"
//#include "IRBeamSv.h"

volatile uint16_t value;

void main(void){
      WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
#ifdef UART
      UART_Configure();               // Configure UART
    UART_send_n("ElectroKart",11);
    UART_send_byte(0xD);
    UART_send_byte(0xA);
    UART_send_n("Counting values under 500 = Weak Signal",39);
    UART_send_byte(0xD);
    UART_send_byte(0xA);
    UART_send_n("Counting values under 200 = Partial Lag",39);
    UART_send_byte(0xD);
    UART_send_byte(0xA);
    UART_send_n("Counting values under 100 = Good Signal",39);
    UART_send_byte(0xD);
    UART_send_byte(0xA);
    UART_send_n("Wear a Helmet! Don't drink and drive!",39);
    UART_send_byte(0xD);
    UART_send_byte(0xA);
                                    // Hit reset to see these messages on the Slave
MSP^^
#endif
#ifdef TIMER
    timerA0_config();               // Configure First Timer for PWM/Motor
    timerA1_config();               // Configure Second Timer for PWM/Motor
```

```
#endif
#ifdef GPIO
    gpio_pwmconfig();               // Configure GPIO for PWM
    gpio_config();                  // GPIO Interrupt Enablers
    gpio_LEDout();
#endif
      __enable_irq();                 // Enable Interrupts
      while(1){
          value = retriever();        // Store filtered value from receiver via
bluetooth
        status();                   // Implements Bluetooth status feedback
        joystickspin();             // Implement Motor status feedback
#ifdef PWM
        pwm(value);                 // Use value from filtered function to generate
proper PWM wave
#endif
      }
}
```

```
/*******************************************************************************
 * @Title: bluetooth.c
 *
 * @authors Brandon Lewien, Adam Smrekar
 * @date December 5th, 2017
 * @version 1.0--12/5/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions for all bluetooth related code and user feedback.
 * Main issues solved with this code include:
 *  - Filter of signal sorting
 *  - Heartbeat Timer for signal feedback
 *  - Filtered value returner
 *
 * Lines UART_send_byte(0xD) and 0xA send a newline without tab.
 * ******************************************************************************/

#include "msp.h"
#include "uart.h"
#include "bluetooth.h"
#include <stdio.h>

volatile uint16_t LASTVALUE = 0xA;
volatile uint16_t COUNTER = 0;
volatile uint16_t HOLDER = 0x00;
HeartBeat HB;

/*
 * Function: retriever
 * ---------------------------
 *   Bluetooth value sorter and code feedback for bluetooth state.
```

```
 *   This function implements a heartbeat timer where the MSP & Bluetooth Master
 *   Device sends a consistent 0xA every 200ms.
 *
 *   This means there should be about 4 0xA's being sent before counter == 1000.
 *   The filtered value being returned will only return a value between 0-9, where
 *   every other value is considered as garbage.
 */
/*   It is also noted that this is improper technique. We are pulling the RXBUF
without an interrupt.
 *   A proper technique is to use the EUSCI_A2 IRQ Function and when a flag is set
then clear the flag.
 *   The reason why this technique wasn't implemented is because a bug kept getting
thrown before the problem
 *   can be solved. However, in the usage that we are doing the values being
retrieved aren't affected
 *   where it will will break the code. The only difference is that INITIAL will be
pulling as fast as the
 *   main while(1) loop, not whenever it is necessary.
 *   With additional testing, an interrupt is not needed. Consult Brandon for more
information if neccessary.
 */
uint16_t retriever(void){
    if(COUNTER<1000){
        HB.INITIAL = EUSCI_A2->RXBUF;                           // Grabs value from A2
Receive Buffer
        if(HB.INITIAL < 10  && HB.HEARTBEATFLAG){               // Filter Unnecessary
Signals when Flag == 1
            HB.FILTERED_VALUE = HB.INITIAL;                     // Store initial value
if conditions are met
        }
        else if(HB.INITIAL == 0xA && LASTVALUE != HB.INITIAL){ // If signal ==
heartbeat and last value received
                                                                // is not equal to the
current value

            HB.HEARTBEATFLAG = 1;                               // If conditions are
met set flag == 1 to enable
                                                                // value change
            COUNTER = 0;                                        // Reset counter
        }
        LASTVALUE = HB.INITIAL;                                 // Update current
value equal to last value
        COUNTER++;                                              // Increase timer, can
be reset if received 0xA above
    }
    else{
        HB.HEARTBEATFLAG = 0;                                   // If timer exceeds
1000 set flag to 0 so no
                                                                // information is
pulled from the receive buffer
                                                                // and sent as a
return
        HB.FILTERED_VALUE = 0;                                  // Always sent a
filtered value of 0 meaning dead
```

