Stamp Applications no. 18 (August '96):

Need Analog Output from the Stamp? Dial it in with a Digital Potentiometer

Using the DS1267 pot as a versatile digital-to-analog converter

by Scott Edwards

GETTING AN ANALOG VOLTAGE out of the Stamp is no problem at all. Just use the PWM (pulse-width modulation) instruction as shown in the manual, and you're done.

There are some drawbacks to PWM, however. The Stamps can't output PWM continuously, so you have to either buffer the output with analog circuitry, or write your program to output PWM as often as possible. PWM is also by definition noisy, since it's made up of a string of pulses. So it may not agree with some loads that require clean input voltages. Depending on the characteristics of your filtering and buffering circuitry, the PWM-generated voltage may not be completely linear in proportion to the input duty-cycle value. Finally, PWM is only readily convertible into an output *voltage*, not a resistance.

What we need is a do-it-all digital-to-analog converter (DAC).

My nominee for the do-it-all DAC title isn't a DAC at all—it's a digital potentiometer called a DS1267. This month will look at how to interface this goody to the BS1 and BS2, and how to use it in a variety of applications.

In BASIC for Beginners we'll look at two's complement—a method of working with negative numbers using integer math.

Stirring the Pot. A potentiometer, affectionately called a *pot* by techies, is a resistor with a movable contact whose position is controlled by turning a knob or sliding a lever. Moving the wiper changes the resistance between it and the fixed legs of the resistor. The closer the wiper is to a leg, the lower the resistance between them.

A pot's resistance rating is the total resistance between the fixed legs. The total resistance is also equal to the sum of the resistances from the wiper to each of the legs. I guess that's somewhat obvious, but it leads to a another conclusion: a pot is a great voltage divider.

When you place a pair of resistors across an input voltage, the voltage across one of them—call it R1—is the ratio of that resistance to the total resistance multiplied by the total voltage. In symbols: $V_{R1} = V_{TOTAL} * (R1/(R1+R2))$.

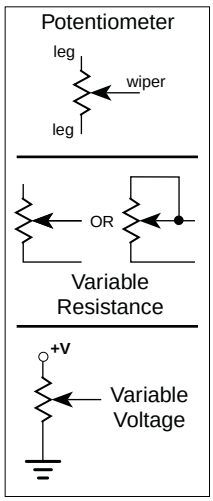


Figure 1.
A pot can be used to adjust resistance or voltage.

You can think of a pot as being two resistors—one above the wiper, and one below. The total of the two resistors is always the same; it's the rated resistance of the pot. When you move the wiper, one resistance goes up and the other goes down, but the total is fixed.

This makes the voltage-divider formula work out neatly, since R1 + R2 never changes. At the bottom of figure 1, labeled Variable Voltage, suppose the pot shown was rated at 10k (10,000 ohms). If the wiper is set so that there's 2k between it and ground, and +V is 5 volts, what's the output voltage between the wiper and ground?

$$V_{WG} = 5V * (2{,}000/10{,}000) = 1V$$

(V_{WG} is just my shorthand for "the voltage from wiper to ground.")

Yet another way of thinking about the voltage-divider characteristics of a pot is this: The output voltage at the wiper is proportional to the position of the wiper as a portion of its travel. In other words, in a circuit like the one at the bottom of figure 1, when the pot is set for 50% of its travel, the voltage out is 50% of +V.

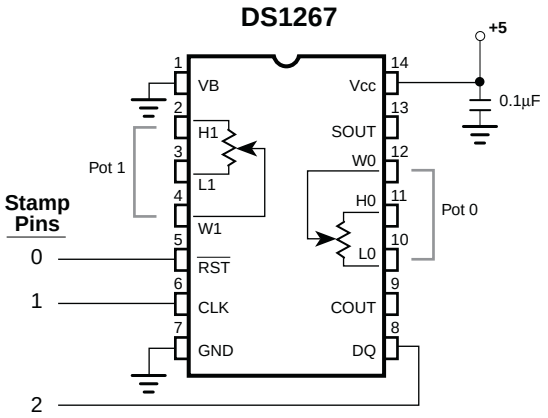
This applies only to *linear* pots; there are also *audio* pots designed for volume controls whose relationship of resistance to travel is warped to match human hearing.

