Experiment #3: Basic Analog to Digital Conversion

Build Your Own Digital DC Voltmeter

A digital DC voltmeter (DC DVM) is a handy tool for measuring voltage between two contact points. In this experiment, we will build a DVM for measuring DC voltage in the 0 to 5 Volt range. A common use for a DC DVM is testing the voltage (potential) between the two terminals on a battery.

A digital voltmeter is so named because it displays its measurements with digits. The digits 0 through 9 and a decimal point are used to display the voltage measurements as decimal values. The digits 0 and 1 could be used. It would still be a "digital" voltmeter, but it would have binary display instead of a decimal display. Making sense out of each measurement would be time consuming. Since our DVM processes its measurements in binary, we'll start with a binary display and then modify it to the more conventional and easy to read decimal display.

What's a...

Continuous range:

A minimum value, a maximum value, and everything in between.. When a source of voltage varies over a continuous range, it is considered an analog voltage.

We will use our DVM to sample voltages over a continuous range, from 0 to 5 Volts. So, the voltage we measure might be 1.234 Volts or 3.857564... Volts, or 4.9999... Volts, etc.

In Experiment #1, we used an LED circuit to display changes in analog voltage level applied to a circuit. As a “continuously variable value”, analog voltage varies within a continuous range. We'll use the potentiometer as we did in Experiment #1 to make a range of voltages that can vary continuously between 0 and 5 volts on the Board of Education.

Although information about analog voltage can be processed efficiently with binary devices, the voltage has to be sampled and described using binary numbers first. The ADC0831 is a common integrated circuit that does this job. It describes the analog information with binary numbers for devices that process binary information, such as the BASIC Stamp.

In this experiment, we will make a DVM using the BASIC Stamp together with the ADC0831 integrated circuit. A pot will be wired to the Board of Education and adjusted to make analog output voltage. The DVM will then be used to measure samples from the pot’s continuous range of voltage outputs.

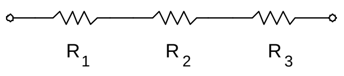
Parts Required

(1) ADC0831
(1) 100K potentiometer
(10) Wires, give or take a few

What's a...

Resistors in Series:
A chain of resistors connected end to end. Three resistors in series are shown below. The three resistors can be viewed as a single resistance whose value is:

$$R_{series} = R_1 + R_2 + R_3$$



FYI:

Resistor Values
When you know the value of the two resistors in Figure 1.2, you can predict the output voltage using this equation.

$$V_{output} = V_{input} \times \frac{R_2}{R_1 + R_2}$$

Not surprisingly, it's called the voltage divider equation, and this technique for scaling down an input voltage is commonly referred to as using a voltage divider.

The Potentiometer, a Source of Variable Voltage

There is a reason why the voltage at the wiper terminal of a pot changes. when you turn the knob. The wiper terminal makes the single resistive element in the pot work like two resistors in series. Figure 3.1 shows two resistors in series. When input voltage is applied and output voltage is measured as shown in Figure 3.1, the circuit is referred to as a voltage divider. R_1 and R_2 are the resistances between the wiper and the other two terminals on the pot, and their values change as the pot is adjusted. Since the pot causes the R_1 and R_2 to vary, we can call our wiper terminal the output of a variable voltage divider.

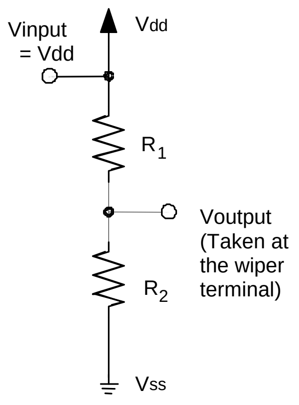


Figure 3.1: A Voltage Divider Circuit shows how the wiper in a potentiometer makes the single resistive element look like two resistors in series.

Voutput is the voltage measured at the wiper terminal.

The ADC0831 Integrated Circuit - An 8-bit Analog to Digital Converter

The ADC0831 is an integrated circuit referred to as an 8-bit analog to digital converter (A/D converter) with synchronous serial output. Let's look at what each of these terms mean:

- An integrated circuit (IC) is a circuit with microscopic components implanted on the surface of a silicon wafer. The Analog and Digital Parts Kit has three chips used in these experiments. Each chip is a black casing with eight pins. The black casing houses and protects an integrated circuit.

- An A/D converter measures an analog voltage sample and returns a binary number that describes the sample.

