Notes:

- Homework is individual work. Adhere to the University’s policy relating to the integrity of scholarship. See [http://catalog.iastate.edu/academiclife/regulations/](http://catalog.iastate.edu/academiclife/regulations/), “Academic Dishonesty”.
- Homework must be typed and uploaded to BlackBoard as a PDF or Word Document (i.e. .doc or .docx) only.
- Late homework is accepted within three days from the due date. *Late penalty is 10% per day (for each 24 hours following your class).*

**Note:** Unless otherwise specified, all problems assume the ATmega128 is being used.

**Question 1: ADC Design Principle (10 pts)**

Suppose that an ATmega128 is used with a pressure sensor to monitor the pressure exerted on a valve. The pressure sensor measures pressure from 50.0 psi (pounds per square inch) to 550.0 psi and converts it proportionally (i.e. linearly) to an electrical signal in the voltage range from 0V to 2.5V. Assume the reference voltage (i.e. max voltage) for the ATmega128’s ADC is 5V.

a. If the gas pressure is 350.0 psi:

i) what is the voltage level at the sensor’s output? (2pts)

**Solution:**

**Step 1:** Find the sensitivity of the sensor. This is the slope \((m)\) of the above figure.

\[
m = \frac{\text{RISE}}{\text{RUN}} = \frac{(550 - 50)}{(2.5 - 0)} = \frac{500}{2.5} = 200 \text{ PSI/V}
\]

**Step 2:** Use equation for a line: \(y = mx + b\). Let \(y\) = pressure, \(m\) = slope of line, \(x\) = sensor output voltage, and \(b\) = \(y\)-intercept.

For this case: \(m = 200\), and \(b = 50\)

So. \(y = 200x + 50\), solve for \(x\), \(x = (y - 50)/200 = (350 - 50)/200 = 300/200 = 1.5\ \text{V}\)
ii) What is the digital reading from the ATmega128's ADC? (3 pts)

**Background:** The above figure can be drawn based on the ATmega128 ADC specifics: i) it is 10-bit, ii) having a Vref of 5.0V, means Vmax is 5.0V, iii) ATmega128 ADC has $2^n$ steps.

**Step 1:** Compute the resolution of the ADC. This is just the slope of the above graph

$$m = \text{RISE/RUN} = \frac{(5 - 0)}{(1024 - 0)} = \frac{5}{1024} = .00488 \text{ V/bit}$$

**Step 2:** Compute digital output

From part a.i), we know the input to the ADC is 1.5 V

Use equation for a line: $y = mx + b$. Let $y =$ input Voltage, $m =$ slope of line, $x =$ digital output, and $b =$ y-intercept $= 0$.

$1.5 = .00488x$. Solve for $x$: $x = 1.5/.00488 = 307.38$. Since the digital output cannot have a fractional part, the digital output of the ADC is **307 or 0x133 or 0b01_0011_0011 (10-bits)**

**Grading Note:** If $2^n - 1$ steps is used, but the answer is consistent with this assumption, then do not deduct any points.
b. If the digital reading was 200, what is the range of possible analog values just read? Note: each digital number corresponds to a step (a small range in the span) of the input analog value. (5pts)

The lowest voltage that will give a digital reading of 200 is:

\[ y = mx + b = 0.00488 \times 200 + 0 = 0.976 \text{ V}. \]

The lowest voltage that will give a digital reading of 201 is:

\[ y = mx + b = 0.00488 \times 201 + 0 = 0.98088 \text{ V}. \]

So the range of voltages that could give a digital reading of 200 is:

\[ 0.976 \text{ V} < \text{Input Voltage} < 0.98088 \text{ V}. \]
**Question 2: Shutdown system (15 pts)**

A pressure sensor is embedded in the ramp below. When pressure is applied to the sensor, a logic 1 is driven on the Timer 1 Input Capture input wire of the ATMega 128, else a logic 0 is driven. You are to write a C-program that will apply the car’s breaks to stop it if its velocity is not large enough to jump the gap shown below. This program is required to use the Timer 1 Input Capture interrupt.

