Column #145, September 2007 by Jon Williams:

Where's Waldo?

It seems like the animation controller from May was a hit. I got a lot of very positive e-mail and many readers have been creating derivative applications for controlling servos. A few weeks ago I got a call from one reader, my friend Dan, who works in one Hollywood's leading prop shops (if you saw the movie "300" then you've seen some of their amazing creature work). Dan suggested that I build a recording servo controller because – in Hollywood prop shops, anyway – the live performance is what really counts and the ability to play back a great performance is a convenience. This kind of device would also be cool for an animated Halloween or Christmas prop, something I have a lot of interest in.

Dan's boss, Mark, allowed me to visit the shop last year and I was like a kid in a candy store – I love special FX and prop technology and was in absolute awe of everything in their shop; I think I had to run to the parking meter three or four times because I just didn't want to leave! Most of the props I got to see (and play with!) were articulated, a few with cables that are attached to a control device called a "waldo" (Author's note: WALDO is a trademark of The Creature Shop, and yet the term is used somewhat generically in the business due to its roots in a Robert Heinlein short story). The waldo allows extraordinarily-skilled puppeteers to bring life to a prop while a scene is being shot. One of the cable-controlled props that I was allowed to play with is the cat from "Date Movie" which, I must say, was much more fun that having a kitten chase a flashlight beam. Other props use servos to control the motion, many being driven by standard RC transmitters – the problem with using an RC transmitter is that the movement values can't be recorded.

Even if you don't want or need a device like this I think that you'll find a lot of the code useful for other projects. In short, what this device will do is take the input from a standard analog joystick and send position values to a couple servos. By pressing a button with a record jumper enabled, the device will store the joystick movements into a 24LC512 (64K) EEPROM; press the button again and the record cycle stops; remove the jumper and the cycle that was just recorded can be played back.

I was really pressed for time this week so I decided to use the SX28 proto board again. It turns out to be a really good choice because there are features of the proto board that are perfect for this application. Specifically, there are two major sub-systems for the waldo: 1) analog-to-digital conversion of joystick data and, 2) storage of servo values in an EEPROM. It turns out that I'm using DIP eight devices: the ADC0832 for analog conversion and the 24LC512 for the memory – and the SX28 proto board includes two DIP8 socket locations in the prototyping area. In Figure 145.1 the red boxes (adjacent boxes to the left) outline the DIP8 locations; note, too, that inside the DIP8 pads are pads for SOIC8 devices.

Figure 145.1: Animatronic Controller Schematic, Page 1

Another – far more subtle – feature of the SX28 proto board is voltage level selection and control for the servo headers (blue box, top center, enclosing 12 holes). Note the purple box

(top right, enclosing three holes); if we solder a male, 3-pin header to this area we can select the voltage applied to the center terminal of the servo headers, either 5v (from the regulator) or the Vin from the power connector (be careful with using Vin, anything above 7.2 volts will probably destroy your servos). The center pin of voltage select header is routed through the power switch to the servo headers so that we can power up the SX without applying power to the servos. When we want the servos to run we move the power switch to the far-right position. These features shouldn't be too surprising as they make using the SX28 proto board on the Parallax Boe-Bot chassis very easy.

Waldo Subsystems

The first major subsystem is the analog-to-digital conversion of the joystick input. Part of the parameters for this project was to use an off-the-shelf, PC-compatible analog joystick. The trouble with analog PC joysticks is that we only get two connections per pot; we don't get the ends plus the wiper – we get one end (common, on pin 1) and the wipers from each axis. What we're forced to do, then, is add a second resistor to create a divider; we'll tap off that divider and run it into an input channel of the ADC0832.

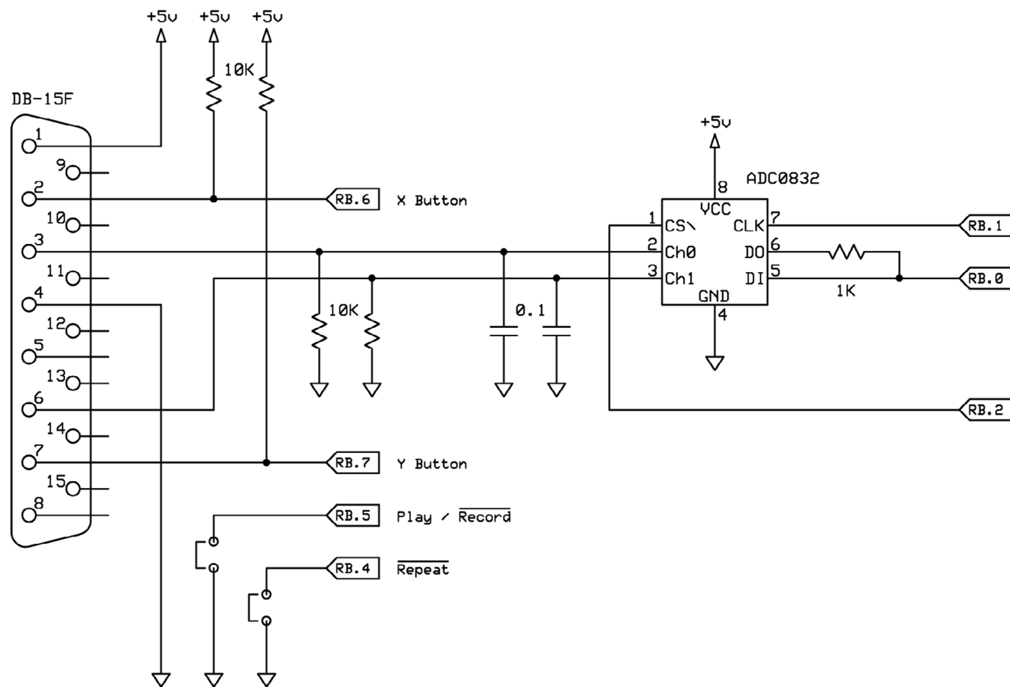

Figure 145.2: Joystick Interface

Figure 145.2 shows the joystick interface for the circuit. As you can see, the pot outputs from the joystick (pins 3 and 6) are connected to 10K pull-downs and then on to the ADC0832. The application of 5v to pin 1 of the joystick connector routes this voltage to the pots. While this arrangement works, we don't get a centered output value when the pot is in its center position; I don't like "fixing" circuits in code, but this is a case where there is just no choice.

