EECS 473 Homework 1 Fall 2026 Answer Key

Show your work where requested. Keep written explanations brief. Use course materials and references as needed; no hardware implementation is required. Do your own work without AI tools. Also practice an old midterm design problem when studying. Do not expect to get this back before the exam. I expect this will take about 6 hours to do.

1. **True/False** (10 points)

Circle all TRUE statements. Start with 10 points; subtract 2 for each statement incorrectly circled or left uncircled, with a minimum of 0.

a. An LDO is a type of linear regulator.

True. An LDO (low-dropout regulator) is a type of linear regulator.

b. Raising a linear regulator's input voltage reduces its heat, with output voltage and load current fixed.

False. It would increase heat dissipation.

c. A 6 cm long, 20 thou wide trace has the same resistance as a 3 cm long, 10 thou wide trace of the same thickness and temperature.

True. Resistance is proportional to length divided by width for the same material and thickness. Both traces have the same length-to-width ratio: 6/20 = 3/10.

d. Gerber files are one of the outputs of the PCB CAD tool process.

True. Gerber files describe PCB fabrication layers and are a common output of PCB CAD tools.

e. If a set of tasks can be successfully scheduled by RM scheduling, it can also be scheduled by using EDF.

True. EDF can successfully schedule in all the situations RM can, but the reverse is not always true.

f. If you redistribute a binary licensed under the GPL to someone, you must make the source code available to them on request.

True. Recipients of a GPL-covered binary must have access to the corresponding source under the GPL. Providing the source with the binary or using an appropriate source offer can satisfy that obligation.

g. Software licensed under the GPL is in the public domain.

False. GPL-covered software is copyrighted and licensed; it is not in the public domain.

h. Capacitors resist a change in current.

False. Capacitors resist a change in voltage. Inductors resist a change in current.

2. **Scheduling algorithms** (18 points)

For Questions 2–4, assume one CPU, independent periodic tasks, deadlines equal to periods, and no scheduling overhead. RM and EDF are preemptive. Below, C is worst-case execution time and T is period, both in ms.

Group	T1 C / T	T2 C / T	T3 C / T
A	1 / 5	2 / 8	3 / 12
B	2 / 5	2 / 7	3 / 12
C	2 / 5	3 / 7	3 / 10
D	2 / 8	2 / 8	3 / 8

For each group, find CPU utilization and determine whether RM and EDF can schedule it. Use the RM utilization bound when it settles the question. Where necessary, show a critical-instant timeline sufficient to show if the group of tasks can be scheduled. In the case of equal priorities, pick one.

For three tasks, the RM utilization bound is about 0.780 (77.98%). At or below this bound, the set is guaranteed schedulable. Above it, a set may still schedule. EDF can schedule these task sets exactly when total utilization is no more than 100%.

Group	CPU utilization	RM schedulable?	EDF schedulable?
A	1/5 + 2/8 + 3/12 = 70.00%	Yes; below the RM bound	Yes
B	2/5 + 2/7 + 3/12 = 93.57%	No; T3 misses its deadline	Yes
C	2/5 + 3/7 + 3/10 = 112.86%	No; utilization exceeds 100%	No
D	2/8 + 2/8 + 3/8 = 87.50%	Yes; see below	Yes

Group B needs critical-instant analysis. Use priorities T1 > T2 > T3, with all first releases at time 0. The schedule below shows the first missed deadline; intervals are in ms.

Interval	0–2	2–4	4–5	5–7	7–9	9–10	10–12
Runs	T1	T2	T3	T1	T2	T3	T1

At time 12, the first T3 job has received only 2 of its required 3 ms, so it misses its deadline. T1 and T2 do meet their deadlines (their worst-case response times are 2 and 4 ms).

Group D: pick the fixed order T1, T2, T3 for the equal priorities. At time 0 all three release. T1 runs 0–2, T2 runs 2–4, and T3 runs 4–7; the CPU is idle 7–8. All finish before their common deadline at 8 ms, and the same pattern repeats each period. Any fixed order works.

3. **Choosing a processor** (10 points)

Each instruction takes the same time and MIPS scales directly with execution speed. Choose among 28, 32, 40, and 48 MIPS processors.

