Name: Solution **RIN:**

Q.1 (3 pnts): Given a PWM signal with a pulse width of 2.5 ms and a duty cycle of 35 %, what is the signal's period?

$DC = \frac{PW}{Period}$ $Period = \frac{2.5ms}{0.35} = \boxed{7.14ms}$

Q.2 (3 pnts): In terms of what a circuit connected to a GPIO pin experiences: What are the differences, if any, between a 100 % duty cycle PWM signal being generated by the GPIO and a digital output HIGH?

Nothing; Both are constant high

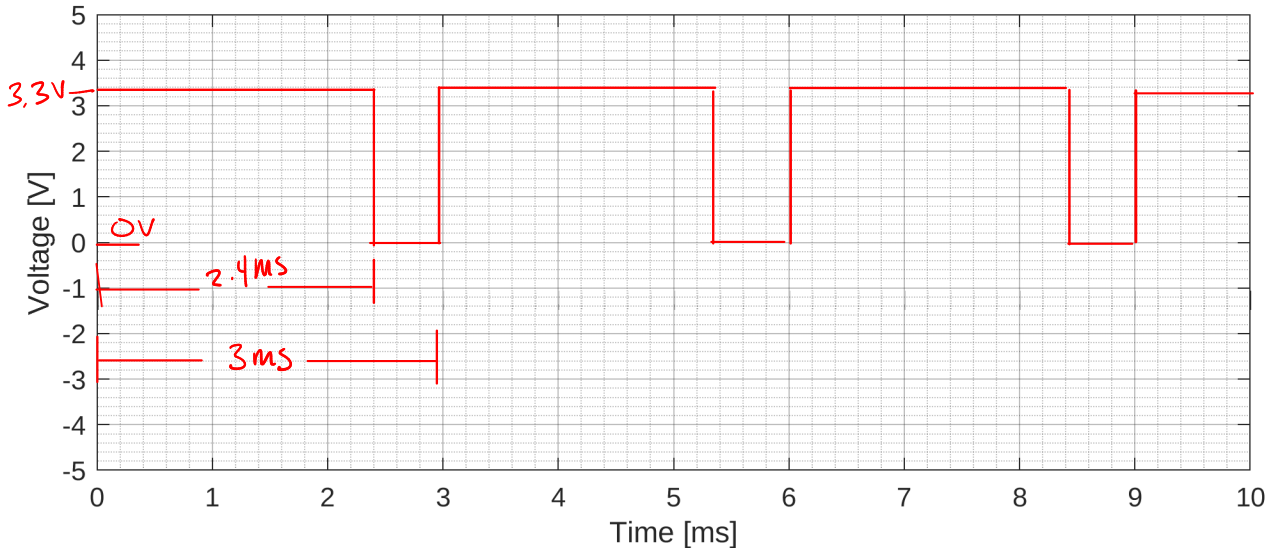
Q.3 (4 pnts): A PWM signal is generated using a **clock divider of 8**. An Encoder measures the PWM signal's period using a timer capture module running with a **clock divider of 2**. Both are being sourced from the same system clock of **16 MHz**. If the PWM period is 5 ms, how many timer counts will be registered between the encoder capture events?

Could be done a few ways... one:

$N_{timer} = \left(\frac{16M}{8}\right) 0.005 = 10{,}000$

$N_{enc} = N_{timer} \times \frac{8}{2} = \boxed{40{,}000}$

Q.4 (4 pnts): Draw a PWM signal that has a period of 3 ms and a duty cycle of 80 %. The PWM signal should have an output voltage consistent with those generated in the laboratory.



Name: **RIN:**

Q.5 (4 pnts): A 120 tooth encoder is mounted to a wheel that has a diameter of 15 cm. How often would the encoder be triggering (in time) if the wheel was moving at 1 m/s? Recall that $v = \omega r$.

$\omega = 1/7.5\text{ms} = 13.\overline{3}\text{ rad/s}$

$f_{wheel} = \omega/2\pi = 2.12\text{ Hz}$ $\quad T_{enc} = \frac{1}{f_{enc}} = \boxed{3.9\text{ms}}$

$f_{enc} = f_{wheel} \times 120 = 254.7\text{Hz}$

Q.6 (6 pnts): An encoder is serviced with the simplified pseudocode `Encoder_ISR()` below. A snapshot of the variable values from two consecutive encoder events is also given. Assuming a 16-bit timer and the timer is incrementing at **16 MHz**, how much time passed between the two events?

```
void Encoder_ISR()
    If timer period triggered interrupt:
        Noverflow += 1;
    If Encoder triggered interrupt:
        Nencoder = compare value
```

```
Event N-1:
   Noverflow = 261
   Nencoder = 8675
Event N:
   Noverflow = 309
   Nencoder = 31415
```

