Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

SoundPAL (#28825): *Miniature Sound Player*

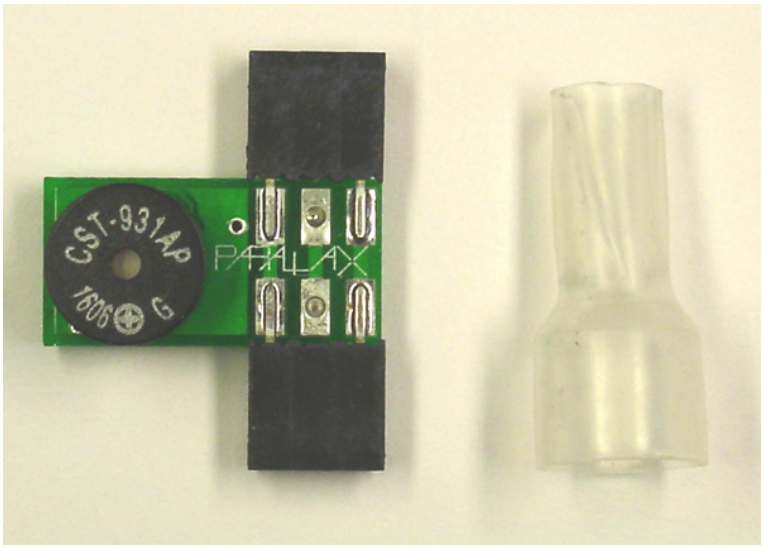
General Description

The SoundPAL is a tiny module that plays canned and custom sound sequences. It is completely self-contained, including a microcontroller for generating the sounds and a small speaker for producing them. The SoundPAL interfaces easily to a BASIC Stamp and can play sounds while the BASIC Stamp is busy with other chores.

Features

- Plugs into servo headers, and works with protoboards.
- Virtually goof-proof power sourcing makes it nearly impossible to connect wrong.
- Single-pin interface uses a simple serial protocol to define and initiate sound production.
- Canned tunes and sound effects can be played with simple commands.
- Custom tunes and effects are simple to program using musical notation instead of frequencies.
- Total audio range is 6½ octaves in four different tempos and four playing styles.
- Onboard EEPROM permits saving custom sound sequences for later playback.
- Autoplay feature permits playing a pre-designated EEPROM sequence with only a power supply.
- Compact size: stackable side-to-side with additional modules on 0.1" servo headers.

What's Included



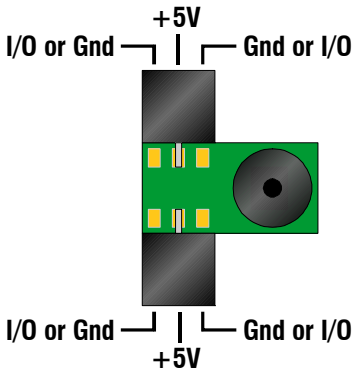
SoundPAL module Sonic Resonator

What You Need to Provide

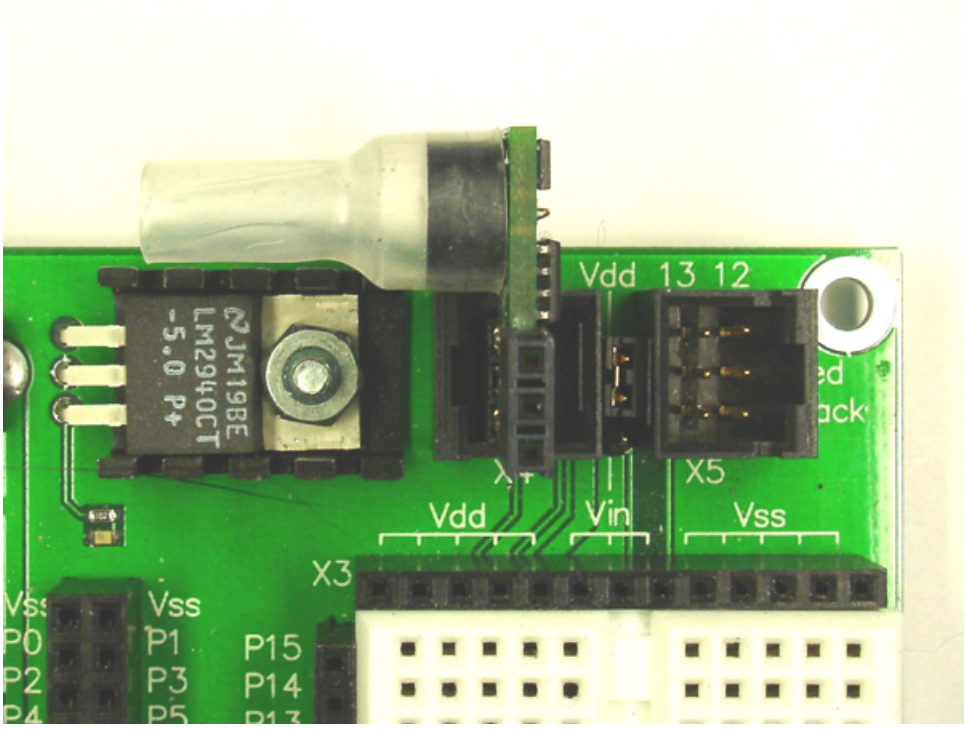
- BASIC Stamp 2 or better and a carrier board, such as Parallax's Board of Education (BOE).

