Introduction. This application note shows how to use the new Xout command to remotely control X-10® lamp and appliance modules.

Background. Home automation—the management of lights and appliances with a computer—promises to increase security, energy efficiency, and convenience around the house. So why aren't home-control systems more common? The answer is probably the wiring; it's hard to think of a nastier job than stringing control wiring through the walls and crawlspaces of an existing home.

Fortunately, there's a wireless solution for home control called X-10, a family of control modules that respond to signals sent through existing AC wiring. The BASIC Stamp II has the built-in ability to generate X-10 control signals with the new Xout instruction.

How it works. From the user's standpoint, an X-10 system consists of a control box plugged into a wall outlet, and a bunch of modules plugged into outlets around the house. The appliances and lights to be controlled are plugged into the modules.

During the installation of the system, the user assigns two codes to each of the modules; a house code and a unit code. As the name suggests, the house code is usually common to all modules in a particular house. There are 16 house codes, assigned letters A through P. The idea of the house code is to avoid interference between adjacent homes equipped with X-10 by allowing the owners to assign different codes to their modules. The control box must be assigned the same house codes as the modules it will control.

There are also 16 unit codes (numbered 1 through 16) that identify the modules within a particular house. If your needs expand beyond 16 modules, it's generally safe to use another house code for the next group of 16, since few if any neighborhoods are so infested with X-10 controllers that all available house codes are taken. X-10 signals don't propagate beyond the nearest utility transformer.

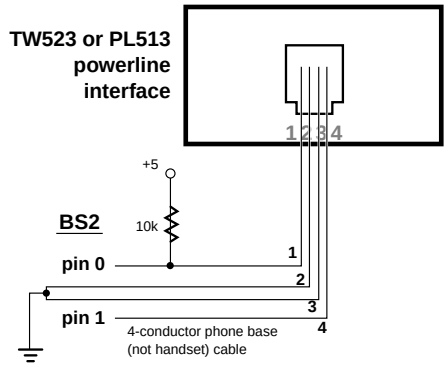
Once this simple setup is complete, the user controls the modules by pressing keys on the control box. Pressing "1 ON" turns module 1 on.

From a more technical standpoint, X-10 signals are digital codes imposed on a 120-kHz carrier that is transmitted during zero crossings of the AC line. To send X-10 commands, a controller must synchronize to the AC line frequency with 50-microsecond precision, and transmit an 11-bit code sequence representing the button pressed.

A company named X-10 owns a patent on this system. To encourage others to use their technology without infringing their patent, X-10 sells a pair of modules that provide a relatively simple, safe, UL- and CSA-approved interface with the AC power line. These interfaces are the PL-513 and TW-523. The PL-513 is a transmit-only unit; the TW-523 can be used to transmit and receive X-10 codes. The Stamp II presently supports only transmission of X-10 codes, but either of the interfaces may be used. The figure shows how they connect to the Stamp II.

A word of caution: The PL-513 or TW-523 provide a safe, opto-isolated interface through their four-pin modular connector. However, they derive power directly from the AC power line. Never open the cases of these devices to make connections or measurements. You'll be exposing yourself to a severe—even deadly—shock hazard.

That said, connecting to the PL-513 or TW-523 is easy. They use a standard four-conductor modular phone base (not handset) connector. Cutting a 12-foot phone cord in half yields two 6-foot X-10 cables. The



Schematic to accompany X10_DEMO.BS2

color codes can vary in phone cables, so be sure to follow the numbers imprinted next to the modular jack on the PL-513 or TW-523 unit.