- 8-bit is the number of binary digits the ADC0831 uses to describe the analog voltage it samples. 8-bit is also the resolution of the A/D converter. You can count from 0 to 255 (decimal) using an 8-bit binary number. This means that the ADC0831 can approximate the voltage it measures as one of 256 levels. A higher resolution converter, such as 12-bit, would break the same voltage range into 4096 levels because you can count from 0 to 4095 with 12 binary bits.

- Synchronous and serial are terms we learned about in Experiment #2. We sent serial binary digits (bits) to the BASIC Stamp using one pushbutton and the bits were synchronized to a second pushbutton that was used to send a clock signal. The ADC0831 works in a similar way. The difference is that the ADC0831 depends on a clock signal sent by the BASIC Stamp to time the sending of each serial output bit.

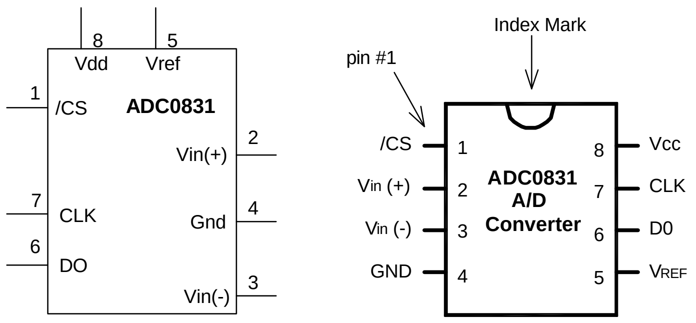
What's a...

Binary control signal:
A voltage signal with two possible states, low or high, that is sent to tell a device how or when to do something. The ADC0831 requires control signals to activate it and a clock signal to synchronize the sending of each of the it's output bits.

The BASIC Stamp will be programmed to read and store the 8-serial-bits transmitted by the ADC0831. We'll also program the BASIC Stamp to display the decimal equivalent of the binary output. Next, we'll use this decimal equivalent to calculate and displays the measured voltage in decimal form (our DVM output). The BASIC Stamp must also be programmed to send binary control signals to make the ADC0831 do its job.

Figure 3.2 shows a pinout map of the ADC0831. Each pin has a number and a label. The number is important for getting the wires connected to the right pins when constructing your circuit. The labels indicate the function of each pin.

Figure 3.2: ADC0831 Circuit Symbol and Pin Map. The pin map on the right shows the pins and labels according to where they are on the chip. The circuit symbol on the left also shows the pins and their labels, but it's typically drawn in a way that most conveniently fits into the schematic.



What's a...
Bias:
A method of applying specific voltage levels at certain places in a circuit to calibrate or tune it.

The notation for the ADC0831's inputs and outputs works as follows: Vin(+) is the analog input, and D0 is the serial output. VREF and Vin(-) are used to bias the IC. Vcc and GND are used for supplying power to the IC. Vcc is essentially the same term as Vdd on the Board of Education, and GND corresponds to Vss. /CS stands for active low chip select, and CLK stands for clock. Both are inputs for binary control signals.

To prime the ADC0831 for taking a measurement, the /CS pin has to receive a signal from the BASIC Stamp that starts high, then goes low. This signal has to stay low for the duration of the conversion. Then the CLK input must receive a single clock pulse (a term introduced in Experiment #2, Figure 2.8) to signify that the conversion should start at the next clock pulse. For this IC, a clock pulse starts low, goes high, then goes low again. It takes 8 more clock pulses to complete the conversion. Each time a clock pulse is received by the CLK input, another of the serial bits is sent by the D0 output.

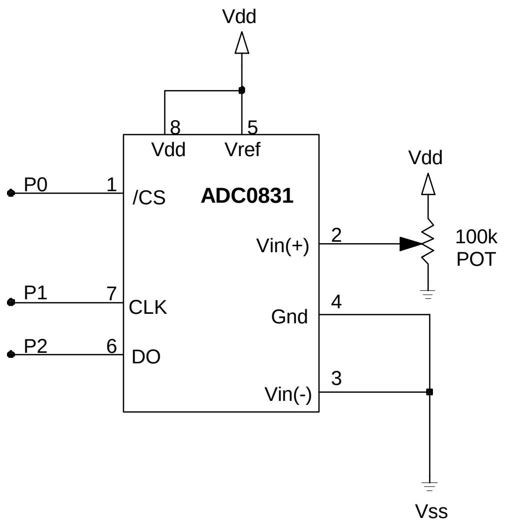
Electronics designers use data sheets to find the kind of information just discussed. Each IC manufacturer publishes data sheets for the integrated circuits they make. The information just covered on the pin map and control signals was condensed from a data sheet published by National Semiconductor, the maker of the ADC0831. Of course all of the datasheets are available on the manufacturer's web sites.

