Experiment #1: Basic Analog to Digital Conversion

We live in an analog world.

What does that mean? Well, to fully understand the answer we'll have to delve into the ways in which a digital device (such as the BASIC Stamp) operates, and how it needs to have data presented to it in the proper "digital" form.

Simply stated, analog means that there are many conditions or levels that a device can assume. For example, you could think of a door as either being "fully open" or fully closed. In a "perfect world" this would be a digital (or more specifically, a **binary**) condition.

In reality however, there are many different conditions (or "**states**") in which the door could be. Yes, it could be fully open or fully closed, but it could also be partially open. In fact, there are an infinite number of positions in which the door could be open or closed. It could be "open" 1.12", or only open 1.00014", etc. Although we would like to think of the door as "open or closed", in the real world we can't, because it could be partially open in varying degrees.

What's a...

Binary:

"bi" means two. Therefore this is a number system that only has two characters available to represent a value. The decimal number system that we use every day has 10 digits (0-9). The Octal number system (as the name implies) only uses 8 characters, and the Hex number system uses 16 (0-9,and the letters a-f).

Computers, because they are a digital device "speak" the binary language. Anytime you see computer data that is not just 1's and 0's, it has undergone a conversion process (probably to a different number system such as base 10). This conversion is for human benefit – so that it's easier for us to understand the value shown.

This presents some interesting challenges to a microcontroller (or for that matter, any digital) device because they only recognize two distinct "states" – 0 or 1. This is known as the binary number system. Many of you may already be familiar with binary. In fact, all of our experiments thus far have had some connection to the binary system.

Looking at the real world door example, we could assume that the closed state of the door could be considered a binary "0". Likewise, the fully open state could be considered a binary "1". If we were to hook up some input sensors, such as contact switches, to the door frame and then connect these switches to the BASIC Stamp, then the BASIC Stamp would be able to detect when the door was fully open or fully closed.

However, in the time between when the door leaves the fully closed condition and travels to the fully open condition, the BASIC Stamp is unable to detect or "follow" the door as it is in the process of opening. With the exception of the two end points (of the swing of the door) there is no data available to the BASIC Stamp – it's blind.

What we need to do is create an electrical interface that allows the BASIC Stamp to follow the progress of the door as it is opening. Then, not only can the microcontroller verify that the door was successful in opening all the way,

What's a State:

is simply a mode that a device can be in. For example, a binary device only has two possible "states" – high or low (1 or 0).

There are other devices that are considered "digital" but are called "tri-state". They're actually still a binary device, however they have the ability to turn their outputs "off" (not just "low"), causing them to be in what's called a "high impedance state". This third state really isn't in addition to two possible output conditions, it's causing the device to essentially (although temporarily and not physically) remove itself from the circuit.

but it would also have the ability to monitor and control how far the door should be opened. For example, let's say that you want to have the door open a little bit for ventilation. Your circuitry, under program control could cause the door to open to say 1.13".

By creating this circuitry (that enables the BASIC Stamp to see the current **state** of an analog condition) we're doing what is called Analog to Digital Conversion.

In this experiment we're going to design a hardware circuit utilizing an integrated circuit known as an "8 bit Analog to Digital Converter".

Parts Required

For each experiment, you need an IBM-compatible PC running DOS 2.0 or higher, Win95/98/or NT4.0. For Experiment #1 you will need the following:

(1) AD0831 – see Appendix B for data sheet
(1) 100K potentiometer
(1) 270 ohm ¼ watt resistors
(1) LED and 220 ohm resistor

Create the circuit as shown in Figure 1.1.

Build it!

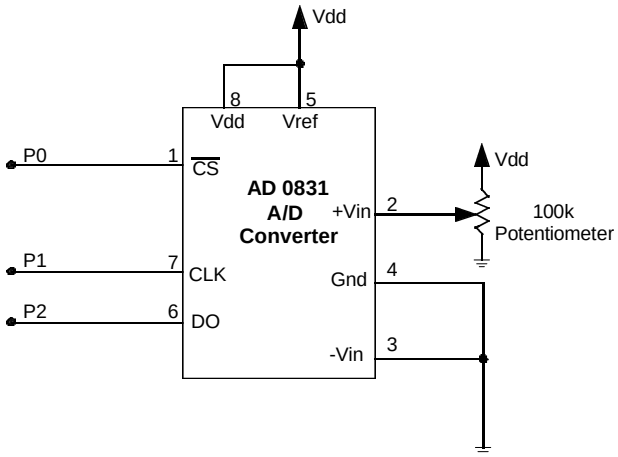
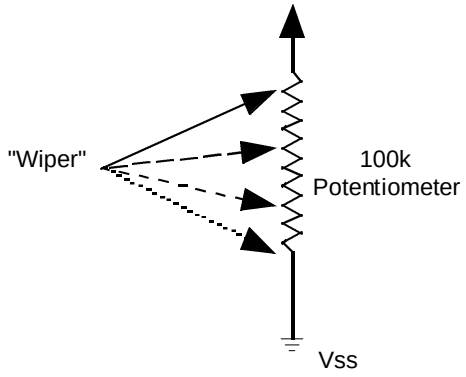


Figure 1.1: AD 0831 Circuit
The AD 0831 pin numbers are oriented in a counter-clockwise direction on the chip itself, starting from the top left-hand corner.

Be sure to pay careful attention to the location of pin 1 on the AD0831. We've used a potentiometer before, but in that instance we used it as a "variable resistor". Recall that a potentiometer is really nothing more than a resistive element that changes its resistance as you rotate the shaft that glides across a (usually) carbon element. In this project the potentiometer will be our source of the analog voltage.