The program listing shows how to send X-10 commands through this hookup. The listing is self-explanatory, and the procedures are simple, as long as you keep some ground rules in mind:

- House codes A through P are represented as values from 0 to 15 in the Xout command.
- Unit codes 1 through 16 are represented as values from 0 to 15 in the Xout command.
- Every X-10 transmission must include a house code.
- Except for Dim and Bright, all codes are sent for a default of two cycles. You don't have to specify the number of cycles for commands other than Dim and Bright, unless you are sending multiple codes in a single instruction. See the listing for examples.
- It takes 19 cycles for a lamp to go from fully bright to fully dim and vice versa. There's also a peculiar logic to the operation of the Dim and Bright commands (see the table). To set a specific level of brightness, you should first reset the dimmer module by turning it off, then back on.
- In some homes, X-10 signals may not be able to reach all outlets without a little help. A device called an ACT CP000 Phase Coupler (about $40 retail from the source below) installed on the electrical breaker box helps X-10 signals propagate across the phases of the AC line. For larger installations, there are also amplifiers, repeaters, etc.

Sources. X-10 compatible modules are available from many home centers, electrical suppliers, and electronics retailers, including Radio

Operation of the Dim and Bright Commands

Lamp is...	And you send... OFF	ON	DIM	BRIGHT
OFF/FULL	no effect	turns ON/FULL	turns ON/FULL	turns ON/FULL
ON/FULL	turns OFF/FULL	no effect	dims	no effect
OFF/DIM	turns OFF/FULL	no effect	no effect	brightens
ON/DIM	turns OFF/FULL	no effect	dims	brightens

Shack. However, relatively few of these carry the PL-513 and TW-523. Advanced Services Inc., a home-automation outlet, sells every conceivable sort of X-10 hardware, including the PL-513 and TW-523 (starting at around $20 at the time of this writing). You may contact them at 800-263-8608 or 508-747-5598.

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

```
' Program: X10_DEMO.BS2 (Demonstration of X-10 control using Xout)
' This program--really two program fragments--demonstrates the
' syntax and use of the new XOUT command. Basically, the command
' works like pressing the buttons on an X-10 control box; first you
' press one of 16 keys to identify the unit you want to control,
' then you press the key for the action you want that unit to
' take (turn ON, OFF, Bright, or Dim). There are also two group-action
' keys, Lights ON and All OFF. Lights ON turns all lamp modules on
' without affecting appliance modules. All OFF turns off all modules,
' both lamp and appliance types.

' Using  XOUT requires a 4-wire (2-I/O pin) connection to a PL-513 or
' TW-523 X-10 module. See the application note for sources.
zPin      con       0               ' Zero-crossing-detect pin from TW523 or PL513.
mPin      con       1               ' Modulation-control pin to TW523 or PL513.

' X-10 identifies modules by two codes: a House code and a Unit code.
' By X-10 convention, House codes are A through P and Unit codes are
' 1 through 16. For programming efficiency, the Stamp II treats both
' of these as numbers from 0 through 15.
houseA    con       0               ' House code: 0=A, 1=B... 15=P
Unit1     con       0               ' Unit code: 0=1, 1=2... 15=16
Unit2     con       1               ' Unit code 1=2.

' This first example turns a standard (appliance or non-dimmer lamp)
' module ON, then OFF. Note that once the Unit code is sent, it
' need not be repeated--subsequent instructions are understood to
' be addressed to that unit.

xout mPin,zPin,[houseA\Unit1]     ' Talk to Unit 1.
xout mPin,zPin,[houseA\uniton]    ' Tell it to turn ON.
pause 1000                        ' Wait a second.
xout mPin,zPin,[houseA\unitoff]   ' Tell it to turn OFF.

' The next example talks to a dimmer module. Dimmers go from full
```

```
' ON to dimmed OFF in 19 steps. Because dimming is relative to
' the current state of the lamp, the only guaranteed way to set a
' predefined brightness level is to turn the dimmer fully OFF, then
' ON, then dim to the desired level. Otherwise, the final setting of
' the module will depend on its initial brightness level.

xout mPin,zPin,[houseA\Unit2]                    ' Talk to Unit 2.
' This example shows how to combine X-10 instructions into a
' single line. We send OFF to the previously identified unit (Unit2)
' for 2 cycles (the default for non-dimmer commands). Then a comma
' introduces a second instruction that dims for 10 cycles. When you
' combine instructions, don't leave out the number of cycles. The
' Stamp may accept your instruction without complaint, but it
' won't work correctly--it may see the house code as the number of
' cycles, the instruction as the house code, etc.

xout mPin,zPin,[houseA\unitoff\2,houseA\dim\10]

' Just to reinforce the idea of combining commands, here's the
' first example again:

xout mPin,zPin,[houseA\Unit1\2,houseA\uniton]   ' Turn Unit 1 ON.
pause 1000                                      ' Wait a second.
xout mPin,zPin,[houseA\Unit1\2,houseA\unitoff]  ' Turn Unit 1 OFF.

' End of program.
stop
```