Build It

Figure 3.3 shows the schematic for this experiment. This is a fairly simple circuit to build, so let's try it without the breadboard example. Hopefully you're getting the hang of the list of connections described by a schematic. Remember, when working with the connections to an IC, use the index mark on the chip along with the pin map to figure out the pin numbers!

Figure 3.3: Schematic
List of connections made on this schematic:

- Pin 1 on the ADC0831 is connected to pin P0 on the BASIC Stamp.
- The wiper terminal of the pot is connected to pin 2 on the ADC0831.
- Of the two remaining terminals on the pot, one is connected to Vdd on the Board of Education, and the other is connected to Vss.
- Pins 3 and 4 on the ADC0831 are connected to Vss.
- Pins 5 and 8 on the ADC0831 are connected to Vdd.
- Pins 7 and 6 on the ADC0831 are connected BASIC Stamp pins P1 and P2 respectively.



Program It

Program Listing 3.1 is the first step to a functional DC voltmeter. This program displays the 8-bit serial output of the ADC0831. Enter the code and save it as P3_1R0.bs2. There will be three revisions of this program listing, so it will become important to keep track of your source code versions.

We'll modify the code so that it also displays the decimal conversion of the 8-bit binary number. Next we'll add some more code to adjust the number to a 5 Volt scale. Make sure your circuit is constructed correctly and your programming cable and power source are connected, then run the program.

```
‘Program Listing 3.1
‘ADC0831 Binary output display.

'Declarations.
adcbits var byte
v var byte
R var byte
v2 var byte
v3 var byte

CS con 0
CLK con 1
D0 con 2

'Start display.
debug cls

'Main routine.
main:

gosub ADCDATA
gosub CALC_VOLTS
gosub DISPLAY

goto main

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

CALC_VOLTS:
return

DISPLAY:
debug home
debug "8-bit binary value:  ", bin8 adcbits
return
```

The Output

If the pot is adjusted somewhere in the middle of its range, the output displayed in the Debug window should look similar to Figure 3.4. As you adjust the pot, the zeros and ones should change rapidly. Each time you stop adjusting the pot, the output should settle, and a new pattern of eight zeros and ones should display.

Figure 3.4: Debug Output for Program Listing 3.1.

```
8-bit binary value:   10110100
```

If your debug window responds this way, it's likely your circuit and program are working right. If it doesn't do this, check the wiring on your circuit. Also make sure code is entered correctly. Sometimes just one wrong letter will cause the program not to work properly. The Debug window could also be hidden from view. It can be accessed from the menus by selecting Run/Debug/New Terminal in the Windows version of the BASIC Stamp Editor.

About the Code

The first few lines of text in this program are comments that begin with apostrophes, and they don't have any function in the program aside from explaining it to someone reading the code.

```
'Program Listing 3.1
'ADC0831 binary output display.
```

The next section is called the variable declarations section, and it begins with a comment explaining that this is the declarations section. This program uses just the `adcbits` variable at present. We'll add code that will make use of the other four variables, `v`, `R`, `v2`, and `v3`.

```
' Declarations
adcbits var byte
v var byte
R var byte
v2 var byte
v3 var byte
```

Following is a new type of declaration we haven't used before. Three constants are defined using the `con` directive. After we define these constants, we can use `CS` in place of the number 0, `CLK` in place of the number 1, and `D0` in place of the number 2. The names for the constants were chosen to correspond with the ADC0831's pin labels. The numbers were chosen based on BASIC Stamp I/O pin numbers.

```
CS con 0
CLK con 1
D0 con 2
```

What's a...
Subroutine:
A subroutine is a small program that does a specific task within a larger program.

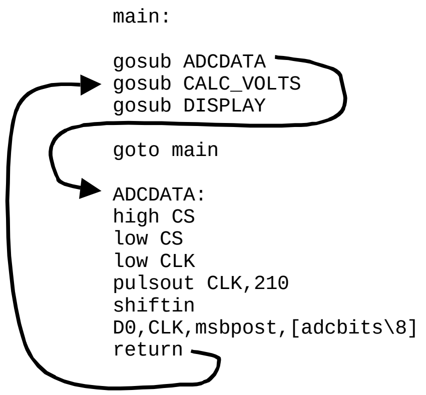
Next there's a loop that contains three `gosub` commands. The `main:…goto main` routine runs 3 different subroutines over and over again. The subroutines are named `ADCDATA:`, `CALC_VOLTS:`, and `DISPLAY:`. The label `main:` is used in the same manner that we used the `loop:` label in the first two programs. The label name `main:` was chosen because, as the comment preceding this routine indicates, it's the "main routine" in the program.

```
'Main routine
main:

gosub ADCDATA
gosub CALC_VOLTS
gosub DISPLAY

goto main
```