Let's have a look at the ADC code. The ADC0832 has two inputs and it can be configured on-the-fly to provide singled-ended or differential output. Configuration of the ADC is accomplished by clocking four (mux) bits into the device. As you can see in the function code, \4 is used to limit the number of bits sent to the ADC. For convenience, the waldo program has constants defined for single-ended channels 0 and 1 that include the ADC start bit. After shifting the mux bits out to the ADC, an eight-bit value can be shifted in.

```
FUNC ADC0832
  tmpB1 = __PARAM1
  Clk = 0
  CS = 0
  SHIFTOUT Dio, Clk, MSBFIRST, tmpB1\4, 4
  SHIFTIN  Dio, Clk, MSBPOST, tmpB1, 4
  CS = 1
  RETURN tmpB1
  ENDFUNC
```

Let me point out a couple things having to do with the use of SHIFTOUT and SHIFTIN. First, both of these instructions simply invert the clock pin (twice) for each bit; what this means is that we need to preset the clock level before using SHIFTOUT or SHIFTIN; neglecting to do this is a common mistake for those porting BASIC Stamp programs to SX/B. And later versions of SX/B include an optional clock speed multiplier. Without the multiplier the data rate is about 83K bits per second, but many devices will operate significantly faster than that so we can take advantage by using the multiplier. Another point of consideration for this particular project is that a "virtual" servo controller is running in the Interrupt Service Routine so all foreground operations are slowed – the multiplier restores the access speed to the ADC.

The other major subsystem is the I2C memory. I selected the 24LC512 because it holds 64K bytes of data and is really simple to use. Figure 145.3 shows the connections to the 24LC512.

When you look through the complete listing you'll see a lot of code having to do with the EEPROM. The reason is that there are four instructions in SX/B (I2CSTART, I2CSTOP, I2CSEND, and I2CRECV) that get encapsulated in subroutines or functions to save code space. I've also created a subroutine that lets us write a byte or word to the EEPROM, and two functions for reading: one for bytes, the other for words. All of this code is highly

portable and you can use it in many applications; the only critical note is that the SCL pin is aligned with the SDA pin, i.e. the SCL pin always follows the SDA pin on the same port (RA, RB, RC, RD, or RE). For example, if SDA is RA.2, then we must connect SCL to RA.3.

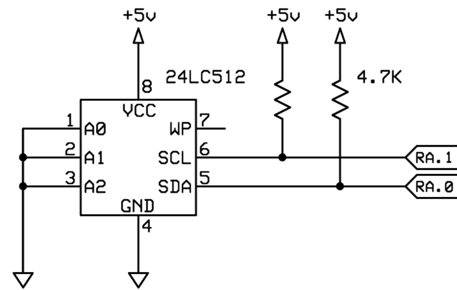

Figure 145.3: 24LC512 Connections

Since writing to the EEPROM is a critical task for this program let's have a look at the PUT_EE function. If you compare the code to the 24LC512 data sheet you should see that it's an easy match – my point is that once you've got the I2C routines setup, access to any I2C device is very straightforward.

```
SUB PUT_EE
  IF __PARAMCNT = 3 THEN
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    i2cMulti = 0
  ELSE
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    tmpB2 = __PARAM4
    i2cMulti = 1
  ENDIF
  I2C_START
  I2C_OUT SlaveWr
  I2C_OUT tmpW1_MSB
  I2C_OUT tmpW1_LSB
  I2C_OUT tmpB1
  IF i2cMulti = 1 THEN
    I2C_OUT tmpB2
  ENDIF
  I2C_STOP
  '{$IFNOTDEF NOEEWAIT}
  DO
    I2C_START
    I2C_OUT SlaveWr
```

```
  LOOP UNTIL ackNak = Ack
  '{$ENDIF}
  ENDSUB
```

This subroutine expects a word address and then one or two bytes after that. Since we want to use the same subroutine to write bytes and words we're forced into accepting the address as two bytes. If we're using a word variable for the EE address the compiler will sort that out for us, but when using a constant value we have to be careful. Let's say, for example, that we want to write a value to EEPROM address $000F. Here's how we have to do that when using constants:

```
  PUT_EE $0F, $00, value
```

As you can see, we're using two bytes for the address and the bytes are aligned Little-Endian. If we did this:

```
  PUT_EE $000F, value
```

… the compiler wouldn't understand that the address is a word since the address value is less than 256 – so a byte is assumed and used. And if the value to be written is also a byte the compiler will complain that we don't have enough parameters for the subroutine. By forcing a two-byte address the subroutine can determine whether we want to write a byte (__PARAMCNT is three) or a word (__PARAMCNT is four). When we are writing two bytes a flag is set that gets used later in the routine.

The subroutine might look a little complicated but it is in fact very straightforward. We start with the I2C start sequence, write the device address (a constant in this program, but could be a variable if you want to expand to multiple EEPROMs), the address to write to, and then the byte(s) to write. As you can see, we use the flag to control writing the second byte. Finally, the I2C stop sequence is generated to tell the EEPROM to save its buffer contents.

EEPROMs are not particularly fast and this device can take up to five milliseconds to store the values we just sent to it. We don't care about this delay because we'll only access the EEPROM every 20 milliseconds, but in other applications we can have the subroutine wait for the write cycle to complete before returning to the caller. We do this by generating another start, and then writing the slave address. While the EEPROM is busy with its write cycle the acknowledge bit will be set to NAK (1). A conditional-compilation constant allows us to enable or disable the write-wait option.

Putting It Together
Like the animation controller project in May, this program uses a "virtual" servo controller that runs in the ISR; since we've been through that in detail we won't hash through it again. The only thing that's been added to the ISR is an LED control option; the reason is that the program has four modes:

1) Idle (servos follow joystick)
2) Recoding (servos follow joystick and positions are saved to EEPROM)
3) Playback (servo positions are played back from the EEPROM)
4) Paused (servos hold position until pause button pressed again)

I started with a 3-leaded, bi-color LED – one of those LEDs that have red and green elements with a common leg. The problem was that the body was clear and when both LEDs were on it didn't really look yellow as I had hoped. Well, in my supplies I found a 2-leaded, bi-color LED that had a milky, translucent body which I thought might work better. It did, but it takes a little more code to create the yellow effect.

This is accomplished reversing current flow through the LED very quickly. Since the interrupt runs 100,000 times per second we can do it there. Here's how:

```
Check_Yellow_LED:
  IF runMode = M_PAUSE THEN
    ledPort = ledPort ^ %1100
  ENDIF
```

What this does is invert the state of the LEDs each pass through the ISR when the program is paused. Figure 145.4 shows how the LED is connected. There is only one resistor because current can only flow through one LED at a time.

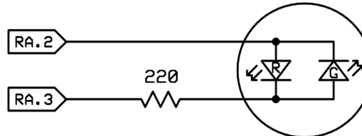

Figure 145.4: LED Connection

Okay, we can read the joystick and save values, so let's get to the meat of the program. At the top we're going to wait for a new servo frame (the ISR sets a flag bit that we'll wait on). When we get the new frame we read the joystick axis values.

```
Main:
  WAIT_SYNC
  joyX = ADC0832 Ch0
  joyY = ADC0832 Ch1
```

Since the output from the ADC doesn't match what we need to drive the servos we'll have to apply a little math to adjust things.

