Build a Resistive Ladder Network

Digital to analog conversion (D/A conversion) is, for the most part, the reverse of A/D conversion. With A/D conversion, we started with a continuous voltage range at the converter's input. The A/D converter rounded to the nearest voltage step and sent a binary output indicating which step it measured.

D/A conversion starts with a binary number as the input, and the output is a voltage step. While the A/D process starts with an analog input and ends with a binary output, the D/A process starts with a binary input and ends with a voltage step for an output. It's not a true analog value that varies continuously, it's a discrete voltage that varies in steps.

The term resolution was introduced at the end of Experiment #3. Since a D/A converter's output always going gets rounded to a voltage step (a discrete voltage value) it's important to pick the right resolution for your D/A converter. Remember that with higher resolution comes higher precision, but it typically comes at the price of greater expense, more memory, and more processing steps.

The number of voltage levels a D/A converter can produce is given by how many counting numbers you can get from the number of binary bits in the resolution. We can use the combinations equation again to figure this out.

\[
\text{combinations} = 2^{\text{bits}}
\]

The D/A converter we will use in this experiment has 4-bit resolution, so the number of output voltage levels for the converter will be:

\[
\text{combinations} = 2^{4} = 16
\]

In Experiment #3, we used an integrated circuit, which performed the A/D conversion. In this experiment, we'll build a D/A converter using resistors. It's called a resistive ladder network, and adding or removing resistors can be done to change the resolution of the converter. With a resistive ladder network, if you start with a 4-bit converter, and you want to increase the resolution by 1-bit, all it takes is two extra resistors added to the network.
In this experiment, we will build a resistive ladder network and program the BASIC Stamp to make the network do D/A conversion. PBASIC will be used to program the BASIC Stamp to send the resistive ladder network sets of binary voltage levels. These sets of binary voltages are converted by the resistive ladder network to discrete output voltages.

The DVM from Experiment #3 will be used to measure the converter's output voltages. Measuring all the D/A converter's output levels is called a voltage sweep. We will use PBASIC to automate our DVM to perform the entire voltage sweep. That way, the D/A converter's output voltages can be measured without manually repeating each measurement.

Gather these parts from your parts kit and let's get started:

- (6) 2 kΩ resistors
- (3) 1 kΩ resistors
- (1) ADC0831 A/D Converter
- (1) Red LED
- (1) 270 Ω resistor

The resistive ladder network for this experiment is shown in Figure 4.1. The name comes from the fact that the resistor network in the schematic resembles a ladder. It's definitely an inexpensive alternative compared to an integrated circuit digital to analog converter (D/A converter or DAC). A few resistors are a fraction of the cost of an integrated circuit.
This being the case, why doesn't everybody use resistive ladder networks for A/D and D/A conversion? The resistive ladder network is also used in many A/D and D/A integrated circuits, such as the ADC0831. The resistors used in integrated circuits are microscopic implants on the surface of a silicon wafer. One advantage of IC converters is that they have a high degree of accuracy. Another advantage an IC has is extra built in circuitry similar to the voltage follower we built in Experiment #1.
Build the circuit as shown in Figure 4.2. Pay careful attention to the values of the resistors as well as what each resistor connects to. If your circuit from Experiment #3 is still intact, just remove the potentiometer and build the resistive ladder network near the sockets for pins P4 through P7. The input lead to the DVM, which was connected to the wiper terminal on the potentiometer, should now be connected to the output of the D/A converter. Be sure to pay close attention so that the resistors don't touch each other except where they're supposed to on the breadboard nodes.

Figure 4.2: Schematic. The DC voltmeter from Experiment #3 is connected to the output of the resistive ladder network D/A converter. The DC voltmeter is used to measure the D/A converter's voltage outputs.

It's best to try to wire the circuit directly from the schematic in Figure 4.2. This will be a complex circuit to fit on the breadboard. The placement of the parts and other wiring specifics are usually left up to the person building the circuit, and you're now on your own!
Not only can we use the DVM to measure the output voltage from the D/A converter, we can automate the testing process to measure all 16 D/A converter’s output voltage levels. This might not be a big deal for just 16 measurements, but just imagine trying to test all 4096 voltage steps on a 12-bit converter!

With some relatively simple additions to the code from Experiment #3, which was saved as file P3_1R3.bs2, we can control both devices. PBASIC can be used to instruct the BASIC Stamp to send an output signal to the D/A converter. The code for this will be added to the final version of Program Listing 3.1. This way we can use our DVM from this experiment to measure the D/A converter’s output.

