I microcontrollori PIC – parte III

In questa terza parte parliamo brevemente dell'ultima famiglia nata: i potenti PIC32, in particolare il PIC32MX795F512H. Accenniamo anche al nuovo sistema di sviluppo MPLABX della Microchip.

Trattiamo del timer 1 e del timer 2, dei comparatori, del modulo PWM e dell'interfaccia seriale asincrona, presentando inoltre altri esempi applicativi in MikroBasic e Assembler, con riferimento al PIC 16F877.

Il timer 1

Il timer 1 è un contatore / temporizzatore a 16 bit in quanto dotato di due registri, TMR1L e TMR1H, ciascuno a 8 bit. Come per il timer 0, questi registri si possono leggere e vi si può scrivere. Tuttavia, il timer 1 è più versatile. Ha le seguenti proprietà:

-16 bit

-il clock può essere interno o esterno

-il prescaler (divisore) ha 3 bit

-può operare in modo sincrono o asincrono

-il timer può essere controllato attraverso il comparatore sul piedino T1G

-avviene un interrupt per overflow

-risveglio all'overflow (clock esterno)

-può operare da temporizzatore per il modulo Capture / Compare

Selezione sorgente di clock

Clock	TMR1CS
Freq. Osc.	0
T1CKI pin	1

RC0/T1OSO e RC1/T1OSI sono usati per contare impulsi dalle periferiche ma hanno anche un'altra funzione: come sorgente di clock addizionale LP (Low Power). Adottando un quarzo da 32.768 KHz si possono facilmente ottenere impulsi di un secondo per orologi.

Se si seleziona il clock interno, TMR1H e TMR1L sono incrementati di multipli della frequenza di Fosc, secondo il prescaler. Se si seleziona il clock esterno, si può avere il funzionamento come timer o contatore, sia sincrono sia asincrono. Il prescaler permette la divisione del clock di ingresso per 2, 4 o 8.I piedini

RC0/T1OSO e RC1/T1OSI permettono di gestire al meglio il conteggio di impulsi provenienti da periferiche. Ecco i valori consigliati per i condensatori

Oscillatore	Frequenza	C1	C2
LP	32 kHz	33 pF	33 pF
LP	100 kHz	15 pF	15 pF
LP	200 kHz	15 pF	15 pF

La porta per il timer 1 è configurabile via software per l'uso del piedino T1G o l'uscita del comparatore C2. In quest'ultimo caso, si usano segnali analogici e se ne misura la durata.

Modo contatore

Il timer 1 nel modo contatore inizia a contare quando viene posto a 1 il bit TMR1CS. Gli impulsi sono collocati sul pin PC0/T1CKI, sul fronte di salita. Se il bit di controllo T1SYNC del registro T1CON è posto a 0, il contatore funziona in modo sincrono, sincronizzato col clock generale del microcontrollore. Se il microcontrollore è posto in modo Sleep (dormiente), TMR1H e TMR1L non sono più modificati in quanto il clock è fermo.

Modo timer

Per questa modalità occorre porre a 0 TMR1CS. Come temporizzatore il timer 1 fa sì che ogni impulso dell'oscillatore interno incrementi i registri. Se il quarzo è di 4 MHz, come esempio, i registri sono incrementati ogni micro secondo.

Esempio di programma in mikroBasic (MikroElektronica, Belgrado)

```
'Time 1 è configurato come timer e il prescaler divide per 8. Quando il timer raggiunge il fondo

'si genera un interrupt

main:

…

        PIR1.TMR1IF = 0         'reset TMR1IF flag

        TMR1H = 0x22            'valore iniziale del timer 1

        TMR1L = 0x00

        T1CON.TMR1CS = 0        'seleziona modo timer

        T1CON.T1CKPS0 = 1       'prescaler divide per 8

        T1CON.T1CKPS1 = 1

        PIE1.TMR1IE = 1         'abilita interrupt
```

INTCON = 0xC0 ‘abilita GIE e PEIE

T1CON.TMR1ON = 1 ‘parte timer

Registro interessato: T1CON

Bit 7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T1GINV	**TMR1GE**	**T1CKPS1**	**T1CKPS0**	**T1OSCEN**	**T1SYNC**	**TMR1CS**	**TMR1ON**

T1GIN Timer 1 Gate Invert bit: inverter sul pin T1G o sul comparatore C2 (uscita C2OUT). Gli impulsi possono essere conteggiati se a livello alto oppure basso.