Installation

Installation of the SoundPAL is a simple. If you're using a system that incorporates three-pin servo headers (such as the BOE), just plug the SoundPAL into one of them. It has two three-conductor sockets to choose from for this purpose, and you can insert either one in either direction, yielding four possible orientations. The SoundPAL will adapt to whichever way you plug it in. Here's a diagram showing the SoundPAL's pinout:



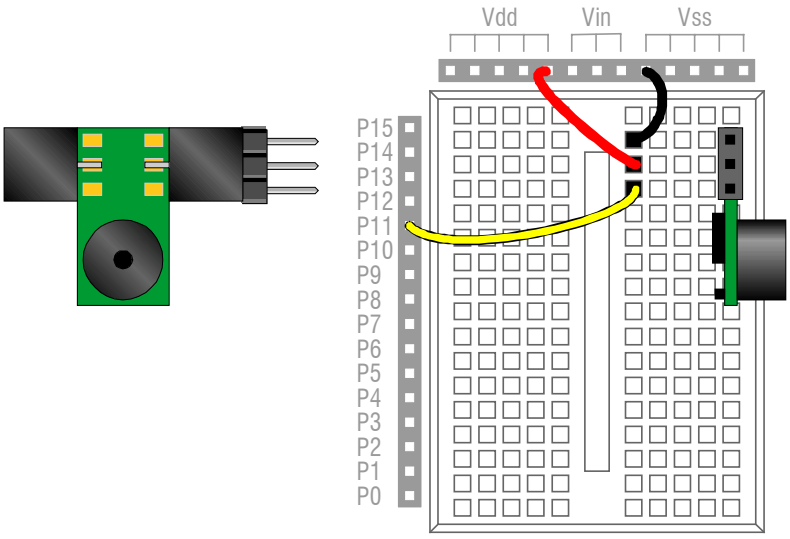
The photo below shows the SoundPAL plugged into a servo connector on the BOE Bot.



You will notice from the photo that the "sonic resonator" has been installed on the speaker. This is provided both to increase the volume level of the SoundPAL and to give it a richer sound.

Important Note: The SoundPAL is designed to work from 5 volts only! Do not jumper your servo headers to Vin or use any supply higher than 5 volts to operate the SoundPAL. Operation at lower voltages down to about 3.3V is okay but will result in diminished output volume.

By installing a 3-pin header (included with the BOE-Bot Full Kit, p/n 28132) in one of the SoundPAL's connectors, you can also use it with a wireless breadboard, as the following diagram illustrates:



In lieu of the three-pin header, you can also use three jumper wires or Parallax's 14-inch LCD Extension Cable and 3-Pin Header (p/n 805-00012).

Hardware Interface and Initialization

Interface to the SoundPAL takes place through either one of its **I/O** pins. When the SoundPAL powers up, both pins are configured as normally-high inputs, pulled up to a nominal +5V through internal 20K to 50K resistances. The BASIC Stamp can thus tell when the SoundPAL has powered up by monitoring the ungrounded **I/O** pin for a high value. This is an important feature when the SoundPAL is plugged into one of the BOE's servo headers, since the BOE's three-way power switch powers up the servos *after* the BASIC Stamp's program starts to run.

Communication with the SoundPAL takes place using the BASIC Stamp's **SEROUT** and **SERIN** commands at any baud rate between 9600 and 19200, positive logic, 8-bit, no-parity. The SoundPAL's **I/O** pin should always be configured as an input by the BASIC Stamp, except when it's being pulled low. It should never be driven high. Therefore, when configuring the baud rate, be sure to set the high bit of the Baudmode to a "one" (i.e. add 32768, or $8000 to the normal Baudmode value). This will configure the pin as an open collector output.

The SoundPAL runs autonomously on power obtained from the servo headers or from another 5V supply. Therefore, it does not reset when the BASIC Stamp resets, and may continue to produce sound during and after reset if it was producing sound before. But it *can* be reset, nonetheless, by outputting a low pulse of 2ms or more to its **I/O** pin. This can be a little tricky with the BASIC Stamp, since the Stamp's **PULSOUT** command always leaves the pin in an output state, which we don't want to do. But there's a simple workaround. Just send a zero byte with **SEROUT** to the SoundPAL at a low baud rate, using the same open-collector protocol used for normal communication.

