EECS 473 Final Exam Key

Fall 2019

Name: _____ unique name: _____

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. **Do not write anything you want graded on the back pages of the exam.**
3. There are **15** 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.

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.

Potentially useful formulas:

$$\sum_{i=1}^{n} U \leq n(2^{1/n}-1) \qquad C = B\log_2\left(1+\frac{S}{N}\right) \qquad r = \frac{10^{(p_t+g_t+g_r-p_r)/20}}{41.88\times f}$$

1) True/False [11 points]

Circle each of the following statements which are TRUE. **[-2 per incorrectly circled/not circled answer, minimum 0]**

a) *Source decoding* generally involves checking error correction bits and doing error correction/detection as needed.

b) We generally prefer to use primary cells rather than secondary cells when powering a low-power electronic device that is rarely used over a period of years.

c) We sometimes use an LDO followed by a buck converter in order to get both a more efficient conversion and less noise on the output.

d) An LDO which converts 20V to 5V will waste at least 75% of the input power as heat, while an switching regulator would almost certainly waste significantly less than that.

e) 16 QAM modulation sends 4 bits of information in a single encoding.

f) You could reasonably expect a LIPO battery being drained at a constant rate of 0.01C to last at least 90 hours but couldn't expect that same battery being drained at 2C to last for at least an hour.

g) Most PCB fab houses today require a clearance of at least 25 mills between traces.

h) A convolution code is an error correction scheme that uses a fixed block of data rather than a sliding window of the data.

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

j) When doing RM scheduling with 4 ideal tasks, any combination of tasks which has a total CPU utilization of 67.6% or less can be RM scheduled.

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

l) All else being equal you would expect higher frequency wireless signal to travel farther than a lower frequency wireless signal.

2) Short answers [18 points]

a) What is the largest *value* (closest to positive infinity) that can be represented as a 6-bit Q3 number? Be exact, you can give your answer in decimal or as a fraction. **[2]**

3.875

b) What is the binary representation of the smallest value (closest to negative infinity) that can be represented as a 6-bit Q3 number? **[2]**

100000

c) What is an isotropic antenna? Your answer must be 15 words or less. **[3]**

An isotropic antenna radiates its power uniformly in all directions.

d) In the context of communications, what is "PSK" stand for and what does it mean? Please keep you answer short and to the point. **[3]**

Phase Shift Keying. A message is sent by varying the phase of a signal among a finite set of specific values.

e) Say you are sending a signal using the band from 2.0GHz to 2.5 GHz and at the receiver the power of the ambient noise is 1000 times greater than the signal power. According to the Shannon–Hartley theorem, what is highest data rate one could hope to sustain with minimal errors? Show your work and include units. **[4]**

500MHz*log2(1+1/1000)=720.987KHz.

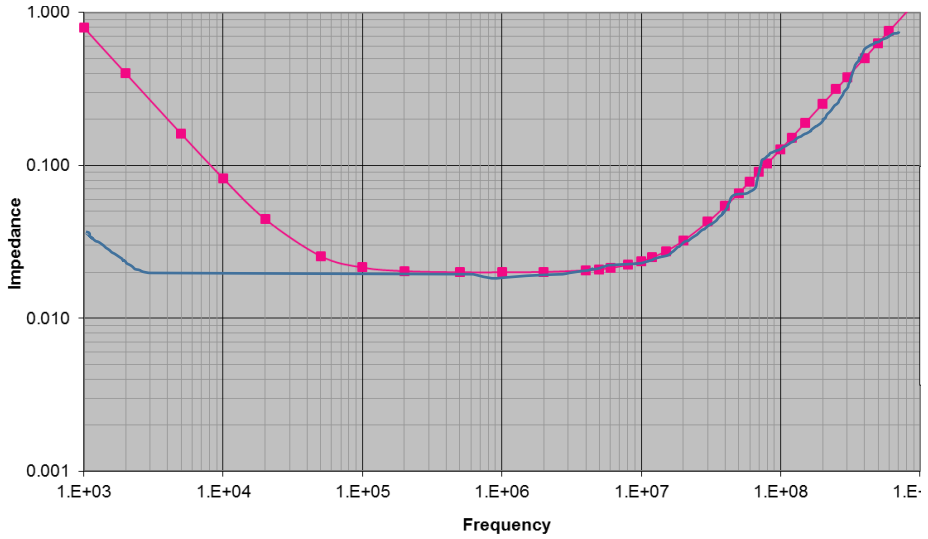
f) How does priority inheritance address priority inversion? Your answer must be 20 words or less. **[4]**

A high priority task waiting on a low priority task using a resource won't wait on a medium priority task.

We will be generous on the word count and clarity as long as you seem to have the right idea.

