Stamp Applications electronic version, vol. 1

This document is a collection of the first seven installments of the *Stamp Applications* column published in *Nuts & Volts* magazine starting in March 1995. The column provides hints, tips, and techniques for users of the popular Parallax BASIC Stamp® single-board computer and its kit equivalent, the Counterfeit. The Counterfeit uses a genuine Parallax BASIC (PBASIC) interpreter chip and software licensed from Parallax. It's 100-percent compatible with the original BASIC Stamp.

I prepared this electronic version of *Stamp Applications* in response to many requests from Stamp users on Compuserve and the Internet. I plan to post additional volumes every six issues/months. Don't bother petitioning me to post more frequently; that would defeat one of the purposes of this document, which is to encourage Stamp users to subscribe to *Nuts & Volts*.

Contacting Us

Scott Edwards Electronics
964 Cactus Wren Lane
Sierra Vista, AZ 85635 USA
ph: 520-459-4802; fax 520-459-0623
e-mail: 72037.2612@compuserve.com

Contacting Nuts & Volts magazine

T&L Publications Inc.
430 Princeland Court
Corona, CA 91719
ph: 909-371-8497; fax: 909-371-3052
subscription order line: 800-783-4624

Contacting Parallax

Parallax, Inc.
3805 Atherton Road, #102
Rocklin, CA 95765 USA
ph: 916-624-8333; fax 916-624-8003; BBS: 916-624-7101
e-mail: info@parallaxinc.com; file transfer via Internet: ftp.parallaxinc.com

Topics in Volume 1

Stamp Applications no. 1 (March '95):

New Column Puts the Spotlight on BASIC Stamp Projects, Hints, and Tips

First Look at the New BS1-IC Stamp-on-a-SIP, by Scott Edwards

WELCOME to the first installment of Stamp Applications, a forum for users of BASIC Stamp single-board computers. Every month we'll introduce new ideas in hardware and software for the Stamp, answer questions, and keep up with news about the Stamp and related products.

To kick things off, let's look at the newest member of the Stamp product line, the BS1-IC, also known as "Stamp on a SIP." SIP stands for single-inline package, and it describes the diminuitive new Stamp's printed-circuit board design. The entire Stamp circuit plugs right into a 14-pin SIP socket. This makes it easier to breadboard with the Stamp, or to incorporate it into your own circuit designs.

Figure 1 shows the layout and important dimensions of the BS1-IC. I elected to sketch the unit rather than photograph it because its small size makes it a real challenge to get a decent picture.

The BS1-IC has four advantages over the regular, full-sized Stamp. The first is size. The BS1-IC is small enough to be treated as a component that can be integrated into ultra-small designs.

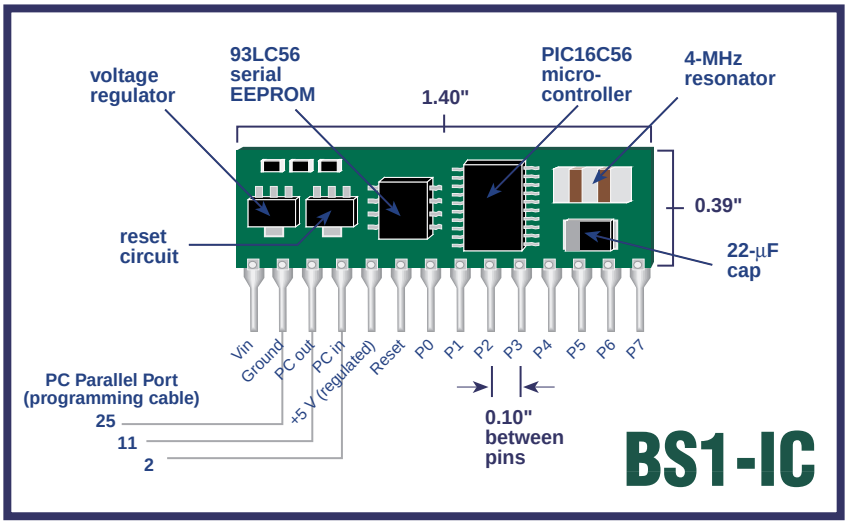


Figure 1. The new BS1-IC.

The second advantage is price. At $29, the BS1-IC has everything the original $39 Stamp had, except for the battery clips and prototyping area. If your power supply is something other than a 9V battery, you won't miss the clips. And if you're not a fan of wire-wrapping (I detest it! Crocheting for engineers!), you'd probably never use the grid-of-holes proto area anyway. The BS1-IC plugs right into socket-type proto boards, like the ones sold by Radio Shack and Jameco, allowing rapid circuit testing. If you want to add the clips and proto grid later, you can buy a "carrier board" for $10 to turn the BS1-IC in a normal-form-factor Stamp.

The third, least tangible benefit of the component-style package is professional appearance. You can integrate a factory-assembled and tested Stamp into a product without having a messy jumble of printed circuit boards under the hood. Put the BS1-IC right on the circuit board with your application-specific components.

The fourth advantage over the original Stamp design is not very glamorous, but nonetheless important: The BS1-IC has a reset input that's accessible to your circuitry. You can add a button between reset and ground. This allows you to reset a runaway program to the beginning. You can also treat the reset connection as an output, using it to reset other logic in your circuit.

In my first application for the BS1-IC, I wasn't too worried about looking good. I just wanted to reduce the jumble of jumper wires that was accumulating as I interfaced Stamps to different projects and test jigs around the shop. My Stamp Stretchers and LCD Serial Backpacks both borrow power from the Stamp and use a standard connector layout: +5V, ground, serial data. I decided to make a bunch of three-wire jumpers with female header sockets at each end. I then designed my own carrier board with eight, three-conductor male headers--one for each input/output pin.

Figure 2 shows the resulting board layout. I present it not as a finished product, but as an example of how simply you can turn Parallax' newest Stamp into Scott's Stamp or Mary's Stamp or Acme Industries' Stamp.

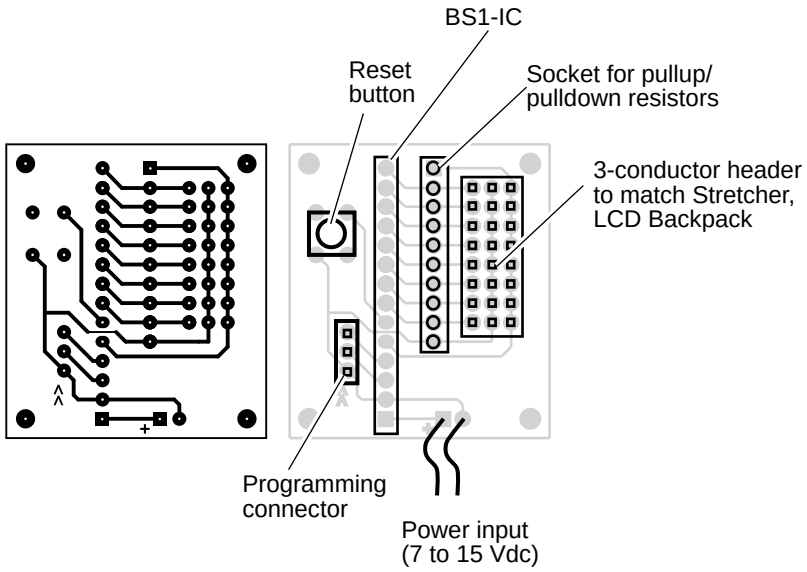


Figure 2. Example of a simple, homemade carrier board for the BS1-IC.

That wraps up this month's installment of Stamp Applications. Future columns will dig into Stamp hardware and software issues, generally presenting at least one schematic and one program listing each month. Initially, I'll take the topics from Parallax's technical support folks and the online venues where the Stamp is discussed. But I'd be grateful for your suggestions and questions. Contact me at:

Scott Edwards Electronics
964 Cactus Wren Lane
Sierra Vista, AZ 85635
phone: 602-459-4802; fax 602-459-0623
Internet: 72037.2612@compuserve.com
Compuserve: 72037,2612

Stamp Applications no. 2 (April '95):

Thermometer-on-a-Chip Simplifies Temperature Measurement

Using the DS1620, by Scott Edwards

DESIGNING a temperature-measurement application for the BASIC Stamp is a lot like voting; you end up selecting the lesser of three evils, shown in figure 1. Cost, calibration, or too many components make these solutions less than ideal.

Now, there's a fourth candidate. It's the Dallas Semiconductor DS1620 digital thermometer/thermostat chip, shown in figure 2. The DS1620 measures temperature in units of 0.5 degrees Centigrade from -55° to +125° C. The DS1620 is calibrated at the factory for exceptional accuracy: ±0.5° C from 0 to +70° C, which, is the Stamp's operating temperature range.

(For fans of the familiar Fahrenheit scale, those °C temperatures convert to: range, -67° to +257° F; resolution, 0.9° F; accuracy, ±0.9° F from 32° to 158° F.)

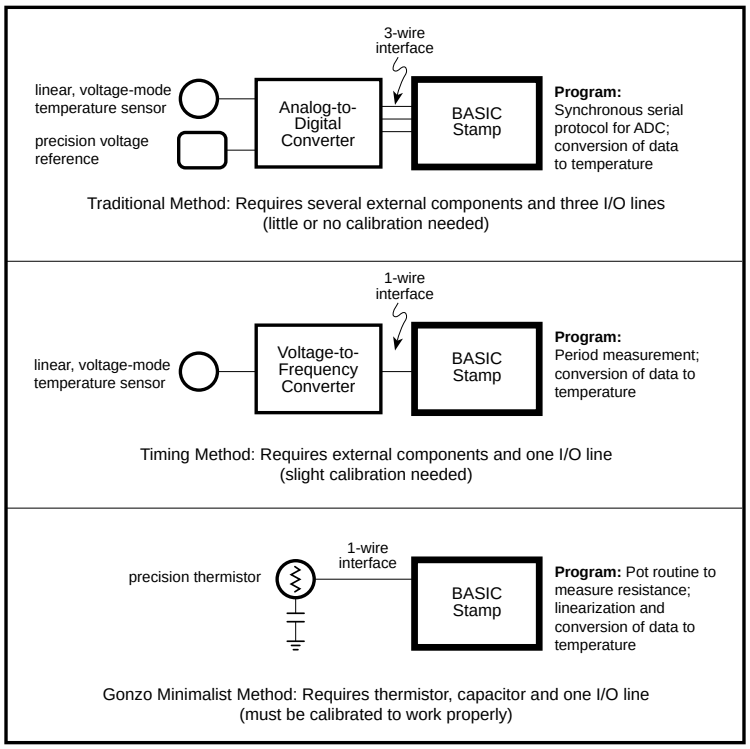


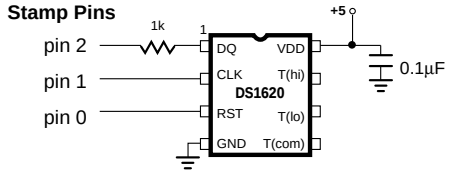
Figure 1. Three methods for measuring temperature with the Stamp.