To summarize: Pots can be wired for variable resistance or variable voltage. Variable voltage is achieved by wiring the pot as a voltage divider. With a linear pot, voltage divider output is proportional to the wiper setting.

A Digital Pot. Pots are so handy that it was inevitable that somebody would make a digital version. Dallas Semiconductor offers the DS1267 dual digital pot, shown in figure 2. It's available in three resistance ratings: 10k, 50k, and 100k. Typical price in single quantity is $5.

The DS1267 is easy to connect to a Stamp via its synchronous-serial interface. This is well-plowed ground by now, since we've featured quite a few serial devices (LTC1298 analog-to-digital converter, MAX7219 LED driver, DS1620 thermometer) that use this kind of interface. To recap, bits are sent one at a time on the DQ (data) line. The CLK (clock) line is pulsed to tell the receiving device when to grab the data bit. Then the next bit is sent the same way until all the bits the receiving device expects have been sent. So many parts use this kind of interface that the BS2 instruction set has it built right in.

So that's how you communicate with the DS1267; what do you say to it? Basically, you tell it two 8-bit settings for the two pots. Those values, 0 to 255, represent the wiper positions relative to the legs of the pots. For example, if you're using a 100k DS1267, and you give pot 0 a setting of 100, the resistance between the wiper and the lower leg of the pot will be (100/255) * 100k = 39.22k.



Explanation of DS1267 Pins

VB: Substrate bias voltage. Usually grounded, but can be connected to voltages as low as –5.5V for signals that go below ground

H0, H1: High end of pots 0 and 1

L0, L1: Low end of pots 0 and 1

W0, W1: Wiper of pots 0 and 1

RST: Reset pin; high to input new data, low to update pot settings

CLK: Clock to synchronize input of serial data

DQ: Serial data input

COUT: Cascade output; connect to next DS1267 DQ for daisychaining

SOUT: Stack output; common wiper output used when pots are stacked

Figure 2. DS1267 with hookup information for demo programs.

There's an additional bit in the DS1267 protocol that allows you to combine the two pots into a single, larger one. That bit is called *stack select*, and it simply determines which pot's wiper will be connected to the Sout pin. The idea is that you connect the low leg of pot 1 to the high leg of pot 0, and use Sout as the wiper connection for the combined pot. Send both pots identical 8-bit settings, and use the stack select bit as a 9th data bit. Voila! You get a single pot with double the rated resistance and twice the resolution (512 resistance steps instead of 256).

Listings 1 and 2 are sample BS1 and BS2 programs that control the DS1267. They don't use the stacking feature, so they don't bother setting the stack-select bit to a particular state; they just send one extra clock pulse to satisfy the DS1267's 17-bit protocol.

To demonstrate the DS1267, I wired its pots as voltage dividers (low legs L1 and L0 to ground; high legs H1 and H0 to +5 volts). I ran the programs in listings 1 and 2 and watched the wiper outputs on a digital oscilloscope. As figure 3 shows, you can plainly see the voltage climbing on one pot's output and falling on the other as their values are incremented and decremented, respectively.

If you're using a BS2 and you have an oscilloscope handy, you can use this setup to get a graphical look at the BS2's integer sine function. In listing 2, just replace the line beginning with DSpot0 = ... with the following:

```
DSPot0 = SIN DSPot1 + 127
```

Adding 127 to the sine value is necessary because the BS2 expresses sines as two's complement values with a range of ±127. (Two's complement is a way of representing negative values in binary. When you add the two's complement of a number to another number, it has the same effect as subtracting that number from the other. See this month's BASIC for Beginners.) Figure 4 shows the sine output.

