1. (10 pts.) Design a shaft encoder capable of detecting direction and distance of rotation with a resolution of at least 22.5 degrees, and also capable of determining absolute orientation with an error of at most 90 degrees. Subject to these requirements, minimize the number of light sensors and emitters; having more increases failure rate and maintenance time.

   No need to specify every detail. However, you must provide the following information.

   a. Absolute orientation
      i. Number of tracks

      2

      ii. Code sequence cycle

      00, 01, 11, 10

      iii. Number of times cycle repeats

      1 (doesn't repeat)

   b. Relative rotation
      i. Number of tracks

      2

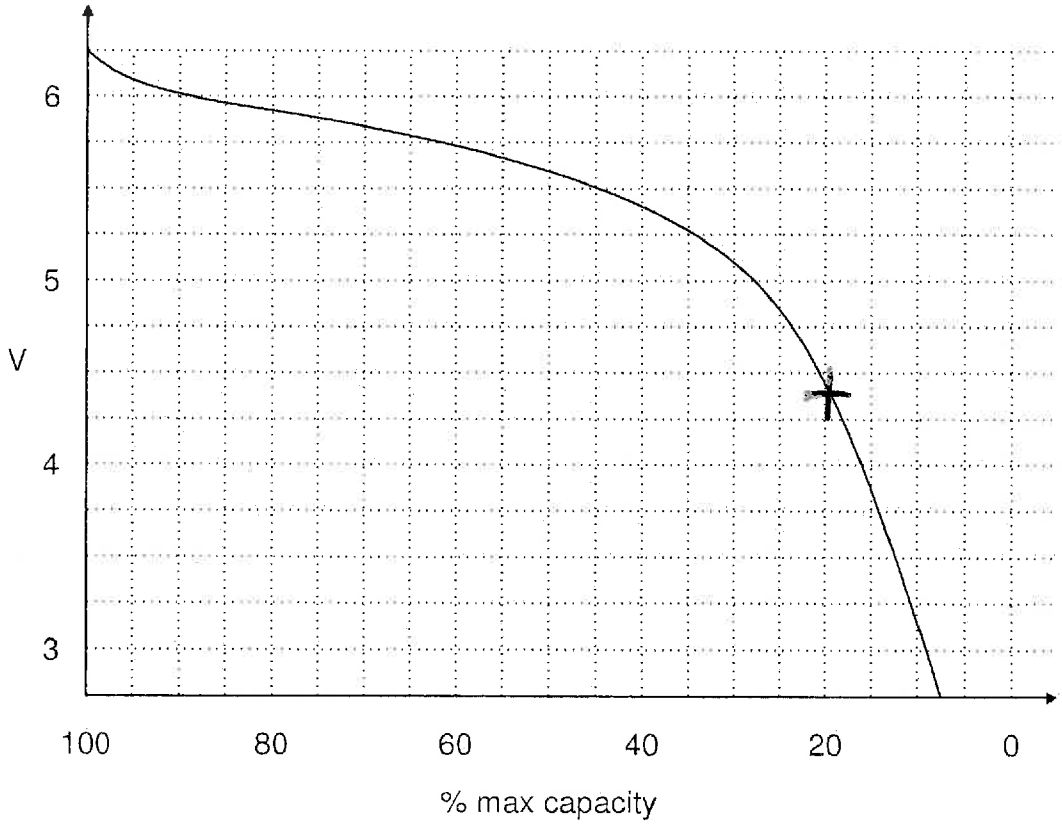
      ii. Code sequence cycle

      00, 01, 11, 10

      iii. Number of times cycle repeats

      4

2. (17 pts) Estimate the battery lifespan for the following embedded system, which is powered from a 2A-h battery pack having the following discharge curve.