When the SoundPAL resets, all output ceases immediately, and it returns to its initial state. Resetting is the only way to halt infinite sound sequences, such as the siren effect included in the SoundPAL's internal repertoire.

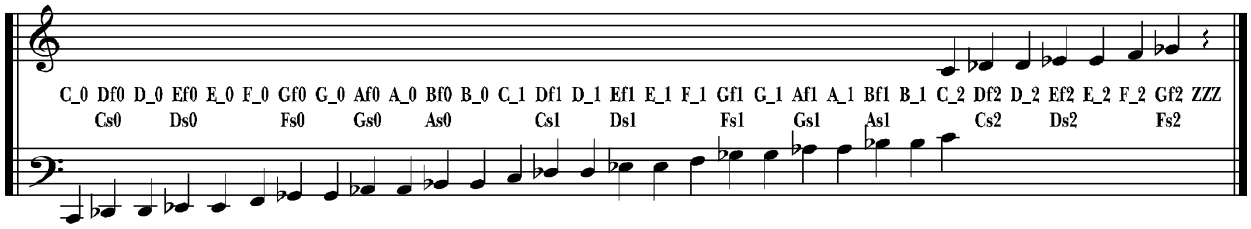
The following code snippet illustrates baud rate settings both for communication and for resetting the SoundPAL. It also includes a reset subroutine.

```
iopin     PIN  15           'Pin number used for communicaiton with SoundPAL.
baud      CON  $8000 + 84   'Communication baudrate is 9600 for BS2, BS2e, and BS2pe.
reset     CON  $8000 + 813  'Reset baudrate is 1200 for BS2, BS2e, and BS2pe.
                            'See BASIC Stamp Manual for settings with other Stamps.

'----Reset the SoundPAL--------------------

DoReset:
  DO:LOOP UNTIL iopin       'Make sure SoundPAL is powered up.
  SEROUT iopin, reset, [0]  'Output 9 low bits at 1200 baud (i.e. a 7.5ms pulse).
  RETURN
```

Playing Individual Notes

Pitch and Duration

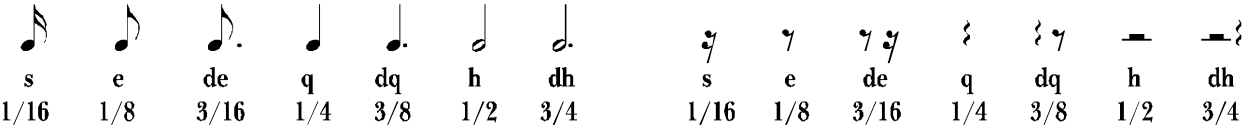
Sounds are played by the SoundPAL as musical notes, each having an associated pitch and duration. The pitches available to the programmer are those of the 12-note chromatic scale, and spanning 2½ octaves, plus four programmable octave offsets, yielding a total range of 6½ octaves. In addition, there is one pitch corresponding to a "rest", which makes it possible to program silent intervals as well. The illustration below shows the notes available in octave zero (**oct0)**.



The note names shown between the staffs above are defined later in a sequence of **CON** statements in a PBASIC code template. This template is also available for download from the SoundPAL product page on the Parallax website.

Three additional octave ranges (**oct1** – **oct3**) are available, each one an octave higher than its predecessor. Without specifiying an octave range, the SoundPAL defaults to **oct2**, in which **C_0** is middle C, and **A_0** is A440.

Each note or rest may have one of eight durations. These durations, along with their musical equivalents and PBASIC names from the SoundPAL template, are shown below:



A single note in PBASIC consists of a pitch and duration and occupies one byte. If the note is a quarter note, only the pitch need be given, since quarter notes are the default. To change to a different duration, just add the duration name to the note name. For example, middle C in **oct0** played as a dotted eighth note would be **de+C_2**, and an eighth rest would be **e+ZZZ**. For consistency, always put the duration ahead of the pitch.

The actual note duration depends on the tempo. Four tempos are available (**tmp0** - **tmp3**). Without specifying a tempo, the SoundPAL defaults to **tmp1**, in which there are 266 quarter note beats per minute. Each successive tempo is double the speed of its predecessor, so the range for quarter note

beats per minute is from 133 to 1064. That makes the shortest sixteenth note 14ms long and the longest dotted half note 677ms long. Although the fastest tempo is *too* fast for most music, it comes in handy for certain non-musical sound effects.

