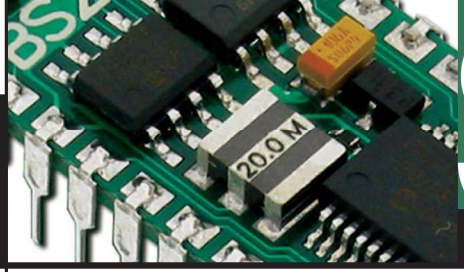
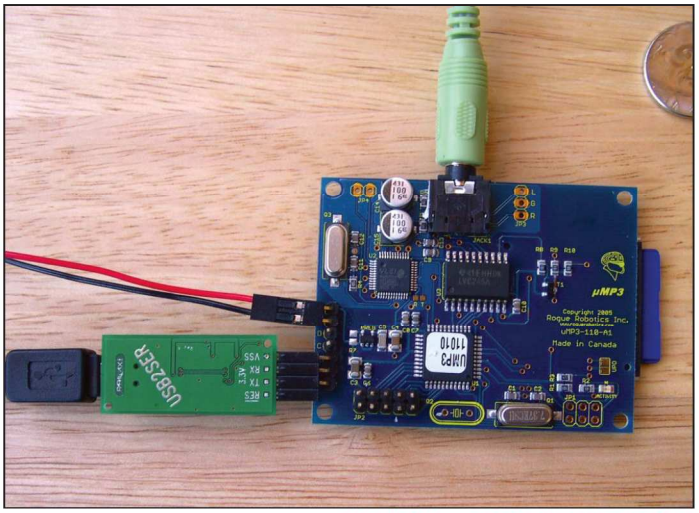


■ BY JON WILLIAMS

THE SOUNDS OF THE SEASON

Wow, can you believe we're at the end of the year already? Another one gone by so quickly. But it's easy to tell — decorations are everywhere, the crowds in the malls are growing larger every day, and the sound of music is in the air. (Boy, that last bit was cheesy, wasn't it?) Have you ever noticed that? This time of year music seems to be playing everywhere. Seasonal music, of course — I guess there's just more of it for this particular season. So, let's go with the seasonal flow, shall we? Let's build a digital music player to compliment the holiday decorating we've done around the house.



■ FIGURE1: uMP3 VIA USB2SER

A few months ago, one of my Texas prop building friends — an exuberant guy named Vern Graner — told me about a cool MP3 player module from Rogue Robotics called the uMP3 (micro MP3) that was BASIC Stamp compatible. It turns out that Vern is right: the uMP3 player is cool, it is BASIC Stamp compatible, and thanks to recent efforts at Rogue Robotics, it's even easier to use with BASIC Stamps — including the Prop-1 controller.

Now, before we get into this let me start telling you that the program we're going to work with here requires the uMP3 to be running firmware 110.11 or later. You can get this from Rogue Robotics and update the uMP3 player through a serial port. Make sure that you do this before you connect it to your BASIC Stamp module.

LET THERE BE MUSIC

The uMP3 player has a lot of interesting features, but for the sake of this month's project we're going to keep things simple. What we're going to do is randomly play one of the songs on the player, and won't repeat a song until all have been played. While the program is neat and straightforward, it has a couple tricks inside that will be useful in other projects.

UMP3 SETUP

Before we connect the uMP3 player to the BASIC Stamp, we need to configure it with a terminal program. Once we've adjusted the settings to our liking, they're non-volatile, and we don't have to worry about making them again unless we update the firmware (which clears all user settings to default values).

The uMP3 has TTL level serial connections so I used the Parallax USB2SER module to connect the uMP3 player to my PC. The photo in Figure 1 shows the USB2SER connected to the uMP3. Note the upside-down orientation of the USB2SER so that it matches the uMP3 connections. And just a comment on the photos: my unit was originally a 111.10 version (hence the label) and didn't have header pins; I added those myself. Later versions include header pins installed by Rogue, so they may look different than mine.

Using a terminal program, we can check the firmware and update the settings to make things operate smoothly with the BASIC Stamp. Figure 2 shows a HyperTerminal session with the uMP3. Out of the box, the uMP3 is configured for 9600 baud so that's what the terminal was set for. At power-up, the uMP3 scans the connected media (SD card) and, when ready, it will

transmit a ">" to the terminal.