Assumptions:
1) The car is moving at a constant velocity while on the ramp
2) The car is treated as a “point mass” for computing the physics of the problem

![Diagram of the ramp and car](image)

**a) Complete Config_Timer1 to program the Timer 1 configuration registers as follows: (5 pts)**

- .25 MHz Timer 1 clock rate (also known as Timer tick rate)
- Input Capture interrupt enabled
- Detect positive edge events

```c
Config_Timer1()
{
    // COM bits set to 0, no output compare being used
    // WGM3 – WGM0 = 0000, for input capture (Normal Mode)
    TCCR1A = 0b00000000; // bits 1–0 (WGM1 WGM0)

    // bits 7: ICNC = 1 (Enabled noise cancellation, **optional**)
    // bits 6: ICES = 1 (Select Positive edge)
    // bits 4–3: WGM1 WGM0 = 00 (Normal mode)
    // bits 2–0: CS2 CS1 CS0 = 011(prescaler 64 gives .25 MHz tick rate)
    TCCR1B = 0b11000011;
    TIMSK = 0b00100000; // Bit 5 enables Input Capture Interrupts
}
```
b) Complete the ISR to store first_wheel_hit and second_wheel_hit (5 pts)

c) Complete Stop_car(). It should return a 1 if the car is not moving fast enough to jump the gap (5 pts)

```c
// Global variables (you may additional variables)
volatile unsigned int first_wheel_hit; // When 1st wheel hits sensor
volatile unsigned int second_wheel_hit; // When 2nd wheel hits sensor
volatile int done_flag = 0; // 1 after both first and second_wheel_hit have been stored

// Store first_wheel_hit and second_wheel_hit
ISR(TIMER1_CAPT_vect)
{
    static int state = 0; // Using a global instead also fine
    // Capture rising edge time for when 1st wheel hits sensor
    if(state == 0)
    {
        first_wheel_hit = ICR1;
        state = 1;
    }
    else // Capture rising edge time for when 2nd wheel hits sensor
    {
        second_wheel_hit = ICR1;
        done_flag = 1; // tell main done capture 1st and 2nd wheel hit times
        state = 0;
    }
}
```
// Return 1 if the car is not fast enough to jump the gap
int Stop_car(void)
{
    float velocity_car;
    float time;

    // Compute car velocity based on captured times and distance
    // between wheels: d = v*t => v = d/t

    // Assuming at most one overflow, so not dealing with overflow
    // .25MHz tick rate => 4us tick period

time=(second_wheel_hit - first_wheel_hit) * .000004 // In sec
velocity_car = 1.5/time; // In m/s

    // Assume min velocity was computed offline, fine if computed
    // in program. Min_velocity is about 31.2 m/s (see below for
    // one way to compute off-line)
if(velocity_car < 31.2)
{
    return 1; // Not fast enough stop
}
else
{
    return 0; // Fast enough make jump
}
}
// Stop Car if it is not fast enough to jump the gap.
main(){
    int stop = 0;
    Config_Timer1();

    while(1)
    {
        // Wait for events to be captured
        while(!done_flag){
        }

        done_flag = 0; // Clear flag

        // Check if car needs to stop
        stop = Stop_car();

        if(stop)
        {
            lprintf("Stopping Car!");
        }
        else
        {
            lprintf("Car going to Jump!!");
        }
    } // end while
}

// Compute min velocity
1) $dx = V_{car\_x} \times t_{air}$, where $t_{air}$ is time car is in air and $V_{car\_x}$ is the car's horizontal velocity, and $dx$ is 100 m for this case

2) $dx = V_{car\_x} \times (2\times t_{up})$, where $t_{up}$ is the time for the car to reach its max height after leaving ramp ($t_{air} = 2\times t_{up}$)

3) $V_{top} = V_{car\_y\_initial} + a \times t_{up}$, where $V_{top}$ is the car's vertical velocity at its highest point (i.e. 0 m/s), $V_{car\_y\_initial}$ is the car's initial vertical velocity after leaving the ramp.

