Multi-pin PWM on the SX

Terry Hitt
July 10, 2009

Generating PWM with the SX processor is fairly simple using the SX/B PWM command. However, no other processing can occur while the PWM voltage is being generated. In order to allow other processing to occur the PWM code is typically placed inside an ISR (Interrupt Service Routine).

The required interrupt rate depends on the PWM output rate, the interrupt rate must be (Hertz * Levels). So let's say you wanted to create an output with a 4KHz update rate using 8-bits. Eight bits is 256 levels, so the interrupt rate would have to be (4000 * 256) or 1,024,000. If you are running the SX from it's internal clock at 4MHz you can see that there would be less than 4 cycles available per interrupt. 4,000,000 / 1,024,0000 = 3.90625. It is clear that we would need to sacrifice either update speed or resolution to be able to use an interrupt for generating this signal.

One way around this problem is to use more than one pin to generate the PWM output. We can create a 3-bit DAC using 3 resistors (1K, 2K and 4K. you can use multiple 1K resistors if you wish). This gives us 8 levels of output voltage without using PWM to toggle any pins. To create the levels in between, we will toggle the DAC pins between two of the eight output levels.

Since the PWM interrupt code will toggle the DAC between two adjacent values we cannot just divide the DAC value by 32 (1/8 of 256) to get the setting we want and let the interrupt toggle between that setting and the next highest setting. Because if we set the DAC to %111 that is the maximum, and there is no next highest setting.

What we need to do is divide the DAC range into 7 settings instead of 8, so the maximum base value is %110 instead of %111. Now you can probably guess that the calculations get sticky. Each DAC setting is 1/7 of the maximum instead of 1/8. There are 37 (256 / 7) values between each of the DAC settings. Since 37 is not a very computer friendly number, it's a little more work to get the PWM calculations correct.

Because there are 37 levels between DAC levels when we calculate our required interrupt rate we get (4000 * 37) = 148,000. This makes our interrupt rate 7 times lower. This gives us 27 (4,000,000 / 148,000) cycles for the interrupt routine and foreground code. With a little clever programming this should be doable.

The odd value calculations requires a fairly large number of instructions, so I'm only going to do the calculations in the main code via a subroutine. The interrupt contains only the bare minimum required to do the PWM.

Let's look at the code:

```
' Multi-Pin PWM
' Terry Hitt
' July 10, 2009
'                 4K
' RA.0 ---------/\/\/\-------+
'                 2K         |        +-----------------> PWM Output
' RA.1 ---------/\/\/\-------+        |  0.1uF
'                 1K         +--------+---|(---|Gnd
' RA.2 ---------/\/\/\-------+
'
' Uses multiple pins to create an 8-bit PWM output with 4 KHz update rate
'
DEVICE SX28,OSC4MHZ
FREQ 4_000_000,1_925_925  ' ISR uses 14 of 27 cycles

DAC        PIN RA OUTPUT

pwmLevel   VAR BYTE

temp1      VAR BYTE ' Used by SetDAC
temp2      VAR BYTE ' Used by SetDAC

' Interrupt variables
pwmDACBase VAR BYTE
pwmAccum   VAR BYTE
pwmFrac    VAR BYTE


INTERRUPT NOPRESERVE 148_000 ' 37 interrupts for 1 complete update (148000 / 37 = 4000)
  pwmAccum = pwmAccum + pwmFrac
  DAC = pwmDACBase + C
RETURNINT

SetDAC  SUB 1

PROGRAM Start NOSTARTUP

Start:
  ' Create Waveform
  SetDAC 0
  PAUSEUS 250
  FOR pwmLevel = 0 TO 255
    SetDAC pwmLevel
    PAUSEUS 50
  NEXT
  GOTO Start
END


SUB SetDAC
  temp1 = __PARAM1
  temp1 = temp1 / 37
  temp2 = __REMAINDER
  temp2 = temp2 * 7
  pwmDACBase = temp1
  pwmFrac = temp2
ENDSUB
```

First we setup the device with the two lines:

```
DEVICE SX28,OSC4MHZ
FREQ 4_000_000,1_925_925  ' ISR uses 14 of 27 cycles
```

