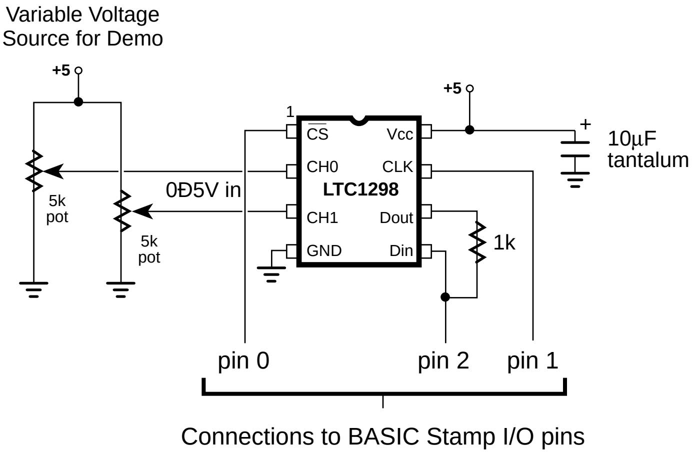
Introduction. This application note shows how to interface the LTC1298 analog-to-digital converter (ADC) to the BASIC Stamp.

Background. Many popular applications for the Stamp include analog measurement, either using the Pot command or an external ADC. These measurements are limited to eight-bit resolution, meaning that a 5-volt full-scale measurement would be broken into units of 5/256 = 19.5 millivolts (mV).

That sounds pretty good until you apply it to a real-world sensor. Take the LM34 and LM35 temperature sensors as an example. They output a voltage proportional to the ambient temperature in degrees Fahrenheit (LM34) or Centigrade (LM35). A 1-degree change in temperature causes a 10-mV change in the sensor's output voltage. So an eight-bit conversion gives lousy 2-degree resolution. By reducing the ADC's range, or amplifying the sensor signal, you can improve resolution, but at the expense of additional components and a less-general design.

The easy way out is to switch to an ADC with 10- or 12-bit resolution. Until recently, that hasn't been a decision to make lightly, since more bits = more bucks. However, the new LTC1298 12-bit ADC is reasonably priced at less than $10, and gives your Stamp projects two channels



Schematic to accompany LTC1298.BAS

of 1.22-mV resolution data. It's packaged in a Stamp-friendly 8-pin DIP, and draws about 250 microamps (µA) of current.

How it works. The figure shows how to connect the LTC1298 to the Stamp, and the listing supplies the necessary driver code. If you have used other synchronous serial devices with the Stamp, such as EEPROMs or other ADCs described in previous application notes, there are no surprises here. We have tied the LTC1298's data input and output together to take advantage of the Stamp's ability to switch data directions on the fly. The resistor limits the current flowing between the Stamp I/O pin and the 1298's data output in case a programming error or other fault causes a "bus conflict." This happens when both pins are in output mode and in opposite states (1 vs. 0). Without the resistor, such a conflict would cause large currents to flow between pins, possibly damaging the Stamp and/or ADC.

If you have used other ADCs, you may have noticed that the LTC1298 has no voltage-reference (Vref) pin. The voltage reference is what an ADC compares its analog input voltage to. When the analog voltage is equal to the reference voltage, the ADC outputs its maximum measurement value; 4095 in this case. Smaller input voltages result in proportionally smaller output values. For example, an input of 1/10th the reference voltage would produce an output value of 409.

The LTC1298's voltage reference is internally connected to the power supply, Vcc, at pin 8. This means that a full-scale reading of 4095 will occur when the input voltage is equal to the power-supply voltage, nominally 5 volts. Notice the weasel word "nominally," meaning "in name only." The actual voltage at the +5-volt rail of the full-size (pre-BS1-IC) Stamp with the LM2936 regulator can be 4.9 to 5.1 volts initially, and can vary by 30 mV.

In some applications you'll need a calibration step to compensate for the supply voltage. Suppose the LTC1298 is looking at 2.00 volts. If the supply is 4.90 volts, the LTC1298 will measure (2.00/4.90) * 4095 = 1671. If the supply is at the other extreme, 5.10 volts, the LTC1298 will measure (2.00/5.10) * 4095 = 1606.

How about that 30-mV deviation in regulator performance, which cannot be calibrated away? If calibration makes it seem as though the LTC1298 is getting a 5.000-volt reference, a 30-mV variation means that the reference would vary 15 mV high or low. Using the 2.00-volt example, the LTC1298 measurements can range from (2.00/4.985) * 4095 = 1643 to (2.00/5.015) * 4095 = 1633.

The bottom line is that the measurements you make with the LTC1298 will be only as good as the stability of your +5-volt supply.

The reason the manufacturer left off a separate voltage-reference pin was to make room for the chip's second analog input. The LTC1298 can treat its two inputs as either separate ADC channels, or as a single, differential channel. A differential ADC is one that measures the voltage difference between its inputs, rather than the voltage between one input and ground.

A final feature of the LTC1298 is its sample-and-hold capability. At the instant your program requests data, the ADC grabs and stores the input voltage level in an internal capacitor. It measures this stored voltage, not the actual input voltage.

