Column #65, September 2000 by Jon Williams:

A Talking Pet Trainer

During my first run with this column I wrote an article on using the long-obsolete but still available SPO256-AL2 allophone speech synthesizer. That one article has generated more e-mail than all the rest of my articles combined. Clearly, Stamp enthusiasts are interested in speech.

Aside from being hard to find, the SPO256-AL2 has a very tinny, robotic quality to the speech. This is actually kind of cool for robotics, but some applications require higher quality speech and may even need non-speech sounds.

Enter the ISD25xxx series of chips. These digital recorders are easy to find and equally easy to use. They differ from the SPO256-AL2 in that they do not come with stored sounds or speech; it is up to you to put those in. Once a message has been recorded, it is stored in non-volatile memory. This means you won’t lose your recording when power is removed from the circuit. This is particularly useful in battery-powered applications.

Polly Want A Stamp?

I got started with the ISD at the request of my friend, Mike. At the time, Mike was a fledgling Stamper that went out and bought a parrot – a big, beautiful, very expensive parrot. With the investment, Mike figured the bird learn to should talk. Using and BS2 and an ISD2560, he built the first version of his parrot trainer. While workable, he felt like the circuit and software could use some refinement so he sent it to me.

This ISD25xxx is a moderately sophisticated component and has several modes of operation. Our application will use the simplest operational mode, M6, also referred to as Push-Button mode. In this mode, the device has the ability to record and playback a single message.

