EEECS 473 Final Exam--**Answers**

Fall 2017

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. **Do not write anything you want graded on the back pages of the exam.**
3. There are **14** 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.

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. Circle each of the following statements which are TRUE. **[12 points, -2 per incorrectly circled/not circled answer, minimum 0] (Key: True answers are in red)**

   a) *Source encoding* generally involves adding error correction bits and is dependent on the characteristics of the "source" data.

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

   c) One primary advantage of round-robin scheduling is that it can schedule any set of tasks that have less than 63% total CPU utilization (assuming dependencies between tasks).

   d) A buck converter is a switching power supply where power is supplied by the capacitor when in what is called "discontinuous mode".

   e) Shannon's Limit provides an upper-bound on the rate of useful data we can send over a given channel.

   f) An LDO which converts 20V to 15V can potentially waste less than 15% of the input power as heat, while an switching regulator would almost certainly waste significantly less than that.

   g) When designing a power distribution network, high frequency noise (say over 500MHz) should be largely handled by the capacitor formed from the power and ground planes.

   h) 8-PSK sends 3 bits of information in a single encoding.

   i) You could reasonably expect a LIPO battery being drained at a constant rate of 0.1C to last close to 10 hours.

   j) The three independent parts of a sinusoid we can manipulate for communication are frequency, power, and amplitude.

   k) The FCC worries about EMI issues from boards because even those not designed to generate radio signals might do so.

   l) Necking down a wire can be a useful way to shorten a wire, but if made too small for the current going through the wire it could result in overheating or even melting.

2. Say you have a code where you have 4 data elements (d_1 to d_4) 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).
    - $p_1=P(d_2, d_3, d_4)$
    - $p_2=P(d_1, d_2, d_4)$
    - $p_3=P(d_1, d_2, d_4)$

   The scheme proposed above will not allow for the correction of any single bit of error. Provide a specific example where the parity is generated as described above, a single bit of error is introduced, and it is impossible for the receiver to know which bit was flipped. Be clear what other bit(s) could be flipped in that case. **[8 points]**

   There are a number of examples. One is:

   Want to send (d_1, d_2, d_3, d_4)=0000. Then (p_1, p_2, p_3)=000.
   So $(d_1, d_2, d_3, d_4, p_1, p_2, p_3)$=0000000.
   Let bit d_4, flip (error in transmission).
   We receive $(d_1, d_2, d_3, d_4, p_1, p_2, p_3)$=0001000.
   All three parity bits are wrong. That could be due to either d2 flipping or d4 flipping. We can't fix it.

3. Answer the following questions about fixed-point numbers. For purposes of this question, assume that the "largest value" means "closest to positive infinity" and "smallest value" means "closest to negative infinity". Assume all values are signed. **[6 points]**
   a. What is the largest *value* that can be represented as a 4-bit Q2 number? Be exact and give your answer as a decimal number. **[3]**
   7/4=1.75.

   b. What is the smallest *value* that can be represented as a 5-bit Q3 number? Be exact, and give your answer as a decimal number. **[3]**
   -16/8=-2

4. 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 and that count will be 1024 any time the function is called.

```
ssize_t memory_read(struct file *filp, char *buf,
                    size_t count, loff_t *f_pos) {
      char abc[30] ="abcdefghijklmnopqrstuvwxyz";
      int x=count;

      if(x>3)
            x=3;
      copy_to_user (buf, abc+*f_pos, x);
      printk("<1> fpos= %d\n",*f_pos);

      if (*f_pos <10) {
         *f_pos+=x+1;
         return x;
      } else {
         return 0;
      }
}
```

Now say the user types "cat /dev/txx2". What will happen? In addition to filling in the boxes below indicating what's on the screen and what's in the log file, but sure to clearly explain why. **[12 points]**

First of all, note that this is horrible code.
Each time this gets called, if *f_pos<10 we will copy three characters to buf but increment fpos by 4. *f_pos is being used to figure out which characters are being copied. So we are basically getting 3 characters from the string, then skipping one, getting the next 3 etc. In this case, during the 4th call f_pos will be 12 and so we'll return 0 which is the way we indicate we are at the end of file.