Task	Maximum instructions per job	Period
A	0.8 million	80 ms
B	1.8 million	150 ms
C	3.0 million	300 ms

a) Find the slowest processor that can schedule these tasks under EDF. **[5]**

For EDF, we just need CPU utilization to be no more than 100%. A needs 0.8/0.080 = 10 MIPS; B needs 1.8/0.150 = 12 MIPS; C needs 3.0/0.300 = 10 MIPS. Total: 32 MIPS. So the slowest listed processor that works is 32 MIPS (exactly 100% utilization).

b) Using only utilization tests, classify each processor under RM as guaranteed schedulable, impossible, or not settled by those tests. Justify each classification; no critical-instant analysis is required. **[5]**

Utilization at a processor speed of S MIPS is 32/S. For three tasks, the RM bound is about 77.98%.

Processor	Utilization	RM classification
28 MIPS	114.29%	Impossible; exceeds 100%
32 MIPS	100.00%	Not settled by these tests
40 MIPS	80.00%	Not settled by these tests
48 MIPS	66.67%	Guaranteed schedulable; below the RM bound

4. **Round Robin** (8 points)

For this problem, you are to assume an ideal scheduler: ready tasks share the CPU equally, the time quantum is negligibly small, and a task stops consuming CPU when its job finishes. All tasks first release at time 0.

a) Give two tasks, specifying execution times and periods, whose total utilization is below 75% but which miss a deadline under Round Robin. One task must have exactly 10% utilization. Show the missed deadline and explain whether EDF could schedule your pair. **[5]**

One example is A: C = 60 ms, T = 100 ms; B: C = 100 ms, T = 1,000 ms. B has exactly 10% utilization and the total is 60% + 10% = 70%.

When first launched, A gets 50% of the CPU and B gets the other 50%. Neither finishes by time 100: each has received only 50 ms. A needs 60 ms and therefore misses its first deadline at 100 ms. EDF can schedule the pair because total utilization is 70%, below 100%.

b) Give two tasks, specifying execution times and periods, whose total utilization is above 95% but which never miss a deadline under Round Robin. **[3]**

One example is A: C = 49 ms, T = 100 ms; B: C = 49 ms, T = 100 ms. Total utilization is 98%. Both get half the CPU and finish at 98 ms, before their common deadline at 100 ms. The CPU is idle for 2 ms, then this pattern repeats forever.

5. **Shared resources and priority inversion** (16 points)

The tasks below run on one CPU and you are to assume there is no scheduling overhead. As normal, each deadline equals its period. A and C lock mutex X before doing any work and hold it for their entire CPU execution time. B does not use X. A task waiting for X blocks rather than spins.

Task	Execution time	Period	Uses X
A	3 ms	10 ms	Yes
B	5 ms	21 ms	No
C	5 ms	33 ms	Yes

a. Ignoring X, can RM schedule these tasks? Can EDF? Briefly justify your answers. **[4]**

Yes for both. Total utilization is 3/10 + 5/21 + 5/33 = 0.68961, or about 69%. RM can schedule any three independent tasks whose total utilization is no more than about 77.98%; EDF can schedule them because utilization is no more than 100%.

b. C first releases at time 0; A and B first release at time 1 ms. Draw the RM schedule through A's first deadline with no priority inheritance. Identify the priority inversion and any missed deadline. **[6]**

RM priorities are A > B > C. At time 1, A tries to lock X but blocks because C holds it. B is ready and preempts C.

Interval (ms)	Runs	State of A / X
0–1	C	C holds X
1–6	B	A blocks; C still holds X
6–10	C	A still blocks until C releases X at 10
10–11	A	A holds X; receives 1 ms before its deadline

A's first deadline is at 1 + 10 = 11 ms. At that time it has run for only 1 of its required 3 ms, so it misses the deadline. The priority inversion is that high-priority A is waiting for low-priority C, while medium-priority B delays C and thereby delays A. B and C meet their first deadlines.

c. Repeat that interval with priority inheritance. Show the effective priorities while A is blocked and say whether A meets this deadline. **[4]**

With priority inheritance, C temporarily inherits A's priority while A is blocked on X. This prevents B from preempting C.

Interval (ms)	Runs	Effective priorities / state
0–1	C	Normal priorities: A > B > C
1–5	C	C has A's priority; A is blocked and B waits
5–8	A	C releases X and returns to its normal priority
8–11	B	A has completed; normal priorities again

C has 4 ms remaining when A blocks at time 1 and finishes at time 5. A then runs for 3 ms and finishes at time 8, so it meets its deadline at time 11.

d. Does priority inheritance eliminate all blocking of A? Explain what blocking remains. **[2]**

No. A still waits for C to finish its critical section and release X. Here that direct blocking lasts 4 ms. Priority inheritance prevents B from adding delay to that blocking; it does not make the resource immediately available.