So, how does a `gosub` command work? As shown in the flow diagram in Figure 3.5, `gosub ADCDATA` means go to the subroutine labeled `ADCDATA:` and come back when finished. The program jumps to the `ADCDATA:` label and starts executing commands. As soon as it gets to the `return` command, the program jumps back to the command just after `gosub ADCDATA`. In this case, the next command is another `gosub` command, `gosub CALC_VOLTS`.



```
main:

gosub ADCDATA
gosub CALC_VOLTS
gosub DISPLAY

goto main

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

Figure 3.5: Flow Diagram
A subroutine sends the program to the specified label. In this case the label is `ADCDATA:.` Then the program continues executing commands until it encounters the `return` command. The `return` command sends the program back to the command immediately after the `gosub` command. In this case it's another `gosub` command.

The subroutine `ADCDATA:` sends control signals to and collects output data from the ADC0831. This subroutine is where the usefulness of the `con` directive really shows. P0 on the BASIC Stamp is connected to the /CS pin on the ADC0831. Likewise, pins P1 and P2 are connected to CLK and D0. When sending signals to the /CS pin, we can enter a command like `high CS` instead of `high 0`. It makes more sense when writing the code, and it makes deciphering the code easier too. It's also easier to change one constant in the top of the program should you decide to connect the ADC0831 to a different BASIC Stamp I/O pin.

The command `high CS` sends a high signal to the ADC0831's /CS pin. To start a conversion, we need to send a high signal (5 volts). Then we need to send a low signal (0 volts) to the /CS input on the ADC0831 using `low CS`. The signal sent to the ADC0831's /CS input needs to stay low for the duration of the conversion.

```
high CS
low CS
```

The `low CLK` command is necessary so that the clock pulses take the right form. Using this command guarantees that the next command (`pulsout`) will send a clock pulse that has the right shape, low-high-low. Sending high and low signals using the `high` and `low` commands is an alternative to the `out0=1` and `out0=0` techniques used in the first experiment.

```
low CLK
```

The `pulsout CLK,210` command sends a clock pulse to the ADC0831's CLK input. This is the first clock pulse, and all it does is tell the ADC0831 to start converting on the next clock pulse. Because of this, we don't need to check for input from D0 after this first clock pulse.

```
pulsout CLK,210
```

Since we set the clock low just before this command, `pulsout` sends the desired low-high-low signal. The duration of the high segment is twice the number specified in the `pulsout` command, in microseconds (us). 1 us = 1/1,000,000 of a second. Therefore the duration of this high segment is 2 us $\times$ 210 = 420 us.

The command `shiftin D0,CLK,msbpost,[adcbits\8]` is a powerful instruction that takes care of all the synchronous serial communication so that we don't have to program it as we did in Experiment #2. In effect, this command sends clock pulses to the ADC0831's CLK input and reads output bits from ADC0831's D0 output. This command also loads each of the ADC 0831's output bits into the `adcbits` byte.

```
    shiftin D0,CLK,msbpost,[adcbits\8]
```

The `shiftin` command is discussed in more detail in the BASIC Stamp Manual Version 1.9, but the general format for the command is:

```
    shiftin data pin,clock pin,mode,[variable\bits]
```

In our case, the data pin is D0, a constant equal to the number 2. This constant is used to reference BASIC Stamp I/O pin P2 in this program. Likewise, the clock pin is CLK, which is a constant equal to the number 1, and it references BASIC Stamp I/O pin P1. The mode in this case is msbpost, and it's one of four transmission modes that can be used in this command. It indicates that the the ADC0831's output bits are ready after the clock pulse's negative edge, the transition from high to low. It also indicates that the bits are transmitted in descending order, starting with the MBS. [adcbits\8] means the data is shifted into the adcbits variable, and 8-bits are expected.

The `CALC_VOLTS:` subroutine is empty right now, but we will develop the code for this subroutine shortly. The subroutine will calculate the measured voltage to the hundredth's decimal place.

```
    CALC_VOLTS:
    return
```

At present, the `DISPLAY:` subroutine just displays the binary output for each analog voltage sample taken by the ADC0831. It will be modified to display the decimal equivalent of the 8-bit binary value. It will also be modified to display the voltage measurement.

The command `debug home, cr, "8 bit binary value:  ", bin8 adcbits` sends the cursor to the top-left "home" position in the Debug window. Then it prints the message in quotes. The modifier `bin8` makes it so the value of the `adcbits` variable is displayed as 8 binary digits.

