Column #118 February 2005 by Jon Williams:

The Sheer Joy of Experimenting

For as long as I can remember I've been interested in electricity and electronics. My first "way cool" experiment happened when I was in the second grade: I (correctly) deduced that a flashlight bulb would burn brighter if I connected wires to it and plugged them into a 120 VAC outlet. Yes indeed, I was correct – for about a millisecond before the bulb exploded. I'm not sure if it was the explosion of the bulb, or the explosion of my mom (she's got 'pipes' when she needs them) that hooked me, but I am forever hooked..

Thankfully, most of my experiments since that fateful day have had better results, and my dear mom is more smiles than fearful for her first-born's life. You're probably wondering by now where in the world I'm going with this; well, let me tell you.

As I've stated many times, I'm a very lucky guy: great job(s), a wonderful life, a wonderful family ... I really have no reason to complain whatsoever. Part of my great job with Parallax is the myriad BASIC Stamp users I have contact with, and on many occasions I get to assist. And that is something I truly enjoy doing. But here's the thing ... in our instant-gratification society, it seems like many BASIC Stamp users, especially those that are young and facing the deadline of a school project, are too easily willing to skip the experimentation part of the learning process. My friend and colleague, John Barrowman, calls this "the big bang technique." They want it all at once; no reading of spec sheets, manuals, or experimenting with parts desired – just results, please. Right now!

To be candid, this is a little disappointing. Why? Well, those that "shotgun" their projects miss out on the sheer joy of experimenting, and all the learning that comes with those experiments. It took Thomas Edison nearly 1000 attempts to get a practical working light bulb – and he was a heck of a lot smarter than most of us. Do you think the Wright brother's looked up at a bird then headed for Kitty Hawk with a working version of the airplane? Of course not; there was a lot of incremental experimenting before the made a successful powered flight.

With my exposure I get bombarded with questions, some nearly as silly as: "Jon, what would happen if I stuck my finger into a light socket?" My standard answer is, "I don't know – give it a try." Okay, okay, I'm being a bit melodramatic (I am an actor in my other job) to illustrate a point, and of course I would never suggest that someone attempt something that would harm any person or BASIC Stamp module. I guess my point is that for many of those questions, the answer would have been derived more quickly, and with much deeper understanding through an experiment versus waiting for an e-mail from me.

I will quite preaching now ... but please let me beseech you – especially those that are young and just getting started – to experiment. It's fun; it's a lot of fun. And I promise that what you learn from your experiments you won't soon forget – especially those experiments with unexpected results.

Analog Fun

Okay, let's experiment.   For reasons I can't explain, a bunch of customers seem to have come up with the ADC0832 (2-channel ADC) and are wondering why the application code for the ADC0831 floating around doesn't work with it (That sound you just heard was the old-timers smacking themselves on the forehead and exclaiming, "Duh!"). The reason, of course, is that the ADC0832 is NOT the same as the ADC0831. It's in the same family, yes, but it is not the same part so it will require different connections and code.

For our experiments this month, we will need just a few parts: the ADC0831 and ADC0832 (or ADC08832) that we just mentioned, a couple 10K trim pots, and a project board to connect things to the BASIC Stamp. You can use anything handy: the BOE, the NX-1000, or – my new favorite – the Parallax Professional Development Board.

Let's start with the ADC0831. I will admit that I've kind of glossed over the details of this part in the few projects where I used it, because I (incorrectly) assumed that it – and code for it – had been around for such a long time that everybody who used the BASIC Stamp had an understanding of how the ADC0831 works and how to apply it. As is occasionally the case ... I was wrong.

Grab your parts and connect the circuit shown in Figure 118.1. For the time being, make sure that you have the pot connected to Vref moved to the +5v position (confirm with a multimeter). Before we get on to the code, let's have a look at a couple technical details that the manufacture provides: a timing diagram. The timing diagram shows the signals in and out of a device and the relationships vis-à-vis time. Figure 118.2 shows the essential signals timing for the ADC0831.