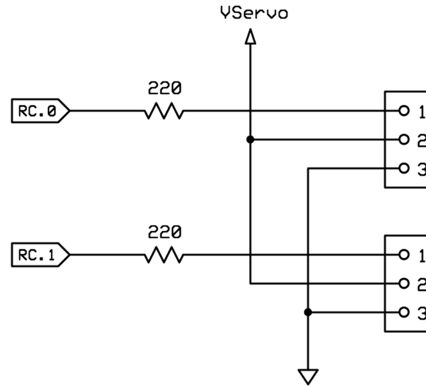

Figure 145.5: Servo Connections

Let me take a bit of a detour here and encourage you to develop your programs in sections and with separate test programs. I did this will all of the systems of the waldo program and have included my test programs for you to work with should you choose. When testing the joystick I got values between 25 and 255, and what we need for the servo is 100 to 200. The challenge doesn't end there, however, as the center value for the joysticks was 50.

So we have two sections: one side goes 25 to 50, the other side of the stick goes 50 to 255. To adjust the first section for servo pulse values the math is pretty simple:

position = position * 2 + 50

By applying this formula we'll move 25-to-50 to 100-to-150. For the other side we get lucky again and the formula works out to:

position = position / 4 + 137

I say that we got lucky because the multiplier and divider in the formulas above are powers of two; this lets us use shift operators instead of multiplication and division operators (both generate a fair bit of code).

Here's the adjustment section:

```
  IF joyX <= 50 THEN
    joyX = joyX << 1
    joyX = joyX + 50
  ELSE
    joyX = joyX >> 2
    joyX = joyX + 137
  ENDIF
  joyX = 300 - joyX
  IF joyY <= 50 THEN
    joyY = joyY << 1
    joyY = joyY + 50
  ELSE
    joyY = joyY >> 2
    joyY = joyY + 137
  ENDIF
  joyY = 300 - joyY
```

Once the program was running I found that the servos moved opposite the actual joystick movement so both axis values had to be inverted (100 becomes 200, and vice-versa). This is easy math, too – just subtract the axis value from 300. Note that the compiler will complain about a truncated literal. This happens because we're using a value greater than 255 in the equation and the output is a byte; it's okay, the result will still be correct.

The joystick has two buttons; one for each axis. We can scan and debounce the buttons like this:

```
Check_Buttons:
  IF BtnX = Pressed THEN
    INC btnTmr(0)
  ELSE
    btnTmr(0) = 0
  ENDIF
  IF BtnY = Pressed THEN
    INC btnTmr(1)
  ELSE
    btnTmr(1) = 0
  ENDIF
```

Each button has its own debounce counter; when either counter reaches five (which means the button was held down for five consecutive cycles) it is considered valid. How a button press

affects the program is based on the current operational mode. And the next step in the program is to jump to the handler for the current mode:

```
Mode_Handler:
  IF runMode = M_IDLE THEN Check_Start
  IF runMode = M_REC THEN Recording
  IF runMode = M_PLAY THEN Playing
  IF runMode = M_PAUSE THEN Play_Paused
```

The first mode, M_IDLE, is where we'll sit and wait for a start button press, all the while the servos will follow any motion of the joystick. When the start (X axis) button is pressed we'll look at the Play/Record jumper and take things from there.

```
  IF btnTmr(0) = BtnOK THEN
    IF PlayRec = RecordNow THEN
      numRecs = 0
      runMode = M_REC
    ELSE
      numRecs = GET_EE2 0, 0
      IF numRecs > REC_LAST THEN
        GOTO Empty_EE
      ELSE
        runMode = M_PLAY
      ENDIF
    ENDIF
    eePntr = 2
    btnTmr(0) = 0
  ENDIF
  GOTO Main
```

When the Play/Record jumper is out the number of records is read from the EEPROM at address $0000. Blank EEPROMs usually have all locations set to $FF, so if we see a bad value in the numRecs variable, the program flashes the red LED and then jumps back to the top in idle mode – what this means is that we need to record some movements first.

Install the Play/Record jumper and then press the start/stop button on the joystick – press it quickly, though. The [movement] records count will be cleared and the mode set to M_REC which directs the program to the section below.

```
Recording:
  ledPort = LED_RED
  pos0 = joyX
  pos1 = joyY
  IF btnTmr(0) = BtnOK THEN
    GOTO Stop_Recording
  ELSE
```

```
    PUT_EE eePntr, joySticks
    INC numRecs
    IF eePntr < REC_LAST THEN
      eePntr = eePntr + 2
    ELSE
      GOTO Stop_Recording
    ENDIF
  ENDIF
  GOTO Main

Stop_Recording:
  PUT_EE 0, 0, numRecs
  runMode = M_IDLE
  btnTmr(0) = 0
  GOTO Main
```

We start by refreshing the red LED to indicate record mode and then update the background servo controller with the current joystick values. If the start/stop button has not been pressed then the axis values are written to the memory, the records count updated, and then we check to see if there's any room left in the EEPROM. If the start/stop button has been pressed or we have run out of memory the recording process is stopped. Here we write the number of records to address $0000 of the EEPROM and reset the mode to idle.

Remove the Play/Record jumper and press the start/stop button again – you should see the moves you just recorded played back. I'm easily entertained, but when this worked the first time my face lit up with a very big smile. Here's the playback code:

```
Playing:
  ledPort = LED_GRN
  pos0 = GET_EE2 eePntr
  pos1 = __PARAM2
  DEC numRecs
  IF btnTmr(0) = BtnOK THEN
    runMode = M_IDLE
    btnTmr(0) = 0
    GOTO Main
  ENDIF
  IF btnTmr(1) = BtnOK THEN
    runMode = M_PAUSE
    btnTmr(1) = 0
    GOTO Main
  ENDIF
  IF numRecs > 0 THEN
    eePntr = eePntr + 2
  ELSE
    IF Repeat = Yes THEN
      numRecs = GET_EE2 0, 0
      eePntr = 2
```

```
    ELSE
      runMode = M_IDLE
    ENDIF
  ENDIF
  GOTO Main
```

In playback mode the green LED is lit and we pull a new set of servo values from the EEPROM. Note that the GET_EE2 function returns a word, but we're placing the result in a byte (pos0). The compiler is smart enough to put the LSB of the returned value into pos0; we can get the MSB of the return manually by copying __PARAM2. When a function returns a word, the LSB is in __PARAM1, the MSB is in __PARAM2.

After retrieving a movement record we decrement the records count and check for button presses. We can stop the playback cycle or, by pressing the y-axis button, put it into a pause mode. Assuming no button presses we check to see if the playback cycle is complete; if not then the EE pointer is advanced to the next set of position values. When we do get to the end, the program checks the loop-back jumper; when this is installed the records count is reloaded, the EE pointer sent back to the beginning, and the playback continues. With no loop-back jumper the mode is returned to idle and playback stops.