TMR1GE Timer1 Gate Enable bit: determina la scelta fra attivare il piedino T1G o l’uscita del comparatore C2.

T1CKPS1 e T1CKPS0 Timer1 Input Clock Prescaler Select bit: scelta del prescaler secondo la seguente tabella

T1CKPS1	T1CKPS0	Prescaler (divide per)
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

T1OSCEN LP Oscillator Enable Control bit: 1 vuol dire che l’oscillatore ausiliario a basso consumo è abilitato; 0 no

T1SYNC Timer 1 External Clock Input Synchronization Control bit: sincronizzazione con il clock di sistema. 1 non sincronizzato; 0 sì

TMR1CS Timer TMR1 Clock Source Select bit: 1 conta gli impulsi sul pin T1CKI (contatore); 0 conta gli impulsi del clock di sistema, funziona cioè come temporizzatore

TMR1ON 1 = abilita timer 1, 0 disabilita.

Riassunto

Per usare il timer 1 occorre:

-selezionare il prescaler

-scegliere il modo contatore o temporizzatore

-abilitare, se necessario, T1OSCEN

Quando tutte e due i registri sono a 1 (overflow) si genera un interrupt.

Timer 2

Il timer 2 ha 8 bit e un funzionamento specifico. Gli impulsi dell'oscillatore al quarzo passano prima dal divisore (prescaler) determinato dai pin T2CKPS1 e T2CKPS0. L'uscita incrementa il registro TMR2 a partire da 00h. I valori di TMR2 e PR2 sono confrontati fino a quando si uguagliano. A quel punto TMR2 viene azzerato. Il postscaler del timer è incrementato ogni volta che si verifica questo evento. L'uscita del postscaler può generare un interrupt, se abilitato. Il momento del reset può anche essere usato come clock per la seriale.

Il modulo CCP (Capture-Compare-PWM, CCP1 e CCP2)

Come suggerisce il nome, si tratta di 3 dispositivi:

Capture: 'cattura' i cambiamenti di stato logico di TMR1

Compare: confronta i valori di 2 registri, di cui uno è TMR1

PWM: Pulse Width Modulation, genera treni di impulsi di frequenza e duty cycle (rapporto fra il tempo alto e il periodo) variabili. E' un modulo fondamentale per controllare la potenza in uscita verso un utilizzatore: motore (controllo di velocità, accelerazione, decelerazione), lampada, ecc. Regolando duty cycle e frequenza si varia l'energia disponibile. Il PIC16F887 che stiamo trattando ha 2 moduli PWM. In genere, per i bracci robotici, sono necessari 5, 6 o anche più PWM. Per fortuna, esistono PIC con molti PWM indipendenti, ad esempio il PIC18F2431 ne ha 8, il PIC24FJ256A 9.

Il periodo degli impulsi dipende dal valore di PR2 del Timer2. La formula è

Periodo PWM = (PR2 + 1) * 4Toscillatore * Prescaler Timer2

Ad esempio

Frequenza [kHz]	1.22	4.88	19.53	78.12	156.3
Prescaler TMR2	16	4	1	1	1
PR2	FFh	FFh	FFh	3Fh	1Fh

La durata dell'impulso è determinata da 10 bit: gli 8 bit di CCPR1L e due bit di CCP1CON. La formula è

Durata impulso = (CCPR1L,DC1B,DC1B0) * Tosc * prescalerTMR2

Il duty cycle è ottenuto dividendo la durata dell'impulso per il periodo:

Duty Cycle = (CCPR1L,DC1B1,DC1B0) / (4 * (PR2 + 1))

PWM con clock di 20 MHz

Frequenza PWM	1.22 kHz	4.88	19.53	78.12	156.3	208.3
Prescaler	16	4	1	1	1	1
Valore PR2	FFh	FFh	FFh	3Fh	1Fh	17h
Risoluzione	10 bit (1024)	1010	8	7	6	