Now we know how to express individual notes, but how do we make the SoundPAL play them? This is a two-step process. First we have to get the notes into the SoundPAL. This is done by loading them into a 48-byte RAM buffer. Next, we have to tell the SoundPAL to execute (i.e. play) what's in the buffer. Both tasks are performed by issuing commands to the SoundPAL. The load command is an equal sign (**=**). It tells the SoundPAL to put everything that follows it, up to and including the first zero byte, into the RAM buffer. The execute command is an exclamation point (**!**). Every time the SoundPAL sees one of these, it will execute the sounds in its buffer, stopping when it encounters the zero. For example, to play a simple C major scale ending on a half-note, we would issue the following:

```
SEROUT iopin, baud, ["=", C_1, D_1, E_1, F_1, G_1, A_1, B_1, h+C_2, 0, "!"]
```

To specify an octave or tempo other than the defaults, just prepend an octave or tempo command to the sound sequence you want played that way. In the following example, the arpeggio C, E, G, C is played in the highest octave at the lowest speed:

```
SEROUT iopin, baud, ["=", oct3, tmp0, h+C_0, h+E_0, h+G_0, h+C_1, 0, "!"]
```

Style

In addition to pitch and duration, notes can be played in one of four different styles. These are named **marcato**, **legato**, **staccato**, and **glissando**. (Musical nomenclature is, by custom, in Italian.) **Marcato** is the default style and simply means that the notes played are demarcated by brief silences between them. That way, two notes in sequence having the same pitch will be heard separately. By contrast, notes played **legato** follow each other without intervening gaps. Two sequential notes having the same pitch and played **legato** will sound like one long note. At the other extreme, notes played **staccato** all have the same, very short length, with the remaining time (defined by the note duration) being filled by silence. And finally, notes played **glissando** are not only played legato, but their pitches also slide into one another, as though they were being played by a trombone or slide whistle. **Glissando** style is used mostly for sound effects, such as the siren and wolf whistle included in the SoundPAL's canned sequences.

To choose a playing style, all you have to do is prepend a style command to the sequence you want to play in that style. For example, to play "Mary Had a Little Lamb" in **staccato**, you'd do the following:

```
SEROUT iopin, baud, ["=", staccato, E_0, D_0, C_0, D_0, E_0, E_0, E_0, 0, "!"]
```

When **glissando** is chosen as the playing style, you can choose from one of 15 glide rates, ranging from **gl1** (fastest) to **gl15** (slowest). Selecting a glide rate by including the glide rate constant in a note sequence automatically selects **glissando** as the playing style, so you don't have to do that separately. The following example shows how the SoundPAL defines a "wolf whistle":

```
SEROUT iopin, baud, ["=",legato,S+G_0,gl2,h+G_1,ZZZ,legato,S+G_0,gl4,C_1,h+C_0,0, "!"]
```

On thing to note from this example is the use of the **legato** style. This is used ahead of the notes that you don't want to glide into. In other words, these notes will begin at their defined pitch but, because of the subsequent **gl2** and **gl4** commands, glide into the notes that follow.

Sometimes you want to play most of a tune **marcato** or **staccato**, but occasionally there will be two or three notes that you want to play **legato**. In musical parlance, this is known as a **slur** when the two notes have different pitches, or a **tie** when they have the same pitch. To this end, the SoundPAL has a special style command called **slur** which is just a **legato** applied to the next note only, causing it to

extend to the note following without any gaps. It is used most frequently to extend the length of a single note beyond the longest duration, which is the dotted half. In the following example, the first three notes of Taps are played with the last being held an extra long time by means of the **slur** command:

```
SEROUT iopin, baud, ["=", tmp0, dq+G_0, e+G_0, slur, dh+C_1, dh+C_1, 0, "!"]
```

Repetition

Many musical sequences and sound effects involve repeated phrases. The SoundPAL provides a **repeat** command to accommodate the need for repetition. Each **repeat** must be followed by a byte indicating how many times (1 – 254) to repeat the phrase that follows. You can also repeat a phrase *ad infinitum* by specifying a repetition of 255. The end of each repeated phrase is indicated by an **again** command. This tells the SoundPAL to go back to the **repeat** until the requisite number of repetitions has been met. A **repeat** … **again** construct with a finite number of repetitions is similar to a **FOR** … **NEXT** loop in PBASIC. With an infinite number of repetitions, it's more similar to a **DO** … **LOOP** construct. Like their counterparts in PBASIC, **repeat** … **again** sequences can be nested. And also like these constructs in PBASIC, it is necessary that each **repeat** be matched with exactly one **again**. Otherwise, the SoundPAL will behave very erratically.

The following example plays some phrases from "Frere Jacques" using the **repeat** … **again** construct:

```
SEROUT iopin, baud, ["=", repeat, 2, C 0, D 0, E 0, C 0, again]
SEROUT iopin, baud, [repeat, 2, E_0, F_0, h+G_0, again, 0, "!"]
```

This example also illustrates the fact that that you don't need to include an entire sequence in one **SEROUT** statement. It can be broken up into pieces, with the zero terminator appearing at the end of the last piece. This is true, so long as the total number of bytes doesn't exceed the RAM buffer size.