Take a look at Figure 1.2. This is the schematic representation of a potentiometer. In this experiment we'll be using all three terminals of the "pot". You can see that one side of the resistive element is connected to Vdd and the other side is connected to Vss. As the wiper is rotated in one direction it gets closer to Vdd (and conversely) and gets farther away from Vss. Since the potential between Vdd and Vss (on the Board of Education) is 5 volts, it stands to reason that the closer the wiper gets to the 5 volt side (Vdd) of the element, the higher the voltage will be as measured at the "wiper" terminal (with reference to Vss).

Figure 1.2: ADC 0831
One terminal of the potentiometer is Vdd and the other is connected to Vss..



In this circuit, the potentiometer is acting as a variable "voltage divider"- a way of saying that the voltage present on the wiper terminal is continuously variable. We'll be using the potentiometer as an analog sensor for our simulated "real world" door. If you have access to a volt-ohm-meter (also called a VOM), take a moment and check out how the voltage on the wiper terminal changes as you turn the shaft. It should go from nearly 5 volts all the way to 0, and an infinite number of values in between.

Program it

After you are sure that the circuit is properly constructed, connect power to the Board of Education, attach the programming cable to your PC, and start the BASIC Stamp Editor.

Type in the following program:

```
a var bit
b var bit
c var bit
d var bit
e var bit
f var bit
g var bit
h var bit
z var word
high 0

here:
low 0
pulsout 1,210
pause 1
pulsout 1,210    'clock pulse
a=in2            'read input
pulsout 1,210    'clock pulse
b=in2            'read input
pulsout 1,210    'clock pulse
c=in2            'read input
pulsout 1,210    'clock pulse
d=in2            'read input
pulsout 1,210    'clock pulse
e=in2            'read input
pulsout 1,210    'clock pulse
f=in2            'read input
pulsout 1,210    'clock pulse
g=in2            'read input
pulsout 1,210    'clock pulse
```

```
h=in2               'read input

debug cls
debug "8 bit binary value is:",dec a,dec b,dec c,dec d,dec e,dec f,dec g,dec h

pause 100
high 0
goto here

debug ? z
high 0
goto here
```

Press Alt-R to run the program.

If your program and hardware are operating properly, then as you rotate the pot back and forth, you should see a string of (changing) 1's or 0's.

Lets look at the program in detail:

The first section is simply defining a whole bunch of single bit variables:

```
a var bit
b var bit
c var bit
d var bit
e var bit
f var bit
g var bit
h var bit
z var word
```

except of course for the last one which is one word in size.

high 0

This command causes P0 to go high. To understand why it needs to go high we need to look at the pinout of the AD0831. P0 of the BASIC Stamp is connected to pin 1 of the AD0831 A/D converter. Pin 1 is the AD0831's "**chip select**" and it is considered an "active low" pin. That is, that in order for the chip to be selected, there must be a low signal present on that pin. In our code (at this stage) we don't want the chip to be selected (or activated), therefore the first thing that we do is set it high which, of course is the binary equivalent of "1".

What's a "chip select":

With many different chips in any one system, there is usually at least one "master control" device, and in our Experiments, it's the BASIC Stamp. This controller needs to be able to enable or disable other components within the circuit, sort of like turning an LED on and off.

In most integrated circuits (i.e., AD0831) the "chip select" signal is designed to effectively activate or de-activate a particular process or condition. Depending upon the circuit design, several chips can be selected at any one time, but in any event the control device has the power to turn them on or off via this signal.

here:

Simply a label for looping purposes

low 0

This means that the AD0831 is selected (or activated).

pulsout 1,210

Ok. BASIC Stamp P1 is connected to pin 7 on the AD0831. Pin seven is the "clock" pin. What does the clock pin do? Well, according to the AD0831 data sheet, the clock pin is the method by which you (or the BASIC Stamp) initiates an operation to occur within the A/D chip. We need to give the AD0831 one clock pulse to "get it ready". The **pulsout 1,210** command creates a 420 ms pulse, about what the AD0831 needs to receive to begin the conversion process.

Now, each additional time you give the A/D converter a clock pulse, a new piece of data is available. This "bit" of data is presented on pin #6 of the AD0831. Therefore each time we issue a "clock pulse" we need to "read" the status on pin #6. Since we've connected pin 6 (on the AD0831) to P2 on the BASIC Stamp, we simply need to read the value (an input), and we do that with the following commands:

```
pulsout 1,210          'issue the clock pulse
a=in2                  'read input
```

Now let's stop for a moment and look into the internal workings of our circuitry. The AD0831 is what's called an "8 bit" A/D converter. If you're familiar with the binary number system, the following information may be readily understandable.

8 bits is just what it sounds like. If a bit is a 0 or 1, then 8 bits would be any combination of 8 "0's" or 8 "1's". For example:

00000000
00000001
11101001
11111111

Each of these "8 bit" numbers is a binary equivalent of a decimal number.

Let's look at the first example: "**00000000**" is the 8 bit binary version of the decimal number "**0**". The next one, **00000001** is the binary equivalent of the decimal number "**1**". If you were to now go through all of the possible number combinations (of 1's and 0's) for an 8 bit binary number, you would come up with a total of 256 different numbers (2 to the 8th power). Each of these 8 bit values would represent the computer's binary version of a decimal number. The binary number "**11111111**" would be the decimal number "**255**".