Program Listing 4.1 is shown below. It's the final revision of Program Listing 3.1 with a subroutine labeled \texttt{DAC:} added to send binary voltages to the D/A converter. There are also a few additional changes that are pointed out using comments such as ‘ which means add this line and ‘ which shows the lines that have been changed.

If you saved the program listing from Experiment #3, add and modify the code for this experiment, and save the program under the name P4_1R1.bs2. If you do not have the code from Experiment #3, enter the entire program listing below using the BASIC Stamp editor and make sure to save it for future use. When the circuit is built and the code is entered and saved, run Program Listing 4.1, and let’s see how it works.

```
'Program Listing 4.1
'Digital Voltmeter (DVM)
'D/A Converter Added

'Declarations
adcbits var byte
v var byte
R var byte
v2 var byte
v3 var byte
n var nib

CS con 0
CLK con 1
DO con 2

'start display
debug cls

'Main routine
main:
```
gosub DAC
gosub ADCDATA
gosub CALC_VOLTS
gosub DISPLAY
goto main

DAC:
  n = 11
  output 7
  output 6
  output 5
  output 4

  out7=n.bit3
  out6=n.bit2
  out5=n.bit1
  out4=n.bit0

  return

ADCDATA:
  high CS
  low CS
  low CLK
  pulsout CLK,210
  shiftin D0,CLK,msbpost,[adcbits\8]
  return

CALC_VOLTS:
  v=5*adcbits/255
  R=5*adcbits//255
  v2=100*R/255
  v3=100*R//255
  v3=10*v3/255
  if v3<5 then skip_a_line
  v2=v2+1
  skip_a_line:
  if v2<100 then skip_out
  v=v+1
  v2=0
  skip_out:
  return
DISPLAY:
debug home, cr, cr, "Decimal value to DAC: ", dec2 n
debug cr, cr, "Binary value to DAC: ", bin4 n
display CR, CR, "DVM Reading: ", dec1 v, ".", dec2 v2, " Volts"
return

The Output

Given perfect resistor values, the output would be 3.00 volts. The resistors used in this sample have a 10% tolerance. This means that the measured resistance for each resistor should have a value within ±10% of what it’s supposed to be. Because of this, we can expect the output to be slightly different than what’s expected, such as the measured value in Figure 4.3. Given perfect resistor values, the measurement would be 2.20 volts.

Figure 4.3: Debug Output for Program Listing 4.1.

<table>
<thead>
<tr>
<th>Decimal Value to DAC: 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Value to DAC: 1011</td>
</tr>
<tr>
<td>DVM Reading: 2.25 Volts</td>
</tr>
</tbody>
</table>

About the Code

The first of these two comments is updated to indicate that this is Program Listing 4.1. The second comment is added to indicate that a function was added to the DVM that does D/A conversion.

'Program Listing 4.1
'D/A Converter Added to DVM

A nibble size variable \( n \) is added to the declarations section, and it will be used to store the binary value for the D/A converter.

\( n \) var nib

A \gosub\ command is added to the main routine that sends the program to the \DAC: subroutine.

\gosub DAC
This is the start of the digital to analog conversion (DAC) subroutine, so it’s descriptively labeled DAC. The value of $n$ is set to 11. This means that the output should be $n$ steps above 0 on an output scale of 0 to 16. The value of $n$ can be changed to specify voltage.

```
DAC:
  n = 11
```

The BASIC Stamp I/O pins connected to the D/A converter are set to output. These commands are normally found in the declarations section. If they were placed in the declarations section, the program would run faster because these commands would only be done once at the beginning of the program. Instead, they are executed each time the subroutine is run. The reason they were placed in the subroutine is to make it easier to present some new PBASIC techniques.

```
output 7
output 6
output 5
output 4
```

Next, the BASIC Stamp’s parallel binary output is sent to the D/A converter. We are using the same command for sending outputs that was used in Experiments #1 and #2, but there is a new feature added. The variable $n$ has an extension to indicate which bit in the nibble value is being used. For example, the command `out7=n.bit3` sets the output value of pin P7 equal to the value of `bit3` in the nibble variable $n$. Since we set the value of $n$ to 11, the binary value of $n$ is 1011. Bit 3 is the leftmost bit of the binary number, so it’s 1, which means the output value of P7 is set high. Also, when $n = 11$, P6 is set low, P5 is set high, and P4 is set high.

```
out7=n.bit3
out6=n.bit2
out5=n.bit1
out4=n.bit0
```

