EECS 473 Midterm Exam Answer Key

Fall 2023

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 **14** pages total for the exam as well as handouts which you will need for the last question.
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. Though the last question is worth significantly less than half the points, we expect it will take at least half of the exam time.

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.

1) **<u>Circle</u> the letter in front of all the true statements.**
[9 points, -2 per wrong circle/lack of a circle, minimum 0]

a) A primary purpose of the silkscreen layer on a PCB is to prevent solder from adhering to parts of the PCB when components are mounted on it.

b) It is very unlikely that a fully-charged 10Ah lead-acid battery will run out of energy before 100 hours have passed when drained at a rate of 50mA.

c) A linear regulator is generally expected to have a less-noisy output than a switching regulator.

d) Software licensed under the GNU Public License are open source and, with minor restrictions, effectively in the public domain.

e) On a PCB, a 30 mil-wide trace has more resistance than a 10mil-wide trace of half the length.

f) When placing multiple capacitors across the same power and ground pins of a given device, you want to put the capacitors as close to the pins as possible and put the smaller-value capacitors closer to those pins than the larger-value ones.

g) In general, Alkaline batteries are better suited for a task than LiPo batteries when the batteries are likely to sit unused for months at a time.

h) One advantage of Rate Monotonic scheduling over Earliest Deadline First scheduling is that RMS doesn't require dynamic priorities.

i) When discussing PCBs, the term "rat's nest" is best understood to a board design that has more than two layers.

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

2) **Short answer/multiple choice/matching** [20]

a) You have an LDO which a coworker says is wasting exactly half the input power as heat. If the input voltage is 8V, the output voltage is 4.5V, and the LDO's input current is 10mA, what is the LDO's quiescent current? Clearly show your work. **[4]**

P=IV Input Power = 8V * 10mA = 80mW Wasted Power = ½ Input Power = 40mW
Current through load = 40mW (used power) / 4.5V = 8.89mA through load
Power Wasted = Voltage Drop + Queiscent => (8V – 4.5V)(8.89mA) + 8V(QmA) = 40mW
Solving for QmA, quiescent current = 1.11mA

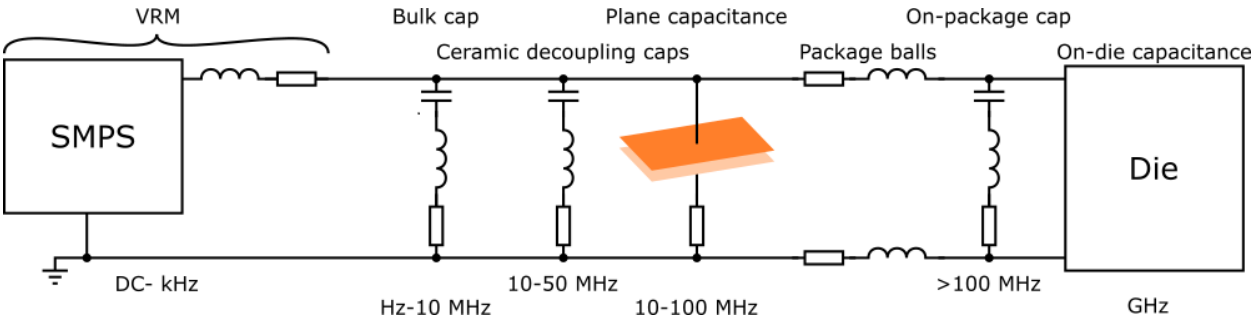
b) Which of the following is a technique we might use on a real-time systems that we would almost never use in a desktop computer? The technique and its reason for use must both be correct. Circle the best answer. **[3]**

- Running code with volatile variables disabled to enable more consistent run times.
- Running code with optimization turned off to enable memory-mapped I/O.
- Running code with the cache turned off to enable memory-mapped I/O.
- Running code with a cache turned off to enable more consistent run times.
- Running code with memory-mapped I/O turned off to enable file-based I/O.

c) Match the following PCB terms to their definitions by writing the appropriate letter (the same definition may be used more than once). **[5, -1 per blank or wrong answer]**

