**Question 1.** Write a C function that samples ADC channel 4. 0 maps to 0, and 1023 maps to 5000. You need to promote to long so that you avoid overflow.

```c
// Sample ADC channel 4
// Inputs: none
// Outputs: voltage in 0.001 V units
unsigned short ADC_In4(void){
    unsigned short sample, voltage;
    unsigned long sample5000;  // sample *5000
    ATDCTL5 = 0x84;  // start ADC, channel 4, right justified
    while((ATDSTAT0&0x80)==0){}; // wait for SCF
    // also while((ATDSTAT1&0x01)==){}; // wait for CCF0
    sample = ATDDR0;
    sample5000 = 5000*sample;
    voltage = (unsigned short)(sample5000/1023); // 0 to 5000
    return voltage;
}
```

**Function: ADC_In4**

0000 c684 [1]     LDAB  #132
0002 5b00 [2]     STAB  _ATDCTL45:1
0004 4f0080fc [4]  BRCLR  _ATDSTAT0,#128,*+0 ;abs = 0004
0008 dd00 [3]     LDY  _ATDDR0
000a cc1388 [2]     LDD  #5000
000d 13 [3]     EMUL
000e ce03ff [2]     LDX  #1023
0011 34 [2]     PSHX
0012 ce0000 [2]     LDX  #0
0015 34 [2]     PSHX
0016 160000 [4]  JSR  _LDIVU
0019 3d [5]     RTS

Leaving it as a short causes overflow at the multiply by 5000 step.

```c
unsigned short ADC_In4Bad (void){
    unsigned short sample, voltage;
    ATDCTL5 = 0x84;  // start ADC, channel 4, right justified
    while((ATDSTAT0&0x80)==0){}; // wait for SCF
    sample = ATDDR0;
    voltage = (5000*sample/1023); // 0 to 5000
    return voltage;
}
```

**Function: ADC_In4Bad**

0000 c684 [1]     LDAB  #132
0002 5b00 [2]     STAB  _ATDCTL45:1
0004 4f0080fc [4]  BRCLR  _ATDSTAT0,#128,*+0 ;abs = 0004
0008 dd00 [3]     LDY  _ATDDR0
000a cc1388 [2]     LDD  #5000
000d 13 [3]     EMUL
000e ce03ff [2]     LDX  #1023
0011 34 [2]     PSHX
0012 ce0000 [2]     LDX  #0
0015 34 [2]     PSHX
0016 160000 [4]  JSR  _LDIVU
0019 3d [5]     RTS

****OVERFLOW****
(5) Question 2. Two inputs mean twice the current is needed.

| VOH ≥ VIH | Ioh ≥ 2 Iih |
| VOH ≤ VIH | Iol ≥ 2 Iil |

(5) Question 3. \( y = 0.78125x0 + 0.78125x2 - 0.5625y2 = \frac{25(x0+x2)-18y2}{32} \)

25*510 <32767, 16-bit math will not overflow

short x0, x2, y2, y;
void calc(void)
{
    y = (25*(x0+x2)-18*y2)/32;
}

Function: calc

0000 fc0000       [3]     LDD   x0
0003 f30000       [3]     ADDD  x2
0006 b746         [1]     TFR   D,Y
0008 c619         [1]     LDAB  #25
000a 87           [1]     CLRA
000b 13           [3]     EMUL
000c 3b           [2]     PSHD
000d fc0000       [3]     LDD   y2
0010 cd0012       [2]     LDY   #18
0013 13           [3]     EMUL
0014 3b           [2]     PSHD
0015 ec82         [3]     LDD   2,SP
0017 a3b3         [3]     SUBD  4,SP+
0019 ce0020       [2]     LDX   #32
001c 1815         [12]    IDIVS
001e 7e0000       [3]     STX   y
0021 3d           [5]     RTS

(5) Question 4. Write a C function that outputs (transmits) one byte using the SPI port.

void SPI_Out(unsigned char data)
{
    while((SPISR&0x20)==0);  // wait for SPTEF
    SPIDR = data;            // start transmission
    while((SPISR&0x80)==0);  // wait for SPIF
    data = SPIDR;            // clear SPIF
    PTT &= ~0x04;            // PT2=0
    PTT |= 0x04;             // PT2=1
}