```
    debug home, cr, "8 bit binary value:  ", bin8 adcbits
```

If the number of digits displayed is likely to vary, when using the `debug home` command, always specify how many digits the numeric outputs should have with modifiers like `bin8`, `dec3`, etc. When `debug cls` is used, it's OK to skip specifying the number of digits, so modifiers such as `bin` and `dec` can be used instead.

The `debug home` command is better for programs that cycle through loops where the Debug window display is updated frequently and rapidly. When `debug cls` is used under these circumstances, the repeated clearing the Debug window causes a flicker that makes the display difficult to read.

The `return` command sends the program back to the line immediately following the `gosub display` command.

We will modify the `DISPLAY:` subroutine to display the decimal equivalent of the binary contents of `adcbits` in the Debug window. Code will also be added to make the Debug window display our DVM reading.

Interpreting the Output

The ADC0831 measures an analog voltage at its input. Then it sends the BASIC Stamp a binary number describing the value it measured. For now, we'll focus on a voltage scale that starts with 0 volts and ends at 5 volts.

With an 8-bit binary number, you can start counting with 00000000 and count all the way up to 11111111. Translated to decimal numbers, it's the same as counting from 0 to 255. When applied to a 5 Volt scale that starts at 0 volts, it's the same as counting from 0 to 5 volts using 255 voltage steps.

For the 5 Volt scale, when the ADC0831 measures 0 volts, you get 00000000. When it measures 5 volts, the output is 11111111. It turns out that the Debug window output 10110100 from Figure 3.4 is the same as the decimal number 180. Decimal-180 in turn corresponds to a measured voltage of 3.53 volts.

Binary to Decimal Conversion Revisited

So how do we know that 256 combinations can come from an 8-bit binary number? Remember, you can always tell how many numbers (combinations of 0s and 1s) can come from a given number of bits by using this formula from Experiment #2:

$$\text{combinations} = 2^{\text{bits}}$$

This means the number of combinations equals two raised to the power of the number of bits. For 8-bits, the number of combinations is $2^8 = 256$. For 12-bits, the number of combinations is $2^{12} = 4096$, and so on.

Let's use the two-step method from Experiment #2 to convert the 8-bit binary number 10100101 to its decimal equivalent. Here is a repeat of the bit multipliers table to work with:

Table 3.1 – Bit Multipliers for an 8-bit Binary Number

Bit	7	6	5	4	3	2	1	0
Multiplier	128	64	32	16	8	4	2	1

First, multiply each bit by its power of two from Table 3.1

128 X 1 = 128
64 X 0 = 0
32 X 1 = 32
16 X 0 = 0
8 X 0 = 0
4 X 1 = 4
2 X 0 = 0
1 X 1 = 1

Second, add all 8 of the decimal values:

128+0+32+0+0+4+0+1 = 165

Now we know the binary number 10100101 is equal to the decimal number 165. To display this conversion, a single debug command can be added to the `DISPLAY:` subroutine. Added lines are shown with a "   ".

```
DISPLAY:
debug home, "8 bit binary value:  ", bin8 adcbits
debug  CR, CR, "Decimal value:  ", dec3 adcbits      '
return
```

The command `debug CR, CR, "Decimal value:  ", dec3 adcbits` tells the Debug window to display two carriage returns followed by the message in quotes, followed again by the 3-digit decimal value of `adcbits`. If the actual number only has one or two digits, the Debug window will automatically display leading zeros since `dec3` was specified. For example, the number 7 will display as 007, and the number 85 as 085, etc.

With some careful adjustment of the pot, we can check our work using output sample shown in Figure 3.6.

Figure 3.6: Debug Output for Program Listing 3.1, Revision 1.

```
8-bit binary value:  10100101

Decimal value:  165
```

Calculate Voltage

Now that we know the decimal equivalent of the ADC0831's binary output, we can do a few calculations to get the measured voltage. To find out what voltage the decimal number corresponds to, we need to calculate where in the voltage range the number falls. Here is an effective way to think about the problem.

- We know that the voltage is on a 0 to 5 Volt scale, and we know that the ADC0831's output is on a scale from 0 to 255.
- In other words, the measured voltage is to 5 as the A/D output is to 255.

This translates to fractions as:

$$\frac{\text{Voltage}}{5} = \frac{\text{Decimal A / D Output}}{255}$$

We can re-arrange this equality to calculate the voltage:

$$\text{Voltage} = \frac{5 \times (\text{Decimal A/D Output})}{255}$$

So, now we know to multiply by 5 and divide by 255 for a 5 Volt scale with 256 levels. We can calculate the voltage from Figure 3.6 where the ADC0831's output is 10100101 = 165. The measured voltage is:

$$\text{Voltage} = \frac{5 \times 165}{255} = 3.24 \text{ Volts rounded to two decimal places.}$$

To calculate and display this voltage using the BASIC Stamp, we'll add some code to both the `CALC_VOLTS:` and `DISPLAY:` subroutines. First, the voltage equation needs to be expressed in PBASIC code. Here is an example of some code that could reasonably be expected to work.