Esempio in Basic

```
Dim duty_c as byte
Sub procedure InitMain()
        ANSEL,ANSELH = 0
        PORTC,TRISC = 0
        PWM1_Init(5000)                 'inizializzazione a 5 KHz
end sub
main:
InitMain()
        duty_c = 127
        PWM_Start
        PWM1_Set_Duty(duty_c)
….
```

Un esempio in assembler della gestione del modulo PWM

```
;Example6.mbas,23 ::          main:          ' Start of program
;Example6.mbas,24 ::          ANSEL = 0      ' All I/O pins are configured as digital
        CLRF    ANSEL+0
```

```
;Example6.mbas,25 ::          ANSELH = 0
        CLRF    ANSELH+0
;Example6.mbas,26 ::          PORTA = 255          ' PORTA initial state
        MOVLW   255
        MOVWF   PORTA+0
;Example6.mbas,27 ::          TRISA = 255          ' All PORTA pins are configured as inputs
        MOVLW   255
        MOVWF   TRISA+0
;Example6.mbas,28 ::          PORTB = 0            ' Initial state of PORTB
        CLRF    PORTB+0
;Example6.mbas,29 ::          TRISB = 0            ' All PORTB pins are configured as outputs
        CLRF    TRISB+0
;Example6.mbas,30 ::          PORTC = 0            ' PORTC initial state
        CLRF    PORTC+0
;Example6.mbas,31 ::          TRISC = 0            ' All PORTC pins are configured as outputs
        CLRF    TRISC+0
;Example6.mbas,32 ::          PWM1_Init(5000)      ' PWM module initialization (5 KHz)
        BCF     T2CON+0, 0
        BCF     T2CON+0, 1
        BSF     T2CON+0, 0
        BCF     T2CON+0, 1
        MOVLW   99
        MOVWF   PR2+0
        CALL    _PWM1_Init+0
;Example6.mbas,34 ::          current_duty = 16    ' Initial value of variable current_duty
        MOVLW   16
        MOVWF   _current_duty+0
;Example6.mbas,35 ::          old_duty = 0         ' Reset variable old_duty
```

```
        CLRF    _old_duty+0
;Example6.mbas,36 ::            PWM1_Start()        ' Start PWM1 module
        CALL    _PWM1_Start+0
;Example6.mbas,38 ::            while 1             ' Endless loop
L__main2:
;Example6.mbas,39 ::            if oldstate and Button(PORTA, 0,1,1) then       ' If the button connected to RA0 is pressed
        MOVLW   PORTA+0
        MOVWF   FARG_Button_port+0
        CLRF    FARG_Button_pin+0
        MOVLW   1
        MOVWF   FARG_Button_time+0
        MOVLW   1
        MOVWF   FARG_Button_activeState+0
        CALL    _Button+0
        MOVF    _oldstate+0, 0
        ANDWF   R0+0, 1
        BTFSC   STATUS+0, 2
        GOTO    L__main7
;Example6.mbas,40 ::            current_duty = current_duty + 1         ' increment variable current_duty
        INCF    _current_duty+0, 1
;Example6.mbas,41 ::            if Button(PORTA, 0, 1, 1) then
        MOVLW   PORTA+0
        MOVWF   FARG_Button_port+0
        CLRF    FARG_Button_pin+0
        MOVLW   1
        MOVWF   FARG_Button_time+0
        MOVLW   1
```

```
        MOVWF   FARG_Button_activeState+0
        CALL    _Button+0
        MOVF    R0+0, 0
        BTFSC   STATUS+0, 2
        GOTO    L__main10
;Example6.mbas,42 ::            oldstate = 255
        MOVLW   255
        MOVWF   _oldstate+0
L__main10:
;Example6.mbas,43 ::            end if
L__main7:
;Example6.mbas,46 ::            if oldstate and Button(PORTA, 1,1,1) then       ' If the button connected to RA1 is pressed
        MOVLW   PORTA+0
        MOVWF   FARG_Button_port+0
        MOVLW   1
        MOVWF   FARG_Button_pin+0
        MOVLW   1
        MOVWF   FARG_Button_time+0
        MOVLW   1
        MOVWF   FARG_Button_activeState+0
        CALL    _Button+0
        MOVF    _oldstate+0, 0
        ANDWF   R0+0, 1
        BTFSC   STATUS+0, 2
        GOTO    L__main13
;Example6.mbas,47 ::            current_duty = current_duty – 1         ' decrement value current_duty
        DECF    _current_duty+0, 1
```

```
;Example6.mbas,48 ::             if Button(PORTA, 1, 1, 1) then
        MOVLW     PORTA+0
        MOVWF     FARG_Button_port+0
        MOVLW     1
        MOVWF     FARG_Button_pin+0
        MOVLW     1
        MOVWF     FARG_Button_time+0
        MOVLW     1
        MOVWF     FARG_Button_activeState+0
        CALL      _Button+0
        MOVF      R0+0, 0
        BTFSC     STATUS+0, 2
        GOTO      L__main16
;Example6.mbas,49 ::             oldstate = 255
        MOVLW     255
        MOVWF     _oldstate+0
L__main16:
;Example6.mbas,50 ::             end if
L__main13:
;Example6.mbas,53 ::             if old_duty <> current_duty then       ' If current_duty and old_duty are not
        MOVF      _old_duty+0, 0
        XORWF     _current_duty+0, 0
        BTFSC     STATUS+0, 2
        GOTO      L__main19
;Example6.mbas,54 ::             PWM1_Set_Duty(current_duty)        ' equal set PWM to a new value,
        MOVF      _current_duty+0, 0
        MOVWF     FARG_PWM1_Set_Duty_new_duty+0
        CALL      _PWM1_Set_Duty+0
```

```
;Example6.mbas,55 ::          old_duty = current_duty          ' save the new value
        MOVF    _current_duty+0, 0
        MOVWF   _old_duty+0
;Example6.mbas,56 ::          PORTB = old_duty                 ' and show it on PORTB
        MOVF    _current_duty+0, 0
        MOVWF   PORTB+0
L__main19:
;Example6.mbas,59 ::          Delay_ms(200)          ' 200mS delay
        MOVLW   3
        MOVWF   R11+0
        MOVLW   8
        MOVWF   R12+0
        MOVLW   119
        MOVWF   R13+0
L__main21:
        DECFSZ  R13+0, 1
        GOTO    L__main21
        DECFSZ  R12+0, 1
        GOTO    L__main21
        DECFSZ  R11+0, 1
        GOTO    L__main21
;Example6.mbas,61 ::          wend
        GOTO    L__main2
        GOTO    $+0
; end of _main
```