Figure 65.1: ISD hookup to BASIC Stamp with microphone and speaker

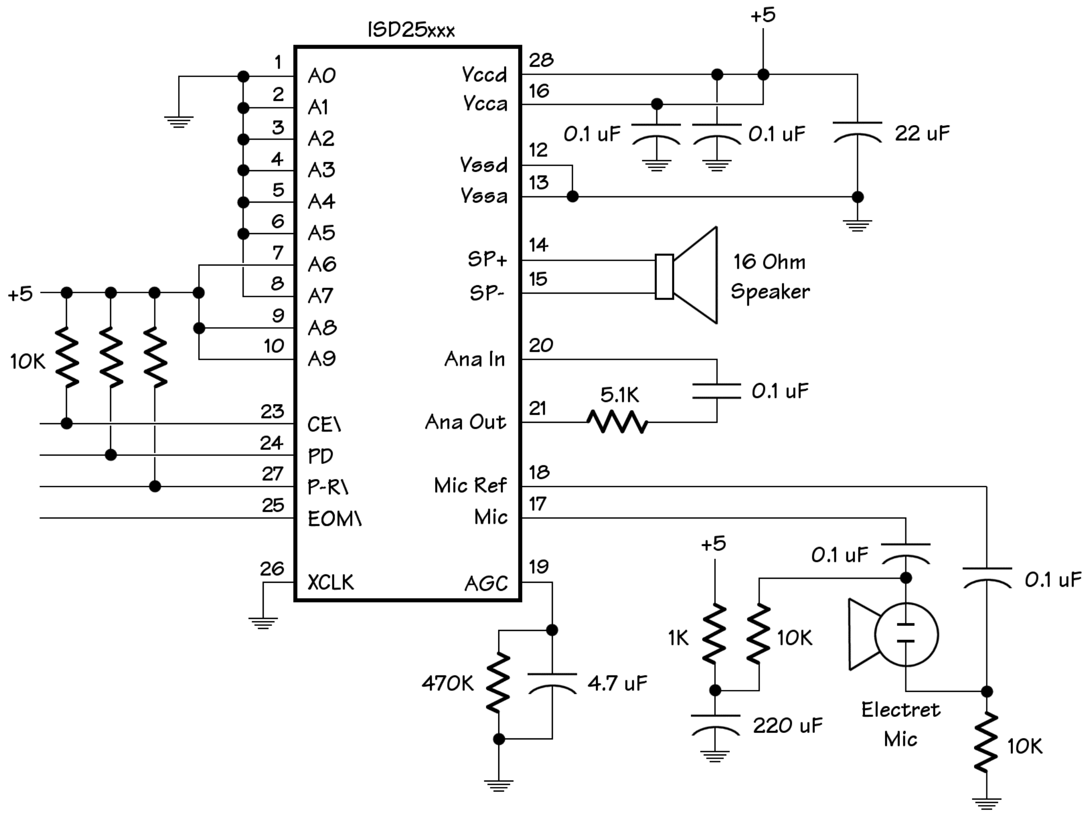


Figure 65.1 is the schematic for the IS25xxx circuit in Push-Button mode. This circuit was lifted right out of the ISD documentation. The only modification I made was to remove the switches and to add 10K pull-ups to the control inputs (CE\, PD, and P-R\). This portion of the project is, in fact, a stand-alone module and you could experiment with the ISD by adding a normally-open push-button to CE\ and SPST switches to the PD and P-R\ inputs.

Note that this circuit is compatible with the ISD2560, ISD2575, ISD2590 and ISD25120. These parts differ in the length of the recorded message space (60, 75, 90 and 120 seconds). The increase is storage space is accomplished by lowering the audio sampling rate. This does give you more storage, but the price you pay is in sound quality. I stuck with the ISD2560 because 60 seconds is quite long and the quality is very good. I personally found the sound quality of the ISD25120 unacceptable for this project.

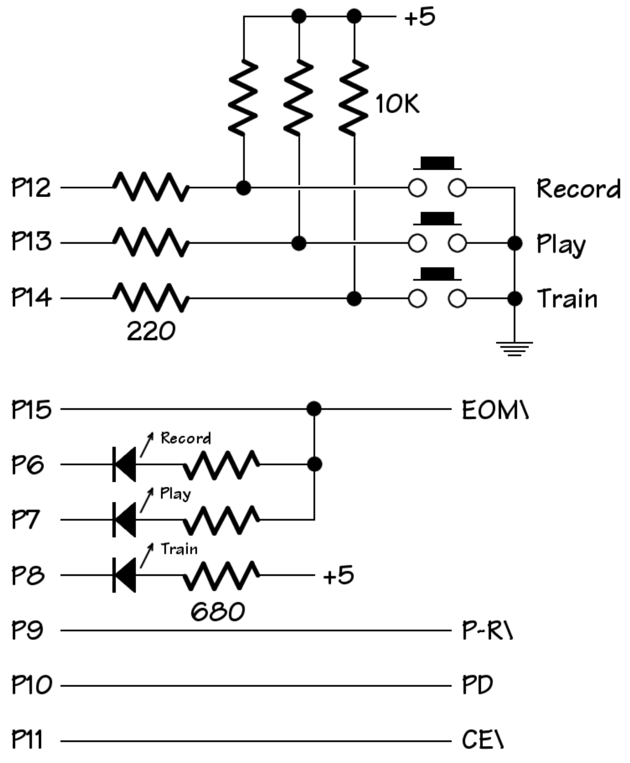
Using the ISD25xxx in Push-Button mode is very easy. Here are the steps:

- Bring PD high, then low to stop the current cycle and reset the device
- Set PR high to play, low to record
- "Blip" the CE\ line (high-low-high) to start the cycle
- Bring PD high to stop record cycle and reset the device

Let's get back to the parrot trainer application. What Mike wanted to do is record a message, press a button to test it and press another button to play the message back several times after a short delay. The idea behind the delay is to allow him to leave the room and let the bird get settled before the training begins. His original circuit connected the push-buttons directly to the ISD and monitored them with the Stamp. I changed this to allow total software control of the ISD. This way, the buttons can be software-debounced and we have more flexibility with the ISD module for future applications.

Figure 65.2 is the control interface for the Stamp 2. On pins 12, 13 and 14 is standard push-button input that I've used in several other projects. For review, the 220-ohm resistors on these pins protect the Stamp in the event that on of those pins is accidentally configured as an output and set high (5 volts) while the button on that line is pressed. If we had this condition without the resistors we'd have a direct short to ground and would damage the Stamp.