Remember computers, whether a Pentium PC or a BASIC Stamp microcontroller, at their very lowest level always operate on the binary number system – either a 1 or a 0.

The AD0831 is taking an analog voltage (as determined by the 100K potentiometer wiper which is connected to AD0831 pin #2), and converting that analog value into a digital value. This digital value is then sent out as a series of bits that are input to BASIC Stamp P2.

As the conversion is taking place on the A/D chip, it continually approximates the value in a digital form. In reality, there are an infinite number of analog values that can be derived from the pot (1.1 volts, 3.089 volts, etc.). Since the A/D converter chip that we are using is only capable of 256 discrete binary values, this means that the entire range of voltage (0 to 5 volts) presented on its input pin is limited to 256 discrete steps (or values).

Using some simple math we see that the resolution of an 8 bit A/D converter is (in this situation):

5 volts / 256 bit combinations = (approximately) .02 volts

With this in mind, each time you twist the shaft of the pot, the converter approximates the analog value – it comes close, but is not an exact representation, because of the resolution constraints. Higher resolution converters are available, such as 12 and 16 bit (and even more), but because of their higher resolutions, they come with a higher cost as well. A 12-bit converter will give you a resolution of 2 to the power of 12, or 4096 "steps". This results in 5 volts / 4096 steps, or one step for every .0012 volts.

Ok, back to the program.

Each of our 8 variables are set up as one bit in size. And each time we send a clock pulse to the converter it gives us the next bit in the series. As the successive clock pulses cause the converter to approximate the analog value, the value of the bit presented on BASIC Stamp P2 changes. With each clock pulse, the A/D converter calculates the next bit. This method, in which the AD0831 makes a conversion, is called Successive Approximation.

Our program simply takes each bit, and stores it as a variable, for later use in the program (there are other ways to accomplish this, but for demonstration purposes this method will probably makes the most sense).

So, as you can see, the **pulsout 1,210** command creates a single pulse that causes the converter to approximate the value of the analog voltage presented on AD0831 Pin 2, and then we read the next state of P2 on the BASIC Stamp. The following code brings in the remaining 7 bits of our 8 bit conversion

```
pulsout 1,210          'clock pulse
b=in2                  'read input

pulsout 1,210          'clock pulse
c=in2                  'read input

pulsout 1,210          'clock pulse
d=in2                  'read input

pulsout 1,210          'clock pulse
e=in2                  'read input

pulsout 1,210          'clock pulse
f=in2                  'read input

pulsout 1,210          'clock pulse
g=in2                  'read input

pulsout 1,210          'clock pulse
h=in2                  'read input
```

Serial:

Serial data transmission is a very efficient method to transfer data between hardware devices – it requires a couple of signal wires. It's strength (few wires) is also it's drawback. Data must travel sequentially, resulting in a slower data rate.

If you need higher speed (actually greater data "throughput"), you can usually increase the data rate, but depending upon the type of wire (or transmission medium) there are practical limitations as to how fast you can go.

If on the other hand you maintain the data (clock) speed, but use a parallel method of data transmission, your data throughput is typically 8 times faster because you're using 8 separate wires.

Of course, if we used a parallel A/D converter in this experiment, then we would use up 8 BASIC Stamps I/O lines just to get the data in. Maybe for this experiment, we'd have enough pins, but in a real world application (and since we're not worried about speed right now) It's better to take the slower way, and save those precious I/O pins for additional interfacing requirements.

This is a very simple form of **serial** communications. Serial data implies that the data comes over a single pin, analogous to a train on a set of tracks. There is only one set of tracks but there are many cars on the train. In order for us to see all the train cars, we have to watch it go by in a serial "stream". The opposite of serial is parallel, wherein the data (in our case, all 8 bits) are presented simultaneously on 8 different pins. This would be like watching a car race – all the cars go by at the same time – you aren't restricted to a single "track" (or I/O pin).

According to the data sheet for the AD0831, the first bit that comes out of the converter after we begin our clock pulses is the MSB, an acronym for Most Significant Bit. Looking at the following example, the "1" on the far left side is the MSB.

10010010

So what's the MSB? Think in terms of the decimal number system. Take the number 38. The "3" represents 30, and the "8" represents 8 – yielding a total value of 38. The value of 30 is more significant than the 8, because of its position in the string of numbers itself. If the same two digits were reversed (83) then the "8" has more significance than the "3", because it represents the "lion's share" of the value. It's the place in numerical sequence that determines whether it's the MSB or the LSB, or someplace in between.

The binary number system operates in the same manner. The MSB carries the most significance because it represents the largest value in the string of numbers. One could also define the MSB as the leftmost digit that has a value.

However, instead of each position being worth 10, 100, 1000, etc., each position is valued as follows:

Bit =	X	X	X	X	X	X	X	X
Value =	128	64	32	16	8	4	2	1

So, as an example, in the binary value of 10100101 the MSB (the left-most bit) has a "weight" of 128. The next bit (if it were a "1") would carry a weight of 64. The next, a value of 32, and so on.

128+0+32+0+0+4+0+1 or decimal 165.

