```c
#include <stdio.h>
#include <stdint.h>
#include <math.h>
#include "eecs388_lib.h"
#include "metal/i2c.h"


struct metal_i2c *i2c;
uint8_t bufWrite[9];
uint8_t bufRead[1];


//The entire setup sequence
void set_up_I2C(){
    uint8_t oldMode;
    uint8_t newMode;
    _Bool success;


    bufWrite[0] = PCA9685_MODE1;
    bufWrite[1] = MODE1_RESTART;
    printf("%d\n",bufWrite[0]);

    i2c = metal_i2c_get_device(0);

    if(i2c == NULL){
        printf("Connection Unsuccessful\n");
    }
    else{
        printf("Connection Successful\n");
    }

    //Setup Sequence
    metal_i2c_init(i2c,I2C_BAUDRATE,METAL_I2C_MASTER);
    success =
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,2,bufWrite,METAL_I2C_STOP_DISABLE);//reset
    delay(100);
    printf("resetting PCA9685 control 1\n");

    //Initial Read of control 1
    bufWrite[0] = PCA9685_MODE1;//Address
    success = metal_i2c_transfer(i2c,PCA9685_I2C_ADDRESS,bufWrite,1,bufRead,1);//
initial read
    printf("Read success: %d and control value is: %d\n", success, bufWrite[0]);

    //Configuring Control 1
    oldMode = bufRead[0];
    newMode = (oldMode & ~MODE1_RESTART) | MODE1_SLEEP;
    printf("sleep setting is %d\n", newMode);
    bufWrite[0] = PCA9685_MODE1;//address
    bufWrite[1] = newMode;//writing to register
    success =
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,2,bufWrite,METAL_I2C_STOP_DISABLE);//sleep
    bufWrite[0] = PCA9685_PRESCALE;//Setting PWM prescale
    bufWrite[1] = 0x79;
    success =
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,2,bufWrite,METAL_I2C_STOP_DISABLE);//sets
prescale
    bufWrite[0] = PCA9685_MODE1;
    bufWrite[1] = 0x01 | MODE1_AI | MODE1_RESTART;
    printf("on setting is %d\n", bufWrite[1]);
    success =
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,2,bufWrite,METAL_I2C_STOP_DISABLE);//awake
    delay(100);
    printf("Setting the control register\n");
    bufWrite[0] = PCA9685_MODE1;
    success = metal_i2c_transfer(i2c,PCA9685_I2C_ADDRESS,bufWrite,1,bufRead,1);//
initial read
    printf("Set register is %d\n",bufRead[0]);

}

void breakup(int bigNum, uint8_t* low, uint8_t* high){
        //Defining the breakup function
    /*
        Task 1: breaking 12 bit into two 8-bit
        Define the function created that recieves a 12 bit number,
        0 to 4096 and breaks it up into two 8 bit numbers.

        Assign these values to a referenced value handed into
        the function.

        ex:
        uint8_t variable1;
        uint8_t variable2;
        breakup(2000,&variable1,&variable2);
        variable1 -> low 8 bits of 2000
        variable2 -> high 8 bits of 2000


    */
   // we did a bitwise and opperation on bigNum to get the lower 8 bits

   //takes in 12 bit in
   // 000000 000000
   // breaks up into 2 8 bit sections
   // high = (0000) 0000     low = 00 000000
    *low = bigNum & 0xFF;
    // we did a bitshift operation on bigNum to get the higher 8 bits
    *high = (bigNum >> 8);
}

void steering (int angle) {
    //Changing Steering Heading
    /*
        Task 2: using getServoCycle(), bufWrite, bufRead,
        breakup(), and and metal_i2c_transfer(), implement
        the function defined above to control the servo
        by sending it an angle ranging from -45 to 45.

        Use the getServoCycle function to get the value to
        breakup.

        ex:
        int valToBreak = getServoCycle(45);
        >>>sets valToBreak to 355

        note: the motor's speed controller is either
        LED0 or LED1 depending on where its plugged into
        the board. If its LED1, simply add 4 to the LED0
        address

        ex: steering(0); -> driving angle forward
    */
uint8_t low;
uint8_t high;
// we take in a servo angle and we split the angle into high 8 bits and low 8 bits
// then we use a pointer to write the 8 bits to our variable
breakup(getServoCycle(angle), &low, &high);
// set the origin of the address
bufWrite[0] = 0x0A;
// start setting the pin to high until the off time is set
bufWrite[1] = 0;
bufWrite[2] = 0;
bufWrite[3] = low;
bufWrite[4] = high;
// the address of the PCA9685 is at 0x40 and we will write sequentally
(METAL_I2C_STOP_ENABLE) from the bufWrite
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,5,bufWrite,METAL_I2C_STOP_ENABLE);
}

    /*
    ############################################################
        ATTENTION: The following section will cause the
        wheels to move. Confirm that the robot is
        Propped up to avoid it driving away, as well as
        that nothing is touching the wheels and can get
        caught in them

        If anything goes wrong, unplug the hifive board from
        the computer to stop the motors from moving

        Avoid sticking your hand inside the
        car while its wheels are spinning
    #############################################################
    */

void stopMotor(){
        //Motor config/stop. This will cause a second beep upon completion
    /*
        -Task 3: using bufWrite, bufRead, breakup(), and
        and metal_i2c_transfer(), implement the funcion made
        above. This function Configure the motor by
        writing a value of 280 to the motor.

