LAB MICROCONTROLLORI

LAB 0: NUOVO PROGETTO

1. Creare una nuova cartella per il progetto
2. Aprire micro-C
3. Chiudere progetti aperti: project ->close project
4. New project->Standard project-> scrivere nome, seleziona cartella, device name: P18F44K22, device clock: 8MHz->nessun fil da aggiungere->finish
5. Salva il progetto
6. Fai una build: nella cartella verranno generati molti file
7. Apri il fil C con visual studio code
8. Scrivi il codice su visual studio code
9. Salvando su visual studio code aggiornerà automaticamente il codice su micro-C
10. Fai la build del programma su micro-C
11. Apri Proteus
12. Open project->file EasyPIC fornito dal prof
13. Clicca sul quadratone in basso a sinistra->click sulla cartella in program file->vai alla cartella del progetto->seleziona file.hex e apri->ok
14. In basso a sinistra run the simulation
15. Se devi cambiare qualcosa non è necessario ricaricare il file su Proteus tutte le volte, si aggiorna da solo quando fai build in micro-c

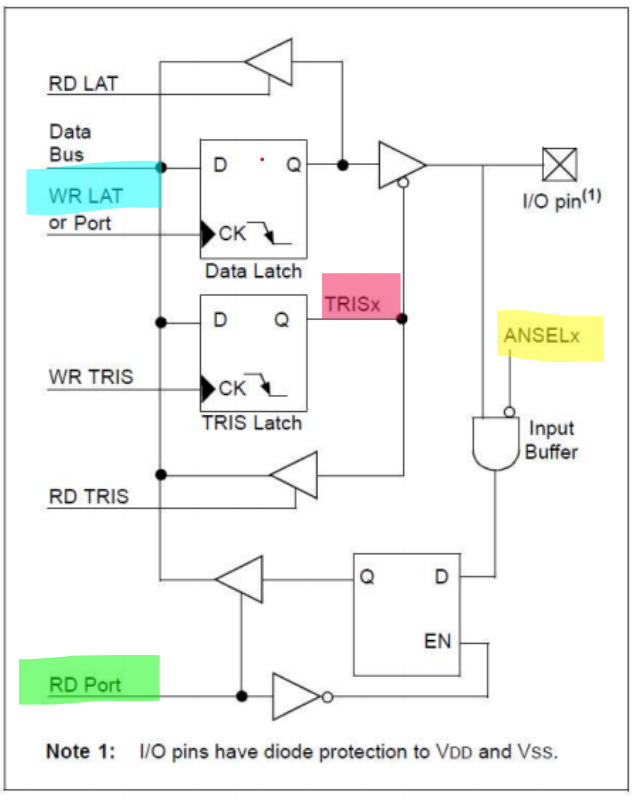
LAB 1: PORTE IO

Slide 2:

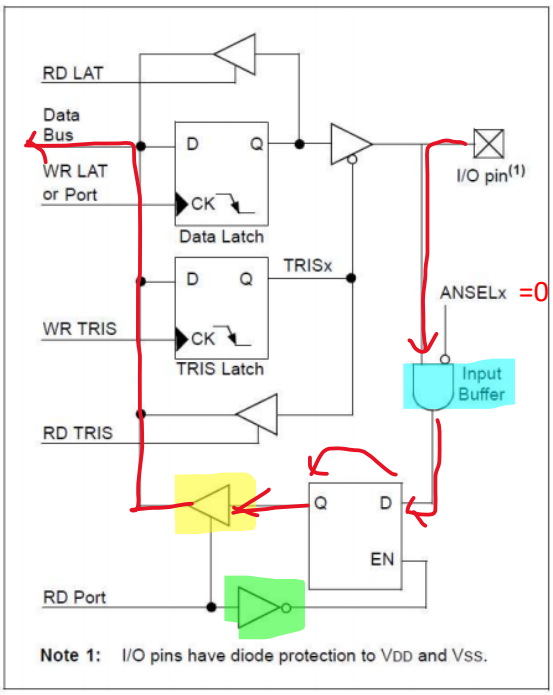
Important I/O REGISTERS

- **ANSELx:** Set Digital (0) or Analog (1) behavior of input stage — Analog selection
- **TRISx:** Set Output (0) or Input (1) direction of the pin
- **PORTx:** Read/Write buffer register — Gestito dal micocontrollore
- **LATx:** Output latch register

x=port name (A to E)

