Column #40, June 1998 by Jon Williams:

Talk Is Cheap!

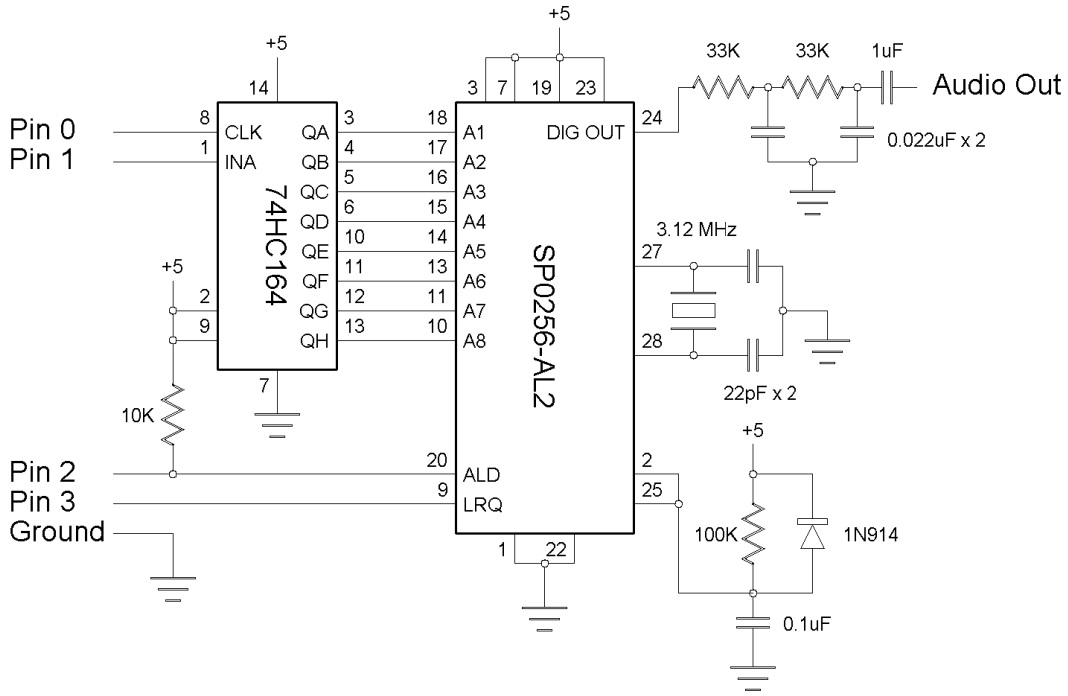
As far as output devices go, LEDs are okay, character LCDs are nice, and graphical LCDs are pretty cool. But a voice? Now you're talking — literally. With a couple of chips, a little time with your soldering iron, and just a bit of code, you can give your BASIC Stamp projects a voice and an unlimited English vocabulary to go along with it.

The SP0256-AL2

The heart of this project is the General Instrument SP0256-AL2 allophone speech processor. I'm sure that somebody will question the logic of doing a project with a chip that went out of production quite some time ago, especially when there are other devices available. I went with the SP0256-AL2 because they're still easy to come by (from B.G. Micro), they're inexpensive, they're easy to interface, and I like the idea of the unlimited (English) vocabulary.

Being a simple guy, I'm going to keep things simple (if you want the gory technical details, you can read them in the GI docs). The SP0256-AL2 is a specialized processor that contains 64 digitally-encoded speech sounds — called allophones — and the software to convert the digital allophone data to an audible sound. By linking allophones, we can create speech.

Figure 40.1: The SP0256-AL2 interfaced with a BASIC Stamp 1



Selecting an allophone is a simple matter of placing its address on the SP0256-AL2 bus and strobing (high to low) the ALD line. If the address buffer is full, the LRQ line will go high, so we should check it before loading another allophone address.