$N(k-1)$ — $N(k)$

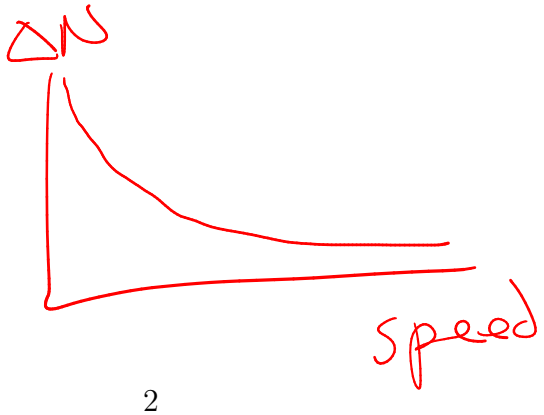
$N_{overflows} = 309 - 261 = 48$

$$\Delta t = \frac{(N(k) - N(k-1)) + 2^{16} N_{overflows}}{f_{tclk}}$$

$$\Delta t = \frac{(31415 - 8675) + 2^{16} \times 48}{16\text{MHz}} = \boxed{198\text{ ms}}$$

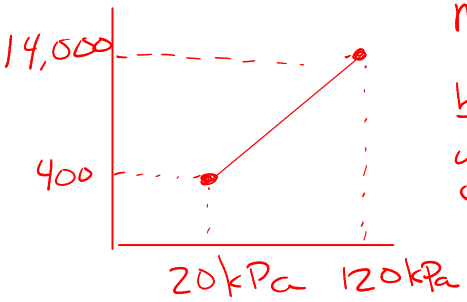
Q.7 (2 pnts): True or **False**: Decreases in the speed of a wheel correspond to decreases in the timer counts between encoder events.

false



Name: **RIN:**

Q.8 (6 pnts): An ADC is configured for single-ended 14-bit resolution to read an analog pressure sensor. If the ADC output value for 20 kPa is 400 and the output for 120 kPa is 14,000, what would the expected output be for 50 kPa? Assume the sensor has a linear output.



$m = \frac{14000 - 400}{120k - 20k} = 136 \frac{ADC}{kPa}$

$b = 400 - m\,20k = -2320\,ADC$

$y = mx + b$

$m(50kPa) + b = \boxed{4480}$

Q.9 (3 pnts): For the same system as above, what is the minimum pressure that the ADC can read?

minimum is @ 0 ADC:

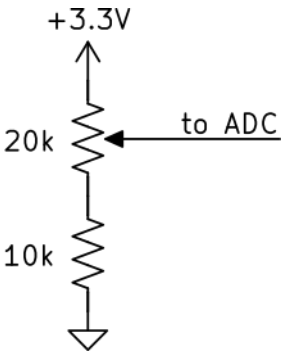
$0 = mx + b \rightarrow x = \boxed{17.1\,kPa}$

Q.10 (3 pnts): What additional information is needed to calculate the corresponding sensor output voltages for the above readings?

Need to know the voltage references

V_{REF}^{+} & V_{REF}^{-}

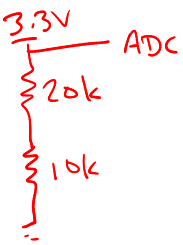
Q.11 (4 pnts): A potentiometer is wired as shown. What is the minimum and maximum voltages that can be measured from the potentiometer's wiper connection?



min:

$\rightarrow 3.3V \frac{10}{10+20} = \boxed{1.1V\ min}$

max:



$\rightarrow$ $\boxed{3.3V\ max}$

Name: **RIN:**

Q.12 (8 pnts): A new I2C peripheral, address 0x42, has the registers as given in the table to the right. The device is read-only. Each register's initial value is also given in the table. Provide the necessary code to read the SW revision byte and all of the data bytes. Further, combine the data bytes into the full 32-bit number they represent and save into a variable `output`. Assume the I2C module to be used is saved into the global variable `uint32_t I2CMOD`.

Register	Function	Init. Value
0	Self-Check	0xFF
1	SW Revision	0x04
2	Data 0, LSByte	0x25
3	Data 1	0x99
4	Data 2	0xA7
5	Data 3, MSByte	0x3D

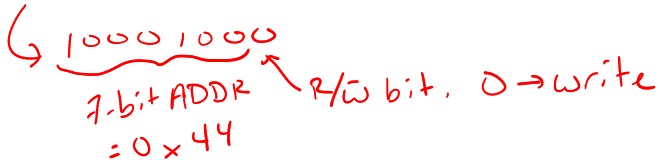
```
uint8_t data[5];
i2c_readData(I2CMOD,0x42,1,data,5);
output = (data[4] << 24) + (data[3] << 16) + (data[2] << 8) + data[1];
    // or anything else that would work...like:

output = 0;
uint8_t i;
for(i=4;i<0;i--){
    output <<= 8;
    output += data[i];
}
```

Q.13 (3 pnts): Given the initial values, what should the value of variable `output` be in hexadecimal?