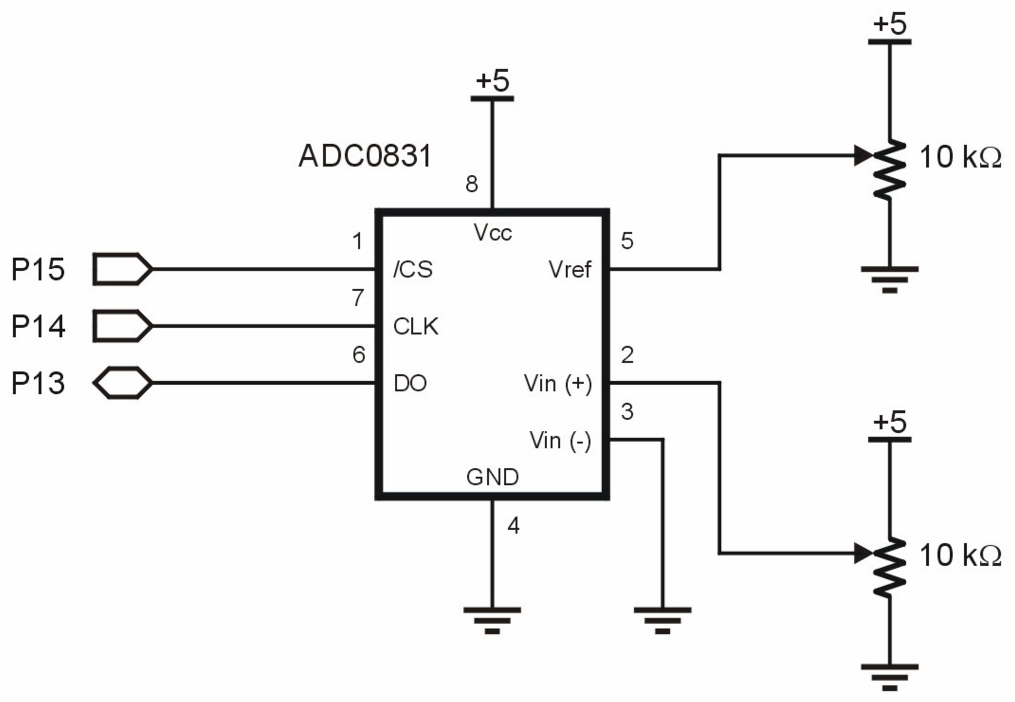


Figure 118.1: ADC0831 Connections to a BASIC Stamp

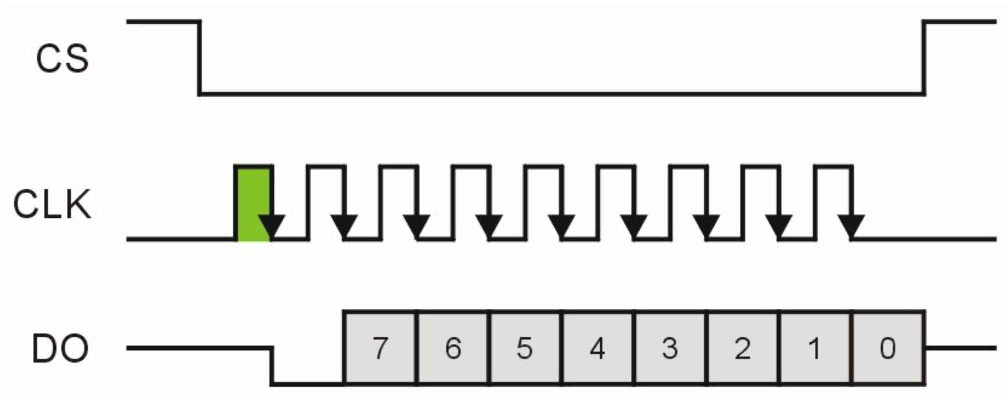


Figure 118.2: ADC0831 Signal Diagram

What this diagram shows us is that the must take the CS (chip select line) low to initiate a read of the ADC0831. Why? Well, maybe we can't find an ADC0832 and we need two distinct channels. By using separate CS lines to control each device, two ADC0831s can share the Clock (CLK) and Data Out (DO) lines. After the CS line is brought low, the CLK line needs to be pulsed activate the device. After that, eight additional clock pulses cause the data bits to be shifted out of the ADC0831.