Making The Connections

To make a straightforward connection to the SP0256-AL2, we need eight lines: six for the allophone address, and one each for ALD and LRQ. You could, of course, connect direct, but this chews up a lot of pins and leaves nothing left if you're using a BS1. I cut the pin count down to four by using a 74HC164 shift register to output the allophone address. I selected the 74HC164 because it only needs clock and data lines; there is no strobe like on the 75HC595. What this means, however, is that the outputs ripple with the clock. This presents no problem for our project since the SP0256-AL2 address lines are ignored until ALD is pulsed low.

Figure 40.2: The SP0256-AL2 amplifier circuit

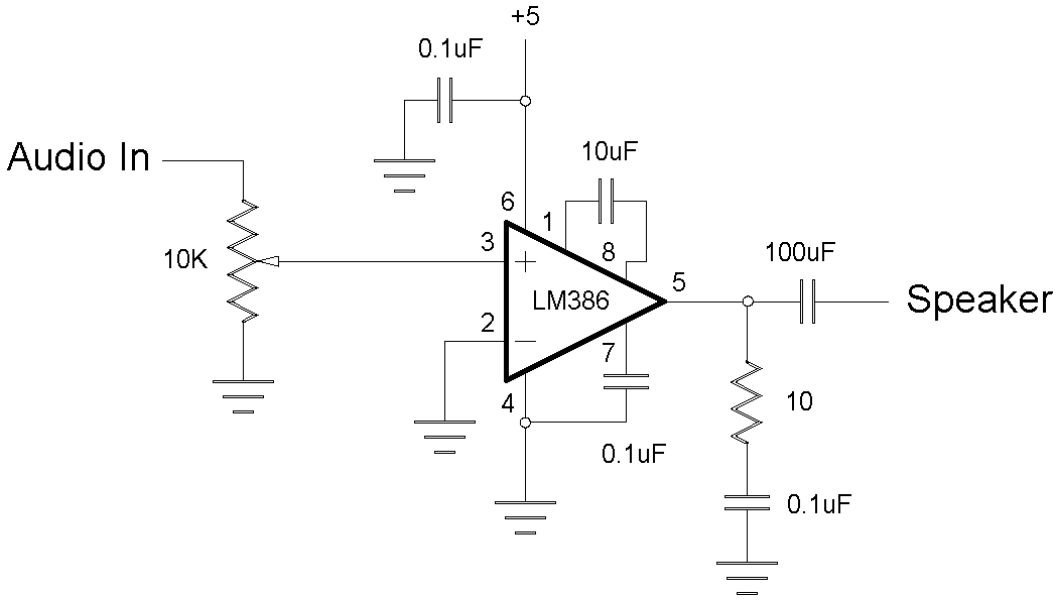


Figure 40.1 shows the schematic for the circuit. You'll notice that I used all of the address lines on the SP0256-AL2. Technically, I could have tied A7 and A8 to ground, but I figured it was just as easy to connect them to the 74HC164. This also makes my circuit compatible with the SP0256, provided I connect an external speech ROM to the appropriate pins on the SP0256.

I also elected to use an external amplifier. If you'd like to have an onboard audio amp, Figure 40.2 shows an appropriate circuit. Just keep your wiring tight around the op-amp to prevent it from oscillating.

Wiring up the circuit will be the most time-consuming task of this project. There are really too many connections to try to put this on a solderless breadboard, so point-to-point wiring (my choice) or wire-wrapping is called for. Of course, if you have the tools and know-how to make your own PC boards, that would be even nicer. Take your time. You certainly don't want to screw things up by rushing through your wiring. You'll be rewarded for your patience.

The Software

BASIC Stamp software really doesn't get any easier than this, folks. After last month (remember the graphing thermometer and all the trouble I had squeezing it into the BS1?), this one's a breeze. Please refer to Program Listing 40.1 for analysis.

My strategy goes like this: I store speech strings in the Stamp's EEPROM. Each string ends with the marker byte $FF. To say a particular string, I load its EEPROM address into the variable called addr and call the subroutine called Speak. Nothing to it.