That’s all it takes for programming digital to analog conversion using a resistive ladder network. The `return` command sends the program back to the command immediately following the `gosub DAC` command in the `main:routine`.

```
return
```
The first two lines in the `DISPLAY:` subroutine are modified to show the decimal and binary values of `n`, the 4-bit binary value used D/A output value.

```plaintext
dec2 n
bin4 n
dec2 n
```

Modify the Code

If the D/A converter works as we expect, each time the value of `n` is incremented by 1, the D/A output should increment by 0.2 volts. Try starting with `n=0` by modifying the value of `n` in the `DAC:` subroutine.

```plaintext
n = 0
```

Then run the program again with `n=1`.

```plaintext
n = 1
```

Then change `n` to 2 and run the program a third time.

```plaintext
n = 2
```

Continue for each value up to `n=15`

Figure 4.4 shows a sample measurement of a resistive ladder D/A conversion with the value of `n` set to 15. Remember that the 10\% resistors lead to some error in the output. In Experiment #3, we were interested in programming our DVM for accurate calculations to the nearest hundredth of a volt. In this experiment, anything within 10\% of the expected value is fine. So your end output when `n=15` might be as low as 2.7 volts or as high as 3.3 volts. If the errors are larger than that, double check to make sure that a 1k\(\Omega\) resistor didn't get swapped with a 2k\(\Omega\) somewhere in the resistive ladder.

![Figure 4.4: Debug Output for Program Listing 4.1, Revision 1.](image)

- **Decimal value to DAC:** 15
- **Binary Value to DAC:** 1111
- **DVM Reading:** 3.10 Volts
Addressing

Until now, we've been addressing each of the I/O lines one at a time. This works well whenever you need to have control over the status of a particular control line. For example, a single LED, is easily addressed by the individual I/O pin to which it is connected using `outp=value`, where `p` is a pin number between 0 and 15, and `value` is either 0 or 1.

Since the I/O pins (P4 through P7) are used as outputs in this experiment, it would be easier and more efficient to have a method of addressing this group of bits. Notice that it takes four lines of code just to set up the bits as outputs, and then it takes 4 additional lines to individually set each bit. It's not a big deal now, but as time goes on and you begin to create more complex programs, you may find yourself looking for ways to get the most out of the least code.

There are two registers that we need to set to control output on a specific group of I/O lines. The first register is called “direction”. The command “output” sets the direction as an “output”. Conversely, “input” sets the I/O line up as input. The second if been set up as an output, then this register’s “data” can be set to either 0 or 1. This in turn sends either a low or high signal to that pin, and the measured output will either be 0 or 5 volts.

Certain commands in PBASIC allow us to directly address its I/O lines as a word (16 individual bits), two bytes (two sets of 8 individual bits) or as 4 nibbles (four sets of four individual bits). To modify our code, we want to address a nibble at a time, and the four bits that we're using are P4-P7. According to the BASIC Stamp Manual Version 1.9 (which you should have a copy of by now - it's free at www.stampsinclass.com and printed copies are not expensive) the group “P4-P7” is called nibble “b”. The next set of four (P8-P11) is nibble “c”, and so on.

Let’s try using this in a program and figure out how it works. Rewrite the `DAC:` subroutine in Program Listing 4.1 as follows:

```plaintext
DAC:

n = 11
dirb = 15
outb = n

return
```

What's... Addressed:
When an I/O pin is addressed, it means that a value has been written into a particular place in the BASIC Stamp’s RAM. For example, a particular memory location can be set high to set a given I/O pin to be an output. Another address might be used to set the pin high or low.

Performing these operations one bit at a time is not always efficient. The memory locations are adjacent to each other so that the operations can be performed 1 nibble (4-bits) at a time or one byte (8-bits) at a time, or even a word (16-bits) at a time.
Figure 4.5 shows the output is identical to the previous version of the DAC: subroutine. We did the same job with two lines of code instead of eight.