One thing to note here is that the size of the RAM buffer, although stated to be 48 bytes, is elastic, depending on the complexity of the sequence it contains. The reason for this is that, after the RAM buffer, there's a 16-byte stack area. The stack holds temporary data that the SoundPAL needs to keep track of what it's doing. It starts at the end of the stack area and builds down toward the RAM buffer. **Repeat** … **again** sequences make use of the stack, as do the SoundPAL's internal subroutines. Each time a **repeat** is encountered, two bytes get "pushed" onto the stack, nudging it two bytes closer to the end of the RAM buffer. For complex sequences with nested **repeat**s, the stack may actually encroach upon the RAM buffer, effectively shortening its useable size. If it overwrites part of a sequence that's stored there, erratic playback will occur. There is no error checking done by the SoundPAL to prevent this from occurring, so it is up to the programmer to balance program length and nesting complexity to keep this from happening.

Playing Pre-programmed Sequences

In additional to individual notes, the SoundPAL can play complete passages stored in its own ROM or EEPROM memory. The SoundPAL comes pre-programmed with 384 bytes of canned sequences in its ROM memory. Its 64-byte EEPROM is available for you to store your own sequences. Playing a pre-programmed sequence is equivalent to calling a subroutine in PBASIC using the **GOSUB** statement. When the sequence has finished playing, it returns to the calling sequence, which then resumes from the calling point. To play a sequence, you have to know its starting address in ROM or EEPROM. Starting addresses for the SoundPAL's canned ROM sequences are provided in the SoundPAL PBASIC template. Each address has a name, corresponding to the particular tune or sound effect that it points to.

EEPROM addresses range from 1 to 63 ($01 to $3F in hex). ROM addresses range from 64 to 255 ($40 to $FF in hex). The ROM address range may appear to include only 192 bytes. But for ROM memory, these are *word* addresses, and each ROM sequence begins on a word (even-byte) boundary. This way a total of 448 bytes (ROM + EEPROM) can be addressed with a one-byte address value.

To play a sequence in ROM or EEPROM, include the **play** command (defined in the template) in the calling sequence followed by the sequence's address. For example, to play the Reveille bugle call, you would use the following PBASIC code:

```
SEROUT iopin, baud, ["=", play, reveille, 0, "!"]
```

The **play** command can be inserted anywhere a note can be used, and multiple play commands can be included in any sequence. A sequence that is **play**ed can even play other sequences, just as subroutines in PBASIC can call other subroutines. As with **repeat**s, you have to be careful about such "nesting" of sequence calls, though. Like **repeat**s, each nested **play** pushes two bytes onto the SoundPAL's stack. So, again, it's important to balance sequence length and nesting complexity to keep the stack from encroaching on a sequence in the RAM buffer.

Global and Local Settings

When a sequence is **play**ed, the octave, tempo, and style settings extant before **play** is encountered are saved and then restored when the **play**ed sequence returns. Therefore, it is not possible for a sequence to influence the octave, tempo, or style of any sequence that calls it. It can only affect its own settings and those of the sequences it **play**s, assuming those sequences do not change those settings for themselves. The tempo, octave, and style settings are thereby considered "local" settings, in that they do not influence anything outside their "scope". The glissando rate, on the other hand, is a "global" setting. It is not saved upon **play**ing a sequence, nor restored when the **play**ed sequence returns. Any change to the glissando rate will persist until it is set to another value.

Local settings are local within an entire sequence. If a setting gets changed inside a **repeat** block, it is not automatically restored when the block encounters an **again** and gets repeated. Therefore, any octave, tempo, or style setting that gets changed inside a **repeat** block should be initialized inside the block at its beginning.

Serial Handshaking

When the SoundPAL is playing a sound sequence, it is not listening to commands coming through its **I/O** pin (except for long reset pulses). This makes it impossible to start a new sequence while one is still playing. However, it *is* possible to poll the SoundPAL so you can tell when a sequence has finished. This is done using the enquiry command (**?**). When the SoundPAL is playing a sequence, it won't see this command at all. But as soon as it has finished, it will respond to an enquiry with a byte equal to 255 ($FF hex), indicating that it's ready for more commands. Here's a subroutine that you can call that will wait for the SoundPAL to finish playing a sequence:

```
'----Wait for the SoundPAL to finish playing----

WaitDone:
  SEROUT iopin, baud, ["?"]                   'Send an enquiry.
  SERIN iopin, baud, 2, Waitdone, [WAIT($FF)] 'Wait 2msec for reply; else try again.
  RETURN                                      'Reply received: return.
```