0x3DA79925

Q.14 (5 pnts): For another I2C device, the following is observed on the bus during a transfer. Assuming `I2CMOD` is used again, provide the C statement(s) that would produce the observed transfer. Assume an array `uint8_t data[6]` is defined.

START – 0x88 – ACK – 0x04 – ACK – 0x05 – ACK – 0x06 – ACK – STOP



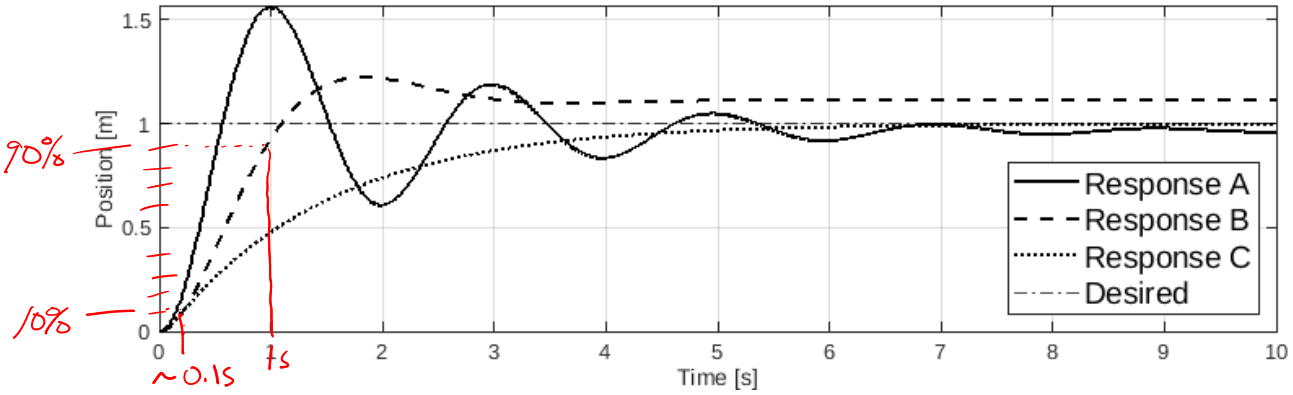

```
data[0] = 0x05;
data[1] = 0x06;
I2C_writeData(I2CMOD,0x44,4,data,2);
```

Q.15 (4 pnts): There are four I2C devices on a bus, with the 7-bit addresses `0x20`, `0x21`, `0x40`, and `0x41`. The first byte (first 8 bits) transferred over the bus for one packet is valued as `0x41`. What does this tell you about the packet?

0x41 → 01000001
7-bit ADDR = 0x20
R/W = 1 Read

Packet is a read to device 0x20

Below is the system responses from Quiz 4. Answer the remaining questions on this page using this figure.



Q.16 (2 pnts): Which response has the lowest amount of inherent damping?

A

Q.17 (2 pnts): What is the approximate rise time of response B?

90% ≈ 1s, 10% ≈ 0.1s, rise time ≈ 0.9s

Q.18 (2 pnts): Assuming each system is controlled by a PID controller. Which response would benefit the most from a **reduced proportional gain constant**?

A: quick rise causes overshoot

Q.19 (2 pnts): Assuming each system is controlled by a PID controller. Which response would benefit the most from a **reduced derivative gain constant**?

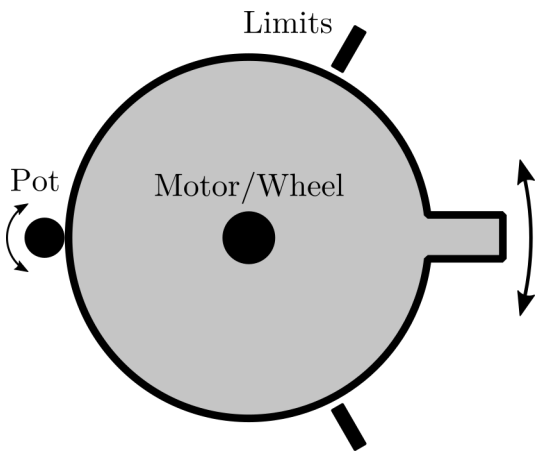
C: too much damping slows it down

Q.20 (2 pnts): **True** or **False**: Response B never reaches "steady-state."

has steady-state error, still reaches steady-state

For all written code, you may shorten the names of constants and functions if what you intend to write is clear. Some provided code may also be shortened. Pseudocode will receive minimal partial credit.