At first we may be inclined to write a subroutine that looks like this:

```
Read_0831:
  LOW CS
  PULSOUT Clk, 50
  SHIFTIN Dio, Clk, MSBPOST, [adc\8]
  HIGH CS
  RETURN
```

It all makes sense, right? We bring the CS line low, blip the clock line with a pulse to wake the ADC0831, shift eight bits in, and then finish up by returning the CS line high. Anything wrong with that? Nope, not a thing. Can the routine be improved? You bet.

Through understanding and experimenting (there's that dreaded "e"-word....) we find that we can remove the PULSOUT instruction, and shorten the code to look like this:

```
Read_0831:
  LOW CS
  SHIFTIN Dio, Clk, MSBPOST, [adc\9]
  HIGH CS
  RETURN
```

The first question that probably comes to mind is, "How can you use nine clock pulses with an eight-bit value?" We can do that by understanding what happens to a value when we use SHIFTIN. Let's get gory with details, shall we?

When using MSB mode, the target variable is shifted left (MSB goes into la-la-land, and a 0 is placed in Bit0), then the data line is sampled (in this case after the clock – POST mode, because the ADC0831 makes data bits available after the clock pulse falls) and that bit is placed into the Bit0 of our target variable. When we get to that ninth clock, the first bit sampled gets shifted from the MSB (Bit7 in this case) into the great bit-bucket and is discarded. Here's how we could simulate the SHIFTIN using a loop:

```
Shift_In_MSBPOST:
  FOR idx = 1 TO 8
    adc = adc << 1
    PULSOUT CLK, 50
    target.Bit0 = Dio
  NEXT
```

For obvious reasons using SHIFTIN is easier, but understanding the mechanics is helpful when we want to optimize code, or – if needed – port to a lower-featured controller like the BS1.

Now that we can read the ADC0831, it's a fairly a fairly simple matter to display the data:

```
Main:
  DO
    GOSUB Read_0831
    mVolts = adc */ Raw2mV
    DEBUG CRSRXY, 10, 2,
          DEC3 adc,
          CRSRXY, 10, 3,
          DEC mVolts DIG 3, ".", DEC3 mVolts
  LOOP
  END
```

This code uses a constant called Raw2mV that is used to convert the input voltage to millivolts – just keep in mind that the value in the program is for +5 on Vref. The value of Raw2mV is $139B which is the equivalent of 19.6 when using the */ (star-slash) operator.

Remember that star-slash multiplies by units of 1/256, so it's a convenient way to multiply fractional values. How'd we get $139B? First we take the Vref voltage of 5.00 and divide it by 255 which is the maximum output value from the ADC0831. This gives us 0.019607. If we multiplied directly by 0.019607 we would end up with very small output values, so what we'll do is multiply by 1000 which will cause the result of our multiplication to be millivolts (1/1000 of a volt). Now we take 19.607 and multiply that by 256 for the star-slash operator. We get 5019. My habit is to convert to hex, $139B, as this puts the whole portion of the value in the high-byte, the fractional portion of the value in the low-byte.