The chip makes its temperature data available as a 9-bit number conveyed over a three-wire serial interface. The DS1620 can be set to operate continuously, taking one temperature measurement per second, or intermittently, conserving power by measuring only when told to.

The DS1620 can also operate as a standalone thermostat. A temporary connection to a Stamp or other controller establishes the mode of operation and high/low-temperature setpoints. Thereafter, the chip independently controls three outputs: T(high), which goes active at temperatures above the high-temperature setpoint; T(low), active at temperatures below the low setpoint; and T(com), which goes active at temperatures above the high setpoint, and stays active until the temperature drops below the low setpoint.

Let's concentrate on applications using the DS1620 as a Stamp peripheral, as shown in the listing. Later, we'll talk about adapting the code for configuring it as a thermostat.

Using the DS1620 requires sending a command (what Dallas Semi calls a "protocol") to the chip, then listening for a response (if applicable). The code under "DS1620 I/O Subroutines" in the listing shows how this is done. In a typical temperature-measurement application, the program will set the DS1620 to thermometer mode, configure it for continuous conversions, and tell it to start. Thereafter, all the program must do is request a temperature reading, then shift it in, as shown in the listing's *Again* loop.



DQ—Data input/output
CLK—Clock for shifting data in/out (active-low conversion start in thermostat/ 1-shot mode)
RST—Reset; high activates chip, low disables it
GND—Ground connection
VDD—Supply voltage; +4.5 to 5.5 Vdc
T(hi)—In thermostat mode, outputs a 1 when temp is above high setpoint
T(lo)—In thermostat mode, outputs a 1 when temp is below low setpoint
T(com)—In thermostat mode, outputs a 1 when temp exceeds high setpoint and remains high until temp drops below low setpoint

Figure 2. DS1620 pinout and connection.

The DS1620 delivers temperature data in a nine-bit, two's complement format, shown in the table. Each unit represents 0.5° C, so a reading of 50 translates to +25°C. Negative values are expressed as two's complement numbers. In two's complement, values with a 1 in their leftmost bit position are negative. The leftmost bit is often called the *sign* bit, since a 1 means - and a 0 means +.

To convert a negative two's complement value to a positive number, you must invert it and add 1. If you want to display this value, remember to put a minus sign in front of it.

Rather than mess with two's complement negative numbers, the program converts DS1620 data to an absolute scale called DSabs, with a range of 0 to 360 units of 0.5° C each. The Stamp can perform calculations in this all-positive system, then readily convert the results for display in °C or °F, as shown in the listing.

Going Further. Once you have configured the DS1620, you don't have to reconfigure it unless you want to change a setting. The DS1620 stores its configuration in EEPROM (electrically erasable, programmable read-only memory), which retains data even with the power off. In memory-tight Stamp applications, you might want to run the full program once for configuration, then strip out the configuration stuff to make more room for your final application.

If you want to use the DS1620 in its role as a standalone thermostat, the Stamp can help here, too. The listing includes protocols for putting the DS1620 into thermostat (*NoCPU*) mode, and for reading and writing the temperature setpoints. You could write a Stamp program to accept temperature data serially, convert it to nine-bit, two's complement format, then write it to the DS1620 configuration register. An example program that does this is available from the source listed below.

Be aware of the DS1620's drive limitations in thermostat mode; it sources just 1 mA and sinks 4 mA. This isn't nearly enough to drive a relay—it's barely enough to light an LED. You'll want to buffer this output with a Darlington

Nine-Bit Format for DS1620 Temperature Data

-Temperature-		-DS1620 Data-		
°F	°C	Binary	Hexadecimal	Decimal
+257	+125	0 11111010	00FA	250
+77	+25	0 00110010	0032	50
+32.9	+0.5	0 00000001	0001	1
+32	0	0 00000000	0000	0
+31.1	-0.5	1 11111111	01FF	511
-13	-25	1 11001110	01CE	462
-67	-55	1 10010010	0192	402

Example conversion of a negative temperature:

-25°C = 1 11001110 in binary. The 1 in the leftmost bit indicates that this is a negative number. Invert the lower eight bits and add 1: 11001110 -> 00110001 + 1 = 00110010 = 50. Units are 0.5°C, so divide by 2. Converted result is -25°C.

or MOSFET switch in serious applications.

For Your Convenience

Dallas Semiconductor offers data and samples of the DS1620 at reasonable cost. Call them at 214-450-0448 or contact your distributor. For a copy of the program listing (plus a thermostat programming example and PIC assembly language source code) on disk, a sample DS1620, and DS1620 documentation, you may order the DS1620 App Kit from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone, 520-459-4802; fax 520-459-0623. Price is $20 postpaid. Visa, Mastercard, American Express, checks, and qualified purchase orders accepted.

```
' Program Listing For Interfacing To DS1620 Digital Thermometer
' Program: DS1620.BAS
' This program interfaces the DS1620 Digital Thermometer to the
' BASIC Stamp. Input and output subroutines can be combined to
' set the '1620 for thermometer or thermostat operation, read
' or write nonvolatile temperature setpoints and configuration
' data.
' ==================== Define Pins and Variables ================
SYMBOL  DQp = pin2          ' Data I/O pin.
SYMBOL  DQn = 2             ' Data I/O pin _number_.
SYMBOL  CLKn = 1            ' Clock pin number.
SYMBOL  RSTn = 0            ' Reset pin number.
SYMBOL  DSout = w0          ' Use bit-addressable byte for DS1620 output.
SYMBOL  DSin = w0           ' "   "   "           word "   "      input.
SYMBOL  clocks = b2         ' Counter for clock pulses.
' ==================== Define DS1620 Constants ==================
' >>> Constants for configuring the DS1620
SYMBOL  Rconfig = $AC       ' Protocol for 'Read Configuration.'
SYMBOL  Wconfig = $0C       ' Protocol for 'Write Configuration.'
SYMBOL  CPU = %10           ' Config bit: serial thermometer mode.
SYMBOL  NoCPU = %00         ' Config bit: standalone thermostat mode.
SYMBOL  OneShot = %01       ' Config bit: one conversion per start request.
SYMBOL  Cont = %00          ' Config bit: continuous conversions after start.
' >>> Constants for serial thermometer applications.
SYMBOL  StartC = $EE        ' Protocol for 'Start Conversion.'
SYMBOL  StopC = $22         ' Protocol for 'Stop Conversion.'
SYMBOL  Rtemp = $AA         ' Protocol for 'Read Temperature.'
' >>> Constants for programming thermostat functions.
SYMBOL  RhiT = $A1          ' Protocol for 'Read High-Temperature Setting.'
SYMBOL  WhiT = $01          ' Protocol for 'Write High-Temperature Setting.'
SYMBOL  RloT = $A2          ' Protocol for 'Read Low-Temperature Setting.'
SYMBOL  WloT = $02          ' Protocol for 'Write Low-Temperature Setting.'
' ==================== Begin Program ============================
' Start by setting initial conditions of I/O lines.
low RSTn                    ' Deactivate the DS1620 for now.
high CLKn                   ' Initially high as shown in DS specs.
pause 100                   ' Wait a bit for things to settle down.

' Now configure the DS1620 for thermometer operation. The
' configuration register is nonvolatile EEPROM. You only need to
' configure the DS1620 once. It will retain those configuration
' settings until you change them--even with power removed. To
' conserve Stamp program memory, you can preconfigure the DS1620,
' then remove the configuration code from your final program.
' (You'll still need to issue a start-conversion command, though.)
let DSout=Wconfig           ' Put write-config command into output byte.
gosub Shout                 ' And send it to the DS1620.
let DSout=CPU+Cont          ' Configure as thermometer, continuous conversion.
gosub Shout                 ' Send to DS1620.
low RSTn                    ' Deactivate '1620.
Pause 50                    ' Wait 50ms for EEPROM programming cycle.
let DSout=StartC            ' Now, start the conversions by
gosub Shout                 ' sending the start protocol to DS1620.
low RSTn                    ' Deactivate '1620.
```

```
' The loop below continuously reads the latest temperature data from
' the DS1620. The '1620 performs one temperature conversion per second.
' If you read it more frequently than that, you'll get the result
' of the most recent conversion. The '1620 data is a 9-bit number
' in units of 0.5 deg. C. See the ConverTemp subroutine below.
Again:
  pause 1000               ' Wait 1 second for conversion to finish.
  let DSout=Rtemp          ' Send the read-temperature opcode.
  gosub Shout
  gosub Shin               ' Get the data.
  low RSTn                 ' Deactivate the DS1620.
  gosub ConverTemp         ' Convert the temperature reading to absolute.
  gosub DisplayF           ' Display in degrees F.
  gosub DisplayC           ' Display in degrees C.
goto Again
' ==================== DS1620 I/O Subroutines =================
' Subroutine: Shout
' Shift bits out to the DS1620. Sends the lower 8 bits stored in
' DSout (w0). Note that Shout activates the DS1620, since all trans-
' actions begin with the Stamp sending a protocol (command). It does
' not deactivate the DS1620, though, since many transactions either
' send additional data, or receive data after the initial protocol.
' Note that Shout destroys the contents of DSout in the process of
' shifting it. If you need to save this value, copy it to another
' register.
Shout:
high RSTn                  ' Activate DS1620.
output DQn                 ' Set to output to send data to DS1620.
for clocks = 1 to 8        ' Send 8 data bits.
  low CLKn                 ' Data is valid on rising edge of clock.
  let DQp = bit0           ' Set up the data bit.
  high CLKn                ' Raise clock.
  let DSout=DSout/2        ' Shift next data bit into position.
next                       ' If less than 8 bits sent, loop.
return                     ' Else return.

' Subroutine: Shin
' Shift bits in from the DS1620. Reads 9 bits into the lsbs of DSin
' (w0). Shin is written to get 9 bits because the DS1620's temperature
' readings are 9 bits long. If you use Shin to read the configuration
' register, just ignore the 9th bit. Note that DSin overlaps with DSout.
' If you need to save the value shifted in, copy it to another register
' before the next Shout.
Shin:
input DQn                  ' Get ready for input from DQ.
for clocks = 1 to 9        ' Receive 9 data bits.
  let DSin = DSin/2        ' Shift input right.
  low CLKn                 ' DQ is valid after falling edge of clock.
  let bit8 = DQp           ' Get the data bit.
  high CLKn                ' Raise the clock.
next                       ' If less than 9 bits received, loop.
return                     ' Else return.
```

```
' ================= Data Conversion/Display Subroutines ===============
' Subroutine: ConverTemp
' The DS1620 has a range of -55 to +125 degrees C in increments of 1/2
' degree. It's awkward to work with negative numbers in the Stamp's
' positive-integer math, so I've made up  a temperature scale called
' DSabs (DS1620 absolute scale) that ranges from 0 (-55 C) to 360 (+125 C).
' Internally, your program can do its math in DSabs, then convert to
' degrees F or C for display.
ConverTemp:
if bit8 = 0 then skip      ' If temp > 0 skip "sign extension" procedure.
  let w0 = w0 | $FE00      ' Make bits 9 through 15 all 1s to make a
                           ' 16-bit two's complement number.
skip:
  let w0 = w0 + 110        ' Add 110 to reading and return.
return
' Subroutine: DisplayF
' Convert the temperature in DSabs to degrees F and display on the
' PC screen using debug.
DisplayF:
let w1 = w0*9/10           ' Convert to degrees F relative to -67.
if w1 < 67 then subzF      ' Handle negative numbers.
  let w1 = w1-67
  Debug #w1, " F",cr
return
subzF:
  let w1 = 67-w1           ' Calculate degrees below 0.
  Debug "-",#w1," F",cr    ' Display with minus sign.
return

' Subroutine: DisplayC
' Convert the temperature in DSabs to degrees C and display on the
' PC screen using debug.
DisplayC:
let w1 = w0/2              ' Convert to degrees C relative to -55.
if w1 < 55 then subzC      ' Handle negative numbers.
  let w1 = w1-55
  Debug #w1, " C",cr
return
subzC:
  let w1 = 55-w1           ' Calculate degrees below 0.
  Debug "-",#w1," C",cr    ' Display with minus sign.
return
```