The pause mode doesn't do anything except monitor button presses; we can either resume the cycle by pressing the y-axis button again, or stop it by pressing the x-axis button.

```
Play_Paused:
  IF btnTmr(1) = BtnOK THEN
    runMode = M_PLAY
    btnTmr(1) = 0
    GOTO Main
  ENDIF
  IF btnTmr(0) = BtnOK THEN
    runMode = M_IDLE
    btnTmr(0) = 0
  ENDIF
  GOTO Main
```

And there we have it – a two-channel servo controller that plays live, records, and plays back. Figure 146.6 shows my final board. The six-pin header on the right edge is where I connected my DB-15 joystick adapter (using 0.025" post-header sockets). Note that the SX28 proto board has busses for Vdd and Vss so connecting the pull-ups and pull-downs for the joystick connection is very easy at this point.

Connections beneath the board are made with wire-wrapping wire, except for the power connections to the joysticks. And here's a bit of a tip for your toolbox: get a roll of blue

painter's tape. It's great for holding wires and components while soldering, and won't leave a sticky mess when you pull it off – I always have it on hand when I'm building circuit boards.

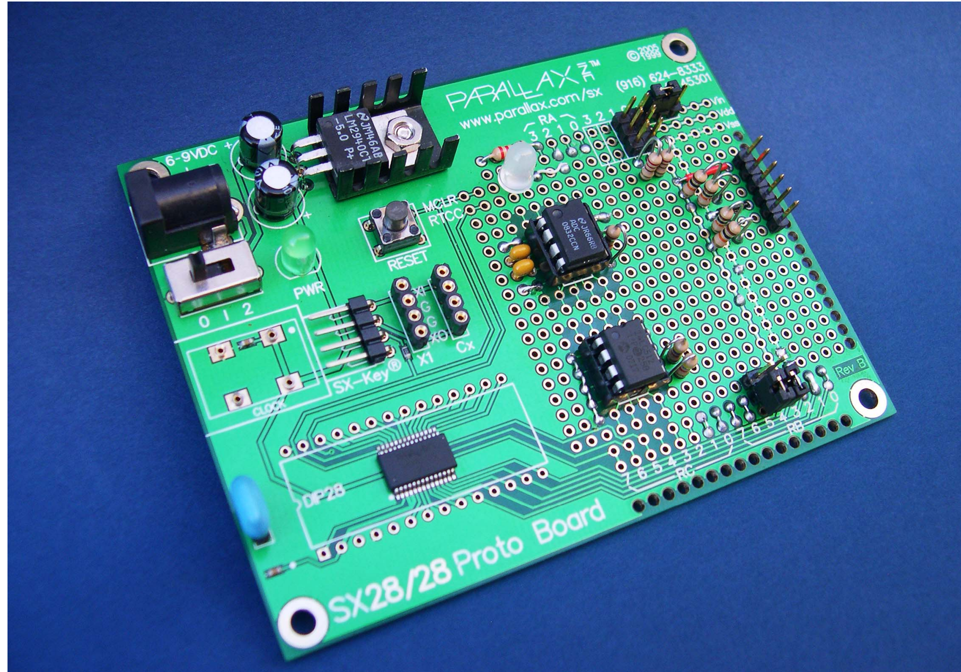

Figure 145.6: Completed Board

The board does work as well as I'd hoped, but one thing I will do is order an LTC1298 and replace the ADC0832 with it; I didn't happen to have an LTC1298 in my supply so I went with what I had. It's a little pricey, about $11, but I think the additional resolution will be worth having, and the pin-out exactly matches the ADC0832. Next up for me is to attach a neat little pan-tilt servo head that I bought from Lynxmotion and make it dance!

Have fun, and until next time, Happy Stamping!

Parts List	
Quantity	**Value**
1	Parallax SX28 Proto Board
1	ADC0832 dual channel A to D converter
1	24LC512 EEPROM
2	8-pin DIP socket
1	DB-15F socket (solder cup type)
3	220-ohm resistor
1	1K resistor
2	4.7K resistor
4	10K resistor
2	0.1 uF capacitor
1	bi-color LED (2 lead)
0.025 male post headers (0.1" centers)	
post-header jumpers	

Source Code

```
' =========================================================================
'
'   File...... 24LC512_Test.SXB
'   Purpose... (for Waldo project)
'   Author.... Jon Williams, EFX-TEK
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated...
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' Conditional Compilation Symbols
' -------------------------------------------------------------------------

'{$DEFINE NOEEWAIT_OFF}


' -------------------------------------------------------------------------
' Device Settings
' -------------------------------------------------------------------------
```

```
DEVICE          SX28, OSCHS2, TURBO, STACKX, OPTIONX, BOR42
FREQ            20_000_000
ID              "24LC512"


' -------------------------------------------------------------------------
' IO Pins
' -------------------------------------------------------------------------

UnusedRC       PIN     RC   INPUT PULLUP
UnusedRB       PIN     RB   INPUT PULLUP
UnusedRA        PIN     RA   INPUT PULLUP

SDA             PIN     RA.0 INPUT  NOPULLUP        ' EE data line (I2C)
SCL             PIN     RA.1 OUTPUT NOPULLUP        ' EE clock line (I2C)


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

SlaveWr         CON     $A0
SlaveRd         CON     $A1
Ack             CON     0
Nak             CON     1

EE_LAST        CON     $FFFE


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

flags           VAR     Byte
ackNak          VAR     flags.0                 ' for I2C routines
i2cMulti       VAR     flags.1                 ' for I2C write

eeWord         VAR     Word
eeByteLo       VAR     eeWord_LSB
eeByteHi       VAR     eeWord_MSB


tmpB1          VAR     Byte                    ' for subs/funcs
tmpB2          VAR     Byte
tmpB3          VAR     Byte
tmpW1          VAR     Word


' -------------------------------------------------------------------------
' INTERRUPT
```

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

' RETURNINT


' =========================================================================
  PROGRAM Start
' =========================================================================


' -------------------------------------------------------------------------
' Subroutine Declarations
' -------------------------------------------------------------------------

PUT_EE         SUB     3, 4                    ' write byte(s) to EEPROM

GET_EE         FUNC    1, 2                     ' read byte from EEPROM
GET_EE2        FUNC    2, 2                     ' read word from EEPROM

' I2C routines used by subs/funcs above

I2C_START       SUB     0                       ' generate I2C Start
I2C_STOP        SUB     0                       ' generate I2C Stop
I2C_OUT         SUB     1, 1                    ' write byte to SDA
I2C_IN          FUNC    1, 1                    ' read byte from SDA


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

Start:


Main:
  ' PUT_EE 0, 0, $ABCD
  ' eeByteLo = GET_EE 0, 0
  ' eeByteHi = GET_EE 1, 0

  'PUT_EE 0, 0, $FFFF
  eeWord = GET_EE2 0, 0

  WATCH eeWord, 16, UDEC
  BREAK

  END


' -------------------------------------------------------------------------
' Subroutine Code
' -------------------------------------------------------------------------
```

