EECS 473 Final Exam
Fall 2021

Name: _____**Key**_____ unique name: _____**Key**_____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

_____

It is quite possible this key has errors, we mostly didn't use it to grade, just as a draft set of answers. The design question likely has version issues as this was done before final editing.
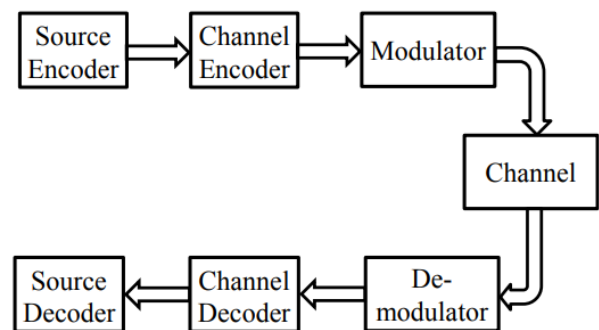
NOTES:

1. Closed book and Closed notes
2. <u>**Do not write anything you want graded on the back pages of the exam.**</u>
3. There are <u>**15**</u> pages total for the exam. There is also a references handout.
4. Calculators are allowed, but no PDAs, Portables, Cell phones, etc. Using a calculator to store notes is not allowed nor is a calculator with any type of wireless capability.
5. You have about 120 minutes for the exam.
6. Be sure you answer each question on the appropriate page.
7. When doing calculations, you must show your work to get credit.

Be sure to show work and explain what you've done when asked to do so. That may be very significant in the grading of this exam.

dBm	mW	dBm	mW	dBm	mW
-3	0.5	9	8	21	126
-2	0.6	10	10	22	158
-1	0.8	11	13	23	200
0	1.0	12	16	24	250
1	1.3	13	20	25	316
2	1.6	14	25	26	398
3	2.0	15	32	27	500
4	2.5	16	40	28	630
5	3.2	17	50	29	800
6	4	18	63	30	1000
7	5	19	79	33	2000
8	6	20	100	36	4000

Source Encoder → Channel Encoder → Modulator → Channel → De-modulator → Channel Decoder → Source Decoder
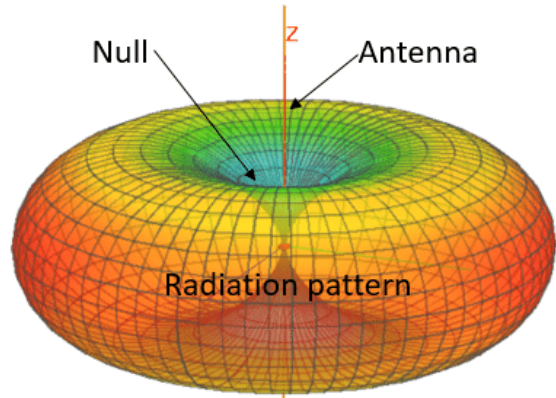
$$C = B \log_2 \left(1 + \frac{S}{N}\right)$$

$$r = \frac{10^{(P_t + g_t + g_r - p_r)/20}}{41.88 \times f}$$

$$\sum_{i=1}^{n} U \le n(2^{1/n} - 1)$$

1. Multiple choice. Circle the correct answer or fill in the blank.

   [12 points, -2 per wrong or blank, minimum 0]

   a) The figure to the right is the radiation pattern of a(n) *isotropic / **dipole** / dish* antenna.

Null↖   Antenna
Radiation pattern

   b) What does Shannon's limit describe?
   - *That you can only move so much current through a fixed-diameter wire.*
   - *That communication over a noisy, bandwidth limited, channel has an absolute limit on the amount of information that can be communicated in a given amount of time.*
   - *The faster you discharge a battery, the less total useful energy that can be extracted.*
   - *That given enough time, all software will be licensed under a viral license such as the GPL*

   c) On a PCB running low-power traces, most low-cost PCB manufacturers typically require a clearance between traces of *about 1 / **5** / 25 / 100* thous.

   d) Using RM scheduling, you can schedule your tasks for certain, no matter how many tasks you have, if your CPU utilization is **100% / 82% / 88% / 75% / 70%** or lower (pick the largest that is true). Question tossed out. It's ~69.2% or so.

