Column #67, November 2000 by Jon Williams:

Sound Ideas with the BASIC Stamp II

Look, Mom, No Chips

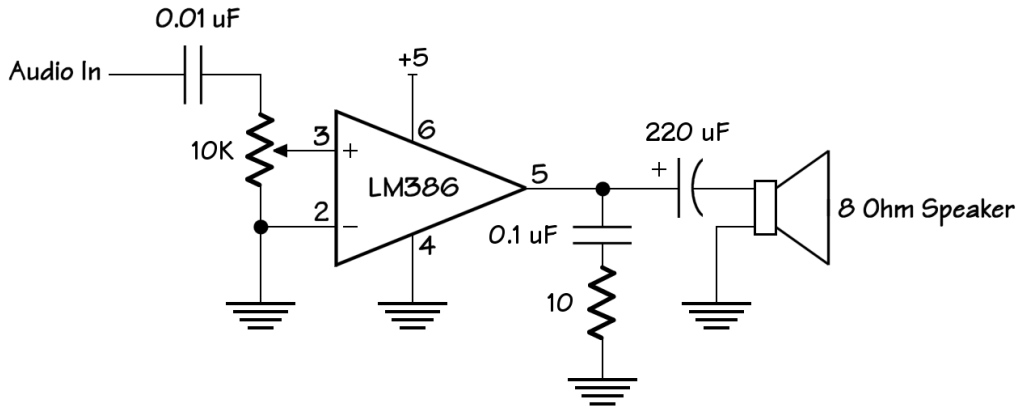
At a recent DPRG gathering, my friend and fellow Stamp enthusiast, Robert Jordan, walked up and handed me a small speaker, the kind that might get attached to a PC. He suggested I flip the switch and turn it on. When I did, the sound of a dial tone poured out, just as if I'd picked up my phone. The realism of the dial tone made me grin. Then it "dialed." Then, a busy signal! It just kept going. My grin turned to a full smile.

"How'd you do it?" I asked. Bob smiled and replied, "With a BS2, of course."

In the past we've talked about adding chips to help the Stamp make sounds. What Bob's neat little project proved to me was that with a little bit of code and imagination, the Stamp's FREQOUT command is capable of some pretty neat things. The best part is that FREQOUT doesn't require any external (sound generating) components.

FREQOUT is used by the Stamp (II and BS2SX) to generate tones. It's very interesting in that it can generate a single tone, or two simultaneously. By mixing tones and code we can create some neat sounds and sound effects. Incidently, DTMFOUT is a specialized version of FREQOUT, designed to generate standard telephone "touch" tones.

Figure 67.1: Driving speaker through capacitor to generate "touch" tones



The only way to appreciate this project is to run it. Note that FREQOUT can drive a high impedance speaker through a capacitor, but you'll get much better sound (and volume control) by using a small amplifier. If you don' have one handy, you can build the circuit in Figure 67.1 for a few dollars in parts.

Sounding Off

Program Listing 67.1 is the code for Bob's (with a little help from Jon) Stamp-based sound effects generator. Load it up and run it. Pretty neat, huh? Okay, let's take a look at the code to see how all the sounds were created.

Since the declarations section contains no magic, jump right down to the code at Dial_Tone, the first effect. The telephone company's dial tone is actually the combination of two frequencies: 350 hertz and 440 hertz. This is perfect for FREQOUT. We only need specify how long to generate the tones. In our case it will be two seconds by using 2000 for the timing parameter in FREQOUT.

Just for fun, I added a "click" sound ahead of the dial tone to give the affect of a receiver being lifted. We'll use the click again later.

After hearing a dial tone, we'll use DTMFOUT to "speed dial" a telephone number that is stored in a DATA statement. This code section starts by initializing the EEPROM pointer to the phone number that we want to dial. One-by-one, we will read a digit, stopping when we read a zero from memory. You'll note that the phone numbers are

actually stored as ASCII strings. This makes them very easy to read in the listing. To convert an ASCII character to the decimal value required by DTMFOUT, we subtract 48 ("0") from the ASCII value

DTMFOUT generally expects the digits zero through nine, so we check to make sure that the current character is a digit that can be dialed (character >= "0"). If the character is not in the valid "dialing" range (as would be the case for "-"), the DTMFOUT command is skipped and we retrieve the next character from EEPROM. If the character can be dialed, our DTMFOUT line "presses the button" for 200 milliseconds and inserts a 150 millisecond break afterward.

Be careful with your phone near this project. If you hold the microphone element of you phone near the speaker when the DTMFOUT demo is running, the number will be dialed. Don't believe me? Give it a try….

If you do decide to create your own dialer from a Stamp, be aware that telephone company standards require a minimum of 50 ms for the DTMF tone with a minimum inter-digit pause of 45 ms. You'll probably want to use longer DTMF tones, especially if your telephone line is noisy and you're using acoustic coupling (from speaker to phone).