BASIC Stamp II Application Notes

Introduction. This application note shows how to use the new Shiftin and Shiftout instructions to efficiently interface the BASIC Stamp II to synchronous serial peripheral chips.

Background. Many of the most exciting peripheral chips for microcontrollers are available only with synchronous-serial interfaces. These go by various names, like SPI, Microwire, three- or four-wire interface, but they are essentially the same in operation. The BASIC Stamp II takes advantage of these similarities to offer built-in instructions—Shiftout and Shiftin—that take most of the work out of communicating with synchronous-serial peripherals.

Before plunging into the nuts and bolts of using the new instructions, let's discuss some fundamentals. First of all, how does a synchronous-serial interface differ from a parallel one? Good question, since most synchronous-serial devices incorporate elements of both serial and parallel devices.

The building block of both parallel and serial interfaces is called a flip-flop. There are several types, but we're going to discuss the D-type or Data flip-flop. A D-type flip-flop has two inputs (Data and Clock) and one output (typically called Q). When the logic level on the Clock input rises (changes from 0 to 1), the flip-flop stores a snapshot of the logic level at the Data input to the Q output. It holds that bit on Q until the power is turned off, or until the opposite state is present on Data when Clock receives another 0-to-1 change. (For the sake of conversation, we call a 0-to-1 transition a "rising edge" and 1-to-0 a "falling edge.")

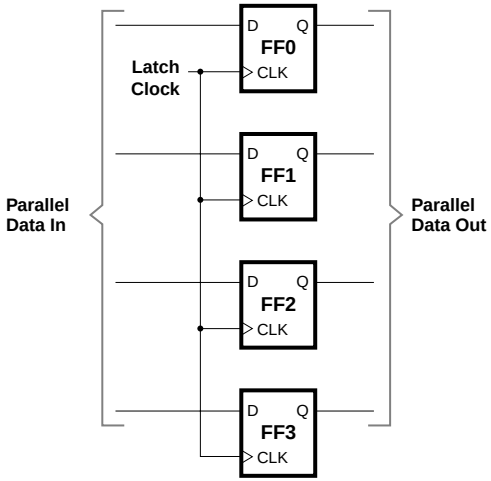


Figure 1. Parallel latch.

The action of a D-type flip-

flop is described as "latching" Data onto Q. Parallel latches, like the one shown in figure 1, allow several bits to be simultaneously latched onto a set of outputs. This is one of the ways that a computer addresses multiple devices on a single parallel data bus—it puts the data on the bus, then triggers one device's Clock. The data is latched into the destination device only; other devices ignore the data until *their* Clock lines are triggered.

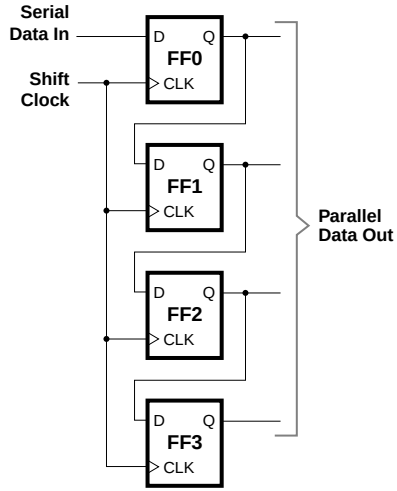


Figure 2. Serial shift register.