In the following example, random 32-note sequences are sent to the SoundPAL and played. The PBASIC program waits for each sequence to finish before sending the next one.

```
DO
  SEROUT iopin, baud, ["=",oct3,tmp3]
  FOR B0 = 1 TO 32
    RANDOM W2
    SEROUT iopin, baud, [W2 // 31 + $81]
  NEXT
  SEROUT iopin, baud, [0, "!"]
  GOSUB WaitDone
LOOP
```

Saving Sequences to EEPROM

Once a sequence is loaded into the RAM buffer, it can either be played or saved to EEPROM. The SoundPAL's EEPROM has room for 63 bytes of sequence data. You can store one long sequence there or multiple short sequences. Once saved in EEPROM, a sequence will remain there – even with the power turned off – until it is overwritten with another sequence. When saving a sequence, you specify the address in EEPROM where it's to begin. The entire sequence is saved there, including the zero at the end. That way, when a saved sequence is played, the SoundPAL knows when it's finished so it can return to the calling sequence.

To save a sequence to EEPROM, use the save (**#**) command, followed by a one-byte address. At this point, the SoundPAL will begin saving the data. While this is happening, the SoundPAL will be deaf to any subsequent commands until it's finished, just as it is when it's playing a sequence. You can call **WaitDone**, though, to tell when its work is done.

In the following example, the first few bars of "Swanee River", followed by a call to the canned "Intro" sequence, are saved at location 1. (Fans of old Jackie Gleason "Honeymooners" episodes will recognize this from the one in which he's a contestant on "Name That Tune". He practices with his neighbor, Norton, at the piano, who prefaces each selection with this sequence. Of course, when he gets to the studio, the song they play for him to recognize is "Swanee River", and he is clueless.) Here's how it's saved to EEPROM:

```
GOSUB DoReset
SEROUT iopin, baud, ["=", tmp0, h+E_0, e+D_0, e+C_0, e+E_0, e+D_0] 'Load to RAM.
SEROUT iopin, baud, [C_0, C_1, e+A_0, C_1, tmp1, play, intro, 0]
WEROUT iopin, baud, ["#", 1]                                      'Save to EEPROM.
GOSUB WaitDone
END
```

To play it back, the following sequence is used:

```
SEROUT iopin, baud, ["=", play, 1, 0, "!"] 'Play the saved sequence.
```

Autoplay Mode

When the SoundPAL powers up or resets, the RAM buffer is preprogrammed with a command to play the sequence at EEPROM address 1. Once a sequence has been programmed there, all you have to do to play it after powerup or reset is to issue a "**!**". To try this, save the sequence, as in the example above, then do the following:

```
GOSUB DoReset
SEROUT iopin, baud, ["!"]
```

But it gets better. If, immediately after reset, the SoundPAL senses that its **I/O** pin is low, it will automatically begin execution of the sequence in the RAM buffer, which immediately plays the sequence at EEPROM address 1. This way, you can store a sequence in EEPROM, then connect the SoundPAL so that *both* **I/O** pins are grounded. Then, whenever the SoundPAL powers up, it will play your programmed sequence. That way, you don't even need a BASIC Stamp connected to it to play a canned

sequence. How about a package with a SoundPAL, a battery, and a limit switch that plays "Happy Birthday" when the package is opened?

And there's still more. If you hold the SoundPAL's **I/O** pin low, then pulse it high for a few microseconds, it will play your preprogrammed tune. This will happen every time you send it a short, high pulse, so long as the low time exceeds 2mS each time. Here's a short code sequence to illustrate:

```
LOW iopin
DO
  PAUSE 7000
  PULSOUT iopin, 2
LOOP
```

This will play your saved sequence once every seven seconds. Notice that we've violated the rule not to drive the I/O pin high. In this case it's okay, since the high period is so brief; and the SoundPAL won't try to drive it low during this time anyway, because it didn't receive an enquiry command.

SoundPAL Code Template

Here's the complete code template for the SoundPAL, including all constant definitions, as well as the **DoReset** and **WaitDone** subroutines. It looks quite lengthy, but it takes up only 45 bytes of program space in a BS2.

```
' =================================================================================
'
'   File..... SoundPAL_Template.bs2
'   Purpose... Constants and Standard Subroutines for SoundPAL
'   Author.... Parallax, Inc.
'   E-mail.... support@parallax.com
'   Started... 2007.05.01
'   Updated... 2007.10.29
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =================================================================================

' -----[ Program Description ]---------------------------------------------

' This is a blank template defining constants and subroutines that
' provide an interface to the Parallax SoundPAL (p/n 28825) Miniature
' Sound Player.

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

iopin     PIN  15           'Pin number used for communicaiton with SoundPAL.
                            'Change as needed.

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

baud      CON  $8000 + 84   'Communication baudrate is 9600 for BS2, BS2e, and BS2pe.
reset     CON  $8000 + 813  'Reset baudrate is 1200 for BS2, BS2e, and BS2pe.
                            'See BASIC Stamp manual for settings with other Stamps.
'Commands

play      CON  $01       'Play the segment at the following address ($01-$FF).
repeat    CON  $02       'Begin a repeat block.
                         '  Next arg is repeat count (1-254; 255 = endlessly).
again     CON  $03       'End the repeat block.

'Playing styles

marcato   CON  $04       'Normal, separated notes.
staccato  CON  $05       'Very short notes.
legato    CON  $06       'Connected notes.
glissando CON  $07       'Connected, sliding notes.

'Tempos
```