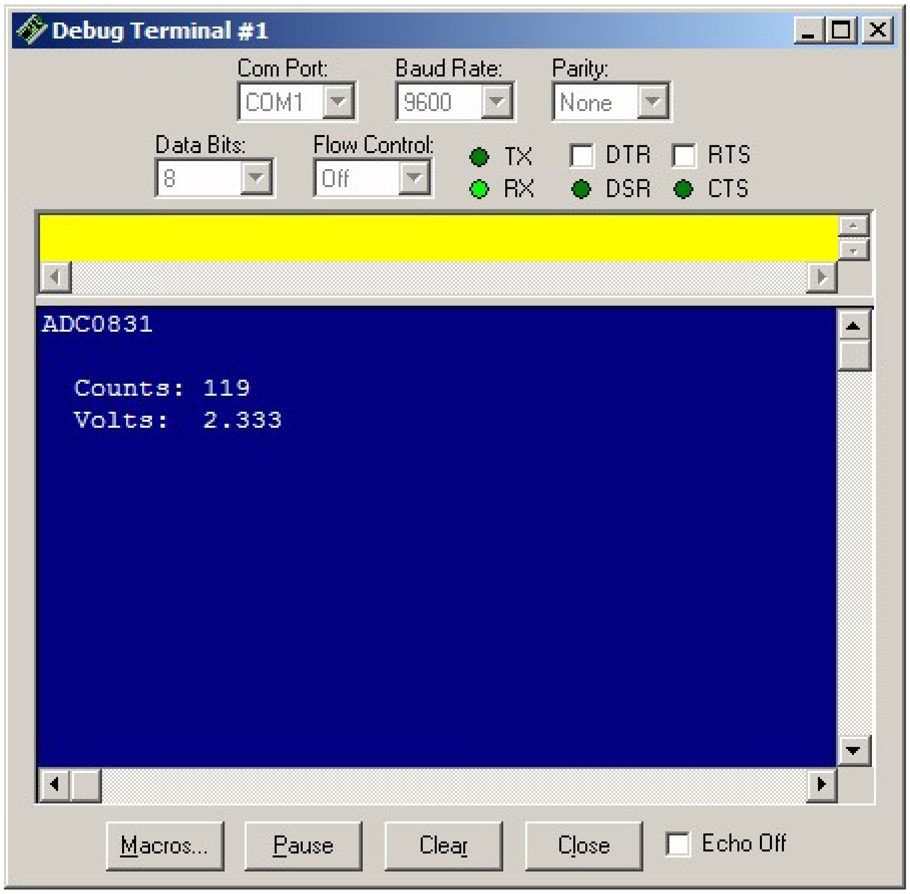
After calculating the millivolts value, the display is handled with DEBUG, and a couple neat tricks using the DIG operator and the DEC modifier (see Figure 118.3). If you don't fully understand what's happening with the output, here's your big chance: open the help file, read up on DEBUG, DIG (in the operators section), and using formatters (like DEC). Then ... you got it ... experiment. A few minutes experimenting will save you hours of frustration later.

Once you've got the display figured out, it's time to play with the Vref voltage and examine its affect on the output data. Using your multimeter you should be able to prove the following behavior of the ADC0831:

```
counts = (Vin / Vref) x 255
```

What you'll also see is that when Vin exceeds Vref the output hits a ceiling of 255 – we need to keep this in mind with designs when Vin could exceed Vref.

One last thing before moving on, and again, you're on your own after I give you a little push. The ADC0831 is typically used as a single-ended device, but can also be used in differential mode. The Vin discussed above is really the difference between Vin(+) and Vin(-). Let's say we had an application where we wanted to know how much greater one voltage is of another. We can do it with the ADC0831 by connecting the greater voltage to Vin(+), and the lesser voltage to Vin(-). The output will be the difference between the two values, relative to Vref. What happens if Vin(+) goes lower than Vin(-)? I know – because I did an experiment to find out. It's your turn....



Com Port: COM1
Baud Rate: 9600
Parity: None
Data Bits: 8
Flow Control: Off

```
ADC0831

  Counts: 119
  Volts:  2.333
```

Figure 118.3: Using DEBUG with DIG and DEC modifiers is helpful

Two for the Price of One

Okay, let's give the ADC0832 (or ADC08832) a try, shall we? I think part of the problem some users have had with this chip is that it comes in the same physical format as the ADC0831 (8-pin DIP). But the connections aren't the same, and neither is the code (it's close, though).

Connect the circuit shown in Figure 118.4. Immediately you'll notice a couple things: the ADC0832 has a DI (Data In) line, and there is no Vref pin (Vref is tied internally to Vcc). One of the nice things about using the BASIC Stamp is that it can change an IO pin's state on the fly, so we don't need separate lines for DI and DO – we can tie them together. But, we don't want to connect them directly to the BASIC Stamp. Why? Well, as you'll see in just a moment the DI line is three expecting control bits after the CS line falls, then it activates the DO line and starts pumping out data. If we made a programming error that caused SHIFTOUT to send more bits than required, we could end up with a data collision. Worse, one side could be high while the other is low, causing a direct short and perhaps doing damage to the ADC0832, the BASIC Stamp, or both. A five-cent resistor is cheap insurance; it will protect us from a problem and has no ill affect on communication between the BASIC Stamp and the ADC0832.

Figure 118.5 shows the ADC0832 interface timing. As with the ADC0831, we start by taking the CS line low. This time, though, the start bit is output (by the Stamp) on the Dio line, and must be a 1. The next two bits configure the ADC0832. The first of those two bits sets single-ended or differential mode. The final bit serves as the channel indicator when in single-ended mode, or which input is positive when using differential mode.