The embedded system nominally operates at 3.3V. The components of the embedded system have the following characteristics:

CPU

- Operating voltage range: 3.2V-3.4V
- Average current
  - Low-power mode: 3mA
  - Normal mode: 300mA
  - Sprint mode: 600mA

Wireless interface

- Operating voltage range: 3.1V-3.4V
- Average current
  - Low-power mode: 5mA
  - Active mode: 500mA

Other information

- The system uses a Buck converter requiring that its input voltage be 1.2V higher than its output voltage.
- The CPU spends 80% of its time in low-power mode, 15% of its time in normal mode, and 5% of its time in sprint mode. It has a hardware timer that remains operational when the CPU is in low-power mode, and the timer can be used to enact a CPU power management state change.
- The wireless interface needs to enter active mode for 1ms per hour. However, it contains internal logic forcing it to remain in active mode for 10 minutes before returning to low-power mode.

Show your work.

a. CPU average power consumption.

$3.3V(0.8 \cdot 3mA + 0.15 \cdot 300mA + 0.05 \cdot 600mA) =$

$\boxed{255mW}$

b. Wireless interface average power consumption.

$3.3V(\frac{10min}{60min} \cdot 500mA + \frac{50min}{60min} \cdot 5mA) =$

$\boxed{289mW}$

c. System average power consumption.

$\boxed{544mW}$

d. System lifespan starting from full battery charge.

Integrating the battery curve would be even better!

Fails at $\max(3.2V, 3.1V) + 1.2V = 4.4V$

80% depletion from graph. $0.8 \cdot 2Ah \cdot (6.25V + 4.4V)/2 = 8.52W\text{-}h$

$8.52W\text{-}h / 544mW = \boxed{15.7h}$

Considering regulator inefficiency is fine, but I didn't ask for it.

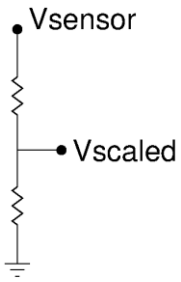
3. In a 32 bit machine, there's a 64 bit hardware timer. Now you need to implement a function to capture the timestamp. The function prototype is given below. The address of the higher 32 bits is 0x40004504, and the address of the lower 32 bits is 0x40004500.

Hint:
* Each load can only load 32 bits data. Therefore, you need at least two loads to get 64 bits value.
* Think carefully about how you should arrange the order of the reads. In an extreme situation, the value you read out would be very far from the valid value because of overflow in the lower 32 bits. In this case, you should return 0 to inform the user the read is not successful.

Answer:

```
#include <stdint.h>

uint64_t Capture(void) {
        volatile uint32_t * high_addr = (uint32_t *)0x40004504;
        volatile uint32_t * low_addr = (uint32_t *)0x40004500;

        uint32_t h_val = *highAddr;
        uint32_t l_val = *lowAddr;
        uint32_t h_new = *highAddr;

        // if the high part before reading low 32 bits is different from the high part
        // after reading low 32 bits, that means an overflow in the lower 32 bits has
        // occurred. In this case, we cannot tell if the lower 32 bits we got are valid
        // or not. So we need to return failure in this case.

        if(h_val != h_new) return 0;

        uint64_t result = ((uint64_t)h_val << 32) | l_val;

        return result;
}
```

