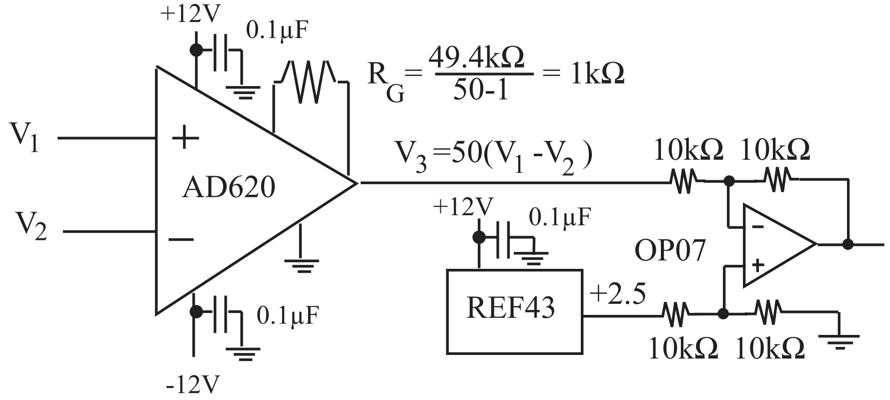
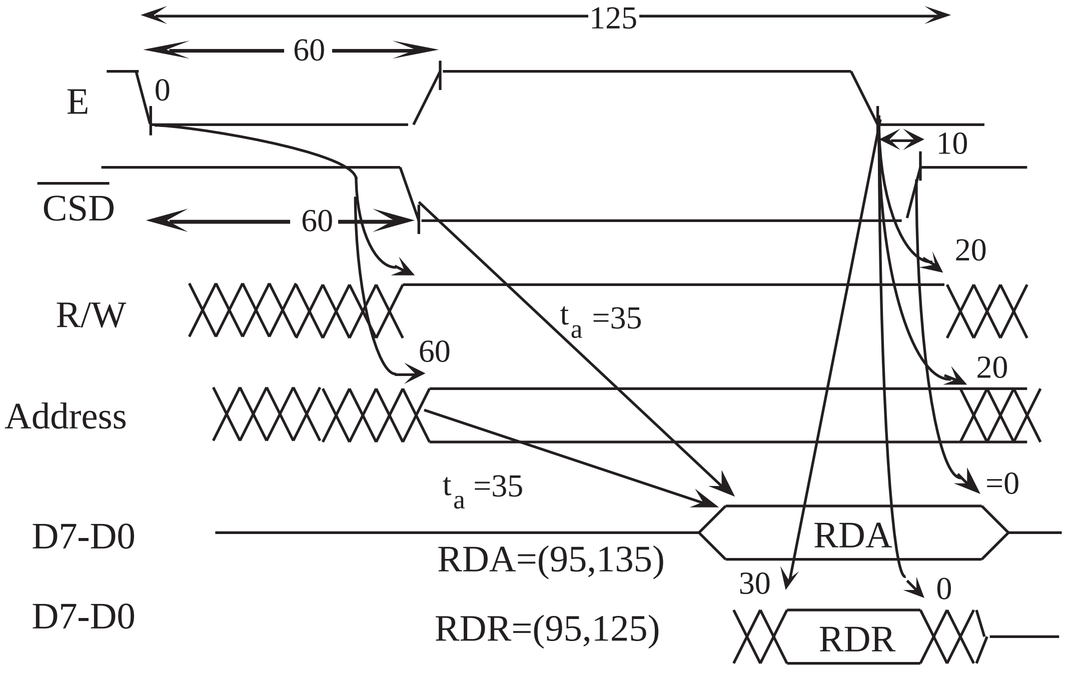
Jonathan W. Valvano

**(20) Question 1.** Here is one possible analog circuit that satisfies the specifications:



**(20) Question 2**. Consider a 128K by 8 bit static RAM interface.

Part a) Draw a combined read timing diagram assuming no cycle stretching.



Part b) If $\mathbf{t_a}$ is 35 ns, then RDA just overlaps RDR.