```
                                                                    // signal is being
sent
        COUNTER = 0;                                                // For reconnection
purposes
    }
    return HB.FILTERED_VALUE;
}
/*
 * Function: status
 * ----------------------------
 *   Function status takes information from the retriever and uses it for
 *   user feedback. This is useful for clear Bluetooth Connectivity indication
 *   where green is a connection while red is a broken connection. UART transmission
 *   is also supported with the master Boosterpack botton button press.
 *
 *   Counter is pretty inaccurate. It only samples the current counting time. The
user can
 *   interpret numbers before 500 are good. 200 is partial lag.
 */
void status(void){
    if(HB.HEARTBEATFLAG == 1){
        P2->OUT = BIT1;
    }
    else if (HB.HEARTBEATFLAG == 0){
        P2->OUT = BIT0;
    }
    if(HB.INITIAL == 0xB){
        if(HB.HEARTBEATFLAG == 1){
            uint8_t text[128];
            sprintf(text,"Counter: %i - ",COUNTER);
            UART_send_n(text,14);
            UART_send_n("BT Status: OK!",15);
            UART_send_byte(0xD);
            UART_send_byte(0xA);
        }
    }
    else if (HB.HEARTBEATFLAG == 0){
        UART_send_n("ERROR! BT OFF!",15);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
    }
}
/*
 * Function: joystickspin
 * ----------------------------
 *   This function allows for feedback to be sent based off of the joystick
 *   location being received
 */
void joystickspin(void){
    if(HB.INITIAL == 4 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Forward",17);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 4;
    }
```

```
    else if(HB.INITIAL == 3 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Partial Forward",25);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 3;
    }
    else if(HB.INITIAL == 2 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = First Quad",20);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 2;
    }
    else if(HB.INITIAL == 1 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Partial First Quad",28);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 1;
    }
    else if(HB.INITIAL == 0 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Off",13);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 0;
    }
    else if(HB.INITIAL == 5 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Partial Second Quad",29);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 5;
    }
    else if(HB.INITIAL == 6 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Second Quad",21);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 6;
    }
    else if(HB.INITIAL == 7 && HOLDER != HB.INITIAL){
        UART_send_n("Motor 1 = Max (Left)",21);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 7;
    }
    else if(HB.INITIAL == 8 && HOLDER != HB.INITIAL){
        UART_send_n("Motor 2 = Max (Right)",22);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 8;
    }
    else if(HB.INITIAL == 9 && HOLDER != HB.INITIAL){
        UART_send_n("Motors = Backwards",19);
        UART_send_byte(0xD);
        UART_send_byte(0xA);
        HOLDER = 9;
    }
}
```

```
/*
 * Function: Bluetooth_config
 * --------------------------
 *   This function uses the same AT commands that are used to
 *   configure the HC-05 with an Arduino. Since the AT commands are
 *   sent over UART, we took that received data over UART and used
 *   it in this function to configure the bluetooth modules. In
 *   order to send these commands, this function needs to be called
 *   while bluetooth modules are in AT command mode meaning the En
 *   pins are high and there should be a slow consistent blinking LED.
 */
void Bluetooth_config(void){
    //UCA2IE |= EUSCI_A_IE_TXIE;             // Enable interrupt for TX send
    UCA2IE |= EUSCI_A_IE_RXIE;             // Enable interrupt for RX receive
    UART_send_n("AT",2);
    UART_send_n("AT+UART=115200",15);
    UART_send_n("AT+ROLE=1",10);
    UART_send_n("AT+CMODE=0",11);
    UART_send_n("AT+BIND=14:3:60d17",19);
}


/*
 * bluetooth.h
 *
 *  Created on: Dec 5, 2017
 *      Author: Brandon
 */

#ifndef BLUETOOTH_H_
#define BLUETOOTH_H_

typedef struct{
    volatile uint16_t HEARTBEATFLAG;
    volatile uint16_t FILTERED_VALUE;
    volatile uint16_t INITIAL;
} HeartBeat;

uint16_t retriever(void);
void status();
void joystickspin(void);
void Bluetooth_config(void);

#endif /* BLUETOOTH_H_ */
```

```c
/*****************************************************************************
 * @Title: buffer.c
 *
 * @author Brandon Lewien, Jean-Christophe Owens, Adam Smrekar
 * @date October 24, 2017
 * @version 1.0
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions to configure the UART peripheral on the MSP432,
 * and send data over UART to a serial terminal.
 *
 * Additional credit goes to Samuel Zimmer.
 * ***************************************************************************/
#include "msp.h"
#include "buffer.h"
#include <string.h>
#include <stdlib.h>

buffer * tx;
buffer * rx;
uint8_t ready;

void buffer_configure(void){
    tx = (buffer *)buffer_initialize(256);
    rx = (buffer *)buffer_initialize(512);
}
/*
 * Function: uart_send
 * ---------------------------
 *   Sends one byte to the UART TX buffer to be written when ready
 *
 *   byte: byte to be written
 *
 *   returns: void
 */
void uart_send(uint8_t byte) {
    buffer_add(tx,byte);
}
/*
 * Function: uart_print
 * ---------------------------
 *   Prints a string of bytes to the UART by adding each to the TX
 *   buffer and sending them via the tx ready interrupt
 *
 *   bytes: byte array to be written
 *
 *   returns: void
 */
void uart_print(uint8_t * bytes) {
    uint32_t i;
    int size = strlen(bytes);
```