We can check the firmware version with the "v" (no quotes) command. Note that all uMP3 commands are terminated with a carriage return, and we especially have to remember this for our BASIC Stamp program. As you can see, I have the version (110.11) that supports a feature we need (prompt delay).

Before connecting to the BASIC Stamp, we want to change the prompt delay to five milliseconds. This will let the SERIN instruction load and get ready after a command is sent to the uMP3 with SEROUT — that way we can catch any response and use it. Change the prompt delay with "st r 5" and then confirm the setting entering "st r" without a delay time. The current delay will be reported before the prompt character (it should be 5).

Okay, now we're ready to get going. Connecting to the uMP3 player falls into the "no-brainer" category — just two wires for serial (TX and RX) and a common ground (I used a cable hacked from a long-dead servo). Remember that the uMP3 player uses TTL level signals, so it's Stamp-compatible without any interfacing.

Figure 3 show the electrical connections between the Stamp and the uMP3, and the photo in Figure 4 shows the player and everything plugged in. Since the output is low level (i.e., for headphones), I connected a set of low-cost amplified computer speakers (green plug). Of course, you'll have to provide (regulated) five volts to the uMP3, as well. If you're experimenting with a BOE or PDB, you can pull power from it.

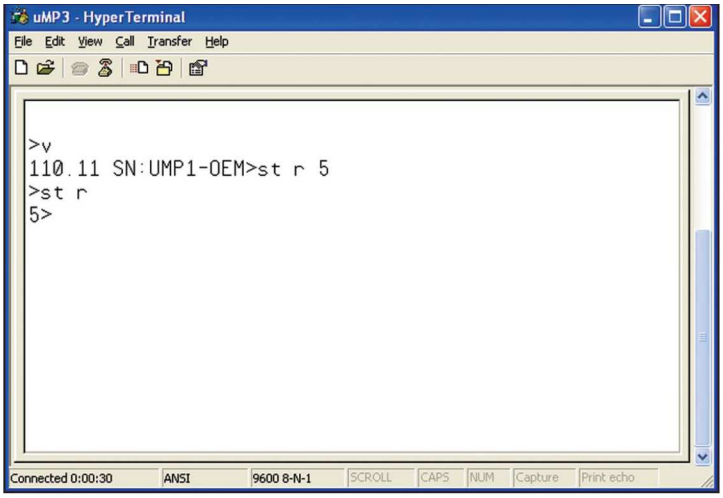
PLAY IT AGAIN ... BUT NOT UNTIL I'VE HEARD 'EM ALL

The purpose of our program this month is to play MP3 files — perhaps background music as we're having holiday dinner with friends and family. Let's make it fun, though; let's make it behave like a CD player in "shuffle" mode and force it to play all songs on the list before repeating. Before we attempt to play any of the songs on the uMP3, we need to make sure we're actually connected to it. Here's the Reset section:

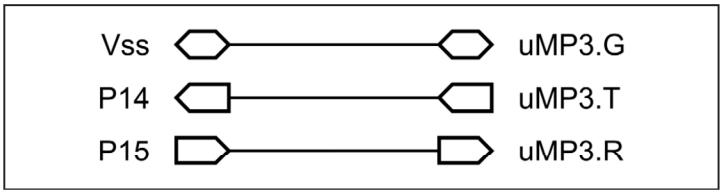
```
Reset:
  PAUSE 2000
  SEROUT TX, Baud, [CR]
  SERIN  RX, Baud, 250, Reset, [WAIT(">")]

  lottery = 1225
  GOSUB Reset_Markers
```

At start-up we wait a couple seconds before attempting to get a prompt from the uMP3; this lets the uMP3 player scan the connected media (note that if you format your SD card for FAT16 it takes a long time to scan — stick to FAT32). After the delay, we'll send CR until the prompt comes back. After that, we seed



