Column #66, October 2000 by Jon Williams:

A Stamp-II ISD Sound Lab

I love "talking" projects. I really do. While I still wish that a semiconductor company would come along and resurrect the SP0256-AL2 allophone synthesizer, I'm not holding my breath. So, last month I used the ISD2560 ChipCorder® to create a talking-pet trainer. The ISD25xx chips are incredibly popular and they can be found in a very wide range of industrial and consumer products.

"Well..." I thought, after playing with the pet trainer, "I guess I'd better crack open the docs and really dig in." The pet trainer used the ISD in its simplest mode, called "Push-Button." This mode is generally intended for stand-alone applications. We used the Stamp to push the buttons and make the device train our pet.

With a little more work and some more connections, we can make the ISD25xx do all sorts of neat things. Let's give it a try.

ISD2560 Basics

For this month's project, we'll use the ISD2560. This is the most popular of the series because it has the best overall sound quality. The ISD2560 can store up to 60 seconds of sound in a unique type of patented, non-volatile memory. This means you can remove

power from the device without losing your recordings. The chip's PD pin puts it into a very low power state (when not operating), making battery-operation viable.

After a sound segment is recorded in the ISD2560, an End-Of-Message (EOM) marker is left in memory. When recording multiple messages, the next recording starts at the address following the EOM. Addressing individual recordings stored in the ISD can be accomplished two ways: indirectly (message cueing) by knowing the message number and directly by knowing the starting address of the message. Once the device starts playing, all we have to do is watch for a low-going pulse on the EOM line to know when the message is complete.

…which is easier said than done in message cueing mode. When we setup for message cueing (details later), the chip actually scans through memory (looking for EOM markers so it can skip messages) at 800 times its normal speed. What this means is that the EOM pulse could be as short as 11 microseconds! That's not much of a pulse. Thankfully, we can use a little PBASIC trick to catch it. We'll get into more ISD details as we analyze the lab code.

Our Lab And The Software That Drives It

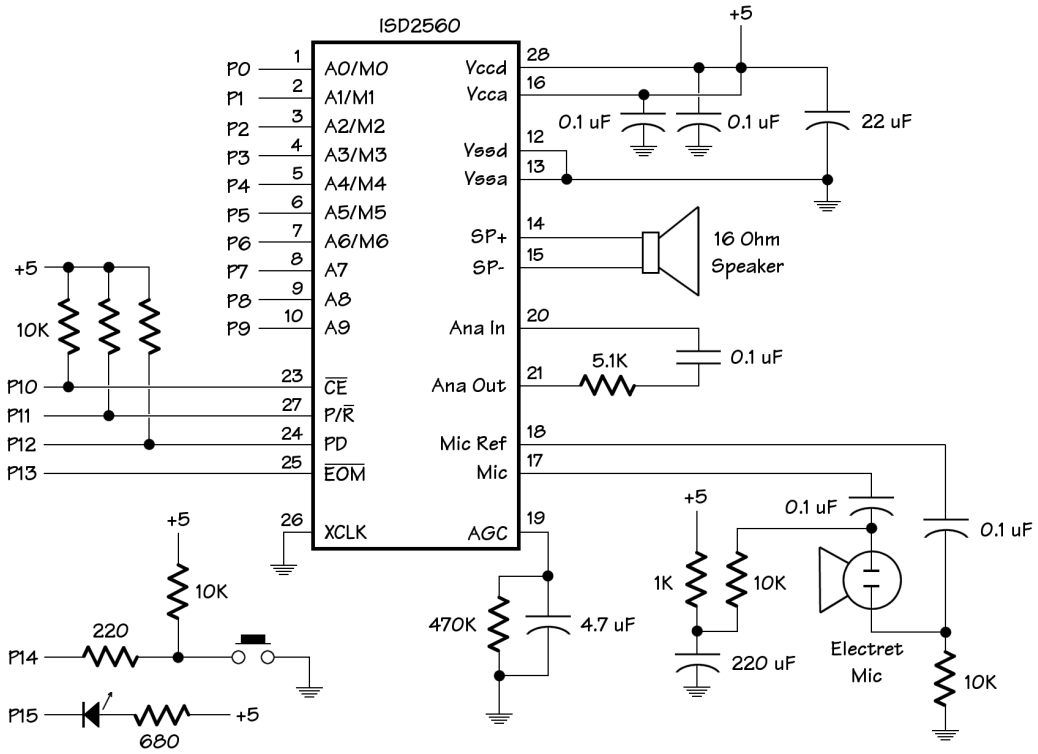
Since the idea behind this project is experimenting, I assembled mine on a solderless breadboard. What you'll immediately notice about this project is that it uses all of the Stamp's I/O lines. Don't fear, we're only doing this now to make experimenting easy. If you find you want a more permanent solution you can tie address lines together (i.e., Push-Button mode) or use shift registers to set the ISD's address lines. More on that later.