Stamp Applications no. 3 (May '95):

Adapt a Keypad for Pro-Quality Data Entry; Included Software Lets Stamp "Type" on a PC

Hacking a Commercial Keypad, by Scott Edwards

THIS month's applications make an off-the-shelf numeric keypad for PCs do double duty; first as a serial data-entry terminal for the Stamp, then as a sneaky software method of getting Stamp data into your favorite PC applications, like spreadsheets and word processors.

I've wanted to manufacture a Stamp-compatible keypad, but the cost of starting from scratch is daunting. A decent keypad requires good switches and a sturdy enclosure, both of which are very expensive in the small quantities that the Stamp aftermarket might use. There are occasional surplus bargains, but there's also the risk that the supply will dry up on the very day that Ultra-Mega Industries Inc. places a big order.

I recently found an accessory keypad for laptop computers—an ORTEK MCK-18S/N, made in Taiwan. It communicates with and is powered by a PC's serial port. Included software redirects data from the keypad to the keyboard buffer, so that it functions just like keys on the standard keyboard. The pad has 18 buttons, consisting of the numbers 0 through 9, decimal point, Escape, Num Lock, Enter, and the math operators (/, *, -, +). The keys are quality Alps switches with standard PC-style keycaps. The unit is enclosed in a nice flat-black case.

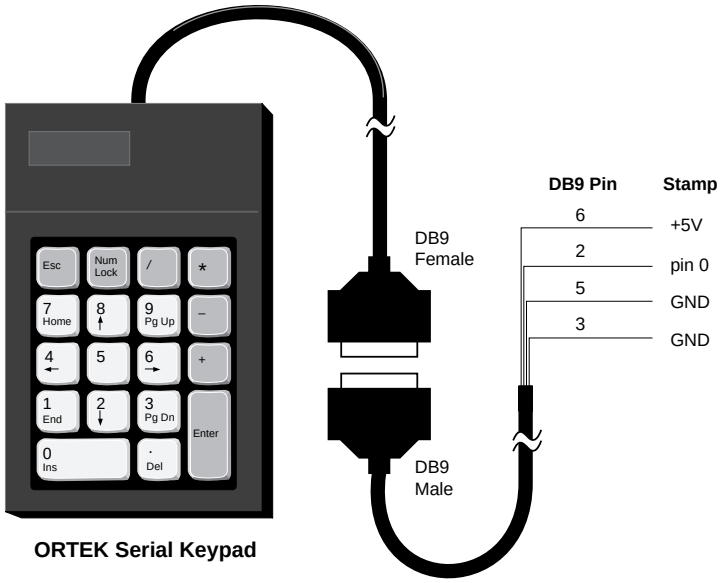


Figure 1. Connecting the serial keypad to the Stamp for data entry requires just one I/O pin plus a few microamps from the Stamp's power supply.

At $49 (see the source list), the keypad isn't cheap, but it's not unreasonably expensive either. So I bought one and proceeded to hack it. Here's what I found:

The heart of the keypad circuitry is a PIC microcontroller, just like the Stamp's own processor. This one runs at a leisurely 38.4 kHz; less than 1/100th the speed of a Stamp. This limits current draw to the 10s of microamps. It also helps the keypad meet FCC limits on radio interference.

When you press one of the pad's keys, its PIC sends a one-byte code (generally a lower-case letter) to the PC via an AT-style DB-9 serial connector. The serial data is sent at 1200 baud, with no parity, 8 data bits, and 1 stop bit (N81). If you hold down a key, the PIC waits a moment, then transmits a string of the same one-byte key code in rapid-fire fashion to simulate the "typeamatic" response of the standard keyboard. When you release a key, the PIC sends a different one-byte code (generally the upper-case version of the key-down code). Table 1 summarizes the codes.

Table 1. Key Codes Used by Unmodified ORTEK Keypad

Key	Key-Down (ASCII)	Key-Up (ASCII)
1	‘ (96)	@ (64)
2	a (97)	A (65)
3	b (98)	B (66)
4	c (99)	C (67)
5	d (100)	D (68)
6	e (101)	E (69)
7	f (102)	F (70)
8	g (103)	G (71)
9	h (104)	H (72)
Esc	i (105)	I (73)
/	j (106)	J (74)
*	k (107)	K (75)
-	l (108)	L (76)
+	m (109)	M (77)
. (point)	n (110)	N (78)
0 (zero)	o (111)	O (79)
Enter	t (116)	T (84)
Num Lock	y (121)	Y (89)

From an electrical standpoint, the keypad is wired to derive power from and send signals to the PC serial port. Its interface generates bipolarity RS-232 signals from the stolen port output voltages. However, the pad will work just fine with a single-ended 5-volt supply, like that of the Stamp. See table 2 for connections.

Table 2. ORTEK Keypad Wiring

DB-9	Color	PC Function	Stamp
2	orange	receive data	serial in
3	yellow	transmit data	ground
6	white	DSR handshake	+5 volts
5	black	ground	ground

Armed with nothing more than the information above, you could make use of the ORTEK keypad for Stamp data entry. Figure 1 shows the hookup, while listing 1 demonstrates how to make sense of the data. However, I couldn't leave well enough alone.

PC Keypad + PIC Program = "Stamp Pad"

The SERIN command can convert strings of ASCII text like "123" into equivalent numbers without the additional programming effort of listing 1. If only the keypad generated terminal-style output...

Instead of wishing, I picked up a soldering iron and removed the keypad's PIC controller. I probed the remaining keypad circuitry and determined the following:

• Like most keypads, this one is wired as a matrix of row and column connections. When a key is pressed, it shorts one row to one column. The processor looks up the row and column coordinates of the short in a table stored in ROM to translate it to a key code.

• The pad's column connections go to the four bits of PIC port RA, which are configured as inputs. (On the Stamp, RA is used to communicate with the PC and manage the EEPROM. It isn't accessible to the user.) When no key is pressed, all bits of RA are pulled high (reading 1s) by four resistors.

• The pad addresses the row connections with

the outputs of a 74HCT138 eight-channel multiplexer. This chip accepts a three-bit address from the lower bits of PIC port RB, and outputs a low on one of its eight outputs. Another bit of RB is used to activate and deactivate the ‘138. Bit RB.7 serves as the serial output. The remaining three bits of RB are unused.

Based on these observations, I coded a replacement PIC in assembly language. Thanks to the clever design of the hardware, this was extremely easy to do. Most of the effort went into preparing tables that unscrambled the row/column coordinate system of the pad, which was apparently designed to permit an inexpensive, one-sided circuit-board layout.

In addition to changing the data format with my new program, I added a hardware feature; a key-acknowledgment beeper. If an inexpensive piezo beeper is connected between pin RB.4 of the PIC and circuit ground, the beeper will make a short “bink” sound to acknowledge each key press. For applications that lack a display to confirm the numbers being entered, this is a tremendous help.

The modified keypad acts like a micro terminal that transmits one ASCII character per keypress (as shown in table 3). The characters correspond to the Stamp’s internal table of the ASCII character set. That is, the following line of Stamp code will respond to the code sent by pressing the “*” key:

```
IF theKey = “*” THEN Asterisk
```

My keypad PIC program has a strict debounce function that requires keys to be pressed and released in a deliberate manner. Speed isn’t compromised, as long as users don’t roll from one key to the other. The first key has to be released before the next will register.

You can make your own Stamp Pad. Figure 2 shows details of the modification. To use the modified pad, connect it to the Stamp as shown in listing 1, then use Serin to collect the data. If, for instance, you want a 16-bit number to be stored to variable w1, use:

```
SERIN 0,N1200,#W1
```

Table 3. Key Codes for “Stamp Pad”

Key	Transmitted Text (ASCII value)
1	1 (49)
2	2 (50)
3	3 (51)
4	4 (52)
5	5 (53)
6	6 (54)
7	7 (55)
8	8 (56)
9	9 (57)
Esc	<ESC> (27)
/	/ (47)
*	* (42)
-	- (45)
+	+ (43)
. (point)	. (46)
0 (zero)	0 (48)
Enter	<RETURN> (13)
Num Lock	<SPACE> (32)

The # symbol before the variable name tells the Stamp to interpret up to five keystrokes as a number ranging from 0 to 65,535 (the range of a 16-bit number). The user must press Enter or any other non-numeric key to finish the entry. If you just want to know which key was pressed, use a byte variable without the #:

```
SERIN 0,N1200,B2
```

After the instruction executes, B2 will contain the ASCII value of the key pressed, as shown in table 3. If you have used SERIN before to accept data from a PC or other computer, this stuff is old hat by now.

You can obtain the Stamp Pad PIC from me, either by buying it outright, or by obtaining the source code free with purchase of my PIC Source Book. This is a collection of assembly language routines for the PIC that mimic the functions of the BASIC Stamp, helping users to move their programs into faster, more efficient PIC hardware. See the source box at the end of this column.