The first line defines that we are using the SX28 device with the internal 4MHz oscillator. The second line defines the clock speed to be 4 MHz (4_000_000), and the apparent clock speed is 1.925925 MHz (1_925_925). This “apparent” clock speed is used by the SX/B compiler to generate the timing for delays. This value is found by taking the (((cycles per interrupt – the cycles used in the ISR) / cycles per interrupt) * SX Clock Speed) since the ISR takes 14 cycles, and the interrupt rate is 27 cycles we get (((27 – 14) / 27) * 4,000,000) = 1,925,925.

Next we declare our I/O pins for the DAC as outputs.

```
DAC         PIN RA OUTPUT
```

Then we declare the variables.

```
pwmLevel   VAR BYTE

temp1      VAR BYTE ' Used by SetDAC
temp2      VAR BYTE ' Used by SetDAC

' Interrupt variables
pwmDACBase VAR BYTE
pwmAccum   VAR BYTE
pwmFrac    VAR BYTE
```

Now comes the interrupt service routine (ISR).

```
INTERRUPT NOPRESERVE 148 000 ' 37 interrupts for 1 complete update (148000 / 37 = 4000)
  pwmAccum = pwmAccum + pwmFrac
  DAC = pwmDACBase + C
RETURNINT
```

The NOPRESERVE option tells SX/B that it does NOT need to preserve the systems variables __PARAM1..5. If you view the code listing using Run->View List you can see that these two lines of SX/B code do not use any of the system variables, so we are safe to not preserve them. Preserving the system variables adds many cycles to the ISR.

Here we also indicate that we want the interrupt routine to be executed 148,000 times a second. Note that you will get a warning that the actual interrupt rate is 148,148.148 because 4MHz / 27 = 148148.148. This doesn’t matter for our code.

Next we declare the subroutine that we are going to use to set the DAC value.

```
SetDAC  SUB 1
```

We indicate that the subroutine will be given 1 byte parameter, which is the value of the DAC.

Next we indicate where our program should start executing from, with the PROGRAM directive.

```
PROGRAM Start NOSTARTUP
```

The NOSTARTUP option tells the compiler to NOT generate the code to zero all the variables. We don't need this, and it makes debugging more difficult because you need to skip through all this initialization code.

Next is the main program code.

```
Start:
  ' Create Waveform
  SetDAC 0
  PAUSEUS 250
  FOR pwmLevel = 0 TO 255
    SetDAC pwmLevel
    PAUSEUS 50
  NEXT
  GOTO Start
END
```

Here we just set the DAC to zero, wait for the voltage level to settle (250uSec), then generate a ramp waveform. Then we repeat the whole process over again.

You will notice that the main code allows only 50uSec for the output to settle when generating the ramp. This is less than the 4KHz update rate (which would be 250uSec), but since we are only moving 1 bit in the output it will work fine. If you were changing from 0 to 255 you do need to wait 250uSec for the output level to change.

Next in the program is the subroutine that sets-up the PWM variables for the ISR.

```
SUB SetDAC
  temp1 = __PARAM1
  temp1 = temp1 / 37
  temp2 = __REMAINDER
  temp2 = temp2 * 7
  pwmDACBase = temp1
  pwmFrac = temp2
ENDSUB
```

Here we need to get the value of (DAC / 37) and (DAC // 37 * 7) instead of doing the MOD (//) 37 calcuation (which would take a lot of code space and execution time) we just use the remainder from the division operation. As the values are being calculated they need to be stored in a temporary variable. If the interrupt should trigger in the middle of the calculation the output would be incorrect. There is a slight chance that the interrupt could occur between the "pwmDACBase = temp1" and the "pwmFrac = temp2" lines, but it doesn't seem to affect the output.

Simulation with SXSim

Open this SX/B program in the SX-Key IDE, then start SXSim, then make sure the I/O panel is open by pressing the "I/O Panel" button in the command dialog box. Then press the RUN button in SXSim. You will see the RA port pins changing as the output value increments, then goes to zero.