The software for our lab is called ISD_LAB.BS2. Since this program is designed to be interactive (modified and re-run), you should make a back-up copy to recover from a potential accident.

There are five modes to the program: record, sequential playback, play a message using cueing, play a message at a specific address and play a message at an address for a specified number of milliseconds (something less than the recorded message length). You control the program's behavior by changing the Mode constant.

Initialization of the Stamp is pretty straightforward and yet, I want to make an important point about the order in which things are done. You'll notice in the code that the Stamp's output lines are setup before they're actually defined as outputs. There is a reason.

Figure 66.1: ISD2560 laboratory



The Stamp initializes all variables to zero, including outputs and direction registers (defaulting them to inputs). Since the CE pin of the ISD wants a high-to-low transition, we put a 10K pull-up on it – that way we have to force it to go low. During the initialization sequence, the CE control pin is set to 1 so that it will go high when that pin is defined as an output.

Don't change this sequence. If you set the CE pin as an output before making its value 1, you'll get a glitch recording at the beginning of your ISD's memory. Go ahead; ask me how I know this. Yes, I chased that bug for a day. It's a good lesson. Always define the value of your output pins before setting them to outputs.

Moving on, the program resets the ISD by pulling the PD line high for 13 milliseconds. After the PD line goes low you must wait for 25 milliseconds before initiating any record or play cycle. You'll see that sprinkled through the code.

After basic initialization and checking to see that the button is not stuck, the program BRANCHes to the section dictated by Mode. Since the first thing we need to do is record some messages, let's go there first. We set the Mode constant to MRec (0).

To record messages into the ISD we'll use Push-Button mode, just like the pet trainer did. The difference is that we won't reset the ISD address after each recording. This way we can record any number of messages, one after the other, up to the limit of memory.

To go into Push-Button mode, we need to set the ISD's A6, A8 and A9 lines high. On the ISD, address lines zero through six are also defined as operational mode inputs. The mode inputs are active when both A8 and A9 are high. If either A8 or A9 is low, the mode inputs are treated as part of a message address.

We start by clearing any old mode bits, setting opMode to six and calling the SetModeBit subroutine. This routine uses the DCD operator to set the correct bit. It also takes care of A8 and A9. Notice too, that is adds the new opMode bit (old bits aren't cleared). The reason for this is that some of the ISD operational modes can work together. By changing opMode and calling SetModeBit again, we can set multiple bits.

With Push-Button mode setup, we set the PR (play / record) control line to record mode by bringing it low and then waiting for the button to be pressed. When the button is pressed, the program waits 250 milliseconds (so we can get our voice ready), then starts the recording by pulsing the CE line low.

In Push-Button mode, the ISD's EOM line becomes a recording indicator. The program watches for this line to go high and then turns on the LED to indicate it's time to start talking. Recording continues until the button is released. Once the button is released, pulsing the CE line again terminates recording. This causes the ISD to put an End-Of-Message marker after our recording and advance its memory pointer to the next available address.

With this message recorded, our program puts the ISD into Play mode (just to be safe), turns off the LED and loops back to wait for another button press. Note that the ISD is not reset after the recording. This would cause the address pointer to be set to zero and we could only record one message (this is how the pet trainer works).

Now it's time to check our messages. To do this, change the Mode constant to MPSeq (1). The section of code called P_Seq works identically to the Record section. Where it differs is that it puts the ISD into Play (PR high) mode, pulses the CE line, then waits for

the EOM line to go low at the end of the message. As in recording, the EOM line acts as an activity indicator when the ISD is in Push-Button mode. Each press of the button will cause another message to play until the end of memory is reached.

Playing messages in order is fun, but not always useful. Wouldn't it be nice if we could play any message or a sequence of messages in any order? We can. The ISD has an operational mode called message cueing (M0). What this mode does is fast forward through its memory to the desired message.