Here is how to count from 0 to 15 using a nibble:

\[
\begin{array}{cccc}
0 &=& 0000 & 4 = 0100 & 8 = 1000 & 12 = 1100 \\
1 &=& 0001 & 5 = 0101 & 9 = 1001 & 13 = 1101 \\
2 &=& 0010 & 6 = 0110 & 10 = 1010 & 14 = 1110 \\
3 &=& 0011 & 7 = 0111 & 11 = 1011 & 15 = 1111 \\
\end{array}
\]

When nibble b is selected using \( \text{dirb} \), each bit in the number \( \text{dirb} \) is set equal to a value corresponding to a data direction as shown below:

\[
\begin{array}{cccc}
\text{Bit in nibble B} & 3 & 2 & 1 & 0 \\
\text{I/O pin} & \text{P7} & \text{P6} & \text{P5} & \text{P4} \\
\end{array}
\]

If we used the command “\( \text{dirb} = 4 \)” the following direction register bits would be set like this:

\[
\begin{array}{cccc}
\text{Bit value} & 0 & 1 & 0 & 0 \\
\text{I/O pin} & \text{P7} & \text{P6} & \text{P5} & \text{P4} \\
\end{array}
\]

This would result in I/O pin P6 being set as an output, and all the other pins (P0, P1, P3) set as inputs. Hence the command \( \text{dirb} = 15 \) (by virtue of the fact that all four bits are a “1”) sets up each of the I/O lines as outputs.

Sweeping the value of \( n \) from 0 to 15 should result in the same values as before.

A truly powerful aspect of using this method of addressing is that we can use PBASIC to count up and down or access values from a look up table to automatically address the I/O pins. The result is that we can program the BASIC Stamp to more effectively control the D/A converter output.
Modify the code immediately following the `'START DISPLAY` label in Program Listing 4.1. First, modify the `debug cls` command, then add a second line as shown.

```
'Start display
debug cls, "DAC Nibble Values", cr
debug "Decimal  Binary  DVM", cr

Modify the main: subroutine as shown:

'Main routine
main:
   for n = 0 to 15
      gosub DAC
      gosub ADCDATA
      gosub CALC_VOLT
      gosub DISPLAY
   next
   stop
goto main
```

Delete the line that sets the value of `n` in the `DAC: subroutine`. Once that's done it should look like this:

```
DAC:
dirb = 15
outb = n
return
```

Also modify the `DISPLAY: subroutine` as shown.

```
DISPLAY:
ddebug dec2 n, " " bin4 n, " ", dec1 v, ".", dec2 v2, " Volts", cr
return
```

Figure 4.6 shows the output. Try that with a hand held voltmeter and you'll begin to see the usefulness of combining the BASIC Stamp with analog interfaces. Imagine trying to test all 4096 levels of a 12-bit DAC one at a time!
### Experiment #4: Basic Digital to Analog Conversion

#### DAC Nibble Values

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>DVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>0.00 Volts</td>
</tr>
<tr>
<td>01</td>
<td>0001</td>
<td>0.20 Volts</td>
</tr>
<tr>
<td>02</td>
<td>0010</td>
<td>0.41 Volts</td>
</tr>
<tr>
<td>03</td>
<td>0011</td>
<td>0.61 Volts</td>
</tr>
<tr>
<td>04</td>
<td>0100</td>
<td>0.82 Volts</td>
</tr>
<tr>
<td>05</td>
<td>0101</td>
<td>1.02 Volts</td>
</tr>
<tr>
<td>06</td>
<td>0110</td>
<td>1.24 Volts</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
<td>1.45 Volts</td>
</tr>
<tr>
<td>08</td>
<td>1000</td>
<td>1.63 Volts</td>
</tr>
<tr>
<td>09</td>
<td>1001</td>
<td>1.84 Volts</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>2.04 Volts</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>2.25 Volts</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>2.47 Volts</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>2.67 Volts</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>2.88 Volts</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>3.10 Volts</td>
</tr>
</tbody>
</table>

This is a very efficient way to collect your voltage sweep data. Looking at the data for the voltage sweep brings several things to light. First, the voltage output for the D/A converter is always a little high. Second, the error increases as the output voltage increases. Third, the largest error is 0.1 volts. This kind of data can be exceedingly useful in electronics design, and the automated testing process is a huge time saver.

#### The Voltage Follower

Let's use the voltage sweep to analyze what happens when the output of the D/A converter is connected to another circuit. We'll use the output of the D/A converter to drive an LED circuit. First we'll connect the D/A converter output directly to the input of the LED circuit. Then we'll use the voltage follower as an intermediate step between the D/A converter output and the LED circuit input.

Figure 4.7 shows the D/A converter with an LED circuit added. The LED circuit is the "load" that the D/A converter must "drive". Run a voltage sweep on this circuit, and fill in the table below. Then try the voltage sweep using the output of the voltage follower in as shown in Figure 4.8. Fill out the same table of voltage sweep information for this second circuit, and compare the two. The voltage follower in Figure 4.8 is referred to as a buffer.
Table 4.1: D/A Converter Output Without Buffer

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>DVM (volts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0011</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>
Experiment #4: Basic Digital to Analog Conversion

Figure 4.8: D/A Circuit With a Buffer to separate the LED circuit's input from the D/A converter's output. SAVE THIS CIRCUIT FOR EXPERIMENT #5.

