EECS 473 Final Exam--KEY

Fall 2022

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.

____

NOTES:

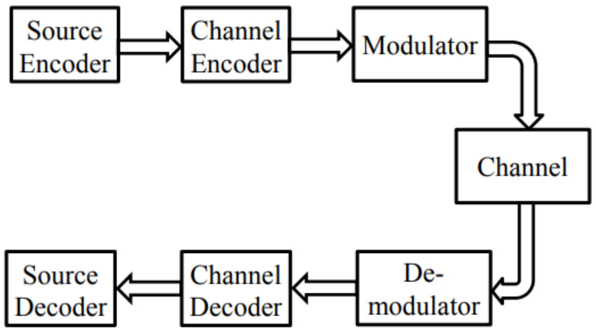
1. Closed book and Closed notes
2. There are **12** pages total for the exam. There is also a references handout.
3. Calculators are allowed, but no PDAs, Portables, Cell phones, etc. Using a calculator to store notes is not allowed nor is a calculator with any type of wireless capability.
4. You have about 120 minutes for the exam.
5. Be sure you answer each question on the appropriate page.

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
-3	0.5
-2	0.6
-1	0.8
0	1.0
1	1.3
2	1.6
3	2.0
4	2.5
5	3.2
6	4
7	5
8	6

dBm	mW
9	8
10	10
11	13
12	16
13	20
14	25
15	32
16	40
17	50
18	63
19	79
20	100

dBm	mW
21	126
22	158
23	200
24	250
25	316
26	398
27	500
28	630
29	800
30	1000
33	2000
36	4000

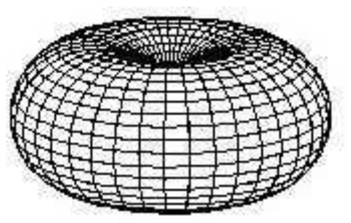


$$C = B\log_2\left(1+\frac{S}{N}\right)$$

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

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

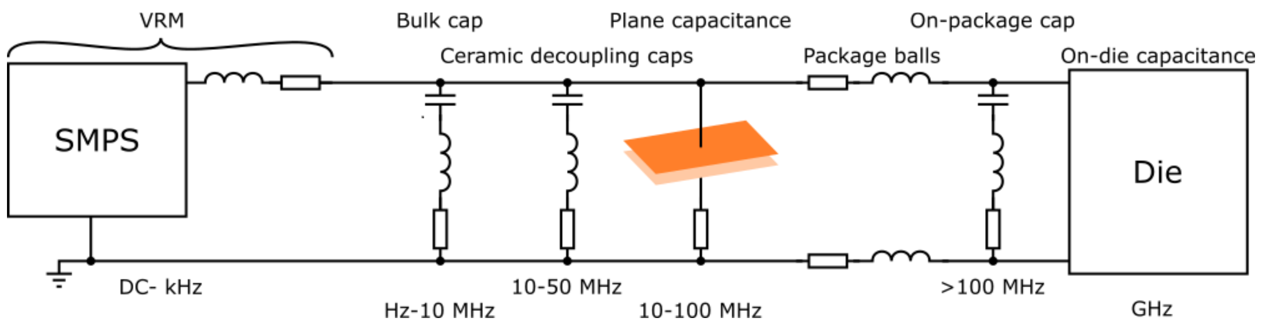
1. Multiple choice. Circle the correct answer or fill in the blank.
   [12 points, -2 per wrong or blank answer, minimum 0]



   a) The figure to the right is a "side view" of the radiation pattern of a/an ***isotropic / [✓ dipole] / dish / monopole*** antenna.

   b) The range of representation of a signed 4-bit Q2 number is ___**-2**______ to __**1.75**_____

   c) On a PCB, traces on two different layers of a board are connected by ***traces / solder mask / artwork / silk screen / [✓ vias].***



   d) In the context of the above figure, which of the following is ***false***?
      - ***The plane capacitance is the capacitance formed by the power and ground planes of the PCB***
      - ***The VRM is the "voltage regulation module" and is basically a control system focused attempting to maintain a constant output voltage.***
      - [✓ ***The on-package capacitors are generally placed on the PCB as close as possible to the die.***]
      - ***Ceramic capacitors can be expected to have the same or lower parasitics than the bulk capacitors.***

   e) In the context of communication theory, a "source encoder" ***[✓ compresses a message] / adds error correction to a message / fixes errors in a message / modulates the signal***.

   f) Say you are sending a signal using the band from 2.0GHz to 2.08 GHz the power of the signal at the receiver is seven times that of the ambient noise. According to the Shannon–Hartley theorem, the highest achievable data rate is *about* **25 / 100 / [✓ 250] / 600 / 2000** Mbits/sec

   g) Using RM scheduling, you can schedule your tasks for certain, no matter how many tasks you have, if your total CPU utilization is **100% / 82% / 78% / [✓ 68%] / 60%** or lower (pick the largest that is true).