```
    for(i=0;i<size;i++) {
        uart_send(bytes[i]);
    }
    buffer_add(tx,13);
    EUSCI_A2->IE |= EUSCI_A_IE_TXIE;
}
/*
 * Function: EUSCIA0_IRQHandler
 * ----------------------------
 *   Handles the interrupts for the EUSCIA0 peripheral
 *
 */
void EUSCIA2_IRQHandler() {
    if(EUSCI_A2->IFG & EUSCI_A_IFG_TXIFG) {
        if(tx->num_items>0) {
            EUSCI_A2->TXBUF=buffer_get(tx);
        }
        else {
            EUSCI_A2->IE &=~EUSCI_A_IE_TXIE;
        }
    }
    if(EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG) {
        uint8_t _r = EUSCI_A2->RXBUF;
        if(_r==13||_r==10||buffer_isfull(rx)) ready=1;
        buffer_add(rx,_r);
    }
}
/*
 * Function: buffer_initialize
 * ----------------------------
 *   Initializes a circular buffer of size len
 *
 *   len: length of the circular buffer
 *
 *   returns: pointer to initialized buffer
 */
buffer * buffer_initialize(uint32_t len) {
    if(len<=0) return NULL;
    buffer * buf;
    buf=(buffer *)malloc(sizeof(buffer));
    if(buf==NULL) {
        return NULL;
    }
    buf->buffer = (uint8_t *)malloc(len*sizeof(uint8_t));
    if(buf->buffer==NULL) {
        return NULL;
    }
    buf->head=0;
    buf->tail=1;
    buf->capacity=len;
    buf->num_items=0;
    buffer_clear(buf);
    return buf;
}
```

```c
/*
 * Function: buffer_clear
 * ----------------------------
 *   Clears a circular buffer by setting the data to all zeros
 *
 *   buf: pointer to the buffer
 *
 *   returns: success or failure
 */
int8_t buffer_clear(buffer * buf) {
    if(!buf) return -1;
    uint32_t i;
    for(i=0;i<buf->capacity;i++)
        buf->buffer[i]=0;
    buf->num_items=0;
    return 0;
}
/*
 * Function: buffer_delete
 * ----------------------------
 *   Deletes the buffer from memory
 *
 *   buf: buffer to be deleted
 *
 *   returns: success or failure
 */
int8_t buffer_delete(buffer * buf) {
    if(!buf) return -1;
    free(buf);
    return 0;
}
/*
 * Function: buffer_isfull
 * ----------------------------
 *   Returns if the circular buffer is full or not
 *
 *   buf: pointer to the buffer
 *
 *   returns: -1 if null pointer, 0 if not full, 1 if full
 */
int8_t buffer_isfull(buffer * buf) {
    if(!buf) return -1;
    return(buf->num_items==buf->capacity);
}
/*
 * Function: buffer_isempty
 * ----------------------------
 *   Checks if the circular buffer is empty
 *
 *   buf: buffer to check
 *
 *   returns: -1 if null pointer, 0 is not empty, 1 if empty
 */
int8_t buffer_isempty(buffer * buf) {
```

```
    if(!buf) return -1;
    return (buf->head==buf->tail);
}
/*
 * Function: buffer_length
 * ---------------------------
 *   Returns the legnth of the buffer (how many items)
 *
 *   buf: pointer to buffer
 *
 *   returns: length of buffer
 */
int8_t buffer_length(buffer *buf) {
    if(!buf) return -1;
    return (buf->num_items);
}

/*
 * Function: buffer_add
 * ---------------------------
 *   Adds an item to the end of the buffer
 *
 *   buf: pointer to the buffer
 *   item: item to add to the buffer
 *
 *   returns: success or failure
 */
int8_t buffer_add(buffer * buf, uint8_t item) {
    if(!buf) return -1;

    if(buffer_isfull(buf)) {
        buf->tail=inc(buf->tail,buf->capacity);
        buf->head=inc(buf->head,buf->capacity);
        buf->buffer[buf->head]=item;
        return 1;
    }
    buf->head=inc(buf->head,buf->capacity);
    buf->buffer[buf->head]=item;
    buf->num_items++;
    if(buf->num_items>buf->capacity) buf->num_items=buf->capacity;
    return 0;
}
/*
 * Function: buffer_get
 * ---------------------------
 *   Retrieves the first item in the FIFO circular buffer
 *
 *   buf: pointer to the buffer
 *
 *   returns: item add the tail of the buffer
 */
uint8_t buffer_get(buffer * buf) {
    if(!buf) return NULL;
    uint8_t item;
    if(buf->num_items) {
```

```
        item = buf->buffer[buf->tail];
        //buf->buffer[buf->tail]=0;
        buf->tail=inc(buf->tail,buf->capacity);
        if(buf->num_items>0) buf->num_items--;
    }
    return item;
}
uint32_t inc(uint32_t var, uint32_t cap) {
    if(var==cap-1)
        return 0;
    return var+1;
}
```