   e) The range of representation of a signed 4-bit Q3 number is from ___-1___ to ___7/8_____

   f) In the context of communication theory, a "source encoder"
   - *compresses a message*   *adds error correction to a message*
   - *adds guard bits to a message*   *modulates the signal*

   g) The Peukert Effect describes the phenomenon that as we draw more current from a battery
   - *the battery gets hotter.*
   - *the actual capacity of the battery drops.*
   - *the battery is more and more likely to be unable to be recharged.*
   - *the cathode of the battery is more likely to get coated by the electrolyte*

   h) Assuming we are encoding our data as d[1:4]p[1:3], when using the scheme shown, we would treat the packet 0001001 as having

   sent what 4 bits of data? _1001___
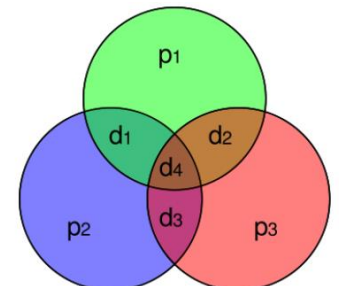
2. Consider the Shannon-Hartley theorem as stated here: **[10 points]**

    a) Circle each of the following that are true. **[6, -2 per wrong answer min 0]**

$$C = B \log_2 \left(1 + \frac{S}{N}\right)$$

   - **C** represents the *channel capacity,* that is the maximum rate that of bits of information plus error correction that can be reliably sent.
   - **B** represents the *bandwidth* available. So if the data were sent using only the frequencies between 2.35 GHz and 2.4 GHz, the bandwidth would be 50 MHz.
   - **S** represents the *signal* measured in units of power.
   - **N** represents the *noise* measured in units of voltage.
   - This formula relies on some assumptions. One such assumption is that all noise is Gaussian.
   - This formula implies that with no noise of any type, the channel capacity would be infinite.

    b) If the SNR is -20 dB and the bandwidth available is 1 MHz, what is the channel capacity? **[4]**

   1Mbit/sec*log$_2$(1.01)=14355 bits/second.

3. Say we have a 7.2V 2.0Ah battery. We are using it through a regulator which outputs 5V to a processor that uses 20mW. Nothing other than the processor is driven by the regulator or battery. You must clearly show your work to get credit. **[10 points]**

    a) How long would you expect the battery to last if the regulator were an otherwise ideal LDO that has a quiescent current of 1mA? **[5]**
    Current needed is 20mW/5V=4mA+1mA. So about 2000mAh/5mA=400 hours.

    b) How long would you expect the battery to last if the regulator were an otherwise ideal buck converter with a quiescent current of 2mA? **[5]**
    In reality this would get a bit tricky as the battery's output voltage would be dropping over time. That said, you'd need 2mA+20mW/7.2V=4.778mA. So 2000mAh/7.2mA~=419 hours.

4. Write a fixed-point function which takes two 8-bit Q4 numbers and returns the product as a 16-bit Q4 rounding to the nearest value (ties may round either way) if needed. You may assume there is no overflow. Also assume that ints are 32-bits, shorts are 16 bits and chars are 8 bits. We have provided the start of the function for you. **[7 points]**

```
short Qmult(char a, char b)
{

    return( (a*b+8)>>4);

}
```

5. Short answer **[7 points]**:
   a) If you have a file named /dev/gpio23 in your Linux directory, provide a one-line Linux command which you could reasonably expect will cause pin 23 to go high. **[3]**

   echo -n 1 >> /dev/gpio23 (We won't be picky on -n or not)

   b) When designing a 2-layer PCB, it is common ground-fill one side and power-fill the other. Describe what that means and _briefly_ explain why we do it. **[4]**

   It means we fill in the metal layer of one side of the PCB with as much metal as we can and put ground on one side and power on the other. It provides a bit more capacitance (with very low parasitics) which for keeping power integrity, especially at high frequencies. (this is more detail than needed…)

6. Wireless communication **[10 points]**
    a) One concern when sending data wirelessly is that a packet may get lost (dropped). Assuming packet drops are rare and the data is of a type that requesting a resend of the data makes sense (so not live voice perhaps), how could the receiver know a packet has been dropped? You may not assume the data is being sent at a constant rate. **[4]**

        Generally this is done by having a packet number sent with each packet. If the receiver gets packet 22 and 24, it can be assumed that 23 got dropped.