Speak reads the data stored at the current value of addr. If it is not $FF (the end of the speech marker), we send it to the SP0256-AL2. But first we have to make sure that the SP0256-AL2 address buffer is not full. If it is, we wait for the LRQ line to go low. When the speech processor is ready, we output an allophone address via the 74HC164 (Most of you have seen this code before. If you're new, or haven't connected a BS1 to a shift register, please refer to the section titled "Shifty Business For Beginners.") With the allophone address placed on the SP0256-AL2, we load it by pulsing the ALD line (high to low, then back to high) with PULSOUT. The transition is high to low because we started with a high on the ALD line (pin 2) before we used PULSOUT.

With the current allophone address loaded, we increment the pointer and try it again. When $FF is encountered, the subroutine is terminated and RETURNs to the caller. Of the 64 allophone addresses, 59 are audible speech sounds and five are pauses of various lengths. It is very important to remember that your speech string must end with a pause. Otherwise, the last allophone will continue to play.

As always, I've made heavy use of SYMBOLs to make the code more readable. This includes all of the allophones. There is one allophone — OR — that collides with a PBASIC keyword. The problem is solved with the addition of an underscore. You'll see "_OR" in the allophone table (Program Listing 40.2).

Talk, Talk, Talk

Once you're all wired up, the fun begins: creating speech data. This is not quite as easy as it might seem, because there is no direct correlation between the spelling of a given word and the allophones used to produce it. Things get even trickier when one considers that the same allophone can have a bit of a different sound based on its position in the word. It won't take too long to get the hang of concatenating allophones to create speech. What I found is that I started listening very carefully to the way I say words. Diphthongs (vowel sounds that glide together) are the trickiest part of the encoding.

And don't worry, you don't have to figure everything out from scratch.

Please see TALKER.ZIP from the CD-ROM directory. Along with the source code for this project, you'll find a couple of PDF documents that I located on the Internet. Both are scans of SP0256-AL2 documentation. One is from GI, the other from Radio Shack. The Radio Shack document contains a lot of application information — good stuff that you'll find extremely helpful when you start working with the SP0256-AL2. There's information about the chip and how it works, as well as a nice library of common words, numbers, names of months, and the days of the week.

Applications

If you consider that we probably get most of our information through spoken words, the applications for this technology are limitless. Here are a couple of ideas that come to mind: a talking thermometer (I told you I had a thing for thermometers!), a talking clock, a talking alarm system, and various devices for the blind. Heck, you could even make your own annoying "Your door is ajar ..." alarm for the family automobile. Another obvious application is giving your Stamp-controlled pet robot a voice.

Have fun. If there's enough interest in this and other voice-oriented projects, we'll put one together with one of the newer options. The call is yours. My E-Mail address is in the Sources section.

Shifty Business For Beginners

Since the Stamp does not use a traditional address/data bus, beginners are often troubled with I/O expansion, especially with the BS1. Upon analysis, we'll find that we usually define pins as inputs or outputs, and rarely do we need to change the state of a pin mid-program. Since this is the case, we can expand our Stamp I/O with shift registers. This project uses the 74HC164 serial-in, parallel-out shift register. With the 74HC164, we can turn two Stamp lines into eight. What we've effectively done is expanded the Stamp's outputs from eight to 14 (remember that two are used to control the shift register). If you need more outputs, you can use the 74HC595 because it's cascadable (you can link two or more together in a chain). Be aware that the '595 also requires a strobe line to transfer your data to the outputs. If your circuit requires more than eight (additional) outputs, or you don't want the outputs to ripple (change) with the clock line, the 74HC595 is the way to go.

For the sake of discussion, let's stick with the '164. When the clock line is pulsed high, the state of the input pin (high or low) is transferred to output A. At the same time, what was on output A is transferred to output B, and so on down the line. The content of output H, the last bit, is transferred into bit-oblivion — it's gone forever. By doing eight such transfers, we can output an eight-bit byte with two pins.

Back in August, I made the comment that it is my habit to reserve variables B0 and B1 for bit-level access. Here's where we'll use bit-level access.