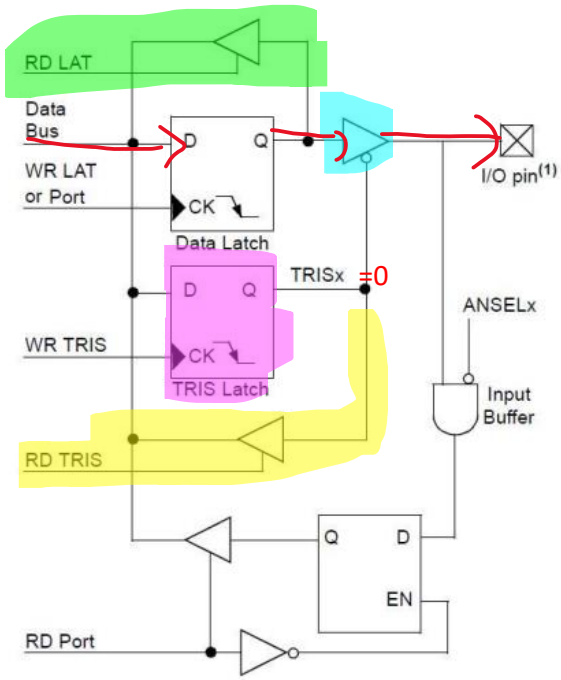


LETTURA DATO IN INPUT



Il dato entra dal pin passa dal buffer d'ingresso scrivendo ANSELX=0, poi dal flip-flop in basso che fa da sample and hold: se port=1 il buffer giallo è accesso e quello verde è spento e il valore di Q viene mandato sul data bus; se port=0 il buffer giallo è spento e nulla viene mandato sul data bus mentre quello verde è acceso e D passa a Q. In questo modo il dato non viene letto mentre sta cambiando. Dal data bus possiamo leggere il valore sul registro PORT

SCRITTURA OUTPUT



Accendo il buffer di uscita scrivendo TRISX=0. WR TRIS è gestito dal microcontrollore, una volta settato TRISX questo rimane grazie al latch.

Il valore da forzare in uscita viene scritto con il registro LATX. Con assembly usiamo il registro port, non farti troppe domande ed usa LAT.

Il ramo superiore e quello inferiore servono per leggere il valore di TRISX e LATX. Questo serve per confrontare il valore effettivo sul pin e quello che sto scrivendo, perché possono esserci delle non idealità (es: capacità parassite) in cui ci sono discrepante. (non credo vedremo questi casi)

REGISTER 10-8: TRISx: PORTx TRI-STATE REGISTER(1)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **TRISx<7:0>:** PORTx Tri-State Control bit
1 = PORTx pin configured as an input (tri-stated)
0 = PORTx pin configured as an output

Note 1: Register description for TRISA, TRISB, TRISC and TRISD.

REGISTER 10-10: LATx: PORTx OUTPUT LATCH REGISTER(1)

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
LATx7	LATx6	LATx5	LATx4	LATx3	LATx2	LATx1	LATx0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **LATx<7:0>:** PORTx Output Latch bit value(2)

Note 1: Register Description for LATA, LATB, LATC and LATD.
2: Writes to PORTx are written to corresponding LATx register. Reads from PORTx register is return of I/O pin values.

10.9 Register Definitions – Port Control

REGISTER 10-1: PORTX(1): PORTx REGISTER

R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown
-n/n = Value at POR and BOR/Value at all other Resets		

bit 7-0 **Rx<7:0>:** PORTx I/O bit values(2)

Note 1: Register Description for PORTA, PORTB, PORTC and PORTD.
2: Writes to PORTx are written to corresponding LATx register. Reads from PORTx register is return of I/O pin values.

REGISTER 10-6: ANSELD – PORTD ANALOG SELECT REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
ANSD7	ANSD6	ANSD5	ANSD4	ANSD3	ANSD2	ANSD1	ANSD0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **ANSD<7:0>:** RD<7:0> Analog Select bit
1 = Digital input buffer disabled
0 = Digital input buffer enabled

ESEMPIO: Setta i pin 0 e 1 della porta B come input e leggi il valore, gli altri pin sono output e scrivi tutti 1 su quei pin

```
void main() {

    TRISB=0b00000011; // in binario
       TRISB=3; //in decimale
       TRISB=0x03; //in esadecimale

    ANSELB=0b11111100;

   int val_pin0=PORTB.RB0;
   int val_pin1=PORTB.RB1;
   LATB.RB7=1;
   LATB.RB6=1;
   LATB.RB5=1;
   LATB.RB4=1;
   LATB.RB3=1;
   LATB.RB2=1;
   // or
   LATB=0b11111100;
}
```