Let's code it up:

```
Read_0832:
  LOW CS
  SHIFTOUT Dio, Clk, MSBFIRST, [%1\1, sglDif\1, oddSign\1]
  SHIFTIN  Dio, Clk, MSBPOST, [adc(oddSign)\8]
  HIGH CS
  RETURN
```

Once again, the code matches the timing diagram without a lot of mystery. After taking the CS line low, we have to shift out a "1" to get things started. Since we only want to send a single bit, the \1 parameter is used with the value. Next comes the mode bit: 1 indicates single-ended, 0 indicates differential. Finally, we shift out a bit that indicates the channel for single-ended mode, or the positive (+) channel for differential mode. When the data is shifted in we place it into the selected element of a two-byte array.

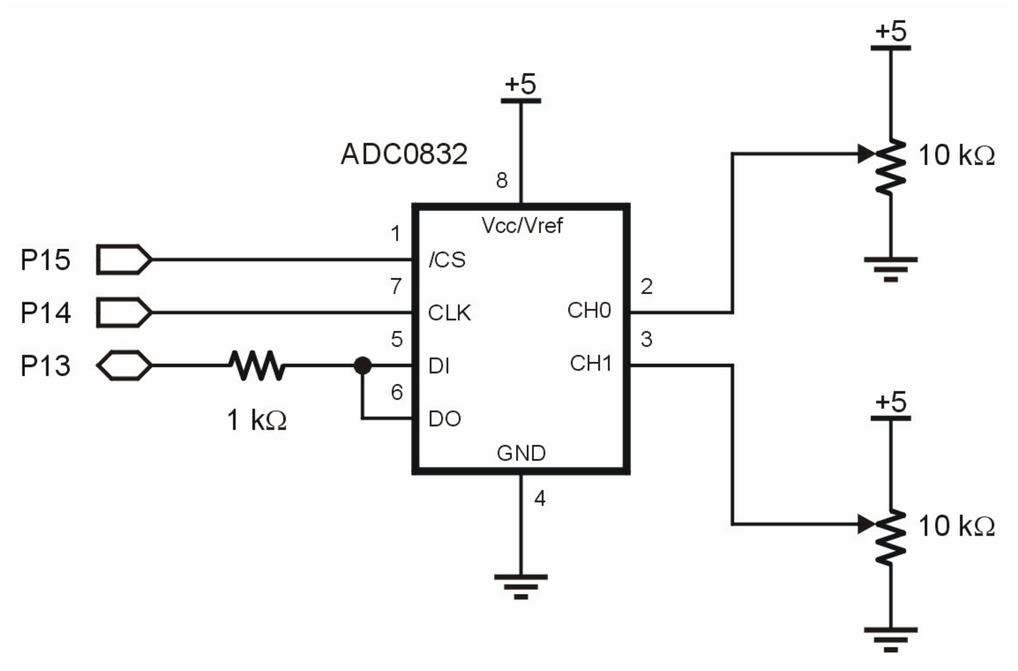


Figure 118.4: ADC0832 Connections to a BASIC Stamp

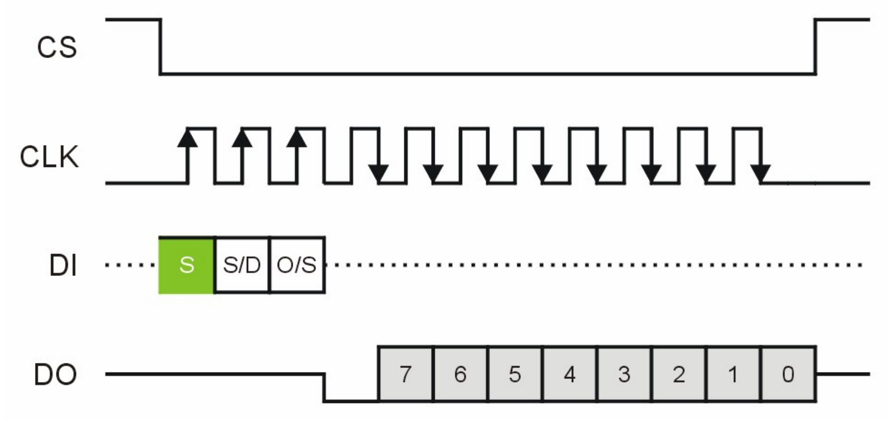


Figure 118.5: ADC0831 Timing Diagram

The neat thing about the ADC0832 and the circuit we put together is we can try its various modes without changing any wires. While the demo for this device is a little more involved than with the ADC0831, it's really no more complicated:

```
Main:
  DO
    sglDif = Sgl
    FOR oddSign = 0 TO 1
      GOSUB Read_0832
      mVolts(oddSign) = adc(oddSign) */ Raw2mV