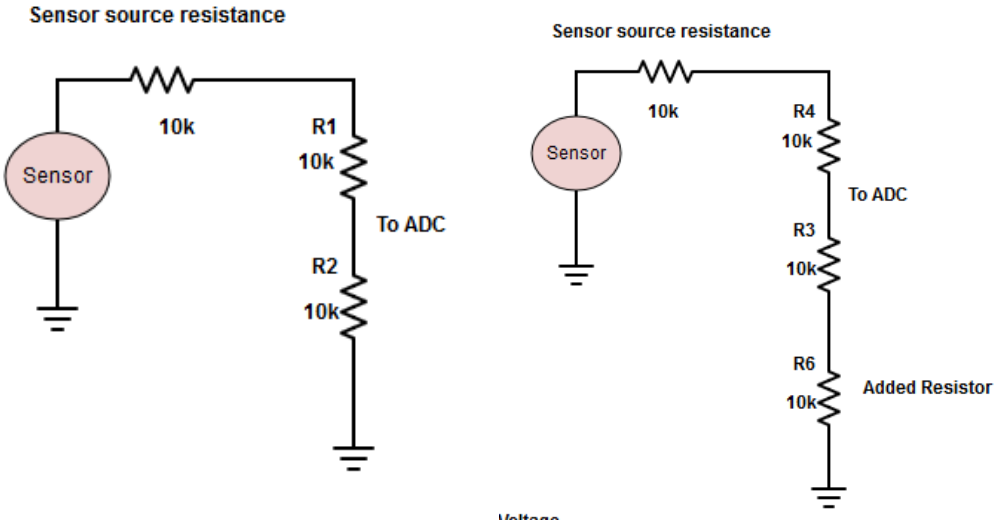
Q4 (10 pts.) Assume you are working with a sensor that operates between 0V and 5V. Your ADC only works between 0V and 2.5V. Construct a simple voltage divider to divide the sensor voltage by 2 with 10K resistors. When you hook up the circuit you discover that the sensor voltage is divided by 3. Yes, 3. Assuming you only have 10K resistors to work with, fix the divider by adding series resistance or parallel resistance so that will produce a 0 to 2.5V output range. It is fine to neglect capacitive and inductive parasitic effects.



There are two possible reasons why the voltage divider is producing 1/3 of the expected value rather than ½ has predicted by the voltage divider. One is the source resistance of the sensor is 10K as shown with the input resistance of the ADC very large. This will produce a voltage division of

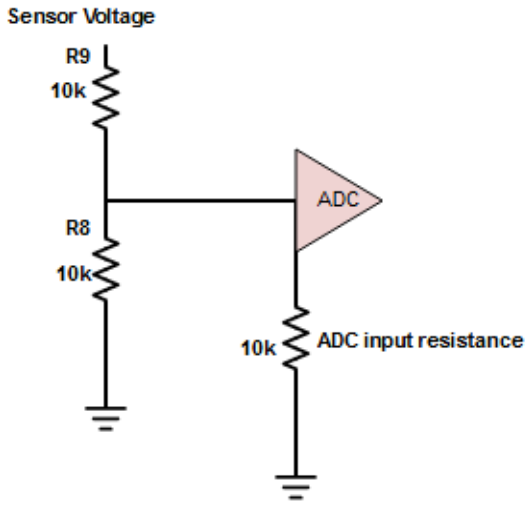
Sensor voltage * R2/(Sensor source resistance * R1 * R2) = Sensor voltage * 1/3

To fix it simply add a resistor in series with R2 to balance the divider.

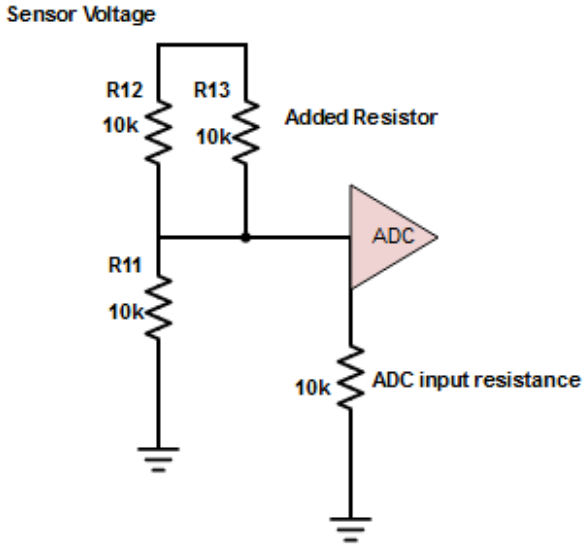


The second possibility is the input of the ADC resistance is 10K and the sensor source resistance is very low. Then the voltage provided to the ADC is