To test message cueing, you need to set the Mode constant to MPMsg (2) and re-run the program. The demo code at P_Msg will take the first five messages in the ISD and play them backward. Cueing messages in the ISD is a bit tricky, so I put together a subroutine to handle the details.

Take a look at the subroutine called PlayMessage – this is the code that does the fast forwarding for us. This code resets the ISD then puts it into cueing mode by setting A0 and A4. This allows cueing and consecutive messaging. To enable the mode bits, we must also set A8 and A9. With everything setup, we have to wait 25 milliseconds before proceeding.

The first thing the routine does is check to see if the message number is zero or one (I decided that the first message in memory is message number 1 – you could modify the code to make the first message number 0 without any difficulty). If the message number is zero or one, the cueing operation is skipped and the first message in the ISD is played.

Let's assume it's not message one and see how the cueing operation works. Remember that with cueing set, the ISD scans through its memory at 800 times its normal rate. What this means is that the EOM pulse could be as short as 11 microseconds. That's too short for our normal polling loop to catch.

Luckily, PBASIC provides a work around. What we do is use PULSIN to watch for the EOM pulse. Since PULSIN can measure a pulse down to 2 microseconds, we're in luck. To skip to a message, we're going to pulse the CE line and watch for the EOM one less time than the value of our message number. Once we've done all this skipping, the ISD address pointer is set to the desired message. To play the message, we must clear the cueing bit (A0) and then proceed as we have before. Once the EOM line pulses low, we know the message is down playing and we can return to the program.

You can create complex phrases playing messages in the order you chose. The only thing you need to know is the message number.

The last two modes of the demo start playback of a sound segment from a specific address in the ISD. These modes are similar in that they reset the ISD, setup the message address, wait for the device to settle (25 ms delay), then pulse the CE line low start playback. Since the ISD25xx chips have 600 memory cells, addresses are limited to 599

With Mode set to MPAddr (3), playback is automatically terminated when the EOM marker is reached. When Mode is set to MPTime (4), playback is manual manually terminated after waiting a specified number of milliseconds (set in *msgLen*) by taking the PD line high, resetting the ISD. The neat thing about MPTime mode is that we can actually parse individually sound segments out of one long recorded message. It takes some work though; we have to find the address and the length of our clip. Will give that a try in the next program.

Talking Numbers

Program Listing 66.2 is the code for a program that demonstrates a couple of the routines we just developed. What this program does is generate random numbers then says them aloud. The number is also displayed on a DEBUG terminal. There are two elements of this program: one that calls out the digits of the number and one that says the number as we humans would.

There are a couple of requirements for this program to work, however. First, you have to record the list of digits and numbers into the device using ISD_LAB.BS2 set to Record mode. What you're going to do is press the button, wait for the LED to come on, say a number, then release the button. You want each number in the list stored as an individual message in the ISD.

Here's the complete list:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9
10, 11, 12, 13, 14, 15, 16, 17, 18, 19
20, 30, 40, 50, 60, 70, 80, 90
"hundred"
"thousand"

You need to record the numbers very carefully, making sure not to create any blank messages in between. You can do a quick test by setting the Mode constant in ISD_LAB.BS2 to MPMsg (2), set the *msgNum* to 30, then running the program. If all went well you should hear "thousand" come out of the speaker. If you get something else,

you need to go back to Record mode and do it all again. Once it's done, it's done – and you'll probably get it on the first go.

Before you load up SAYNUMBR.BS2 though, you have to find the addresses and message lengths of the first ten digits (0 – 9). Change the Mode constant to MPTime (4). What you're going to do is experiment with addresses and message lengths, listening for the best results.

The way to do this is to find the address first. I found that "sneaking up" on the address was the easiest way to go. Once you've found the best starting address, work on the message length. Again, I started shorter than I thought I'd need and extended out until the message sounded good.

My numbers won't be exact, but if you speak at an "average" speed, they can get you in the ballpark.

Clip	Addr	Length
"zero"	2	550
"one"	13	660
"two"	25	570
"three"	33	680
"four"	42	690
"five"	52	700
"six"	62	750
"seven"	73	830
"eight	84	780
"nine"	95	850

Start at zero and work your way through the list. Don't forget to write down your own addresses and clip lengths!

Okay…now you can load up SAYNMBR.BS2. Don't run it yet though, you need to edit the DATA statements, substituting you clip addresses and lengths for mine. Notice that the clip lengths are stored in 10 millisecond increments. I did this because the PBASIC READ command will only read bytes from EEPROM. By using a resolution of 10 ms, we could have clip lengths of up to 2.55 seconds. This is fine for our single-digit recordings.