You can use the DS1267's digital pots in pretty much any circuit that employs a mechanical pot. There are just a few limitations to bear in mind:

• All pots have some wiper resistance—additional resistance that looks like a resistor in series with the wiper. In mechanical pots, this resistance is generally too low to worry about. In the DS1267, it can be as high as 1000 ohms. This won't affect a voltage-divider circuit, provided that you keep current draw through the pot to a minimum. But in a variable resistance application, your minimum pot setting may be as high as 1k.

• All pots have some limit as to the amount of current they can handle safely. In the case of the DS1267, the limit is pretty low; 1 mA. Make sure that your circuit never draws more than this amount of current through the DS1267, or you risk damaging it.

• If the signal or voltage you plan to control with the DS1267 can be negative with respect to ground, you must connect the VB (bias) pin to a supply that's more negative; up to –7 volts.

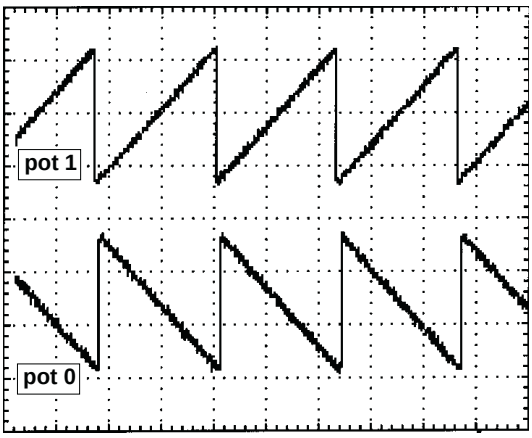


Figure 3. Oscilloscope trace of listings 1 and 2.

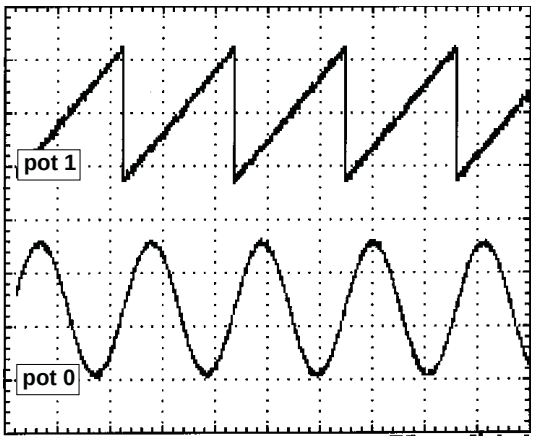


Figure 4. Listing 2 modified for sine output.

Other Digital Pot Applications. I could probably devote a half-dozen columns to potential applications for the DS1267, because anywhere there's a pot, there could be Stamp/computer control. That's a lot of territory. The audio and electronic music possibilities alone are mind-boggling. Not to mention electronic control and calibration of analog instruments, interfacing to conventional motor controls, management of old-fashioned 555 timer circuits, creation of automated test equipment, control of linear power supplies, etc.

BASIC for Beginners. The Stamps' 16-bit integers have a range of possible values of 0 to 65,535. They don't support larger values, decimal points, or negative numbers. That's what it says in the manual, and that's what I always say here.

But it's not entirely true. Any computer that can handle positive integers of a particular size can also work with negative numbers.

Let's start by defining a negative number. They take about 10 pages to do this in a math textbook, but I'm going to use a short and convenient definition: A negative number is what you get when you subtract the corresponding positive number from zero. For instance, –10 is the result of subtracting +10 from zero.

That definition is almost meaningless in human terms, but really profound when you're working with computers. Try this: on a BS1, run the following lines of code:

```
w1 = 0 - 10
debug "-10 is equal to: ",#w1,cr
w1 = 50 + w1
debug "50 + (-10) = ", #w1
```

The first debug instruction shows the number 65526, and the second one—the result of adding 50 and 65526—shows 40. So subtracting 10 from 0 gives us 65526, but adding 65526 to another number has the same effect as subtracting 10. Hey, adding 65526 works the same as –10!