    b) In free space, you would expect that signal centered at exactly 128MHz would go how much farther than one centered at 1MHz? Circle one. **[2]**
        - About the same distance   About 7 times as far
        - About 1/7 as far   About 128 times as far
        - About 1/128 as far   About 1/11.3 as far

    c) Label each figure as one following: nFSK, nPSK, nQAM, or nASK where you are to supply the n (so 4FSK or 8QAM) for example. If more than one answer applies, you may select any. **[4]**
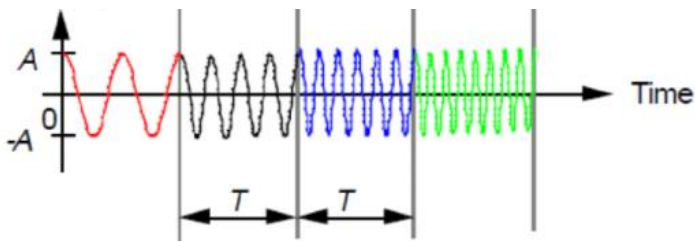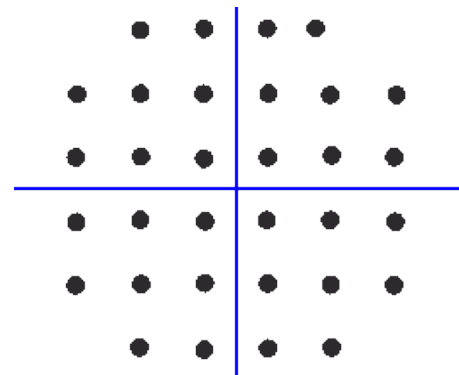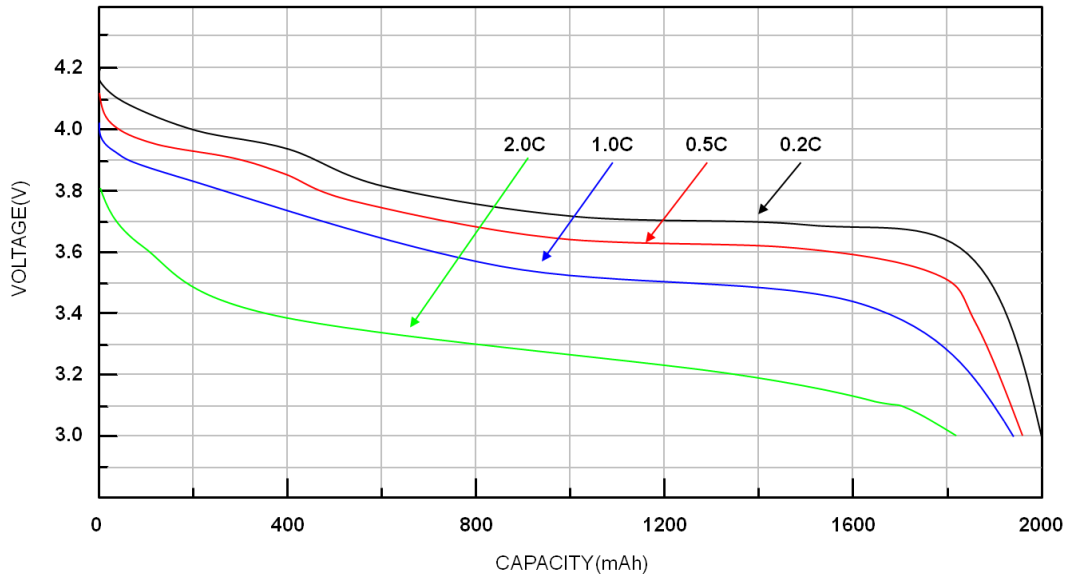
4FSK
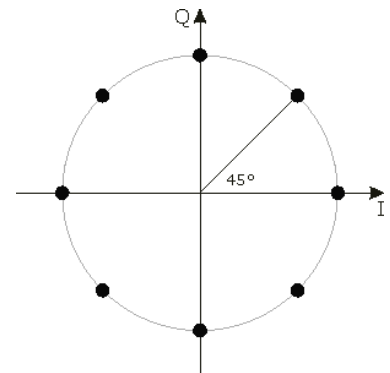
8PSK

8ASK

32QAM

7. Consider the following LIPO battery discharge characteristics. **[5 points]**

If you draw 4A from this battery, about how long will it last before the voltage drops below 3.2V? Show your work.

Looks like a capacity of 2000mAh. So 4A is 2C. That would give about 1200mAh. 1.2Ah/4A=20 minutes.

You should read the entire question before starting.