(5) Question 5. Review your EE411 before going on the interview trail. \( R*C= 10 \text{ ns} \). There are 10 time constants within this 100ns window, so the time constant is small compared to the pulse width. (This is good) From 0 to 100ns, \( V_{out} = 5-5e^{-t/10}\text{ns} = 5-5e^{-t/10}\text{ns} \)

At 10ns, \( V_{out} = 5-5e^{-100\text{ns}/10\text{ns}} = 4.9998\text{V} \)

\[ V_{in} \quad \begin{array}{c}
V_{out} \quad \text{time}
\end{array} \]
(5) **Question 6.** First discuss this with a coworker or your boss, because most likely your observation is either wrong, or based on a faulty assumption. Most companies have an explicit protocol for handling ethics in general and error reporting in specific.

(5) **Question 7.** Since the software is generating data, and software execution speed depends on the CPU, this system is **A** The system is CPU bound

(5) **Question 8.** Since the input channel is generating data, and the I/O speed depends on the I/O bandwidth this system is **C** The system is I/O bound

(5) **Problem 9.** A) This is a perfectly appropriate usage of Count, because there are two permanently allocated variables with private scope, such that each variable counts the number of interrupts for each ISR.

(20) **Question 10.** A 6811 system will be designed to provide for a latched input port.

Part a) Show the design of the address decoder for the latched input port.

<table>
<thead>
<tr>
<th>Address</th>
<th>Decoder</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000$ to $01FF$</td>
<td>$0000,000X,XXXX,XXXX$</td>
<td>A15 or A13</td>
</tr>
<tr>
<td>$1000$ to $103F$</td>
<td>$0001,0000,00XX,XXXX$</td>
<td>A15 or A13 or A12</td>
</tr>
<tr>
<td>$2000$</td>
<td>$0010,0000,0000,0000$</td>
<td>A15</td>
</tr>
<tr>
<td>$A000$</td>
<td>$1010,0000,0000,0000$</td>
<td></td>
</tr>
<tr>
<td>$B600$ to $B7FF$</td>
<td>$1011,011X,XXXX,XXXX$</td>
<td>A12</td>
</tr>
<tr>
<td>$BF00$ to $BFFF$</td>
<td>$1011,1111,XXXX,XXXX$</td>
<td>A12</td>
</tr>
<tr>
<td>$C000$ to $FFFF$</td>
<td>$11XX,XXXX,XXXX,XXXX$</td>
<td>A14</td>
</tr>
</tbody>
</table>

We must choose A15, A14 and A12 (we could add additional lines, but if we did the Kmap would eliminate them)

<table>
<thead>
<tr>
<th>A14, A12</th>
<th>0, 0</th>
<th>0, 1</th>
<th>1, 1</th>
<th>1, 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0, 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1, 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1, 0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Part b) You want RDA to overlap RDR and you do not want to drive the address bus during the first half of the bus cycle, so D Synchronized, negative logic.

Part c) Drive data onto the data bus (/OE=0) only if Select is true, it is a read cycle (R/W=1), and during the second half of the bus cycle (E=1).

<table>
<thead>
<tr>
<th>Select</th>
<th>R/W</th>
<th>E</th>
<th>/OE</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Wrong address</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Wrong address</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Wrong address</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Wrong address</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write cycles ignored</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Write cycles ignored</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>First half</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Drive data</td>
</tr>
</tbody>
</table>

/\OE = A15 \cdot A14 \cdot A12 \cdot R/W \cdot E

Part d) The 74HC573 address latch is not needed
Part e) This is a synchronized interface, meaning the rise and fall of /OE is controlled by the E clock. There is only one gate delay from E to /OE (the 74HC04 and 74HC11 have no effect on the timing of /OE). /OE falls at 250+\([5,15\text{ns}]\), and rises at 500+\([5,15\text{ns}]\). The worst-case is the later for the fall and the earlier for the rise, so

Read Data Available = \((250+\[5,15\text{ns}] + 25, 500+\[5,15\text{ns}] + 10) = (290, 515)\)