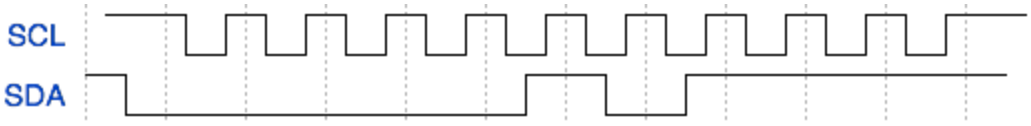
Sensor Voltage * (ADC input resistance || R8)/(R9 + R8||ADC input resistance) = Sensor Voltage * 1/3



To fix it, put a 10k in parallel with R9 to balance the divider.

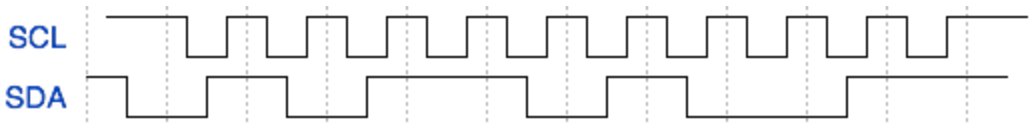


5. (15 pts.) See reference at end of exam.
   a. You have written a library to have the microcontroller interface over I2C with the RGB LED device at address 0x4D and an accelerometer at address 0x05. Reading from the accelerometer at address 0x05 returns three bytes representing the acceleration in the X, Y, and Z axes. However, when you attempt to do a read from the accelerometer's X axis register, you receive no response. The logic analyzer trace is below. What is the problem?



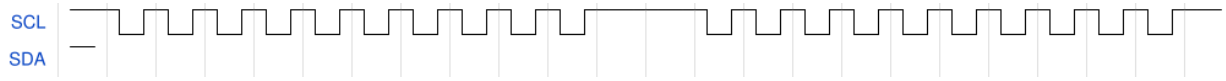
The issue is that the accelerometer is not returning an ACK in response to the micrcontroller’s read request.

b. Now that you can read from the accelerometer's X axis register, you have moved on to writing to the RGB LED. To write to the RGB LED, you send three bytes: one representing the amount of red desired, one representing the amount of green, and one representing the amount of blue (all from 0 to 255). If only one byte is sent, then only the red component of the RGB LED's color is updated. You attempt a write to the RGB LED. However, the RGB LED does not change color. A copy of the first frame of the logic analyzer reading is below. What is the problem?

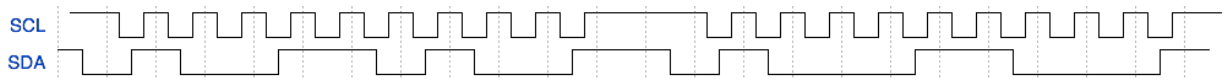


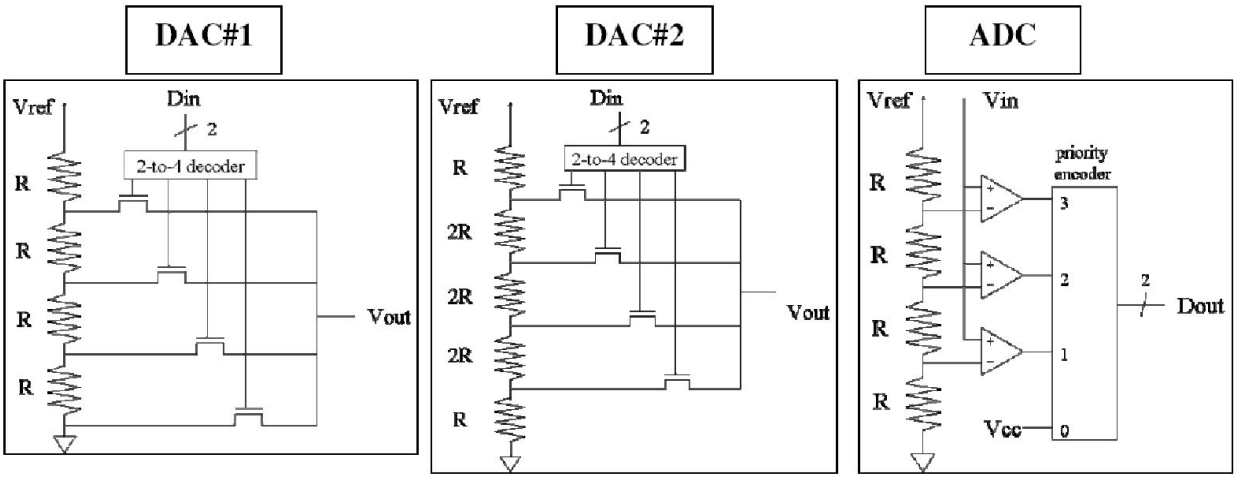
The primary issue is that the microcontroller is not sending the correct address as part of the write request (0x5A is being sent instead of 0x4D). A secondary issue is that the LED is not returning an ACK in response to the microcontroller (likely due to the incorrect address).