```
' Use: PUT_EE address, byteVal1 {, byteVal2 }
' -- writes 'byteVal1' to 'address'
' -- writes [optional] 'byteVal2' to 'address'+1
' -- 'address' is a word (0 to 65535)

SUB PUT_EE
  IF __PARAMCNT = 3 THEN                        ' write one byte
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    i2cMulti = 0
  ELSE
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    tmpB2 = __PARAM4
    i2cMulti = 1
  ENDIF

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT tmpW1_MSB                             ' send address, high byte
  I2C_OUT tmpW1_LSB                             ' send address, low byte
  I2C_OUT tmpB1                                 ' send data byte
  IF i2cMulti = 1 THEN
    I2C_OUT tmpB2                               ' send second data byte
  ENDIF
  I2C_STOP                                      ' finish

  '{$IFNDEF NOEEWAIT}
  DO                                            ' let write cycle finish
    I2C_START
    I2C_OUT SlaveWr
  LOOP UNTIL ackNak = Ack
  '{$ENDIF}

  ENDSUB

' -------------------------------------------------------------------------

' Use: byteVal = GET_EE address
' -- reads 'byteVal' from EEPROM location 'address'
' -- 'address' is a word (0 to 65535)

FUNC GET_EE
  tmpW1 = __WPARAM12

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT tmpW1_MSB                             ' send address, high byte
  I2C_OUT tmpW1_LSB                             ' send address, low byte
  I2C_START
```

```
  I2C_OUT SlaveRd
  tmpB1 = I2C_IN Nak
  I2C_STOP
  RETURN tmpB1
  ENDFUNC

' -------------------------------------------------------------------------

' Use: wordVal = GET_EE2 address
' -- reads [Little-Endian] 'wordVal' from EEPROM location 'address'
' -- 'address' is a word (0 to 65535)

FUNC GET_EE2
  tmpW1 = __WPARAM12

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT tmpW1_MSB                             ' send address, high byte
  I2C_OUT tmpW1_LSB                             ' send address, low byte
  I2C_START
  I2C_OUT SlaveRd
  tmpW1_LSB = I2C_IN Ack
  tmpW1_MSB = I2C_IN Nak
  I2C_STOP
  RETURN tmpW1
  ENDFUNC

' -------------------------------------------------------------------------

' Use: I2C_START
' -- generates I2C start condition on SDA/SCL pins

SUB I2C_START
  I2CSTART SDA
  ENDSUB

' -------------------------------------------------------------------------

' Use: I2C_STOP
' -- generates I2C stop condition on SDA/SCL pins

SUB I2C_STOP
  I2CSTOP SDA
  ENDSUB

' -------------------------------------------------------------------------

' Use: I2C_OUT byteVal
' -- writes 'byteVal' to SDA pin
' -- affects global var "ackNak"
' -- EE address pointer must be preset before call
```

```
SUB I2C_OUT
  I2CSEND SDA, __PARAM1, ackNak
  ENDSUB

' -------------------------------------------------------------------------

' Use: byteVal = I2C_IN AckBit
' -- reads 'byteVal' from SDA pin
' -- EE address pointer must be preset before call

FUNC I2C_IN
  ackNak = __PARAM1.0
  I2CRECV SDA, tmpB3, ackNak
  RETURN tmpB3
  ENDFUNC

' =========================================================================
' User Data
' =========================================================================
```

```
' =========================================================================
'
'   File...... LED_Test.SXB
'   Purpose... (for Waldo project)
'   Author.... Jon Williams, EFX-TEK
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated...
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' Conditional Compilation Symbols
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' Device Settings
' -------------------------------------------------------------------------
```

```
DEVICE          SX28, OSCHS2, TURBO, STACKX, OPTIONX, BOR42
FREQ            20_000_000
ID              "Leds"


' -------------------------------------------------------------------------
' I/O Pins
' -------------------------------------------------------------------------

UnusedRC       PIN     RC   INPUT  PULLUP
UnusedRB       PIN     RB   INPUT  PULLUP
UnusedRA       PIN     RA   INPUT  PULLUP


LedPort        PIN     RA
RedLed         PIN     RA.2 OUTPUT NOPULLUP   ' bi-color, 2-lead LED
GrnLed         PIN     RA.3 OUTPUT NOPULLUP


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

LED_RED        CON     %1000
LED_GRN        CON     %0100


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

idx            VAR     Word

tmpB1          VAR     Word                   ' for subs / funcs
tmpW1          VAR     Word


' =========================================================================
'  INTERRUPT
' =========================================================================


'  RETURNINT


' =========================================================================
  PROGRAM Start
' =========================================================================


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

```
' Subroutine / Function Declarations
' -------------------------------------------------------------------------

DELAY_MS        SUB     1, 2                    ' delay in milliseconds
DELAY_US        SUB     1, 2                    ' delay in microseconds


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

Start:


Main:
  LedPort = LED_RED
  DELAY_MS 500

  LedPort = LED_GRN
  DELAY_MS 500

  ' yellow test
  '
  FOR idx = 1 TO 25000
    LedPort = LED_RED
    DELAY_US 10                                     ' simulate ISR timing
    LedPort = LED_GRN
    DELAY_US 10
  NEXT

  GOTO Main


' -------------------------------------------------------------------------
' Subroutine / Function Code
' -------------------------------------------------------------------------

' Use: DELAY_MS mSecs
' -- delays program in milliseconds

SUB DELAY_MS
  IF __PARAMCNT = 1 THEN
    tmpW1 = __PARAM1
  ELSE
    tmpW1 = __WPARAM12
  ENDIF
  PAUSE tmpW1
  ENDSUB

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

```
' Use: DELAY_US uSecs
' -- delays program in microseconds

SUB DELAY_US
  IF __PARAMCNT = 1 THEN
    tmpW1 = __PARAM1
  ELSE
    tmpW1 = __WPARAM12
  ENDIF
  PAUSEUS tmpW1
  ENDSUB


' -------------------------------------------------------------------------
' User Data
' -------------------------------------------------------------------------
```

```
' =========================================================================
'
'   File...... Joystick_Test.SXB
'   Purpose... (for Waldo project)
'   Author.... Jon Williams, EFX-TEK
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated...
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------
'
' References:
' -- http://www.epanorama.net/documents/joystick/pc_joystick.html


' -------------------------------------------------------------------------
' Conditional Compilation Symbols
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' Device Settings
' -------------------------------------------------------------------------

DEVICE          SX28, OSCHS2, TURBO, STACKX, OPTIONX, BOR42
FREQ            20_000_000
ID              "Joystick"
```

```
' -------------------------------------------------------------------------
' I/O Pins
' -------------------------------------------------------------------------

UnusedRC        PIN     RC   INPUT  PULLUP
UnusedRB        PIN     RB   INPUT  PULLUP
UnusedRA        PIN     RA   INPUT  PULLUP

