Pulse Width Modulated 'Linear' LED Bar Graph Display

Introduction

This application note presents a circuit which implements two design and programming techniques for SX virtual peripherals. The first technique is for reading the value of a potentiometer by measuring the time it takes to partially charge a capacitor through the potentiometer. The second technique is for displaying the eight bit result in a simulated linear manner using pulse width modulation to proportionally vary brightness of adjacent LEDs on a bar graph composed of sixteen LEDs. The result is a signal that 'slides' smoothly up and down the bar graph as the potentiometer is rotated instead of hopping from one LED to the next.

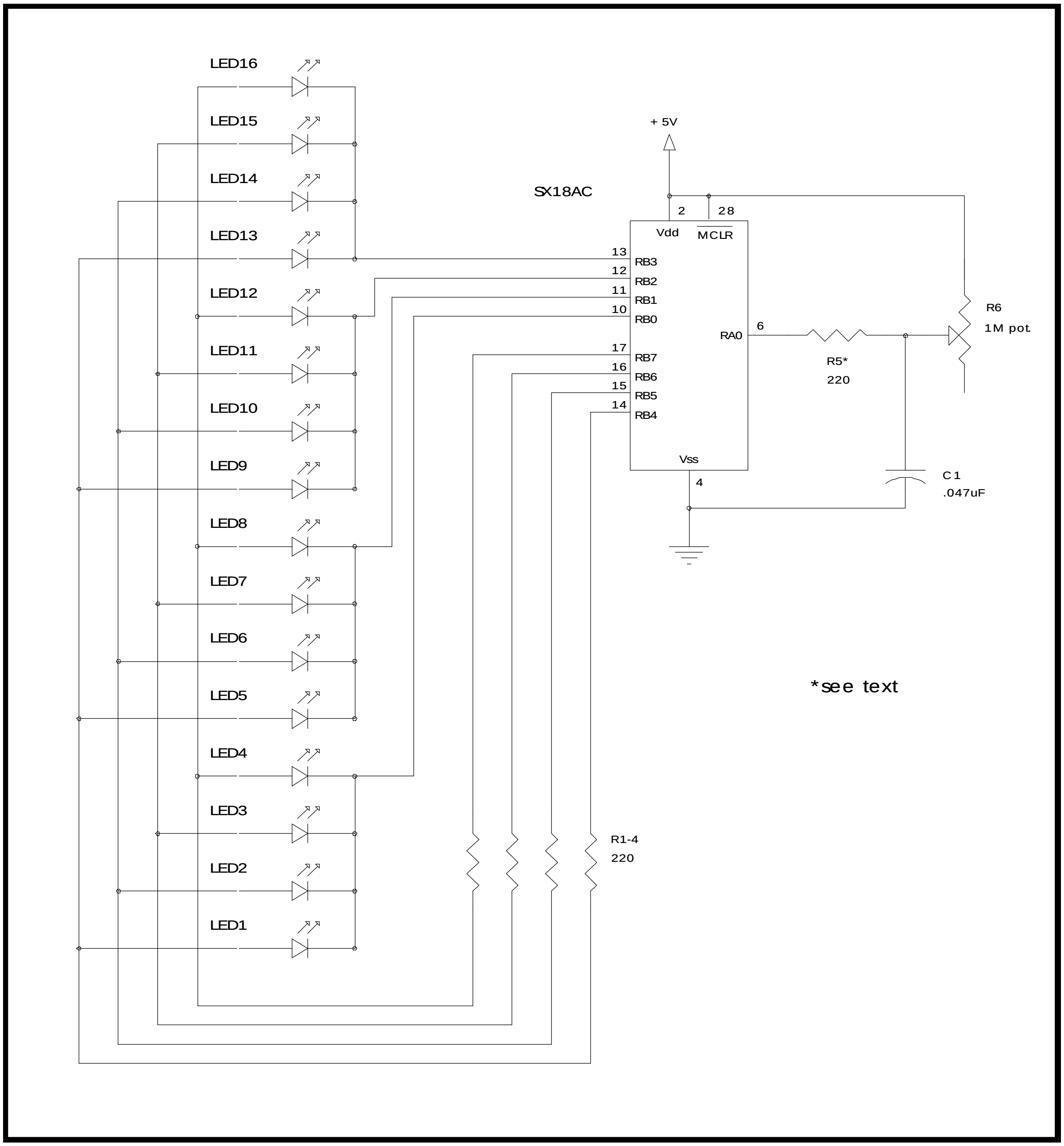

Figure 1 - Tested circuit of the pulse width modulated 16 LED bar graph display.

How the circuit and program work

Both sections of the circuit are set up as virtual peripherals which take advantage of the SX's internal interrupts to simplify programming and timing issues. 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 equal to the oscillator frequency, which is the internal 4MHz oscillator 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.