Figure 65.2: Control interface for the Stamp 2



Pin 15 (RecRun) monitors the EOM\ line from the ISD25xxx. In Push-Button mode, this line goes high while a message is playing or being recorded. Notice that I've also tied this line to the anode side of the Record (red) and Play (green) LEDs. By doing this I know that the ISD is working. The appropriate LED is selected by making its output line (6 for record, 7 for play) low.

Pin 8 controls the Train (yellow) LED. In our application, this LED will blink while in the delay period and be on solid while the training period (the time when the recorded message is being repeated) is in progress. Pins 9, 10 and 11 are the control outputs to the ISD. Okay, let's jump into the code.

Setting PD, CE\ and PR high initializes the ISD25xxx. This causes the device to reset and defaults to play mode. All of the LEDs are extinguished by setting their control outputs high. Finally, PD is brought back low; bringing the ISD out of reset allowing a new cycle can be initiated.

After initializing the program makes sure that there are no stuck buttons by scanning the button inputs until none are pressed. This section of code calls a subroutine GetBtn to scan and debounce the button inputs. I've used this code before, but I think its operation is worth repeating since button debouncing is important and this subroutine handles multiple inputs.

Let's start at the end. When we return from GetBtn, the variable called *btns* will hold the debounced inputs. We start, then, by enabling specific inputs by setting those bits in *btns*. Then we scan the inputs and logically AND the new inputs with the current value. Since our inputs are active-low, we use the inversion operator (~) to correct them. Any input that is not active will then have a value of zero. As you know, zero ANDed to any value is zero. This eliminates that particular input for this iteration of GetBtn. The inputs are scanned five times using a FOR-NEXT loop. With a 10-millisecond PAUSE in the loop, the subroutine takes just over 50 milliseconds to run. If an input is active through all five loops of the routine, its bit in *btns* will remain set (1) and be returned to the program as a valid input.

Back in the main program, we'll actually spend most of our time at the label called Scan. At this point we get the inputs, use LOOKDOWN to convert the buttons to a value from 0 to 3, then use BRANCH with this value to jump to the code that corresponds to our button input.

Let's start by recording a message. To do this, we press and hold the Record button. This causes the program to BRANCH to _Rec. There's a 250-millisecond delay here to allow us to be ready to record. PR is taken low to put the ISD25xxx into record mode then the record cycle is started by "blipping" the CE line. Advanced Stamp programmers will probably wonder why I didn't use PULSOUT to take care of the blip. I tried this method and got inconsistent results (maybe I have a marginal part). Anyway, I ended up just brute-forcing it with the Blip subroutine.

Once the ISD25xxx is activated, making Pin 6 low enables the Record LED. Remember that the EOM\ line on the ISD25xxx goes high during playback and recording. With Pin 6 low and the EOM\ high the Record LED will light.

The program then monitors the Record input and stays in a tight loop (at the label RLoop) until the button is released. When this happens the record cycle is halted and making PD high resets the ISD25xxx. The program then jumps back to the initialization section to clean up everything before waiting for the next input.

Now that we've got a message recorded, we can test it by pressing the Play button. This causes the program to BRANCH to Play. This code starts with a half-second PAUSE to allow us to release the Play button. The PR line is set high for playback mode. As with recording, blipping the CE\ line starts the cycle and making Pin 7 low enables the Play LED. The play LED will remain lit for the duration of the recorded message.

The buttons are monitored during playback to allow us to stop before the end of the message. We can do this by pressing any button. If we press a button during playback, the program jumps back to the initialization section where the cycle is halted and the ISD25xxx is reset.

If no button is pressed we need to wait until the message is finished before resetting the device and waiting for our next input. We do this by monitoring Pin 15 (RecRun). As we stated before, this line is connected to the EOM\ line of the ISD25xxx and will be stay high while the message is playing. While this is the case, the program continues to loop back to the label called PLoop. As soon as this input goes low (playback is finished) the test falls through and the device is reset.