Now run the program. Once it's downloaded, you should get a DEBUG screen that tells you to press the button. Go ahead and do it. While the Stamp is waiting for you to press

the button, it's continuously calling the RANDOM function to generate a new number. This brings a "human" factor into the randomness. If we didn't do this, the program would give us the same pseudo-random list every time it was run.

When you do press the button, the DEBUG screen will clear and you'll see the message "Saying digits: " followed by five digits. Then, in your own voice, you'll hear the digits called out. Pretty neat, huh?

What I noticed – and you might too – is that I was a little stingy on addresses and clip lengths when I did my earlier analysis. This caused the digit list demo to sound a bit choppy and unnatural. Since the addresses and lengths are stored in DATA statements, you can fine-tune them to get a really great sounding demo. Even though it's not particularly practical in and of itself, I like this demo because it shows how cleanly sound clips can be concatenated with the ISD using the PlayLength subroutine. Yes, it takes some work, but I think it's worth it.

And keep in mind that PlayLength (with proper analysis, of course) can be used to break a single long clip into shorter fragments, reducing redundant recordings. Perhaps you want to be able to play one long segment, but also a single word or phrase from it. By locating the address and length of the word or phrase, you don't have to record it separately. This will save ISD memory space, allowing you to store a greater vocabulary. I'll bet you're already thinking about adding speech to one of your favorite projects, aren't you?

Okay, let's check out the next demo. The DEBUG screen will have cleared by now and prompted you for another button press. This time, you'll get the message: "Saying number: " followed by a number. Then, you hear the number spoken, just like you would if someone asked you to.

This segment of the program demonstrates the strengths and the weaknesses of using the ISD's message cueing mode. The strength of this mode is that we don't have to know the address or length of the message; we only need to know which message to play (starting with 1). The weakness is that clips often have "dead air" (silence) before and after the actual sound. Those silences and the occasional EOM pop become apparent when concatenating messages with cueing mode.

Let's take a look at the SayNumber code to see how this segment works. First, it checks to see if the number is zero. If it is, we set *msgNum* to 1 ("zero") and play it. If the number is greater than zero, it is broken down into three sections: thousands, hundreds

and tens/ones. We do this because it's the way we say numbers: "X thousand, Y hundred, Z."

A word-sized variable has 16 bits, so the largest number that will be returned by RANDOM is 65,535. Since the thousands portion and the tens/ones portion will be 99 or less, it makes sense to create a common subroutine to say a number from one to 99. Jump down to the subroutine called SayValueXX.

The first thing that this routine does is make sure that its target variable, *tmpVal*, is within the range of one to 99. If not, the routine is skipped, "saying" nothing. While this sort of range checking is not strictly necessary, it will save use headaches when we start copying this code to other programs.

Assuming the value is good, it's checked to see if it's less than 20. If this is the case, the code jumps down to the label SV_1TO19. At this point, we simply need to add one to the value (since "zero" is message 1) to get the message number and say it. After that we can return to the caller.

If the target value is greater than 19, we calculate the first message number by dividing the value by 10, then adding 19. This will cause the ISD to say "twenty", "thirty" and so on. Now we take *tmpVal* // 10 to see if there are any ones in our value (remember that // returns the remainder of a division). If there is a ones value, the code at SV_1TO19 handles it as I described above. If the result is zero, we're done.

Back to saying our number. The first thing we're going to do is divide our random number, *randW*, by 1000. If the result is greater than zero, we'll use SayValueXX to say it and then we'll say "thousand" by playing message 30. Saying the hundreds value works the same except the "hundreds" message is number 29. Finally, we're down to less than 100 and we can just say it. It's really very easy once the process has been broken into parts.

So once again, we've concatenated discrete sound clips into a single message. If you're careful not to leave too much dead space at the beginning and end of your messages, this technique is easy to use and can be quite effective.

Making It Permanent

You may get the itch – as I have – to build a permanent version of the ISD Lab. The problem is all the Stamp I/O lines that get eaten up. My suggestion is that you use two 74HC595 shift registers, connected in series to handle all of the address lines (A0..A9),

PD (pin 24) and PR (pin 27). The CE and EOM pins should be connected directly to the Stamp.