6. **Licensing** (12 points)

Read the Wikipedia article on "Gratis versus libre". Answer briefly in your own words. You may wish to consult the GNU GPL FAQ and Creative Commons license descriptions. Assume ordinary GPL-covered software with no special linking exception.

a. Explain gratis versus libre. Can software be libre without being gratis? Give an example. **[3]**

Gratis means "with no cost"; libre refers to the freedom to run, study, modify, and redistribute the software. Software can be libre without being gratis. You are allowed to charge for software licensed under the GPL, so it is not necessarily gratis. For example, a consultant might be paid to add support for a new ISA to gcc and distribute the GPL-covered result.

b. A company modifies a GPL program for internal use only. It later starts shipping the modified binary in its product. Explain how its source-code obligations change once it starts shipping that product and whether charging for the product is allowed. **[4]**

For internal use only, the company can keep its modified version private; it does not have to publish the source. Once it distributes the modified binary in the product, it must provide recipients license-compliant access to the corresponding source, including its modifications, under the GPL. It must also preserve the required notices and license information.

Charging for the product is allowed. The GPL is about the freedom to copy, modify, run, and redistribute the software, not about requiring a zero price. Recipients keep those freedoms even when they paid for the product.

c. What is the practical difference between the GPL and LGPL when it comes to someone using a library in a proprietary application? **[2]**

The LGPL allows a library to be used by a proprietary application without requiring that application to be licensed under the GPL or LGPL, provided the LGPL's conditions are met (including the ability to replace or relink the library). Modifications to the library remain covered by the LGPL. Linking an ordinary GPL library into a distributed application generally requires the combined application to be GPL-compatible and its corresponding source to be available.

d. Describe CC BY-NC-SA, including what ShareAlike requires when an adapted work is shared. **[3]**

You can use and share the material for noncommercial purposes as long as you provide attribution to the creator(s), identify changes, and retain the license information. BY means attribution; NC means noncommercial; SA means ShareAlike. If you share an adapted work, you must license the adaptation under CC BY-NC-SA (or an officially compatible license). You do not have to distribute an adaptation merely because you made it.

7. **RTOS timing and C** (12 points)

a. Define the term "jitter" in the context of an RTOS. **[2]**

Jitter is variability in response (or release) time for a given operation. It is generally associated with delays in interrupt response time.

b. An interrupt-triggered operation takes exactly 0.8 ms of CPU time. Some task can disable interrupts for at most 4 ms at a time. Find the worst-case event-to-start latency, event-to-completion time, and the range of variation in completion time. Assume no other delays exist. **[3]**

Worst-case event-to-start latency: 4 ms. Worst-case event-to-completion time: 4 + 0.8 = 4.8 ms. Completion can occur from 0.8 to 4.8 ms after the event, so the range of variation (jitter) is 4 ms.

c. What is a critical section? Why can protecting a long section by disabling interrupts be a problem even if it prevents a race? **[2]**

A section of code where it could be problematic if some other task were accessing the same shared resource at the same time. We prevent those conflicting accesses while the section runs. Disabling interrupts for a long section can delay interrupt handling and other work, increase jitter, and cause missed deadlines even though it prevents the race.

d. What is a void * in C? Why do FreeRTOS tasks take a void* as an argument? **[3]**

A void * is a generic pointer to an object; in C it can be converted to and from other object-pointer types without an explicit cast. FreeRTOS uses it so a task can receive a pointer to an arbitrary data structure. For example, a task can receive a pointer to a structure containing its pin number and sampling period, then interpret the argument as that structure's pointer type.

e. You hear someone say: “The volatile keyword is often thought to have something to do with caches—that isn’t the case. It is directed to the compiler, not the hardware.” Explain what they mean. **[2]**

There seems to be a common belief that volatile tells the processor not to use the cache for a data access, requiring things to go “all the way to memory.” That is not the case. It tells the compiler to preserve all accesses to that volatile data rather than optimize them away. Cacheability is controlled by the hardware’s memory attributes and configuration.