Now that we received all eight of the bits (stored as individual variables), simply print the value on your screen using the debug command, and go do it again.

```
debug ? z
high 0
goto here
```

Let's modify the program as follows:

```
a var bit
b var bit
c var bit
d var bit
e var bit
f var bit
g var bit
h var bit
z var word
high 0

here:
low 0
pulsout 1,210

pulsout 1,210    'clock pulse
a=in2            'read input
pulsout 1,210    'clock pulse
b=in2            'read input
pulsout 1,210    'clock pulse
c=in2            'read input
pulsout 1,210    'clock pulse
d=in2            'read input
pulsout 1,210    'clock pulse
e=in2            'read input
```

```
pulsout 1,210     'clock pulse
f=in2             'read input
pulsout 1,210     'clock pulse
g=in2             'read input
pulsout 1,210     'clock pulse
h=in2             'read input

z= (a*128)+(b*64)+(c*32)+(d*16)+(e*8)+(f*4)+(g*2)+(h*1)

debug cls

debug "8 bit binary value is:",dec a,dec b,dec c,dec d,dec e,dec f,dec g,dec h
debug cr, "Decimal equivalent is:",dec z

pause 100
high 0

goto here
```

Using our weighting system from above, the BASIC Stamp can now make a simple calculation. The program now calculates the decimal value based on this binary information, and displays the value in both number systems.

So what have we accomplished? Well, as an example, the potentiometer could be connected to a door hinge. Each time the door is opened, there is a digital value available to the BASIC Stamp for interpretation. Using this new data, the BASIC Stamp can turn on a motor and open the door up "half-way", or more precisely, when the digital "value = 128" then stop the drive motor.

By interfacing a specialized IC like the AD0831 to the BASIC Stamp, we have given the microcontroller additional capabilities beyond trying to interpret the world as a binary environment. By using progressively higher and higher resolutions of A/D converters, we can attain very precise digital representations of an analog voltage device.

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 A/D converter in this experiment?

3. What does "chip select" mean?

4. Describe how the pot is used differently in this experiment, as opposed to being used as a variable resistor.

5. Why do we need to convert analog signals to digital values?

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

2. Write a program, that creates a "safety zone", that is between say 1.0 volts and 2.0 volts. However, if the analog voltage goes outside of these boundaries, an LED blinks.

3. Write a program that will display the analog voltage in decimal (base 10) and binary (base 2).

4. Draw the complete schematic for Challenge #2, 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.

What have I learned?

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

binary

decisions

digital

interfacing

data

switch

control

A microcontroller can only make _____ based on what _____ it receives. Many times the data from other electronic devices doesn't present itself in a _____ fashion (such as a _____ closure). Analog to digital converters are a very important tool in the design engineers toolbox. They give _____ devices the ability to _____ (or at least monitor) processes such as a door opening to a specific point.

The whole world of A/D conversion is complicated by the fact that we must understand many different facets of the _____ discipline, and this will always be a learning process.

Why did I learn it?

You may not realize it, but analog to digital conversion is going on all around you.

Do you like music? Remember the old LP phonograph record? That was an analog device. As the record spun 'round and 'round, the slight imperfections in the surface of the record were picked up by a small "needle". These "imperfections" were amplified & output through a speaker. There were lots of "hisses" and "pops", because the needle picked up every single imperfection there was, even if it was an (unwanted) spec of dust. This was a completely "analog" system.

If you don't listen to LP's, you probably listen to Compact Discs (CD's). A CD is a digital device that reads 1's and 0's into a very fast microcontroller and re-creates the music by outputting it through a speaker. So where's the analog? When you connect a pickup coil to your guitar, or sing into a microphone – that's analog. This analog signal then goes through a conversion process very similar to what we've experimented with here. The only difference is that the equipment and processors are bigger, more expensive, and have higher resolution.

But understand this... if it were not for the A/D conversion we couldn't get that song pressed onto a digital CD, and you wouldn't be able to enjoy "hiss" free music.

How can I apply this?

The field of Analog to Digital Conversion is an industry in itself. There are many things to consider when attempting to connect analog devices to a digital microcontroller. In this Experiment we've only scratched the surface of the whole conversion process & in a future Experiment we'll look deeper into the process, and the many different nuances A/D conversion has.

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 "solutions" level, there will always be a need for creative analog interfacing – simply because the world isn't black & white (binary)!

The neat part about microcontrollers (and something you may consider as a future career), is that the world of "smart devices" is expanding at an incredible rate, and doesn't show any sign of slowing down. As technology advances in all areas of our lives, we're surrounded by an ever-increasing number of technologically advanced machines and gadgets. You can help develop these, and perhaps invent the next "great product", or just have fun, building "stuff". The technology is the same, it's just applied differently!

Experiment #2: Basic Digital to Analog Conversion

By now we're well aware that a microcontroller is a "digital device". The BASIC Stamp operates on the binary number system, that is "1's" and "0's".

The world however, is rarely "binary" in nature. There are many instances where a microcontroller must know what a particular analog voltage is on a given device or sensor.

Utilizing an IC, such as the AD0831, analog to digital conversion became quite simple as we've seen. The real "hard part" is actually handled inside the chip – and is done automatically for us. All we need to do is give the AD0831 certain signals (generated by the BASIC Stamp), and the converted value was available as a serial output "stream". This resulting 8-bit number can then be used to represent digital values over a certain span (the range of the analog measurement).

What's an...

Op-amp:

A complete electronic circuit, usually contained in one integrated circuit. The most popular op-amp is probably the "741". It is a very simple to use circuit that has many applications in the linear (analog) world. Op-amps can be used to amplify the voltage levels of many different types of devices, such as microphones, pressure transducers or radio signals. Many op-amps require a special type of power supply that can deliver a "plus and minus" voltage, in relation to the circuit ground. Our BOE has only a "single ended" power supply, so we're using a slightly different version of op-amp (the LM324) that only requires the presence of +5 volts.