SLIDE 4:
Schema easyPIC quella che si usa nel laboratorio ad usa esclusivamente didattico
ho una linea nera grossa di data bus che racchiude altre 8 linee .
La cosa cerchiata in rosso e un pulsante che ci permette di fissare in modo oanalogico un pin come input o output (cioè non a livello di frimware) se TRIS non coincideva con il valore selezionata da questo pulsante creo un corto circuito pericoloso (perciò è stata inserita una resistenza)

RETE DI PULL UP/PULL DOWN: in ingresso devo evitare di avere dei pin flottanti perché se si depositano delle cariche che rimangono li che possono portare ad un livello logico sbagliato anche se non sta succedendo nulla. Perciò devo inserire un collegamento a massa (con una resistenza per evitare corti) per evitare problemi

Su ogni PIN posso attivare la rete di pull up o pull down a seconda del caso.

SLIDE 6:
ATTENZIONE: non tutti i pin delle porte possono essere usati come input o output perché alcuni hanno delle funzioni speciali. Se ad esempio sul manuale del Pic si guarda il registro ANSEL della porta A si osserva che i bit 7,6 e 4 non possono essere modificati (o comunque non serve a nulla) questo perché i pin 6 e 7

servono per attaccare un possibile oscillatore esterno e il 4 è multiplexato con il timer 0. In definitiva **quando devi usare una porta come input o Output assicurati dal manuale che sia possibile.**

Not available PINs:

- RA6/RA7 OSC connected.
- RB6/RB7 Not available in debug mode.
- RE3 set to MCLR

LAB 2: INTERRUPT

Le slide sono fatte abbastanza bene comunque lascio gli appunti scritti a lezione.

Gli interrupt possono avere priorità diversi noi, comunque, useremo solo quelli ad alta priorità.

Se è una low priority indichiamo l'indirizzo 0018h della memoria mentre per gli high allo 008h

Grafico LP vs HP: arriva un LP interrupt che sospende l'esecuzione del codice se mentre si sta risolvendo il LP arriva un HP allora il LP viene interrotto e si fa quello ad HP, poi si torna a risolvere il LP e infine si torna alla normale esecuzione

Ie: interrupt enable per abilitare o meno l'interrupt delle porte, si trova nello special function register

GIE: global interrupt enable: può bloccare tutti gli interrupt

IF: bit per visualizzare qualche periferica ha scaricato l'interrupt, nota: il flag si attiva anche se l'enable è nullo

ISR: quello che viene fatto quando arriva l'interrupt: cercare di scriverla breve!!!

Dopo schematico di come funzione, nota che se gli enable sono 0 l'interrupt non passa.

L'interrupt flag va resettata a livello di firmware

HOW TO=regole d'oro da seguire

GIE da settare per ultimo in modo che non parte l'interrupt prima di aver finito di inizziallizare

ISR: prima si trova chi è stat poi si ressettano le flag e infine si scrive

RICORDA: oltre a resettare il flag dobbiamo leggere il valore della porta in modo tale che il valore si aggiorni e il flag non si alzi di nuovo

REGISTRI IMPORTANTI

INTCON : gestisce gli interrupt ad alta priorità

IOCB: permette di mascherare gli interrupt della porta B

ATTENZIONE: l'interrupt flag di B diventa 1 quando il vecchio valore di B e quello nuovo sono diversi (il valore di port B cambia) se io nella routine di interrupt io non faccio un operazione di lettura della porta B il vecchio valore non viene aggiornato e l'interrupt viene scatenato al infinito quindi **al interno della ISR fai una lettura della portB**

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts including peripherals
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts including low priority

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority interrupts
0 = Disables all low priority interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** Port B Interrupt-On-Change (IOCx) Interrupt Enable bit(2)
1 = Enables the IOCx port change interrupt
0 = Disables the IOCx port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared by software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared by software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** Port B Interrupt-On-Change (IOCx) Interrupt Flag bit(1)
1 = At least one of the IOC<3:0> (RB<7:4>) pins changed state (must be cleared by software)
0 = None of the IOC<3:0> (RB<7:4>) pins have changed state

Note 1: A mismatch condition will continue to set the RBIF bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.
2: RB port change interrupts also require the individual pin IOCB enables.