Why does this work? Remember that the Stamp uses a limited number of bits, in this case 16, to represent numbers. When you count up and exceed the maximum number that the available bits can hold, the value wraps around to 0 and starts counting over. For example: 65533, 65534, 65535, 0, 1, 2... The same wraparound occurs when you count down. 2, 1, 0, 65535, 65534, 65533...

You can regard addition and subtraction as just special cases of counting up and down. To subtract 10, just count down 10 times. Or, since the numbers wrap around, you could count up 65,526 times to subtract 10. That's the basis for *two's complement* negative numbers.

These numbers are called two's complement because of the other way of calculating them: Take a number and invert the individual bits; that is, change all 1s to 0s and 0s to 1s. That's called a complement, or a one's complement. Add one to the one's complement, and you have a two's complement. The result is the same as subtracting the same number from 0.

There are a couple of peculiarities in the two's complement system. The first is that a two's complement negative number is only sure to act like a proper negative number within its original bit size. For example, the 16-bit version of –10 is 65526, but the 8-bit version is 246, and the 32-bit version is 4,294,967,286. The conversion is pretty easy; going from a larger bit size to a smaller one just requires lopping off the extra bits. This happens automatically when you put a 16-bit value into an 8-bit variable in PBASIC. Naturally, things fall apart if the 16-bit negative number is larger than an 8-bit variable can hold. The ranges of possible values are:

Four bits (nibbles):	±7
Eight bits (bytes):	±127
Sixteen bits (words):	±32767

To convert in the other direction—to move an 8-bit negative value into a 16-bit variable—you must *pad* the resulting 16-bit number with 1s. To do this properly, you must first determine whether the number is negative. Technically, an eight-bit two's complement number is negative if its most-significant bit is 1. In regular unsigned math, that bit is 1 when a number is greater than or equal to 128. If the 8-bit number is negative, then its 16-bit equivalent must have all of the upper 8 bits set to 1s. Here's the code

to convert an 8-bit, two's complement number into a 16-bit variable (PBASIC 1):

```
w1 = b2
if b2 <= 128 then skip
w1 = b2 | $FF00
skip:      ' Program continues.
```

The other peculiarity of two's complement is that there's always one outlaw value that is its own two's complement. This value has a 1 in the leftmost bit, and 0s in all the lower bits; for example, the 8-bit value 128 (%10000000 binary). What makes this value an outlaw is this: subtract 128 from 0 in an 8-bit integer. What do you get? Yep, 128. Obviously, we can't have a system in which +128 and –128 are represented by the same number. So programs that use two's complement should be written to regard these values as errors:

Four bits (nibbles):	4
Eight bits (bytes):	128
Sixteen bits (words):	32768

Although two's complement gives you a way to represent negative numbers in PBASIC, remember that you have to adjust your thinking. Comparison and math operations assume positive integers, so two's complement values won't always return the results you expect. And the debug/serial output instructions of the BS1 don't automatically handle the minus sign; you have to do that yourself. (The BS2 has functions for displaying numbers in signed decimal and hex formats.)

Sources

The DS1267 is available from Newark Electronics, phone 312-907-5436. You may also contact Dallas Semiconductor directly, phone 214-450-0448; fax 214-450-0470.

NOTE: This article was originally published in 1996. The Stamp Applications column continues with a changing roster of writers. See www.nutsvolts.com or www.parallaxinc.com for current Stamp-oriented information.