2. Short answer: **[13 points]**

a) Say you have two different radios that are identical in all ways except one uses a 5 GHz signal and the other uses 2.4 GHz signal. Which would you expect to travel farther and by about how much? **[3]**

Per the formula on page 1, range is inversely proportional to frequency. So 2.4GHz would travel about 2x as far as the 5GHz one.

b) If you have three tasks that have a total CPU utilization of 40%, will rate monotonic scheduling *without preemption* always successfully schedule those tasks? If so, explain why. If not, provide an example that fails. **[6]**

Example:

Task 1 & Task 2: CPU time of 1 sec, period of 10 seconds
Task 3: CPU time of 2ms, period of 10ms.

Total CPU utilization is 40% (10%+10%+20%).
If task 1 or task 2 are running, task 3 will not be able to preempt and so will not be able to run until the other task ends and so will miss it's deadline. So it won't always successfully schedule the tasks.

c) How does priority inheritance address priority inversion? Your answer must be 25 words or less. **[4]**

Low-priority task is promoted to the priority of a task trying to use a shared resource so no task with an intermediate priority can interfere.

(It's hard to do a good job with this in 25 words, we tried to be generous on the grading)

3. Say we have a 7.2V battery with 2.0 Ah capacity to drive a device that needs 3.3V and has an average power draw of 100mW. You are to assume the battery maintains a 7.2V output until it is drained. **[8 points]**

   a) How long would the battery last if we use an ideal LDO? Clearly show your work. **[4]**

      2000mAh/(100mW/3.3V)=66 hours

   b) How long would the battery last if we use an ideal buck converter? Clearly show your work. **[4]**

      7.2V*2Ah=14.4Wh. 14.4Wh/0.1W=144h.

4. Say we are sending three bits of data (d1, d2, d3) and three bits of parity (p1, p2, p3). And say that the we generate the parity bits as follows (where P is the even 1s parity function as done in class)

   p1=P(d1, d2)
   p2=P(d2, d3)
   p3=P(d1, d3)

   So for example if d[1:3]=101 then p[1:3]=110.

   Would this scheme allow for the correction of one bit of the message? If yes, explain why. If not, provide an example where it would fail. **[8 points]**

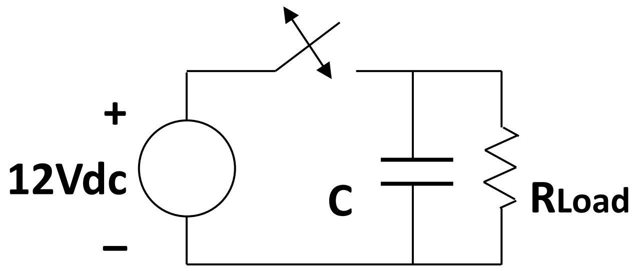
   There are a few ways to do this. One is to show that A) if any one data bit goes bad, there will be a *unique* group of two parity bits that will appear correct then B) show that if any one parity bit goes bad, that only that parity bit will appear to be bad. So we have a unique way to take the parity bits that go bad and figure out which bit flipped.

   You could also do this by showing the set of all legal encodings have a (set) Hamming distance of 3 and making a quick argument from there.

5. Below is a simplistic idea of what a buck converter might be. If the switch were open half the time, one should get an output voltage of 6V (half of Vin). However, real implementations will burn out the switch. **[6 points]**

   a) Why is that? **[2]**.

   When the switch closes, there will be a large current though the switch and the switch will likely burn out.