With different wiring, the parallel latch becomes a serial one, known as a shift register (figure 2). See how this works: When a rising edge appears on the Clock input, all of the flip-flops latch their Data inputs to their Q outputs. Because they are wired in a chain with each Q output connected to the next flip-flop's Data input, incoming bits ripple down the shift register.

You can picture this process as working like a bucket brigade or a line of people moving sandbags. In perfect coordination, each person hands their burden to the next person in line and accepts a new one from the previous person.

Looking at this from the standpoint of the parallel output, there's a potential problem. When data is being clocked into the shift register, the data at the output isn't stable—it's rippling down the line. The cure for this is to add the previously described parallel latch after the shift register, and clock it only when we're finished shifting data in. That's the arrangement shown in figure 3.

It isn't too much of a stretch to imagine how this kind of circuit could be turned around and used as an input. Data would be grabbed in parallel by a latch, then transferred to a shift register to be moved one bit at a time to a serial data output.

Now you understand the communications hardware used in synchronous serial peripherals; it's basically just a collection of shift registers, latches and other logic. The Stamp II's built-in Shiftout and Shiftin instructions provided general-purpose tools for working with this kind of hardware. Let's look at some examples.

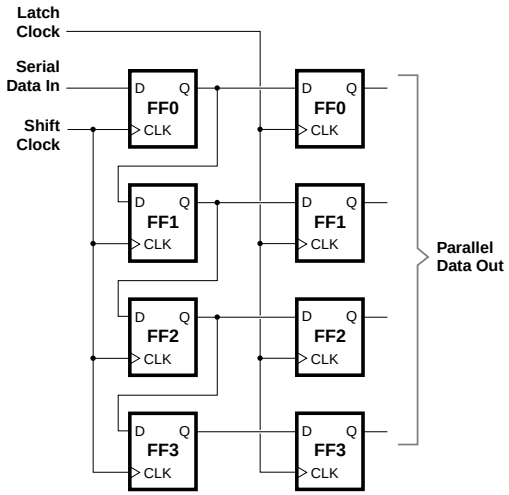


Figure 3. A shift register plus a latch makes a serial-to-parallel converter.

Shift-Register Output with Shiftout. The most basic use for Shiftout is to add an output-only port based on a shift register / latch combination like the 74HC595 shown in figure 4. Listing 1 demonstrates how simple it is to send data to a device like this.

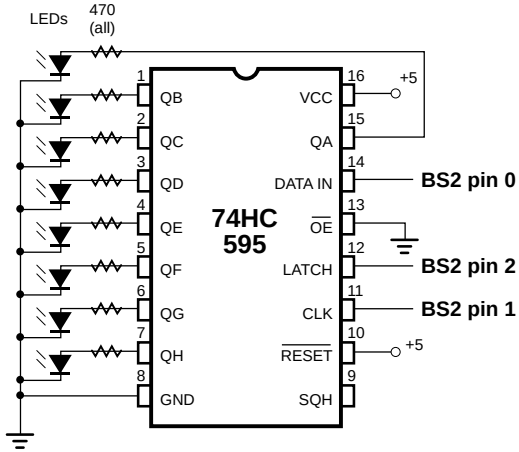


Figure 4. Schematic to accompany 74HC595.BS2.

Shiftout requires just five pieces of information to do its job:

- The pin number of the data connection.
- The pin number of the shift-clock connection.
- The order in which the bits should be sent—least-significant bit (lsb) first or most-significant bit (msb) first. For the '595, we chose msb first, since the msb of the output is farthest down the shift register from the data input. For other devices, the order of bits is prescribed by the manufacturer's spec sheet.
- The variable containing the data to output.
- The number of bits to be sent. (If this entry is omitted, Shiftout will send eight bits).