Finally we get to the real purpose behind this project: bird training. The training cycle is initiated by pressing the Train button. This causes the program to BRANCH to _Train. Again, we use a short delay to give time for the Train button to be released. Then we light the Train LED by making Pin 8 low.

What we want to do here is flash the Train LED; one half-second on, one half-second off. This would be pretty easy to do with PAUSE statements, but it wouldn't allow us to stop the cycle without "pulling the plug" on our project. Instead, we create a half-second (approximate) delay by calling GetBtn nine times with a loop. The process gives us the time delay we need and allows us to abort the training cycle if any button is pressed. If no button is pressed we turn the Train LED off and wait again using the same technique. This on-wait-off-wait cycle is repeated 280 times using a FOR-NEXT loop. This results in a delay of about five minutes.

Once we're past the delay we turn the Train LED on solid and go into the actual training cycle. This is where we will repeat our recorded message 100 times. There is a 30-second delay between the plays unless a button is pressed which would abort the cycle. The code in this section of the program is identical to the code we've just covered so we don't have to go into details.

Wa-la, a bird trainer. Mike reported that it worked exceptionally well and yet, there is plenty of room for upgrades. If found that the ISD25xxx will drive a speaker pretty loudly. If you want volume control (more or less), you can run the output into an LM386 audio amplifier circuit. You might also make the delay time and number of plays variable by adding a couple of switches or pots. For those of you interested Mike's other bird training projects, check out his web site at www.birdwords.com.

Now some of you know that I have a goldfish and no matter how many times I play messages for him, he's not talking. So what am going to do with my trainer? Well, Halloween (my favorite day of the year) is next month and I've got plans to load this dude into a plastic skull. My intention is to use an infrared beam to start the playback, with a randomized delay before the message starts. The randomized delay will help prevent smart-Alecs from figuring out how the talking skull works. I'm also going to experiment with using the audio output to modulate glowing LEDs in the eye-sockets of the skull. It should be a fun Halloween....

If you're not inclined to build your own circuit, Parallax has a new ISD AppMod that gives you serial control over the ISD2560 and up to 8 messages. The part number is #29111 and the price is only $89.

And for those of you that are inclined to do your own thing and want to pursue more advanced options of the ISD chips, you might want to take a look at products offered by Quadravox. They have parts and modules that give you serial control over the ISD and supports multiple messages. I don't have any personal experience with their products, but I've seen some very favorable messages posted to the Stamp mail list about them.