3) Power integrity [6 points]

The graph on the right shows the frequency vs. impedance for a given capacitor. Say you have a new capacitor with 10 times the capacitance but only twice parasitic resistance and inductance. Redraw the graph showing the same information for two of these new capacitors in parallel.



4) Hamming Codes [7 points]

Say you have a Hamming(6,3)-code where you have 3 data elements (d_1 to d_3) followed by 3 parity bits (p_1 to p_3) where the parity bits are defined as follows (the function P() returns the even one's parity as we did in class, so e.g., P(1,1,1) =1 and P(0,1,1)=0). If p_1 and p_2 are described as follows:

- $p_1=P(d_1, d_2)$; $p_2=P(d_1, d_3)$

What could p_3 be? If there is more than one answer, list them all. *Clearly* justify your answer.

You need to be able to distinguish which bit flipped. Right now if d1 were to flip, p1 and p2 would be wrong, so we'd know that it must be d1 that flipped. But if d2 changed, you couldn't tell if d2 or p1 flipped. And same with d3 and p2. So we need p3 to "watch" d2 and d3. Watching d1 won't hurt. So p3=P(d2,d3) or p3=P(d1,d2,d3) would work.

5) EDF and more [15 points]

Consider an embedded application which consists of 3 tasks named A, B, and C. Each task is CPU bound (that is, there is no I/O or memory operations which take significant time to execute) and periodic. Each task must complete before the next instance of the task is ready to start. These tasks have the following properties and requirements.

- You can assume there is no overhead of any type (including scheduling overhead)
- You can assume that this machine runs each and every instruction in exactly the same amount of time.

Task	Maximum number instructions executed by a single instance of the task	How often the task needs to run
A	1 Million	10ms
B	6 Million	50ms
C	3 Million	90ms

a) Which task would have the highest priority under RM scheduling? Which would have the lowest? **[2]**

- Highest: ___A_____ Lowest: ____**C**_____

b) You are choosing between 4 different processors. Which of these would be the lowest MIPS processor which would be able to schedule these tasks using ***EDF***? You must *clearly* explain your work to get any credit. If none will work, indicate that. **[6]**

- 200 MIPS
- 250 MIPS
- **300 MIPS**
- 400 MIPS

In EDF, you just have to show that the tasks use less than 100% of the CPU. 1MI/.01s+6MI/.05s+3MI/.09s=253.3MIPS. So 250 won't work, but 300 will.

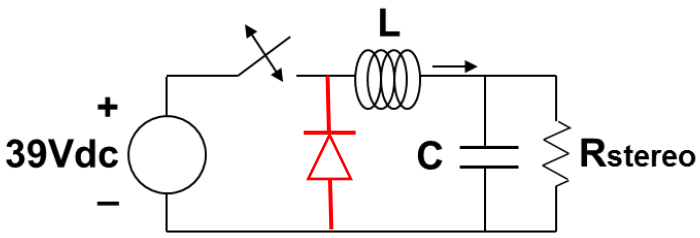
c) You are choosing between 4 different processors. Which of these would be the lowest MIPS processor which would be able to schedule these tasks using ***RM scheduling***? You must clearly explain your work to get any credit. If none will work, indicate that. **[7]**

- 200 MIPS
- 250 MIPS
- **300 MIPS**
- 400 MIPS

Given the above, only 300 and 400 are in play. At 300MIPS 1Million instructions takes 3.3ms. So A takes 3.3ms to turn, B takes 20ms and C takes 10ms. Could do critical instant by hand. But note that in 90ms there would have been 9*3.3ms+2*20ms+10ms of execution time. That's 80ms which we have time for. So task C will complete and thus the whole thing will schedule.

6) Switching power supplies [10 points]

To the right is a simplistic idea of what a buck converter might be.



a) If the switch were open (open circuit) one third of the time and closed the rest, what voltage would you expect across the load? **[2]**

___**26V**_____ Volts

b) However, a real implementation would burn out the switch. Why is that? Your answer must be 15 words or less. **[4]**

When the switch closes, the inductor will continue to try to pull current.

c) How can we fix the problem of the switch burning out? Either modify the above circuit or draw a new one. **[4]**
Diode added in red above.

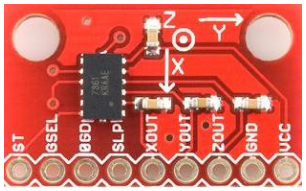
7) Design Problem: Tanks for Everything [33 points]

You are interning for a company that makes radio controlled tanks for hobbyists. They are controlled with a conventional wireless joystick controller. They also have a button to fire the cannon that is an infrared beam that another tank can detect and register as a hit. The hit is transmitted back to the controller that flashes an LED and vibrates the controller.

The company wants to explore a few ideas for the game controller.