1) Reading the potentiometer

To read the value of the potentiometer, we measure the partial charging time (until the port input triggers high) of a simple RC circuit, which is directly proportional to the potentiometer value. The SX begins the timing measurement by discharging capacitor C1 through port RA bit 0 by setting the port to an output and driving it low, thereby essentially shorting the capacitor to ground through the current-limiting resistor R5 which prevents damage[1] to the SX port. The program code leaves port RA.0 low long enough to assure that the capacitor is discharged enough not to affect the next reading by more than 1/256th = 0.4% to maintain resolution on the order of 8 bits. It then clears the charge time counter register and switches port RA.0 to an input to begin charging the capacitor through the potentiometer. It increments the charge time register until the capacitor charges enough to trigger an input reading of high on the port. To reduce power consumption and avoid high current draws through the potentiometer when it is set near its lower limit, the program waits a specified time between potentiometer samples before clearing and taking another reading.

The capacitor and potentiometer values are chosen so that if the potentiometer is at its maximum value, the time to charge the capacitor enough to trigger the input to high and provide 8 bit timing resolution will be just slightly above 2^8 (= 256) interrupt cycles since each interrupt cycle corresponds to one pass through the count loop. By using the 'RETIW *int_period*' instruction to end the interrupt, the RTCC rolls over and an interrupt is triggered every *int_period*[2] RTCC cycles[3]. With the SX running on the internal 4MHz oscillator in turbo mode (i.e. 1 RTCC count/hardware clock count), the interrupt timing is:

$$t_{\text{interrupt}} = \textit{int\_period} / 4\text{MHz} = 200 / 4 \times 10^6 = 50\ \mu\text{sec}, \quad \text{so:} \quad t_{256\text{ loops}} = 12.8\text{ msec}$$

Starting with a potentiometer of 1MΩ, and knowing that the SX port input triggers high at about $0.25V_{CC}$, we can use the formula $V=V_{CC}\cdot(1-e^{-t/RC})$ for the charging voltage on the capacitor after time *t* to calculate the required capacitor value:

---

[1] R5 limits the maximum discharge current through the I/O port to under 30mA (i.e. $R_{discharge} \geq V_{dd} / I_{max} = 5V / 30mA = 167\Omega$). In practice, for small capacitor values (on the order of 0.01µF or less), R5 can probably even be omitted, since the total charge stored on the capacitor is minimal and shouldn't damage the SX.

[2] The value for the *int_period* variable controls the interrupt period. It is suggested that the minimum value for this variable be kept longer than the longest possible interrupt routine duration (converted to RTCC counts), and the maximum value should be 256, in which case *int_period*=0. Values less than the execution time of the interrupt routine will cause interrupt periods longer than 256 RTCC counts.

[3] Actual CPU cycle count may be multiples of *int_period* if the RTCC is operating with a prescaler other than 1.

$0.25V_{dd} = V_{dd} \cdot (1 - e^{-t/RC})$

hence: $C = -t / (R \cdot \ln(1 - 0.25)) = - 12.8 \times 10^{-3} / (1 \times 10^{6} \cdot \ln(0.75)) = 0.044$ uF

We want to be sure to use the whole 8 bit counter range, so we choose C a little large in order that the charge time extends slightly beyond 256 iterations of the count loop when the potentiometer is at its maximum, hence the value of C1 = 0.047uF[4]. Then, within the capacitor charge counting loop, we also watch the charge time counter for overflow and skip ahead with a charge time value set to maximum when this occurs (so that the bar graph doesn't loop around and display low values again if the potentiometer is near its maximum). To allow for tolerance values of the capacitor and potentiometer, final adjustments can be made to the *int_period* variable to assure that the whole 0-255 range is properly spanned by the potentiometer.

2) The LED bar graph output and pulse-width modulation

It is often desirable to minimize the number of pins used on the SX while maximizing the number of devices accessed. An effective and cost-efficient manner of doing this is to set the pins up as a matrix to read devices such as switch/keypad inputs or, as in this case, as outputs to access an array of 16 LEDs. While the LEDs are physically positioned one after another in a line as a bar graph, the circuit connections are made in a 4x4 matrix using only 8 SX port pins rather than 16. The program code then controls which LED row and column is active which then lights the corresponding LED. Here we have used the 4 high bits of port B as the source output rows for the LEDs and the lower 4 bits as sink (output=low) input columns.