■ FIGURE 2: uMP3 SETUP



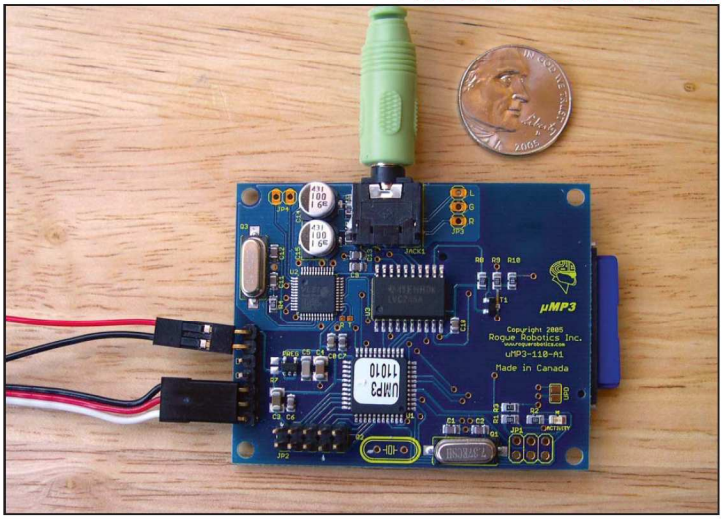
■ FIGURE 3: uMP3 CONNECTIONS

the random number generator value (*lottery*) and reset the song-played markers.

To keep everything manageable, I decided to limit the play list to 16 songs. What this lets us do is use a single Word variable to track the songs that have played in the current cycle. Since we'll want to reset the play list at the beginning of the program and when all songs are played, we'll put that function into a subroutine:

```
Reset_Markers:
  markers = $FFFF
  FOR idx = 1 TO NumSongs
    markers = markers << 1
  NEXT
  RETURN
```

■ FIGURE 4: uMP3 PLAYER



This is pretty simple, really, and yet a very handy trick. We start by setting all of the bits in markers to one, and then looping through the number of bits (starting at the LSB) that match our song count, clearing them to zero by shifting the other bits left. If, for example, we had on six songs in our play list, *markers* would be set to

%1111111111000000

by this routine. What if we want to extend the play list past 16 songs? That's not a big challenge, we just use two (or more) variables. Here's how we could reconfigure for 24 songs (one Word plus one Byte required to store the markers):

```
Reset_Markers:
  IF (NumSongs < 17) THEN
    markersHi = $FF
    markersLo = $FFFF
    FOR idx = 1 TO NumSongs
      markersLo = markersLo << 1
    NEXT
  ELSE
    markersHi = $FF
    markersLo = $0000
    FOR idx = 17 TO NumSongs
      markersHi = markersHi << 1
    NEXT
  ENDIF
  RETURN
```

Okay, now we get to the meat of things. The core of the program starts by tumbling the random number generator and then uses that value to select a song from the embedded play list. If the song has already played in this cycle, then we will go back to **Main** and try again. If the current song selection hasn't played yet, and it wasn't the last song in the previous cycle, we'll mark it, pass its number to **Play_MP3**, and on returning, check to see if all of the songs on our list have played. If that's the case, the list gets reset. Here's the core code:

```
Main:
  RANDOM lottery
  theSong = lottery // NumSongs
  IF (markers.LOWBIT(theSong) = 1) THEN Main
  IF (theSong = lastSong) THEN Main

Play_The_Song:
  markers.LOWBIT(theSong) = 1
  lastSong = theSong
  GOSUB Play_MP3
  IF (markers = $FFFF) THEN
    GOSUB Reset_Markers
  ENDIF
  PAUSE SongDelay
  GOTO Main
```

Okay, now for the hard work and, truthfully, it's not that hard. With the song number in hand, what we have to do now is tell the uMP3 to play it. Here's where we need to make a choice: We could choose to make the code very simple by forcing the file names into a non-friendly mode (e.g, "F001.MP3") or, we could work a bit with our program so that we can use the file names as-is. My choice is to spend a little effort writing code so that we don't have to do anything special with the file names on the uMP3 player module.

Let's say we had "Jingle Bells.MP3" in the root folder of our uMP3 and wanted to play it using a terminal program. We'd enter this command:

>PC F /Jingle Bells.MP3 [Enter]

If we keep all of our files in the root of the SD card, and know that all files have the ".MP3" extension, all we have to do is store the song title; we can do that like this:

```
MP3s    DATA    "Jingle Bells", 0
```