First of all, they want to replace the joystick control with something similar to a Wiimote. Holding the controller perpendicular to the ground and the car is stationary. Tilting the controller like a joystick and the car responds similarly. For example, tilt it forward, the car goes forward, tile it backward, it goes backward, etc.

Secondly, they want to add a serially-interfaced OLED display to show the number of hits the tank has taken.



Tilt Sensor

To implement the Wiimote tilt sensor, you suggest using a 3 axis breakout accelerometer like the one in the Wiimote. The accelerometer will allow you to measure tilt with respect to gravity like the Wiimote. You mount the sensor in a plastic tube such that tilting forward produces a positive force on the X axis and backward a negative force. Similarly tilting the sensor right produces a positive force on the Y axis and left a negative force. The X and Y outputs are **2.5 volt at zero force or zero G. At 1G they produce 5 volts. At -1G they produce 0 volts**.

Wireless Interface

Upon taking apart the company's wireless controller you discover the joystick assembly consists of two right angle mounted potentiometers. The potentiometers (POTs) are 10k with the center tap and end taps connected to the wireless control chip. You take the following readings from the middle wiper pin2 of the potentiometers with respect to pin 1.

	Middle	Forward	Back	Right	Left
F/B Pot	5k	10k	0k	5k	5k
R/L Pot	5k	5k	5k	10k	0k

With a bit research you find a dual 10k digitally controlled potentiometer with an SPI interface. It can provide 256 discrete values over the 10k range which you reason is sufficient resolution.

The Display

The display is a simple 16x2 character UART device. Assume the serial port is 9600 baud, 8 bits, no parity and one stop bit. Writing ASCII characters to the port will cause them to be displayed at the current cursor position. The cursor position is set with the following ASCII sequence: "@cxx,y" where is xx is the character position along the width of the display starting at 00 and y is the row number. The position "00,0" is the position in the upper left corner. Sending the ASCII sequence "@C" will clear the screen and put the cursor at the "00,0" position. Those special commands are not themselves printed when sent to the screen.

The Controller

To provide flexibility and room for development you select an Arduino running FreeRTOS to manage the tilt sensing, digital potentiometer control and score display. You will need 2 tasks and an interrupt driven semaphore. **Assume the RTOS tick rate is exactly 16ms.**

- **Task 1:** This task will read the voltage from the tilt sensor, calculate a corresponding resistance value and set the respective digital potentiometer with that value. The task will have the highest priority and run every 80ms.

- **Task 2:** This task will display the number of hits on the serial display as "Total Hits:##". When the number is 16, the display should first be cleared and then print "Game Over". This task will have a priority lower than task 1. The display should only update the value when there is a hit. We will use an ISR (below) to give a semaphore to this task instructing this task to update the value.

- **Interrupt:** This interrupt is generated from a hit signal. The interrupt should just provide a semaphore for Task 2 indicating a hit has occurred.

Reference Pack

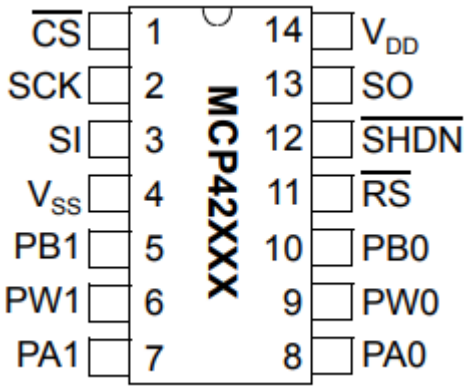
A reference pack is included with the exam with info on Arduino functions, the relevant RTOS functions and selected pages from the data sheet for the digital potentiometer.

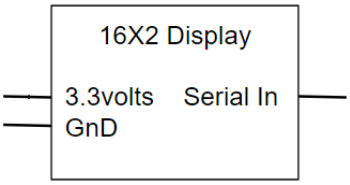
Part 1: Connections [5]

Provide the connections between the Arduino, the digital potentiometer, and the joystick.

- Digital potentiometer A should replace the old forward/backward potentiometer.
- Digital potentiometer B should replace the old the right/left potentiometer
- Assume Accelerometer X axis (Forward/Back) is connected to A0
- Assume Accelerometer Y axis (Right/Left) is connected to A1
- The Hit signal is connected to pin 2.

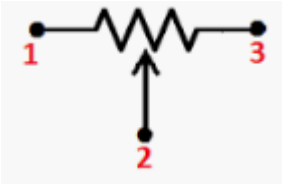
YOU MAY USE LABELS TO MAKE CONNECTIONS





MCP ~CS	AR 10
MCP SCK	AR 13
MCP SI	AR 12
MCP Vss	AR GND
MCP PB1	R/L Pot 1
MCP PW1	R/L Pot 2
MCP PA1	R/L Pot 3
MCP Vdd	AR 5v, 3.3v
MCP SO	AR 11
MCP ~RS	AR 3
MCP PB0	F/B Pot 1
MCP PW0	F/B Pot 2
MCP PA0	F/B Pot 3
Display 3.3	AR 3.3v
Display Gnd	AR GND
Display Serial In	AR 1

Front/Back Pot Connections



Right/Left Pot Connections

The figure above shows where the analog POTs *had* been connected. They have been removed from the joystick and those points (1, 2, 3 for each) are open for you to wire into.