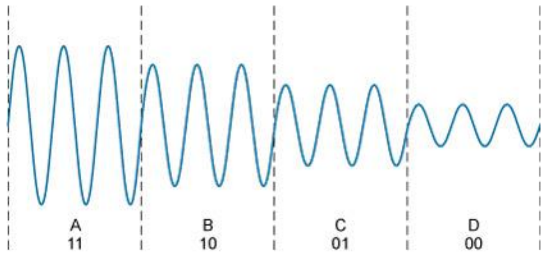


   b) How can we fix that problem? Either modify the above circuit or draw a new one. **[4]**

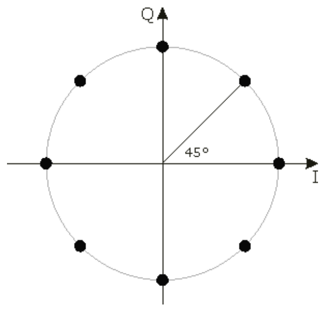
   (Didn't draw it) Add an inductor in series with the switch.

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

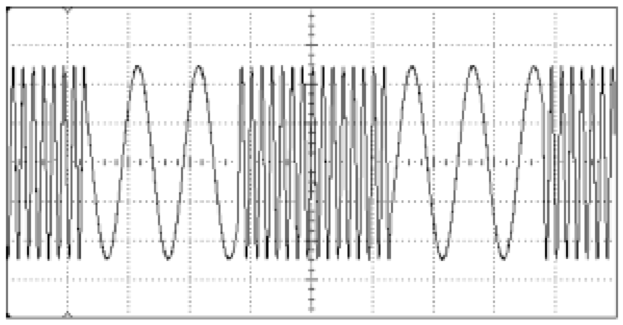
.



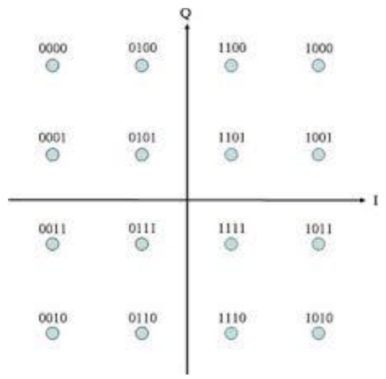
4 ASK



8 PSK

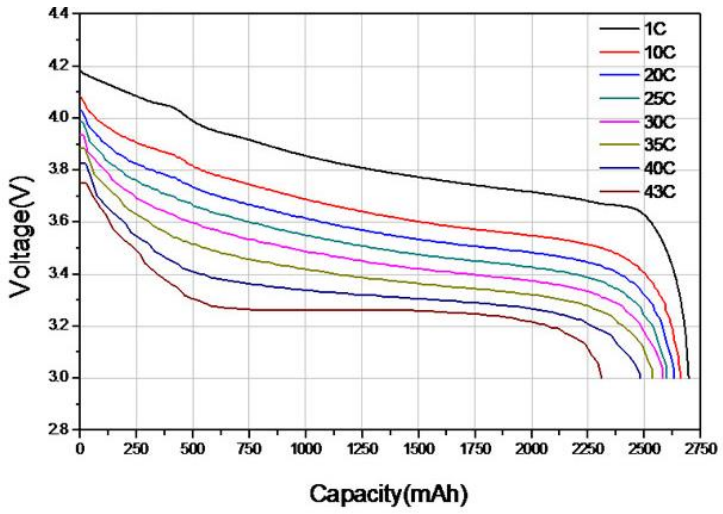


2 FSK



16 QAM

7. Consider the battery discharge curve on the right. Show your work for all questions.



a) Approximately how long would this battery be able to drive a device that requires 10mA @3.6V? [**3**]

At a lot less than 1C it would probably have ~2700mAh of effective capacity. But safest to use something closer to 2550mAh (which is about what we'd get for 1C). So 2550mAh/10mA=255h. We took a lot of similar numbers.

b) About how long would this battery be able to drive a device that requires 52A @3.4V? **[5]**

With a capacity of ~2600mAh, that's about 20C. At 20C we'd get about 2400mAh. 2.4Ah/52A~=0.046 or about 2.8 min.

8. Circle all the true statements. **[7 points, -2 per incorrectly circled/not circled answer, minimum 0]**

a) Linux user-space programs are generally expected to use memory-mapped I/O addresses to talk with I/O devices.

b) Alkaline batteries are a common type of primary-cell batteries while lead-acid batteries are a common type of secondary-cell batteries.

c) Deferred interrupts are generally used so that low-priority tasks can be done outside of the ISR.

d) On-off keying (OOK) can be viewed as a simple form of ASK.

e) A buck converter and a LDO both drive a lower voltage out than was provided as input.

f) An isotropic antenna is one that radiates power equally in all directions.

g) 32 QAM sends 8 bits of data per symbol.