```
/*
 * buffer.h
 *
 *  Created on: Dec 5, 2017
 *      Author: Brandon
 */

#ifndef BUFFER_H_
#define BUFFER_H_

typedef struct {
    volatile uint32_t num_items;
    volatile uint32_t capacity;
    volatile uint32_t head;
    volatile uint32_t tail;
    uint8_t * buffer;
} buffer;

void buffer_configure(void);
void uart_send(uint8_t byte);
void uart_print(uint8_t * bytes);
buffer * buffer_initialize(uint32_t len);
int8_t buffer_clear(buffer * buf);
int8_t buffer_delete(buffer * buf);
int8_t buffer_isfull(buffer * buf);
int8_t buffer_isempty(buffer * buf);
int8_t buffer_length(buffer *buf);
int8_t buffer_add(buffer * buf, uint8_t item);
uint8_t buffer_get(buffer * buf);
uint32_t inc(uint32_t var, uint32_t cap);

#endif /* BUFFER_H_ */
```

```c
/*****************************************************************************
 * @Title: gpio1.c
 *
 * @author Brandon Lewien
 * @date November 23rd, 2017
 * @version 1.1--11/23/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions to configure different GPIOs and port interrupt
 * handlers.
 *
 * Other Notes:
 * The reason why this file is called gpio1.c instead of gpio.c is because
 * I didn't want to have a conflict with the pre-made gpio class file within
 * the driverlib folder for the LCD. Since LCD is not used for the slave,
 * I left the porting the same.
 *
 * ***************************************************************************/

#include <gpio1.h>
#include <stdlib.h>
#include "timer.h"

/*
 * Function: gpio_config
 * ---------------------------
 *   General NVIC_EnableIRQ Handler Enables
 */
void gpio_config() {
    NVIC_EnableIRQ(PORT1_IRQn);
    //NVIC_EnableIRQ(PORT2_IRQn);
    //NVIC_EnableIRQ(PORT3_IRQn);
    //NVIC_EnableIRQ(PORT4_IRQn);
    //NVIC_EnableIRQ(PORT5_IRQn);
}
/*
 * Function: gpio_pwmconfig
 * ---------------------------
     * P2.4 is the configuration for PWM A0.1
     * P2.5 is the configuration for PWM A0.2
     * Refer to the MSP432 Overview page 8 for further pin-outs
     * Remember to uncomment needed port interrupts from gpio_config
 */
void gpio_pwmconfig(){
    P1->DIR |= BIT6;
    P1->SEL1 &= ~ BIT6;
    P1->SEL0 |= BIT6;
    P1->OUT &= ~BIT6;

    P1->DIR &= ~BIT6;

    P1->DIR |= BIT7;
    P1->SEL1 &= ~ BIT7;
```

```
    P1->SEL0 |= BIT7;
    P1->OUT &= ~BIT7;

    P2->DIR |= BIT4;
    P2->SEL1 &= ~ BIT4;
    P2->SEL0 |= BIT4;

    P2->DIR |= BIT5;
    P2->SEL1 &= ~ BIT5;
    P2->SEL0 |= BIT5;

    P7->DIR |= BIT7;
    P7->SEL1 &= ~ BIT7;
    P7->SEL0 |= BIT7;
}
/*
 * Function: gpio_LEDconfig
 * ----------------------------
     * General LED Bitmapping
 */
void gpio_LEDconfig() {
    P2->SEL0&=~BIT6&~BIT4;
    P2->SEL1&=~BIT6&~BIT4;
    P2->DIR|=BIT6|BIT4;
    P2->OUT&=~BIT6&~BIT4;

    P5->SEL0&=~BIT6;
    P5->SEL1&=~BIT6;
    P5->DIR|=BIT6;
    P5->OUT&=~BIT6;
}
/*
 * Function: gpio_LEDout
 * ----------------------------
     * Different inputs for specific color combinations.
     * This function specifically turns specific lights on
     * or off on the MSP (Not the Boosterpack)
     * Bits 0, 1, 2.
 */
void gpio_LEDout(){
    P2->SEL0 &= ~BIT0 & ~BIT1 & ~BIT2;
    P2->SEL1 &= ~BIT0 & ~BIT1 & ~BIT2;
    P2->DIR |= BIT0 | BIT1 | BIT2;
    P2->OUT |= BIT0 | BIT1 | BIT2;                  //Test All LEDS (White)
}
/*
 * Function: PORT1_IRQHandler
 * ----------------------------
     * Clear Interrupt Flags with bitshifting.
 */
void PORT1_IRQHandler(){
    int i;
    for(i=0;i<8;i++)                                //if ANY port is called
    {
        if(P1->IFG&(1<<i)) {
```

```
            P1->IFG&=~(1<<i);                       //clear interrupt flag
        }
    }
}
```

```
/*
 * gpio1.h
 *
 *  Created on: Nov 20, 2017
 *      Author: Brandon
 */

#ifndef GPIO1_H_
#define GPIO1_H_

#include <stdint.h>
#include "msp.h"

void gpio_config();
void gpio_pwmconfig();
void gpio_LEDconfig();
void gpio_LEDout();

