A Virtual Peripheral DAC: Implementing Pulse Width Modulation

Introduction

This application note presents programming techniques for producing a programmable output voltage by smoothing a pulse width modulated (PWM) digital output with a simple resistor-capacitor low-pass filter, and essentially creating a digital to analog converter. This implementation uses the SX's internal interrupt feature to allow background operation of the code as a virtual peripheral, and uses the Parallax demo board, taking advantage of Parallax' *SX demo* software user interface and UART features to allow the SX to communicate simply and directly with a personal computer via a serial RS232C port.

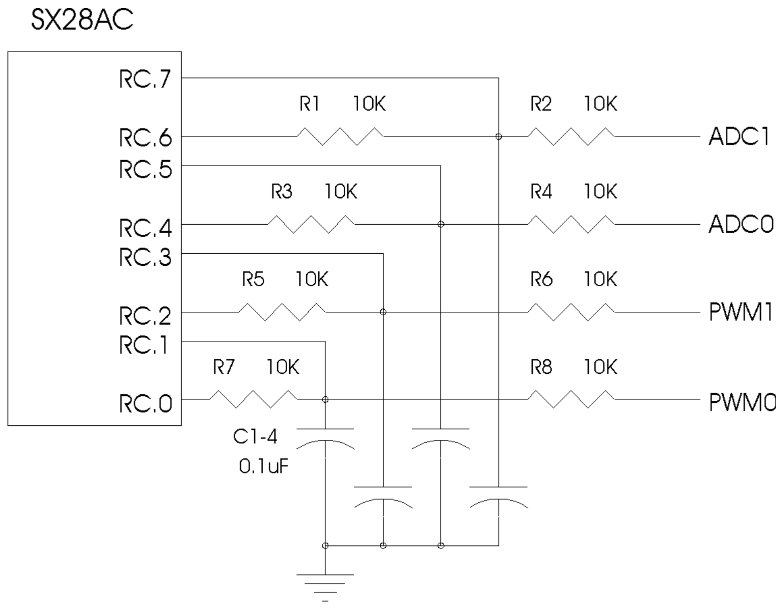

Figure 1 - Parallax demo board ADC/PWM circuit diagram

How the circuit and program work

The circuit is a simple resistor capacitor network (R5&C1 for pwm0, R7-C2 for pwm1) on each pwm port pin (see figure 1), which acts as a low-pass filter and smoothes the oscillating digital signal output so that it appears as a linear voltage whose value is determined by the duty cycle[1] of the signal, which is directly controlled by the 8 bit value in the *pwm* register.

The interrupt code segment is quite straight forward. The pwm value is added to an accumulator register during each pass through the interrupt and the pwm output is set to high every time the accumulator overflows. This technique allows the eight bit value in the pwm register to directly set the duty cycle of the pwm signal output and hence directly control the voltage on the capacitor.

With the resistor and capacitor values shown, the corner frequency or 3 dB point[2] for the low-pass filter is calculated as follows:

[1]The duty cycle is the proportion of time which the signal is high or active (i.e. charging the capacitor)

[2]The 3 dB point is the frequency at which the filter cuts the incoming signal level by 50% (to ½ it's original value). For a low-pass filter, signals at frequencies higher than the 3 dB point are attenuated even further.

$$\text{frequency}_{3dB} = 1 / (2 * \pi * R * C) = 1 / (2 * 3.1416 * 10k\Omega * 0.1uF) = 159 \text{ Hz}$$

In order to generate a fixed voltage from the pwm output from the changing digital pwm signals, a low-pass filter must be used whose corner frequency is lower than the lowest frequency component of the digital pwm signal. The length of the pwm cycle varies, depending upon the value in the corresponding *pwm* register, but the worst case is a pwm value of either 1 or 0FFh, in which case 256 interrupt passes are required to complete the pwm cycle[3]. We can calculate the period between interrupt passes as follows:

period (sec) = mode * prescaler * RETIW value[*] / osc. frequency, where mode=1 (turbo) or =4 (normal)

So, for the worst case of 256 interrupt passes, at a crystal frequency of 50 MHz, in turbo mode, with a prescaler of 1, and with an RETIW value of 163, the lowest frequency present in the pwm signal is:

$$\text{frequency}_{min} = 1 / \text{period} * 256 = 50 \text{ MHz} / (1 * 1 * 163 * 256) = 1.2 \text{ kHz}$$

By this we can see that the pwm signal will be excellently smoothed by the resistor and capacitor values chosen, since the lowest frequency component possible is well above the corner frequency for the low-pass filter.