Using the z-string approach will allow us to scan through the memory to find the selected song, we just need to know where the list starts and what song number to play. Let's have a look at the code that sends the play command and song name to the uMP3.

```
Play_MP3:
  eePntr = MP3s
  IF (theSong > 0) THEN
    zCount = 0
    DO
      READ eePntr, char
      eePntr = eePntr + 1
      IF (char = 0) THEN
        zCount = zCount + 1
      ENDIF
    LOOP UNTIL (zCount = theSong)
  ENDIF

  SEROUT TX, Baud, ["PC F /"]
  DO
    READ eePntr, char
    eePntr = eePntr + 1
    IF (char = 0) THEN EXIT
    SEROUT TX, Baud, [char]
  LOOP
  SEROUT TX, Baud, [".MP3", CR]
  PAUSE 100
  DO
    SEROUT TX, Baud, ["PC Z", CR]
    SERIN  RX, Baud, [char, DEC pos, DEC loopNum]
  LOOP UNTIL (char = "S")
  RETURN
```

The routine starts by setting the variable eePntr to the beginning of the songs list (in **DATA** statements). If the song number is greater than zero, what we have to do is fast-forward through the list to the selected song title. This is a pretty simple matter: we simply read characters from EEPROM and count the number of zeros encountered. Notice that the loop that reads the characters from the **DATA** statements always increments the pointer after the **READ** instruction. This positions the pointer properly when the final zero

is located.

Okay, now that our EEPROM pointer is set we can start the command by sending "PC F /" to the uMP3. We'll follow that with a loop that reads and sends the characters in the song title. Finally, we append ".MP3" and activate the uMP3 with a CR. Pretty simple, isn't it? And remember that if we use a member of the BS2p family, we could put the songs list in another program slot and use the **STORE** instruction to point to that slot — this would let us have a very long list of songs. Using this strategy is great for multilingual applications where each slot represents a different language — the program just needs to point to the slot that has the language of choice.

One of the features of the uMP3 that's useful for this application is its ability to report status while a song is playing. We can get status from the uMP3 by sending "PC Z." This will give us a status character ("P" for playing, "S" for stopped, "D" for paused), the current position (in seconds), and the loop counter for the file. Note that the position and loop counter are returned as text, so we use the DEC modifier to convert the values on-the-fly. This is another reason we left the uMP3 player at 9600 baud — using numeric conversion modifiers (DEC, BIN, HEX) in **SERIN** doesn't work well above 9600 baud (because of the inter-byte processing required by the modifier).

A **DO-LOOP** structure is used to send the "PC Z" command and then wait for the status character, position, and loop count. In other applications, we might use the position value for prop synchronization, but we'll ignore that in this program. Once the status character changes from "P" to "S," we can return to the main part of the program and play another song.

And that's about it — a fairly simple program that lets us play full-blown MP3 files with a cool little player module. For those that are into prop building (and the candles program in October proved there are a bunch of you), this opens up a lot of neat possibilities. The program we just worked with plays continuously, but we could easily add a trigger input, perhaps playing a random tune when someone rings the doorbell. That would certainly be more cheerful that the ubiquitous "*ding-dong*," wouldn't it?

MP3S MADE EASY

If you're new to audio projects and don't have any tools for creating MP3s, you're in luck as a great one is available at no charge: Audacity. This cool bit of freeware lets you edit audio and export to a variety of standard formats. Audacity does not have the ability to create MP3s built in, but it does support an MP3 encoder DLL (called LAME), and that's available on the Audacity site, as well. I've used Audacity and LAME to create MP3s for my uMP3 player, and even ring tones for my cell phone.

A particularly useful feature of Audacity is the ability to record from your system's audio mixer. I've used this feature to convert MIDI files (available by the truckload on the Internet) to MP3s that I can load into the uMP3. To do this, you set Audacity's record input to Mixer, and then load the MIDI file into your normal media player (Audacity can open MIDI files, but not play or convert them directly). Once the MIDI file is loaded into your media player, click the Record button in Audacity, and then Start on the media player. Audacity will capture the audio stream and record it as a new track. When the MIDI file is done playing, click on Stop in Audacity. It's a pretty simple matter to trim the leading and trailing blank space before exporting the audio as an MP3. I've used this same technique to pull songs from a CD and sound FX from a DVD.

Finally, let me remind you that just because you can "rip" audio from a CD or DVD that you purchased, that doesn't mean you can play the file publicly — to do so without getting clearance from the publisher is illegal, so please don't do it. If you do want to build a prop or display that uses commercial audio, contact the publisher and do the right thing (i.e., pay the license fee). The artists of the world — starving or otherwise (and including me, an actor) — will thank you for your support.

Well, that's it for this month — and this year. Please accept my very best wishes for you and yours this holiday season, and let's hope that we all have a very happy new year. Until 2006, Happy Stamping! ■