```
v = 5*adcbits/255
```

This PBASIC calculation looks like it will give us the output we want, but it won't. It's instructive to try it this way and see what happens. Modify the `CALC_VOLTS:` and `DISPLAY:` subroutines in Program Listing 3.1 as follows:

```
CALC_VOLTS:
v = 5*adcbits/255                                    '
return

DISPLAY:
debug home, "8 bit binary value:  ", bin8 adcbits
debug  CR, CR, "Decimal value:  ", dec3 adcbits
debug CR, CR, “DVM Reading:  “,  dec3 v, “ Volts”     '
return
```

We calculated that 165 would lead to a measured voltage of 3.24 volts. The 003 volts shown in Figure 3.7 is only accurate to the nearest volt! What happened?

Figure 3.7: Debug Output for Program Listing 3.1, Revision 2.

```
8-bit binary value:  10100101

Decimal value:  165

DVM Reading:  003 Volts
```

The PBASIC command set for the BASIC Stamp does arithmetic using integer values. Integers are the counting numbers: ...-2, -1, 0, 1, 2, 3, etc. The largest integer the BASIC Stamp can process is 65535. When using integer arithmetic, the fractional part of any answer is discarded. Fortunately, we can still use integer arithmetic to find the fractional values we are trying to display.

Before dividing, the A/D output is multiplied by 5. This doesn't cause any problems.

$5 \times \text{(Decimal A/D Output)}$ is essentially the same as $5 \times \text{adcbits}$.

In our voltage calculation example, that's $5 \times 165 = 825$. Since 825 is an integer that is less than 65535, this part of the calculation goes without a hitch. The problem occurs when we try to divide 825 by 255. The answer has a fractional component that never gets calculated with integer math.

Doing "long division" with a pencil and a piece of paper takes several steps, and it relies solely on integer arithmetic. Let's look at how to calculate the answer for this division problem.

$$\text{Voltage} = \frac{5 \times \text{(Decimal A/D Output)}}{255} = 3 + \text{a remainder of } 60$$

In long division, calculating the part of the answer to the right of the decimal point is repetitive. We multiply the remainder by 10, then divide by 255 again, then take another remainder, multiply it by 10, and divide by 255 again, etc. A shortcut to this procedure is to take the remainder, and multiply it by 100, then divide by 255. This gives us two decimal places. Let's try it.

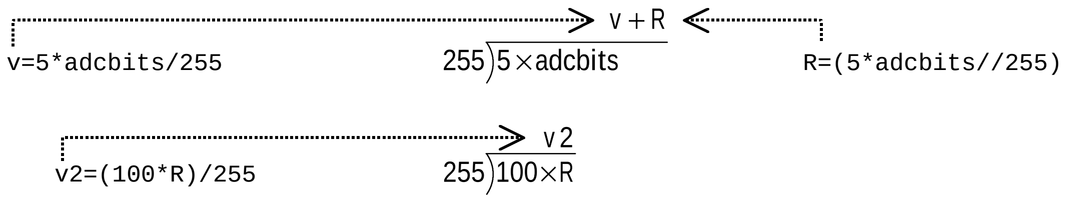
$$(60 \times 100) \div 255 = 6000 \div 255 = 23.5924\ldots \xrightarrow[\text{Integer Math}]{} 23$$

Remember: the BASIC Stamp cuts off everything to the right of the decimal point without rounding. This is called truncating. The result we get is 23. This result should have been rounded up to 24 because 23.5294 is more than half way to the next integer value, 24. For now, let's stick with 23 to the right of the decimal point.

What's a...
Algorithm:
A procedure for solving a problem. The procedure is broken down into repeatable steps.

Our answer using this algorithm is the integer 3 to the left of the decimal point, and the integer value 23 to the right of the decimal point. Since we used only integers in our arithmetic, it should work using PBASIC and the BASIC Stamp.

Since the BASIC Stamp works with integers, it's not surprising that there is a PBASIC command to calculate the integer remainder of a division problem. The operator for division is / and the operator for getting the remainder is //. Let's try converting this algorithm into PBASIC code to do the work for us. The steps for long division below show the PBASIC commands corresponding to the steps in the algorithm.



This gives us our 3 PBASIC commands for calculating the values to the left and right of the decimal point. To reconstruct the fractional value on the display, we'll print a period "." in-between the two values.

The first of the three commands is already in our `CALC_VOLTS:` subroutine. Just add the other two instructions to complete the algorithm.

