Column #90 October 2002 by Jon Williams:

Play It Again, Stamp!

In August I made a quick trip to Las Vegas to visit friends. I love that city. It never sleeps and no matter how wacky you think you are, there's a good chance you'll encounter someone even wackier in Las Vegas.

One evening we were strolling the famous Las Vegas strip and stopped in front of the Bellagio hotel to watch an incredible fountain display. Those of you who have seen it know what I'm talking about; it's nothing short of spectacular. In front of the hotel is a pond that I would guess covers about three acres of ground. In the pond are hundreds of water jets that are synchronized to music. The jets move and modulate the water in seemingly magical ways.

While my friends and the thousands gathered on the sidewalk that evening were in awe of the beauty of it all, the only thing I could think about was the control system required to create such an incredible display. Honestly, I think I was salivating – it was just that awesome.

On returning to Dallas I got an e-mail from a customer in California. He was looking to make a programmable sequencer with the BASIC Stamp. With the thoughts of the fountain fresh in my mind, I jumped right on it. He was just looking for guidance but I couldn't help myself – I wrote a whole program.

Figure 90.1: Channel Switches and Load Input Button

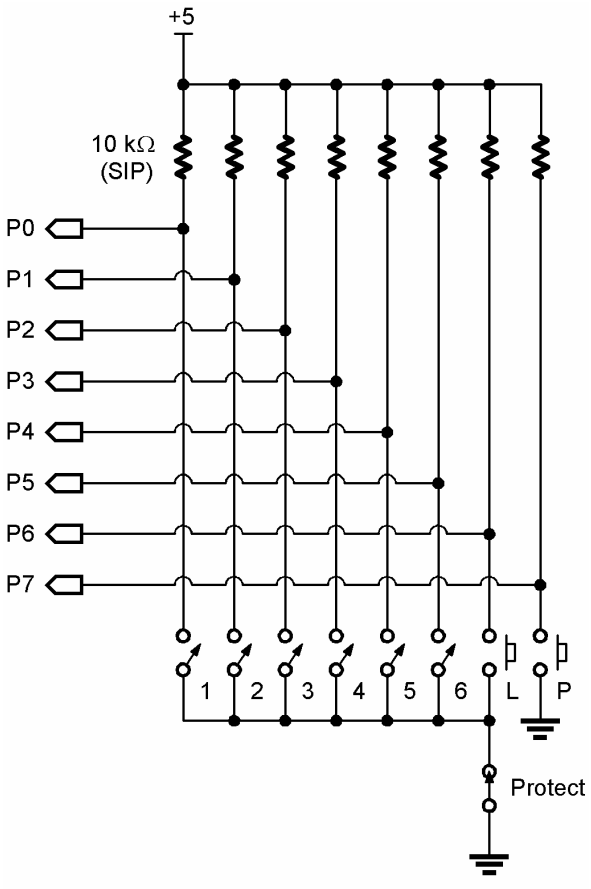
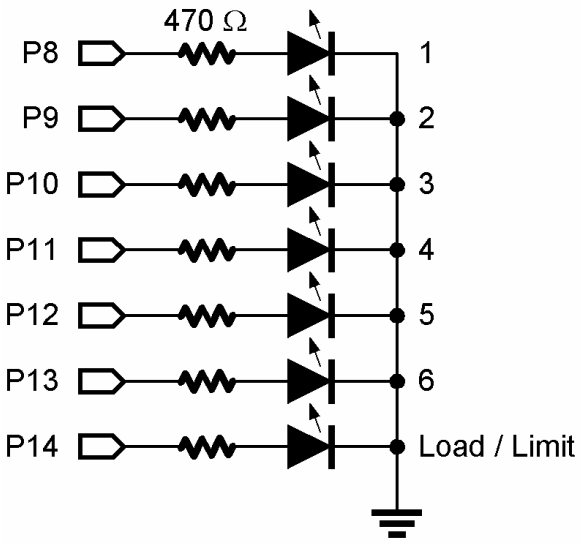


Figure 90.2: LED Circuit



So that's what our project is this month. A programmable sequencer using the BASIC Stamp. My outputs are going to be LEDs (see Figure 90.2). If you want to control something else, you'll need to add appropriate conditioning circuitry to drive your loads. By the way, my new friend in California? ... his outputs are connected to electronically-controlled flame throwers. I can't wait to see that in action!

With Halloween this month and the holiday season right around the corner, this is a useful little project. And for those of you in the film or theater special effects busines, this little controller can be used for anything from lights to flash pots to squibs.