What about going the other way? Although it may not be as common, there are many situations in which you might want to have a digital system generate an analog voltage *output*. An example might be to control the intensity of a lamp or speed of a motor.

In this experiment, we're going to enable the digital BASIC Stamp to generate an analog voltage. We're also going to learn about a very popular integrated circuit called an "operational amplifier" or **op-amp**. This is a "linear" (analog) type of device, belonging to the same family as the 555 timer chip that we used in "What's a Microcontroller?".

Dig out your Board of Education, collect the parts listed below and let's "go the other way" - convert from digital to analog.

Parts Required

Experiment #2 requires the following parts, but is built in small steps so that the first schematic does not actually use all of the components:

(10) 2K ¼ watt resistors
(8) 1K ¼ watt resistors
(1) 270 ohm ¼ watt resistor
(1) LM324CN op-amp DIP format
(1) LED

Build It!

Create the circuit as shown in Figure 2.1. Pay careful attention to the values of the resistors and their locations in the circuit. This circuit is called a "resistive ladder D/A converter". The name comes from the fact that the resistor network (as drawn in the schematic) actually looks like a "ladder".

Important tip: later in this experiment we will modify this circuit by adding the 14-pin DIP LM234CN IC towards the top of the breadboard. Build this portion of the circuit on the bottom half of the breadboard.

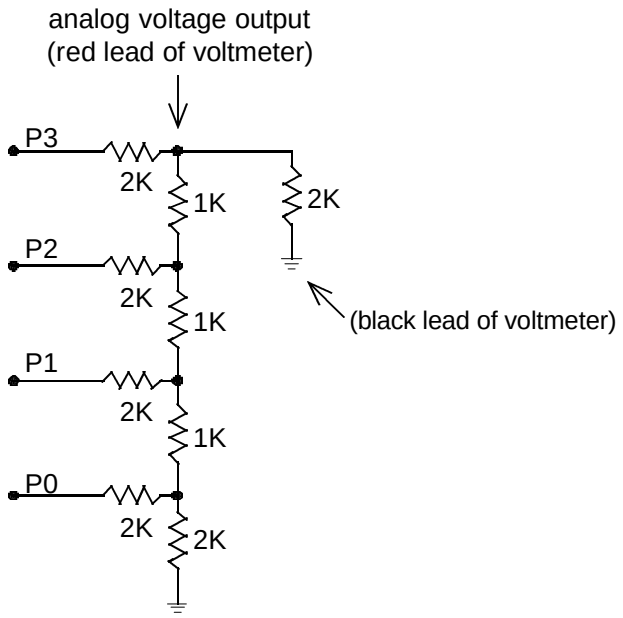


Figure 2.1: Resistive Ladder D/A Converter Schematic

What's a...

Linearity:

Since we're using resistors with different values (and tolerances), the binary output may be exact, but the analog voltage could vary (from one value to the next) because of slightly different values in the resistors. For example, our circuit may in fact generate an analog value within 5% of our desired amount. This is due to the tolerances of the resistors. We could use more expensive resistors (tighter tolerance) in the circuit, or we could use a commercially available DAC. DAC's are typically very accurate, and can deliver an accurate "linear" voltage over the complete D/A span.

We're creating a "4 bit" D/A converter in this experiment. You can see that with a little extra effort, we could add 4 more bits and have an 8-bit version.

Aside from doing the opposite of what the AD0831 A/D converter did, this circuit also has other significant differences.

The first is the fact that we're using very inexpensive components to accomplish the conversion. The type of resistors we're using are cheap. Whereas, the AD0831 is significantly more expensive.

Secondly, the number of bits used in the conversion (in this case 4), are all presented to the "converter circuit" simultaneously. That is to say that the binary data to be converted to an analog voltage arrives at the "converter" (portion of the circuit) as parallel data.

The AD0831 is a serial device. Recall that we had to "clock" the digital data out of the converter in a sequential manner. The advantage of this is that we only used one I/O pin to bring the data into the BASIC Stamp. Serial, however is considerably slower than an equivalent parallel device because of its sequential nature.

There are commercially available digital to analog converters (known as DAC's). They come in both serial and parallel versions, and have a very high degree of **linearity**. In this experiment, we're not so much concerned with a high degree of accuracy, as we are interested in learning the basics of a DAC conversion.

Program It!

Connect the cables, power up the Board of Education, and the BASIC Stamp Editor. Type in the following program:

```
output 0
output 1
output 2
output 3

out0=1
out1=1
out2=1
out3=1

stop
```

What's a...

Addressed:

When we write data to the EEPROM (using the WRITE command), we are placing a binary value into a memory "cell". This cell must be selected or "addressed" by our program. For example, "write 5,7" writes the value of "7" to the selected (addressed) memory location "5". The BASIC Stamp's I/O pins are actually a form of "memory location" as well. The primary difference being that we can attach devices to these "cells", and by writing a value, cause an action to occur (i.e. blink an LED). In any event, the selection of the appropriate group of I/O pin(s) is known as "addressing".

Run the program. If your circuit is operating properly, the voltmeter should read approximately 3.0 volts. Now change all the "out" statements to be equal to "0" & re-run the program. The voltage should be approximately 0 volts.

By now we're quite familiar with the "output" and "out" PBASIC commands. "Output" causes the corresponding bit to be an output. Once the bit has been "set up" to be an output, then we give the bit a value (0 or 1) using the "out" command, effectively turning the bit on or off.

Until now, we've been addressing each of the I/O lines as a single bit. This works well whenever you need to have tight 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.

Since each of our I/O lines (P0 through P3) are used as outputs in this experiment, it would be easier and more efficient to have a method of addressing groups 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

What's a...