By measuring a snapshot of the input voltage, the LTC1298 avoids the errors that can occur when an ADC tries to measure a changing voltage. Without going into the gory details, most common ADCs are *successive approximation* types. That means that they zero in on a voltage measurement by comparing a guess to the actual voltage, then determining whether the actual is higher or lower. They formulate a new guess and try again. This becomes very difficult if the voltage is constantly changing! ADCs that aren't equipped with sample-and-hold circuitry should not be used to measure noisy or fast-changing voltages. The LTC1298 has no such restriction.

Parts source. The LTC1298 is available from Digi-Key (800-344-4539) for $8.89 in single quantities (LTC1298CN8-ND). Be sure to request a data sheet or the data book (9210B-ND, $9.95) when you order.

Program listing. This program may be downloaded from our Internet ftp site at ftp.parallaxinc.com. The ftp site may be reached directly or through our web site at http://www.parallaxinc.com.

```
' Program: LTC1298.BAS (LTC1298 analog-to-digital converter)
' The LTC1298 is a 12-bit, two-channel ADC. Its high resolution, low
' supply current, low cost, and built-in sample/hold feature make it a
' great companion for the Stamp in sensor and data-logging applications.
' With its 12-bit resolution, the LTC1298 can measure tiny changes in
' input voltage; 1.22 millivolts (5-volt reference/4096).

' =========================================================
'                  ADC Interface Pins
' =========================================================

' The 1298 uses a four-pin interface, consisting of chip-select, clock,
' data input, and data output. In this application, we tie the data lines
' together with a 1k resistor and connect the Stamp pin designated DIO
' to the data-in side of the resistor. The resistor limits the current
' flowing between DIO and the 1298's data out in case a programming error
' or other fault causes a "bus conflict." This happens when both pins are
' in output mode and in opposite states (1 vs 0). Without the resistor,
' such a conflict would cause large currents to flow between pins,
' possibly damaging the Stamp and/or ADC.

SYMBOL    CS = 0              ' Chip select; 0 = active.
SYMBOL    CLK = 1             ' Clock to ADC; out on rising, in on falling edge.
SYMBOL    DIO_n = 2           ' Pin _number_ of data input/output.
SYMBOL    DIO_p = pin2        ' Variable_name_ of data input/output.
SYMBOL    ADbits = b1         ' Counter variable for serial bit reception.
SYMBOL    AD = w1             ' 12-bit ADC conversion result.

' =========================================================
'                  ADC Setup Bits
' =========================================================

' The 1298 has two modes. As a single-ended ADC, it measures the
' voltage at one of its inputs with respect to ground. As a differential
' ADC, it measures the difference in voltage between the two inputs.
' The sglDif bit determines the mode; 1 = single-ended, 0 = differential.
' When the 1298 is single-ended, the oddSign bit selects the active input
' channel; 0 = channel 0 (pin 2), 1 = channel 1 (pin 3).
' When the 1298 is differential, the oddSign bit selects the polarity
' between the two inputs; 0 = channel 0 is +, 1 = channel 1 is +.
' The msbf bit determines whether clock cycles _after_ the 12 data bits
' have been sent will send 0s (msbf = 1) or a least-significant-bit-first
' copy of the data (msbf = 0). This program doesn't continue clocking after
' the data has been obtained, so this bit doesn't matter.

' You probably won't need to change the basic mode (single/differential)
' or the format of the post-data bits while the program is running, so
' these are assigned as constants. You probably will want to be able to
' change channels, so oddSign (the channel selector) is a bit variable.
```

```
SYMBOL   sglDif = 1        ' Single-ended, two-channel mode.
SYMBOL   msbf = 1          ' Output 0s after data transfer is complete.
SYMBOL   oddSign = bit0    ' Program writes channel # to this bit.


' ==========================================================
'              Demo Program
' ==========================================================

' This program demonstrates the LTC1298 by alternately sampling the two
' input channels and presenting the results on the PC screen using Debug.

high CS                      ' Deactivate the ADC to begin.
Again:                       ' Main loop.
 For oddSign = 0 to 1        ' Toggle between input channels.
  gosub Convert              ' Get data from ADC.
  debug "ch ",#oddSign,":",#AD,cr  ' Show the data on PC screen.
  pause 500                  ' Wait a half second.
 next                        ' Change input channels.
goto Again                   ' Endless loop.

' ==========================================================
'              ADC Subroutine
' ==========================================================

' Here’s where the conversion occurs. The Stamp first sends the setup
' bits to the 1298, then clocks in one null bit (a dummy bit that always
' reads 0) followed by the conversion data.

Convert:
 low CLK                     ' Low clock—output on rising edge.
 high DIO_n                  ' Switch DIO to output high (start bit).
 low CS                      ' Activate the 1298.
 pulsout CLK,5               ' Send start bit.
 let DIO_p = sglDif          ' First setup bit.
 pulsout CLK,5               ' Send bit.
 let DIO_p = oddSign         ' Second setup bit.
 pulsout CLK,5               ' Send bit.
 let DIO_p = msbf            ' Final setup bit.
 pulsout CLK,5               ' Send bit.
 input DIO_n                 ' Get ready for input from DIO.
 let AD = 0                  ' Clear old ADC result.
 for ADbits = 1 to 13        ' Get null bit + 12 data bits.
  let AD = AD*2+DIO_p        ' Shift AD left, add new data bit.
  pulsout CLK,5              ' Clock next data bit in.
 next                        ' Get next data bit.
 high CS                     ' Turn off the ADC
return                       ' Return to program.
```