8. Deferred interrupts and semaphores (12 points)

SemaphoreDeclare(BUSY) creates an empty (taken) binary semaphore; passing FREE creates an available one. SemaphoreTake blocks until available, then consumes it. SemaphoreGive makes it available; additional gives while available are ignored. Treat the `SemaphoreGive` in the ISR as the appropriate ISR-safe operation.

```
Init:
    semaphore a = SemaphoreDeclare(BUSY);
    int i = 0, j = 0;
    int x[6] = {0};

ISR:
    i++;
    SemaphoreGive(a);

Task1:
    while (1) {
        SemaphoreTake(a);
        if (i > 5) { i = 0; j++; }
        x[i] = j;
    }
```

a. The terms “top half” and “bottom half” (sometimes “upper-half” and “lower-half”) are often used when discussing deferred interrupts. Explain that terminology **[2]**

The “top half” is the ISR code. The “bottom half” is the task that the ISR causes to be run.

b. Explain deferred interrupt processing and identify the top and bottom halves in the above code. Give one advantage of deferring the work. **[4]**

The ISR does only the immediate work needed for the event and signals a task to do the remaining work. Here the top half is the ISR containing i++ and SemaphoreGive(a). The bottom half is Task1, which wakes after SemaphoreTake(a), handles the wraparound, and writes x[i].

It allows us to have code that responds to an event without that code necessarily having the highest priority. It also helps keep the ISR short so other interrupts are not delayed as much.

c. Why initialize **a** to BUSY? What could happen if it started FREE? **[2]**

It was initialized to BUSY because we did not want Task1 to do its thing until after the first interrupt had occurred. If it started FREE, Task1 could run one iteration before any interrupt. In this code that iteration only writes x[0] = 0, so it does not change the values. In a system where the ISR supplies data, the task might run before there is any valid data.

d. Assume Task1 finishes one loop iteration after every interrupt, before the next interrupt occurs. Find x, i, and j after 22 interrupts have occurred and the corresponding 22 Task1 iterations have completed. **[4]**

x = {3, 3, 3, 3, 3, 2}, i = 4, j = 3.

Every sixth interrupt makes i reach 6; Task1 then resets i to 0 and increments j. After 18 interrupts, i = 0 and j = 3. Interrupts 19–22 write 3 into x[1] through x[4]. x[5] still has the 2 written on interrupt 17.

9. Mutexes (8 points)

Briefly answer the following questions:

a. What is the difference between a binary semaphore and a mutex for the FreeRTOS implementation used in lab? In particular, fill in the following table with each cell having a bolded "answer" and a brief (20 words or less) explanation. Some cells have been done for you as an example. **[6]**

Feature	Binary Semaphore	Mutex
Primary Intent	**Signaling**. Task A (or an ISR) tells Task B that an event happened.	**Mutual exclusion.** Protects a shared resource so only one task uses it at a time.
Concept of Ownership	**No ownership.** A different task or an ISR may give a semaphore taken by another task.	**Strict Ownership**. Only the specific thread that locked it can unlock it.
ISR safe?	**Yes..** ISR operations must not block and must obey the port's interrupt-priority restrictions. So it can Give (though not Take)	**No.** A FreeRTOS mutex cannot be taken or given in an ISR. More generally, because of the strict ownership, the fact you cannot take means you also cannot give a Mutex in an ISR.
Priority Inheritance?	**No.** FreeRTOS binary semaphores do not include priority inheritance. More generally you don't need it because we are doing producer-consumer work and so the issue will not arise.	**Yes.** A blocked higher-priority task can temporarily raise the priority of the task holding the mutex.

b. What is the difference between a general (or counting) semaphore and a binary semaphore? **[2]**

A counting semaphore keeps a count of available resources or pending events, up to its configured maximum. Taking it decrements the count; giving it increments the count. A binary semaphore has only the states 0 and 1, so additional gives while it is already available are not accumulated.

10. Linux device drivers (16 points)

This function is the read member of a registered character driver for /dev/review. Assume a fresh open starts at position 0, all copies succeed, and the function is called only with a positive count. The unusual position update and return value are intentional.

```
const char s[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
ssize_t memory_read(struct file *filp, char __user *buf,
                    size_t count, loff_t *f_pos) {
    if (*f_pos >= 10)
        return 0;
    size_t x = count > 3 ? 3 : count;
    copy_to_user(buf, s + *f_pos + 1, x);
    printk(KERN_INFO "pos=%lld\n", (long long)*f_pos);
    *f_pos += x + 1;
    return x - 1;
}
```

For parts a and b, the caller repeatedly reads until read returns 0, printing exactly the number of bytes returned by each successful call.

a. The caller requests 1 byte each time. What is logged, what is printed, and why does the caller stop? **[3]**

The log contains one line: pos=0. Nothing is printed by the caller. With count = 1, x = 1; the driver copies B into the buffer and advances the position to 2, but returns x − 1 = 0. The caller treats that return as EOF and stops immediately, without printing the copied byte.

b. After a separate fresh open, the caller requests 4 bytes each time. What is logged and printed? **[3]**

The log contains pos=0, pos=4, and pos=8, each on its own line. The caller prints BCFGJK.