Appears on the screen:

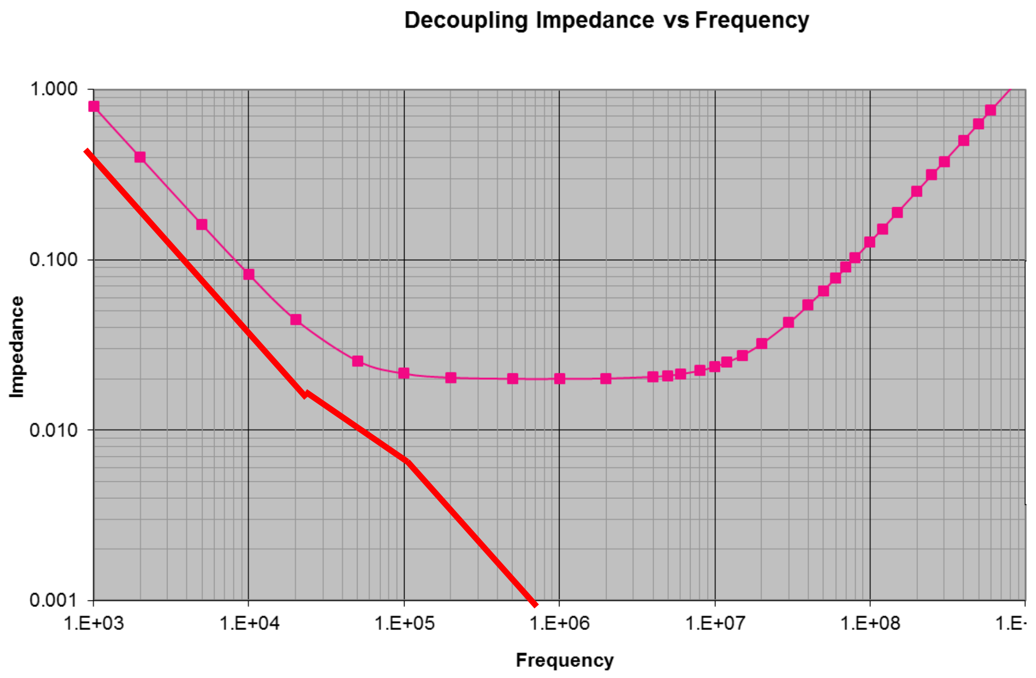
abcefgijk

Appears in the log file:

```
fpos=0
fpos= 4
fpos= 8
fpos= 12
```

5. Explain why a buck converter is more likely to enter discontinuous mode as the load resistance increases. It might help to draw a picture. **[6 points]**

As the load resistance increases, the current supplied dropped. That means the inductor has a lower average current and so is more likely to reach zero. When the inductor has zero current, we are in discontinuous mode.



6. The above graph shows the frequency vs. impedance for a given non-ideal capacitor. Redraw the graph showing the same information if an ideal capacitor with the same capacitance was placed in parallel with the non-ideal capacitor. **[5 points]**

7. Scheduling theory: **[10 points]**
   a. Propose a set of two tasks that are schedulable under EDF scheduling but not when using RM scheduling. All numbers must be single-digit integers. **[5]**

      There are a handful of answers that will work.

Task name	Execution time	Period
A	**3**	**6**
B	**4**	**9**

   b. Propose a set of two tasks that have a total CPU utilization of less than 10% but can't be scheduled using ~~round robin~~ FIFO[1] scheduling (no matter how often the scheduler runs). **[5]**

Task name	Execution time	Period
A		
B		

8. What is the maximum data rate that can be sustained (i.e. channel capacity) for a signal having a being sent in the band between 100MHz and 110MHz, received with a SNR of -30dB? Show your work. Recall that a SNR of 20dB is 100. **[5 points]**

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

10MHz*log$_2$(1+1/1000)~=14420 bits/second

[1] This correction was not made during the exam. We accepted all answers and gave 1 point of extra credit for clear statements that it is impossible.

Never Enough GPIO

It’s the night before your embedded system project demo. You are developing a maze navigating robot and had to add some extra sensors at the last moment. Consequently, you don’t have enough GPIO remaining to control the two motors on the chassis. There are 2 PWM pins are available was well as 3 GPIO pins, but you will need 1 more pin for complete H-Bridge control. Additionally, you need 2 more GPIO pins to detect the rotational speed. We will detect that speed by using a sensor on each wheel that provides a pulse every rotation.

Fortunately, the lab has a SPI interfaced GPIO expander (PCA9502) providing 8 GPIOs. The 8 GPIOs can be configured to be inputs, outputs, and interrupts. If a pin is configured as an interrupt, it will generate an interrupt on a separate IRQ pin. This will come in handy to measure the wheel speed. Thankfully you already have an SPI device attached to your Arduino. It is using the default pins for SPI on the Uno. This means you can attach this breakout to that bus (and thus don’t need to allocate new pins for all of the SPI bus to talk to the PCA9502).

The relevant data sheets for the device are provided in the reference pack including pinouts, SPI requirements, and register maps. The reference pack also includes all the Arduino APIs you will need.