Table 4.2: D/A Converter Output with Buffer

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>DVM (Volts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0011</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>
Comparing the two tables, it should be fairly clear that the buffer (voltage follower) eliminated a problem caused by connecting the LED circuit directly to the D/A converter's output. The voltage of the D/A output without a buffer reached a maximum well below the D/A converter's 3 Volt maximum. On the other hand, the buffered output went all the way to 3 volts with no problem.

The reason for this goes back to Ohm's Law: \( V = I \times R \) (Voltage equals current multiplied by resistance). Each BASIC Stamp I/O pin can supply up to 20 mA of current. In the case of the resistive ladder network without the buffer, the BASIC Stamp I/O pins reached their maximum current output. Meanwhile, the resistance seen by the I/O pins stayed about the same. In other words, \( I \times R \) reached a limit because \( I \) (the current) could no longer increase, and \( R \) is a fixed value. So the output voltage is equal to a current value that can't get any higher multiplied by a fixed resistance. This is why the voltage stops increasing.

Aside from isolating two circuits from each other, an op-amp in voltage follower configuration can typically supply more current than the circuit connected to its input. The name buffer is commonly used when a voltage follower is used to supply extra current.

The circuit from Figure 4.9 will also be used at the beginning of Experiment #5, so do not disassemble the circuit after completing Experiment #4.
On the lines below, insert the appropriate words from the list on the left.

- binary
- discrete
- bit
- output
- current
- addressing
- direction
- sweep
- automate

A set of ___________ voltage values sent to the inputs of a resistive ladder network results in a ___________ voltage level at the output. This method is an inexpensive alternative to integrated circuits. Its advantages are flexibility in resolution and low cost. Its main disadvantage is accuracy.

The ___________ extension can be attached to a variable to select a particular ___________ within a nibble, byte or word. This can be used to select bits from a binary number in memory and set ___________ values equal to those bits.

PBASIC can be used to address BASIC Stamp I/O pins a bit at a ___________ at a time, or alternatives exist for ___________ nibble, byte or word size groups of I/O pins. This allows the ___________ and output values of the I/O pins to be addressed as groups of bits instead of as a single bit.

A voltage ___________ can be run on a D/A converter. This allows one to view the D/A converter’s outputs for all possible binary inputs. The BASIC Stamp can be programmed to ___________ this process and display the data in a table format.

The voltage follower can be used as a buffer, which can supply extra ___________ to the input of a circuit connected to the D/A converter's output.
Questions

1. “1101” is what value in the decimal number system? What voltage would you expect from the D/A converter output if you sent it this binary number?

2. What function is provided by a D/A converter?

3. What are some of the advantages and disadvantages of the resistive ladder network?

4. Why does the D/A voltage “jump” from one value to another, unlike the voltage available on a pot?

5. How does the voltage follower solve the output voltage range problem when the D/A converter’s output is

Challenge!

1. Create an 8 bit “resistive ladder” D/A converter. Draw the complete schematic.

2. Write a program that will step through 256 different analog voltages. Each voltage should be present for 100 milliseconds on the voltmeter.
There are many different “real world” circuits that require some sort of analog voltage. For example, when you listen to a compact disc, you’re listening to sounds that started as analog signals from a microphone. Then the signal underwent the A/D process when it was digitized. The CD player then performs the D/A conversion using the digital information on the CD. The analog signal is amplified and played on a speaker.

When designing commercial products, it is your responsibility to determine the most appropriate (and cost effective) method to accomplish a given task. A resistive D/A converter is a very economical method to get an analog voltage from a digital device.

You’ve just built a variable voltage source and a digital voltmeter on the same breadboard. You also used some PBASIC Programming techniques to obtain information about the converter using a voltage sweep. One application of this can be testing circuits, but there are many others. In Experiment #5, we’ll see how PBASIC can be used to control the volume of tones emitted by a speaker. In Experiment #7, we’ll use D/A conversion to control the brightness of an LED transmitting a signal to a photoresistor.

The applications for D/A and A/D interfaces combined with a microcontroller are only limited by one’s imagination. These techniques can be applied to automated houses, irrigation systems, rocket guidance systems, and robotics to name a few. Control systems engineering is a field of study within electrical engineering that you might look into if you find designing such systems interesting.