How It Works

The concept is very simple. After reset, we will set the channel input switches for a given step and then press the Load button (For us "old timers" this is reminicent of the way we programmed computers way back when...). If the step is good, the Load/Limit LED will blink. So long as we don't press Play, the sequence will continue to build up to our defined step limit. Once we do press Play, the sequence will be run on our outputs at a rate determined by the StepTime potentiometer.

Pressing Play also resets the steps counter and allows us to reprogram a new sequence. If you want to protect your current sequence, it's a simple matter of breaking the ground connection to the channel switches and Load input button (see Figure 90-1).

Let's go ahead and jump into the details. Once the I/O pins are initialized, the program waits for a Load or Play input and takes action.

```
Main:
  GOSUB Scan_Inputs                             ' check switches &
buttons
  IF (play = Yes) THEN Play_Back                ' play current sequence
  IF (load = No) THEN Main                      ' nothing to do, scan
again
```

Most of the time we're going to be waiting for an input. Let's take a look at the code than actually handles the button and switch inputs.

```
Scan_Inputs:
  swInputs = %11111111                          ' assume all are active
  FOR idx = 1 TO 10
    swInputs = swInputs & ~Inputs               ' get current inputs
    PAUSE 5                                     ' delay between readings
  NEXT

Check_Channel_Count:
  numChans = 0
  FOR idx = 0 TO 5
    numChans = numChans + swInputs.LowBit(idx)  ' add channel value
  NEXT
  IF (numChans <= MaxChannels) THEN Scan_Exit   ' count okay
  swInputs = swInputs & %10000000               ' mask out channel inputs

Scan_Exit:
  RETURN
```

Most of this code should look familiar as we've used it in the past. The first part of the subroutine scans a debounces the input switches and buttons. This is done by starting with the assumption that all are active, then using a loop to AND the current inputs with our result value. Since the inputs are active low, the inversion operator (~) is used. If at any point during the loop a button or switch opens (bounces), that bit will become zero in the ***swInputs*** variable. Since zero ANDed with anything is zero, the channel will be excluded from the current scan. We can fine tune the debounce performance by changing the number of loops and the delay time between each.

The second section of the subroutine is used to control the number of simultaneous outputs when we playback the sequence. On some devices, we may not want to allow all six output channels to be on at the same time. This bit of code counts the switch bits that are set by taking advantage of the LowBit modifier. LowBit allows us to specify an index so we can cycle through the switch bits with our loop control variable, ***idx***. If the number is less than or equal to the MaxChannels limit, the scan will be allowed. If there are too many inputs, all switch channels are cleared – as if they'd never been closed. The Load input is also masked out so that we don't create a blank spot in our sequence. The Play button is still available for the scan.

When a valid input is accepted and it was not the play button, the new step will be recorded.

```
Check_Limit:
  IF (numSteps < maxSteps) THEN Record_Input    ' room for this step?
  LoadLED = On                                  '  - no Load LED on solid
  GOTO Main

Record_Input:
  numSteps = numSteps + 1                       ' update count
  WRITE SeqLen, numSteps                        ' write events count to
EEPROM
  swInputs = swInputs & %00111111               ' mask out Load & Play
buttons
  WRITE (Sequence + numSteps - 1), swInputs     ' write step data to
EEPROM

  LoadLED = On                                  ' show good load
  PAUSE 50
  LoadLED = Off
```

Before we record a step, we need to make sure we haven't hit the step limit. If we have, the Load/Limit LED will be lit solid program will cycle back to main to wait for another input. This will let us detect the Play button.

If can record a new step, the events counter is incremented and saved to the EEPROM. The input channels are also saved to EEPROM. The reason for this is that we can have longer sequences than could be stored in RAM and we won't loose them on reset. The Load/Limit LED will blink to let us know that the step was accepted.

With the current step recorded, we need to make sure that the Load button is released before looking for another step. This will prevent us from filling our step memory with the same data, no matter how long we hold the Load button.

```
Wait_For_Clear:
  GOSUB Scan_Inputs
  IF (Load = Yes) THEN Wait_For_Clear           ' force release of Load
button
  GOTO Main
```

There's no magic here – we simply scan the inputs again and check the Load button. If we want to create a longer section with the same channel data, we'll just press and release that Load button multiple times. There is no blanking between steps unless we deliberately insert it.