Note that once the data is shifted into shift register, an additional program step—pulsing the Latch line—is required to move the data to the output lines. That's because the 74HC595 is internally similar to the schematic in figure 3. The two-step transfer process prevents the outputs from rippling as the data is shifted.

The 74HC595 also has two control lines that are not used in our demonstration, but may prove useful in real-world applications. The Reset line, activated by writing a 0 to it, simultaneously clears all of the shift register flip-flops to 0 without affecting the output latch. The Output-enable (OE) line can effectively disconnect the output latch, allowing other devices to drive the same lines. A 0 on OE connects the outputs; a 1 disconnects them.

Serial ADC with Shiftin. Figure 5 and listing 2 demonstrate how to use Shiftin to obtain data from an 8-bit serial analog-to-digital converter, the ADC0831.

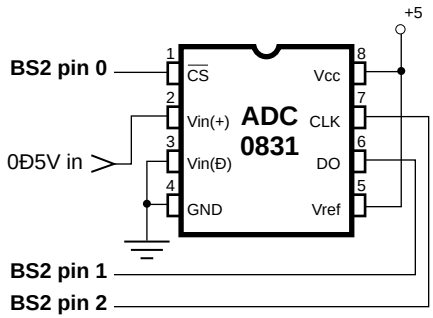


Figure 5. Schematic for ADC0831.BS2.

Shiftin requires the same five pieces of information as

Shiftout, plus one more, the relationship of valid data to clock pulses. Some devices latch bits onto the serial data output on the rising edge of the clock line. Output bits remain valid until the next rising edge. In these cases, your program must specify *post*-clock input for the Shiftin mode. When bits are latched on the falling edge of the clock, specify *pre*-clock input.

With pre-clock input, we sometimes encounter a chicken-and-egg problem. How can the first bit be clocked out *before* the first clock pulse? It can't, of course. The simple solution is to specify one additional bit in the Shiftin instruction.

However, most serial peripherals require that some instructions be sent to them before they return any data. In this case, the falling edge of the last Shiftout clock cycle clocks the first bit of the following pre-clock Shiftin instruction.

Serial ADC with Shiftout and Shiftin. The third example (figure 6, listing 3) uses Shiftout and Shiftin to hold a two-way conversation with an LTC1298 ADC. An initial Shiftout sends configuration bits to the LTC1298 to select channel and mode, then a Shiftin gets the 12-bit result of the conversion. The program listing concentrates on the mechanics of the Shift instructions; for more detailed information on the ADC itself, see Stamp Application Note #22 or the manufacturer's spec sheet.

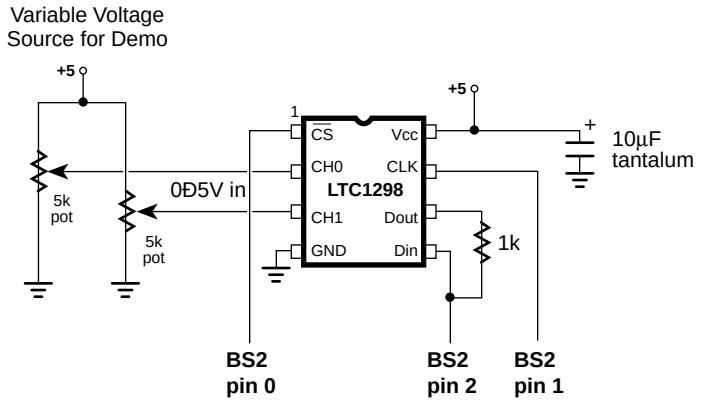


Figure 6. Schematic for LTC1298.BS2.

Custom Shift Routines. The key to successful use of the Shift instructions is obtaining, reading, and understanding the manufacturer's specification sheets. In addition to providing the data required to fill in the parameters for the Shift instructions, the data sheets document configuration bits, operating modes, internal register arrangements, and lots of other valuable data.