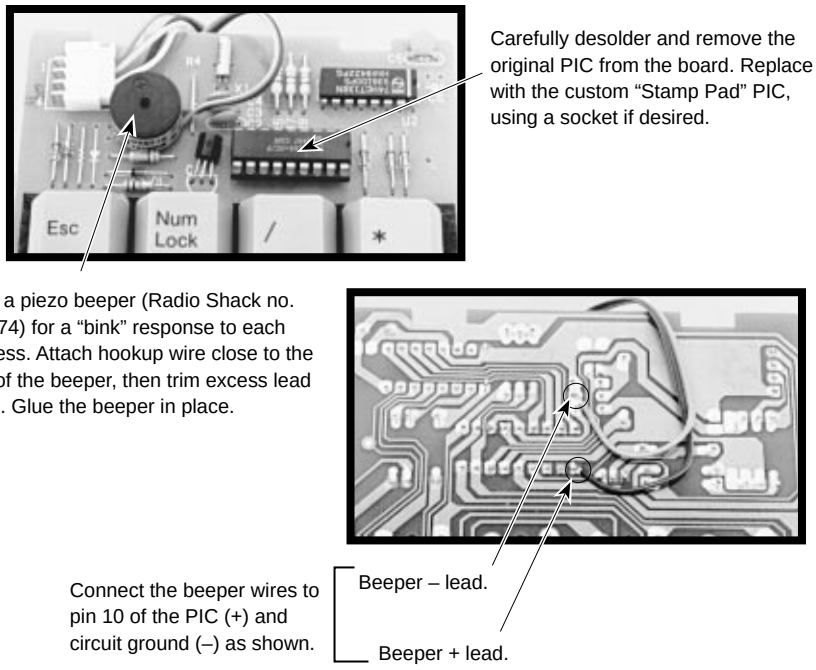


Figure 2. Details of the Stamp Pad modification.

But Wait, There’s More!

There’s an expression about sausage makers using “every part of the pig except the squeal.” In that spirit, I couldn’t ignore the software that came with the keypad. It redirects codes from the serial port to the keypad buffer, allowing you to use numeric input from the pad in any DOS or Windows application.

It occurred to me that if the Stamp were programmed to mimic the key codes, and the keypad software were installed, then the Stamp could “type” data at the keyboard. Think about it--Stamp-collected data magically appears in your spreadsheet, database, or word processor document. No fumbling with terminal software or file transfers.

There are general-purpose software utilities and hardware devices sold for this very purpose, with prices from $100 to $500. They accept any kind of serial input, not just numbers. But if numbers are all you need, check out listing 2. To use the program, install the keypad software according to the instructions that come with the pad. Temporarily disable the keypad driver by typing “KEYPAD OFF” at the DOS prompt. Connect a Stamp programmed with listing 2 as shown in table 4, but don’t connect power to the Stamp yet. Next, turn on the keypad driver by typing “KEYPAD” at the DOS prompt.

Table 4. Stamp-to-PC Connections for Listing 2

Pin	DB9	DB25
pin 0	2	3
GND	5	7

You’re ready to go. For a test drive, boot the BASIC Stamp host program STAMP.EXE, but don’t load a program. Reconnect power to the Stamp. A column of numbers will be typed onto the screen. That’s the Stamp, communicating with your keyboard buffer through the keypad software. Now you can write data-acquisition programs for the Stamp that can communicate directly with any piece of PC software you own!

By the way, the reason for the somewhat roundabout setup procedure above is to avoid trashing your work. When I wrote the program in listing 2, I already had the keypad driver resident in my PC’s memory, and the Stamp connected to the serial port. As soon as I ran the program, it began typing into my just-finished Stamp program listing. I had to quickly

disconnect the Stamp, and erase all of the numbers it had added to the listing.

Conclusion

If you're interested in other keypad solutions for the Stamp, make sure to get Parallax application note no. 3, which shows how to connect a 74C922 16-key pad and an LCD to the Stamp. Looking for an interesting application for our serial keypad? Try Parallax application note no. 6, a stepper-motor driver that can be controlled directly from the Stamp Pad.

Sources

The ORTEK keypad is available from Jameco (part no. 114841) for $49.95, plus applicable shipping and handling charges. To order or request a catalog: Jameco Electronic Components, 1355 Shoreway Drive, Belmont, CA 94002-4100; phone 1-800-831-4242 or 415-592-8097.

The Stamp Pad PIC required to modify the keypad is $10 postpaid from Scott Edwards Electronics. Source code for the Stamp Pad PIC is free with purchase of *The PIC Source Book*, a cookbook of assembly language PIC routines based on the instruction set of the BASIC Stamp. Prototype your ideas with the Stamp, then convert them into fast, professional, one-chip solutions with the help of the cut-and-paste routines from the *Source Book*. Price is $39 postpaid. Order from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623. Use Visa or Mastercard for phone/fax orders; checks, money orders, or POs from approved institutions also accepted. CODs and foreign orders incur additional charges.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

```
' Listing 1: OR_KEYS.BAS (interpret ORTEK MCK-18S/N keypad codes)
' This program accepts serial data from the ORTEK MCK-18S/N numeric
' keypad and converts it into a 16-bit value in variable w1. Users
' must be careful not to hold keys down, or they will activate the
' pad's autorepeat function, causing entry errors.

' Main program loop.
Loop:
gosub GetKeys              ' Getkeys does all the work.
debug w1                   ' Show the result on the PC screen.
goto Loop                  ' Do it forever.

' Subroutine to receive the serial data, filter out extraneous key-up
' codes, and convert received data bytes to a 16-bit value. GetKeys
' will keep accepting and interpreting digits until the user presses
' a non-numeric key, like <Enter>. The routine takes advantage of the
' fact that the key codes for the numbers 1-9 are sequential;
' subtracting 95 from them leaves you with the number itself.
' Although0 (zero) is out of sequence, an additional IF/THEN
' statement recognizes its code ("o").
GetKeys:
let w1 = 0                 ' Clear w1 to start.
Again:
 Serin 0,N1200,b0          ' Get code from the keypad.
 if bit5 = 0 then Again    ' Bit5 is 0 in key-up codes; ignore em.
 if b0 <> "o" then skip    ' Change 0 code from "o" (111) to 95.
 let b0 = 95
skip:
 if b0 > 104 then done     ' Non-numeric key pressed; we're done.
 let b0 = b0 - 95          ' Otherwise convert to 0-9, multiply
 let w1 = w1 * 10 + b0     ' old total by 10, add new value, and
goto Again                 ' get the next digit.
done:
 return                    ' Done: return to main program.
```

```
' Listing 2: FAKE_PAD.BAS (imitate the ORTEK keypad codes)
' This program mimics the codes produced by the ORTEK keypad,
' allowing a PC running the ORTEK software to receive Stamp data
' to its keyboard buffer. The Stamp "types" directly into programs
' that are incapable of normal serial input. To demonstrate this
' capability, the Stamp will count upward by 1 and type the result
' of each calculation to the PC. Remember that the keypad software
' must be installed on the PC for this to work. Before running
' this program, make sure that this program is saved, since the
' Stamp may begin typing numbers into it, if the keypad software
' is active.

SYMBOL nonZero = bit0      ' Leading-zero suppression flag.
SYMBOL code = b1           ' Key code to send.
SYMBOL decade = w2         ' Power-of-10 divisor for conversion.
SYMBOL count = w3          ' Counter for demo.

' Main program loop.
Loop:
 let w1 = count            ' Transfer value of copy to w1.
 gosub typeData            ' "Type" the data to PC.
 pause 100                 ' Wait briefly
 let count = count + 1     ' Increment counter.
goto Loop                  ' Do it forever.

' Subroutine to convert the value stored in w1 to ORTEK keypad codes.
typeData:
let nonZero = 0            ' Clear flag that indicates first non-0 digit.
let decade = 10000         ' Start with highest digit of w1.
nextDigit:
 let code = w1/decade      ' Get value of current digit.
 let w1 = w1//decade       ' Leave remainder in w1.
 if code=0 AND nonZero=0 AND decade <> 1 then skip3   ' No leading 0s.
 if code=0 then skip1
 let nonZero = 1
 goto skip2
skip1:
 let code = 16             ' Code for 0, minus 95.
skip2:
 let code = code + 95
 serout 0,N1200,(code)     ' Send key-down code of digit.
 let bit13 = 0             ' Clear bit5 of b1 (bit13) for key-up code.
 serout 0,N1200,(code)     ' Send key-up code.
skip3:
 let decade = decade/10    ' Get ready for next lower digit
 if decade > 0 then nextDigit
 serout 0,N1200,("tT")     ' Done. Send <Enter> key.
return
```

Stamp Applications no. 4 (June '95):

High-Precision Measurement Made Easy With New 12-bit Analog-to-Digital Converter

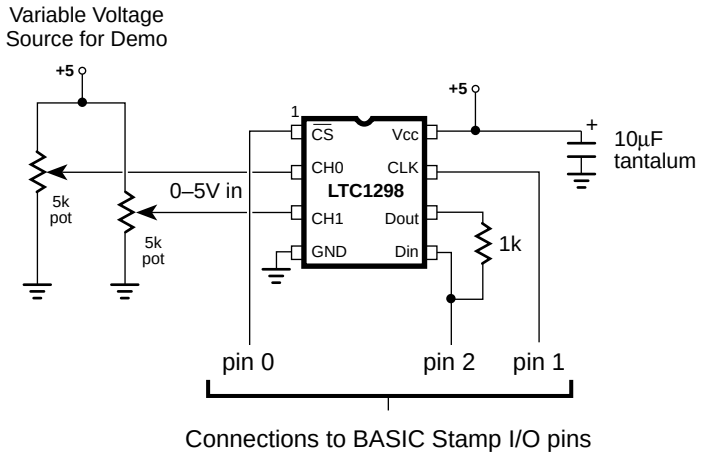
Using the LTC1298, by Scott Edwards

MANY popular applications for the Stamp include analog measurement, either using the built-in Pot (resistance measurement) command or an external analog-to-digital converter (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. An eight-bit conversion gives lousy 2-degree resolution. By reducing the ADC's range, or amplifying the sensor signal, you can improve resolution at the expense of more 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 of 1.22-millivolt resolution data. It's available in a Stamp-friendly 8-pin DIP, and draws about 250 microamps (uA) of current.

The figure shows how to connect the LTC1298 to the Stamp, and the listing supplies the necessary driver code.



Pinout and connection details for the LTC1298.

If you have used other synchronous serial devices with the Stamp, such as EEPROMs, other ADCs or the DS1620 thermometer described in a previous column, 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.

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 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 a source of 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-millivolt 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-millivolt variation means that the voltage would vary 15 millivolts 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.

I suppose the reason for leaving 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 the 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 this voltage snapshot, 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 game 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.

The program listing is thoroughly commented, so I won't waste ink by repeating that stuff here. Instead, I want discuss a subtlety that trips up many Stamp users: the difference between an I/O pin *number* and a pin *variable*.

Look at the beginning of the listing, under ADC Interface Pins. We've assigned names (SYMBOLs) to each of the pins that the Stamp uses to talk to the LTC1298. CS is 0, CLK is 1, and DIO (data in/out) is 2. DIO actually has two SYMBOLs: DIO_n (2) and DIO_p (pin2). Why?