Design problem: Bike Logger

Overview

You are working for a company developing embedded systems for competitive bikers. Your team is working on a new device that measures and logs bike speed solely using a sensor on the front wheel as well as the bike's angle of ascent/decent (what angle it is going up or down). You have been asked to implement a quick Arduino prototype using a pre-integrated wheel encoder sensor that is to fire an interrupt upon each revolution of the wheel and a ADXL345 measure the angle of ascent. This system will log data by sending information over a UART connection.

Project Requirements

You are to log two types of data: bike speed and angle of assent. All the data should be recorded with a timestamp where the timestamp should just be the value generated by the function `millis()`.

- **Ascent angle** should be sampled and sent at a rate of 10Hz. The format sent to the UART should be:
  ```
  T: xxxx A: aa.a
  ```
  Where xxxx is the integer timestamp and aa.a is the angle in degrees.

- **The bike speed data** should be sampled and sent at a rate of 1 Hz. The format sent to the UART should be:
  ```
  T: xxxx S: ss.s
  ```
  Where xxxx is the integer timestamp and ss.s is the bike speed in meters per second.

The exact number of characters used for each field can vary as you wish.

Parts available

You have the following parts available:

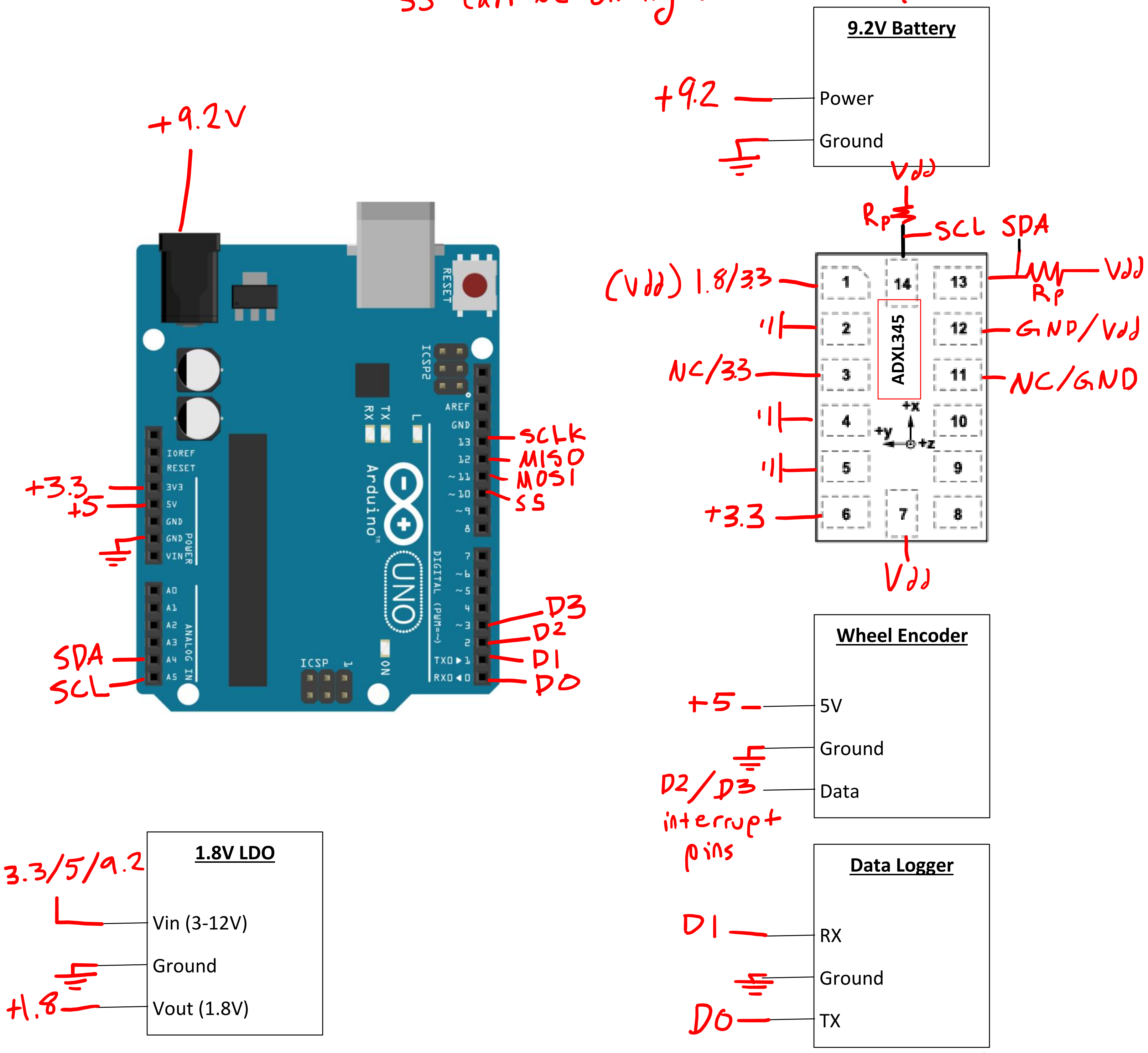
- **An Arduino UNO** with a version of FreeRTOS which has a tick rate of 1 ms rather than the normal tick rate of an UNO running FreeRTOS.
- **A wheel encoder**. It has only three terminals: 5V, ground, and data. Data goes high (5V) for 1ms every time the wheel has turned to a specific point and is otherwise low. The wheel has a circumference of 2.23m.
- **An ADXL345** accelerometer mounted so that the x-axis is facing forward at 0 degree angle.
- **A battery** with a nominal 9.2V output.
- **A UART data logger**. It just logs anything you send over it. It is running at 115200 baud (makes scheduling at such a high frequency easier).
- **An LDO** which can take inputs from 3.0V to 12.0V and generates 1.8V out.

