Embedded Systems (CS 503/591C)
Homework 4 Solutions

December 11, 2003

Question 1: Op-Amps

1. What are three characteristics of an ideal operational amplifier?

   (a) *Infinite input impedance (the inputs do not draw current)*

   (b) *Zero output impedance (the output is capable of generating infinite current)*

   (c) *Infinite open-loop voltage gain*

   (d) *Output voltage is zero when the inputs are the same*

   (e) *The output can respond to changes in the inputs infinitely fast*

2. How does an actual op-amp differ from each of these ideal characteristics?

   (a) *Some small input impedance*

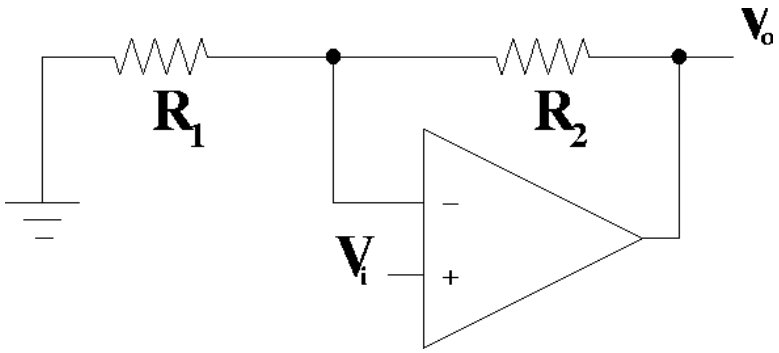
   (b) *Output current is limited*

   (c) *Finite voltage gain and output voltages are limited to the supply voltage range*

   (d) *Some small non-zero difference in voltage input results in a zero voltage output*

   (e) *Some delay in response to changes in inputs*

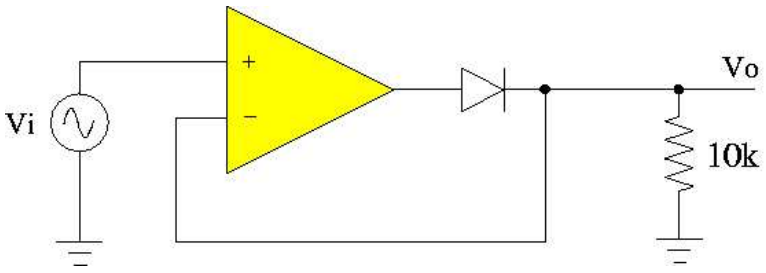
3. Consider the following circuit:



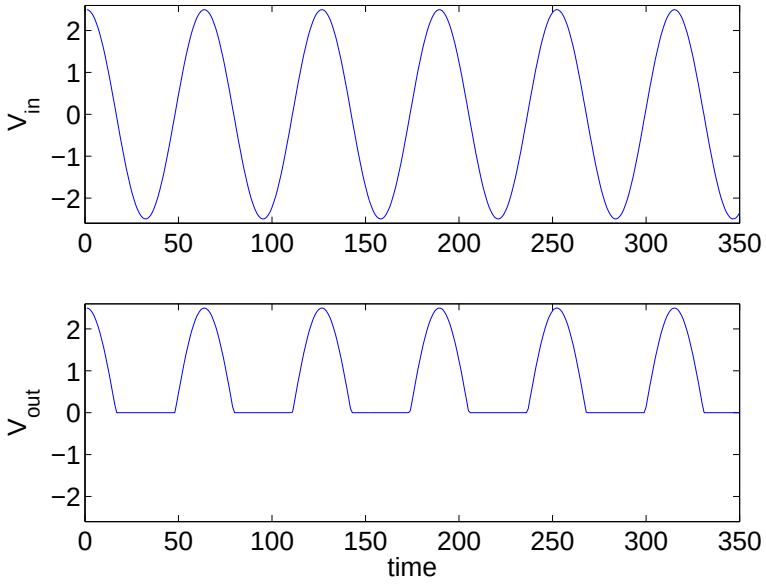
If $R1 = 10k?$, what is the value of $R2$ required to produce a circuit with gain of 25?

$(25 - 1) * 10K\Omega = 240K\Omega$

4. Consider the following circuit:

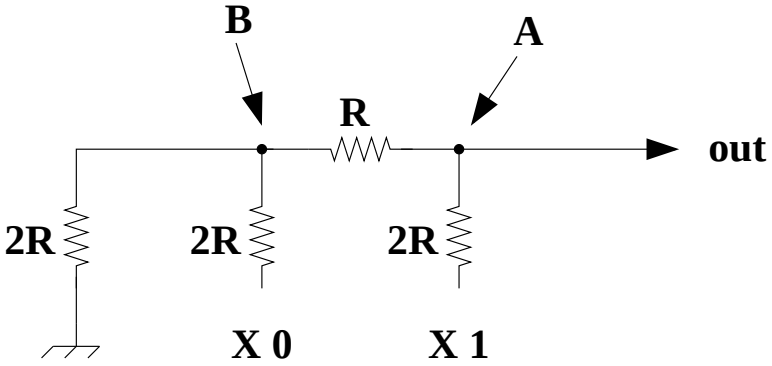


Draw the output V_o for a sine wave with a $2.5V$ peak and a $-2.5V$ trough at the input V_i



Question 2: ADC and DAC

1. Consider the following 2-bit digital-to-analog converter:



For the four possible digital inputs (to X_0 and X_1), give the output voltage.