PBASIC refers to the I/O pins in one of two ways; as numbers (0 through 7) or as bit

variables with preassigned names (pin0 through pin7). The following commands use pin numbers:

High, Low, Toggle, Input, Output, Reverse, Pot, Pulsin, Pulsout, PWM, Serin, Serout, Sound

Take High for example. The following are valid PBASIC commands:

```
High 3        ' Make pin 3 output high.
High b2       ' Make the pin number (0-7)
              ' contained in b2 output high.
High bit7     ' Make pin 0 high if bit7 = 0;
              ' make pin 1 high if bit7 = 1.
```

You can specify the pin as either a constant like "3" or as a variable like b2 or bit7.

On the other hand, the math and logic instructions look at the pins as bit variables with a value of 0 or 1. Suppose pin 1 is set up as an input, and the following instruction executes:

```
Let b2 = b2 + pin1  ' Add pin 1 to b2.
```

If there's a 0 at pin 1, 0 is added to b2. If there's a 1 at pin 1, 1 is added to b2. This example points out why we sometimes refer to pins by their numbers, and sometimes by their variable names. Would the following instruction have the same result as the one above?

```
Let b2 = b2 + 1     ' Add 1 to b2.
```

Nope. We have to make the distinction between *1* and *pin1*.

Back to the question I originally posed: Why assign two SYMBOLs for the DIO pin? The reason is that this pin is used with High, which requires a pin number, and also with a Let expression, which requires a variable name. The listing's convention for this is to add the ending "_n" for the number and "_p" for the pin-variable name.

The common bug that arises from the two ways of referring to pins looks like this:

```
High pin3    ' Usually a bug!
```

The programmer believes that he or she is making pin 3 output high. PBASIC interprets this line to mean, "if pin3 = 0 then make pin 0 high; if pin3 = 1 then make pin 1 high." Probably not what the programmer had in mind.

Conclusion. The LTC1298 is a Stamp-friendly peripheral for precision sensing applications. It's available from Digi-Key (800-344-4539) for $8.89 in single quantity as part number LTC1298CN8-ND (8-pin DIP) or LTC1298CS8-ND (surface-mount). Be sure to request a data sheet or order the data book (9210B-ND, $9.95) when you order.

For a copy of the program listing (plus a thermostat programming example and PIC assembly language source code) on disk, a sample LTC1298, and LTC1298documentation, you may order the LTC1298App Kit from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone, 520-459-4802; fax 520-459-0623. Price is $25 postpaid. Visa, Mastercard, American Express, checks, and qualified purchase orders accepted.

```
' Program: LTC1298 (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.
```

Stamp Applications no. 5 (July '95):

Checking Battery Condition and Multiplexing I/O Lines

Two Mini Applications, by Scott Edwards

THIS month's first application was contributed by Guy Marsden of ART TEC, Oakland, California. Guy, a former visual effects specialist for the movies (Star Trek the Motion Picture, Ghostbusters, 2010) makes his living helping artists incorporate electronics into their work. One such work was powered by 12-volt lead-acid battery, and Guy devised a simple, effective way for the Stamp to monitor the battery and sound an alarm at charging time.

Figure 1 and listing 1 show Guy's method. He's using the brightness of the LED as a relative indication of battery voltage. The Stamp reads this brightness as a variable resistance across the photocell. When the photocell resistance exceeds a preset limit, the Stamp sounds an alarm.

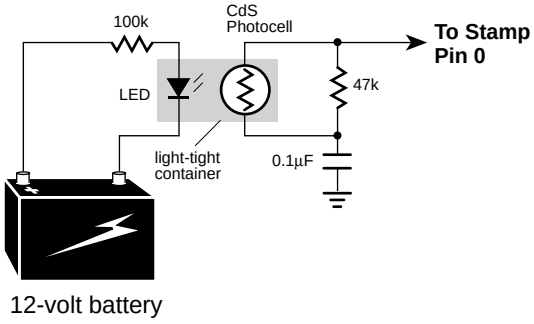


Figure 1. Battery-monitor setup.

Although Guy used a commercially made optoisolator, you should get similar results with a roll-your-own version. Just mount an LED facing a photocell and cover the assembly to block outside light.

This approach can be used more generally to convert a variable voltage into a form that can be read with the Pot command. Just keep a couple of LED characteristics in mind:

• LEDs have a relatively high forward voltage of approximately 1.5 to 2.1 volts. They don't light at all if the input voltage is below their forward voltage. They require a series resistor to make sure that the current through them is reasonable (usually up to 25 mA continuous). To calculate the value of this resistor, use the following formula:

Series Resistor = (Input Voltage–LED Forward Voltage) / LED current

For example, suppose your input is 9 volts. Just guessing, you figure the LED forward voltage at 1.9 volts. And you decide that 10 milliamperes is a safe bet for current. (Remember that most electronic formulas involving current want the value in amperes; 10 mA = 0.01 amperes.) The series resistor should be (9-1.9)/0.01 = 710 ohms. Pick the closest standard value (or the closest value that you have on hand) and you're done.

• An LED's forward voltage decreases with temperature, increasing the current that will pass through it with a given series resistance. More current usually means more light output, but brightness decreases with temperature. To make matters worse, data isn't available for most common LEDs. Even when data is

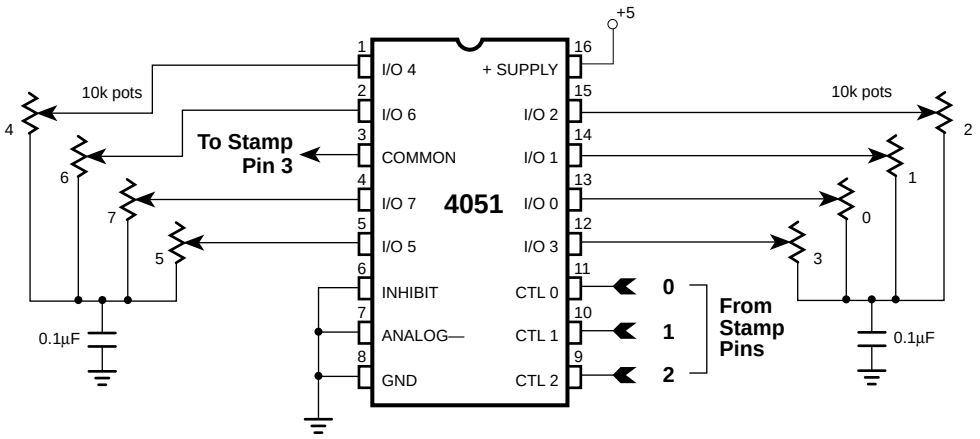


Figure 2. Four Stamp pins read eight pots through a 4051 analog multiplexer.

available, it doesn't usually provide much detail on these effects.

So I'm wondering: Is Guy's circuit usable across a wide range of temperatures, or is it restricted to the great indoors? I started to set up an experiment to find out, then decided that a contest would be more fun.

Here's the deal: Design and construct a Stamp project to test the output of Guy's circuit over a range of at least 32 to 90 degrees F with a steady input voltage. (Temperature control can be as simple as an ice bucket in which the ice is allowed to melt.) Send me a drawing of the circuit, the BASIC program listing, a brief description of your procedure, and a copy of the data generated by it. I'll accept entries until October 31, 1995.

The reader who submits the best entry (my call) will receive the newest version of the LCD Serial Backpack display with 16x1 LCD. This latest model features switchable 2400- or 9600-baud operation, low-voltage reset circuit, improved contrast control, and an easy-to-read 16-character display. All runners-up will receive a coupon good for 10 percent off anything I sell; see the Sources box.

Double Your I/O

The next application is for those of you who can never have enough input/output (I/O) lines. Although I'm demonstrating it with the Pot command, the same method will work with almost any Stamp input or output statment.

Take a look at figure 2 and listing 2. It's based around a 4051 multiplexer/demultiplexer chip. This device works like a digitally controlled rotary switch, shown conceptually in figure 3. Depending on the binary number on its control bits, the chip connects one of eight I/Os to a single common pin. This connection is analog and bidirectional, so you can use it for input, output, or both. The Pot command is a good example of both, since it switches back and forth between input and output as it measures the time required to discharge a capacitor through an unknown resistance.

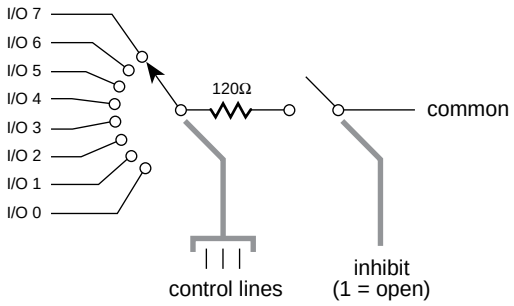


Figure 3. The 4051 multiplexer works like a digitally controlled rotary switch. The three control bits select I/O 0 (000 binary) through 7 (111 binary).

For an investment of four pins--three control bits and one common—you receive a dividend of eight I/Os. Better than Wall Street most days.

There are a few limitations to this trick. The first is evident if you try the pot demonstration. You can never quite get a reading of zero; the

lowest the readings will go is about 8. This is because the 4051's electronic rotary switch isn't perfect--it has a resistance of approximately 120 ohms.

A second limitation is that inputs to the 4051 must never exceed 0.5 volts above the supply voltage or below ground. This means that the 4051 can't be used to directly receive RS-232 serial signals through a series resistor as the Stamp's I/O pins can.

The last restriction is common to both the 4051 and the Stamp, so it really isn't much of a limitation: Current through the 4051 should never exceed 25 mA.

If you like the 4051, but still need more I/O, try the 4067. It provides 16 I/Os for five Stamp pins. Check your favorite data book (such as the CMOS Cookbook by Don Lancaster) for details.