__C__	via
__B__	trace
__D__	mil
__D__	thou
__G__	copper weight
__F__	power integrity
__I__	silk screen

A. technical drawing that illustrates the connections between PCB components.
B. the copper path printed on a PCB
C. a plated through-hole that connect signals on different layers.
D. a thousandth of an inch
E. a millimeter
F. keeping the ground/power differential at a desired value
G. used to describe the height of copper on the PCB
H. about 14g/cm^3
I. the layer of the board which includes the drawing and writing of text
J. none of the above



d) In the context of the above figure, which one of the following is <u>false</u>? **[3]**

- 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 ceramic decoupling capacitors generally have lower parasitic values than the bulk capacitors.
- The PCB plane generally provides more capacitance than any single bulk capacitor.

e) If you have two tasks that have a total CPU utilization of 40%, will rate monotonic scheduling <u>*without preemption*</u> always successfully schedule those tasks? If so, explain why. If not, provide an example that fails. **[5]**

No
Example:

	Execution Time	Period
Task 1	1	10
Task 2	30	100

Cpu utilization = 1/10 + 30/100 = 0.4 = 40%
Once Task 2 starts running, it will run for its full duration, causing Task 1 to miss it's deadline.

3) **Scheduling** [11 points]

Say you have the following groups of tasks. For each group find the CPU utilization and identify which groups are RM and which are EDF schedulable. Indicate if you needed to do the critical instant analysis. *If needed, **clearly** show that analysis.* The following equation may prove useful.

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

Group	T1 Execution Time	T1 Period	T2 Execution Time	T2 Period	T3 Execution Time	T3 Period	% Utilization
A	1	3	1	4	3	6	108%
B	1	4	2	9	5	10	97%
C	4	7	2	9	--	--	79%
D	1	3	2	5	2	9	96%

Group	EDF Schedulable?	RM Schedulable?	Did you need to examine the critical instance?
A	No	No	No
B	Yes	No	Yes
C	Yes	Yes	No
D	Yes	Yes	Yes

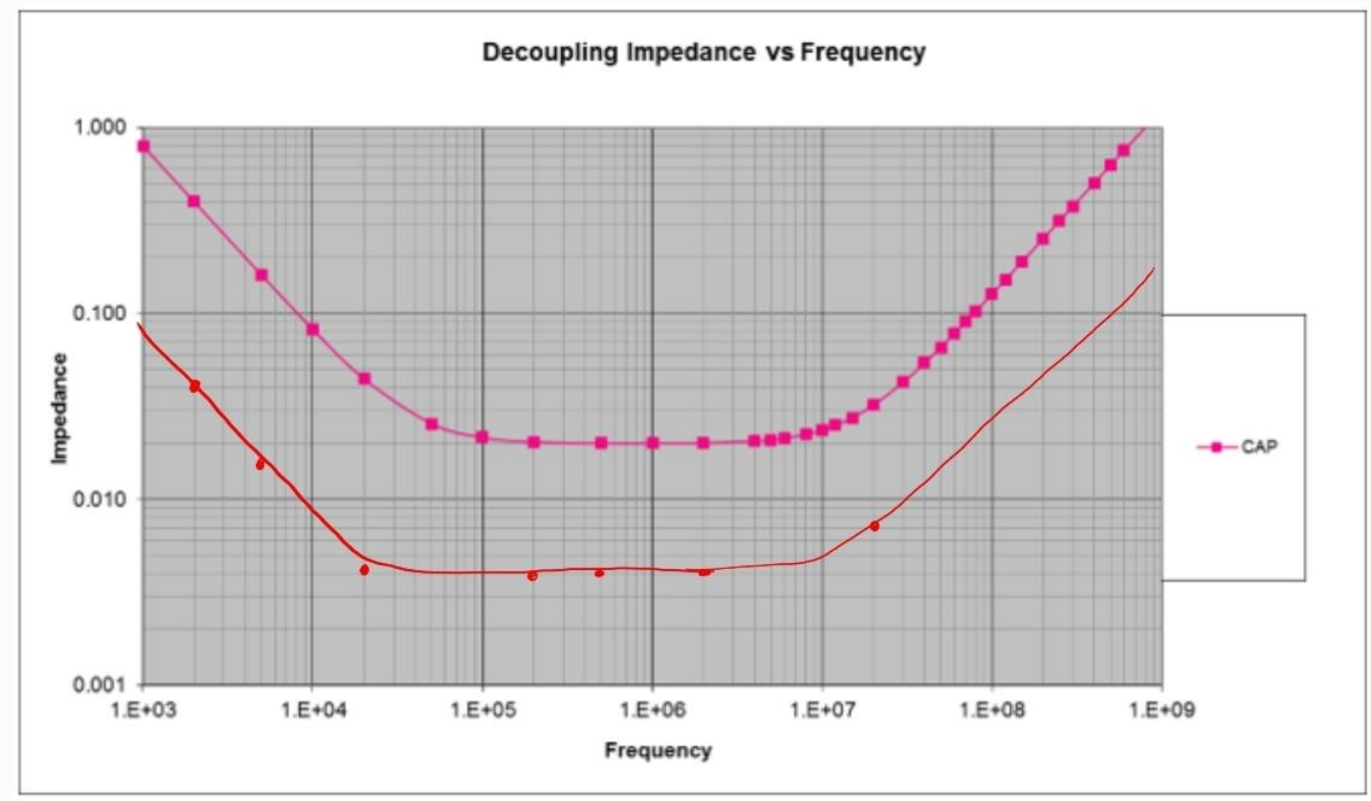
Group B	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11
T3										Failed	
T2											
T1											