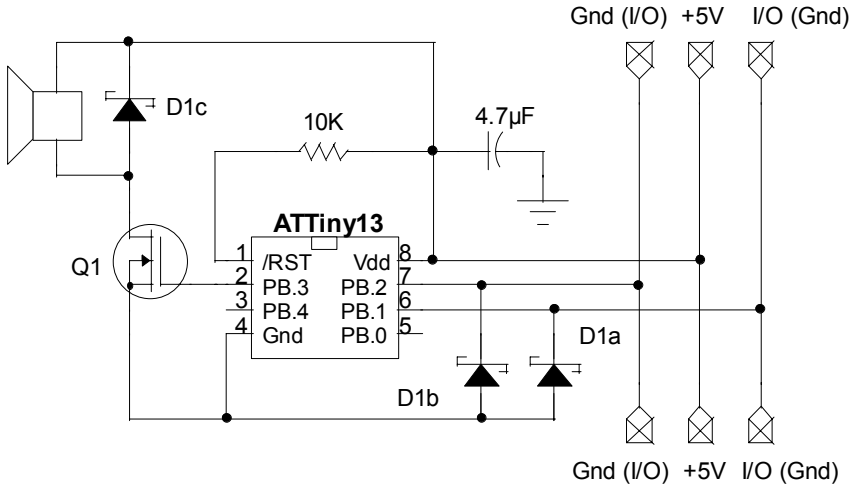
```
tmp0      CON  $08       'Quarter note =  133 beats/min.
tmp1      CON  $09       'Quarter note =  266 beats/min. (default)
tmp2      CON  $0A       'Quarter note =  532 beats/min.
tmp3      CON  $0B       'Quarter note = 1064 beats/min.

'Octaves

oct0      CON  $0C       'A_0 = 110Hz
oct1      CON  $0D       'A_0 = 220Hz
oct2      CON  $0E       'A_0 = 440Hz (default)
oct3      CON  $0F       'A_0 = 880Hz

'Slur or tie, connecting two subsequent notes only.

slur      CON  $10

'Glissando rates: gl1 is fastest; gl15 is slowest.

gl1       CON  $11
gl2       CON  $12
gl3       CON  $13
gl4       CON  $14
gl5       CON  $15
gl6       CON  $16
gl7       CON  $17
gl8       CON  $18
gl9       CON  $19
gl10      CON  $1A
gl11      CON  $1B
gl12      CON  $1C
gl13      CON  $1D
gl14      CON  $1E
gl15      CON  $1F

'Notes. When unmodified by addition of duration, these are all quarter notes.

ZZZ       CON  $80       'Rest
C_0       CON  $81       'Low C natural.    (Middle C in oct2.)
Cs0       CON  $82       'Low C sharp.
Df0       CON  $82       'Low D flat.
D_0       CON  $83       'Low D natural.
Ds0       CON  $84       'Low D sharp.
Ef0       CON  $84       'Low E flat.
E_0       CON  $85       'Low E natural.
F_0       CON  $86       'Low F natural.
Fs0       CON  $87       'Low F sharp.
Gf0       CON  $87       'Low G flat.
G_0       CON  $88       'Low G natural.
Gs0       CON  $89       'Low G sharp.
Af0       CON  $89       'Low A flat.
A_0       CON  $8A       'Low A natural.
As0       CON  $8B       'Low A sharp.
Bf0       CON  $8B       'Low B flat.
B_0       CON  $8C       'Low B natural.
C_1       CON  $8D       'Medium C natural. (Middle C in oct1.)
Cs1       CON  $8E       'Medium C sharp.
Df1       CON  $8E       'Medium D flat.
D_1       CON  $8F       'Medium D natural.
Ds1       CON  $90       'Medium D sharp.
Ef1       CON  $90       'Medium E flat.
E_1       CON  $91       'Medium E natural.
F_1       CON  $92       'Medium F natural.
Fs1       CON  $93       'Medium F sharp.
Gf1       CON  $93       'Medium G flat.
G_1       CON  $94       'Medium G natural.
Gs1       CON  $95       'Medium G sharp.
Af1       CON  $95       'Medium A flat.
A_1       CON  $96       'Medium A natural.
As1       CON  $97       'Medium A sharp.
Bf1       CON  $97       'Medium B flat.
B_1       CON  $98       'Medium B natural.
C_2       CON  $99       'High C natural.   (Middle C in oct0.)
Cs2       CON  $9A       'High C sharp.
Df2       CON  $9A       'High D flat.
D_2       CON  $9B       'High D natural.
```

```
Ds2       CON  $9C       'High D sharp.
Ef2       CON  $9C       'High E flat.
E_2       CON  $9D       'High E natural.
F_2       CON  $9E       'High F natural.
Fs2       CON  $9F       'High F sharp.
Gf2       CON  $9F       'High G flat.

'Duration modifiers. Add value to note to change duration.

s         CON  $20-$80   'Sixteenth note.
e         CON  $40-$80   'Eighth note.
de        CON  $60-$80   'Dotted eighth note.
q         CON  $80-$80   'Quarter note.
dq        CON  $C0-$80   'Dotted quarter note.
h         CON  $A0-$80   'Half note.
dh        CON  $E0-$80   'Dotted half note.

'Canned sound sequences, accessed via the "play" command.

charge    CON  $40       'Charge!
taps      CON  $44       'Taps
reveille  CON  $5D       'Reveille
firstpost CON  $7D       'First Post (horse race bugle call)
intro     CON  $8D       'Doo-doot doo doot doot DOOT
nyah      CON  $93       'Nyah nyah nyah nyah NYAH nyah!
dead      CON  $97       'Funeral dirge
batthymn  CON  $9D       'Battle Hymn of the Republic
dixie     CON  $A5       'Dixie
cucaracha CON  $AC       'La Cucaracha
popweasel CON  $AF       'Pop! Goes the Weasel
marsell   CON  $B3       'Marsellaise
rulebrit  CON  $B9       'Rule Brittania
matilda   CON  $C0       'Walzing Matilda
kradoucha CON  $C6       'Kradoutcha ("There's a place in France...")
wedding   CON  $CD       'Wedding March
ode2joy   CON  $D2       'Ode to Joy
dudu      CON  $DA       'Du, Du Liegst Mir im Herzen
notme     CON  $E1       'Rude sound
uhoh      CON  $E5       'Uh oh!
siren     CON  $E8       'American style siren. Infinite loop: reset to exit.
phone     CON  $EE       'Rings once.
whistle   CON  $F3       'Wolf whistle.
cricket   CON  $FA       'Play using oct3 for cricket; oct0 for frog.

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

PAUSE 10
GOSUB DoReset

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

'             ********** INSERT YOUR PROGRAM HERE. **********

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

'------[ WaitDone ]----------------------------------------------------------

'Wait for the SoundPAL to stop playing.

WaitDone:
  SEROUT iopin, baud, ["?"]                  'Send an enquiry.
  SERIN iopin, baud, 2, Waitdone,[WAIT($FF)] 'Wait 2msec for reply or try again.
  RETURN                                     'Reply received: return.

'------[ DoReset ]----------------------------------------------------------

'Reset the SoundPAL.

DoReset:
  DO UNTIL iopin : LOOP     'Make sure SoundPAL is powered up.
  SEROUT iopin, reset, [0]  'Output 9 low bits at 1200 baud (i.e. a 7.5ms pulse).
  RETURN
```