#endif /* GPIO1_H_ */
```

```
/*
 * IRBeamsSv.c
 *
 *
 *  Created on: Dec 8, 2017
 *      Author: Jean-Christophe, Adam Smrekar
 *
 *
 * Description: Contains functions that configures pins on the MSP to
 * inputs or outputs. Also contains interrupt handlers for these pins,
 * such as the one used for our IR beam break. Finally this contains
 * functions that can calculate distance and velocity from an interrupt
 * count and SysTick times. SysTick times are converted to seconds via
 * a separate function.
 *
 */

#include "msp.h"
#include "IRBeamSv.h"
#include "uart.h"
#include "timer.h"
```

```c
#include <math.h>
#include <stdlib.h>
#include <stdio.h>


//extern scooter_t * data; //externally declare the data struct

/* GPIO global variables */
volatile uint16_t ir_count = 0;
volatile uint16_t checkSumFlag2 = 0;
volatile uint16_t checkSumFlag3 = 0;

volatile uint16_t time1 = 0;
volatile uint16_t time2 = 0;
volatile uint16_t i = 0;
uint8_t * msg;

/* ***********************
 * configure P2
 * Wheel A *
 * ***********************/
void configure_WheelA() {
    /* Configuring P2 Inputs */
    P2->SEL0 = 0x00; //Sets port to general IO Mode
    P2->SEL1 = 0x00; //Sets port to general IO Mode
    P2->DIR = 0x00; //Sets direction to Input
    P2->REN = BIT1 | BIT4 | BIT2; //Ensures pull up resistors for pin 1, 4, and 2
    P2->OUT = BIT1 | BIT4 | BIT2; //Enables pull up for pin 1, 4, and 2
    P2->IFG = 0x00; //Clears port pin interrupt flag
    P2->IES = BIT1 | BIT4 | BIT5| ~BIT2; //Sets port pin edge trimmer to Hi-Lo
transition
    P2->IE = BIT1 | BIT4 | BIT5| BIT2; //Enables Interrupts

    /* configures red LED */
    P2->DIR |= BIT0; //Sets P2.0 to output

    /*P1->SEL0 = 0x00;
    P1->SEL1 = 0x00;
    P1->DIR |= BIT0;*/

}

/*******************************
 * CONFIGS
 * ***************************/
void configure_WheelB() {
    /* Configuring P Inputs */
    P2->SEL0 = 0x00; //Sets port to general IO Mode
    P2->SEL1 = 0x00; //Sets port to general IO Mode
    P2->DIR = 0x00; //Sets direction to Input
    P2->REN = BIT1 | BIT4 | BIT2; //Ensures pull up resistors for pin 1, 4, and 2
    P2->OUT = BIT1 | BIT4 | BIT2; //Enables pull up for pin 1, 4, and 2
    P2->IFG = 0x00; //Clears port pin interrupt flag
    P2->IES = BIT1 | BIT4 | BIT5| ~BIT2; //Sets port pin edge trimmer to Hi-Lo
transition
```

```
    P2->IE = BIT1 | BIT4 | BIT5| BIT2; //Enables Interrupts

    /* configures red LED */
    P2->DIR |= BIT0; //Sets P2.0 to output
}
/*TIMER CONFIG*/

void timer_a2_config(){
    TIMER_A2->R = 0;                                        // Reset count
    TIMER_A2->CTL = TIMER_A_CTL_TASSEL_2 | TIMER_A_CTL_ID_3 | TIMER_A_CTL_IE |
TIMER_A_CTL_MC__UP; // SMCLK, Clock Divider /8, TimerA interrupt enable, Up mode
    TIMER_A2->CCR[0] = 3750;                                // Value to count to
    TIMER_A2->CCTL[0] = TIMER_A_CCTLN_CCIE;                 // TACCR0 interrupt
enabled
    TIMER_A2->EX0 |= TIMER_A_EX0_TAIDEX_7;                  // divide by another
8
}

void timer_a3_config(){
    TIMER_A3->R = 0;                                        // Reset count
    TIMER_A3->CTL = TIMER_A_CTL_TASSEL_2 | TIMER_A_CTL_ID_3 | TIMER_A_CTL_IE |
TIMER_A_CTL_MC__UP; // SMCLK, Clock Divider /8, TimerA interrupt enable, Up mode
    TIMER_A3->CCR[0] = 3750;                                // Value to count to
    TIMER_A3->CCTL[0] = TIMER_A_CCTLN_CCIE;                 // TACCR0 interrupt
enabled
    TIMER_A3->EX0 |= TIMER_A_EX0_TAIDEX_7;                  // divide by another
8
}