Optimize:
The art of enhancing your code so that it will run quicker and require less memory. Always attempt to use the fewest lines of code to accomplish a given task. This will also make it easier to debug (or even enhance) your program. Optimization becomes more important with larger (and more "critical") applications. The BASIC Stamp (or any other microcontroller) has a limited amount of memory – use it sparingly.

more complex programs, you may find yourself looking for ways in which to **optimize** your code.

Until now, we really haven't addressed one of the most important aspects of programming – efficiency.

Efficiency and optimization can be important when your program sizes increase and you don't want to add more hardware. You should always be looking for ways in which to reduce the number of lines of code to accomplish a certain task. In our example, in order to make 4 I/O lines outputs, and set to all "1's" we used the following:

```
output 0
output 1
output 2
output 3

out0=1
out1=1
out2=1
out3=1
```

Eight lines in all. Maybe you're thinking about using the "high" and "low" commands? The single command **high** actually accomplishes the two tasks (set direction to "output" and set value as "1") simultaneously. Using **high** our code would look like this:

```
high 0
high 1
high 2
high 3
```

Our code has been "optimized". We've reduced the number of lines from eight down to four. Not bad, but thanks to additional PBASIC capabilities, we can do even better.

Recall in our previous experiment (Basic Analog to Digital) that the BASIC Stamp operates on the binary number system. A "bit" is a single binary bit. We also know that a "byte" is a group of 8 bits. And of course, a "word" is a group of 16 bits.

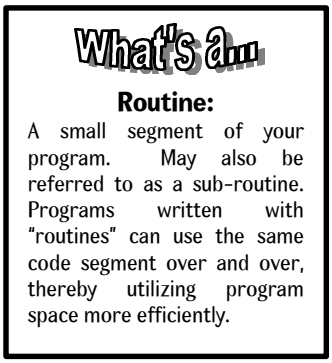
Well, if a "bit" is one, and a "byte" is 8 bits, what do you call a group of 4 bits? A "nibble". Who came up with these names anyway? (Maybe "word" should have been called "meal" instead?)

Certain commands in PBASIC allow us to directly address its I/O lines as (16) individual bits, (4) separate 4 bit nibbles, (2) 8 bit bytes, or (1) 16 bit word.

Remember that there are two registers that we need to set that will allow a certain output on a specific I/O line. The first register is called "direction". The command "output" sets the direction as an "output". Conversely, "input" sets the I/O line up as an input.

The second is the "data" register. If the I/O pin has been set up as an output, then this register's "data" can be set to either a 1 or 0.

Ok, we're dealing with 4 bits which is called a nibble. The four bits that we're using are P0-P3. According to the BASIC Stamp Manual (which you should have a copy of by now - it's free at www.stampsinclass.com) the group "P0-P3" is called nibble "a". The next set of four (P4-P7) is nibble "b", and so on.



Using this information, we can replace the previous piece of code with a **routine** like the following:

dira=15
outa=15

The command "**dira=15**" simply states that nibble "**a**" (consisting of P0-P3) should have all 4 of the pins set up as outputs. How does it do this? Let's look deeper.

Since we're dealing with a (4 bit) nibble, and we want to set up all four as "outputs", we need to have a "1" placed into each of the (P0-P3) I/O line direction registers. If there is a "0" in the direction register, then the pin is an input.

A nibble can have the following bit combinations:

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6

0111 = 7
1000 = 8
1001 = 9
1010 = 10
1011 = 11
1100 = 12
1101 = 13
1110 = 14
1111 = 15

Each bit in the above (binary) nibbles corresponds to the appropriate data direction bit as shown below:

Bit	**0**	**0**	**0**	**0**
I/O pin	**P3**	**P2**	**P1**	**P0**

Therefore, if we used the command "dira =4" the following direction register bits would be set:

Bit	**0**	**1**	**0**	**0**
I/O pin	**P3**	**P2**	**P1**	**P0**

This would result in I/O pin P2 being set as an output, and all the other pins (P0,P1,P3) set as inputs.

Hence the command, **dira=15** (by virtue of the fact that all four bits are a "1") sets up each of the I/O lines as outputs.

Now, the second line of the program should be relatively easy to understand:

outa=15

It operates in exactly the same (binary) manner. All four of the data output register bits are now set to a "1".

Try the program:

```
dira=15
outa=15
stop
```

Your result should be exactly the same as the "8 line" program that we started with. Therefore, not only do we have to "type less", but we don't waste valuable program memory (usually limited on microcontroller systems), and the program actually executes faster.

Remember, optimization and efficiency – characteristics of great code!

Ok, back to our D/A experiment. Replace the program as follows:

```
dira=15
outa=0
stop
```

Now connect a voltmeter (using the red or "positive" lead) to the "voltage output" point as shown in Figure 2.3. The black lead (negative) of the voltmeter should be attached to Vss ("ground"). If everything is operating properly, you should measure approximately "0" volts.

Change the second command to "**outa=15**". Your VOM should now measure approximately 3.0 volts. If not recheck your circuitry, being especially careful that the resistor leads aren't shorting against each other.