We know that:

$\frac{V_B - V_A}{R} + \frac{V_{X1} - V_A}{2R} = 0$, *and*

$\frac{-V_B}{2R} + \frac{V_{X0} - V_B}{2R} = \frac{V_B - V_A}{R}$.

Note that the R's cancel out. Therefore:

$V_B = \frac{3V_A - V_{X1}}{2}$, *and*

$V_A = 2V_B - \frac{V_{X0}}{2}$.

Solving for V_A*:*

$V_A = \frac{V_{X1}}{2} + \frac{V_{X0}}{4}$.

Therefore:

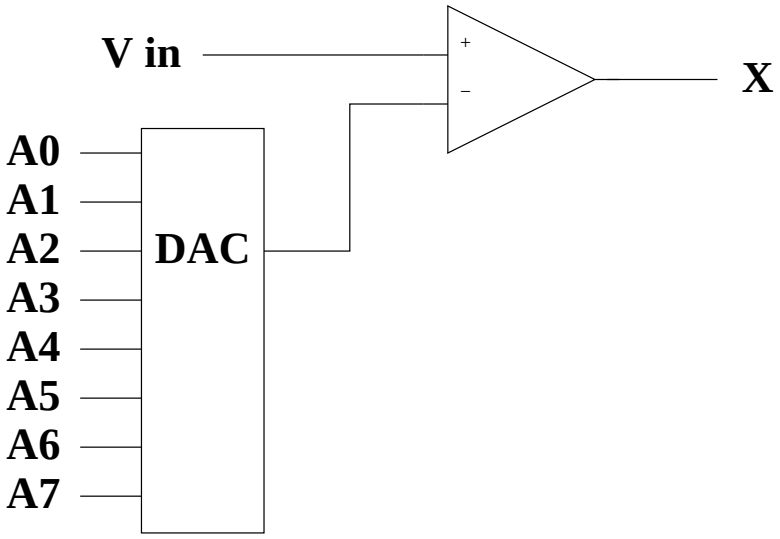
$V_A(00) = 0V$,

$V_A(01) = 1.25V$,

$V_A(10) = 2.5V$, *and*

$V_A(11) = 3.75V$.

2. Consider the following successive approximation analog-to-digital conversion circuit:



V_{in} is the input voltage; $A_0 ... A_7$ are the digital outputs from your PIC (A_0 is the LSB); and X is a digital input to your PIC.

Give a successive approximation algorithm in pseudo-code.

We will assume that the power supply to the Op-Amp is such that if $Neg <= Pos$ *then* $X = 0$ *and otherwise* $X = 1$.

```
// Initialize A
for(i=0; i < 8; ++i) {
   A[i] = 0;
}

// Check each bit, starting with the highest order bit
for(i=7; i <= 0; --i) {
   A[i] = 1;   // Try setting the bit
   usleep(1);  // Let the Op-Amp settle
   if(X) {
     A[i] = 0; // Too high - reset
   }
}
// Digital representation is now encoded in the A array
```

Question 3: Asynchronous Serial Protocols

1. Explain in brief how asynchronous serial protocols differ from synchronous ones.

   In asynchronous serial transmission, there is no shared clock. Therefore, before data can be sent, the receiver's clock must be synchronized with that of the transmitter. This is typically done by first sending a known sequence of bits (in the simplest case, this is a single bit). The proper phase of the receiver clock can be inferred from the bit transitions for these "start bits".

   We also often use a known sequence of "stop bits" that are used to confirm that the data is framed properly.

   Note that flow control and addressing can be done in either synchronous or asynchronous protocols (so these are not correct answers).

2. Give a pseudo-code implementation of read_bit() from the lecture. You may assume that a timer ISR is incrementing a globally-visible counter.

   We will assume that an ISR will increment a global variable *clock* and that the bit starts at $clock = 0$ and ends at $clock = 255$

```
int read_bit() {
   // We assume that this routine is called shortly
   //  after the start of a bit
   unsigned long count_high = 0;
   unsigned long count_low = 0;

   while(clock < 255) {
     if(read_sensor()) {
       count_high++;
     }else{
       count_low++;
  };

  // There are many different decisions one could make
  //   here...
  return(count_high > count_low);
}
```

3. Give a (simple) pseudo-code implementation of wait_for_start_bit() from the lecture.

   We are looking to be somewhat robust to noise here...

```
void wait_for_start_bit() {
   short int flag = 1;
   unsigned long count_high;
   unsigned long count_low;

   //  Loop until we have a valid start bit
   while(flag) {
    // Wait for line to go low
    while(read_sensor()){};

    clock = 0;  // Start counting
    count_high = 0;
    count_low = 1;

    // Wait for the duration of a bit
    //   But - bail if we see too many highs
    while(clock < 255 && count_low > count_high) {
      if(read_sensor()) {
        count_high++;
      }else{
        count_low++;
      };
    }
    if(clock == 255) {
      // We have a valid start bit - drop out of the function
      flag = 0;
    };
  };
};
```