La seriale EUSART

Il PIC16F887 dispone del modulo di interfaccia seriale EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter).

Alcuni PIC gestiscono anche la USB, come ad esempio il PIC 18F2550 e il PIC 18F2455.

L'EUSART del PIC 16F887 ha le seguenti caratteristiche:

-ricezione e trasmissione in Full Duplex

-lunghezza carattere 8 o 9 bit

-indirizzo riconosciuto in modo 9 bit

-riconoscimento overrun

-funzionamento anche in modo sincrono Half Duplex

Il più comune abbinamento del PIC 16F887 per la seriale è con il notissimo integrato MAX232.
Naturalmente il collegamento è incrociato: R1OUT di MAX232 con RX del PIC e T1IN del MAX232 con il pin TX del PIC.

La trasmissione di un dato avviene così:

-in stato di attesa la linea dati è al livello logico 1

-il dato in trasmissione ha un bit di start che vale 0

-il dato è lungo 8 o 9 bit

-il dato termina con un bit di stop che vale 1

L'abilitazione della trasmissione richiede i seguenti passi:

TXEN = 1 la trasmissione della EUSART è abilitata ponendo a 1 il bit TXEN del registro TXSTA

SYNC = 0 per funzionare in modo asincrono

SPEN = 1 EUSART è abilitata e il pin TX/CK è configurato come uscita

I dati da trasmettere devono essere scritti nel registro TXREG.

Il dato viene collocato nello shift register TSR.

La gestione della ricezione è analoga.