The Problem
It turns out you did well enough with the satellite Electrical Power System (from the midterm), that you've now been tasked with interfacing with the various satellite elements in the **Attitude Determination and Control System (ADACS)**. The ADACS is responsible for using sensors to determine the attitude (i.e. rotational state: angular position and angular velocity) of the satellite, then using actuators to control the attitude so that it matches what is needed for mission parameters (e.g. pointing a very sensitive and very expensive camera at the earth rather than the blinding sun).

You have been tasked with designing an early prototype to interface with the sensors and actuators of a satellite ADACS. You will NOT have to develop the attitude determination algorithm or the attitude control algorithm, as that will be the responsibility of another part of the team. The system has the following components:

1. 1x Arduino Uno Board
2. 1x Sun Sensor custom-built by the electrical hardware team. This device contains three photodiodes with transimpedance amplifiers so that there are three analog voltage signals between 0 and 5V that correspond to how much light is hitting each photodiode. These analog voltages can be used to calculate a "sun vector" needed to determine attitude.
3. 1x LIS3MDL Magnetometer: This sensor measures the earth's magnetic field vector, which is used as the second vector necessary to determine attitude. You should communicate with the Magnetometer with I2C
4. 2x Magnetorquers placed in 2 perpendicular axes: These are coils of wire that form electromagnets, which can be controlled with PWM signals **(just like a motor)**. These electromagnets interact with the earth's magnetic field to cause a torque that rotates the satellite. They should be driven with +10V when active. The arrows point in the direction of "forward" current.
5. 1x LD293D H-Bridge Driver: This H-bridge driver can be used to control the direction of current flow in the magnetorquers so that the magnetic field can be generated in both directions.
6. 1x 10V Power Rail + GND coming from the EPS
7. Any other passives or connectors you may need.

Arduino: = 6
Lis3mdl: 1, 6-8, 14, 17-18, 23-28 = 12
AN4602: 1, 7-9, 13 = 5
L293D: 1, 3, 8 = 3You will be implementing 1 interrupt service routine, and 2 of 3 tasks in FreeRTOS. Assume you have a special Arduino where the FreeRTOS Tick is set to 1ms

- **ISR:** `drdy_isr()` should be run every time there is new data from the magnetometer. It should read the current time and the raw sensor data and store them in global variables:

  ```
  unsigned long sense_time; //read with the millis() function call
  uint8_t mag_bytes[6]; //raw magnetometer data
  ```

```
unt16_t sun_data[3]; //raw sun sensor data
```

After storing the values, the ISR should release task 1.

- **TASK 1:** `task1_sensors` will run at medium priority. The task should perform appropriate calculations on the magnetometer and sun sensor data to convert it into milligauss and millivolts respectively, and then call a special function:

```
void send_data(unsigned long sense_time, float *mag_mgauss,
                                          float *sun_mvolts);
```

Which will enqueue the data for the control algorithm task.

- **TASK 2:** `task2_actuators` will run at high priority every 10ms. The task should read the global variables:

```
float magtor_duty_cycles[2]; //float from 0.0 to 100.0 (percent)
bool magtor_dirs[2]; //0 - Forward, 1 - Reverse
```

and send out appropriate signals that will control the magnetic fields generated by the magnetorquers. The global variables will be set by the control algorithm task.

- **TASK 3:** `task3_control` will be running at lowest priority, but **YOU WILL NOT BE IMPLEMENTING THIS TASK.** You only need to initialize the task.