```
CALC_VOLTS:
v=5*adcbits/255
R=5*adcbits//255                               '
v2=100*R/255                                   '
return
```

The `DISPLAY:` subroutine also needs to be updated to print the two variable values with a period in between them. Make sure to update the line in the `display:` subroutine exactly as shown below.

```
DISPLAY:
debug home, "8 bit binary value:  ", bin8 adcbits
debug  CR, CR, "Decimal value:  ", dec3 adcbits
debug CR, CR, “DVM Reading:  "
debug dec1 v, ".", dec2 v2, “ Volts”            '
return
```

Now run the program again, and see what happens. Figure 3.8 shows an output sample, which is now almost ready to display to the nearest hundredth of a Volt. All that needs to be corrected is a rounding error in the hundredths decimal place.

Figure 3.8: Debug Output for Program Listing 3.1, Revision 3.

```
8-bit binary value:  10100101

Decimal value:  165

DVM Reading:  3.23 Volts
```

All that remains to be done is to correct the rounding error in the hundredth's place. This rounding problem can be corrected by adding the segment of code shown below to the `CALC_VOLTS:` subroutine.

```
CALC_VOLTS:
v=5*adcbits/255
R=5*adcbits//255
v2=100*R/255
v3=100*R//255                        '  new line
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
```

<u>The Output</u>

The output sample from Figure 3.9 indicates that the DVM is now calculating to the hundredth's decimal place correctly.

Figure 3.9: Debug Output for Program Listing 3.1, Revision 4.

```
8-bit binary value:  10100101

Decimal value:  165

DVM Reading:  3.24 Volts
```

IMPORTANT: As soon as you're sure your program works right, save it as P3_1R4.bs2. We will add to both the code and circuit in the next experiment.

About the Code

To round off to the nearest hundredth, we need to know the digit in the thousandth's place. Using the rules of long division, we can simply set a new variable, `v3` equal to the remainder from the calculation for `v2`, and divide by 255 again.

```
v3=100*R//255
v3=10*v3/255
```

Instead of using another variable, `v3` is simply redefined in the second line. The value of `v3` to the right of the equals sign is the one calculated in the first line. The value of `v3` to the left of the equals sign is the redefined value, which is ten times the old `v3`, divided by 255.

This process could be repeated over and over again to get the digit in the ten-thousandth's place, the hundred-thousandth's place, and so on.

Once the digit in the thousandth's place is known, the rules for rounding apply as follows:

- If the digit in the thousandth's decimal place is less than 5, skip adding 1 to the hundredth's decimal place.
- Otherwise, add 1 to the hundredth's decimal place.
- In either case, truncate everything after the hundredth's place.

Since the value `v2` is already truncated, we just need code for deciding whether or not to add 1 to the hundredth's place. It turns out to be a decision on whether or not to add 1 to `v2` as shown below.

```
if v3<5 then skip_a_line
v2=v2+1
skip_a_line:
```

Since the value in the ones place is stored in a different value, we need to check and see if adding one to the hundredth's place increments that value. Without this code, 3.996 would round to 3.00 instead of 4.00.