Okay, now that we have our steps in memory, we can play them back. Pressing the Play button runs this bit of code:

```
Play_Back:
  READ SeqLen, numSteps                         ' get length of sequence
  IF (numSteps = 0) THEN Main                   ' nothing to play back
  FOR idx = 0 TO (numSteps - 1)
    READ (Sequence + idx), LEDs                 ' put step data on LEDs
    GOSUB Play_Delay                            ' delay between steps
  NEXT

  GOTO Initialize                               ' reset everything
```

We start by reading the length of the sequence from EEPROM. This lets us run a previously-stored sequence after a reset or power-up. Then it's just a matter of reading the events table to the outputs and inserting our step delay. At the end of the sequence we jump back to the initialization code where the outputs and step counter are cleard. We can create a repeating sequence by holding the Play button or placing a SPST switch in parallel with it.

The delay between steps during playback is controlled by a potentiometer and read using RCTIME.

```
Play_Delay:
  HIGH StepTime                                 ' discharge RCTIME cap
  PAUSE 1
  RCTIME StepTime, 1, stepDelay                 ' read pot value
  stepDelay = (stepDelay */ $013C) + 100        ' convert to 100 - 1000
  PAUSE stepDelay
  RETURN
```

Figure 90.3: Standard "RCTime" Circuit

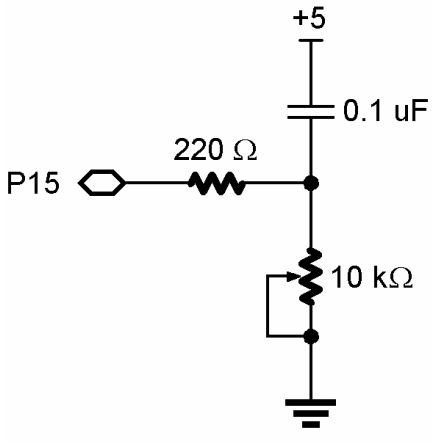


Figure 90.3 shows a standard RCTIME circuit. When using RCTIME, we start by discharging the capacitor. Since the other end of the capacitor is tied to Vdd, we'll make the StepTime pot pin high. With a 0.1 uF cap and a 220 ohm resistor, it doesn't take long to discharge the capacitor, so the one millisecond PAUSE is plenty of time to get the job done.

The pot position is read using **RCTIME** and the raw value is placed in ***stepDelay***. On my prototype, the maximum **RCTIME** value returned was 728. What I wanted to have was a step delay that was minimum of 100 milliseconds and maximum of one second (1000 milliseconds). What I need to do is scale 728 to 900. I needed a scaling factor for the ***stepDelay*** value. This is done by dividing 900 by 728 – the result is 1.236.

Now we all know that the BASIC Stamp doesn't do floating-point math, but it does have that tricky */ (star-slash) operator. The */ operator multiplies by a fractional value expressed in units of 1/256. To find our */ parameter, then, we simply multiply our fractional value by 256. In our case the result is 316 (1.236 x 256). My habit is to convert this value to hex ($013C) because the upper byte represents the whole part of the value. The Stamp doesn't care and will happily work with any numeric format you choose. In our delay conversion code, we add 100 to guarantee a minimum step delay when the pot is turned all the way down to zero. **PAUSE** takes care of creating the delay.

Taking It Further

This project is really simple, yet incredibly useful and an excellent candidate for further development. Here are a few ideas that you could incorporate to stretch the project and your Stamp programming skills.

By using our old favorites the 74HC165 and 75HC595, we could monitor eight inputs and control eight outputs using only five Stamp pins. This frees a lot of Stamp I/O.

With enough free I/O pins, you could add an LCD display and a couple of control buttons (i.e., Step Forward, Step Back). This would be especially useful for long sequences and making adjustments without reprogramming the whole series.

Need really long sequences? Add an I2C memory like we did with the data logger project.

Want to get really fancy? Read the pot when you press Load and store the timing with the step data.

I hope you find this project as fun and useful as I did; it really can be adapted for a wide variety of applications. Please let me know if you do something interesting with it.