Until next time, happy Stamping.

```
' ----[ Title ]-----------------------------------------------------------
'
' File...... PARROT.BS2
' Purpose... ISD25xxx Controller
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 03 APR 1999
' Updated... 29 JUL 2000


' ----[ Program Description ]---------------------------------------------
'
' This program, based on an original by Mike Politoski, controls
' anISD25xxx series ChipCorder(r) IC.  This program allows the user to
' record a message (length determined by the chip) and play it back
' several times to train a parrot or other talking bird.
'
' Note: The ISD25xxx is configured for Mode 6, push-button mode.
'
' Modes:
'
' - Record:   Press and hold record button
' - Play:     Press play button
' - Training: Press start button


' ----[ I/O Definitions ]-------------------------------------------------
'
LEDr_   CON      6                     ' recording LED
LEDp_   CON      7                     ' playback LED
LEDt_   CON      8                     ' training LED

PR      CON      9                     ' ISD Play/Record_
PD      CON     10                     ' ISD Power Down
CE_     CON     11                     ' ISD Chip Enable

BtnRec  VAR     In12                   ' Record control button
BtnPlay VAR     In13                   ' Play control button
BtnGo   VAR     In14                   ' Start control button
RecRun  VAR     In15                   ' recording/running out from ISD
bPort   VAR     InD                    ' buttons port


' ----[ Constants ]-------------------------------------------------------
'
On      CON     0                      ' active low
Off     CON     1

Yes     CON     1                      ' active high
No      CON     0
```

```
' ----[ Variables ]-------------------------------------------------------
'
btns    VAR    Nib                    ' debounced inputs
state   VAR    Byte                   ' button input state
delay   VAR    Word                   ' delay before training
x       VAR    Byte                   ' loop counter
y       VAR    Word                   ' loop counter
plays   VAR    Word                   ' plays counter


' ----[ Initialization ]--------------------------------------------------
'
Init:
  HIGH PD                             ' reset ISD address
  HIGH CE_
  HIGH PR                             ' default to Play mode
  HIGH LEDr_                          ' record LED off
  HIGH LEDp_                          ' play LED off
  HIGH LEDt_                          ' train LED off
  PAUSE 30                            ' allow device to reset
  LOW PD                              ' bring out of reset


' ----[ Main Code ]-------------------------------------------------------
'
Main:
  GOSUB GetBtn
  IF btns > %000 THEN Main            ' wait for release

Scan:
  GOSUB GetBtn
  LOOKDOWN btns,[%000, %001, %010, %100],state
  BRANCH state,[Scan, _Rec, _Play, _Train]
  GOTO Scan

_Rec:
  PAUSE 250                           ' time to get ready
  LOW PR                              ' record mode
  GOSUB Blip                          ' initiate recording
  LOW LEDr_                           ' record LED on

RLoop:
  GOSUB GetBtn
  IF btns = %001 THEN RLoop           ' record until release
  HIGH PD                             ' reset device
  GOTO Init

_Play:
  PAUSE 500                           ' time to release play
  HIGH PR                            ' playback mode
  GOSUB Blip                         ' initiate playback
  LOW LEDp_                          ' play LED on

PLoop:
  GOSUB GetBtn
  IF btns > %000 THEN Init           ' abort if any button pressed
  IF RecRun = Yes THEN PLoop         ' check until message done
  HIGH PD                            ' reset device
  GOTO Init

_Train:
  PAUSE 500                          ' time to release start button

  FOR delay = 1 to 280               ' 5 minute delay
    LOW LEDt_                        ' train LED on

    FOR y = 1 TO 9                   ' ~1/2 second delay
      GOSUB GetBtn                   ' - check inputs
      IF btns > %000 THEN Init       ' abort if button pressed
    NEXT

    HIGH LEDt_                       ' train LED off

    FOR y = 1 TO 9                   ' ~1/2 second delay
      GOSUB GetBtn                   ' - check inputs
      IF btns > %000 THEN Init       ' abort if button pressed
    NEXT

  NEXT ' delay

  LOW LEDt_                          ' train LED on
  FOR plays = 1 TO 100               ' play message 100 times
    HIGH PR                          ' play mode
    GOSUB Blip                       ' initiate play
    LOW LEDp_                        ' play LED on

SLoop:
    GOSUB GetBtn
    IF btns > %000 THEN Init         ' abort if any button pressed
    IF RecRun = Yes THEN SLoop       ' wait until finished playing
    HIGH PD                          ' reset the ISD
    HIGH LEDp_                       ' play LED off
    PAUSE 25
    LOW PD                           ' out of reset

    FOR y = 1 TO 500                 ' 30 second delay between plays
      GOSUB GetBtn                   ' scan buttons
      IF btns > 0 THEN Init          ' abort if button pressed
    NEXT
  NEXT

  GOTO Init                             ' reset and start over


' ----[ Subroutines ]-------------------------------------------------------
'
Blip:
  LOW CE_                               ' initiate playback or record
  PAUSE 25
  HIGH CE_
  RETURN


' scan and debounce button inputs
' - inputs must stay low for 50 ms
' - any debounced button returns high bit in "btns"
'
GetBtn:
  btns = %0111                          ' scan record, play and wait
  FOR x = 1 TO 5
    btns = btns & ~bPort                ' check current state
    PAUSE 10
  NEXT
  RETURN
```