CREN = 1 abilitazione di EUSART alla ricezione

SYNC = 0

SPEN = 1.

Un esempio di gestione della seriale in assembler

```
;Example11.mbas,24 ::          main:                    ‘ Start of program
;Example11.mbas,25 ::          UART1_Init(19200)        ‘ Initialize USART module
        MOVLW     25
        MOVWF     SPBRG+0
        BSF       TXSTA+0, 2
        CALL      _UART1_Init+0
;Example11.mbas,28 ::          while 1                  ‘ Endless loop
L__main2:
;Example11.mbas,29 ::          if UART1_Data_Ready() then      ‘ If data has been received
        CALL      _UART1_Data_Ready+0
        MOVF      R0+0, 0
        BTFSC     STATUS+0, 2
        GOTO      L__main7
;Example11.mbas,30 ::          I = UART1_Read()         ‘ read it
        CALL      _UART1_Read+0
        MOVF      R0+0, 0
        MOVWF     _i+0
;Example11.mbas,31 ::          UART1_Write(i)           ‘ and send it back
        MOVF      R0+0, 0
        MOVWF     FARG_UART1_Write_data_+0
        CALL      _UART1_Write+0
L__main7:
;Example11.mbas,33 ::          wend
        GOTO      L__main2
        GOTO      $+0
; end of _main
```

Caratteristiche principali:

-32 bit; 80 MHz

-USB 2.0

-Ethernet 10/100

-8 DMA

-pipeline a 5 stadi

-moltiplicazione implementata nell'hardware ed eseguita in 1 solo ciclo macchina

-32 x 32 registri nel banco di memoria principale

-altri 32 x 32 registri ausiliari

-memoria Flash da 512 K (più 12 K per l'avvio)

-RAM di 128 K

-16 canali di A/D da 10 bit con velocità di 1 Mega campioni al secondo

-2 oscillatori interni da 8 MHz e 31 KHz

-Real-Time Clock e calendario con sveglia

-Watchdog

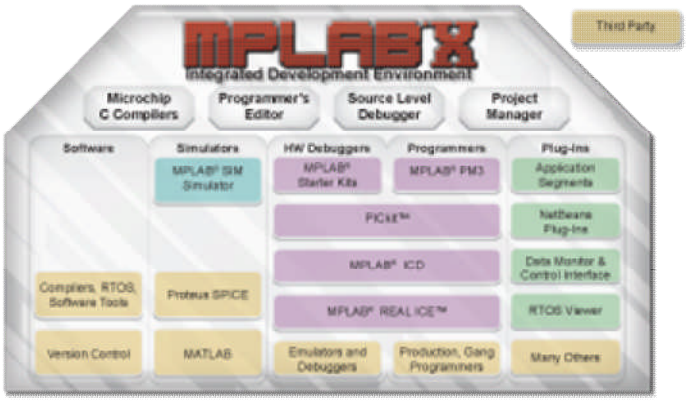
-compatibilità con i PIC a 16 bit

-UART e I2C.

Consigliamo, ove possibile, di passare progressivamente a questi nuovi PIC32.

Un nuovo ambiente di sviluppo: MPLABX

Microchip ha annunciato alla fine del 2010 il nuovo ambiente di sviluppo MPLABX che è destinato a sostituire il vecchio MPLAB IDE 8. Ora è in versione beta e free. E' basato su Java e quindi è disponibile anche per Linux. Rimangono comunque tutti i compilatori C e C++, ovviamente l'assembler, il simulatore e il debugger. MPLABX è compatibile con tutti i PIC esistenti, compresi i PIC32. Numerosi tutorial anche con video parlato (in Inglese) aiutano l'apprendimento e la migrazione da MPLAB IDE 8. Il nostro consiglio è di abbandonare il vecchio ambiente e passare a MPLABX.



Bibliografia

-Milan Verde, PIC Microcontrollers Programming in Basic, MikroElektronika, Belgrado, 2010

Hardware (oltre a quello descritto nelle parti I e II)

-Devantech PIC32 Based Microcontroller Module

-Digilent PIC32 MAX32 Arduino compatible chipKIT (Arduino Mega compatible)