Since we have 16 LEDs and a value for the potentiometer input of 0-255 (8 bits), we can create a smooth linear signal 'slide' effect by using pulse width modulation to partially and proportionally light adjacent LEDs when the output value lies between them. For example, assume we read a potentiometer value of 44. Since there are 16 LEDs, this value would correspond to LED number 44/16=2.75. Normally in this case, we'd have to choose between the 2nd and 3rd LEDs, but using pulse width modulation we can resolve the fraction after the decimal point by sending a 25% duty cycle to the second LED and a 75% duty cycle to the third one, providing the visual effect that the signal is somewhere between the two LEDs, and closer (3/4 way) to the 3rd one. We use the higher nibble of the potentiometer reading to select which of the 16 LED(s) will be lit, and the lower nibble for the pulse width modulation duty cycle calculation, which makes possible 16 levels of brightness. During each interrupt cycle, the pulse width modulation duty cycle count is incremented, and with the aid of a lookup table, the matrix on port B is set up to light the appropriate LED. In this way, each reading value from 0-255 is given a unique output pattern the LED bar graph.

Resistors R1-4 limit current to the LEDs, which at 100% duty cycle draw an $I_{LED} = V_{dd}/R = 5V/220\Omega \cong 23mA$. The values of R1-4 may be adjusted to reduce/increase the overall brightness, though the 30 mA source/sink maximum port current limit should be kept in mind.

[4] If the value of C1 is changed, the *clear_time* constant for discharging it should also be adjusted appropriately.