/* *************************
 *  P2 IRQ Handler
 *  *************************/
// Change to Port 5.1, 5.2 maybe
void PORT2_IRQHandler() {
    if (P2->IFG & (BIT2 | BIT4)) { //IR beam
        //time1 = TIMER_A2->CCR[0];
        //time2 = TIMER_A3->CCR[0];
        P2IFG &= ~BIT2 | ~BIT4; //reset interupt flag
        //ir_count += 1; //increment the number of interrupts
    }
}

void TA2_N_IRQHandler(){
    if(TIMER_A2->CCTL[0] & TIMER_A_CCTLN_CCIFG){ // Check Timer A0 Interrupt Flags
        TIMER_A0->CCTL[0] &= ~(TIMER_A_CCTLN_CCIFG); // Clear flag
        checkSumFlag2 = 4;
        time1++;
    }
}
void TA3_N_IRQHandler(){
    if(TIMER_A3->CCTL[0] & TIMER_A_CCTLN_CCIFG){ // Check Timer A0 Interrupt Flags
        TIMER_A0->CCTL[0] &= ~(TIMER_A_CCTLN_CCIFG); // Clear flag
        checkSumFlag3 = 4;
        time2++;
```

```
    }
}
/******************/

void checkSum(uint16_t valueReceived){
    if(valueReceived && checkSumFlag2 && checkSumFlag3){
        while(valueReceived == 4){
            if (i < 1000){ // checks for ~ms
                i++;
            }
            else if (i == 1000){
                if ((time1 - time2) <= 1000 && (time1 - time2) >= -1000) {         //
Will change the values of 1000 later
                    msg = "Wheels are in sync";
                    UART_send_n(msg,sizeof(msg));   // Success
                }
                else{
                    msg = "Error: Wheels are not in sync";    // Error
                    UART_send_n(msg,sizeof(msg));
                }
                i = 0;      // reset i counter to 0
            }
        }
    }
}
```

```
#ifndef GPIO_H_
#define GPIO_H_

/* Macros to convert systick to seconds */
#define tick2sec (0.000000333333) //this is how many sec 1 tick equals

/* Macros for distance calculation */
#define diameter (31) //diameter of wheel in meters
#define pi (3.14159265359) //define pi
#define circumference (diameter*pi) //calculates circumference
#define arcLength (circumference/14) //find the meaning of life



void configure_WheelA();
void configure_WheelB();
void PORT2_IRQHandler();

//float systick_to_secs(uint32_t systick);
void timer_a2_config();
void timer_a3_config();
```

```
void checkSum(uint16_t valueReceived);

#endif /* GPIO_H_ */
```

```
/*******************************************************************************
 * @Title: timer.c
 *
 * @authors Brandon Lewien
 * @date November 23rd, 2017
 * @version 1.3--12/5/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains functions to configure Pulse-Width Modulation Timers.
 * Two timers are being used with various pin-outs for different PWM
 * configurations. Refer to gpio1.c for specific pin outputs. Additional PWM
 * pins can be set over there.
 * ******************************************************************************/

#include <gpio1.h>
#include "msp.h"
#include "timer.h"

//#define TIMERINTERRUPTS

/*
 * Function: timerA0_config
 * ---------------------------
 *   First Speed Control PWM Configuration
 *
 *   !Note: TIMER_A0->CCTL[x] (4 PWM waves can be set) MUST be set in order for PWM
to work.
 */
void timerA0_config(){
    TIMER_A0->R = 0;                            // Reset count
    TIMER_A0->CTL = TIMER_A_CTL_TASSEL_2 |      // SMCLK
                    TIMER_A_CTL_ID_3 |          // Divider = 8
                  //TIMER_A_CTL_IE |            // Timer A0 Interrupt Enable
                    TIMER_A_CTL_MC__UP|         // Up Mode
                    TIMER_A_CTL_IFG;            // Timer A0 Interrupt Flag
    TIMER_A0->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7;// Reset/Set PWM for A0.x where
0<=x<=6
    TIMER_A0->CCR[0] = 200;                     // Frequency
#ifdef TIMERINTERRUPTS
    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE;  // Enable CCTL interrupts
    NVIC_EnableIRQ(TA0_0_IRQn);
#endif
}
/*
```

