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

(50) Question 1. In this problem we design a digital thermometer using period measurement.

(5) Part a) The precision is 100 alternatives, so the desired resolution is (1514-1324)/100=1.9µs. Therefore I will use 1 µs resolution. From the table it looks like 2µs should work, but it is not quite enough.

(5) Part b) First I convert I to the proper units (P is already in µs) by multiplying the RHS by 10,

$$T=1748.8-0.528 \bullet P$$

Then, I convert to integer arithmetic (0.528=66/125). I must divide last to get the correct answer, but I subtract first to reduce the amplitude of the intermediate result, which eliminates overflow.

$$T=950+((1514-p)*66)/125$$

As a test, I desk check the equation for high, middle, and low temperatures. Considering overflow I carefully watch the intermediate calculations for values above 32767.

```
high  T=950+((1514-1324)*66)/125=950+(190*66)/125=950+(12540)/125=950+100=1050
mid   T=950+((1514-1419)*66)/125=950+( 95*66)/125=950+( 6270)/125=950+ 50=1000
low   T=950+((1514-1514)*66)/125=950+(  0*66)/125=950+(    0)/125=950+  0= 950
```

(15) Part c) I write the ritual subroutine that initializes the interface.

```
#define C6  0x40
#define C5  0x20
unsigned short First; // time of first edge
unsigned short p;     // period in usec
void Ritual(void){
    asm(" sei");    // make atomic
    TIOS &= ~C6;    // PT6 input capture
    TIOS |=OC5;     // PT5 is output compare
    DDRT &= ~C6;    // PT6 is input
    TSCR = 0x80;    // enable TCNT
    TMSK2= 0x33;    // 1us clock
    TCTL3 = (TCTL3&0xCF)|0x10; // bits 5:4=0,1 rising
    First = TCNT;   // first will be wrong
    bOK=0;          // set on subsequent
    TFLG1 = C6;     // Clear C6F
    TMSK1 |= C6+C5; // Arm IC6 and OC5
    TC5=TCNT+3000;  // timeout after 3 ms
    asm(" cli");}
```

(25) Part d) I write the interrupt service routines that measure temperature.

```
#pragma interrupt_handler TC6handler()
void TC6handler(void){ // called on a rising edge of PT6
    Period=TC6-First;  // units are usec
    First=TC6;         // Setup for next
    TC5=TCNT+3000;     // timeout after 3 ms
    TFLG1=C6+C5;       // ack by clearing C6F
    if(period>=1324)&&(period<=1514)){
      T=950+((1514-period)*66)/125; // units 0.1F
      bOK=1;}
    else
      bOK=0;            // out of range
}
#pragma interrupt_handler TC5handler()
void TC5handler(void){
    TFLG1=C5;        // ack OC5F
    TC5=TC5+3000;    // Executed every 1 ms
    bOK=0; }
```

(10) Question 2. In this problem consider two C functions.

Part a) These two functions are friendly because they do not undo each other's action. In particular, it does not matter in what order they are executed.

Part b) It depends whether the compiler produces atomic code or not. If the compiler generates the following nonatomic code, then they have critical sections between the read and write DDRH.

```
_Ritual0 ldab DDRH        _Ritual1 ldab DDRH
         orab #$01                 orab #$01
         stab DDRH                 stab DDRH
         rts                       rts
```

If the compiler generates the following atomic code, then they have no critical sections.

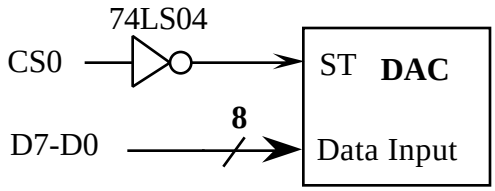
```
_Ritual0 bset DDRH,#$01   _Ritual1 bset DDRH,#$02
         rts                       rts
```

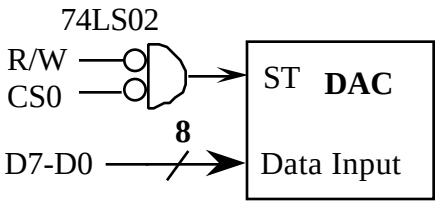
(10) Question 3. Consider the LED interface to a 6812.
Part a) The largest possible LED current is determined by the IOH of the 6812, which is 0.8 mA.
Part b) R = (5-2V)/500μA=6000Ω

(40) Question 4. In this problem I will interface a DAC as an output port to the MC68HC812A4.
(5) Part a) I use expanded narrow because it has an 8-bit wide data path.

(5) Part b) I choose synchronized because the timing of the edge matters, and I choose positive logic because I want the falling edge to occur when the data is available.

(10) Part c) There are two good answers. The one on the right only activates for write cycles.





(10) Part d) Let t_1 be the E clock period (125, 250, 375 or 500ns)

WDA = Write Data Available = (106, t_1 + 20)

From the DAC timing

WDR = Write Data Required = (↓ST-80, ↓ST)

Since ST will be generated from the CS0, ↓ST=↑CS0+10 (the 10ns is due to one gate delay). So,

WDR = (↑CS0+10-50, ↑CS0+10) = (t_1+10+10-80, t_1+10+10) = (t_1-60, t_1+20)

To make WDA overlap WDR, we must make have one cycle stretch

$106 \le t_1-60$, or $166 \le t_1$

(10) Part e) The combined **write cycle** timing diagram is important to verify proper timing.