Sources. The components used in the example applications are available from Digi-Key, 710 Brooks Avenue South, P. O. Box 677, Thief River Falls, MN 56701-0677; phone 1-800-344-4539. Packages of components, documentation, and source code listings for the Stamp I, Stamp II and PIC microcontrollers are available from Scott Edwards Electronics; phone 520-459-4802, fax 520-459-0623. These packages, known as AppKits, are available for the LTC1298 ADC, DS1620 digital thermometer, Xicor X25640 8-kB EEPROM, and others.

Program listings. These programs may be downloaded from our Internet ftp site at ftp.parallaxinc.com. The ftp site may be reached directly or through our web site at http://www.parallaxinc.com.

```
' LISTING 1. SHIFTOUT TO 74HC595
' Program: 74HC595.BS2 (Demonstrate 74HC595 shift register with Shiftout)
' This program demonstrates the use of the 74HC595 shift register as an
' 8-bit output port accessed via the Shiftout instruction. The '595
' requires a minimum of three inputs: data, shift clock, and latch
' clock. Shiftout automatically handles the data and shift clock,
' presenting data bits one at a time on the data pin, then pulsing the
' clock to shift them into the '595's shift register. An additional
' step—pulsing the latch-clock input—is required to move the shifted
' bits in parallel onto the output pins of the '595.

' Note that this application does not control the output-enable or
' reset lines of the '595. This means that before the Stamp first
' sends data to the '595, the '595's output latches are turned on and
' may contain random data. In critical applications, you may want to
' hold output-enable high (disabled) until the Stamp can take control.

DataP   con     0       ' Data pin to 74HC595.
Clock   con     1       ' Shift clock to '595.
Latch   con     2       ' Moves data from shift register to output latch.
counter var     byte    ' Counter for demo program.

' The loop below moves the 8-bit value of 'counter' onto the output
```

```
' lines of the '595, pauses, then increments counter and repeats.
' The data is shifted msb first so that the most-significant bit is
' shifted to the end of the shift register, pin QH, and the least-
' significant bit is shifted to QA. Changing 'msbfirst' to 'lsbfirst'
' causes the data to appear backwards on the outputs of the '595.
' Note that the number of bits is _not_ specified after the variable
' in the instruction, since it's eight, the default.
Again:
  Shiftout DataP,Clock,msbfirst,[counter]  ' Send the bits.
  pulsout Latch,1                          ' Transfer to outputs.
  pause 50                                 ' Wait briefly.
  counter = counter+1                      ' Increment counter.
goto Again                                 ' Do it again.
```

```
' LISTING 2. SHIFTIN FROM ADC0831
' Program: ADC0831.BS2
' This program demonstrates the use of the BS2's new Shiftin instruction
' for interfacing with the Microwire interface of the Nat'l Semiconductor
' ADC0831 8-bit analog-to-digital converter. It uses the same connections
' shown in the BS1 app note.

ADres       var     byte        ' A-to-D result: one byte.
CS          con     0           ' Chip select is pin 0.
AData       con     1           ' ADC data output is pin 1.
CLK         con     2           ' Clock is pin 2.

high CS                         ' Deselect ADC to start.

' In the loop below, just three lines of code are required to read