Part 1
Provide the connections between the different components of the system. You should also provide power and GND to all components. Add resistors and capacitors as needed. **Use labels where applicable to make your connections easier to understand.**
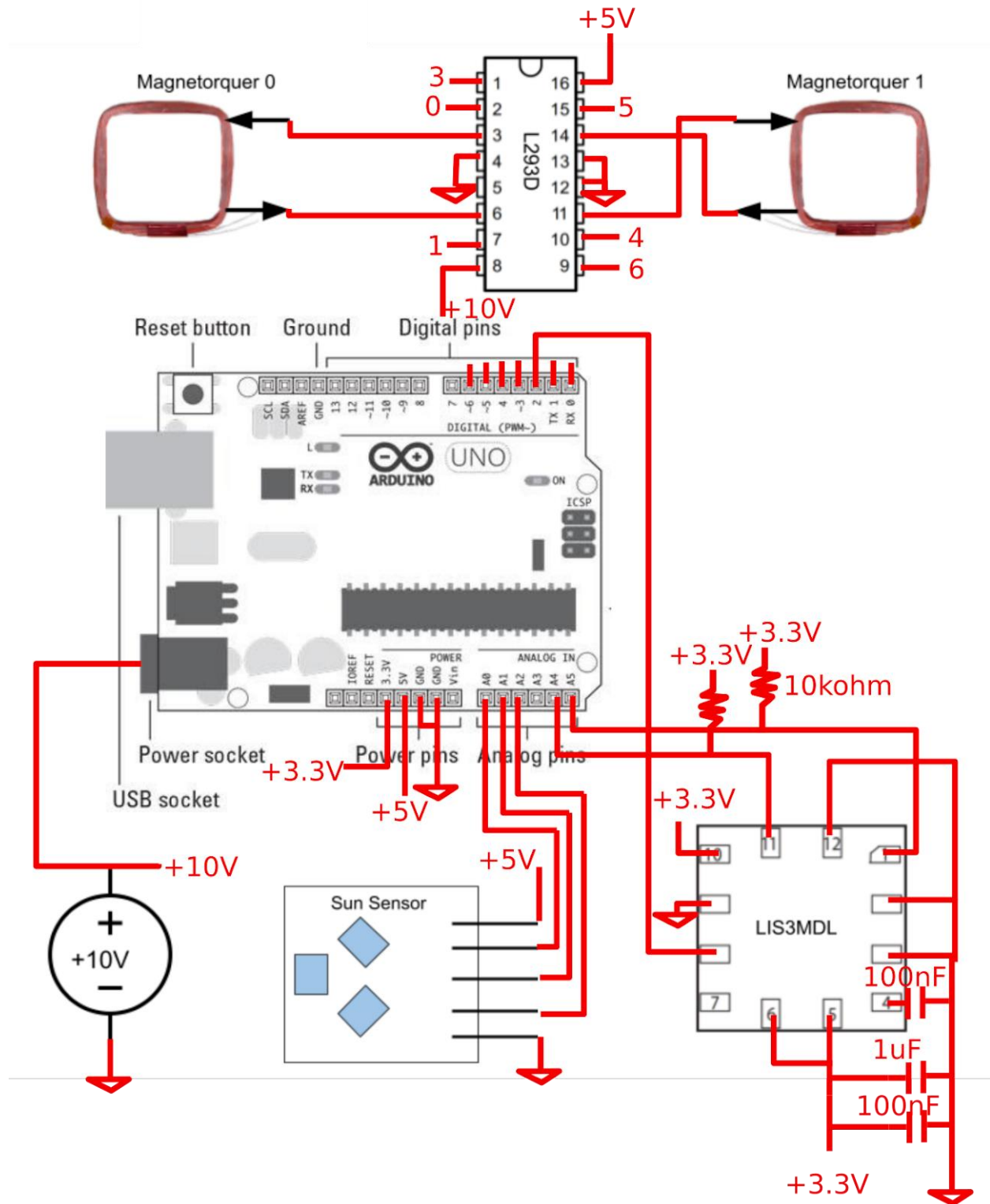
Part 2
Write some helper functions that will let you communicate and initialize the magnetometer in the following way:
1. All 3 axes are in Ultra-High-Performance mode with an Output Data Rate of 80 Hz.
2. The full scale range is +-4 gauss
3. The device is in continuous conversion mode

```
//I2C address depends on wiring in part A
#define I2C_ADDR 0x1C

void lis3mdl_write(uint8_t reg_addr, unit8_t value) {

    Wire.beginTransmission(I2C_ADDR);

    Wire.write(reg_addr);  //send register address first
    Wire.write(value); // Write Byte

    Wire.endTransmission(); //Sends everything out

}


void lis3mdl_read(uint8_t reg_addr, uint8_t *bytes, uint8_t count) {

    Wire.beginTransmission(I2C_ADDR);
    Wire.write(reg_addr);  //send register address first
    Wire.endTransmission(); //Sends everything out

    Wire.requestFrom(I2C_ADDR, count); //Request count bytes


    for (int i = 0; i < count; i++) {
        bytes[i] = (Wire.available() > 0) ? Wire.read() : 0;
    }
}


int lis3mdl_init() {
    lis3mdl_write(0x20, 0b01111100) //Enable X&Y 80Hz UHP
    lis3mdl_write(0x21, 0b00000000) //Set Full scale +-4 gauss (default)
    lis3mdl_write(0x22, 0b00000000) //Set to continuous conversion
    lis3mdl_write(0x23, 0b00001100) //Enable Z UHP




}
```