Of course, the software will need updating too. If I get enough requests, I'll post my schematic and updated code.

Next Time

I seem to be on a sound wave, as it were, and we'll continue with that theme next month. My friend and fellow DPRG member, Robert Jordan, has shown me some really neat tricks with the Stamp's FREQOUT and DTMFOUT commands. I'll be passing them on to you.

Until then, Happy Stamping.

```
' Nuts & Volts - Stamp Applications
' October 2000 (Program Listing 66.1)

' ----[ Title ]-----------------------------------------------------------
'
' File..... ISD_LAB.BS2
' Purpose... ISD2560 ChipCorder Lab
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 04 SEP 2000
' Updated... 08 SEP 2000


' ----[ Program Description ]---------------------------------------------
'
' This program facilitates experimenting with the ISD2560 ChipCorder.
'
' Program Modes:
'
' 0 - Record one or more messages to the ISD
' 1 - Play messages, one at a time, in order
' 2 - Play a specific message
' 3 - Play sound clip from specific address - auto stop at EOM
' 4 - Play sound from address for number of milliseconds


' ----[ I/O Definitions ]-------------------------------------------------
'
AddrL   VAR     OutL                    ' low bits of ISD address (0..7)
AddrH   VAR     OutC                    ' high bits of ISD address (8..9)

CE_     CON     10                      ' ISD chip enable
PR      CON     11                      ' ISD play/record (play = 1)
PD      CON     12                      ' ISD power down (reset = 1)
EOM_    VAR     In13                    ' ISD end of message indicator
EOMpin  CON     13                      ' EOM pin number

BtnIn   VAR     In14                    ' button input
LED_    CON     15                      ' LED output (active low)


' -----[ Constants ]-----------------------------------------------------
'
MaxAddr CON     599                     ' last address in ISD25xx

MRec    CON     0                       ' record
MPSeq   CON     1                       ' play back messages in order
MPMsg   CON     2                       ' play specific message
MPAddr  CON     3                       ' play sound from address
MPTime  CON     4                       ' play for n milliseconds
```

```
Mode    CON     MPSeq                ' default to sequential play


' ----[ Variables ]-----------------------------------------------------
'
btn     VAR     Bit                  ' pushbutton input
x       VAR     Byte                 ' loop counter
msgNum  VAR     Byte                 ' message number
msgAddr VAR     Word                 ' clip segment address (0 - 599)
msgLen  VAR     Word                 ' clip length in milliseconds
waste   VAR     Word                 ' workspace for PULSIN
opMode  VAR     Nib                  ' for use with SetModeBit
cntr    VAR     Byte                 ' loop counter


' ----[ Initialization ]-------------------------------------------------
'
Init_IO_Pins:
  AddrL = %00000000                  ' clear message address bus
  AddrH = %1100                      ' play, CE_ high, clear A9 & A8

  DirL = %11111111                   ' low bits of address
  DirC = %1111                       ' PR, CE_, A9, A8
  DirD = %1001                       ' LED_, Btn, EOM_, PD

ISD_Setup:
  GOSUB ResetISD
  PAUSE 25                           ' let ISD settle (Tpud)

LED_Off:
  HIGH LED_


' ----[ Main Code ]------------------------------------------------------
'
Main:
  GOSUB GetBtn
  IF btn = 1 THEN Main               ' wait for release

CheckMode:
  BRANCH Mode,[Record,P_Seq,P_Msg,P_Addr,P_Time]
  GOTO Main

' ===============
' Record messages
' ===============

Record:
  AddrL = 0                          ' clear old mode bits
  opMode = 6
```

```
  GOSUB SetModeBit                    ' record in pushbutton mode
  LOW PR                              ' record mode

R_Hold:
  GOSUB GetBtn
  IF btn = 0 THEN R_Hold              ' wait for button

  PAUSE 250                           ' time to get voice ready
  PULSOUT CE_,50                      ' initiate recording

R_Wait:
  IF EOM_ = 0 THEN R_Wait             ' wait for recording indicator
  LOW LED_                            ' record LED on

R_Loop:
  GOSUB GetBtn
  IF btn = 1 THEN R_Loop              ' record until button release