A motor with a wheel attached is used to accurately position a device along it's arc. The wheel edge drives the shaft of a multi-turn potentiometer; where the potentiometer is used to provide a positioning feedback signal for the mechanical system. The potentiometer, a 10 kΩ pot, is wired between 0 V and 3.3 V such that as the motor turns clockwise (motor "forward," or positive direction) the potentiometer wiper output voltage increases. The position of the wheel is restricted by two "limits" which prevent the wheel from turning further. Both limits have "limit switches" mounted on them to indicate that the motor has contacted the limit. The limits are located at ±60° from the center position, 0°, as shown.



The motor operates the same as in the RSLK (P3.6:ENABLE(1), P5.5:DIR(0-FORWARD), P2.6:PWM). The +60° limit switch is on P4.0, and the −60° switch is on P4.1, both LOW when pressed.

Q.21 (6 pnts): Write a function to read and return the ADC value of the potentiometer in terms of a percentage rotated; that is, 0 ADC should correspond to 0% and the maximum value of the conversion returns 100%. Assume that the ADC is initialized to sample and save the potentiometer to `ADC_MEM0` with 10-bit conversions. All other configuration is as done in the laboratories.

```
float potVal(void){

  ADC14_toggleConversionTrigger();
  while(ADC14_isBusy());
  uint16_t result = ADC14_getResult(ADC_MEM0);
  return result*100/(1024.0);



}
```

Name: **RIN:**

Q.22 (12 pnts): The system must be calibrated each use. To do this, the program must calculate the `gain` and `offset` values for the equation `position = gain * PotADC + offset`, where `PotADC` is the ADC output from measuring the potentiometer voltage and `position` is the angle of the wheel, in degrees. Two data points are needed for this calculation: the potentiometer value at each limit switch. Assume that all GPIO pins are configured as necessary and the motor PWM has been set to 50 % duty cycle.

Finish the below function to complete the calibration. The two data points must be acquired within this function. You must use the `potVal()` function from the previous question.

```
float gain,offset; // Globals
void Calibrate(void){
  // Turn the motor on and spin counterclockwise ("reverse" direction)
  GPIO_setOutputHighOnPin(GPIO_PORT_P3,GPIO_PIN6);
  GPIO_setOutputHighOnPin(GPIO_PORT_P5,GPIO_PIN5);


  // Wait for -60deg limit switch to be pressed
  while(GPIO_getInputPinValue(GPIO_PORT_P4,GPIO_PIN1) == 1);



  // Aquire data point 1
  float dp1 = potVal();


  // Turn wheel clockwise and get data point 2
  GPIO_setOutputLowOnPin(GPIO_PORT_P5,GPIO_PIN5);
  while(GPIO_getInputPinValue(GPIO_PORT_P4,GPIO_PIN0) == 1);

  float dp2 = potVal();



  // Turn motor off
  GPIO_setOutputLowOnPin(GPIO_PORT_P3,GPIO_PIN6);


  // Calculate gain and offset
  gain = 120/(dp2 - dp1); // 120 = 60deg - (-60deg)
  offset = 60 - gain*dp2;




}
```

Q.23 (10 pnts): The desired wheel location is set by an **input** PWM signal, measured by a timer capture module. A 0% duty cycle corresponds to a desired -50° position and a 100% duty cycle corresponds to a desired +50° position. Assume that the capture module is configured to measure the total timer counts between the **rising and falling edges of the pulse** and save into the global variable `uint16_t pw`, where the period of the PWM is known to always be 50,000 encoder timer counts.

Write the control functionality within the `while(1)` loop below such that the motor will spin to the desired location. The implementation **must use proportional control**, where the actual position is measured using the equation derived on the previous page.

The alternate function for P2.6 is TA0.3. The PWM duty cycle should never exceed 50%, which corresponds to 400 pwm timer counts. Assume the motor is enabled and initializations are complete.

```
float kp = #; // Proportional gain constant (leave as unknown)
while(1){

  // Get desired position:
  float desired = 100*(pw/50000.0) - 50;

  // Get actual position:
  float actual = gain*potVal()+offset;

  // calculate error:
  float error = desired - actual;

  // calculate control signal
  int16_t pwm = kp*error;

  // check if error is positive or negative, adjust wheel direction accordingly
  if(pwm > 0)  GPIO_setOutputLowOnPin(5,GPIO_PIN5);
  else         GPIO_setOutputHighOnPin(5,GPIO_PIN5);

  // Limit the pwm to positive and 50%
  pwm = abs(pwm);
  if(pwm > 400) pwm = 400;

  // Assign the calculated pulsewidth to the motor
  Timer_A_setCompareValue(TIMER_A0_BASE,CCR3,abs(pwm);




}
```