(10) Question 1. The first error component is due to the finite DAC resolution. The DAC resolution is the range (5V) divided by the precision ($2^n$, $\Delta = 5/2^n$). Therefore the maximum error (in V) at that time is $\Delta/2$ or $5/2^{n+1}$. The other error arises from the DAC output rate. If the desired signal has a slope of $m$, it will change by $m/f_s$ during the interval between periodic interrupts. The worse case occurs when $5/2^{n+1}$ and $m/f_s$ have the same sign, therefore

$$e_{\text{max}} = 5/2^{n+1} + m/f_s.$$ 

(30) Question 2. Design a pulse width measurement system with a range of 10ms to 50s and a resolution of 10ms. You may use any of the PORTT input capture, output compare features, but you MUST USE interrupt synchronization. Assume the digital signal is connected to PT0. You need not worry about pulse width too small, but you need to check for overflow, i.e., if the pulse-width is longer than 50s, then an error condition must be set.

(10) Part a) Design the user-level device driver prototypes. You will need to define initialization, and measurement functions. In particular, give the *.H header file, including documentation. This file contains the public functions and public data structures. This section will be graded on style.

```c
#define TOOBIG -1
#define WAIT 0
#define BUSY 1
#define DONE 2

void PulseMeasInit(void); // initialize, start measurements
short PulseMeasStatus(void); // 0 waiting for rise, +1 busy counting, +2 done, -1 too big
unsigned short PulseMeasResult(void); // return last measurement, 1 to 50000 ms, restart next
/* example usage
void main(void){ unsigned short data;
PulseMeasInit(); // start measurements
while(1){         // infinite loop
  if(PulseMeasStatus()==TOOBIG){
    OutString("error\n");
PulseMeasInit(); // restart measurements
  }
  if(PulseMeasStatus()==DONE){
    data=PulseMeasResult(); // get and continue measurements
    OutString("PW = "); OutUDec(data); OutString(" ms\n");
  }
  // other stuff
} */
```

(20) Part b) This software contains the implementation and is protected from the user.

```c
short theStatus; // 0 waiting for rise, +1 busy counting, +2 done, -1 too big
unsigned short cnt; // number of 10ms intervals in this pulse
unsigned short PW; // measured pulse width in 10 ms units
#define resolution 10000
#define interrupt_handler TC1handler()

void TC1handler(void){
  TFLG1= 0x02;         // Acknowledge
  TC1=TC1+resolution;  // every 10 ms
  if(theStatus==BUSY){ // currently measuring
    Cnt++;
    if(Cnt>50000U){ // current measurement
      PW=50000U;
      theStatus=TOOBIG;
    }
  }
}

#define interrupt_handler TC0handler()

void TC0handler(void){
  TFLG1=0x01;              // ack, clear C0F
  if(PORTT&0x01){          // rising interrupt
    if(theStatus==WAIT){ // currently waiting
      theStatus=BUSY; // now start counting
      Cnt=0;
    }
  }
  else{ // falling edge interrupt
```
Jonathan W. Valvano

```c
if (the_status == BUSY) // currently counting
    the_status = DONE; // now finished
PW = Cnt;
}

void PulseMeasInit(void) { // initialize, start measurements
    asm(" sei"); // make atomic
    TI0S |= 0x01; // enable OC1
    TSCR = 0x80; // enable
    TMSK2 = 0x33; // 1 us clock
    TFLG1 = 0x03; // Clear C1F, C0F
    TMSK1 = 0x03; // Arm OC1 and IC0
    TCTL4 |= 0x03; // both edges */
    TC1 = TCNT + resolution;
    the_status = WAIT; // first look for rising edge
    asm(" cli");
}

short PulseMeasStatus(void) { // 0 waiting for rise, +1 busy counting, +2 done, -1 too big
    return the_status;
}

unsigned short PulseMeasResult(void) { // 1 to 50000 ms
    the_status = WAIT; // first look for rising edge
    return PW;
}
```

(40) **Question 3.** In this problem, you will design a 6812-based ohmmeter.

(5) Part a) The measurement resolution is \(1000/256 = 3.9\ \Omega\).