Take a look at the FOR-NEXT loop embedded in the Speak subroutine. This loop runs eight times; enough to shift out a byte. The first thing we do is grab Bit7 and output it to the data pin. Bit7 is the MSB (most significant bit) of B0, the variable that holds our shifted data (it is SYMBOLically renamed data). We grab Bit7 because we want this bit to be present on 74HC164 output H when we're done with eight clock pulses.

After Bit7 is output, we want to do the same thing with Bit6, then Bit5, and so on, down to Bit0. We could do this in hard code, but that technique would be wasteful. It's more convenient to move Bit6 to Bit7 and repeat the same set of code. We call this a left shift since the bit values move to their left during this process. Since we're dealing with binary numbers, multiplying the current value of data by two causes a left shift. It's important to keep in mind that the value of data will change during this subroutine. If you need to save it for some reason, make a copy to another variable before calling Speak.

Okay, what about inputs? Not a problem. Go get a 74HC165: a parallel-input, serial output shift register. This chip complements the '164. With the ground we've just covered, you should be able to check out the specs and put together the code, so I will not deny you the opportunity to learn from the experience. I will, however, give you a hint. In order to share the data line with the '164, you'll need to change it from an output to input and back. For an example, take a look at last month's project — it used this technique.

```
' Program Listing 40.1
' Nuts & Volts: Stamp Applications, June 1998

' ----[ Title ]---------------------------------------------------------
'
' File..... TALKER.BAS
' Purpose... BASIC Stamp -> SP0256-AL2 Allophone Speech Processor
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' WWW....... http://members.aol.com/jonwms
' Started... 03 MAY 98
' Updated... 03 MAY 98

' ----[ Program Description ]-------------------------------------------
'
' This program enables the BASIC Stamp to talk by interfacing with the GI
' SP0256-AL2 Allophone Speech Processor.
'
' A 74HC164 parallel output shift register is used to provide allophone
' address to the SP0256-AL2. Four I/O lines are used:
'
' Pin 0 (Out)  Clock to 74HC164 (can be shared)
' Pin 1 (Out)  Data to 74HC164 (can be shared)
' Pin 2 (Out)  ALD - pulsed low causes allophone address to be loaded
' Pin 3 (In)   LRQ - goes high when the SP0256-AL2 address buffer is full

' ----[ Revision History ]----------------------------------------------
'
' 03 MAY 98 : Version 1

' ----[ Constants ]-----------------------------------------------------
'
SYMBOL  Clk     = 0
SYMBOL  Dio     = Pin1
SYMBOL  ALD     = 2                   ' Address Load
SYMBOL  LRQ     = Pin3                ' Load Request


' Copy and Paste "ALLOPHON.BAS" here


' ----[ Variables ]-----------------------------------------------------
'
SYMBOL  data    = B0                  ' allophone data
SYMBOL  shift   = B2                  ' loop counter for shift out
SYMBOL  addr    = B3                  ' EE address of allophone


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

```
        ' Hello,
        EEPROM (HH1,EH,LL,AX,OW,PA4)
        ' I am the
        EEPROM (AA,AY,PA2,AE,MM,PA2,DH1,AX,PA2)
        ' BASIC Stamp
        EEPROM (BB2,EY,SS,IH,KK2,PA3,SS,SS,PA1,TT2,AE,AE,MM,PP,PA1,$FF)


' ----[ Initialization ]----------------------------------------------------
'
Init:   Pins = %00000100              ' begin with ALD high
        Dirs = %00000111


' ----[ Main Code ]---------------------------------------------------------
'
Main:   addr = $00                    ' point to start of speech
        GOSUB Speak                   ' speak!
        PAUSE 2000                    ' wait for two seconds
        GOTO Main                     ' speak again...