  PULSOUT CE_,50                      ' mark end of message
  HIGH PR                             ' back to play (for safety)
  GOTO LED_off                        ' clear LED

' ======================
' Play messages in order
' ======================

P_Seq:
  AddrL = 0                           ' clear old mode bits
  opMode = 6
  GOSUB SetModeBit                    ' play with pushbutton mode

PS_Hold:
  GOSUB GetBtn
  IF btn = 0 THEN PS_Hold             ' wait for button

  PULSOUT CE_,50                      ' start message
  PAUSE 50                            ' let EOM go HIGH

PS_Wait:
  IF EOM_ = 1 THEN PS_Wait            ' wait for message to end

  GOTO PS_Hold                        ' get ready for next

' ======================
' Play specified message
' ======================

P_Msg:
  GOSUB GetBtn
  IF btn = 0 THEN P_Msg               ' wait for button

  FOR cntr = 5 TO 1                   ' say 5 messages backward
```

```
    msgNum = cntr
    GOSUB PlayMessage
  NEXT

  GOTO P_Msg

' ================================
' Play message at specified address
' ================================

P_Addr:
  GOSUB GetBtn
  IF btn = 0 THEN P_Addr            ' wait for button

  msgAddr = 107                     ' set address
  GOSUB PlayClip                    ' play to EOM marker

  GOTO P_Addr

' ================================
' Play message for specified duration
' ================================

P_Time:
  GOSUB GetBtn
  IF btn = 0 THEN P_Time            ' wait for button

  msgAddr = 95                      ' set address
  msgLen = 850                      ' set length (milliseconds)
  GOSUB PlayLength                  ' play for specific length

  GOTO P_Time


' ----[ Subroutines ]------------------------------------------------------
'
ResetISD:
  HIGH PD                           ' reset the ISD
  PAUSE 13
  LOW PD
  RETURN


' scan and debounce button input
' - button must stay pushed for 25 ms and not change during routine
'
GetBtn:
  btn = 1                           ' assume pressed
  FOR x = 1 TO 5
    btn = btn & ~BtnIn              ' test input
    PAUSE 5                         ' delay between tests
```

```
  NEXT
  RETURN


PlayMessage:                          ' skip to specific message
  GOSUB ResetISD
  AddrL = %00010001                   ' consecutive messages with cueing
  AddrH = %1111                       ' play, CE_ high, enable mode bits
  PAUSE 25                            ' wait Tpud

  IF (msgNum < 2) THEN PM_Play_It     ' play first if 0 or 1

PM_Fast_Forward:
  msgNum = msgNum - 1                 ' don't skip selected message
  FOR x = 1 TO msgNum                 ' fast forward to it
    PULSOUT CE_,3                     ' start FF
    PULSIN EOMpin,0,waste             ' wait for EOM of current message
  NEXT

PM_Play_It:
  AddrL = %00010000                   ' remove cueing bit
  PULSOUT CE_,50                      ' start the message
  PAUSE 50                            ' let EOM set

PM_Wait:
  IF EOM_ = 1 THEN PM_Wait            ' wait for EOM to pulse low
  RETURN


PlayClip:                             ' play until EOM hit
  GOSUB ResetISD
  GOSUB SetAddress                    ' set address of message
  PAUSE 25
  PULSOUT CE_,50                      ' start play
  PAUSE 50                            ' allow EOM to get set

PC_Wait:
  IF EOM_ = 1 THEN PC_Wait            ' wait for EOM to pulse low
  RETURN


PlayLength:                           ' play for msgLen milliseconds
  GOSUB ResetISD
  GOSUB SetAddress                    ' set address of message
  PAUSE 25                            ' wat Tpud
  PULSOUT CE_,50                      ' start play
  PAUSE msgLen                        ' wait for message to end
  HIGH PD                             ' stop
  RETURN
```

```
SetAddress:
  msgAddr = msgAddr MAX MaxAddr             ' limit address to chip
  AddrL = msgAddr.LowByte                   ' set A0..A7
  AddrH = %1100 | (msgAddr >> 8)            ' set play, CE_ high, A8..A9
  RETURN


SetModeBit:
  AddrL = AddrL | (DCD opMode)              ' add operational mode bit
  AddrH = %1111                             ' set play, CE_ high, enable
mode
  RETURN
```

```
' Nuts & Volts - Stamp Applications
' October 2000 (Listing 66.2)

' ----[ Title ]---------------------------------------------------------
'
' File..... SAYNUMBR.BS2