Group D	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11
T3											
T2											
T1											

All tasks in Group D finish on time.

4) **Decoupling capacitors** [5 points]

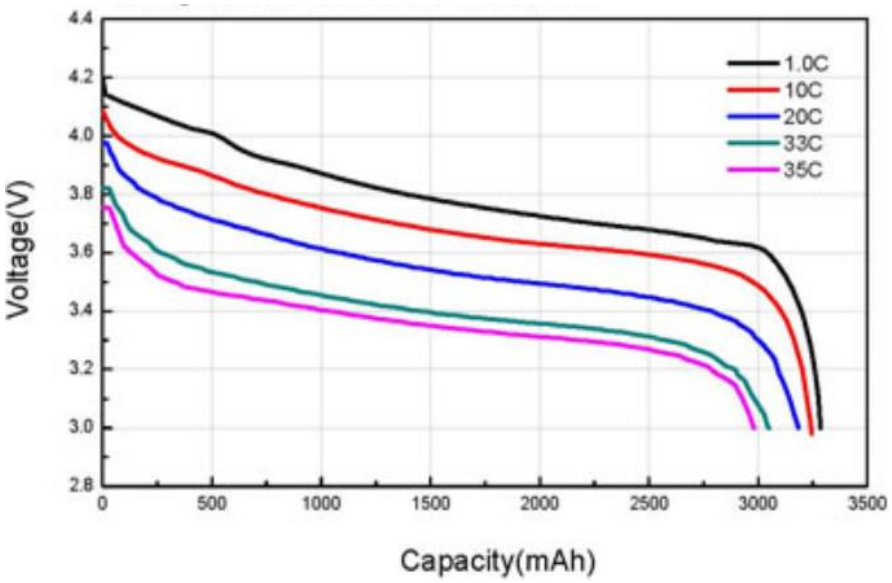
The graph below shows the frequency vs. impedance for a given capacitor. Redraw the graph showing the same information but if instead we used 5 new capacitors (in parallel), *each* of which had twice the capacitance of the original, while each new capacitor has the same resistance and inductance as the original.



2x5 = 10x more capacitance and x5 less ESR and ESL. Dots are to help drawing.

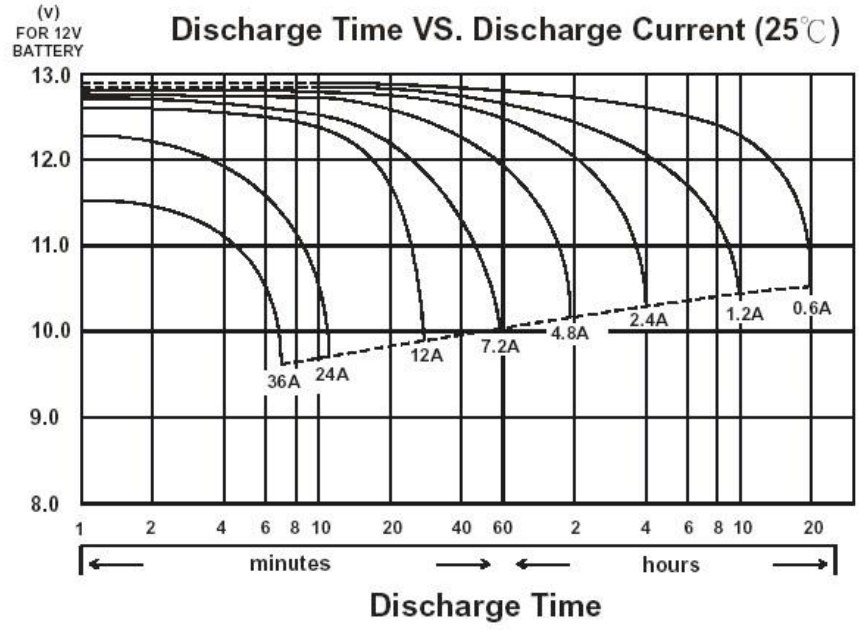
5) **Batteries** [10 points]