The next sound effect is a telephone busy signal. This effect is created my mixing tones of 480 and 620 hertz. The tones last for 400 milliseconds and a separated by at 620 millisecond break. FREQOUT embedded in a FOR-NEXT loop takes care of creating this effect.

For the sake of continuity through the demonstration, I inserted another dialing demo. It works exactly like the first, except that this one points at a different telephone number. In a dialer application, this code could be converted to a generalized subroutine that takes the EEPROM address of the number to be dialed before being called.

After the second number is dialed, we hear the phone ring. Once again, this is very simple with the FREQOUT command. The ring back tone (which is actually created by the telephone company central office, not the phone you're calling) is a mixture of 440 and 480 hertz tones for two seconds, followed by a four second gap.

Okay, we can't use FREQOUT to simulate someone answering that phone call, so we'll play a little tune instead. Music generation is probably the most popular use for FREQOUT.

The tune is stored in three LOOKUP tables. The first table contains the notes and rests that were defined earlier. Note that sharp notes are designated by the note followed by a small "s." We can't use the "#" sign like on music since this is not a valid character for constants.

The second table contains the octave for the corresponding note in the first table. When creating your own songs, you must be take care that each of the tables have the same number of entries.

The final table contains the duration for each note. Since all notes less than a whole note (N1) are derived from the whole note value, you can change the timing of a song very easily by changing the value of the whole note.

With all of the information about a note collected (tone, octave and duration), the Play1Note subroutine is called to make the sound. This routine calculates the proper frequency of the note for the octave specified, then uses FREQOUT to play it. For musical notes, each octave represents a doubling of the note's frequency. The left shift operator (<<) makes calculating the frequency for the specified easy.

There is an additional subroutine called WarbleNote that I lifted from the Stamp manual. This routine takes the first tone, then creates a second that is of a slightly lower pitch. The closeness in pitch between the two tones creates an interesting warbling effect.

There's no need to describe the details of the rest of the program. Each of the effects is simple and uses FREQOUT as we already have. Just a word of warning: if you're running through an amplifier, be sure to have it turned down low before it hits the Howler demo. It's amazing how the combination of a couple of tones can be incredibly annoying. This is a great routine for a Stamp-based alarm system.

More on Music

The CONstants declarations in the program contain comments for the ideal frequency of each note in the scale for the first octave. I put that information in the code for those who may want to create more accurate music notes than what is possible with Play1Note and Play2Notes.

Remember that the Stamp uses integer mathematics (whole numbers only). Right off the bat we're at a slight disadvantage because we have to round the first octave tones to integers. The rounding problem gets compounded with Play1Note. Every time we

double the frequency of a note (go up an octave), the rounding error gets bigger. It's possible, then, for the higher octave notes to be out of tune.

The problem can be fixed by creating a table of CONstants for all the notes you might want to use. To convert the first octave tones, use the following multipliers, then round to the closest integer value.

2nd: tone x 2
3rd: tone x 4
4th: tone x 8
5th: tone x 16
6th: tone x 32
7th: tone x 64
8th: tone x 128

For example, B in the 8th octave would be 61.735 x 128 = 7902.08, which rounds nicely to 7902.

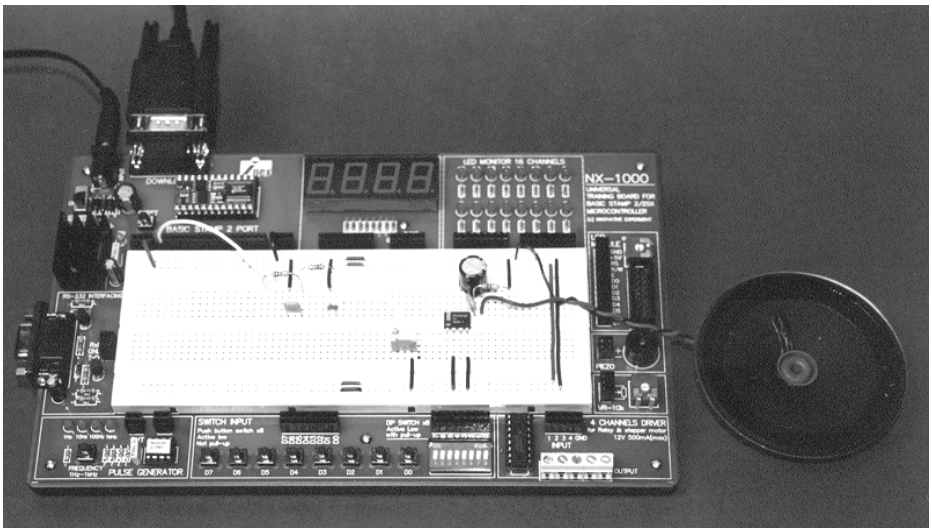
What might you do with Stamp music? How about a custom doorbell that plays your favorite holiday song?