Dio             PIN     RB.0 INPUT  NOPULLUP   ' ADC0832 data (DO + DI)
Clk             PIN     RB.1 OUTPUT NOPULLUP   ' ADC0832 clock
CS              PIN     RB.2 OUTPUT NOPULLUP   ' ADC0832 CS\

BtnX            PIN     RB.6 INPUT  NOPULLUP   ' from joystick
BtnY            PIN     RB.7 INPUT  NOPULLUP   ' from joystick


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

Ch0             CON     %110                   ' ADC0832 mux select
Ch1             CON     %111


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

joyX1           VAR     Byte
joyY1           VAR     Byte
joyX2           VAR     Byte
joyY2           VAR     Byte
btns            VAR     Byte

tmpB1           VAR     Byte
tmpW1           VAR     Word


' =========================================================================
  INTERRUPT 100_000                            ' run every 10 uS
' =========================================================================

  ' just kill a little time

  NOP
  NOP
  NOP
  NOP

  RETURNINT
```

```
' =========================================================================
  PROGRAM Start
' =========================================================================


' -------------------------------------------------------------------------
' Subroutine / Function Declarations
' -------------------------------------------------------------------------

ADC0832        FUNC    1, 1


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

Start:
  CS = 1

Main:
  PAUSE 100

  joyX1 = ADC0832 Ch0
  joyY1 = ADC0832 Ch1

  btns = 0
  btns.0 = BtnX
  btns.1 = BtnY

  ' adjust for 10 uS servo values
  '
  IF joyX1 <= 50 THEN
    joyX2 = joyX1 << 1
    joyX2 = joyX2 + 50
  ELSE
    joyX2 = joyX1 >> 2
    joyX2 = joyX2 + 137
  ENDIF

  joyX2 = 300 - joyX2
  'joyX2 = tmpW1_LSB


  'IF joyY1 <= 50 THEN
  '  joyY2 = joyY1 << 1
  '  joyY2 = joyY2 + 50
  'ELSE
  '  joyY2 = joyY1 >> 2
  '  joyY2 = joyY2 + 137
```

```
  'ENDIF


  ' run in Debug mode; click on [Poll]

  WATCH joyX1, 8, UDEC
  'WATCH joyY1, 8, UDEC
  WATCH joyX2, 8, UDEC
  'WATCH joyY2, 8, UDEC
  WATCH btns, 2, UBIN
  BREAK

  GOTO Main


' -------------------------------------------------------------------------
' Subroutine / Function Code
' -------------------------------------------------------------------------

' Use: bResult = ADC0832 mux
' -- where "mux" are mux selection bits

FUNC ADC0832
  tmpB1 = __PARAM1                              ' save MUX selection
  Clk = 0                                       ' preset clock polarity
  CS = 0                                        ' activate ADC
  SHIFTOUT Dio, Clk, MSBFIRST, tmpB1\4, 4       ' setup MUX
  SHIFTIN  Dio, Clk, MSBPOST, tmpB1, 4          ' read channel data
  CS = 1                                        ' release ADC
  RETURN tmpB1
  ENDFUNC

' -------------------------------------------------------------------------
' User Data
' -------------------------------------------------------------------------
```

```
' =========================================================================
'
'   File...... Waldo.SXB
'   Purpose... 2-channel, recording "waldo" for servo playback
'   Author.... Jon Williams, EFX-TEK
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated... 27 JUL 2007
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
```

```
' -------------------------------------------------------------------------
'
' References:
' -- http://www.epanorama.net/documents/joystick/pc_joystick.html


' -------------------------------------------------------------------------
' Conditional Compilation Symbols
' -------------------------------------------------------------------------

'{$DEFINE NOEEWAIT}                          ' don't wait on EE write


' -------------------------------------------------------------------------
' Device Settings
' -------------------------------------------------------------------------

DEVICE          SX28, OSCHS2, TURBO, STACKX, OPTIONX, BOR42
FREQ            20_000_000
ID              "Waldo"


' -------------------------------------------------------------------------
' IO Pins
' -------------------------------------------------------------------------

UnusedRC       PIN     RC   INPUT PULLUP

ServoCtrl      PIN     RC
Servo0          PIN     RC.0 OUTPUT NOPULLUP ' X servo
Servo1          PIN     RC.1 OUTPUT NOPULLUP ' Y servo

Dio            PIN     RB.0 INPUT            ' ADC0832 data (DO + DI)
Clk            PIN     RB.1 OUTPUT           ' ADC0832 clock
CS             PIN     RB.2 OUTPUT           ' ADC0832 CS\

UnusedRB3      PIN     RB.3 INPUT PULLUP

PlayRec        PIN     RB.4 INPUT PULLUP     ' play/record jumper
Repeat         PIN     RB.5 INPUT PULLUP     ' auto repeat
BtnX           PIN     RB.6 INPUT            ' from joystick
BtnY           PIN     RB.7 INPUT            ' from joystick

LedPort        PIN     RA
RedLed         PIN     RA.2 OUTPUT           ' bi-color, 2-lead LED
GrnLed         PIN     RA.3 OUTPUT

SDA             PIN     RA.0 INPUT           ' EE data line (I2C)
SCL             PIN     RA.1 OUTPUT          ' EE clock line (I2C)
```

```
' -------------------------------------------------------------------------
' Constants
' -------------------------------------------------------------------------

LED_OFF         CON     %0000
LED_RED         CON     %1000
LED_GRN         CON     %0100

Pressed         CON     0                       ' for active-low buttons
NotPressed      CON     1

Yes             CON     0
No              CON     1

RecordNow       CON     0

Ch0             CON     %110                    ' ADC0832 mux select
Ch1             CON     %111

M_IDLE          CON     %000                    ' program modes
M_REC           CON     %001
M_PLAY          CON     %010
M_PAUSE         CON     %110

SlaveWr          CON     $A0                    ' I2C
SlaveRd          CON     $A1
Ack              CON     0
Nak              CON     1

REC_LAST        CON     32767

BtnOK           CON     5                       ' button debounce count


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

flags            VAR     Byte
sync             VAR     flags.0                ' marks start of frame (ISR)
ackNak           VAR     flags.1                ' for I2C routines
i2cMulti        VAR     flags.2                 ' for I2C read / write

tix             VAR     Word                    ' for delays (ISR)

runMode         VAR     Byte
joySticks       VAR     Word
joyX            VAR     joySticks_LSB
joyY            VAR     joySticks_MSB
eePntr          VAR     Word                    ' ee address pointer
numRecs         VAR     Word                    ' records in sequence
```