Part 2: Setup Function [9]

Write a function to initialize the digital potentiometer, the SPI port, the UART port and tasks. The digital potentiometer should be reset and the POTs set to mid value. The hit interrupt handler is called *hit_interrupt*.

```
#include <SPI.h>
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

void setup(void) {
SemaphoreHandle_t sem_1;

pinMode(4, OUTPUT);            //~RS
digitalWrite(4,1);             // set inactive, digital outputs are
indeterminate until set.
pinMode(5, OUTPUT);            //~SHDN
digitalWrite(5,1);             // set inactive

SPI.begin();
digitalWrite(4,0);             // reset digital-pot, CS must be high first, so
                               // must occur after SPI.begin
                               //this sets pots to mid value 0x80
digitalWrite(4,1);             // set inactive

SPISettings my_SPI_setting(8 *10^6 ,MSBFIRST, SPI_MODE0);
SPI.begnTransaction(SPI_settings);

//Analog pins do not need to be set to input since they default to input
//init interrupt
pinMode(2, INPUT); //Arduino pins default to input so this is not necessary.
attachInterrupt(2, hit_interrupt, RISING or FALLING);

Serial.begin(9600);  //set baud rate
Serial.print("@c");  //clear display

xTaskCreate(task1, "Task 1", 200, NULL, 2, NULL);  //higher priority than
task 2
xTaskCreate(task2, "Task 2", 200, NULL, 1, NULL);

 sem_1 = xSemaphoreCreateBinary();
  if (sem_1 == NULL) {
    Serial.printl("Semaphore can not be created");
  }

vTaskStartScheduler(); //not necessary in Arduino Free RTOS

void loop() {}
}
```

Part 3: Task 1 [9]

Write task 1 to read the current accelerometer values and set the digital potentiometer accordingly. **[9]**

```
void task1(void *pParam) {

byte digi_pot_x, dig_pot_y;

 portTickType xLastWakeTime;

 const portTickType xFrequency = 5;  //5*16ms = 80ms
//or use pdMS_TO_TICKS(80);

   xLastWakeTime = xTaskGetTickCount(); // initialize the xLastWakeTime variable with the current time.

    for( ;; )    {
      vTaskDelayUntil( &xLastWakeTime, xFrequency );        // Wait for the next cycle.

                    SPI.begnTransaction(SPI_settings); //or In setup

                    byte ax = analogRead(A0)>>2; //ignore low order bits
                    byte ay = analogRead(A1)>>2;
                    digitalWrite(10,0);                 //select SPI device
                    spi.transfer16(0x1100 | ax);
                    spi.transfer16(0x1200 | ay);
                    digitalWrite(10,1);                 //deselect SPI device
                    SPI.endTransaction(); //necessary in
          }
}
```

Alternate 8 bit write method

```
                    digitalWrite(10,0);                 //select SPI device
                    SPI-transfer(0bxx01xx01);         //write cmd, write for pot 0   0x11
                    SPI-transfer(digi_pot_x);
                    SPI-transfer(0bxx01xx10);         //write cmd, write for pot 1   0x12
                    SPI-transfer(digi_pot_y);
                    digitalWrite(10,1);                 //deselect SPI device
                    SPI.endTransaction(); //only necessary if another resource is using spi
```

Part 4: Task 2 [6]

Write task2 that updates the display when the hit count changes displaying the appropriate message.

```
void task2(void *pParam) {


 portTickType xLastWakeTime;
 const portTickType xFrequency = 25;  //25*16=400ms

  xLastWakeTime = xTaskGetTickCount(); // Initialize the xLastWakeTime variable with the current time.

  for( ;; )    {
        vTaskDelayUntil( &xLastWakeTime, xFrequency );        // Wait for the next cycle.

       if (xSemaphoreTake(sem_1, portMAX_DELAY) == pdPASS) {
              if (number_of_hits == 10) {
                     serial.print("@C");
                     serial.print("Game Over");
              }
              else {
                     serial.print("@C");
                     serial.print("total_hits:")
                     serial.printf(number_of_hits, DEC);
              }
       }
}
```

Part 5: ISR [4]

Write the interrupt handler that will set the semaphore.

```
void hit_interrupt(void){

    xSemaphoreGiveFromISR(sem_1, NULL);

}
```