Overview of the wheel control

The goal is to get each of the robot’s wheels moving at a certain speed (measured in rotations per minute or RPM). The current speed of the wheels is measured by using a wheel sensor which goes low for a short time once every wheel rotation. The desired speed is communicated to you via global variables “`desired_left_RPM`” and “`desired_right_RPM`” which will vary over time (set by a function that we are not writing on this exam). Your job is to use PWM to get the wheels to those desired speeds. You will do that using the following equation:

```
PWM = current PWM + (desired speed – current speed)
```

where PWM is the standard Arduino PWM values (which are limited to values in the range of 0 to 255) and the speeds are all in RPM. All RPM values (measured or desired) will have an absolute value no greater than 60. Note that a negative RPM corresponds to going in reverse.

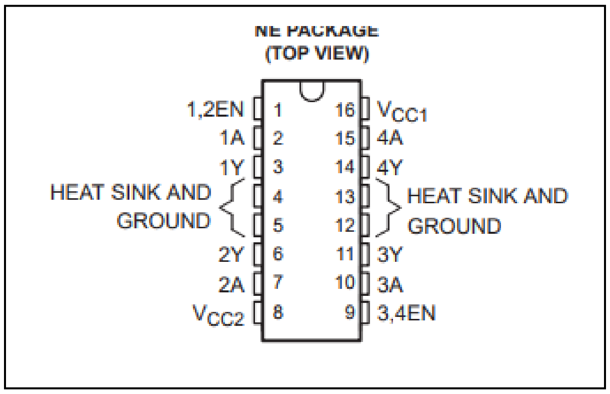
Because our sensor only tells us how fast the wheels are going and not what direction, you can’t know what direction the wheels are actually spinning. You are to assume that if we *desire* to be in reverse that the current speed is in fact in that direction. That won’t always be true, but things will work out if you assume it.

Part 1: Provide the connections between the Arduino and PCA9502. You should also provide power and GND to the PCA9502. Add passive devices as needed. *<u>You may use labels to make connections</u>*. Assume the UNO is powered by USB.

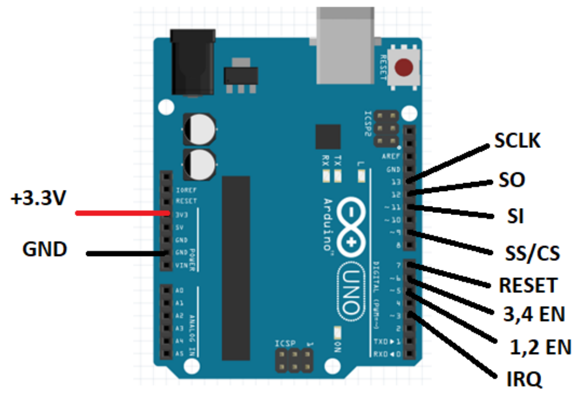
Arduino GPIO pins 3, 7 and 9 are available as general purpose GPIO. Arduino GPIO 5 is connected to "1,2EN" on the H-bridge and GPIO 6 to "3,4EN". The SPI device already connected to the Arduino is using the standard SPI pins (found in the reference pack).

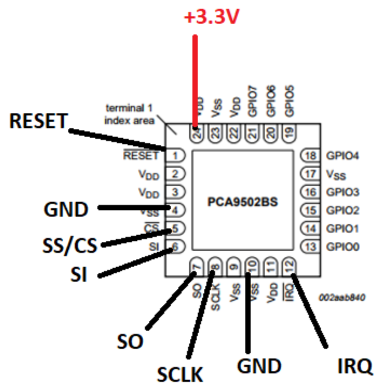
Assume the following connections are made for you between the H-Bridge and wheel sensors to the PCA9502 pins. You do not need to draw the connections to the H-bridge or sensors. **[6 points]**

PCA9502	**H-Bridge**	**motor**
GPIO0	no connect	
GPIO1	1A	left
GPIO2	2A	left
GPIO3	Sensor 1	left
GPIO4	no connect	
GPIO5	3A	right
GPIO6	4A	right
GPIO7	Sensor 2	right



.