```
btnTmr          VAR     Byte(2)                 ' for debouncing

tmpB1            VAR     Byte                   ' for subs/funcs
tmpB2            VAR     Byte
tmpB3           VAR     Byte
tmpB4           VAR     Byte
tmpW1            VAR     Word
tmpW2           VAR     Word

svoData         VAR     Byte (8)                ' servo data
pos             VAR     svoData(0)              ' position table
pos0            VAR     svoData(0)
pos1            VAR     svoData(1)
svoFrame_LSB    VAR     svoData(2)              ' frame timer
svoFrame_MSB    VAR     svoData(3)
svoPntr         VAR     SvoData(4)              ' active servo pointer
svoTimer        VAR     svoData(5)              ' pulse timer
svoTemp         VAR     svoData(6)


' -------------------------------------------------------------------------
  INTERRUPT NOPRESERVE 100_000                  ' run every 10 uS
' -------------------------------------------------------------------------

Update_Wait_Timer:
  BANK 0
  IF tix > 0 THEN
    DEC tix
  ENDIF

Check_Yellow_LED:
  IF runMode = M_PAUSE THEN
    ledPort = ledPort ^ %1100                   ' flip red & green
  ENDIF

Check_Frame_Timer:
  ASM
    BANK svoData
    TEST svoFrame_LSB                           ' frame timer done?
    JNZ  Update_Frame_Timer
    TEST svoFrame_MSB
    JNZ  Update_Frame_Timer
    MOV  svoFrame_LSB, #2000 & 255              ' yes, svoFrame = 2000
    MOV  svoFrame_MSB, #2000 >> 8
    CLR  svoPntr                                ' point to servo 0
    SETB Servo0                                 ' start servo sequence
    MOV  svoTimer, pos0                         ' start servo 0 timer
    SETB sync                                   ' sync bit on
    JMP  ISR_Exit
```

```
Update_Frame_Timer:
    SUB  svoFrame_LSB, #1                        ' DEC svoFrame
    SUBB svoFrame_MSB, /C

Check_Servo_Timer:
    MOV  W, ServoCtrl                            ' copy servo pins
    AND  W, #%00000011                           ' strip input bits
    SNZ                                          ' any servos on?
    JMP  ISR_Exit                                ' no, exit
    DEC  svoTimer                                ' yes, update timer
    SZ                                           ' still running?
    JMP  ISR_Exit                                ' yes, exit

Reload_Servo_Timer:
    INC  svoPntr                                 ' point to next
    CLRB svoPntr.1                               ' keep 0 - 1
    MOV  W, #pos                                 ' get pulse timing
    ADD  W, svoPntr
    MOV  FSR, W
    MOV  svoTimer, IND                           ' move to timer

Update_Servo_Outs:
    MOV  W, ServoCtrl                            ' copy servo pins
    AND  W, #%00000011                           ' strip input bits
    MOV  svoTemp, W                              ' save
    AND  ServoCtrl, #%11111100                           ' clear servo pins
    CLC
    RL   svoTemp                                 ' point to next
    CLRB svoTemp.2                               ' limit to lower bits
    OR   ServoCtrl, svoTemp                      ' update outputs

ISR_Exit:
    BANK 0
  ENDASM
  RETURNINT


' =========================================================================
  PROGRAM Start
' =========================================================================


' -------------------------------------------------------------------------
' Subroutine Declarations
' -------------------------------------------------------------------------

WAIT_SYNC       SUB     0                           ' wait for 20 ms frame
WAIT_TIX        SUB     1, 2                        ' wait in 10 us units
WAIT_MS         SUB     1, 2                        ' wait in 1 ms units
PUT_EE          SUB     3, 4                        ' write byte(s) to EEPROM
```

```
ADC0832        FUNC    1, 1                  ' read ADC
GET_EE         FUNC    1, 2                  ' read byte from EEPROM
GET_EE2        FUNC    2, 2                  ' read word from EEPROM

' I2C routines used by subs/funcs above

I2C_START       SUB     0                    ' generate I2C Start
I2C_STOP        SUB     0                    ' generate I2C Stop
I2C_OUT         SUB     1, 1                 ' write byte to SDA
I2C_IN          FUNC    1, 1                 ' read byte from SDA


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

Start:
  pos0 = 150                                 ' set start up positions
  pos1 = 150


Main:
  WAIT_SYNC                                  ' wait for 20 ms flag
  joyX = ADC0832 Ch0
  joyY = ADC0832 Ch1

  ' adjust for 10 uS servo values
  '
  IF joyX <= 50 THEN
    joyX = joyX << 1
    joyX = joyX + 50
  ELSE
    joyX = joyX >> 2
    joyX = joyX + 137
  ENDIF
  joyX = 300 - joyX                          ' invert X direction
  IF joyY <= 50 THEN
    joyY = joyY << 1
    joyY = joyY + 50
  ELSE
    joyY = joyY >> 2
    joyY = joyY + 137
  ENDIF
  joyY = 300 - joyY                          ' invert Y direction


Check_Buttons:
  IF BtnX = Pressed THEN
    INC btnTmr(0)
  ELSE
    btnTmr(0) = 0
```

```
  ENDIF
  IF BtnY = Pressed THEN
    INC btnTmr(1)
  ELSE
    btnTmr(1) = 0
  ENDIF


Mode_Handler:
  IF runMode = M_IDLE THEN Check_Start
  IF runMode = M_REC THEN Recording
  IF runMode = M_PLAY THEN Playing
  IF runMode = M_PAUSE THEN Play_Paused


Check_Start:
  ledPort = LED_OFF                             ' clear mode LED
  pos0 = joyX                                   ' follow joystick
  pos1 = joyY
  IF btnTmr(0) = BtnOK THEN
    IF PlayRec = RecordNow THEN
      numRecs = 0
      runMode = M_REC
    ELSE
      numRecs = GET_EE2 0, 0
      IF numRecs > REC_LAST THEN
        GOTO Empty_EE                           ' can't play blank EE
      ELSE
        runMode = M_PLAY                        ' playback activated
      ENDIF
    ENDIF
    eePntr = 2                                  ' beginning of servo data
    btnTmr(0) = 0                               ' reset for next process
  ENDIF
  GOTO Main


Recording:
  ledPort = LED_RED                             ' red on
  pos0 = joyX                                   ' follow joystick
  pos1 = joyY
  IF btnTmr(0) = BtnOK THEN                     ' stop recording
    GOTO Stop_Recording
  ELSE
    PUT_EE eePntr, joySticks                    ' save position values
    INC numRecs
    IF eePntr < REC_LAST THEN                   ' room in EE?
      eePntr = eePntr + 2                       ' yes, move pointer
    ELSE                                        ' no, stop recording
      GOTO Stop_Recording
    ENDIF
```

```
  ENDIF
  GOTO Main

Stop_Recording:
  PUT_EE 0, 0, numRecs                          ' write records count
  runMode = M_IDLE                              ' back to idle state
  btnTmr(0) = 0                                 ' reset debounce timer
  GOTO Main