Each call has x = 3, advances the position by 4, and reports only 2 bytes. At position 0 it copies BCD and reports BC; at 4 it copies FGH and reports FG; at 8 it copies JKL and reports JK. The next call sees position 12 and returns 0 without logging.

c. Rewrite the function body to return the first 10 characters of **s** contiguously across reads. Respect count and return 0 only at EOF once the 10 characters have been read. You need not include the printk. **[6]**

One answer:

```
int x;

if (*f_pos >= 10)
    return 0;
x = count;
if (x > (10 - *f_pos))
    x = 10 - *f_pos);
copy_to_user(buf, s + *f_pos, x);
*f_pos += x;
return x;
```

The offset advances by exactly the number of bytes copied and returned. Each read is capped at the remaining portion of the first 10 characters, ABCDEFGHIJ. With positive count, a non-EOF call returns a positive number of bytes. Ideally you'd also check for errors from copy_to_user…

d. What does mknod /dev/review c 61 2 create? Explain the roles of the major and minor numbers in connecting a device node to a driver and its devices. **[4]**

It creates the character-device node /dev/review with major number 61 and minor number 2. The major identifies the registered driver/device-number range that handles operations on the node; the minor is an argument to that device and usually identifies a particular device or instance. Note: mknod creates the node (device file); it does not load the driver or create the physical hardware.

11. PCB design and diagnosing noise (12 points)

a. Briefly define PI, SI, EMI, and EMC. **[4]**

Power Integrity (PI): maintaining the desired voltage from power to ground at the relevant parts of the PCB.

Signal Integrity (SI): ensuring that the signal carried on wires/traces is not corrupted by trace effects, internal noise, reflections, or other distortion.

Electromagnetic Interference (EMI): electromagnetic energy that can cause issues for an electronic device. We don't want to generate this.

Electromagnetic Compatibility (EMC): how well a device resists expected EMI (sometimes also not generating EMI).

b. For each observation below, give the most likely problem category and one measurement or experiment that would help test your explanation. A symptom may have more than one plausible cause. **[6]**

i. The processor resets when a motor starts

Likely PI: motor startup current may cause a supply droop or ground shift. Scope the processor's supply relative to its local ground during motor startup, and compare against the reset/brownout threshold. Other well-supported causes and diagnostic tests, including EMI, are acceptable.

ii. SPI data is occasionally wrong; MOSI rings across the input threshold even when the motor is not connected.

Likely SI: ringing can cause false transitions or timing/logic errors. Scope MOSI at the receiver with a short ground connection. Test whether a suitable series termination or slower edge rate reduces the ringing and errors; also check timing relative to SCLK.

iii. The LCD shows unexpected characters when a separately powered motor runs nearby.

Likely EMI, showing inadequate EMC immunity: the nearby motor's fields or switching noise may couple into the LCD or its wiring. Increase the motor's distance or change the wiring orientation while logging errors. A strong distance/orientation dependence supports this explanation. Check local supply and signal waveforms as another way to distinguish coupling paths.

c. Your board has a 0.1 µF capacitor connected to an IC through long, narrow traces. Why might moving the same capacitor next to the IC improve PI? **[2]**

The long, narrow traces add inductance and resistance between the capacitor and the IC. Moving it close to the IC's power and ground pins reduces those parasitics, so it can supply fast current changes more easily. The capacitor value is the same, but its effective connection is better.

12. Linear regulators (10 points)

a. A regulator has 7.2 V input, 5.0 V output, a 400 mA load, and 8 mA quiescent current drawn from the input. Find the input power, output power, heat dissipation, and efficiency. **[6]**

Total power in is (400 mA + 8 mA) * 7.2 V = 2.94 W. Output power is 400 mA × 5.0 V = 2.000 W. Heat dissipation is input minus output: 2.94 − 2.000 = 0.94 W. Efficiency is 2.000/2.94 ~= 68%.