```
if v2<100 then skip_out
v=v+1
v2=0
skip_out:
```

Save this code, and if possible, leave the circuit as it is because we can use this DVM to take measurements on the circuit we build in Experiment #4.

Resolution

The BASIC Stamp is now programmed to accurately calculate the voltage associated with the ADC0831's binary output, and the calculation is accurate to the hundredth's decimal place. Although sources of calculation error have been been eliminated, there is another source of error that caused by the resolution limitation of the A/D converter.

The A/D converter chip we are using is capable of 256 binary values. This means that each measured voltage gets rounded to one of 256 discrete values. The step size is the amount of the voltage range covered between each of these discrete values. Since the first value is zero, there are 255 voltage steps. The step size is given by :

$$\text{Step Size} = \frac{5\ \text{Volts}}{255\ \text{steps}} = 0.0196\ \text{Volts/step} \cong 0.02\ \text{Volts/step}$$

With this in mind, each time you adjust the pot, the converter comes close to approximating the analog value, but it's not exact because of the resolution constraints. So, there is still some uncertainty at the hundredth's decimal place. In some applications, the uncertainty is stated along with the measurement. Assuming the ADC0831 rounds at the half way point, we can use this convention to read the voltage from Output Sample 1.5 as: "3.24 volts plus or minus 0.01 volts."

Higher resolution converters are available, such as 12 and 16 bit (and higher), but because of their higher resolutions, they come with a higher cost as well. The improvement in resolution is significant. As mentioned before, a 12-bit converter will give you a resolution of 4095 steps. This results in 5 volts / 4095 steps, or one step for every 0.0012 volts. A 12-bit converter typically costs more money than an 8-bit converter. There is also a cost in terms of the amount of memory the measurement takes, (12 as opposed to 8-bits) and the amount of processing it takes to get each measurement (13 clock pulses instead of 9).

Calibration

What if the power supply on the Board of Education only supplies 4.963 volts instead of 5.000 volts? The BASIC Stamp voltmeter can be calibrated using a second voltmeter known to be highly accurate. The difference between Vdd and Vss can be measured using the accurate voltmeter. Developing the code to correct for this error requires more representation of fractional values using integer math and is best left as a challenge problem.

Another item to consider if you're shooting for a high degree of precision is that different current draws on the power supply can also cause variations in power supply output voltage. This is an experiment unto itself that would also require additional equipment. As you probably guess, designing for a high degree of precision involves a number of design challenges. For the remaining experiments, the present degree of accuracy of our DVM is sufficient.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

binary

resolution

analog

serial

input

8-bit

Using the A/D converter makes it possible to process ___________ information with the BASIC Stamp, a digital (___________) device. The converter used in this experiment is the ADC0831 integrated circuit, an 8-bit, ___________ A/D converter.

For a given analog ___________ , the A/D converter outputs an ___________ binary number. The BASIC Stamp can be used to control and collect data from the A/D converter. Various programming techniques can be used to read, remember, and display this data.

The digital representation of the converted analog signal is very good, but not perfect due to inherent ___________ limitations of the A/D converter. A second source of error for the Stamp DC DVM can result from the fact that the power supply on the Board of Eductaion does not necessarily supply exactly 5 volts.

Questions

1. In your own words, explain the function of an A/D converter.

2. What would be the resolution if you were to use a 16-bit A/D converter in this experiment?

3. How does the voltage divider equation relate to the wiper terminal of the pot? What would you expect the output to be if the resistors are equal? Can you prove this?

4. How do the measurements taken at the wiper terminal of the pot in this experiment differ from those taken in Experiment #1? What is gained by using the ADC0831 for measurements instead of just using a BASIC Stamp I/O pin to check the voltage?

5. Given the resolution of our 8 bit A/D converter, when the voltage on the pot is set to 3.6 volts, what decimal value will be displayed? What's the binary value?

Challenge!

1. Use another jumper wire to connect the wiper terminal of the pot to an unused BASIC Stamp I/O pin. Add a subroutine to the DVM program that monitors the state of the I/O pin set to input. Determine if the threshold voltage you were working with in Experiment #1 is indeed 1.4 volts.

2. Write a program that will monitor the analog value of the 100 kΩ potentiometer, and alert you to the fact that it has gone beyond a certain pre-set limit.

3. Write a program and build a circuit that creates a "safety zone" between 1.0 volts and 2.0 volts. If the analog voltage goes outside of these boundaries, an LED blinks.

4. Draw the complete schematic for Challenge #3, and modify the program so that the LED is only on when the voltage (as set on the pot) is set on exactly 2.0 volts.

5. Assume the power supply on your board of education supplies 4.960 volts. Develop a subroutine to adjust the voltage measurements to this scale.

Why did I learn it?

There is a wide variety of electronic applications where analog signals are measured and digital devices are used to process the analog signal data. In this experiment, we used a BASIC Stamp and A/D converter combination to construct a digital DC voltmeter. We'll be using the BASIC Stamp DC-DVM in several of the remaining experiments. There is a surprising variety of uses for such a device as you will discover with each new experiment.

How can I apply this?

In developing the digital voltmeter, the process of sampling voltage, converting it, and processing it in digital form was introduced. An example of another use for an A/D interface is the digital sampling of the analog signal from a microphone for digital recording purposes. Another example that could make use of the circuit we built is a door sensor. Our potentiometer could be attached to a door hinge, and the analog information could be used to monitor how far the door is open. This circuit could in turn be incorporated into a larger system that controls how far open the door swings.

The field of analog to digital conversion is an industry in itself. There are semi-conductor manufacturing companies that specialize solely in creating A/D conversion chips and systems. Whether you'd like to design at the component level, or at the integrated circuit level, there will always be a need for creative analog interfacing – simply because the world isn't black and white (binary), it's all the colors in-between as well (analog).