        END


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

Speak:  READ addr, data               ' get allophone from EE table
        IF data = $FF THEN Done       ' if $FF, we're done
Busy:   IF LRQ = 1 THEN Busy          ' wait if SP0256-AL2 buffer full
        FOR shift = 1 TO 8            ' shift out the allophone address
          Dio = Bit7                  ' get a bit
          PULSOUT Clk, 10             ' clock it out
          data = data * 2             ' left-shift the byte
        NEXT
        PULSOUT ALD, 10               ' load the allophone
        addr = addr + 1               ' point to next allophone address
        GOTO Speak
Done:   RETURN
```

```
' Program Listing 40.2
' Nuts & Volts: Stamp Applications, June 1998

' Allophone constants for "TALKER.BAS"

SYMBOL  PA1     = $00   '  10 ms pause (before BB, DD, GG and JH)
SYMBOL  PA2     = $01   '  30 ms pause (before BB, DD, GG and JH)
SYMBOL  PA3     = $02   '  50 ms pause (before PP, TT, KK and CH)
                        '              (and between words)
```

```
SYMBOL  PA4     = $03   ' 100 ms pause (between clauses and sentences)
SYMBOL  PA5     = $04   ' 200 ms pause (between clauses and sentences)

                        ' sample word

SYMBOL  OY      = $05   ' bOY
SYMBOL  AY      = $06   ' skY
SYMBOL  EH      = $07   ' End
SYMBOL  KK3     = $08   ' Comb
SYMBOL  PP      = $09   ' Pow
SYMBOL  JH      = $0A   ' doDGe
SYMBOL  NN1     = $0B   ' thiN
SYMBOL  IH      = $0C   ' sIt
SYMBOL  TT2     = $0D   ' To
SYMBOL  RR1     = $0E   ' Rural
SYMBOL  AX      = $0F   ' sUcceed

SYMBOL  MM      = $10   ' Milk
SYMBOL  TT1     = $11   ' parT
SYMBOL  DH1     = $12   ' THey
SYMBOL  IY      = $13   ' sEE
SYMBOL  EY      = $14   ' bEIge
SYMBOL  DD1     = $15   ' coulD
SYMBOL  UW1     = $16   ' tO
SYMBOL  AO      = $17   ' AUght
SYMBOL  AA      = $18   ' hOt
SYMBOL  YY2     = $19   ' Yes
SYMBOL  AE      = $1A   ' hAt
SYMBOL  HH1     = $1B   ' He
SYMBOL  BB1     = $1C   ' Business
SYMBOL  TH      = $1D   ' THin
SYMBOL  UH      = $1E   ' bOOk
SYMBOL  UW2     = $1F   ' fOOd

SYMBOL  AW      = $20   ' OUt
SYMBOL  DD2     = $21   ' Do
SYMBOL  GG3     = $22   ' wiG
SYMBOL  VV      = $23   ' Vest
SYMBOL  GG1     = $24   ' Got
SYMBOL  SH      = $25   ' SHip
SYMBOL  ZH      = $26   ' aZure
SYMBOL  RR2     = $27   ' bRain
SYMBOL  FF      = $28   ' Food
SYMBOL  KK2     = $29   ' sKy
SYMBOL  KK1     = $2A   ' Can't
SYMBOL  ZZ      = $2B   ' Zoo
SYMBOL  NG      = $2C   ' aNchor
SYMBOL  LL      = $2D   ' Lake
SYMBOL  WW      = $2E   ' Wool
SYMBOL  XR      = $2F   ' repaiR
```

```
SYMBOL WH      = $30   ' WHig
SYMBOL YY1     = $31   ' Yes
SYMBOL CH      = $32   ' CHurCH
SYMBOL ER1     = $33   ' fIR
SYMBOL ER2     = $34   ' fIR
SYMBOL OW      = $35   ' bEAU
SYMBOL DH2     = $36   ' THey
SYMBOL SS      = $37   ' veSt
SYMBOL NN2     = $38   ' No
SYMBOL HH2     = $39   ' Hoe
SYMBOL _OR     = $3A   ' stORe        (note leading underscore)
SYMBOL AR      = $3B   ' alARm
SYMBOL YR      = $3C   ' cleaR
SYMBOL GG2     = $3D   ' Guest
SYMBOL EL      = $3E   ' saddLE
SYMBOL BB2     = $3F   ' Business

' end of Allophone table
```