Consider the following battery discharge curves from real battery specifications.



a) *About* how long would you expect the battery on the left to be able to drive 24 Amps while maintaining at least 3.6 V output? Briefly justify your answer. **[5]**

About 3300mA battery. 24A is ~7C. Assume 10C curve. 3.6V intersects at ~2500mAh. 2.5Ah / 24A = about 6.25 min. 5-8min is reasonable.



b) *About* how long could battery of the chemistry to the left drive a 5Ω load that requires at least 11 Volts? Briefly justify your answer. **[5]**

13V / 5Ω =2.6A
11V / 5Ω = 2.2A
Using 2.4A line, about 3.8 hours. That's probably pretty close, but 3.5 to 4.1 or so is pretty reasonable

6) **Linux device drivers** [8 points]

Consider the following code found as the read function member of the file_operations struct for a Linux kernel module. It is associated with the device file "/dev/txx2" (so a read of the file /dev/txx2 will result in this function being called). Assume that everything is set up appropriately beforehand. Ignore the fact that copy_to_user's return value is being ignored (it's just a warning…).

```
const char s[] = "0123456789ABCDEFGHIJ";
ssize_t memory_read(struct file *filp, char *buf,
                    size_t count, loff_t *f_pos) {

     printk("<1> fpos= %d\n",*f_pos);

     /* Transferring data to user space */
     copy_to_user (buf, s+*f_pos, 4);

     /* Changing reading position as best suits */

     if(*f_pos>=8)
          return 0;

     *f_pos+=3;
     return 2;

}
```

Say that someone does a cat of /dev/txx2.

a) What will appear in the log file? **[3]**

<1> fpos= 0
<1> fpos= 3
<1> fpos= 6
<1> fpos= 9

b) What will be printed by the cat command? **[5]**

013467

7) **Design Problem—Deep depths and deeper thoughts** [37 Points]

Please read the entire problem BEFORE starting

Underwater diving presents many dangers due to the inhospitable environment. Modern SCUBA diving equipment facilitates exploration but becomes dangerous with depth. As a result, DiveSafe is developing a device for divers to wear that alerts the dive boat if a diver exceeds safe depth limits. You will be developing a prototype of the warning device for DiveSafe. You will be using the following components to build the system:

- 1 Arduino UNO
- 1 MS5837-30BA Water Pressure Sensor
- 1 bi-directional level shifter.
    - Connect the lower voltage source to LV, the higher voltage source to HV. The device connects LV1 to HV1, LV2 to HV2, etc.
- 1 7.2V Lithium Ion Battery
- 1 5V Buzzer Alarm -- providing power causes it to buzz (i.e. no PWM needed)
- 1 LED
- 1 3.3 V regulator
- Resistors/Capacitors/Inductors as needed

The system should have the following characteristics:

- You must use an external 3.3V regulator to power the MS5837-30BA sensor
- The Arduino should monitor the depth via the water pressure sensor.
    - Activate caution LED for diver at depths exceeding **30 meters**
    - Activate warning alarm for boat at depths exceeding **40 meters**.
- Your buzzer may be connected to a GPIO pin, provided a 100Ω current-limiting resistor is used.
- Your LED requires a 180Ω current-limiting resistor.
- Please use the maximum oversampling resolution option (**OSR=4096**) option when reading BOTH pressure and temperature.
- The function to apply temperature compensation is written **FOR YOU**, see Part C. You do not need to implement those functions.
- You will need to calculate depth based on water pressure. A table relating them has been provided to help you with the calculation.
- For the Arduino UNO, use the following pin mapping for your components: **A4 – SDA, A5 – SCK, 2 – Buzzer GPIO, 3 – LED GPIO**