4) $V_{car\_x} = \cos(45) \times V_{car} \approx .71\times V_{car}$, and $V_{car\_y\_initial} = \sin(45)\times V_{car} \approx .71V_{car}$, where $V_{car}$ is the speed of the car.

5) Solve for $t_{up}$, given $V_{top} = 0$, $a = -9.81$.

   $t_{up} = .71\times V_{car} / 9.81$.

6) Subsitute $t_{up}$ and $.71\times V_{car}$ into 2) and solve for $V_{car}$

   $100 = .71\times V_{car} \times (2\times.71\times V_{car}/9.81)$, $V_{car} \approx 31.2 \text{ m/s}$
Question 3: Software implemented Input Capture (10 pts)

a) Assume the ATMEGA128 does not have Input Capture hardware or Interrupts. Write a C program to save TIMER1’s count value (i.e. TCNT1) when a positive edge event occurs on PortD, pin4. Note: you may want to do 3c first. (4 pts)

```c
int main(void)
{
    unsigned int rising_edge_time;

    // For grading do not mark off for not having this
    DDRD &= 0b11101111; // set wire 4 of PortD for input

    while(1)
    {
        // Check if the current value of wire 4 is 0
        if( ( PIND & 0b00010000) == 0)
        {
            // wait for wire 4 to become a 1
            while( !(PIND & 0b00010000))
            {
            }
            rising_edge_time = TCNT1; // Store counter value
        }
        return 0;
    }
}
```

b) Describe two disadvantages of your software-implemented input capture program, as compared to using Input Capture hardware with Interrupts. (3 pts)

1. Does not allow other parts of the code to run, while it is running.

2. With the "software only" approach an interrupt could occur after detecting a positive edge, but before the TIMER value is read. Thus adding to the error of when the positive edge occurred.

3. The Input Capture hardware stores the value of TCNT as soon as the event occurs, while the software implementation will have error related to where in the polling loop the code is when the positive edge occurs.
c) Complete the bubble diagram below to implement a Mealy State Machine that implements a positive edge detector (i.e. the state machine outputs 1 each time a positive edge occurs on the input). Assume at Start the input to the state machine is initially 0 (3 pts)
**Question 4: Edge Detection (15 pts)**

The following simple block diagram illustrates how one may implement an Input Capture circuit.

An example timing diagram for capturing the rising edge of Input_signal is given below. In summary, on the occurrence of a positive edge on Input_signal: 1) Write_EN pulses ‘1’ for one sys_clk cycle, and 2) the value in the Timer/Counter Register (TCNT) is loaded into the Input Capture Register (ICR).

Assume: Any time Input_signal transitions is will hold its value for at least 1 sys_clk cycle.
a) Draw out a digital circuit to implement an Edge Detector that creates a 1-sys_clk-wide pulse on Write_EN to load the TCNT register into the ICR register when a rising edge occurs on Input_signal. The only components you can use are D-Flip Flops, and AND, OR, NOT gates (7 pts)

![Positive Edge Detector Diagram](image1)

b) Repeat a) for an edge detector that creates a Write_EN pulse for detecting the falling edge of Input_signal. (3 pts)

![Negative Edge Detector Diagram](image2)
c) The ATMEGA128 allows for detecting either the positive edge or negative edge of an input by configuring the “Edge Select” bit of a configuration register. Draw the digital circuit to allow the Edge Detector to detect a positive edge when ES=1, and a negative edge when ES=0. You may now use multiplexers in addition to D-Flip Flops, and AND, OR, NOT gates (5 pts)

![Configurable Edge Detector Diagram]