      DEBUG CRSRXY, 7, (4 + oddSign),
            DEC3 adc(oddSign), TAB,
            DEC mVolts(oddSign) DIG 3, ".",
            DEC3 mVolts(oddSign)
    NEXT

    sglDif = Dif
    FOR oddSign = 0 TO 1
      GOSUB Read_0832
      mVolts(oddSign) = adc(oddSign) */ Raw2mV
      DEBUG CRSRXY, 7, (9 + oddSign),
            DEC3 adc(oddSign), TAB,
            DEC mVolts(oddSign) DIG 3, ".",
            DEC3 mVolts(oddSign)
    NEXT
  LOOP
```

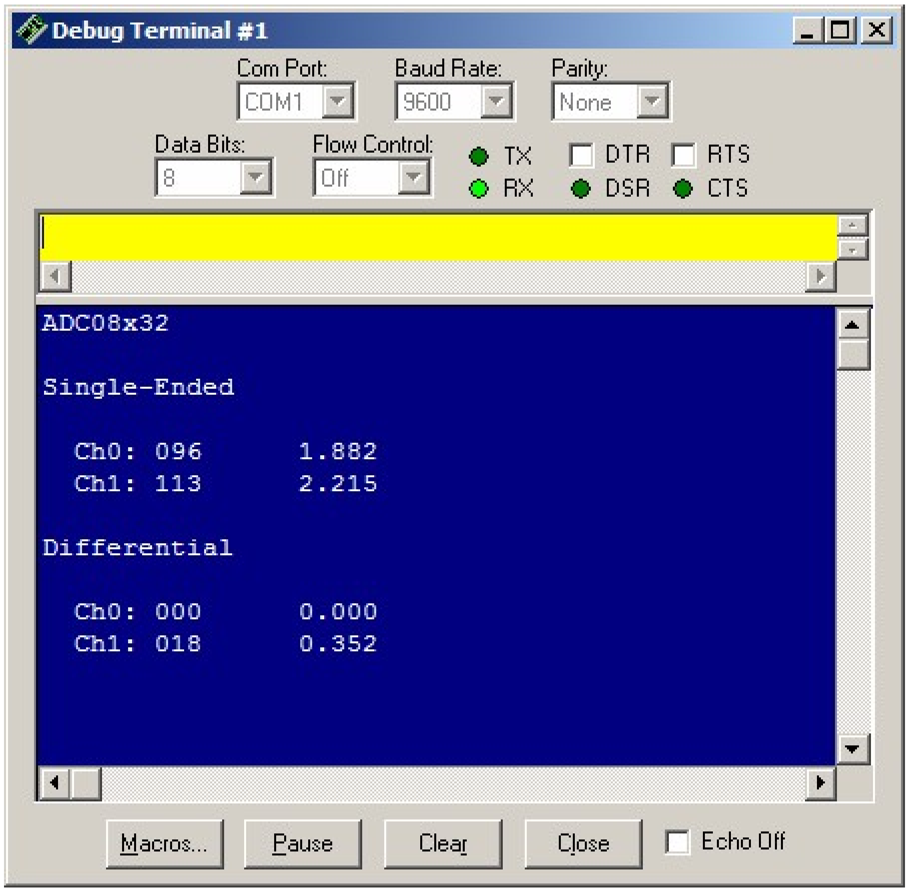
The program simply loops though both channels in each of the two operating modes, displaying the raw counts and calculated voltage output as before. Figure 118.6 shows the results. If you look very closely at that display you'll notice that the differential mode differs from the single-ended mode by about two counts – this had to do with a noisy test setup, and I could never really get a single-ended reading to hard zero.

Finally, Figure 118.7 shows you the result of an experiment of mine with the ADC0832 setup. I recently purchased a digital scope/logic analyzer and I hadn't had a chance to use the logic analyzer portion of it. So I connected three probes (one each for CS, Clk, and Dio) and captured a part of the transmission that corresponds to the values in Figure 118.6.