With a properly operating circuit, change the command "**outa=x**" (re-run the program each time) with "x" equal to each of the (16) available values. If you know how to use a "**for . . . next**" loop and the "**pause**" command this task could be a bit easier than reprogramming the BASIC Stamp 16 times. Write down the measured voltages:

Output nibble value:	**Analog Voltage measured (approx.):**
Outa=0	0 volts
Outa=1	_____
Outa=2	_____
Outa=3	_____
Outa=4	_____
Outa=5	_____
Outa=6	_____
Outa=7	_____
Outa=8	_____
Outa=9	_____
Outa=10	_____
Outa=11	_____
Outa=12	_____
Outa=13	_____

Outa=14
Outa=15 3.0 volts

If everything is operating properly, the analog voltage should gradually increase as you increase the "**outa**" value.

We've just created a 4-bit digital to analog converter using very inexpensive components.

Because of the manner in which the resistive divider operates, the circuit will never be able to attain the full 5 volt span (of the circuit's operating voltage). Therefore, the highest analog voltage that we can achieve is approximately 3.0 volts (with these components).

If we were to use a commercially available D/A convertor, we could set the span (similar to how we'll do it in our next experiment – Advanced Analog to Digital) to any value we desired.

There are however, many applications that utilize this "resistive ladder" method to produce analog values from a digital device.

Add the LED as shown in 2.2.

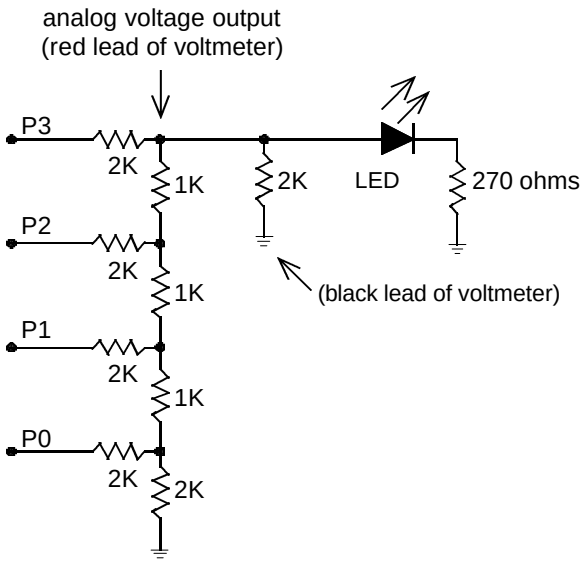


Figure 2.2: Add the LED into your circuit

Run the following program:

```
dira=15
outa=15
stop
```

The LED should be lit. However, take a look at your voltmeter. What voltage do you see? Remember, with all 4 bits of the nibble set to "1", you should be measuring approximately 3.0 volts.

What's the voltage?

It's probably considerably less than 3.0 volts because of the "load".

What's a...

Amperage:
This refers to the flow of electrons in a circuit. Think of "amps" as the number of gallons of water passing through a pipe. The higher the amperage requirement of the circuit, the greater the "load". Also known as "current".

A "load" is a device in a circuit that uses power. The portion of the circuit supplying power to the LED, is only capable of providing a limited amount of **amperage**. The LED is attempting to draw too much current from our D/A converter. Therefore, the "load" imposed by the LED is too great for proper circuit operation.

The voltmeter also represents a load to the D/A converter. The voltmeter draws such a very small amount of current that the D/A converter circuit really doesn't even know it's "connected".

The output voltage of the converter will drop if the load is too heavy, therefore we need to **buffer** the D/A converter.

What's a...

Buffer:
An electronic circuit component that senses one voltage on its input, and creates a duplicate voltage on an output pin. The component (in our case, an op-amp) has increased drive capability. This allows us to connect a load (such as an LED) to the D/A converter circuit, without disrupting proper circuit operation.

The LM324 is a chip that belongs to the "linear" (analog) family of integrated circuits. The LM324 is called an "operational amplifier", which means that it can be used to amplify (generally) analog signals. It can also be used in many different applications that require some form of "buffering" (such as is our requirement here).

Inside this single 14 pin dip IC (see the "pinout" in Figure 2.3), there are actually 4 separate "op-amps". Each one can be connected to a separate portion of a larger circuit and used completely independently of its (internal) neighbors.

The LM324 (unlike many other op-amps) requires only a single 5 volt supply (like that which is available on the Board of Education).

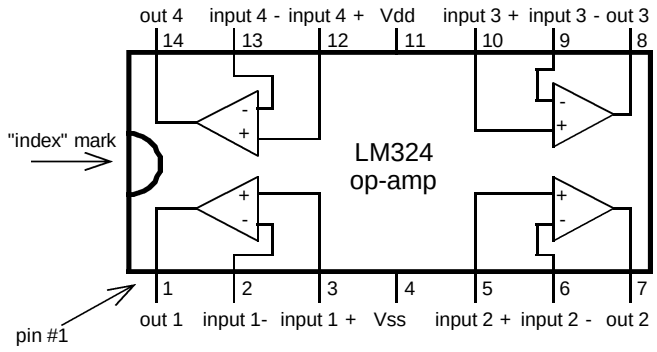


Figure 2.3: LM324 op-amp
Note the location of pin #1 and the index mark.

Add the additional circuitry to your board, as shown in Figure 2.4. This type of circuit (using the op-amp) is called a "voltage follower" or more simply, an analog buffer. The voltage present on the input, is the same voltage available on the output – but with the addition of the LM324 op-amp, the output current capability is enhanced.