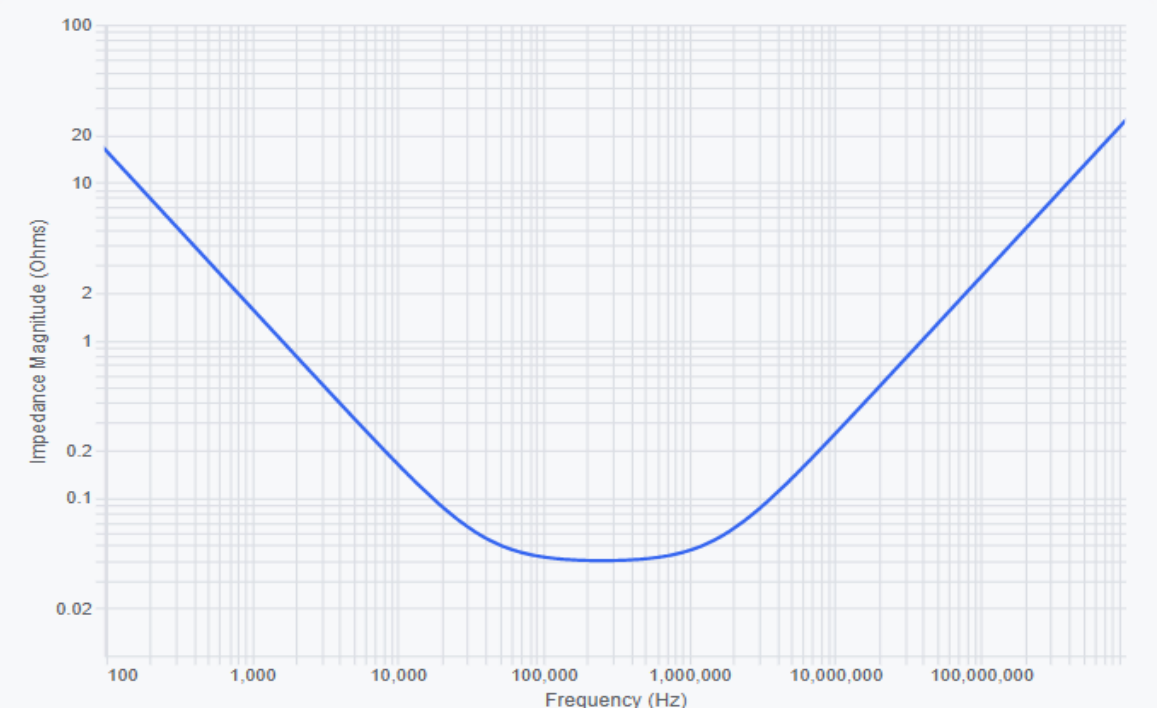
b. The same regulator supplies a 2 mA load while its quiescent current remains 8 mA. Find the efficiency and explain why quiescent current matters especially in a sleeping embedded system. **[4]**

Input power is (2 mA + 8 mA) × 7.2 V = 0.072 W. Output power is 2 mA × 5.0 V = 0.010 W. Efficiency is 0.010/0.072 = 13.89%.

The regulator draws quiescent current even when the load sleeps. Here that current is four times the load current, so it dominates input consumption and greatly reduces efficiency and battery life.

13. Decoupling capacitors (12 points)

The graph shows a series RLC model of a 100 µF capacitor with a resistance of 0.04 Ω and an inductance of 4 nH.