Sources

Enter the LCD Serial Backpack contest and win a new LCD display for your Stamp projects (a $40 value) or 10 percent off Stamp accessories and microcontroller projects seen here in Nuts & Volts. Entries (or questions, suggestions, requests) to: Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; or via Internet 72037.2612@compuserve.com.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

```
' Listing 1. BASIC Stamp Battery Monitor
' CHEAP AND SIMPLE LOW BATTERY WARNING
' Guy Marsden, March 1995.  tekart@well.com
'
' Uses a Cds opto coupler with LED input such as: CLM6000.
' Put a 33 or 47K resistor across cell and a .1uF cap to gnd.
' Use a high value resistor in series with the LED wired directly
' to the battery.  I used 100k for a 12volt system.  This value is
' enough to produce a resistance of 20k at nominal battery voltage.
'
' When the voltage drops, the Cds resistance increases.  By setting
' the SCALE of the POT function under a low battery condition, you
' will then have a range to work with.  Determine your setpoint
' using a variable bench supply and a DVM.

symbol Batt = b2
symbol LoBatt = 220

CheckBatt:
        pot 0,76,Batt          ' check battery voltage
        if Batt > LoBatt Alarm      ' if less than established value
goto CheckBatt

Alarm:
        sound 1,100,100        ' beep piezo alarm
        pause 100
goto Alarm
```

```
' Listing 2. Multiplexing Stamp I/O Lines
' Program: MULTIPOT.BAS (Multiple pots using a 4051 multiplexer)
' This program demonstrates how to connect and measure multiple
' pots using a 4051 multiplexer chip. The 4051's control inputs
' (11,10,9) connect to Stamp pins 0, 1, and 2 respectively.
' The common I/O pin (3) of the 4051 goes to Stamp pin 3.
' By writing a value between 0 and 7 to its pins, the Stamp can
' select one of eight variable-resistance inputs through the
' 4051. See the schematic for details.

SYMBOL pot_sel = b2           ' Pot number 0-7 selected through 4051.
SYMBOL pot_val = b3           ' Result of the pot measurement.

Let dirs = %0111              ' Make the lower 3 pins outputs to drive 4051.
Again:
  for pot_sel = 0 to 7        ' For each of the eight pots:
   let pins = pot_sel         ' Write pot number to the 4051.
   pot 3,150,pot_val          ' Perform pot measurement on selected pot.
   debug "pot #",#pot_sel," ",#pot_val,cr  ' Display result.
  next pot_sel                ' Read the next pot.
  pause 2000                  ' Wait two seconds.
  debug cr                    ' Insert a carriage return on screen.
goto Again                    ' Do it again (endless loop).
```

Stamp Applications no. 6 (August '95):

Silicon Steroids for the Stamp
Help Your Projects Heft Big Loads

Using Switching Trasistors, by Scott Edwards

ONE of the outstanding characteristics of the PIC microcontroller used in the BASIC Stamp is its ability to directly drive loads like LEDs through its input/output pins. The Stamp can source (conduct to +5) up to 20 mA and sink (conduct to ground) up to 25 mA. Total current sourced or sunk by all eight pins should not exceed 40 or 50 mA, respectively.

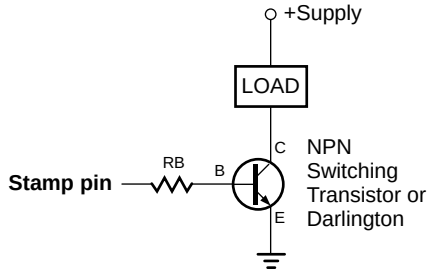
Now I'll grant that 25 mA doesn't sound like a lot of current. But shop around. Other microcontrollers make a big deal out of having a few "high-current" pins capable of sourcing 2 mA and sinking 10.

OK, so the Stamp is a muscle-bound brute by microcontroller standards. It still gets sand kicked in its face by applications that need to drive motors, incandescent bulbs, relays, solenoids, etc. In today's column we're going to pump up the Stamp to new levels of power.

Let's start with the basic transistor switch. To keep matters simple, we will limit our discussion to current-sink capability--switching current to ground.

The tools for the job are simple and easy to obtain; a resistor and an NPN transistor. Figure 1 shows the capabilities of a common 2N2222, a high-gain transistor, and a low-power Darlington transistor.

In figure 1, you can think of the collector (C) and emitter (E) of the transistor as forming a switch to ground. Current at the base turns the switch on. If Stamp pin 0 were connected to this circuit, the instruction High 0 would turn on current to the load; Low 0 would turn it off.



Transistor	RB	Base Current	Load Current	Max +Supply	C–E Voltage Drop	Remarks
2N2222	390Ω	11 mA	100 mA	30 V	0.5 V	Common
ZTX689B	390Ω	11 mA	2 A	12 V	0.5 V	High-gain
ZTX605	3.3k	1 mA	1 A	100 V	1.5 V	Darlington

Figure 1. Simple one-transistor switch boosts the Stamp's current-switching capability.

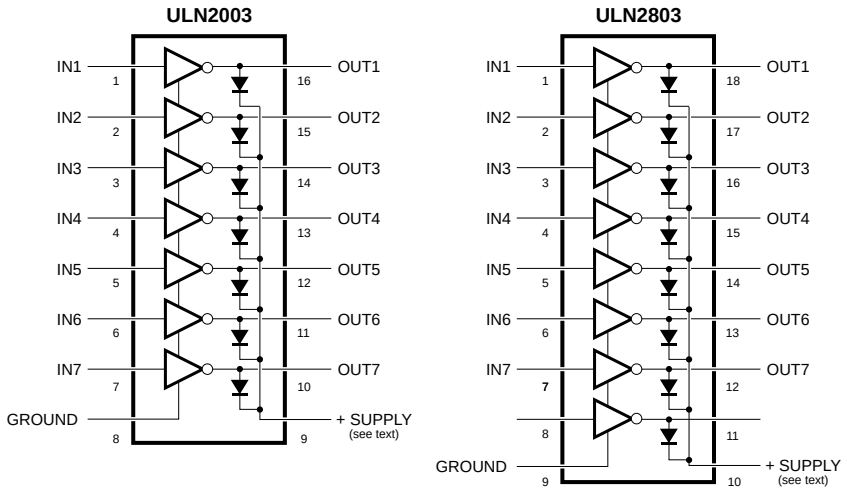


Figure 2. Handy power switches on-a-chip: the ULN 2x03 ICs.

Transistors switches are good, but not perfect. See the column in figure 1 that says C-E Voltage Drop? The voltage across the load will be that much smaller than the supply voltage. For example, if you're driving a flashlight bulb with a 2N2222 transistor and a 6-volt supply, the bulb will actually see only 5.5 volts. The other half volt will be lost, "dropped," across the transistor.

The voltage drop, multiplied by the current drawn by the load, gives you the power in watts wasted by the switching transistor. Where does the wasted power go? It produces heat, sometimes lots of it. For example, the Zetex ZTX689B can conduct as much as 2 amperes of current with a 0.5-volt drop, losing 1 watt to waste heat in the process. When you consider that a small soldering iron has a 15-watt heating element, you can imagine how hot that little transistor can get. Larger switching transistors have metal tabs on their cases for attachment to heat sinks. The large surface area and other properties of the heat sink help spread and dissipate all that waste heat. (A 15-watt soldering iron is very hot; an electric blanket with the same wattage is barely warm.)

All three transistors in the figure-1 table are compatible with Stamp output capabilities, but the third--the Darlington transistor--looks especially good from the Stamp's point of view. It requires only 1 mA to drive a 1-ampere load. But whoa, the C-E voltage drop is terrible at 1.5 volts. Can't we get a Darlington with better C-E specs?

In a word, no. Darlingtons consist of two NPN transistor in the same case wired in a way that multiplies their overall gain (ratio of current in to current controlled). In the process, the Darlington adds one base-to-emitter junction worth of voltage drop, approximately 0.7 volts, to the C-E drop.

Despite this drawback, Darlingtons are so handy for interfacing logic to loads that IC manufacturers offer arrays of seven or eight Darlington switches, complete with appropriate base resistors and protective diodes, in neat IC packages. Figure 2 shows two such units, the ULN2003 and ULN2803. These interface directly to a Stamp pin to drive loads of up to 500 mA per output.

The input pins of the ULN2x03s can connect directly to Stamp I/O pins. They're equivalent to connecting the pin to the left of RB in figure 1. The output pins of the ULN2x03s are equivalent to the collector (C) connection of the transistor switch in figure 1. They provide a switched ground connection for the load.

The ULN2x03s also feature something not shown in figure 1, a series of diodes connected to their outputs. When the devices are used to power inductive loads like relays and motors, these diodes should be connected to the positive

supply that powers the load. When one of the ULN2x03 switches cuts power to the inductive load, the load's magnetic field collapses, generating a nasty negative power spike. The diodes short out this spike, preventing it from damaging the transistor switch.

If you construct your own switches with discrete transistors, you'd do well to copy this protective feature. Just add a common rectifier diode like a 1N4002 with its banded end (cathode; the negative connection when the diode is conducting) to the + connection of the relay or motor. The diode won't interfere with the normal operation of the motor or relay, but it'll snub those spikes!

A real-world example

Francis Rogers of Sun City West, Arizona wrote me to describe an application he'd like to build with the Stamp. He has a PC with barcode software that can read membership cards for his S.C.W. Metals Club. The barcode software generates a code through the PC serial port when it's presented with a valid card. Mr. Rogers would like the Stamp to read this code and energize a relay to unlatch a door.

This fits perfectly with the theme of this month's column. Figure 3 is the schematic. I've made some assumptions about Mr. Rogers' barcode software: that it can be set to output at 2400 baud, and that all valid cards output some common code for the Stamp to recognize.

Thanks to the serial-input (Serin) command's built-in "qualifier" feature, the entire program takes just a few lines of Stamp code:

```
loop:
  low 7         ' Pin 7 low to latch door
                ' (relay open)
  serin 0,N2400,("OK")  ' Watch serial input
                ' until "OK" rec'd.
  high 7        ' Pin 7 high to unlatch door
                ' (relay closed).
  pause 5000    ' Wait 5 seconds.
goto loop       ' Latch door and resume
                ' watching serial input.
```

Of course, Mr. Rogers will have to substitute the actual password for "OK" in the program above.

That's it for this month. Next time, we'll look at a nifty IC that lets the Stamp transmit and receive DTMF tones (telephone touch tones). In addition to obvious telephone applications, DTMF can be used as a form of low-speed, high-reliability data transfer. Stay tuned!

Sources

For switching transistors and ULN2x03 ICs get a catalog from Jameco Electronic Components, 1355 Shoreway Road, Belmont, CA 94002-4100; phone 1-800-831-4242.

Questions, suggestions or comments about this column? Contact Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; or via Internet 72037.2612@compuserve.com.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

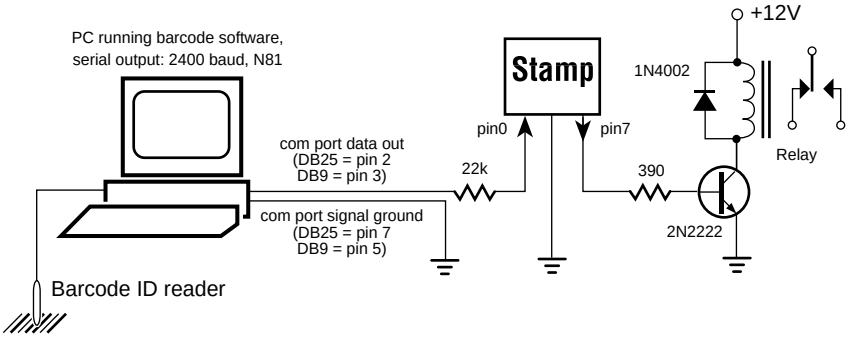


Figure 3. When the Stamp receives the proper code from the PC bar-code reader, it activates a relay to unlock the door.

Stamp Applications no. 7 (September '95):

DTMF "Touch" Tones are Music to the Ears of this Stamp Transmit/Receive Circuit

Working with the CM8880 DTMF Transceiver, by Scott Edwards

ENCODING and decoding DTMF tones—those musical "touch tones" you hear when you dial the phone—is the current fad in electronics projects. There are gadgets that capture and display or store dialed digits, others that dial stored numbers, and still others that use the hardy dual-tone, multifrequency digits for data transmission or remote control.

