Jonathan W. Valvano July 31, 2000, 2:30-3:45pm

(25) Question 1. Rewrite IIR digital filter using integer math.

y(n) = (10•x(n)+4•x(n-1)+2•x(n-2)+y(n-2))/16

```
unsigned char x[3],y[3]; // 8-bit unsigned numbers, 0 to 255
#define C5F  0x20
#pragma interrupt_handler TC5Handler()
void TC5Handler(void){
   TFLG1=C5F;         // ack interrupt
   TC5=TC5+8333;      // fs=240Hz
  // add code here to shift the MACQ
   y[2]=y[1];
   y[1]=y[0];
   x[2]=x[1];
   x[1]=x[0];
   x[0] = A2D(0);        // new 8-bit data, 0 to 255
// add code here to execute the filter
// 16 bit math, largest numerator = (10+4+2+1)*255 = 4335 < 32767
   y[0] = (10*x[0]+4*x[1]+2*x[2]+y[2])/16;
}
```

(30) Question 2. Consider a pressure data acquisition system.
(5) Part a) Make the analog noise less than the resolution of the ADC, which is 10/4096 = 2.4 mV.

(5) Part b) Pressure resolution is range divided by precision, which is 10/4096 = 0.0024 psi.

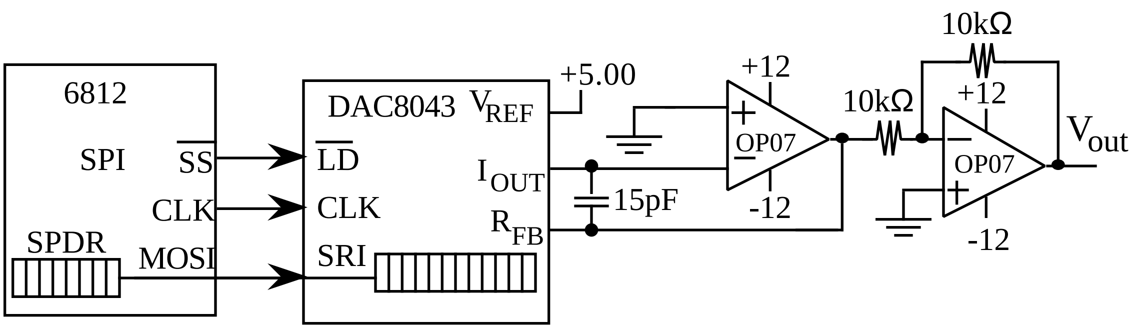
(5) Part c) According to Nyquist, the sampling rate should be greater than 50 Hz.

(5) Part d) In order to prevent loading, make the input impedance of the analog amplifier greater than 4096*5kΩ, which equals 20MΩ. For added safety, you might make it greater than 8192*5kΩ, which equals 41MΩ.

(5) Part e) The range of transducer output is ±5*50mV=±250mV, thus gain should be 5/0.25, which is 20.

(5) Part f) The best performance is achieved with the integrated instrumentation amp (e.g., AD620).

(35) Question 3. You will design a signal generator.
(10) Part a) I started with the 0 to -5V interface shown in Figure 7.44 on page 405, and Figure 11.76 on page 632, then added an analog inverter to make the range 0 to +5V.



(5) Part b) The output period is 25ms, so there are 40 samples/sec

```
// DAC value is 4095*v/5 = 819*v
const unsigned short wave[40]={
   1638, 1638, 1638, 1638, 2048, 2048, 1638, 1638, 1638, 4095,
   1638, 1638, 1638, 1638, 2457, 2457, 2457, 1638, 1638, 1638,
   1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638,
   1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638, 1638};
```

(10) Part c) Show the ritual that initializes the system. Once initialized, the waveform will be generated in the background via a periodic interrupt.

```
unsigned short I; // index into wave
void ritual(void) {
asm(" sei");     // make atomic
  DACInit();    // Program 7.19
  TIOS|=0x20;   // enable OC5
  TSCR|=0x80;   // enable
  TMSK2=0x32;   // 500 ns clock
  TMSK1|=0x20;  // Arm output compare 5
  I = 0;
  TFLG1=0x20;   // Initially clear C5F
  TC5=TCNT+10;  // First one in 20us
asm(" cli"); }
```

(10) Part d) Show the interrupting software that produces the waveform.

```
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
   TFLG1=0x20;         // ack C5F
   TC5=TC5+50000U;     // Executed every 25 ms
   if((++I)==40) I=0;
   DACout(wave[I]);}   // Program 7.20
```

(10) Question 4. Look up each assembly instruction. Very briefly state what it does and how it might be used in an embedded system. Don't worry about the details, just the general idea.

Part a) `emac` is a 16-bit by 16-bit multiply resulting in a 32-bit product (a16*b16), followed by a 32-bit addition.

n32=n32+a16*b16

It is used for digital filters and digital controllers.

Part b) `mem` performs a fuzzification step converting a crisp input into an input fuzzy membership set. It is used for fuzzy logic controllers.

Part c) `callf` performs a far call, similar to jsr except it saves the 24-bit return address. It is used for calling functions with extended addressing. The 6812 can handle up to 4Mbytes of extended addressing. See Chapter 9 for more details.

Part d) `rti` pulls all the registers (except SP itself) off the stack. It is used for returning from a hardware or software interrupt.

Part e) `etbl` performs a table interpolation. You create a table of x,y points, and use this instruction to perform a linear interpolation to find y given x.