c. Please draw the proper timing diagram to write a red component value of 0x8C to the RGB LED.



Answer:





6. (17 pts.) The following questions are related to ADC and DAC conversion. Refer to the figures above. Vref=16V. The ADC and DAC output values have INL error up to ± ¼ the least significant bit. In DACs, the INL is the maximum output error. In ADCs, the INL is the maximum input threshold error. This error could make it possible for multiple outputs to be produced, given a particular input. Report all possible answers, or a range, as appropriate. The voltage comparators return true if the comparison voltage is greater than or equal to the reference voltage.
    a. If 12 V is put on the ADC input and D_{out} is connected to D_{in} of of DAC#1, what are the values that might possibly be present on V_{out} of the DAC?

    D_{out} is either 10 or ~~01~~ 11

    DAC#1 therefore outputs either 7v-9v or 11v-13v

    b. If "01" is put on the DAC#2 input and Vout is connected to the ADC's Vin, what are the values that might possibly be present on the ADC's Dout?

    V_{out} outputs 5v-7v

    ADC has cutoff ranges of 7v-9v and 3v-5v, so the output could be ~~01 or 10~~

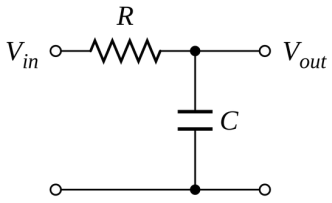
    00, 01, or 10, although it will probably be 01 in most cases.

c. One application of ADCs is in the sampling and manipulation of audio signals. In order to properly sample audio signals and avoid aliasing, a filter must be applied before the signal is sampled and converted. Use the fact that humans can process audio signals up to 20kHz to answer the following two questions about the design of an audio sampling ADC. State the minimum sampling rate that the ADC must sample at in order to avoid perceptible information loss.

Shannon-Nyquist rate is 2*max frequency. Therefore min sampling at 40kHz

d. Given a capacitor value of C = 0.1uF, choose an R value to complete a low pass filter with a cutoff of 3dB, i.e., $\sqrt{1/2}$ the passband voltage, at the proper frequency. Use the following formula to do your calculations:

$$|V_{out}| = |V_{in}| \cdot \frac{1}{\sqrt{1 + w^2 \cdot R^2 \cdot C^2}} \quad \text{where} \quad w = 2\pi f$$



$$\frac{1}{\sqrt{2}} = \frac{1}{\sqrt{1+w^2*R^2*C^2}} where\ w = 2 * \pi * 20,000$$

$$1 = w^2 * R^2 * C^2$$

$$R = \sqrt{\frac{1}{w^2*C^2}} = 79.57\Omega$$

7. SOLUTION:

a) 0xE000E108
b) Bit 9
c) *(SETEN + i/32) = 1 << (i % 32);
d) Prevent preemption, turning interrupts off, change priorities, make isr atomic