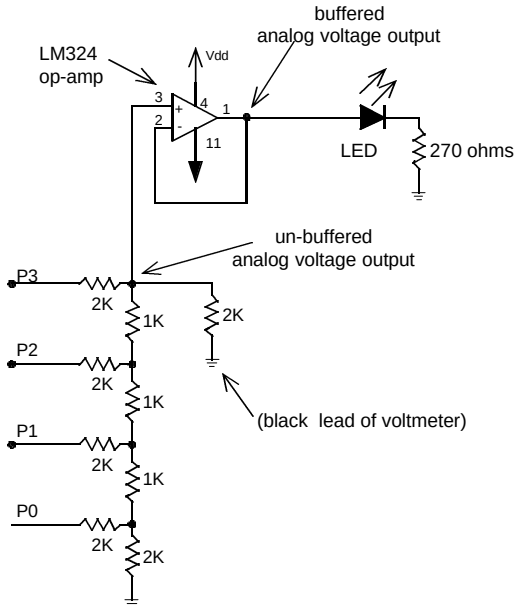


Figure 2.4: LM324 op-amp Schematic
Add the LM324 into the circuit as shown in the schematic

Now when you run the program:

```
dira=15
outa=15

stop
```

You'll notice that the voltage input (on pin #3 of the op-amp), is back to 3.0 volts.

Try this. Remove the LED from the circuit. Notice that with or without a "load", the total measured voltage is still about 3.0 volts.

Replace the LED. Due to the increased drive current available to the LED (via the buffering of the LM324), we're not over-loading the circuit, and our D/A converter functions properly.

Ok, let's modify the program:

```
dira=15
outa=0

stop
```

Is the LED on? Of course not, the output voltage (measure it) is approximately zero.

Try this:

```
dira=15
outa=1
stop
```

The output voltage is not zero (if your circuitry is operating properly). It should be about .20 volts. But the LED isn't on at all. It's not even "dim". Why?

An LED is (as it's name implies) a form of solid state diode. A diode is a semiconductor device.

By now we should know what a conductor is. It's simply a device (such as our "jumper wires") that allows electrons to flow within a circuit.

Semi-conductor implies "somewhat or sometimes conductor". In the case of a typical LED (as we're using here), it won't conduct (at all) until a certain threshold voltage is attained. It's usually somewhere in the 1.2 to 2.0 volt range. Therefore, you'll notice that until this voltage is exceeded, the LED won't light up – not even a little bit. It's completely off until its threshold voltage is exceeded.

Short-circuit:

When your circuit has a "short", the power supply (the battery or wall transformer) is attempting to supply more amperage than it is (usually) capable of delivering. This is a faulty condition, and should be avoided. Besides destroying the components in the circuit, there is a risk of personal injury. Whenever somebody flips on a light switch (& the light doesn't come on), they may refer to this as a "short" circuit. It's usually not. It's probably an "open" – an incomplete circuit.

Now, once the threshold voltage has been reached, the LED becomes a **short circuit**. In order to prevent damage, we need to limit the amount of current flowing in the circuit. That is the purpose of the 270 ohm resistor – it restricts the amount of amperage (or current) to a safe level.

Therefore, an LED doesn't conduct until a certain voltage is reached, and then it'll become a short circuit and destroy itself unless there is some method of limiting the current.

Questions

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

2. What type of D/A converter did we create in this experiment and what are its limitations?

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

4. What's a nibble?

5. "1011" is what value in the decimal number system?

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.

3. Write a program that will only step through the analog voltages that "matter" to the LED. In other words, create code that will cause the LED to go from "off" to "full on" (as "on" as the LM324 will allow). Any voltages outside these boundaries should not be generated. The Program should make the LED brightness cycle over a 10 second interval.

What have I learned?

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

resistive ladder

interface

semiconductor

converting

span

drive

amperage

The analog world requires _____ circuitry between itself and digital microcontrollers. On the input side, the microcontroller must have some method of _____ an analog voltage into a digital value. This is called A/D conversion.

In this experiment, we've reversed the process. If the BASIC Stamp needs to output an analog voltage, there are several methods to choose from. We created a _____ D/A converter made from very inexpensive components. The maximum output voltage (due to limitations in this type of D/A converter) was about 3.0 volts. If we were to use a commercially available converter, then we could control the entire _____ (in our case 0-5 volts).

The resistive ladder converter circuit has very little "_____" capability. That is, the load needs to be very light. Even something as efficient as an LED causes the output voltage to drop.

To counteract this excessive load (imposed by the LED), we connected a buffer, which "followed" the analog voltage and increased "drive capability" (higher _____) enough to drive the LED.

An LED is a _____ device. There is a certain threshold voltage under which the LED won't conduct. Once this threshold voltage has been exceeded, the LED will turn on. Since the analog voltage present on the LED is controllable (with our 4 bit converter), we can increase the intensity of the LED by stepping up the analog voltage, which in turn increases the current flowing through the circuit. A resistor is required in the LED circuit to prevent *excessive* current (which would destroy the device).

Why did I learn it?

When designing circuits, its important to pay attention to the specifications of the devices that you're thinking of using. For example, if we need to have a voltage output capability (from our op-amp buffer) of more than 3.75 volts, then we should choose a different device. The LM324 was chosen for this Experiment because it is widely available and inexpensive. Plus, it doesn't require a more sophisticated (bi-polar) power supply.

The D/A converter we've developed in this experiment does not have the ability to span the full 5 volts (Vss to Vdd). However, there are commercial D/A converters available that will, but they cost more.

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.

How can I apply this?

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 an analog signal (from the microphone) that has been digitized (A/D) and stored on the CD media. This digital data is then output to an amplifier and speaker (after undergoing the D/A process).

The higher the bit resolution, the more realistic (and accurate) the sound (or music) reproduction. Think of how far we've come, from the phonograph record to the compact disc. It makes you wonder what's coming next.