**(20) Question 3.** Conversions from real variables to fixed-point versions. Overflow will be handled by promotion to 32-bits, performing the controller in 32-bit math, then performing a ceiling/floor operation before demotion.

xstar = 100•X*
x(n) = 100•X(t)
u(n) = 1000•V(t)
e(n) = xstar - x(n)

proportional term

| | |
|---|---|
| $Vp(t) = 0.0512 \bullet e(t)$ | original proportional term |
| $up(n) = 1000 \bullet 0.0512 \bullet e(t)$ | convert Vp to up |
| $up(n) = 1000 \bullet 0.0512 \bullet e(n)/100$ | convert e(t) to e(n) |
| $up(n) = (512 \bullet e(n))/1000$ | make it fixed-point |
| $up(n) = (64 \bullet e(n))/125$ | simplify |

integral term

| | |
|---|---|
| $Vi(t) = 0.0408 \bullet \int e(\tau)d\tau$ | original integral term |
| $Vi(t) = 0.0408 \bullet \sum e(\tau)\Delta t$ | approximate integration with sum |
| $ui(n) = 1000 \bullet 0.0408 \bullet \sum e(\tau)\Delta t$ | convert Vi to ui |
| $ui(n) = 1000 \bullet 0.0408 \bullet \sum e(n)\Delta t/100$ | convert e(t) to e(n) |
| $ui(n) = 0.0408 \bullet \sum e(n)$ | simplify, $\Delta t = 0.1s$ |
| $ui(n) = ui(n-1)+0.0408 \bullet e(n)$ | simplify sum |
| $ui(n) = ui(n-1)+408 \bullet e(n)/10000$ | make it fixed-point |
| $ui(n) = ui(n-1)+51 \bullet e(n)/1250$ | simplify |

put together

$$u(n) = up(n) + ui(n)$$

**(10) Question 4**. If the FIFO is big enough, then the system will run continuously if the sum of the average execution times is less than $1/\mathbf{f_s}$. In particular, the FIFO will not overflow. The system will be real-time if the main program runs with interrupts enabled, and the other ISRs have short and bounded execution times. So

$1/\mathbf{f_s} >$ `Adin+Fifo_Put+Fifo_Get+Process`=(25+15+20+1000) = 1060 µsec

so

$\mathbf{f_s} < 943$ Hz

**(10) Question 5.** First, write $15BCD in binary 0001,0101,1011,1100,1101. The offset is the bottom 14 bits 01,1011,1100,1101 = $1BCD. The memory address is $8000+offset =$9BCD. The program page number is the rest = 000101 = $05

```
PPAGE = 0x05;
data = *((char *)(0x9BCD));
```

Part b) Again, write $15BCD in binary 0001,0101,1011,1100,1101. The offset is the bottom 12 bits 1011,1100,1101 = $0BCD. The memory address is $7000+offset =$7BCD. The data page number is the rest = 00010101 = $15

```
DPAGE = 0x15;
data = *((char *)(0x7BCD));
```

Part c) The two have separate windows. The data page window is $7000-$7FFF and program page window is $8000-$BFFF. The RAM uses CSD and the ROM uses CSP0. So when 0x9BCD is accessed CSP0 is active. When 0x7BCD is accessed CSD is active.

**(20) Question 6.** Develop an interrupt-based square-wave generator.

Part a) The header file has prototypes for public functions.

```
void Square_Start(unsigned short frequency); // units in Hz
// works from 1 to 10000 Hz
```

Part b) The implementation file has private variables and implementations.

```
unsigned short rate;
void Square_Start(unsigned short frequency){
long count;    // number of 125 cycles per toggle
  if((frequency>10000)||(frequency==0))
    return;
asm(" sei");         // make atomic
  TIOS |= 0x40;      // enable OC6
  DDRT |= 0x40;      // PT6 is output
  TSCR |= 0x80;      // enable
  TCTL1 = (TCTL1&0xCF)|0x10; // PT6 toggle (or TCTL1 = 0x10)
  count = 4000000L/frequency;
  TMSK2 = 0x30;      // start at 8 MHz
  while(count>65535){
    count = count>>1;  // half as many counts
    TMSK2++;           // twice the period
  }
  TMSK1 |= 0x40;     // Arm output compare 6
  rate = count;
  TFLG1 = 0x40;      // Initially clear C6F
  TC6 = TCNT+10;     // First right away
asm(" cli");
}

#pragma interrupt_handler TC6handler()
void TC6handler(void){
  if(--count == 0){
    PORTT ^= 0x40;     // toggle output
    count = maxCount;
  }
  TFLG1 = 0x40;      // ack C6F
  TC6 = TC6+800;     // Executed every 100us
}
#pragma abs_address:ffe2
void (*OCinterrupt_vector[])() = {
  TC6handler    /* ffe2 TC6 */
}
#pragma end_abs_address
```