' Purpose... Uses an ISD2560 to "say" and number
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 07 SEP 2000
' Updated... 08 SEP 2000


' ----[ Program Description ]-------------------------------------------
'
' This program demonstrates ISD message concatenation.  The program has
' two modes:
'
' MSDig (say digits)
' ==================
' In this mode, the individual digits of the random number are played.
' This code demonstrates playing sounds from a specific address for a
' specified length of time.  Digit addresses and durations are stored in
' DATA statements (duration stored in 10 ms increments to fit into Byte).
'
' MSNum (say number)
' ==================
' In this mode, the number is spoken like we would.  This code demos
' cueing.
'
' Requirements
' ============
' For this program to work, you need to use ISD_LAB.BS2 to record the list
' of digits (below) and determine clip timing (in 10 ms units) of the
' digits 0 through 9.
```

```
'
' Record this list (as individual clips) to the ISD:
'
'       0, 1, 2, 3, 4, 5, 6, 7, 8, 9
'       10, 11, 12, 13, 14, 15, 16, 17, 18, 19
'       20, 30, 40, 50, 60, 70, 80, 90,
'       "hundred", "thousand"


' ----[ I/O Definitions ]---------------------------------------------
'
AddrL   VAR     OutL                    ' low bits of ISD address (0..7)
AddrH   VAR     OutC                    ' high bits of ISD address (8..9)

CE_     CON     10                      ' ISD chip enable
PR      CON     11                      ' ISD play/record (play = 1)
PD      CON     12                      ' ISD power down (reset = 1)
EOM_    VAR     In13                    ' ISD end of message indicator
EOMpin  CON     13

BtnIn   VAR     In14                    ' button input
LED_    CON     15                      ' LED output (active low)


' ----[ Constants ]---------------------------------------------------
'
MaxAddr CON     599                     ' last address in ISD25xx

MSDig   CON     0                       ' say digits
MSNum   CON     1                       ' say number


' ----[ Variables ]---------------------------------------------------
'
btn     VAR     Bit                     ' pushbutton input
x       VAR     Byte                    ' loop counter
mode    VAR     Bit                     ' demo mode
msgNum  VAR     Byte                    ' message number
msgAddr VAR     Word                    ' clip segment address (0 - 599)
msgLen  VAR     Word                    ' clip length in milliseconds
waste   VAR     Word                    ' workspace for PULSIN
opMode  VAR     Nib                     ' for use with SetModeBit
randW   VAR     Word                    ' random number
digit   VAR     Byte                    ' work value for saying digits
tmpVal  VAR     Byte                    ' work value for number to speech


' ----[ EEPROM Data ]-------------------------------------------------
'
DAddr   DATA    2,13,25,33,42,52,62,73,84,95 ' ISD address ("0" - "9")
DTime   DATA    55,66,57,68,69,70,75,83,78,85' time (10 ms units)
```

```
' ----[ Initialization ]-----------------------------------------------------
'
Init_IO_Pins:
  AddrL = %00000000                  ' clear message address bus
  AddrH = %1100                      ' play, CE_ high, clear A9 & A8

  DirL = %11111111                   ' low bits of address
  DirC = %1111                       ' PR, CE_, A9, A8
  DirD = %1001                       ' LED_, Btn, EOM_, PD

ISD_Setup:
  GOSUB ResetISD
  PAUSE 25                           ' let ISD settle (Tpud)

LED_Off:
  HIGH LED_


' ----[ Main Code ]-----------------------------------------------------------
'
Main:
  GOSUB GetBtn
  IF btn = 1 THEN Main               ' wait for release

  DEBUG CLS, "Press the button..."

BWait:
  RANDOM randW                       ' generate new random number
  GOSUB GetBtn
  IF btn = 0 THEN BWait              ' wait for press
  BRANCH Mode,[SayDigits,SayNumber]  ' branch to selected demo
  GOTO Main


' ============================
' Say individual digits
' - use clip address and length
' ============================

SayDigits:
  DEBUG CLS, "Saying digits: ", DEC5 randW, CR

  FOR x = 4 TO 0                     ' say all 5 digits
    digit = randW DIG x              ' get the digit
    READ (DAddr + digit),msgAddr     ' set the ISD digit address