Listing 1. DS1267 Demo Program for BS1

```
' Program: DS1267.BAS
' This program controls the DS1267 digital potentiometer chip.
' This chip is very versatile as a digital-to-analog converter.
' It can output a variable voltage, can adjust current (up to
' 1 mA), or it can serve as the variable resistance in a
' resistor-capacitor timing circuit such as a timer or oscillator.

' Hardware interface with the DS1267:

SYMBOL RST     = 0     ' Pin number of reset connection.
SYMBOL CLK     = 1     ' Pin number of clock connection.
SYMBOL DQ_n    = 2     ' Pin number of data (DQ) connection.
SYMBOL DQ      = pin2  ' Pin variable of data connection.

' Variables used by the program:
SYMBOL DSpot0  = b2    ' Variable for setting of pot 0.
SYMBOL DSpot1  = b3    ' Variable for setting of pot 1.
SYMBOL DSpots  = w1    ' Word variable holding both pot values.
SYMBOL DSxfer  = w0    ' Word variable for transferring pot values.
SYMBOL clocks  = b4    ' Index variable for counting clock pulses.

let dirs = %00000111            ' Output pins 0,1,2 to DS1267.
' The loop below increments pot 1 in 10-unit steps from 0 to 255.
' by subtracting pot 1 value from 0 and writing that to pot 0,
' it makes pot 0 the inverse of pot 1. In other words, as pot 1
' increases, pot 0 decreases.
Begin:
for DSpot1 = 0 to 255 step 10   ' Pot 1 increasing: 0 to 255.
  let DSPot0 = 0 - DSPot1       ' Pot 0 decreasing.
  let DSxfer = DSpots           ' Store data in transfer variable.
  gosub outPot                  ' Send to the pots.
next                            ' Next value for pots.
goto Begin                      ' Repeat endlessly.
```

```
'====================DS1267 SUBROUTINE====================
' This code shifts data out to the DS1267. Because the shift
' process causes the data to be lost, we use a copy of the
' data to perform the transfer (DSxfer). The DS1267 expects
' a total of 17 bits: first the stack-select bit, which
' selects wiper 0 or wiper 1 for connection to Sout; then
' 16 bits representing the 8-bit values of pots 1 and 0,
' most-significant bit (msb) first.
outPot:
high RST               ' Take RST high to start transfer.
low DQ_n               ' Set stack-wiper to 0.
pulsout CLK,10         ' Pulse the clock line.
for clocks = 0 to 15   ' Now send 16 data bits.
  let DQ = bit15               ' Put msb on data line.
  pulsout CLK,1                        ' Pulse the clock
  let DSxfer = DSxfer * 2      ' Shift 1 bit to the left.
next                   ' Repeat for all 16 bits.
low RST                ' Take RST low to finish transfer.
return                 ' Return to program.
```

Listing 2. DS1267 Demo Program for BS2

```
' Program: DS1267.BS2
' This program controls the DS1267 digital potentiometer chip.
' This chip is very versatile as a digital-to-analog converter.
' It can output a variable voltage, can adjust current (up to
' 1 mA), or it can serve as the variable resistance in a
' resistor-capacitor timing circuit such as a timer or oscillator.

' Hardware interface with the DS1267:

RST     con 0          ' Pin number of reset connection.
CLK     con 1          ' Pin number of clock connection.
DQ_n    con 2          ' Pin number of data (DQ) connection.

' Variables used by the program:
DSpots  var    word             ' Word variable holding pot values.
DSpot0  var    DSPots.lowbyte   ' Variable for setting of pot 0.
DSpot1  var    DSPots.highbyte  ' Variable for setting of pot 1.

DIRA = %0111           ' Output pins 0,1,2 to DS1267.

' The loop below increments pot 1 in 10-unit steps from 0 to 255.
' by subtracting pot 1 value from 0 and writing that to pot 0,
' it makes pot 0 the inverse of pot 1. In other words, as pot 1
' increases, pot 0 decreases.
Begin:
for DSpot1 = 0 to 255 step 10   ' Pot 1 increasing: 0 to 255.
  DSPot0 = 0 - DSPot1           ' Pot 0 decreasing.
  gosub outPot
next                            ' Next value for pots.
goto Begin                      ' Repeat endlessly.

'====================DS1267 SUBROUTINE=====================
' This code shifts data out to the DS1267. Since it uses
' the Shiftout instruction, which does not alter the variable
' being shifted, we don't have to make a copy of the pot data.
outPot:
  high RST             ' Take RST high to start transfer.
  pulsout CLK,1        ' Pulse for stack-select bit (don't care).
  Shiftout DQ_n,CLK,msbfirst,[DSpots\16]        ' Shift out pot values.
  low RST              ' Take RST high to end transfer.
return
```