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

\[ R_g = \frac{49.4 \text{k}\Omega}{50-1} = 1 \text{k}\Omega \]

\[ V_3 = 50(V_1 - V_2) \]

(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 \( t_a \) 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.
\[
\begin{align*}
\text{xstar} &= 100 \cdot X^* \\
x(n) &= 100 \cdot X(t) \\
u(n) &= 1000 \cdot V(t) \\
e(n) &= x_{\text{star}} - x(n)
\end{align*}
\]
proportional term

\[ V_p(t) = 0.0512 \cdot e(t) \]  
original proportional term

\[ u_p(n) = 1000 \cdot 0.0512 \cdot e(t) \]  
convert \( V_p \) to \( u_p \)

\[ u_p(n) = 1000 \cdot 0.0512 \cdot e(n)/100 \]  
convert \( e(t) \) to \( e(n) \)

\[ u_p(n) = (512 \cdot e(n))/125 \]  
make it fixed-point

\[ u_p(n) = (64 \cdot e(n))/125 \]  
simplify

integral term

\[ V_i(t) = 0.0408 \cdot \int e(\tau) \, d\tau \]  
original integral term

\[ V_i(t) = 0.0408 \cdot \sum e(\tau) \Delta t \]  
approximate integration with sum

\[ u_i(n) = 1000 \cdot 0.0408 \cdot \sum e(\tau) \Delta t \]  
convert \( V_i \) to \( u_i \)

\[ u_i(n) = 1000 \cdot 0.0408 \cdot \sum e(n) \Delta t/100 \]  
convert \( e(t) \) to \( e(n) \)

\[ u_i(n) = 0.0408 \cdot \sum e(n) \]  
simplify, \( \Delta t = 0.1s \)

\[ u_i(n) = u_i(n-1)+0.0408 \cdot e(n) \]  
simplify sum

\[ u_i(n) = u_i(n-1)+408 \cdot e(n)/10000 \]  
make it fixed-point

\[ u_i(n) = u_i(n-1)+51 \cdot e(n)/1250 \]  
simplify

put together

\[ u(n) = u_p(n) + u_i(n) \]

(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 \( 1/f_s \). 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

\[ 1/f_s > Adin+Fifo_Put+Fifo_Get+Process=(25+15+20+1000) = 1060 \, \mu\text{sec} \]

so

\[ f_s < 943 \, \text{Hz} \]

(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

\[
\begin{align*}
PPAGE &= \text{0x05}; \\
data &= *((\text{char} \,*\,)\,(\text{0x9BCD}));
\end{align*}
\]

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

\[
\begin{align*}
DPAGE &= \text{0x15}; \\
data &= *((\text{char} \,*\,)\,(\text{0x7BCD}));
\end{align*}
\]

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

```c
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.

```c
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;

    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");
}
```

```c
#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
}
```

```c
#pragma abs_address:ffe2

void (*OCinterrupt_vector[])(void) = { TC6handler /* ffe2 TC6 */}
```

```c
#pragma end_abs_address
```