```
 * Function: timerA1_config
 * ---------------------------
 *   Second Speed Control PWM Configuration
 *
 *   Note that CCTL has 4 Capture/Compare Modes.
 *   We are using Compare Mode for PWM. Change and/or
 *   configure TIMER_A1->CCTL[x] accordingly.
 */
void timerA1_config(){
    TIMER_A1->R = 0;                            // Reset count
    TIMER_A1->CTL = TIMER_A_CTL_TASSEL_2 |      // SMCLK
                    TIMER_A_CTL_ID_3 |          // Divider = 8
                  //TIMER_A_CTL_IE |            // Timer A0 Interrupt Enable
                    TIMER_A_CTL_MC__UP|         // Up Mode
                    TIMER_A_CTL_IFG;            // Timer A0 Interrupt Flag
    TIMER_A1->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;// Reset/Set
    TIMER_A1->CCR[0] = 200;                     // Frequency
}
/*
 * Function: pwm
 * ---------------------------
 *   Main Wave Generator. TIMER_A0->CCR[x] allows for different
 *   duty cycles to be generated that varies the output of the
 *   speed controller.
 *   TIMER_A0->CCR[1] uses P2.4, while CCR[2] uses P2.5. This information
 *   is found in the gpio1.c file.
 *
 *   Using enumerations for the duty cycle readings. It makes the code cleaner.
 *   The enumeration function is found in timer header file.
 *
 *   The speed controller needs at least a 1,000 Hz Frequency in order to be read.
 *   2,000 Hz is the max.
 *
 *   Specific else-if functions are commented out because the functionality of the
 *   speed controller cannot vary the PWM signals being sent. a 95% percent duty
cycle
 *   at 1,000Hz-2,000Hz allows the initial current to flow to allow the motors to
kick.
 *   When going down to a 5% duty cycle the motors will stay the same speed.
 *   If starting from a 5% duty cycle the motors only have enough power to allow them
 *   to click and move every so slowly. A literal 'kick' allows the motors to have
 *   enough power to go in the right speed (but maxing out with no speed varying).
 *
 *   There are several explanations for this. Error with the calibration for the
speed controller
 *   might play a huge factor. With tests, scaling the project down allowed the speed
controller
 *   to vary a 6V 1.5A motor to adjust speeds accordingly. It is also known that a
change of
 *   frequency from 2k to 1k allowed the motors to spin in opposite directions (speed
 *   controller swapped polarities like a relay). However, adding current to have an
average of
 *   20A/12V allows the speed controller to only have one output. It is noted that
 *   the specific speed controllers *can* tolerate the voltage and is rated for 12V
input.
```

```
 *   The current on the other hand might be the factor. There are other tests that
were conducted
 *   however, consult Brandon for more details.
 *
 *   NOTE! p7.7 is TA1.1. This will be used for the RIGHT Motor
 *         p2.5 is TA0.2. This will be used for the LEFT Motor
 *
 */
void pwm(uint8_t inputvalue){
//FORWARD
    if(inputvalue == 4){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A1->CCR[0] = freqmax;
        TIMER_A0->CCR[0] = freqmax;
        TIMER_A0->CCR[2] = max;
        TIMER_A1->CCR[1] = max;
    }
/*  Refer above for detailed explanation of why these are commented out.
 * ------------------------------------------------------------------------
 *
    else if (inputvalue == 3){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = twentyfive;
        TIMER_A1->CCR[1] = twentyfive;
    }

//RIGHT
    else if (inputvalue == 2){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = max;
        TIMER_A1->CCR[1] = twentyfive;
    }
    else if (inputvalue == 1){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = fifty;
        TIMER_A1->CCR[1] = twentyfive;
    }
    else if (inputvalue == 8){
        P1->OUT |= BIT7;
        TIMER_A0->CCR[2] = twentyfive;
        TIMER_A1->CCR[1] = twentyfive;
    }
//LEFT
    else if (inputvalue == 5){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = twentyfive;
        TIMER_A1->CCR[1] = fifty;
    }    */
    else if (inputvalue == 6){
        P1->OUT &= ~BIT7;
```

```
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = zero;
        TIMER_A1->CCR[1] = max;
    }
    else if (inputvalue == 2){
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = max;
        TIMER_A1->CCR[1] = zero;
    }
    /*
    else if (inputvalue == 7){
        P1->OUT |= BIT6;
        TIMER_A0->CCR[2] = twentyfive;
        TIMER_A1->CCR[1] = twentyfive;
    }
*/
    else if (inputvalue == 9){
        P1->OUT |= BIT6;
        P1->OUT |= BIT7;
        TIMER_A0->CCR[0] = 50;
        TIMER_A1->CCR[0] = 10;
        TIMER_A0->CCR[2] = 45;
        TIMER_A1->CCR[1] = 45;
    }
//NOTHING
    else{
        P1->OUT &= ~BIT7;
        P1->OUT &= ~BIT6;
        TIMER_A0->CCR[2] = zero;
        TIMER_A1->CCR[1] = zero;
    }
}
```

```
/*
 * timer.h
 *
 *  Created on: Nov 3, 2017
 *      Author: Brandon
 */

#ifndef TIMER_H_
#define TIMER_H_


enum dutycycle{
    zero = 0,
    twentyfive = 50,
    fifty = 100,
    max = 190,
```

```
    freqmax = 200
};

void timerA0_config();
void timerA1_config();
void pwm(uint8_t inputvalue);
uint16_t retriever(void);

#endif /* TIMER_H_ */
```

```
/*****************************************************************************
 * @Title: uart.c
 *
 * @authors Adam Smrekar
 * @date November 30th, 2017
 * @version 2.0--12/7/17BL
 *
 * Compiled using CCSv7
 *
 * Description:
 * This file contains all configurations for UART Enhanced Universal Serial
 * Communication Interfaces. Specifically, UCA0 and UCA2 are used, 0 for
communication
 * through wire and 2 for communication through bluetooth P3.2 and P3.3.
 *
 * Uncomment UCA0 when necessary.
 * ***************************************************************************/

#include "uart.h"
#include "msp.h"

//#define UCA0
#define UCA2

void UART_Configure(){
#ifdef UCA0
    UCA0CTLW0 |= UCSWRST;                   // Put eUSCI in reset
    P1SEL0 |= (BIT2 | BIT3);                //TX & Rx Primary mode
    P1SEL1 &= ~(BIT2 | BIT3);