Stamp users are an independent lot, so we're going to buck the trend of one-trick DTMF projects. In this edition of Stamp Applications, we'll look at a universal DTMF send/receive solution that can serve as the basis for decoders, loggers, dialers, and even data transceivers.

A Quick Review of DTMF Principles

All the recent excitement over the decades-old DTMF signaling method makes the first part of my job easy. I can breeze through the explanation of the signals themselves with confidence that the curious reader who wants more background can find it his or her recent-magazine stack, for example, *DTMF IR Remote Control System*, *N&V* June '95.

A DTMF signal consists of a mixture of two sinewave tones, often called the *row* and *column* frequencies because of their correspondence to the layout of the phone keypad (figure 1). Engineers at then-mighty Ma Bell chose these tones carefully to ensure that none had a harmonic relationship with the others and that mixing the frequencies would not produce sum or product frequencies that could mimic another valid tone. They specified that the tones be free of distortion, and that the high-group frequencies (the column tones) be slightly louder than the low-group to compensate for the high-frequency rolloff of voice audio systems.

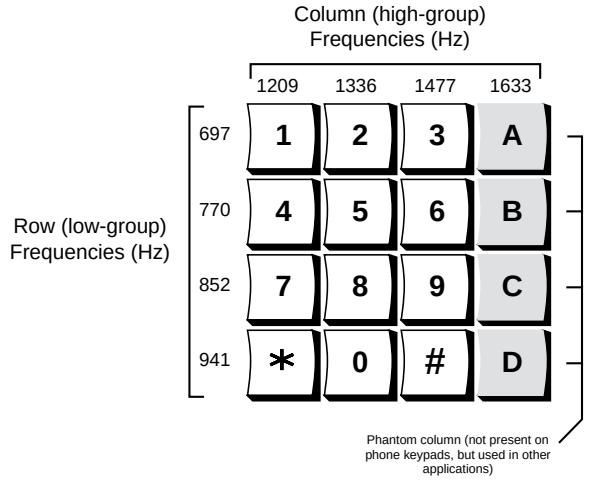


Figure 1. The DTMF tone that's generated when you press a button on the phone keypad consists of a mixture of the row and column frequencies. For example, pressing 9 mixes a column frequency of 1477 Hz with a row frequency of 852 Hz.

Because DTMF was devised in the days before you could hear a pin drop or appreciate Whitney Houston's singing over long distance, it's an exceptionally noise-resistant signaling method. If a properly designed DTMF decoder thinks there's a valid DTMF signal present in some lousy audio, there almost surely is.

DTMF tones can represent one of 16 different states or symbols, as shown in figure 1. That is equivalent to four bits of data, also known as a nibble. Most DTMF decoders can process at least 10 tones per second under the worst of conditions, so DTMF can easily convey 40 bits (5 bytes) of data per second. That's nowhere near the performance of a good communications modem, which can operate nearly 600 times as fast (28,800 bits per second), but it's a lot more robust under noisy line conditions.

Note that the numbers and symbols on the phone keypad don't always match the binary values of the DTMF nibbles. Most notably, the "0" on the keypad is represented in DTMF by a value of 10 (decimal) or 1010 binary. Table 1 summarizes the rest of the four-bit values.

Table 1. DTMF Values, Keypad Symbols

Binary Value	Decimal Value	Keypad Symbol
0000	0	D
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	0
1011	11	*
1100	12	#
1101	13	A
1110	14	B
1111	15	C

A DTMF Transceiver: the CM8880

All you need to generate DTMF tones are two precise, low-distortion sine-wave oscillators and a mixer. To decode DTMF takes just eight tone decoders and a little glue logic. The circuit board to accommodate all this stuff shouldn't be much more than 4 by 6 inches and $50.

Don't like those numbers? Then you will like the California Micro Devices CM8880 Integrated DTMF Transceiver. For $10 or less in single quantity, this chip does everything associated with sending and receiving DTMF tones. The schematic in figure 2 shows a Stamp hookup for both send and receive applications. Listing 1 demonstrates DTMF dialing. Listing 2 shows DTMF reception and display.

Rather than rehash the comments from the program listings, I'd like to show you how to set up and use the CM8880 in your own applications.

Communication with the 8880 takes place over a 4-bit bus, consisting of D0 through D3, with three additional bits selecting modes of operation. Those bits are chip select (CS), read/write (RW) and register select (RS0). Table 2 summarizes the effects of all combinations of CS, RW, and RS0.

To sum up the table, the 8880 is active only when CS is 0. The RW bit determines the data direction; 1 = read (data from 8880 to Stamp) and 0 = write (data from Stamp to 8880). The RS bit determines whether the transaction involves data (DTMF tones) or internal CM8880 functions (instructions or status); 1 = instructions/status and 0 = data.

Once the CM8880 is set up, the Stamp writes 000 to CS, RW, and RS0 to send DTMF; or 010 to read DTMF. Simple as that.

Setting up the CM8880

Before you can use the CM8880, you have to set it up. The device has two control registers, A and B. When you put 001 in bits CS, RW and RS0, the data on D0 through D3 is written to

Table 2. Effects of the CS, RW, and RS0 Bits

CS	RW	RS0	Description
0	0	0	Active: write data (i.e., send DTMF)
0	0	1	Active: write instructions to 8880
0	1	0	Active: read data (i.e., receive DTMF)
0	1	1	Active: read status from 8880
1	0	0	Inactive
1	0	1	Inactive
1	1	0	Inactive
1	1	1	Inactive

Table 3. Functions of Control Register A

Bit	Name	Function
0	Tone Out	0 = tone generator disabled
		1 = tone generator enabled
1	Mode Control	0 = Send and receive DTMF
		1 = Send DTMF, receive call-progress tones
		(DTMF bursts lengthened to 104 ms)
2	Interrupt Enable	0 = Make controller check for DTMF rec'd
		1 = Interrupt controller via pin 13 when DTMF rec'd
3	Register Select	0 = Next instruction write goes to CRA.
		1 = Next instruction write goes to CRB.

Table 4. Functions of Control Register B

Bit	Name	Function
0	Burst	0 = Output DTMF bursts of 52 or 104 ms
		1 = Output DTMF as long as enabled
1	Test	0 = Normal operating mode
		1 = Present test timing bit on pin 13
2	Single/ Dual	0 = Output dual (real DTMF) tones.
		1 = Output separate row or column tones
3	Column/ Row	0 = If above = 1, select row tone.
		1 = If above = 1, select column tone.

control register A (CRA). Table 3 summarizes the functions of the four bits of CRA.

Looking at listing 1, the dialer, we see that 1011 was written to CRA. That works out to (right to left) 1 (tone generator on), 1 (Send DTMF, long tones), 0 (don't generate interrupts), and 1 (next write to CRB). Listing 2 writes 1000 to CRA: 0 (tone-generator off), 0 (receive DTMF), 0 (don't generate interrupts) and 1 (next write to CRB).

Although we don't use it in our applications, the 8880 has a call-progress detection feature, which can be enabled via bit 1 of CRA. Call-progress is a collective term for the dial tone, busy signal, and ringing signal. These tones are all around 400 Hz. The CM8880's call-progress

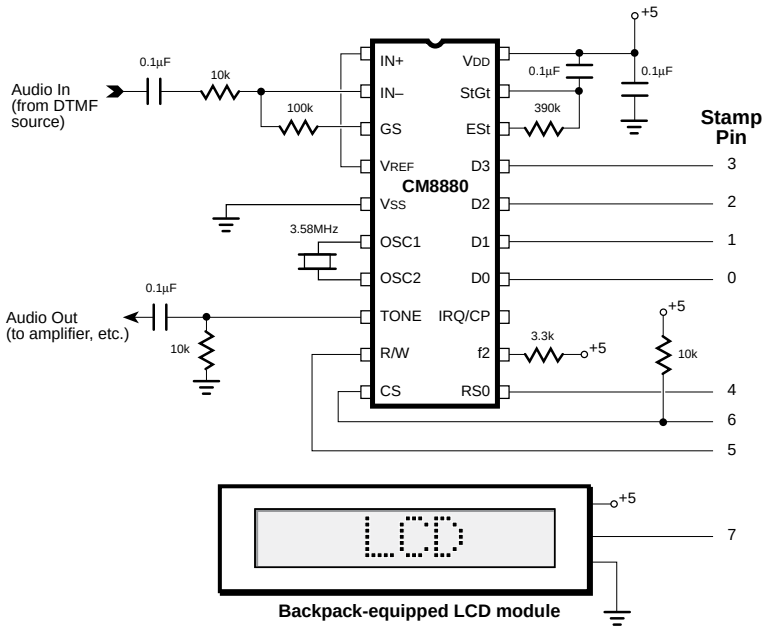


Figure 2. Schematic diagram for DTMF send and receive applications.

filter accepts audio from the normal input and passes frequencies from 300 to 500 Hz to the IRQ/CP pin. You could add hardware to further filter and detect these tones.

Since both listings also set up register B (CRB), take a look at table 4 to see how it works.

Listing 1 (dialer) writes 0s to all bits of CRB, meaning burst mode, normal operation, real DTMF, and (if DTMF weren't selected) row tones. Listing 2 (decoder) also clears CRB. Although CRB has little or nothing to do with DTMF reception, it's wise to always initialize it to a known state.

A Few Hardware Notes

To hear the DTMF tones generated by the CM8880, I connected a small speaker/amplifier (Radio Shack part no. 277-1008C) to the audio output. The amplifier's high gain and the 8880's healthy audio output quickly taught me to set the volume control low! I was able to dial the phone by holding the speaker close to the mouthpiece.

I used two approaches to generate DTMF test tones for the decoder application. The first was to connect an old touch-tone phone to a 9V battery and the audio-input coupling capacitor (0.1µF shown in figure 2) to the positive terminal. The polarity of the connection didn't matter, as phones are designed to tolerate wiring mixups. However, this setup sucked better than 50 mA from the battery.

I found a better DTMF generator at one of my favorite surplus dealers. Marlin P. Jones and Associates (MPJA) has a bunch of dialers that were apparently made for one of those early flash-in-the-pan long-distance services. It's preprogrammed to autodial an access number. If you press the manual-dial button (MAN'L) on the keypad, it works like a normal DTMF keypad. Output is through a headphone speaker built into the case.

MPJA also stocks a couple of other essentials for this circuit. The decoder application uses an LCD equipped with one of my LCD Serial Backpacks. This allows the Stamp to display data on it using a convenient one-wire serial connection. MPJA stocks both the Backpack and a range of LCDs from 1-line by 16 characters to 2x40. See Sources for more information.

DTMF and a glass of red...