    READ (DTime + digit),msgLen      ' read the length
    msgLen = msgLen * 10             ' set to milliseconds
    GOSUB PlayLength                 ' say the digit
  NEXT
```

```
  mode = ~mode                        ' flip mode
  GOTO Main                           ' do it all again


' ================================
' Say number as we would speak it
' ================================

SayNumber:

  DEBUG CLS, "Saying number: ", DEC randW, CR

  IF randW = 0 THEN SN_Zero           ' if zero, just say it

  tmpVal = randW / 1000               ' get 1000s
  IF tmpVal = 0 THEN SN_100s          ' if zero, skip to 100s
  GOSUB SayValueXX                    ' say thousands value
  msgNum = 30
  GOSUB PlayMessage                   ' say "thousand"

SN_100s:
  tmpVal = (randW // 1000) / 100      ' get 100s
  IF tmpVal = 0 THEN SN_10            ' if zero, skip to 10s
  GOSUB SayValueXX                    ' say 100s value
  msgNum = 29
  GOSUB PlayMessage                   ' say "hundred"

SN_10:
  tmpVal = randW // 100               ' get 10s and 1s
  GOSUB SayValueXX                    ' say value
  GOTO SN_Done

SN_Zero:
  msgNum = 1
  GOSUB PlayMessage                   ' say "zero"

SN_Done:
  mode = ~mode                        ' flip mode
  GOTO Main


' ----[ Subroutines ]-------------------------------------------------
'
ResetISD:
  HIGH PD                             ' reset the ISD
  PAUSE 13
  LOW PD
  RETURN
```

```
' scan and debounce button input
' - button must stay pushed for 25 ms and not change during routine
'
GetBtn:
  btn = 1                             ' assume pressed
  FOR x = 1 TO 5
    btn = btn & ~BtnIn                ' test input
    PAUSE 5                           ' delay between tests
  NEXT
  RETURN


PlayMessage:                          ' skip to specific message
  GOSUB ResetISD
  AddrL = %00010001                   ' consecutive messages with cueing
  AddrH = %1111                       ' play, CE_ high, enable mode bits
  PAUSE 25                            ' wait Tpud

  IF (msgNum < 2) THEN PM_Play_It     ' play first if 0 or 1

PM_Fast_Forward:
  msgNum = msgNum - 1                 ' don't skip selected message
  FOR x = 1 TO msgNum                 ' fast forward to it
    PULSOUT CE_,3                     ' start FF
    PULSIN EOMpin,0,waste             ' wait for EOM of current message
  NEXT

PM_Play_It:
  AddrL = %00010000                   ' remove cueing bit
  PULSOUT CE_,50                      ' start the message
  PAUSE 50                            ' let EOM set

PM_Wait:
  IF EOM_ = 1 THEN PM_Wait            ' wait for EOM to pulse low
  RETURN


PlayClip:                             ' play until EOM hit
  GOSUB ResetISD
  GOSUB SetAddress                    ' set address of message
  PAUSE 25
  PULSOUT CE_,50                      ' start play
  PAUSE 50                            ' allow EOM to get set

PC_Wait:
  IF EOM_ = 1 THEN PC_Wait            ' wait for EOM to pulse low
  RETURN


PlayLength:                           ' play for msgLen milliseconds
  GOSUB ResetISD
```

```
  GOSUB SetAddress                    ' set address of message
  PAUSE 25                            ' wat Tpud
  PULSOUT CE_,50                      ' start play
  PAUSE msgLen                        ' wait for message to end
  HIGH PD                             ' stop
  RETURN


SetAddress:
  msgAddr = msgAddr MAX MaxAddr       ' limit address to chip
  AddrL = msgAddr.LowByte             ' set A0..A7
  AddrH = %1100 | (msgAddr >> 8)      ' set play, CE_ high, A8..A9
  RETURN


SetModeBit:
  AddrL = AddrL | (DCD opMode)        ' add operational mode bit
  AddrH = %1111                       ' set play, CE_ high, enable mode
  RETURN


' ===============
' Talking Numbers
' ===============
'
' say a value between 1 and 99
'
SayValueXX:
  IF (tmpVal < 1) | (tmpVal > 99) THEN SV_Done
  IF tmpVal < 20 THEN SV_1TO19        ' skip if less than 20
  msgNum = (tmpVal / 10) + 19         ' calculate 10s message #
  GOSUB PlayMessage                   ' say 10s
  tmpVal = tmpVal // 10               ' calculate 1s message #
  IF tmpVal = 0 THEN SV_Done          ' skip if zero

SV_1TO19:
  msgNum = tmpVal + 1                 ' correct 0-offset
  GOSUB PlayMessage                   ' say 1s

SV_Done:
  RETURN                              ' all done
```