(5) Part b) A sampling rate of 2 Hz satisfies the Nyquist Theorem. I choose 2 Hz to improve the LPF approach to removing 60 Hz noise.

(5) Part c) Since 60 Hz does not exist in the frequencies of interest, the best way in this system to eliminate 60 Hz noise pickup is to use an analog LPF. A 10 Hz LPF will pass 0 to 1 Hz, but reject 60 Hz and its harmonics.

(15) Part c) The first Op Amp creates the 1 mA constant current across R. The instrumentation amp provides a high input impedance gain=5 amplifier, and the LPF at 10 Hz prevents aliasing and removes 60 Hz.

1) select the cutoff frequency, \(f_c = 10\ \text{Hz}\)

2) divide the two capacitors by \(2\pi f_c\)

\[
C_{1A} = 141.4\mu F / 2\pi f_c = 141.4\mu F / (2\pi 10) = 2.25\ \mu F
\]

\[
C_{2A} = 70.7\mu F / 2\pi f_c = 70.7\mu F / (2\pi 10) = 1.125\ \mu F
\]

3) resistance scale to get R= 200 kΩ. let \(x = 20\)

\[
R = 10 \text{ k}Ω \cdot x
\]

\[
C_{1B} = C_{1A}/x = 2.25/20 \mu F = 0.112\ \mu F
\]

\[
C_{2B} = C_{2A}/x = 1.125/20 \mu F = 0.056\ \mu F
\]

\[
I = \frac{V_{in}}{R_1} = \frac{5V}{5k\Omega} = 1\ mA
\]

\[
5 = \frac{25k\Omega}{10k\Omega} \left[ 1 + \frac{2 \cdot 5k\Omega}{10k\Omega} \right]
\]

Jonathan W. Valvano
(5) Part e) Show the ritual that initializes the system.
(5) Part f) Show the interrupt service routine that samples the ADC and calculates resistance in $\Omega$.

```c
#define OC5 0x20
unsigned short R;   // resistance measurement in ohms
unsigned short cnt; // used to count interrupts to create 2 Hz sampling
void ritual(void){
    asm(" sei");      // make atomic
    TIOS|=OC5;      // enable OC5
    TSCR|=0x80;     // enable
    TMSK2=0x35;     // 4 us clock
    TMSK1|=OC5;     // Arm output compare 5
    cnt= 0;
    TFLG1=OC5;      // Initially clear OC5F
    TC5=TCNT+25000; // First one in 100 ms
    ATDCTL2 = 0x80; // Activate ADC
    ATDCTL5=0;      // Start A/D, channel 0
    asm(" cli");
}
#pragma interrupt_handler TC5handler()
void TC5handler(void){
    TFLG1=OC5;      // Ack interrupt
    TC5=TC5+25000; // Executed every 100 ms
    if((++cnt) == 5){
        cnt=0;               // start over
        R=(ADR0H*125+16)>>5; // 1000/256=125/32
        ATDCTL5=0;           // Start A/D for next time, channel 0
        asm(" cli");
    }
}
```

(20) Question 4. Consider the 8K RAM MC68HC812A4/60L64 interface presented in Section 9.7.2.

(10) Part a) For a read cycle, the terms that matter are $t_{AVQV}$ and $t_{E1LQV}$
- Address $60 + t_{AVQV} \leq 125 - 30$ so $t_{AVQV} \leq 35$
- $E1$ $60 + t_{E1LQV} \leq 125 - 30$ so $t_{E1LQV} \leq 35$
For a write cycle, the term that matters is the setup $t_{DVWH}$
- data setup $106 \leq 125+10-t_{DVWH}$ so $t_{DVWH} \leq 29$

(10) Part b) For a read cycle, again the terms that matter are $t_{AVQV}$ and $t_{E1LQV}$
- Address $60 + 150 \leq t_{1} - 30$ so $t_{1} \geq 240$
- $E1$ $60 + 150 \leq t_{1} - 30$ so $t_{1} \geq 240$
For a write cycle, the term that matters is the setup $t_{DVWH}$
- data setup $106 \leq t_{1}+10-60$ so $t_{1} \geq 156$
The read cycle is the worst case, so $t_{1} \geq 240$

Jonathan W. Valvano