(10) Question 11. Add ground gain so sum of gains is 1

\[V_{\text{out}} = 100*V_{\text{in}} - 2*V_{\text{ref}} - 97*V_{g}\]

Choose \(R_f\) (194 \(\text{k}\)\(\Omega\)) a common multiple of gains, 100, 2, 97

Choose input resistors to get the desired gain

- \(R_f / R_{\text{in}} = 100\) thus \(R_{\text{in}} = 1.94 \text{k}\(\Omega\),
- \(R_f / R_{\text{ref}} = 2\) thus \(R_{\text{ref}} = 97 \text{k}\(\Omega\),
- \(R_f / R_g = 97\) thus \(R_g = 2 \text{k}\(\Omega\).

Build circuit, \(V_{\text{in}}\) is positive, the other two are negative.

(5) Question 12. Because this is a single supply system, we need a LPF with a positive gain. Because the signals of interest are 0 to 10 kHz, we will set the cutoff at 10 kHz. The two-pole Butterworth LPF has gain=1 and is very inexpensive.

1) select the cutoff frequency, \(f_c\)
2) divide the two capacitors by \(2\pi f_c\) (let \(C_{1A}, C_{2A}\) be the new capacitor values)
   \[C_{1A} = 141.4 \mu\text{F}/2\pi f_c = 141.4 \mu\text{F}/2\pi 10\text{k} = 0.00225 \mu\text{F}\]
   \[C_{2A} = 70.7 \mu\text{F}/2\pi f_c = 70.7 \mu\text{F}/2\pi 10\text{k} = 0.001125 \mu\text{F}\]
3) locate two standard value capacitors (with the 2/1 ratio) with the same order of magnitude as the desired values
   let \(C_{1B}, C_{2B}\) be these standard value capacitors, let \(x\) be this convenience factor
   \[C_{1B} = C_{1A}/x = 0.002 \mu\text{F}\]
   \[C_{2B} = C_{2A}/x = 0.001 \mu\text{F}\]
4) adjust the resistors to maintain the cutoff frequency
   \[R = 10\text{k}\Omega x = 11.252 \text{k}\Omega\]
5) adjust the cutoff frequency to 10.2 kHz, to use a standard resistor values, 11 \(\text{k}\Omega\)
(10) **Question 13.** Because the motor is 5V, we use the +5 V supply. There are lots of drivers that can handle 5 amps, but the driver in the book with a current rating above 5 amps is the IRF540. A 1kΩ resistor can be added to limit current into the MOSFET when the transistor switches on or off. The voltage controlled MOSFET only requires gate current during transitions.

(10) **Question 14.** Create a delayed pulse output, triggered on the rise of PT0

**Part a)** Give the initialization code that sets PT0 as input, PM0 as output (initially zero).

```c
void Pulse_Init(void){
    asm sei           // make atomic
    DDRT &= ~0x01;  // PT0 input
    DDRM |= 0x01;   // PM0 output
    PTM &= ~0x01;  // PM0=0
    TSCR1 = 0x80;   // enable TCNT, 1us
    TSCR2 = 0x02;   // divide by 4 TCNT prescale, TOI disarm
    TCTL4 = TCTL4&0xFC+0x01;   // rising edge PT0
    TIE = 0x01;     // Arm only IC0
    TIOS |= 0x02;   // PT1 output compare
    TIOS &= ~0x01;  // PT0 input capture
    asm cli
}
```

**Part b)** Show the input capture 0 interrupt service routine. No backward jumps allowed.

```c
void interrupt 8 IC0Han(void){ // rising edge of PT0
    TC1 = TC0+T1;     // T1 us after rising edge of PT0
    TIE |= 0x02;      // arm OC1
    TFLG1 = 0x03;     // clear C1F and C0F
}
```

**Part c)** Show the output compare 1 interrupt service routine. No backward jumps allowed.

```c
void interrupt 9 OC1Han(void){
    if(PTM&0x01){     // PM0 is 1 on second interrupt
        PTM &= ~0x01;  // PM0=0
        TIE &~ ~0x02;  // disarm OC1
    } else{            // first interrupt
        PTM |= 0x01;   // PM0=1
        TC1 = TC1+T2;  // T2 us later
    }
    TFLG1 = 0x02;     // clear C1F
}
```