Command Summary

Direct Commands to SoundPAL			
Command	**Value**	**Description**	**Arguments**
"="	$3D	Queue following bytes (up through zero byte) in RAM.	Sequence data.
"!"	$21	Play sequence in RAM buffer.	
"#"	$23	Save sequence in RAM buffer to EEPROM.	EEPROM address.
"?"	$3F	Enquire whether SoundPAL is ready for new commands. Response from SoundPAL is $FF when ready.	
Commands Within Tune Sequence			
Command	**Value**	**Description**	**Arguments**
0	$00	End-of-sequence delimiter: quit if in RAM; return to caller if in ROM or EEPROM; quit queuing, if queuing.	
play	$01	Play the EEPROM or ROM sequence.	Address of sequence.
repeat	$02	Repeat the following commands up until an **again**.	Number of times to repeat (1 – 254), or 255 for infinite.
again	$03	End of **repeat** sequence.	
marcato	$04	Play the following notes with slight separations.	
staccato	$05	Play the following notes with very short durations.	
legato	$06	Play the following notes in a connected fashion.	
glissando	$07	Play the following notes by gliding their pitches together.	
tmp0 - tmp3	$08 - $0B	Set the tempo for the following notes.	
oct0 - oct3	$0C - $0F	Set the base octave for the following notes.	
slur	$10	Play next note **legato**, extending duration to following note.	
gl1 – gl15	$11 - $1F	Set **glissando** rate (1: fastest, to 15: slowest) and set **glissando** style.	
s+C_0 – dh+Gf2	$20 - $FF	Notes.	

Schematic



Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Parallax:
28825