' the ADC0831. The Shiftin instruction does most of the work. Shiftin
' requires you to specify a data pin and clock pin (AData, CLK), a
' mode (msbpost), a variable (ADres), and a number of bits (9). The
' mode specifies msb or lsb-first and whether to sample data before
' or after the clock. In this case, we chose msb-first, post-clock.
' The ADC0831 precedes its data output with a dummy bit, which we
' take care of by specifying 9 bits of data instead of 8.

again:
  low CS                                ' Activate the ADC0831.
  shiftin AData,CLK,msbpost,[ADres\9]   ' Shift in the data.
  high CS                               ' Deactivate '0831.
  debug ? ADres                         ' Show us the conversion result.
  pause 1000                            ' Wait a second.
goto again                              ' Do it again.
```

```
' LISTING 3. BIDIRECTIONAL COMMUNICATION WITH LTC1298
' Program: LTC1298.BS2 (LTC1298 analog-to-digital converter)
' This program demonstrates use of the Shiftout and Shiftin instructions
' to communicate with an LTC1298 serial ADC. Shiftout is used to
' send setup data to the ADC; Shiftin to capture the results of the
' conversion. The comments in this program concentrate on explaining
' the operation of the Shift instructions. for more information on
' the ADC, see Stamp app note #22 or the Linear Tech spec sheets.

CS          con     0           ' Chip select; 0 = active
CLK         con     1           ' Clock to ADC; out on rising, in on falling edge.
DIO_n       con     2           ' Data I/O pin _number_.
config      var     nib         ' Configuration bits for ADC.
AD          var     word        ' Variable to hold 12-bit AD result.

startB      var     config.bit0 ' Start bit for comm with ADC.
sglDif      var     config.bit1 ' Single-ended or differential mode.
oddSign     var     config.bit2 ' Channel selection.
msbf        var     config.bit3 ' Output 0s after data xfer complete.

' 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 ADC to begin.
high DIO_n                      ' Set data pin for first start bit.
again:                          ' Main loop.
  for oddSign = 0 to 1          ' Toggle between input channels.
    gosub convert               ' Get data from ADC.
    debug "channel ",DEC oddSign, ": ",DEC AD,cr    ' Display data.
    pause 500                   ' Wait a half second.
  next                          ' Change channels.
goto again                      ' Endless loop.

' Here's where the conversion occurs. The Stamp first sends the config
' bits to the 1298, then clocks in the conversion data. Note the use of
' the new BS2 instructions Shiftout and Shiftin. Their use is pretty
' straightforward here: Shiftout sends data bits to pin DIO and clock
' the CLK pin. Sending the least-significant bit first, it shifts out
' the four bits of the variable config. Then Shiftin changes DIO to
' input and clocks in the data bits—most-significant bit first, post
' clock (valid after clock pulse). It shifts in 12 bits to the variable AD.

convert:
  config = config | %1011       ' Set all bits except oddSign.
  low CS                        ' Activate the ADC.
  shiftout DIO_n,CLK,lsbfirst,[config\4]' Send config bits.
  shiftin DIO_n,CLK,msbpost,[AD\12] ' Get data bits.
  high CS                       ' Deactivate the ADC.
return                          ' Return to program.
```