What's interesting to note is the spacing between the configuration bits versus the data bits. Why do you suppose this is? If you go back and look at the subroutine you will see this line:

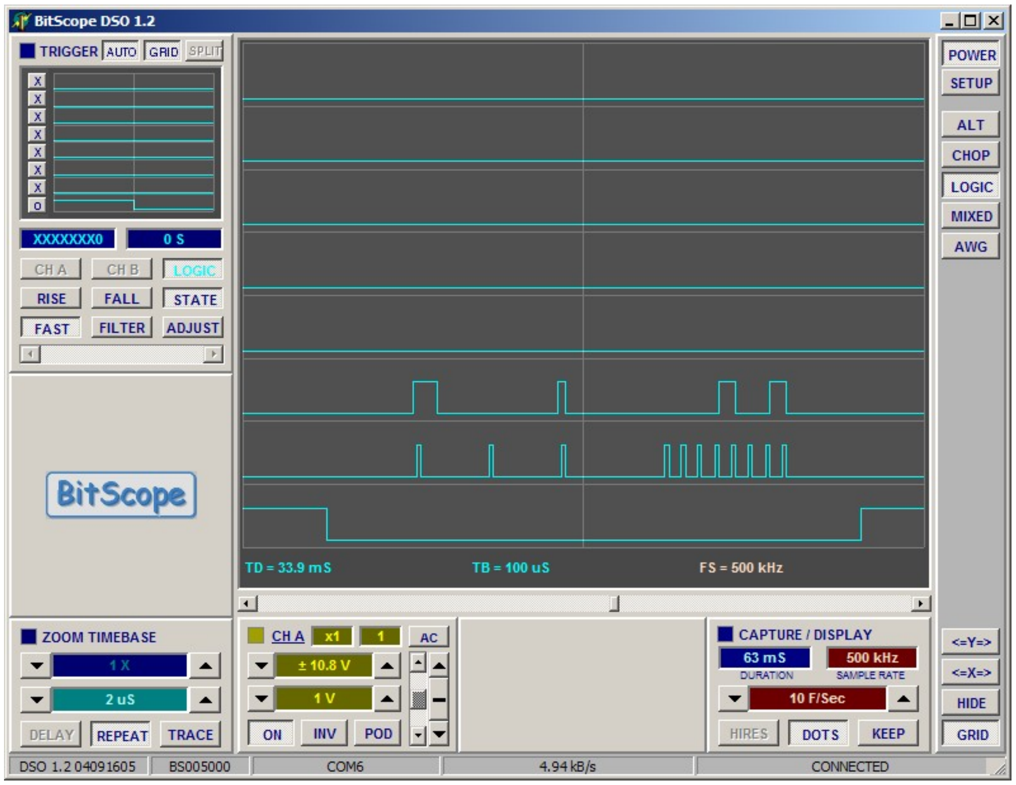
```
  SHIFTOUT Dio, Clk, MSBFIRST, [%1\1, sglDif\1, oddSign\1]
```

This is actually quite complicated. We're asking the BASIC Stamp to send one bit of a constant value, then go retrieve a variable and send one bit from it, then go get another bit and send a single bit from it. This explains the wider timing between the clock pulses going out versus the pulses for the data coming in (which only has to deal with a single variable).



```
ADC08x32

Single-Ended

  Ch0: 096      1.882
  Ch1: 113      2.215

Differential

  Ch0: 000      0.000
  Ch1: 018      0.352
```

Figure 118.6: ADC0832 Output



XXXXXXX0 0 S

TD = 33.9 mS TB = 100 uS FS = 500 kHz

1 X 2 uS

CH A x1 1 ± 10.8 V 1 V

63 mS DURATION 500 kHz SAMPLE RATE 10 F/Sec

DSO 1.2 04091605 BS005000 COM6 4.94 kB/s CONNECTED

Figure 118.7: ADC0832 Timing Diagram

I'm leaving it up to you know – go forth and EXPERIMENT! Remember that Rome wasn't built in a day, and neither will all of your projects. What you'll find is that after you've experimented for a while, you internal knowledge base will grow to the point where projects will come faster. That said, don't be afraid to try little things, as lots of little things add up to greatness.