Prototyping Paradise

A recent addition to the Parallax product line and now my favorite Stamp developing tool is the INEX-1000 prototyping and development board. The INEX-1000 holds a BS2 (or BS2-SX), has a 5-volt power supply (good for about an amp) and a full-sized, solderless breadboard that is surrounded by a variety of Stamp-useful components:

- Four 7-segment LEDs (common cathode)
- 16 LEDs with current limiters (active high)
- A 14-pin IDC socket for a parallel LCD
- Piezo speaker
- 10K pot
- Four high-current outputs (12 VDC, through ULN2003)
- An 8-position SIP with pull-ups
- Eight pushbuttons (one side connected to ground, the other floats)
- A pulse generator (1 Hz, 10 Hz, 100 Hz, 1 kHz)
- An RS-232 interface with DB9 connector

Figure 67.2: INEX-1000 laboratory



The INEX-1000 includes a 12-volt "wall wart" supply and a parallel LCD module that's ready to plug into the IDC connect. You add a Stamp, some 22-guage solid wire, any interfacing components you might need and you're ready to develop.

In the photo you can see the INEX-1000 with an LM386 amplifier built in the solderless breadboard.

For those of you "lurking" this column and, perhaps, the Stamp mailing list that haven't bought a Stamp yet because you didn't know what else you might need, Parallax is building a solution for you. Just in time for the holidays, Parallax will release a kit called "StampWorks" that includes the INEX-1000, a BS2, several electronic (i.e., shift registers, LED drivers) and mechanical (servo and stepper motors), tools, wire, documentation with over 35 experiments…in short, the works; everything you need to learn to program the BS2 for a variety of applications.

The kit is targeted for the $300 range – a tremendous value considering all that is included in the kit. Stay tuned to the Parallax web site for the final release date and details.