        -include a 2 second delay after calling this function
        in order to calibrate

        -Note: the motor's speed controller is either
        LED0 or LED1 depending on where its plugged into
        the board. If its LED1, simply add 4 to the LED0
        address

        ex: stopMotor();
    */
uint8_t low = 0;
uint8_t high = 0;
```

```
breakup (280, &low, &high) ;
// set the origin of the address
bufWrite[0] = 0x06;
// start setting the pin to high until the off time is set
bufWrite[1] = 0;
bufWrite[2] = 0;
bufWrite[3] = low;
bufWrite[4] = high;
// the address of the PCA9685 is at 0x40 and we will write sequentally
(METAL_I2C_STOP_ENABLE) from the bufWrite
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,5,bufWrite,METAL_I2C_STOP_ENABLE);
}

void driveForward(uint8_t speedFlag){
        //Motor Forward
    /*
        -Task 4: using bufWrite, bufRead, breakup(), and
        and metal_i2c_transfer(), implement the function
        made above to Drive the motor forward. The given
        speedFlag will alter the motor speed as follows:

        speedFlag = 1 -> value to breakup = 313
        speedFlag = 2 -> value to breakup = 315(Optional)
        speedFlag = 3 -> value to breakup = 317(Optional)

        -note 1: the motor's speed controller is either
        LED0 or LED1 depending on where its plugged into
        the board. If its LED1, simply add 4 to the LED0
        address or type and replace LED1 with LED0

        ex: driveForward(1);
    */
uint8_t low = 0;
uint8_t high = 0;

if (speedFlag == 1){
    breakup (313, &low, &high);
}
else if (speedFlag == 2){
    breakup (315, &low, &high);
}
else if (speedFlag == 3){
    breakup (317, &low, &high);
}
// set the origin of the address
bufWrite[0] = 0x06;
// start setting the pin to high until the off time is set
bufWrite[1] = 0;
bufWrite[2] = 0;
bufWrite[3] = low;
bufWrite[4] = high;
// the address of the PCA9685 is at 0x40 and we will write sequentally
(METAL_I2C_STOP_ENABLE) from the bufWrite
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,5,bufWrite,METAL_I2C_STOP_ENABLE);
}

void driveReverse(uint8_t speedFlag){
        //Motor Reverse
    /*
```

```
        -Task 5: using bufWrite, bufRead, breakup(), and
        and metal_i2c_transfer(), implement the function
        made above to Drive the motor backward. The given
        speedFlag will alter the motor speed as follows:

        speedFlag = 1 -> value to breakup = 267
        speedFlag = 2 -> value to breakup = 265(Optional)
        speedFlag = 3 -> value to breakup = 263(Optional)

        -note 1: the motor's speed controller is either
        LED0 or LED1 depending on where its plugged into
        the board. If its LED1, simply add 4 to the LED0
        address or type and replace LED1 with LED0

        ex: driveReverse(1);
    */
uint8_t low = 0;
uint8_t high = 0;
if (speedFlag == 1){
    breakup (267, &low, &high);
}
else if (speedFlag == 2){
    breakup (265, &low, &high);
}
else if (speedFlag == 3){
    breakup (263, &low, &high);
}
// set the origin of the address
bufWrite[0] = 0x06;
// start setting the pin to high until the off time is set
bufWrite[1] = 0;
bufWrite[2] = 0;
bufWrite[3] = low;
bufWrite[4] = high;
// the address of the PCA9685 is at 0x40 and we will write sequentally
(METAL_I2C_STOP_ENABLE) from the bufWrite
metal_i2c_write(i2c,PCA9685_I2C_ADDRESS,5,bufWrite,METAL_I2C_STOP_ENABLE);
}

void raspberrypi_int_handler(int devid, int * angle, int * speed, int * duration)
{
    char str_buffer[11];
    int my_angle, my_speed, my_duration;
   // Extract the values of angle, speed and duration inside this function
   // And place them into the correct variables that are passed in

    ser_readline(devid, 11, str_buffer);
    sscanf(str_buffer, "%d %d %d", &my_angle, &my_speed, &my_duration);
    * angle = my_angle;
    * speed = my_speed;
    * duration = my_duration;
    //free(str_buffer);
}


int main()
{
    // Initialize I2C
    set_up_I2C();
    delay(2000);

    // Calibrate Motor
    printf("Calibrate Motor.\n");
    stopMotor();
    delay(2000);

    // initialize UART channels
    ser_setup(0); // uart0 (receive from raspberry pi) /dev/ttyAMA1
    ser_setup(1);

    printf("Setup completed.\n");
    printf("Begin the main loop.\n");

    int angle, speed, duration;
    while (1){
        if (ser_isready(1)){
            raspberrypi_int_handler(1, &angle, &speed, &duration);
            if ((angle < 46) && (angle > -46) && (duration != 0))
            {
            printf("%d, %d, %d\n", angle, speed, duration);
            steering(angle);
            if (speed == 0)
            {
                stopMotor();
            } else if (speed > 0)
            {
                driveForward(speed);
            } else if (speed < 0)
            {
                driveReverse(-speed);
            }
            }
        }
    }

}
```