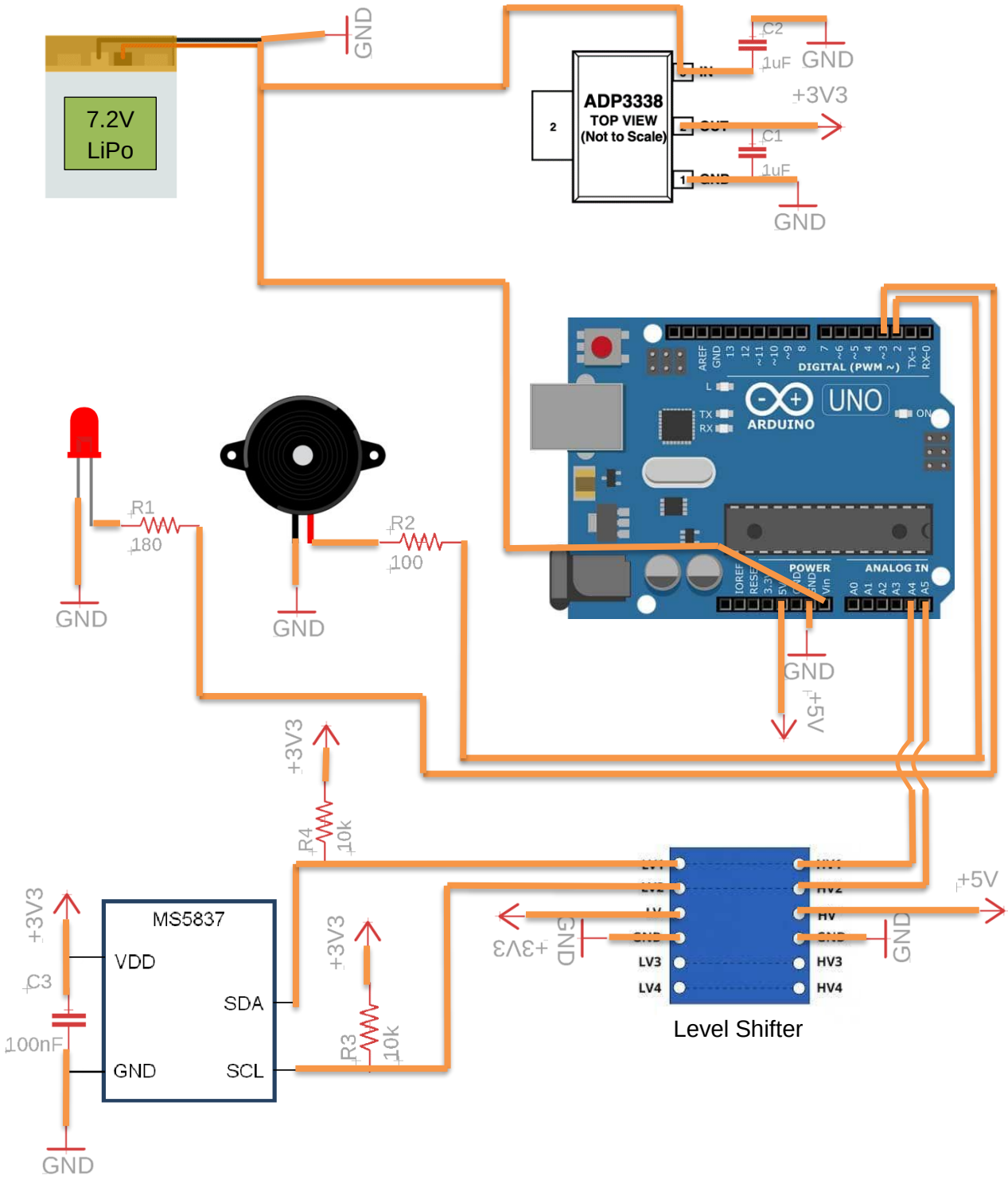
Water Pressure vs. Depth Table:

Depth (meters):	Pressure (millibars)
0	1013
5	1519
10	2026
20	3039
30	4052
40	5065
50	6078

- Water pressure increases predictably with depth due to the weight of the water above. This table will help you implement the code converting pressure to depth.