The only inconvenience that the low-pass filter causes is that it also limits the pwm's output settling time (i.e. the maximum frequency and linearity if the pwm is being used as a signal generator). With the RC values shown, the settling time is calculated from the equation $V=V_{CC}(1-e^{-t/RC})$ for the charging[4] voltage on the capacitor after time *t*. If we want to calculate the settling time to eight bit resolution, we have:

$$V = V_{dd}(2^8 - 1) / 2^8 = V_{dd}(1 - e^{-t/RC}) \qquad \text{so:} \qquad \ln(1/2^8) = -t / RC$$

$$\text{therefore:} \quad t_{settle} = - RC * \ln(1/2^8) = - 10k\Omega * 0.1uF * \ln(1/2^8) = 5.5 \text{ msec}$$

So, with the above RC values, a maximum frequency of $f_{max} = 1/ t_{settle} = 1 / 5.5 \text{ ms} = 180$ Hz is possible on the pwm outputs for signals that vary if eight bit resolution is required[5].

The second 10kΩ resistor attached to each pwm output (i.e. R6 and R8), acts as a current limiting resistor to avoid discharging the voltage stored on the capacitor in case the output is driving a source which doesn't have a high input impedance. Ideally, the input impedance of the circuit being driven is at least a couple orders of magnitude larger (i.e. ≥ 1MΩ) than this resistor value so that the voltage on the capacitor remains accurate.

Modifications and further options

The Parallax demo board is designed so that port pins adc0 and adc1 can be swapped for pwm's simply by adjusting the program code. To do this (if you'd like 1-2 more pwm outputs) requires commenting out (or

[3]With a pwm value of 1, there will only be one carry generated every 256 passes by adding the pwm value to the pwm accumulator. In the case of a pwm value of 0FFh, only 1 of every 256 passes *won't* generate a carry (and hence a low at the pwm output).

[*] The interrupt is triggered each time the RTCC rolls over (counts past 255 and restarts at 0). By loading the OPTION register with the appropriate value, the RTCC count rate is set to some division of the oscillator frequency (in this case they are equal), which is the external 50 MHz crystal in this case. At the close of the interrupt sequence, a predefined value is loaded into the W register using the RETIW instruction which determines the period of the interrupt in RTCC cycles.

[4]The charging and discharging times are calculated similarly.

[5]In practise, circuit noise is usually larger than the 8-bit resoltion of the pwm, so that the settling time can be considered as somewhat less than the calculated value

removing) the *adc* code section, and reproducing the three lines of *pwm0:* code once (or twice for two more pwm outputs), while replacing *pwm0* with *pwm2*, *pwm0_acc* with *pwm2_acc*, *port_buff.0* with *port_buff.4*[6] in the three code lines. New register definition(s) should also be added in the *analog* bank for *pwm2* & *pwm2_acc* (and *pwm3* & *pwm3_acc*, if required). An example follows:

```
...
analog        =      $                      ;pwm bank
;
port_buff     ds     1                      ;buffer - used by all
pwm0          ds     1                      ;pwm0 - value
pwm0_acc      ds     1                      ;     - accumulator
pwm1          ds     1                      ;pwm1 - value
pwm1_acc      ds     1                      ;     - accumulator
pwm2          ds     1                      ;pwm2 - value
pwm2_acc      ds     1                      ;     - accumulator
pwm3          ds     1                      ;pwm3 - value
pwm3_acc      ds     1                      ;     - accumulator
...
:pwm0         add    pwm0_acc,pwm0          ;adjust pwm0 accumulator
              snc                           ;did it trigger?
              setb   port_buff.0            ;yes, toggle pwm0 high
:pwm1         add    pwm1_acc,pwm1          ;adjust pwm1 accumulator
              snc                           ;did it trigger?
              setb   port_buff.2            ;yes, toggle pwm1 high
:pwm2         add    pwm2_acc,pwm2          ;adjust pwm2 accumulator
              snc                           ;did it trigger?
              setb   port_buff.4            ;yes, toggle pwm2 high
:pwm3         add    pwm3_acc,pwm3          ;adjust pwm3 accumulator
              snc                           ;did it trigger?
              setb   port_buff.6            ;yes, toggle pwm3 high
;
;  <adc code removed except for this one line>
              mov    rc,port_buff           ;update port pins (port RC)
...
```

[6]for the 4th pwm output *pwm0* should be relaced with *pwm3*, *pwm0_acc* with *pwm3_acc,* and *port_buff.0* with *port_buff.6*