Part 3: Setup Function

Write a function to initialize the I2C Connection, the magnetometer, set up the interrupt `drdy_isr` based on the signal from the LIS3MDL, set up any necessary semaphores, and initialize the tasks.

```
#include <I2C.h>
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <time.h>

unsigned long sense_time; //read with the millis() function call
int8_t mag_bytes[6]; //raw magnetometer data
uint16_t sun_data[3]; //raw sun sensor data

float magtor_duty_cycles[3] = {0}; //float from 0.0 to 100.0 (percent)
bool magtor_dirs[3] = {0};   //0 - Forward, 1 - Reverse

void send_data(unsigned long sense_time, float *mag_mgauss, float
*sun_mvolts);

SemaphoreHandle_t sem_1 = NULL;

void setup(void){
    Wire.begin();  //Initialize I2C
    lis3mdl_init(); // Initialize Magnetometer

    //Initialze H-Bridge
    pinMode(0, OUTPUT);   //Initialize A1
    pinMode(1, OUTPUT);   //Initialize A2
    pinMode(3, OUTPUT);   //Initialize EN1
    pinMode(4, OUTPUT);   //Initialize A3
    pinMode(5, OUTPUT);   //Initialize A4
    pinMode(6, OUTPUT);   //Initialize EN2

    //Initialize Interrupt
    pinMode(2, INPUT);
    attachInterrupt(2, drdy_isr, RISING);

    sem1 = xSemaphoreCreateBinary(); //Initialize Semaphore

    xTaskCreate(task1_sensors, "Task 1", 200, NULL, 2, NULL);
    xTaskCreate(task2_ADAC, "Task 2", 200, NULL, 3, NULL);
    xTaskCreate(task3_actuators, "Task 3", 200, NULL, 1, NULL);
    vTaskStartScheduler(); //Start Scheduler
}
```

Part 4: ISR & Task 1
Implement the ISR and task 1 so that the current time and sensor data is read, converted to appropriate values, and sent to the control algorithm every time data is ready.

```
void drdy_isr() {
      BaseType_t xHigherPriorityTaskWoken = pdFalse;
      sense_time = millis();

      lis3dml_read(0x28, mag_bytes, 6)
      sun_data[0] = analogRead(A0);
      sun_data[1] = analogRead(A1);
      sun_data[2] = analogRead(A2);

      xSemaphoreGiveFromISR(sem1, &xHigherPriorityTaskWoken);
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}


void task1_sensors(void *pParam){
      float mag_mgauss[3];
      float sun_mvolts[3];

      for(;;) {
            if(xSemaphoreTake( sem1, portMAX_DELAY) == pdTrue)) {

                  for (int i = 0; i < 3; i++) {
                        int16_t temp;
                        temp = ((mag_bytes[2*i+1]<<8) | mag_bytes[2*i]);
                        mag_mgauss[i] = (float) temp * 1000/6842;
                        sun_mvolts[i] = (float) sun_data[i] * 5000/1024;
                  }


                  send_data(sense_time, mag_mgauss, sun_mvolts);

            }
      }




}
```

Part 5: Task 2
Implement task 2 to use the magnetometers as calculated by the control algorithm in Task 3.

```
void task2_actuators (void *pParam) {

        TickType_t xLastWakeTime;

        xLastWakeTime = xTaskGetTickCount();

        for(;;) {

                //Set values for Magnetorquer 0
                if(magtor_dirs[0] == 0) {
                        digitalWrite(0, HIGH);  //A1 High
                        digitalWrite(1, LOW);  //A2 Low
                }
                else {
                        digitalWrite(0, LOW);  //A1 Low
                        digitalWrite(1, HIGH);  //A2 High
                }

                //Set PWM for EN1
                analogWrite(3, (uint8_t)(magtor_duty_cycles[0]*255/100.0));

                //Set values for Magnetorquer 1
                if(magtor_dirs[1] == 0) {
                        digitalWrite(4, HIGH);  //A3 High
                        digitalWrite(5, LOW);  //A4 Low
                }
                else {
                        digitalWrite(4, LOW);  //A3 Low
                        digitalWrite(5, HIGH);  //A4 High
                }

                //Set PWM for EN2
                analogWrite(6, (uint8_t)(magtor_duty_cycles[1]*255/100.0));

                vTaskDelayUntil( &xLastWakeTime, pdMS_TO_TICKS(10));
        }




}
```