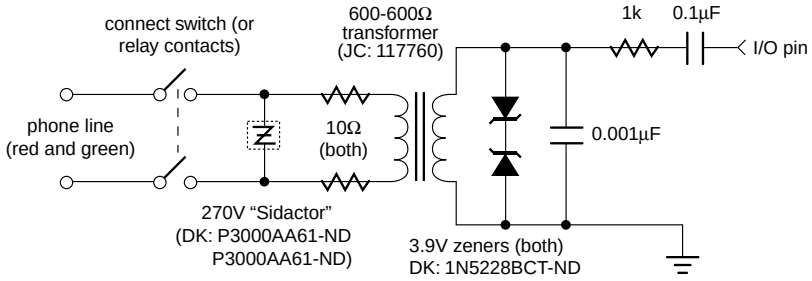
Introduction. This application note shows how to interface the BS2 to the phone line in applications that use the DTMFout instruction.

Background. The BS2 instruction DTMFout generates dual-tone, multifrequency signals—the same musical beeps used to dial the phone, activate pagers, and access repeaters in ham-radio applications.

Commercial designs that interface electronic devices to the phone line normally require the approval of the Federal Communications Commission (FCC) to ensure the quality and reliability of telephone service. Manufacturers of phone accessories often take the shortcut of using an off-the-shelf interface, known as a Data Access Arrangement (DAA). Since the DAA has already been checked out by the FCC, it's generally much easier to get a DAA-based design approved than a from-scratch circuit. Unfortunately, DAAs tend to be somewhat expensive in small quantities ($25+ each), and are sold primarily through high-volume distributors geared toward serving manufacturers.

Where does this leave experimenters, hobbyists, and one-off instrument makers? Pretty much on their own. For them, we present the circuit below. It's not a full-blown DAA suitable for production designs, but it is a good starting point for prototype DTMF-transmit

Interfacing to the Telephone Line



Parts Sources

Digi-Key (DK), 1-800-344-4539 or 218-681-6674	Jameco (JC), 1-800-831-4242 or 415-592-8097

Schematic of the phone-line interface.

applications using the BS2. It's based on a circuit presented in *Encyclopedia of Electronic Circuits, Volume 5*, by Graf and Sheets (TAB / McGraw Hill, 1995; ISBN 0-07-011077-8). We've filled in specific component values and sources, added parts for coupling the BS2, and tested the circuit's ability to dial the phone.

How it works. Starting at the phone-line end of the circuit, a double-pole single-throw (DPST) switch or set of relay contacts isolates the circuit from the phone line when the circuit is not in use. Closing the switch puts the phone into the "off-hook" condition, which causes the phone company to generate a dialtone. Although a single set of contacts would be sufficient to break the circuit, a tradition of robust design in phone circuits makes it normal for a hook switch to break both sides of the circuit.

After the switch, a Sidactor surge-protection device clips large voltage spikes that might result from nearby lightning strikes. Its voltage rating is selected to let it do its surge-protection job without interfering with relatively high ringing voltages or phone-company test voltages. Note that nothing can provide 100-percent lightning immunity, but the Sidactor is cheap insurance against most routine surges.

A 600-to-600-ohm transformer isolates the BS2 from the line's DC voltages. On the other side of the transformer, a pair of zener diodes clips any voltage over approximately 4.6 volts. The remaining resistors and capacitors couple the DTMF tones from the BS2 into the transformer. They also work together to smooth the ragged edges of the DTMF tones, which are generated using fast pulse-width modulation (PWM). Before filtering, these tones contain high-frequency components that can make them sound distorted or fuzzy. With the circuit shown, the tones come through crystal clear.

Programming. You'll be amazed at how easy it is to dial the phone with the DTMFout instruction. Suppose you want to dial 624-8333—one line will do the trick:

```
DTMFout 0,[6,2,4,8,3,3,3]
```

where 0 is the pin number (0-15) connected to the interface and the

numbers inside the square brackets are the numbers to dial. Values of 0-9 represent those same buttons on the phone keypad; 10 is the star (*) key; 11 is the pound sign (#); and 12 through 15 are additional tones that aren't meant for phone-subscriber use. They're included primarily for non-phone DTMF applications like remote controls and ham-radio purposes. You may specify values as literal numbers, as we did above, or as variables. Nibble-sized variables are perfect for holding DTMF digits.

For each digit in square brackets, DTMFout sends the corresponding tone for 200 milliseconds (ms), followed by a silent pause of 50 ms. This timing gives the phone company equipment plenty of time to recognize and respond to the tones. If you want some other timing scheme, you can place on and off times between the pin numbers and the tone list, like so:


```
DTMFout 0,1000,500,[6,2,4,8,3,3,3]
```

That instruction would transmit each tone for a full second (1000 ms), and pause in silence for a half second (500 ms) after each tone.

Sources. Components needed for the simple phone-line interface are available from Digi-Key and Jameco; see the contact information in the schematic. For commercial applications, one manufacturer of DAAs is Cermetek Microelectronics, 406 Pasman Drive, Sunnyvale, CA 94089; phone 800-882-6271; fax 408-752-5004.

BASIC Stamp II Application Notes