Playing:
  ledPort = LED_GRN                             ' green on
  pos0 = GET_EE2 eePntr
  pos1 = __PARAM2
  DEC numRecs                                   ' update played count
  IF btnTmr(0) = BtnOK THEN                     ' stop
    runMode = M_IDLE
    btnTmr(0) = 0
    GOTO Main
  ENDIF
  IF btnTmr(1) = BtnOK THEN                     ' pause
    runMode = M_PAUSE
    btnTmr(1) = 0
    GOTO Main
  ENDIF
  IF numRecs > 0 THEN                           ' more movement records?
    eePntr = eePntr + 2                         ' point to next
  ELSE
    IF Repeat = Yes THEN                        ' loop?
      numRecs = GET_EE2 0, 0                    ' yes, reload records count
      eePntr = 2                                ' back to beginning
    ELSE
      runMode = M_IDLE                          ' no, reset
    ENDIF
  ENDIF
  GOTO Main


Play_Paused:
  IF btnTmr(1) = BtnOK THEN                     ' restart
    runMode = M_PLAY
    btnTmr(1) = 0
    GOTO Main
  ENDIF
  IF btnTmr(0) = BtnOK THEN                     ' stop
    runMode = M_IDLE
    btnTmr(0) = 0
  ENDIF
  GOTO Main
```

```
Empty_EE:
  FOR runMode = 1 TO 10                               ' flash red LED
    ledPort = LED_RED
    WAIT_MS 100
    ledPort = LED_OFF
    WAIT_MS 100
  NEXT
  runMode = M_IDLE
  GOTO Main


' -------------------------------------------------------------------------
' Subroutine Code
' -------------------------------------------------------------------------

' Use: WAIT_SYNC
' -- holds until the start of the next servo frame (up to 20 ms)

SUB WAIT_SYNC
  DO                                                ' wait for sync flag
  LOOP UNTIL sync = 1
  sync = 0                                          ' clear sync flag
  ENDSUB

' -------------------------------------------------------------------------

' Use: WAIT_TIX value
' -- waits in 10 us units; "value" can be a byte or word
' -- uses ISR-driven timer

SUB WAIT_TIX
  IF __PARAMCNT = 1 THEN
    tix = __PARAM1                                 ' get byte value
  ELSE
    tix = __WPARAM12                               ' get word value
  ENDIF
  DO WHILE tix > 0                                 ' hold while running
  LOOP
  ENDSUB

' -------------------------------------------------------------------------

' Use: WAIT_MS value
' -- waits in 1 ms units; "value" can be a byte or word
' -- uses ISR-driven timer

SUB WAIT_MS
  IF __PARAMCNT = 1 THEN
    tmpW1 = __PARAM1                               ' get byte value
  ELSE
    tmpW1 = __WPARAM12                             ' get word value
```

```
  ENDIF
  DO WHILE tmpW1 > 0
    tix = 100
    DO WHILE tix > 0
    LOOP
    DEC tmpW1
  LOOP
  ENDSUB

' -------------------------------------------------------------------------

' Use: bResult = ADC0832 mux
' -- where "mux" are mux selection bits

FUNC ADC0832
  tmpB1 = __PARAM1                             ' save MUX selection
  Clk = 0                                      ' preset clock polarity
  CS = 0                                       ' activate ADC
  SHIFTOUT Dio, Clk, MSBFIRST, tmpB1\4, 4      ' setup MUX
  SHIFTIN  Dio, Clk, MSBPOST, tmpB1, 4         ' read channel data
  CS = 1                                       ' release ADC
  RETURN tmpB1
  ENDFUNC

' -------------------------------------------------------------------------

' Use: PUT_EE address, byteVal1 {, byteVal2 }
' -- writes 'byteVal1' to 'address'
' -- writes [optional] 'byteVal2' to 'address'+1
' -- 'address' is a word

SUB PUT_EE
  IF __PARAMCNT = 3 THEN                          ' write one byte
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    i2cMulti = 0
  ELSE
    tmpW1 = __WPARAM12
    tmpB1 = __PARAM3
    tmpB2 = __PARAM4
    i2cMulti = 1
  ENDIF
  I2C_START
  I2C_OUT SlaveWr                                 ' send slave ID
  I2C_OUT tmpW1_MSB                               ' send address, high byte
  I2C_OUT tmpW1_LSB                               ' send address, low byte
  I2C_OUT tmpB1                                   ' send data byte
  IF i2cMulti = 1 THEN
    I2C_OUT tmpB2                                 ' send second data byte
  ENDIF
  I2C_STOP                                        ' finish
```

```
  '{$IFNOTDEF NOEEWAIT}
  DO                                            ' let write cycle finish
    I2C_START
    I2C_OUT SlaveWr
  LOOP UNTIL ackNak = Ack
  '{$ENDIF}
  ENDSUB

' -------------------------------------------------------------------------

' Use: byteVal = GET_EE address
' -- reads 'byteVal' from EEPROM location 'address'
' -- 'address' is a word (0 to 65535)

FUNC GET_EE
  tmpW1 = __WPARAM12

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT tmpW1_MSB                             ' send address, high byte
  I2C_OUT tmpW1_LSB                             ' send address, low byte
  I2C_START
  I2C_OUT SlaveRd
  tmpB1 = I2C_IN Nak
  I2C_STOP
  RETURN tmpB1
  ENDFUNC

' -------------------------------------------------------------------------

' Use: wordVal = GET_EE2 address
' -- reads 'wordVal' from EEPROM location 'address'
' -- 'address' may be a byte (0 to 255) or word (0 to 65535)

FUNC GET_EE2
  tmpW1 = __WPARAM12

  I2C_START
  I2C_OUT SlaveWr                               ' send slave ID
  I2C_OUT tmpW1_MSB                             ' send address, high byte
  I2C_OUT tmpW1_LSB                             ' send address, low byte
  I2C_START
  I2C_OUT SlaveRd
  tmpW1_LSB = I2C_IN Ack
  tmpW1_MSB = I2C_IN Nak
  I2C_STOP
  RETURN tmpW1
  ENDFUNC

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

```
' Use: I2C_START
' -- generates I2C start condition on SDA/SCL pins

SUB I2C_START
  I2CSTART SDA
  ENDSUB

' -------------------------------------------------------------------------

' Use: I2C_STOP
' -- generates I2C stop condition on SDA/SCL pins

SUB I2C_STOP
  I2CSTOP SDA
  ENDSUB

' -------------------------------------------------------------------------

' Use: I2C_OUT byteVal
' -- writes 'byteVal' to SDA pin
' -- affects global var "ackNak"
' -- EE address pointer must be preset before call

SUB I2C_OUT
  I2CSEND SDA, __PARAM1, ackNak
  ENDSUB

' -------------------------------------------------------------------------

' Use: byteVal = I2C_IN AckBit
' -- reads 'byteVal' from SDA pin
' -- EE address pointer must be preset before call

FUNC I2C_IN
  ackNak = __PARAM1.0
  I2CRECV SDA, tmpB3, ackNak
  RETURN tmpB3
  ENDFUNC

' =========================================================================
' User Data
' =========================================================================
```