Allow me to wish you and your sweetie a Happy Valentine's Day (Guys, robots do NOT make cool gifts – unless your girlfriend is an engineer, or reads Nuts & Volts and Servo. Remember, chocolate still works), and until next time, Happy Stamping.

```
' =========================================================================
'
'   File...... ADC0831.BS2
'   Purpose... Experiments with the ADC0831 ADC
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 06 DEC 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =========================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demostates the ADC0831 ADC.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

CS              PIN     15                      ' ADC0831.1
Clk             PIN     14                      ' ADC0831.7
Dio             PIN     13                      ' ADC0831.6



' -----[ Constants ]-------------------------------------------------------

Raw2mV          CON     $139B                   ' 19.6 mV per count


' -----[ Variables ]-------------------------------------------------------

adc             VAR     Byte                    ' raw adc value
mVolts          VAR     Word                    ' millivolts


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------

Reset:
  HIGH CS                                       ' deselect ADC
  DEBUG CLS,                                    ' output screen
        "ADC0831", CR, CR,
        "  Counts:", CR,
        "  Volts: "


' -----[ Program Code ]----------------------------------------------------

Main:
  DO
    GOSUB Read_0831
    mVolts = adc */ Raw2mV
    DEBUG CRSRXY, 10, 2,
          DEC3 adc,
          CRSRXY, 10, 3,
          DEC mVolts DIG 3, ".", DEC3 mVolts
  LOOP
  END


' -----[ Subroutines ]-----------------------------------------------------

' Reads ADC0831

Read_0831:
  LOW CS
  SHIFTIN  Dio, Clk, MSBPOST, [adc\9]
  HIGH CS
  RETURN
```

```
' =========================================================================
'
'   File...... ADC08x32.BS2
'   Purpose... Experiments with the ADC08x32 ADC
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 06 DEC 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =========================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demostates the various modes of the ADC08x82 ADC.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

CS              PIN     15                      ' ADC0832.1
Clk             PIN     14                      ' ADC0832.7
Dio             PIN     13                      ' ADC0832.5 / ADC0832.6



' -----[ Constants ]-------------------------------------------------------

Sgl             CON     %1                      ' single ended
Dif             CON     %0                      ' differential

Raw2mV          CON     $139B                   ' 19.6 mV per count


' -----[ Variables ]-------------------------------------------------------

sglDif          VAR     Bit                     ' adc mode (1 = SE)
oddSign         VAR     Bit                     ' chan (Sgl), sign (Dif)
adc             VAR     Byte(2)                 ' channel values
mVolts          VAR     Word(2)                 ' millivolts


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------
```

```
Reset:
  HIGH CS                                       ' deselect ADC

  DEBUG CLS,
        "ADC08x32",
        CRSRXY, 0, 2, "Single-Ended", CR, CR,
        "  Ch0:", CR,
        "  Ch1:", CR,
        CRSRXY, 0, 7, "Differential", CR, CR,
        "  Ch0:", CR,
        "  Ch1:"

' -----[ Program Code ]----------------------------------------------------

Main:
  DO
    sglDif = Sgl
    FOR oddSign = 0 TO 1
      GOSUB Read_0832
      mVolts(oddSign) = adc(oddSign) */ Raw2mV
      DEBUG CRSRXY, 7, (4 + oddSign),
            DEC3 adc(oddSign), TAB,
            DEC mVolts(oddSign) DIG 3, ".", DEC3 mVolts(oddSign)
    NEXT

    sglDif = Dif
    FOR oddSign = 0 TO 1
      GOSUB Read_0832
      mVolts(oddSign) = adc(oddSign) */ Raw2mV
      DEBUG CRSRXY, 7, (9 + oddSign),
            DEC3 adc(oddSign), TAB,
            DEC mVolts(oddSign) DIG 3, ".", DEC3 mVolts(oddSign)
    NEXT
  LOOP

  END


' -----[ Subroutines ]-----------------------------------------------------

' Reads ADC08x32

Read_0832:
  LOW CS
  ' send start, mode, channel
  SHIFTOUT Dio, Clk, MSBFIRST, [%1\1, sglDif\1, oddSign\1]
  ' read raw counts from ADC
  SHIFTIN  Dio, Clk, MSBPOST, [adc(oddSign)\8]
  HIGH CS
  RETURN
```