It's almost embarrassing to admit that that's all there is to designing DTMF send/receive applications based on the CM8880 and the Stamp. I deliberately skirted issues having to do with direct connection to the phone line, since that's a subject worthy of two (or three!) articles of this size. Your best bet is to see how others have done it successfully, or to purchase a

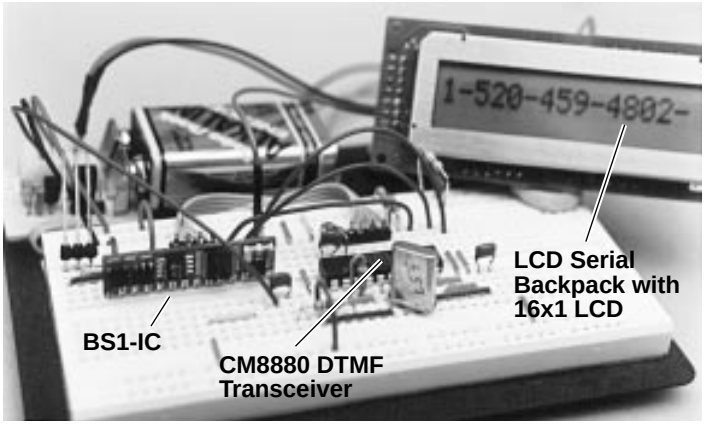


Figure 3. DTMF send/receive setup.

commercially made Data Access Arrangement (DAA) to serve as an interface. One manufacturer of DAAs is Cermetek, Sunnyvale, CA, 408-752-5000.

Don't limit yourself to phone applications. Remember what I said about DTMF serving as a reliable data-transfer method? You could connect CM8880-equipped Stamps to inexpensive two-way radios to create a wireless network for data acquisition over a 1/4-mile radius.

I experimented briefly with this idea, using the DTMF output from the amplified speaker to trigger the voice-activated transmit feature of a Radio Shack headset walkie-talkie (part no 21-406). The voice activation was fast enough that even the first tone got through. This means that you wouldn't need an additional pin to switch between transmit and receive.

This is an application that a lot of Stamp users have been clamoring for. I remember in particular that grape growers need to gather and evaluate "microclimate" data regarding temperature and humidity among their pampered vines, and would kill for a wireless way to obtain it. Here it is: Cheers. (And I prefer red, if you please, a Cabernet Sauvignon or Merlot.)

Return of the Counterfeit

Many of you have called my order line to purchase BASIC Stamps. It's a natural assumption: I'm cheerleading for a great product—I've got to *carry* that product, right?

Unfortunately, I don't. However, the manufacturer (Parallax) makes PBASIC chips available to those who want to build their own controllers based on Stamp technology. I showed how to do just that in my "Counterfeit Stamp" article last spring (*N&V*, May '94).

In response to many requests, I'm making the Counterfeit available again as a complete kit for $29. The design incorporates features I have found handy in developing applications you've seen here: heavy-duty (100+ mA) voltage regulator, socket for pull-up/pull-down resistors, ground and +5V connections paired with every I/O pin, and Turbo options for running the Counterfeit at 8 or 16 MHz—two to four times standard speed. The board is compact, but easy to assemble; see the photo (figure 4).

I have also put together a Counterfeit Development System for $69. This includes everything required to program Counterfeits (or BS1 Stamps, for that matter), including your first Counterfeit kit. See the Sources listing for further information.

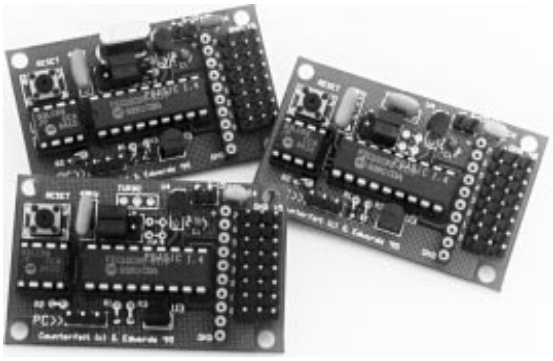


Figure 4. A trio of Stamp-compatible Counterfeit Controllers.

Sources

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101; e-mail info@parallaxinc.com.

The CM8880 is available from electronics distributors who carry California Micro Devices' product line (CMD phone, 602-961-6000), or from Scott Edwards Electronics (see listing below).

The DTMF box (part no. 5824-EN, $4.95), LCD Serial Backpack (7074-SE, $29), and a variety of LCD displays are available from Marlin P. Jones and Associates, P.O. Box 12685, Lake Park, FL 33403-0685; phone 407-848-8236; fax 407-844-8764.

Scott Edwards Electronics:

Questions, suggestions, or requests for future Stamp Applications to: Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; Internet 72037.2612@compuserve.com. Scott also offers the following Stamp-related kit goodies:

The CM8880 is $10; an App Kit containing one CM8880, the Stamp programs shown here (plus PIC assembly language versions) on disk, and complete documentation, is $22.

The Counterfeit controller, a kit alternative to the BASIC Stamp, is $29. Double- and quad-speed options are $2 and $4, respectively. The Counterfeit Development System, required to program Counterfeits (also for programming original BASIC Stamps, like the BS1-IC) is $69 and includes 150-page manual, downloading cable kit, Parallax software, and one Counterfeit controller kit.

Prices are postpaid (express shipping and CODs extra). Visa, Mastercard, American Express accepted for phone/fax orders. POs accepted on approved credit. Personal checks, money orders welcome for orders by mail.

' Listing 1. Stamp-Based Autodialer

```
' Program: DIAL.SRC (Sends a string of DTMF tones via the 8880)
' This program demonstrates how to use the CM8880 as a DTMF tone
' generator. All that's required is to initialize the 8880 properly,
' then write the number of the desired DTMF tone to the 8880's
' 4-bit bus.

' The symbols below are the pin numbers to which the 8880's
' control inputs are connected, and one variable used to read
' digits out of a lookup table.

SYMBOL      RS_p = 4          ' Register-select pin (0=data).
SYMBOL      RW_p = 5          ' Read/Write pin (0=write).
SYMBOL      CS_p = 6          ' Chip-select pin (0=active).
SYMBOL      digit = b2        ' Index of digits to dial.

' This code initializes the 8880 for dialing by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high. See the accompanying
' article for an explanation of the 8880's registers.

let pins = 255          ' All pins high to deselect 8880.
let dirs = 255          ' Set up to write to 8880 (all outputs).
let pins = %00011011    ' Set up  register A, next write to register B.
high CS_p
let pins = %00010000    ' Clear register B; ready to send DTMF.
high CS_p
```

```
' This for/next loop dials the seven digits of my fax number. For
' simplicity, it writes the digit to be dialed directly to the output
' pins. Since valid digits are between 0 and 15, this also takes RS,
' RW, and CS low--perfect for writing data to the 8880. To complete
' the write, the CS line is returned high. The initialization above
' sets the 8880 for tone bursts of 200 ms duration, so we pause
' 250 ms between digits. Note: in the DTMF code as used by the phone
' system, zero is represented by ten (1010 binary) not 0. That's why
' the phone number 459-0623 is coded 4,5,9,10,6,2,3.

for digit = 0 to 6
  lookup digit,(4,5,9,10,6,2,3),pins ' Write current digit to pins.
  high CS_p                  ' Done with write.
  pause 250                  ' Wait to dial next digit.
next digit
end
```

' Listing 2. DTMF Decoder and Display

```
' Program: DTMF_RCV.BAS (Receives and display DTMF tones using the 8880)
' This program demonstrates how to use the 8880 as a DTMF decoder. As
' each new DTMF digit is received, it is displayed on an LCD Serial
' Backpack screen. If no tones are received within a period of time
' set by sp_time, the program prints a space (or other selected character)
' to the LCD to record the delay. When the display reaches the righthand
' edge of the screen, it clears the LCD and starts over at the left edge.

SYMBOL      RS_p = 4          ' Register-select pin (0=data).
SYMBOL      RW_p = 5          ' Read/Write pin (0=write).
SYMBOL      CS_p = 6          ' Chip-select pin (0=active).
SYMBOL      dtmf = b2         ' Received DTMF digit.
SYMBOL      dt_Flag = bit0    ' DTMF-received flag.
SYMBOL      home_Flag = bit1  ' Flag: 0 = cursor at left edge of LCD.
SYMBOL      polls = w2        ' Number of unsuccessful polls of DTMF.
SYMBOL      LCDw = 16         ' Width of LCD screen.
SYMBOL      LCDcol = b3       ' Current column of LCD screen for wrap.
SYMBOL      LCDcls = 1        ' LCD clear-screen command.
SYMBOL      I = 254           ' LCD instruction toggle.
SYMBOL      sp_time = 1000    ' Print space this # of polls w/o DTMF.

' This code initializes the 8880 for receiving by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high.

let pins = %01111111    ' Pin 7 (LCD) low, pins 0 through 6 high.
let dirs = %11111111    ' Set up to write to 8880 (all outputs).
let pins = %00011000    ' Set up register A, next write to register B.
high CS_p
let pins = %00010000    ' Clear register B.
high CS_p
let dirs = %11110000    ' Now make set the 4-bit bus to input.
high RW_p               ' And set RW to "read."
serout 7,n2400,(I,LCDcls,I)   ' Clear the LCD screen.

' In the loop below, the program checks the 8880's status register
' to determine whether a DTMF tone has been received (indicated by
' a '1' in bit 2). If no tone, the program loops back and checks
```

```
' again. If a tone is present, the program switches from status to
' data (RS low) and gets the value (0-15) of the tone. This
' automatically resets the 8880's status flag.
again:
  high RS_p             ' Read status register.
  low CS_p              ' Activate the 8880.
  let dt_flag = pin2    ' Store status bit 2 into flag.
  high CS_p             ' End the read.
if dt_Flag = 1 then skip1     ' If tone detected, continue.
let polls = polls+1           ' Another poll without DTMF tone.
if polls < sp_time then again ' If not time to print a space, poll again.
if LCDcol = LCDw then skip2   ' Don't erase the screen to print spaces.
let dtmf = 16                 ' Tell display routine to print a space.
gosub Display                 ' Print space to LCD.
skip2:
let polls = 0           ' Clear the counter.
goto again              ' Poll some more.
skip1:                  ' Tone detected:
let polls = 0           ' Clear the poll counter.
low RS_p                ' Get the DTMF data.
low CS_p                ' Activate 8880.
let dtmf = pins & %00001111   ' Strip off upper 4 bits using AND.
high CS_p               ' Deactivate 8880.
gosub display           ' Display the data.
goto again              ' Do it all again.

Display:
if LCDcol < LCDw then skip3   ' If not at end of LCD, don't clear screen.
serout 7,n2400,(I,LCDcls,I)   ' Clear the LCD screen.
let LCDcol = 0                ' And reset the column counter.
skip3:                        ' Look up the symbol for the digit.
if LCDcol=0 AND dtmf=16 then ret    ' No spaces at first column.
lookup dtmf,("D1234567890*#ABC-"),dtmf
serout 7,n2400,(dtmf)         ' Write it to the Backpack display.
let LCDcol = LCDcol + 1       ' Increment the column counter.
ret:
return
```