    // Select Frame parameters and clock source
    UCA0CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK;
    UCA0CTLW0 &= ~EUSCI_A_CTLW0_PEN         //parity disabled
              & ~EUSCI_A_CTLW0_MODE0        // set to uart mode
              & ~EUSCI_A_CTLW0_MODE1
              & ~EUSCI_A_CTLW0_MSB          //LSB first
              & ~EUSCI_A_CTLW0_SEVENBIT     //8 bit character length
              & ~EUSCI_A_CTLW0_SPB;         //one stop bit one start bit is default
    //Next couple lines does baud rate
    UCA0MCTLW = 0xB5A1;
```

```
    UCA0BR0 = 0x01;
    UCA0BR1 = 0x00;
    UCA0CTLW0 &= ~UCSWRST;                  // Initialize eUSCI
  //EUSCI_A0->IE |= EUSCI_A_IE_TXIE;        //enable transmit interrupts
    EUSCI_A0->IE |= EUSCI_A_IE_RXIE;        //enable receive interrupts
#endif
#ifdef UCA2
    UCA2CTLW0 |= UCSWRST;                   // Put eUSCI in reset
/* Set Pins */
    P3SEL0 |= (BIT2 | BIT3);                // TX & RX Primary mode
    P3SEL1 &= ~(BIT2 | BIT3);
/* Select Frame parameters and clock source */
    UCA2CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK;
    UCA2CTLW0 &= ~EUSCI_A_CTLW0_PEN         // Parity disabled
              &  ~EUSCI_A_CTLW0_MODE0       // Set to uart mode
              &  ~EUSCI_A_CTLW0_MODE1
              &  ~EUSCI_A_CTLW0_MSB         // LSB first
              &  ~EUSCI_A_CTLW0_SEVENBIT    // 8 bit character length
              &  ~EUSCI_A_CTLW0_SPB;        // One stop bit one start bit is default
/* Baud Rate == 115200 */
    UCA2MCTLW = 0xB5A1;
    UCA2BR0 = 0x01;                         // Set Baud Rate
    UCA2BR1 = 0x00;
    UCA2CTLW0 &= ~UCSWRST;                  // Initialize eUSCI
  //UCA2IE |= EUSCI_A_IE_RXIE;              // Enable interrupt for RX receive
    NVIC_EnableIRQ(EUSCIA2_IRQn);           // Enable IRQ for UART
#endif
}
/*
 * Function: UART_send_byte
 * ----------------------------
 *   Sends only 1 byte of data through EUSCI_A2 Transmit Buffer.
 *   Is also used for the UART_send_n.
 *   Remember to use P3.3 for transmit.
 *   Baud Rate = 115200.
 */
void UART_send_byte(uint8_t tx_data){
    while(EUSCI_A_IFG_TXIFG & ~UCA2IFG);    // While there is a Transmit flag
    EUSCI_A2->TXBUF = tx_data;              // TX is the data that you want to
transmit
}
/*
 * Function: UART_send_n
 * ----------------------------
 *   Sends multiple bytes of data with the usage of multiple
 *   calls to UART_send_byte.
 *   Baud Rate = 115200.
 */
void UART_send_n(uint8_t * data, uint32_t length){
    volatile int i = 0;                     // Initialize counter
    for(i; i < length; i++){
        char test = data[i];
        UART_send_byte(data[i]);            // Loop through data and send
    }
}
```

```
#ifdef UCA0
/*
 * Function: UART_sent_byte
 * ---------------------------
 *   Sends only 1 byte of data through EUSCI_A0 Transmit Buffer.
 *   Is also used for the UART_send_n.
 *   Remember to use a microusb for data transmission.
 *   Ports vary from computer to computer with Realterm.
 *   Baud Rate = 115200.
 */
void UART_sent_n(uint8_t * data, uint32_t length){
    uint32_t i = 0;
    while(i < length)
    {
        UART_sent_byte(data[i]);
        i++;
    }
}
/*
 * Function: UART_sent_n
 * ---------------------------
 *   Sends multiple bytes of data with the usage of multiple
 *   calls to UART_send_byte.
 *   Baud Rate = 115200.
 */
void UART_sent_byte(uint8_t data){
    while(!(EUSCI_A0->IFG & BIT1));
    EUSCI_A0->TXBUF = data;
}
#endif
```

```
/*
 * uart.h
 *
 *  Created on: Nov 25, 2017
 *      Author: Adam
 */

#include "msp.h"

#ifndef UART_H_
#define UART_H_

void UART_Configure(); /* UART Configuration */
void UART_sent_n(uint8_t * data, uint32_t length); /* Send multiple bytes of data */
void UART_sent_byte(uint8_t data); /* Send a byte of data */
void UART_send_n(uint8_t * data, uint32_t length); /* Send multiple bytes of data */
void UART_send_byte(uint8_t data); /* Send a byte of data */

#endif /* UART_H_ *
```