Measuring angle using an accelerometer is a bit tricky on something like a bike. For sake of this question, assume there is no significant bouncing or other source of acceleration other than the angle change.

You are to use FreeRTOS and rate-monotonic scheduling when addressing this problem.

Part A: Wiring [8 points]
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. You may (and in fact *should*) use labels to make connections.

For SPI, connect SCLK to ADXL pin 14, MISO to 12, MOSI to 13, and SS to 7. SS can be on any arduino GPIO pin.



Part B: Some math and helper functions [3 points]

Assuming the circumference of the bike wheel is 2.23 meters, answer the following. Show your work.

a) How fast you are traveling (in m/s) if a wheel rotation takes 240ms? **[1]**

$$\frac{2.23\text{ m}}{.240\text{ s}} = 9.29\text{ m/s}$$

b) If you see exactly 0g of acceleration in a given axis if that axis is perpendicular to the gravity vector, what angle (in degrees) is that axis if it sees: **[2]** Physics: $\sin^{-1}(.03) = 1.7°$

- 0.03g? 1.7/2.7
- -0.03g? -1.7/-2.7

Note: Negative acceleration implies negative angle.

Linear: $90° \cdot .03 = 2.7°$

c) Write any helper functions that will help you with later parts. ***Any function you write here must start with the name help_*** (e.g. help_I2C_write() or help_read_it()). [0 points, this is really graded as part of the code that uses the functions]

//Could use I2C or SPI. Helpers for both provided here, only need one.

// I2C Helpers

```
#define WHEEL_CIRCUMFERENCE 2.23F
#define MS_PER_SECOND 1000U

void help_I2C_write(uint8_t device_addr, uint8_t reg_addr, uint8_t value) {
    Wire.beginTransmission(device_addr);
    Wire.write(reg_addr);
    Wire.write(value);
    Wire.endTransmission();
}

void help_I2C_read(uint8_t device_addr, uint8_t reg_addr, uint8_t *bytes,
uint8_t count) {
    Wire.beginTransmission(device_addr);

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

    Wire.requestFrom(device_addr, count); //Request count bytes
    for (int i = 0; i < count; i++) {
        bytes[i] = (Wire.available() > 0) ? Wire.read() : 0;
    }
}
```

```
// SPI Helpers
void help_SPI_write(uint8_t reg_addr, uint8_t value) {
    SPI.beginTransaction();
    digitalWrite(CHIP_SELECT, LOW);
    SPI.transfer(reg_addr);
    SPI.transfer(value);
    digitalWrite(CHIP_SELECT, HIGH);
    SPI.endTransaction();
}

uint8_t help_SPI_read(uint8_t reg_addr) {
    SPI.beginTransaction();
    digitalWrite(CHIP_SELECT, LOW);
    SPI.transfer(reg_addr);
    uint8_t value = SPI.transfer(0x00);
    digitalWrite(CHIP_SELECT, HIGH);
    SPI.endTransaction();
}

// Generic helpers.
float help_calc_speed(int timeFirst, int timeSecond) {
    return (WHEEL_CIRCUMFERENCE * MS_PER_SECOND) / (timeSecond - timeFirst)
}

float help_calc_angle(int16_t value) {
    // straight up is -512/2 (+-2g scaling) straight down
    // is 511/2 (+-2g scaling)
    // 10 bit ADC
    return map(value, -512/2, 511/2, 90, -90);
}
```