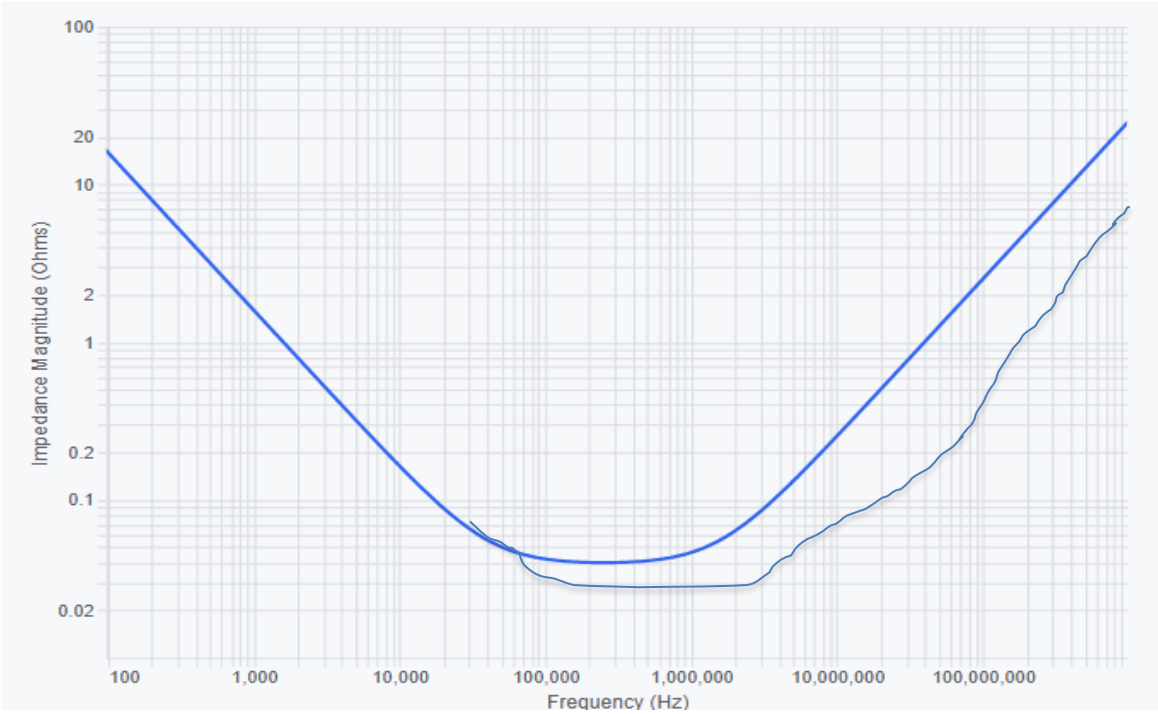


a. Say we were to replace this capacitor with four identical capacitors in parallel, each with C = 25 µF, R =0.08 Ω, and L = 2 nH. Ignore interconnect parasitics. Find the equivalent C, ESR, and ESL. **[4]**

The four capacitances add: C = 4 × 25 µF = 100 µF. For identical branches in parallel, ESR = 0.08/4 = 0.02 Ω and ESL = 2/4 = 0.5 nH. The total capacitance is unchanged, while ESR is halved and ESL is reduced by a factor of eight compared with the original capacitor.

b. Sketch the new impedance curve on the same log axes (try to be close, you don't need to be exact). **[5]**

At low frequencies it basically follows the original curve because the total capacitance is unchanged. Its minimum is 0.02 Ω compared with the original 0.04 Ω. At sufficiently high frequencies it approaches one eighth of the original impedance because ESL is one eighth as large. Sorry the line isn't great…



c. Why can a PCB power/ground plane help at very high frequencies despite having much less capacitance than a bulk capacitor? **[3]**

A power/ground plane pair has very low inductance and resistance. At very high frequencies a bulk capacitor and its leads/traces are mostly inductive, so the power/ground planes can provide a lower-impedance path for fast transient currents despite having much less capacitance.

14. Processor modes and batteries (16 points)

Processor Bob requires 3.3–4.5 V. It executes 120,000 instructions every 500 ms and otherwise sleeps. Ignore wake/sleep overhead.

Mode	Speed	Current
Fast	8 MIPS	180 mA
Slow	4 MIPS	100 mA
Stopped	0 MIPS	4 mA

a. At a fixed supply voltage, compare Fast-then-Stopped with Slow-then-Stopped. Find the average current for each and choose the lower-energy strategy. **[4]**

Fast takes 120,000/(8,000,000) = 0.015 s = 15 ms, leaving 485 ms stopped. Average current is (180 * 15 + 4 * 485)/500 = 9.28 mA.

Slow takes 120,000/(4,000,000) = 0.030 s = 30 ms, leaving 470 ms stopped. Average current is (100 * 30 + 4 * 470)/500 = 9.76 mA.

Fast-then-Stopped uses the least average current, and at a fixed voltage therefore uses the least energy.

Now use the selected strategy for the problems below. Each fully charged cell starts at 4.2 V. The following hypothetical test data apply to this exact pulsed load; no external datasheet or discharge-rate correction is needed. Cells share current equally when paralleled. Protection and balancing are already provided.