Part 2 Write the initialization function for the PCA9502 GPIO expander. This should include Arduino pin initialization and creating an SPI configuration. Note you must reset the device before communicating to it. You also will want to have the function "`measure_speed`" called when there is a change in one of the signals coming from the wheel. Fill in the blank for the SPISettings. **[10 points]**

```
#include <SPI.h>
#define SPI_RESET    7
#define SPI_CS       9
#define SPI_IRQ      3

SPISettings SPIA(10000000, MSBFIRST, SPI_MODE0);
void GPIO_expander_init(void) {

SPI.begin();
pinMode(SPI_RESET, OUTPUT); digitalWrite(SPI_RESET, HIGH);
pinMode(SPI_CS, OUTPUT); digitalWrite(SPI_CS, HIGH);
pinMode(SPI_IRQ, INPUT);

//Reset the GPIO Expander PCA9502 before configuring
digitalWrite(SPI_RESET, LOW); //Reset is active low
delay(1);
digitalWrite(SPI_RESET_HIGH);

//Configure the PCA9502
SPI.beginTransaction(SPIA);
digitalWrite(SPI_CS, LOW);
SPI.transfer(0x0A << 3); //IO Dir register
SPI.transfer(0x66);      //GPIO 1,2,5,6 are set as output
SPI.transfer(0x0C << 3); //IO Enable Int register
SPI.transfer(0x88);      // Interrupt on GPIO 3 and 7
SPI.transfer(0x0E << 3); // IO Config register
SPI.transfer(0x01);      // Enable Interrupt latch
digitalWrite(SPI_CS, HIGH);
SPI.endTransaction();

//Enable interrupt on IRQ pin
attachInterrupt(SPI_IRQ, measure_speed, LOW);
}
```

Extra space for Part 2

Part 3 Write the interrupt routine to measure the wheels speed. You are provided the following timer function that provides a simple way to measure elapsed time.

```
int timer1()
int timer2()
```

Each function returns the time since that function was last called in 100th of a second. The first time each function is called it returns a 0.

The time should be measured for both motors and saved in the global variables `left_mot_RPM`, and `right_mot_RPM`. These are absolute values ranging from 0 to 60 (as integers). You may not use floating point value here, don't worry about which way you round. **[10 points]**

```
int left_mot_RPM;
int right_mot_RPM;
void measure_speed(void) {
   uint8_t val=0;
   SPI.beginTransaction(SPIA);
   digitalWrite(SPI_CS, LOW);
   SPI.transfer(0x80 | (0x0B << 3)); //Write to register 0x0B
   val = SPI.transfer(0); //Dummy data so to read on the SPI bus
   digitalWrite(SPI_CS, HIGH);
   SPI.endTransaction();
   if(!(val & 0x08)) //If 3rd bit is set
   {
      left_mot_RPM = 6000/timer1(); //timer1() is cleared
   }
   if(!(val & 0x80)) //7th bit is set
   {
      right_mot_RPM = 6000/timer2(); //timer2() is cleared
   }
}
```

Extra space for Part 3

Part 4 The wheels have different voltage vs speed characteristics and need to be monitored and adjusted constantly.

Write the function that controls the motor speed by monitoring the measured wheel speed (e.g. `left_mot_RPM`) and adjusting the motor speed based on the desired motor speed (`desired_left_RPM`). You are to use the formula found in the "Overview of the wheel control" section on the first page of this problem. You should ensure that if the desired speed has a negative value, you are putting the motor in reverse. Assume the motors are set up so that 1A=1 and 2A=0 moves the left motor in a forward direction and 3A=1 and 4A=0 moves the right motor in a forward direction. **[10 points]**

```
#define LEFT_MOT_EN       5
#define RIGHT_MOT_EN      6

int left_PWM = 0;
int right_PWM = 0;

void adjust_speed(void) {
//Probably can do error band check
if(left_mot_RPM != desired_left_RPM)
{
      SPI.beginTransaction(SPIA);
      digitalWrite(SPI_CS, LOW);
      if(left_PWM < 0)
      {
            SPI.transfer(0x0B << 3); //Write to GPIO
            SPI.transfer(0x04);
      }
      else
      {
            SPI.transfer(0x0B << 3); //Write to GPIO
            SPI.transfer(0x02);
      }

      digitalWrite(SPI_CS, HIGH);
      SPI.endTransaction();
      left_PWM = left_PWM + (desired_left_RPM - left_mot_RPM);
      analogWrite(LEFT_MOT_PIN, left_PWM);
}

if(right_mot_RPM != desired_right_RPM)
{
      SPI.beginTransaction(SPIA);
      digitalWrite(SPI_CS, LOW);
      if(right_PWM < 0)
      {
            SPI.transfer(0x0B << 3); //Write to GPIO
            SPI.transfer(0x40);

      }
      else
      {
            SPI.transfer(0x0B << 3); //Write to GPIO
            SPI.transfer(0x20);
      }
      digitalWrite(SPI_CS, HIGH);
      SPI.endTransaction();
      right_PWM = right_PWM + (desired_right_RPM - right_mot_RPM);
      analogWrite(RIGHT_MOT_PIN, right_PWM);
}
```