Part C: Setup [8 points]

Write a setup function that will initialize any inputs, outputs, and sensors you may need. Define any variables that you may need.

```
unsigned long timeFirstSpeed = 0;
unsigned long timeSecondSpeed = 0;

#define ACCEL_WRITE_ADDR 0x3A
#define ACCEL_READ_ADDR 0x3B
#define DATA_X0_REG 0x32
#define DATA_X1_REG 0x33

#define BW_RATE_REG 0x2C
#define BW_RATE_CONFIG 0x0A

#define POWER_CTL_REG 0x2D
#define POWER_CTRL_CONFIG 0x08

#define DATA_FORMAT_REG 0x31
#define DATA_FORMAT_CONFIG 0x04 //spi mode, full scale
#define ANGLE_TASK_FREQ 100
#define SPEED_TASK_FREQ 1000

// create semaphores and mutexes for protecting serial port and global data
xSerialMutex = xSemaphoneCreateMutex();
void setup() {

// setup uart for serial device
Serial.begin(115200);

// setup interrupt for wheel encoder
pinMode(2, INPUT);
attachInterrupt(2, wheel_isr, RISING);

if ( (xSerialSemaphore) != NULL )
    xSemaphoreGive( xSerialMutex ); //serial port available

// accelerometer Config  I2C
help_I2C_Write(ACCEL_WRITE_ADDR, BW_RATE_REG, BW_RATE_CONFIG)
help_I2C_Write(ACCEL_WRITE_ADDR, POWER_CTRL_REG, POWER_CTRL_CONFIG)
help_I2C_Write(ACCEL_WRITE_ADDR, DATA_FORMAT_REG, DATA_FORMAT_CONFIG)

// accelerometer Config  SPI
help_SPI_write(BW_RATE_REG, BW_RATE_CONFIG)
help_SPI_write(POWER_CTRL_REG, POWER_CTRL_CONFIG)
help_SPI_write(DATA_FORMAT_REG, DATA_FORMAT_CONFIG);

xTaskCreate(task1, “Angle”, 200, NULL, 2, NULL);
xTaskCreate(task2, “Bike_Speed, 200, NULL, 1, NULL);

vTaskStartScheduler();
}
```

Part D: Interrupt [5 points]

Write the interrupt function you need here.

```
void wheel_isr {
    timeSecondSpeed = timeFirstSpeed;
    timeFirstSpeed = millis();
}
```

Part E: Speed [5 points]
Write the task which logs the speed of the bike.

```
// TASK 2

void task2(void *pParam)  {
    // Angle Task
     const portTickType xFrequency = SPEED_TASK_FREQ;
     xLastWakeTime = xTaskGetTickCount();

    for(;;) {
        vTaskDelayUntil(&xLastWakeTime, xFrequency);

        float speed = help_calc_speed(timeFirstSpeed,
                              timeSecondSpeed);
        unsigned long time = millis();
        if(xSemaphoreTake(xSerialMutex, portMAX_DELAY) {
            Serial.print("T: %ul S: %f\n", time, speed);
        }
    }
```

Part E: Angle [5 points]
Write the task which logs the angle of the bike.

```
void task1(void *pParam)  {
    // Angle Task
     const portTickType xFrequency = ANGLE_TASK_FREQ;
     xLastWakeTime = xTaskGetTickCount();

    for(;;) {
        vTaskDelayUntil(&xLastWakeTime, xFrequency);
        int16_t angle_data = 0;

        help_I2C_Read(ACCEL_READ_ADDR, DATA_X0_REG, angle_data, 2);

        or

        help_SPI_read(DATA_X0_REG, angle_data);
        help_SPI_read(DATA_X1_REG, temp);
        angle_data = (temp << 8) || angle_data;

        float angle = help_calc_angle(angle_data);
        if(xSemaphoreTake(xSerialMutex, portMAX_DELAY) {
            Serial.print("T: %ul A: %f\n", time, angle);
        }
    }

}
```