Part A: Wiring [8 Points]

Please complete the circuit here. For 3.3V, 5V and GND you are **highly encouraged to use labels** to keep the diagram neat. Draw passives (resistors, capacitors, inductors) where needed. Be sure to include values for all passive components you use.



Part B: Interface [6 Points]

Please implement the following I2C read and write functions:

```
uint32_t sensorReadI2C(uint8_t cmd, uint8_t resp_bytes)
// resp_bytes is the number of bytes we want to read
{
      uint32_t response = 0;

      Wire.beginTransmission(0x76); // set device address
      Wire.write(cmd);
      Wire.endTransmission();

      // Read specified number of bytes from device
      Wire.requestFrom(0x76, 3);
      while(Wire.available() < resp_bytes);
      while(resp_byte-- > 0) {
            response = (response << 8) | Wire.read();
      }

      return response;


}
```

```
void sensorWriteI2C(uint8_t cmd)
{
      // write command, none require data
      Wire.beginTransmission(0x76); // set device address
      Wire.write(cmd);
      Wire.endTransmission();




}
```

Part C: Short Answer [6 points]
The following helper function prototypes available for you to use. You **DO NOT** need to implement them. Please review and answer the following questions.

```
/* This function reads ALL calibration values and stores them in a
global struct for use by the pressure to depth function. Call ONCE
before reading any data from the device. */

void get_calibration_vals(void);

/* This function computes the pressure in millibars*/
// Params: RAW pressure, RAW temperature (Note: OSR=4096)
// Returns: pressure in millibars

uint16_t compute_pressure_mbar(uint32_t pres, uint32_t temp);
```

1. Why does `compute_pressure_mbar()` need the temperature? Please refer to the datasheet describing how the sensor operates to help guide your answer.

The temperature is used to compensate for non-idealities in the pressure sensor. The device Is not perfect, so pressure readings are not independent of temperature.

2. What do D1 and D2 refer to in the datasheet? (Hint: these are **ONLY** defined in the "**Conversion Sequence**" section)

D1 refers to a pressure conversion and D2 refers to the temperature conversion.

3. Why do you need to call `get_calibration_vals()`?

Each unit produced has some variation in its readings. At the factory, calibration values are loaded into the device. Since these values differ by unit, they must be read from it.

4. Write the equation to convert pressure in millibars to depth in meters. Please use the appended table to help you.

depth = (pressure / 101.3) - 10

Part D: Setup [5 points]

Please implement the setup function for this device. Define any variables or constants when appropriate to maximize readability.

```
#define BUZZER 2
#define STATUS_LED 3

#define RESET_CMD 0x1E


void setup()
{
      // Initialize I2C for pressure sensor
      Wire.begin();

      sensorWriteI2C(RESET_CMD);
      get_calibration_vals();


      // Initialize GPIO for output devices
      pinMode(BUZZER, OUTPUT); // Buzzer pin
      pinMode(STATUS_LED, OUTPUT); // LED pin
      digitalWrite(BUZZER, 0); // Buzzer pin
      digitalWrite(STATUS_LED, 0); // LED pin

}
```

Part E: Main Loop [12 Points]

Please implement the loop function for the system. This function should read the pressure and sound the buzzer alarm if depth exceeds 40 meters. Use the calibration function that **GIVEN TO YOU** in part C to process the raw readings.

```
# define PRES_CONV 0x48
# define TEMP_CONV 0x58
# define ADC_RES 0x00

# define DEPTH_CAUT 30
# define DEPTH_WARN 40

void loop()
{
      uint32_t raw_pres, raw_temp;
      uint32_t corrected_pres;
      uint16_t depth;

      // Measure pressure
      sensorWriteI2C(PRES_CONV);
      raw_pres = sensorReadI2C(ADC_RES, 3);

      // Measure temperature
      sensorWriteI2C(TEMP_CONV);
      raw_temp = sensorReadI2C(ADC_RES, 3);


      // Convert pressure reading to depth
      corrected_pres = compute_pressure_mbar(raw_pres, raw_temp);
      depth = (corrected_pressure * 10 / 1013) - 10;


      // Take appropriate actions
      digitalWrite(STATUS_LED, (depth > DEPTH_CAUT));
      digitalWrite(BUZZER, (depth > DEPTH_WARN));




}
```