Program Listing

```
;******************************************************************************
;       Pulse Width Modulated 'Linear' LED Bar Graph Display - © Copyright 1998
;
;       Length: 87 bytes
;       Author: Craig Webb
;       Written: 97/02/26
;
;       This program implements two virtual peripherals using interrupts.
;       It shows to read a potentiometer as an 8 bit value and pulse width
;       modulate a bar-graph of 16 LEDs arranged in a 4x4 matrix on port B
;       in order to provide a smooth 'sliding' signal effect by varying the
;       brightness of adjacent LEDs when the potentiometer 8 bit value lies
;       somewhere between them.
;
;******************************************************************************
;
;****** Assembler directives
;
               DEVICE         pins18,pages1,banks8,stackx,optionx
               DEVICE         osc4mhz,turbo
               ID             '16 LEDs'
               RESET          Start                 ;set reset/boot address
;
;****************************** Program Variables ***************************
;
;****** Register definitions (bank 0)
;
               ORG     08h                      ;global variables 08-0Fh
reading        DS      1                        ;potentiometer reading
display        DS      1                        ;LED output to display
flags          DS      1                        ;program flags
;
; variables for LED interrupt routine
;
               ORG     10h                      ;bank 0 variables
mainbank       EQU     $
;
LED_bank       EQU     $                        ;(can be other than bank 0)
LED            DS      1                        ;holds which LED to light
cycle_count    DS      1                        ;pwm cycle count
pot_count      DS      1                        ;temporary pot timing count
clear_delay    DS      1                        ;delay period to clear cap.
sample_delay   DS      1                        ;delay period per sample
                                                ; (reduces power consumption)
;
;****** Bit variable definitions
;
pot            EQU     RA.0                     ;potentiometer in RC (input)
triggered      EQU     flags.0                  ;status of pot. reading
clearing       EQU     flags.1                  ;hi while cap. is clearing
;
;****** Constants
;
sample_time    =       2                        ;time between pot. readings
clear_time     =       45                       ;delay value for clearing
                                                ; the capacitor (>=2)
int_period     =       200                      ;interrupt period (based
                                                ;          on RTCC counts)
IO_portA       =       00001111b                ;Port A input/output setup
LEDs_off       =       0Fh                      ;RB value for LEDs=off
;
;************************ Interrupt Routines ***************************
;
               ORG     0
;
```

```
;*****  Virtual Peripheral : Read potentiometer
;
; This routine reads the value of the potentiometer by clearing the
; capacitor in the RC timing circuit and then measuring the time it takes
; the capacitor to charge until the port input goes high. To avoid high
; current draws at low potentiometer values, the routine only resamples
; after (256*sample_time) interrupt cycles. The maximum potentiometer
; reading is 255.
;
;       Input variable(s) : none
;       Output variable(s) : reading
;       Variable(s) affected : pot_count, clear_delay, sample_delay
;       Flag(s) affected : clearing, triggered
;       Timing cycles (turbo) : 12-charging, 14-triggered, 14-clearing
;
               JNB     clearing,:charge        ;are we clearing cap.?
               MOV     W,#11111111b            ;get port mask (=done)
               DECSZ   clear_delay             ;is count done?
               MOV     W,#11111110b            ;no, get port mask (=clearing)
               TEST    clear_delay             ;is count done?
               SNZ                             ;if not, skip ahead
               CLRB    clearing                ;yes, reset clearing flag
               AND     W,#IO_portA             ;get port setup byte
               MOV     !RA,W                   ;adjust pot port status
               CLR     pot_count               ;clear timing count
               JMP     :done_pot               ;jump past checking routine
:charge        JNB     pot,:adjust_count       ;triggered yet?
               MOV     W,pot_count             ;get timing count
               SB      triggered               ;is this first trigger cycle?
               MOV     reading,W               ;yes, store result
               SETB    triggered               ;set trigger flag
:adjust_count  INCSZ   pot_count               ;adjust reading counter
               JMP     :done_pot               ;was counter at maximum?
               MOV     W,#255                  ;no, store max. value
               SB      triggered               ;did we already get reading?
               MOV     reading,W               ;no, so set it to max.
               SETB    triggered               ; and flag that we got value
               DECSZ   sample_delay            ;time for new sample?
               JMP     :done_pot               ;if not, keep cycling
:trig          CLRB    triggered               ;yes, reset trigger flag
               SETB    clearing                ;set flag to clear cap.
               MOV     sample_delay,#sample_time       ;load sample and
               MOV     clear_delay,#clear_time         ; clear delay time counts
:done_pot                                      ;end of pot. reading routine
;
;
;*****  Virtual Peripheral : LED driver
;
; This routine drives the LED bar-graph array, providing 16 levels
; of brightness to allow an output slide effect between adjacent LEDs
; It must be called fairly often, otherwise the pulsing effect will
; become noticeable.
;
;       Input variable(s) : display
;       Output variable(s) : none
;       Variable(s) affected : cycle_count, LED
;       Timing cycles (turbo) : 21
;
;next instruction needed only if multiple variable banks are used
;
               MOV     W,<>display             ;get input (nibble-swapped)
               AND     W,#0Fh                  ;keep high 4 bits (which LED)
               MOV     LED,W                   ;save it
               MOV     W,display               ;get input reading again
               AND     W,#00001111b            ;keep lower 4 bits for PWM
               MOV     W,cycle_count-W         ;calculate which LED to have on
:zero_point    SZ                              ;adjust zero baseline up one*
               SC                              ;next one up? If not skip ahead
```

```
		INC	LED			;yes, increment to next LED
		MOV	W,LED			;get LED number
		CALL	LED_Table		;fetch LED value
		MOV	RB,W			;light LED
		INC	cycle_count		;adjust PWM cycle
		SNB	cycle_count.4		;time to reset (16 cycles)?
		CLR	cycle_count		;yes, start new cycle
;
;*this instruction shifts the whole display range up by one, thus making the
;  first LED dimly lit on a reading of 0, and the last lit fully on a reading
;  of 255. If it's preferable that all LEDs be off on a reading of 0, this
;  instruction may be removed or commented out.
;
;
		MOV	W,#-int_period		;interrupt again after
		RETIW				;   'int_period' RTCC counts
;
;****************************** Subroutines ********************************
;
;
;*****  Subroutine : LED_Table
;
; This is a look-up table that returns the output port value to light the LED
; contained in the W register. If W holds 0, then all LEDs are turned off.
;
LED_Table	ADD	PCL,W			;get RB value for LED1-16
		RETW	0Fh			;LEDs all off
		RETW	1Eh			;LED1
		RETW	2Eh			;LED2
		RETW	4Eh			;LED3
		RETW	8Eh			;LED4
		RETW	1Dh			;LED5
		RETW	2Dh			;LED6
		RETW	4Dh			;LED7
		RETW	8Dh			;LED8
		RETW	1Bh			;LED9
		RETW	2Bh			;LED10
		RETW	4Bh			;LED11
		RETW	8Bh			;LED12
		RETW	17h			;LED13
		RETW	27h			;LED14
		RETW	47h			;LED15
		RETW	87h			;LED16
;
;********************************* Main Program ******************************
;
;*****  Initialization routine
;
Start		CLR	RA			;clear port A
		MOV	!RA,#IO_portA		;set up port A
		MOV	RB,#LEDs_off		;set all LEDs off
		MOV	!RB,#0			;configure port B as outputs
		CLR	FSR			;reset all ram starting at 08h
:zero_ram	SB	FSR.4			;are we on low half of bank?
		SETB	FSR.3			;If so, don't touch regs 0-7
		CLR	IND			;clear using indirect addressing
		IJNZ	FSR,:zero_ram		;repeat until done
		MOV	!OPTION,#10001000b	;enable interrupt on rtcc=xtal/1
		MOV	sample_delay,#sample_time	;load sampling period
;
;***** Main program loop
;
Mainloop
;
		MOV	display,reading		;copy pot. to LED output display
;
;		<program code goes here>
;
```

```
                JMP     Mainloop                ;keep looping
;
                END
```