Well, that's enough for this month. Have a safe, sane and happy Halloween. Until next time, Happy Stamping.

```
' =============================================================================
'
'   Program Listing 90.1
'   File...... Sequencer.BS2
'   Purpose... Records and plays back a button sequence
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started... 21 AUG 2002
'   Updated... 30 AUG 2002
'
'   {$STAMP BS2}
'
' =============================================================================


' -----------------------------------------------------------------------------
' Program Description
' -----------------------------------------------------------------------------
'
' This program allows the user to write a sequence to memory.  The outputs for
' each step in the sequence with SPST switches, then stored in memory with a
' press of the Load button.  When play is pressed, the sequence will be re-
' played on the output LEDs.
'
' The Load/Limit LED will blink on the acceptance of new step and will stay
' on solid if the sequence limit is reached.
'
' Connections:
'
' Channel 1     P0      SPST                    ' inputs are active low
' Channel 2     P1      SPST
' Channel 3     P2      SPST
' Channel 4     P3      SPST
' Channel 5     P4      SPST
' Channel 6     P5      SPST
' Load          P6      Push button             ' load new step data
' Play          P7      Push button             ' play sequence
'
' Output 1      P8                              ' outputs are active high
' Output 2      P9
' Output 3      P10
' Output 4      P11
' Output 5      P12
' Output 6      P13

' Load/Limit    P14                             ' step loaded (blink) / limit
' Step Time     P15                             ' POT input for step timing


' -----------------------------------------------------------------------------
```

```
' Revision History
' -----------------------------------------------------------------------------


' -----------------------------------------------------------------------------
' I/O Definitions
' -----------------------------------------------------------------------------

Inputs          VAR     InL                     ' switches and buttons
LEDs            VAR     OutH                    ' channel LEDs

LoadLED         VAR     Out14                   ' shows step loaded / hit limit
StepTime        CON     15                      ' RCTIME input for play speed


' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

Yes             CON     1
No              CON     0

On              CON     1
Off             CON     0

MaxChannels     CON     6                       ' max simultaneous outputs
maxSteps        CON     30                      ' max events in sequence
                                                '  - absolute maximum is 255

' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------

swInputs        VAR     Byte                    ' switches/button inputs
load            VAR     swInputs.Bit6           ' isolate load button
play            VAR     swInputs.Bit7           ' isolate play button
idx             VAR     Byte                    ' loop counter
numSteps        VAR     Byte                    ' steps in sequence
numChans        VAR     Nib                     ' channels in step
stepDelay       VAR     Word                    ' timer for step playback


' -----------------------------------------------------------------------------
' EEPROM Data
' -----------------------------------------------------------------------------

SeqLen          DATA    0                       ' events in sequence
Sequence        DATA    0 (255)                 ' sequence data


' -----------------------------------------------------------------------------
```

```
' Initialization
' -----------------------------------------------------------------------------

Initialize:
  DirL = %00000000                              ' switches & buttons are inputs
  DirH = %01111111                              ' LEDs are outputs

  numSteps = 0                                  ' reset steps counter
  LEDs = 0                                      ' clear outputs


' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------

Main:
  GOSUB Scan Inputs                             ' check switches & buttons
  IF (play = Yes) THEN Play_Back                ' play current sequence
  IF (load = No) THEN Main                      ' nothing to do, scan again

Check Limit:
  IF (numSteps < maxSteps) THEN Record Input    ' room for this step?
  LoadLED = On                                  '  - no Load LED on solid
  GOTO Main

Record Input:
  numSteps = numSteps + 1                       ' update count
  WRITE SeqLen, numSteps                        ' write events count to EEPROM
  swInputs = swInputs & %00111111               ' mask out Load & Play buttons
  WRITE (Sequence + numSteps - 1), swInputs     ' write step data to EEPROM

  LoadLED = On                                  ' show good load
  PAUSE 50
  LoadLED = Off

Wait_For_Clear:
  GOSUB Scan_Inputs
  IF (Load = Yes) THEN Wait For Clear           ' force release of Load button
  GOTO Main

Play Back:
  READ SeqLen, numSteps                         ' get length of sequence
  IF (numSteps = 0) THEN Main                   ' nothing to play back
  FOR idx = 0 TO (numSteps - 1)
    READ (Sequence + idx), LEDs                 ' put step data on LEDs
    GOSUB Play Delay                            ' delay between steps
  NEXT

  GOTO Initialize                               ' reset everything
```

```
' -----------------------------------------------------------------------------
' Subroutines
' -----------------------------------------------------------------------------

Scan Inputs:
  swInputs = %11111111                          ' assume all are active
  FOR idx = 1 TO 10
    swInputs = swInputs & ~Inputs               ' get current inputs
    PAUSE 5                                     ' delay between readings
  NEXT

Check_Channel_Count:
  numChans = 0
  FOR idx = 0 TO 5
    numChans = numChans + swInputs.LowBit(idx)  ' add channel value
  NEXT
  IF (numChans <= MaxChannels) THEN Scan Exit   ' count okay
  swInputs = swInputs & %11000000               ' mask out channel inputs

Scan Exit:
  RETURN


Play_Delay:
  HIGH StepTime                                 ' discharge RCTIME cap
  PAUSE 1
  RCTIME StepTime, 1, stepDelay                 ' read pot value
  stepDelay = (stepDelay */ $013C) + 100        ' convert to 100 - 1000
  PAUSE stepDelay
  RETURN
```