Until next time, Happy Stamping.

```
' =====================================================================
' Program... SOUNDS.BS2
' Author.... Robert Jordan (modified by Jon Williams)
' Started... 03 SEP 2000
' Updated... 07 SEP 2000
' =====================================================================


' -----[ Program Description ]---------------------------------------
'
' This program demonstrates the versatility of the Stamp 2 FREQOUT
' command.


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


' -----[ I/O Definitions ]-------------------------------------------
'
Spkr    CON     15                  ' speaker port


' -----[ Constants ]-------------------------------------------------
'
R       CON     0                   ' rest
C       CON     33                  ' ideal is 32.703
Cs      CON     35                  ' ideal is 34.648
D       CON     37                  ' ideal is 36.708
Ds      CON     39                  ' ideal is 38.891
E       CON     41                  ' ideal is 41.203
F       CON     44                  ' ideal is 43.654
Fs      CON     46                  ' ideal is 46.249
G       CON     49                  ' ideal is 48.999
Gs      CON     52                  ' ideal is 51.913
A       CON     55                  ' ideal is 55.000
As      CON     58                  ' ideal is 58.270
B       CON     62                  ' ideal is 61.735

N1      CON     500                 ' whole note
N2      CON     N1/2                ' half note
N3      CON     N1/3                ' third note
N4      CON     N1/4                ' quarter note
N8      CON     N1/8                ' eighth note


' -----[ Variables ]------------------------------------------------
'
x       VAR     Word                ' loop counter
note1   VAR     Word                ' first tone for FREQOUT
```

```
note2   VAR     Word                    ' second tone for FREQOUT
dur     VAR     Word                    ' duration for FREQOUT
oct1    VAR     Nib                     ' octave for freq1 (1 - 8)
oct2    VAR     Nib                     ' octave for freq2 (1 - 8)
ee      VAR     Byte                    ' EEPROM pointer
digit   VAR     Byte                    ' DTMF digit
clkDly  VAR     Word                    ' delay betweens "clicks"


' -----[ EEPROM Data ]----------------------------------------------------
'
PN1     DATA    "972-555-1212",0        ' a stored telephone number
PN2     DATA    "916-624-8333",0        ' another number


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


' -----[ Main ]-----------------------------------------------------------
'

Main:
  DEBUG CLS
  DEBUG "Robert Jordan's BS2 Sound Demo",CR
  DEBUG "-------------------------------",CR


Dial_Tone:
  DEBUG "Dial tone",CR
  FREQOUT Spkr,35,35                    ' "click"
  PAUSE 100
  FREQOUT Spkr,2000,350,440             ' combine 350 Hz & 440 Hz


Dial_Phone1:                            ' dial phone from EE
  DEBUG "Dialing number: "
  ee = PN1                              ' initialize ee pointer
GetNum1:
  READ ee,digit                         ' read a digit
  IF digit = 0 THEN Phone_Busy          ' when 0, number is done
  DEBUG digit                           ' display digit
  IF digit < "0" THEN IncEE1            ' don't dial non-digits
  DTMFOUT Spkr,200,150,[digit-48]       ' dial digit (convert from ASCII)
IncEE1:
  ee = ee + 1                           ' update ee pointer
  GOTO GetNum1                          ' get another digit


Phone_Busy:
  PAUSE 1000
  DEBUG CR, " - busy...",CR
```

```
  FOR x = 1 TO 8
    FREQOUT Spkr,400,480,620         ' combine 480 Hz and 620 Hz
    PAUSE 620
  NEXT
  FREQOUT Spkr,35,35                 ' "click"


Dial_Phone2:
  DEBUG "Calling Parallax: "
  ee = PN2
GetNum2:
  READ ee,digit
  IF digit = 0 THEN Phone_Rings
  DEBUG digit
  IF digit < "0" THEN IncEE2
  DTMFOUT Spkr,200,150,[digit-48]
IncEE2:
  ee = ee + 1
  GOTO GetNum2


Phone_Rings:
  PAUSE 1000
  DEBUG CR, " - ringing"
  FOR x = 1 TO 4
    FREQOUT Spkr,2000,440,480        ' combine 440 Hz and 480 Hz
    PAUSE 4000
  NEXT


Camptown_Song:
  DEBUG CR,"Play a camptown song",CR
  FOR x = 0 TO 13
    LOOKUP x,[ G, G, E, G, A, G, E, R, E, D, R, E, D, R],note1
    LOOKUP x,[ 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 1, 4, 4, 1],oct1
    LOOKUP x,[N2,N2,N2,N2,N2,N2,N2,N2,N2,N1,N2,N2,N1,N8],dur
    GOSUB Play1Note
  NEXT


Howler:
  DEBUG "Howler -- watch out!!!",CR
  FOR x = 1 TO 4
    FREQOUT Spkr,1000,1400,2060      ' combine 1400 Hz and 2060 Hz
    FREQOUT Spkr,1000,2450,2600      ' combine 2450 Hz and 2600 Hz
  NEXT


Alt_Dial_Tone:
  DEBUG "Alternate Dial Tone",CR
  FREQOUT Spkr,5000,600,133          ' combine 600 Hz and 133 Hz
```

```
Fast_Busy:
  DEBUG "Fast Busy signal",CR
  FOR x = 1 TO 10
    FREQOUT Spkr,200,480,620        ' combine 480 Hz and 620 Hz
    PAUSE 310
  NEXT


' *************************
' Additional sounds from Jon
' *************************

Roulette_Wheel:
  DEBUG "Roulette Wheel",CR
  note1 = 35                        ' frequency for "click"
  dur = 5                           ' duration for "click"
  clkDly = 250                      ' starting delay between clicks
  FOR x = 1 TO 8                    ' spin up wheel
    FREQOUT Spkr,dur,note1          ' click
    PAUSE clkDly
    clkDly = clkDly */ $00BF        ' accelerate (speed * 0.75)
  NEXT
  FOR x = 1 TO 10                   ' spin stable
    FREQOUT Spkr,dur,note1
    PAUSE clkDly
  NEXT
  FOR x = 1 TO 20                   ' slow down
    FREQOUT Spkr,dur,note1
    PAUSE clkDly
    clkDly = clkDly */ $010C        ' decelerate (speed * 1.05)
  NEXT
  FOR x = 1 TO 30                   ' slow down and stop
    FREQOUT Spkr,dur,note1
    PAUSE clkDly
    clkDly = clkDly */ $0119        ' decelerate (speed * 1.10)
  NEXT


Computer_Beeps:                     ' looks great with randmom LEDs
  DEBUG "50's Sci-Fi Computer",CR
  FOR x = 1 TO 50                   ' run about 5 seconds
    RANDOM note1                    ' create random note
    note1 = note1 // 2000           ' don't let note go to high
    FREQOUT Spkr,50,note1           ' play it
    PAUSE 100                       ' short pause between notes
  NEXT

  DEBUG CR,"Sound demo complete."
```

```
  END


' -----[ Subroutines ]-------------------------------------------------
'
Play1Note:
  note1 = note1 << (oct1-1)          ' get frequency for note + octave
  FREQOUT Spkr,dur,note1             ' play it
  RETURN


Play2Notes:
  note1 = note1 << (oct1-1)          ' get frequency for note + octave
  note2 = note2 << (oct2-1)          ' get frequency for note + octave
  FREQOUT Spkr,dur,note1,note2       ' play both
  RETURN


WarbleNote:
  note1 = note1 << (oct1-1)          ' get frequency for note + octave
  note2 = note2 - 8 MAX $7FFF        ' create slightly lower frequency
  FREQOUT Spkr,dur,note1,note2       ' play warbled
  RETURN
```