Discharge endpoint per cell	Usable charge per cell
3.3 V	90 mAh
3.0 V	100 mAh

b. With one cell connected directly, estimate operating time. Explain why using 100 mAh would be inappropriate. **[4]**

Bob stops operating when the direct-connected cell falls below 3.3 V. Usable charge is therefore 90 mAh. Operating time is 90 mAh/9.28 mA = 9.70 hours. Using 100 mAh would include charge delivered between 3.3 V and 3.0 V, where Bob's supply is below its allowed range.

c. Repeat for two cells in parallel connected directly. **[3]**

Two parallel cells still reach the 3.3 V endpoint, but provide a total of 2 * 90 = 180 mAh. Under the stated assumptions, operating time is 180/9.28 = 19.40 hours. Parallel connection doubles charge capacity while keeping the same voltage.

d. Repeat for two cells in series supplying a 3.7 V linear regulator with 1 mA quiescent current and 0.2 V dropout. Discharge may continue until each cell reaches 3.0 V. Explain why series connection does not double the pack's mAh rating. **[5]**

The series pack has 100 mAh usable capacity to the stated endpoint, not 200 mAh. Pack current is Bob's average current plus regulator quiescent current: 9.28 + 1 = 10.28 mA. Operating time is 100/10.28 = 9.73 hours.

At the endpoint, the pack voltage is 2 × 3.0 = 6.0 V, well above the regulator's required 3.7 + 0.2 = 3.9 V. Its output of 3.7 V is within Bob's 3.3–4.5 V range. The same current passes through both series cells, so their mAh ratings do not add. Series connection doubles voltage and stored energy; a linear regulator dissipates the excess voltage as heat.

15. **Design question** (22 points)

Consider the *answer* to the design question found on the Fall 2024 midterm. We ***strongly*** suggest you try to do the problem on your own before answering this question to get the most out of this. Where appropriate your answers should reference the handout (including a page number) and/or the problem specification. Use the page numbers of the pdf, not the page numbers of the individual documents in the pdf.

a. Part A: why are there pull up resistors on SDA and SCL? **[3]**

I²C uses open-drain signaling: devices pull a line low or release it rather than actively driving it high. Pull-up resistors bring SDA and SCL high when released, allowing devices to share the bus. The VoiceBox interface specifies 10 kΩ pull-ups (handout PDF page 1); the MLX90614's SDA is open-drain in SMBus mode (page 8). On the level-shifted bus, pull-ups establish the proper voltage on each side.

b. Part A: explain why HV1 is connected to A4 on the Arduino. **[4]**

HV1 is the high-voltage side of the level-shifter channel carrying SDA. The Arduino Uno's Wire interface uses A4 for SDA and A5 for SCL, as stated on handout PDF page 17. Thus HV1 must connect to A4 to route SDA to the Arduino's I²C hardware; HV2 carries SCL to A5.

c. Part B: Where is the 0x5A coming from? **[3]**

0x5A is the MLX90614's factory-default 7-bit slave address, listed in the electrical specifications on handout PDF page 7. The read-word example on page 11 also uses SA = 0x5A. It identifies the sensor on the bus; it is different from the register address passed to irSensorReadWord().

d. Part B: Why 3 bytes? **[4]**

The SMBus read-word response contains the low data byte, the high data byte, and one PEC (packet error code) byte. The first two bytes form the 16-bit value; the third is a CRC-8 used for error checking. The supplied exam code reads all three and then ignores the PEC. See the read-word format on handout PDF pages 9 and 11.

e. Part C: Why is the VoiceBox address 7-bit number and why can't it be 0x5A? **[3]**

The VoiceBox uses the standard 7-bit I²C device-address format. The read/write direction is a separate bit on the wire; it is not part of the 7-bit address passed to Wire. The VoiceBox cannot use 0x5A on this bus because the temperature sensor already uses that address, so both would respond to the same transaction. Choose another available, nonreserved 7-bit address. The VoiceBox interface is on handout PDF page 1 and the sensor address is on page 7.

f. Part E: Explain the math of "tempC" and "compensated". **[5]**

tempC = rawTemp × 0.02 − 273.15. The temperature word represents kelvin in steps of 0.02 K per LSB. Multiplying by 0.02 converts the raw count to kelvin; subtracting 273.15 converts kelvin to degrees Celsius. See handout PDF page 12.

compensated = tempC − (3.3 − 3.0) × 0.6 = tempC − 0.18 °C. The sensor is calibrated at 3.0 V, but the prototype supplies 3.3 V. The stated temperature dependence is +0.6 °C/V, so the extra 0.3 V causes a +0.18 °C offset that must be subtracted. See handout PDF page 14 for the dependence and page 15 for the compensation equation.

16. **Project** (8 points)

Which project team are you on? Briefly, what do you think is the most likely reason your project might not be ready by the design expo? What can you do to help avoid that? What help could the 473 staff provide? Your total answer should be about 75 to 150 words.

We'll take any answer that address all four questions in about 75–150 words.