REGISTER 10-13: IOCB: INTERRUPT-ON-CHANGE PORTB CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	U-0	U-0	U-0
IOCB7	IOCB6	IOCB5	IOCB4	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-4 **IOCB<7:4>:** Interrupt-on-Change PORTB control bits
1 = Interrupt-on-change enabled(1)
0 = Interrupt-on-change disabled

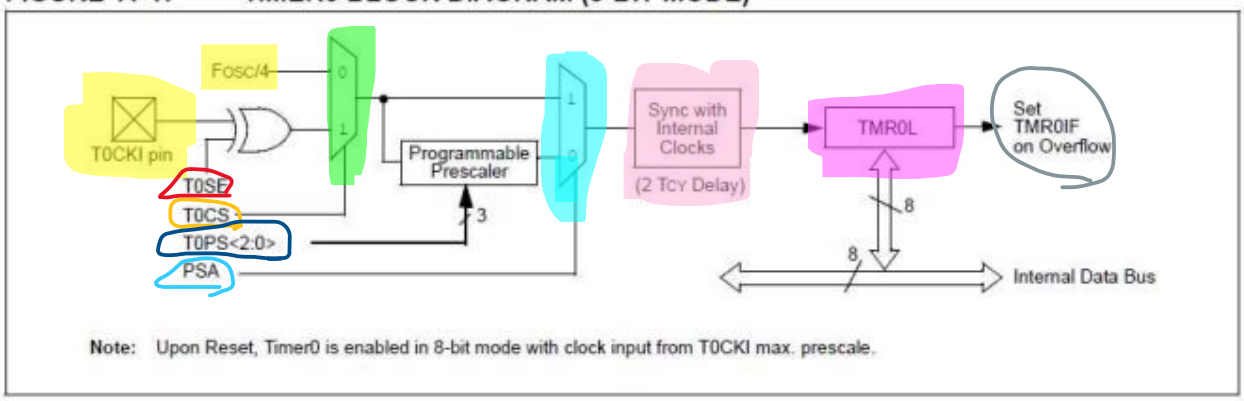
Note 1: Interrupt-on-change requires that the RBIE bit (INTCON<3>) is set.

ESEMPIO: quando premo il pin 0 della porta B fai qualcosa

```
void main() {

 TRISB.RB0=1; // chiudi il buffer di uscita
 ANSELB.RB0=0; // apri il buffer d'ingresso

 INTCON.RBIE=1; // attivo l'interruprt enable della porta B
 INTCON.RBIF=0; // azzero la flag di port B
 IOCB=0b00000001; // solo il pin 0 può scatenare un interruprt quando il suo valore cambia
 INTCON.GIE=1; // attivo SEMPRE PER ULTIMO (in modo tale da non avere interruprt durante l'inizzializzazione) il global interrupt enable
 // potevo scrivere in un colpo solo INTCON=0b10001000

}

void interrupt {
   if (INTCON.RBIF==1){ // la porta B ha scatenato un interrupt (potevo anche non scrivere ==1)
     if (PORTB.RB0){ // verifico che il pin 0 è passato da 0 a 1 in questo modo faccio anche quella famosa lettura che mi serve per aggiornare il valore
                     // nota che rilasciando il bottone passo da 1 a 0 scatenando un altro interrupt, ma con questo if non entrerò a fare le mie cose al rilascio del bottone

       FAI COSE
       INTCON.RBIF=0;// risetto a zero il flag

     }
   }
}
```

LAB 3: TIMER 0 E DISPLAY LCD

FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)



Il timer 0 può ricevere come frequenza di clock un input esterno dal pin T0CKI oppure del clock interno Fosc/4= 20/4=5 MHz, seleziono i due ingressi con l'input del multiplexer T0CS (Timer 0 clock source select bit).
Con T0SE (Timer 0 source edge selct bit) posso scegliere se far incrementare il clock sul fronte di salita oppure su quello di discesa

Posso prescalare il counter (cioè far incrementare il timer non ad ogni colpo ma ogni tot colpi) modificando PSA (timer 0 prescaler assigment bit) e T0PS (Timer 0 prescaler select bits). PSA attiva il multiplexer per scegliere tra il segnale prescalato oppure quello non prescalato. T0PS è il valore per qui lo prescalo ed è un numero a 4 bit. Se ad esempio T0PS=110-> PR=128 cioè ogni 128 colpi incremento il timer. (per come viene assegnato il valore di PR a seconda di T0PS bisogna guardare sul datasheet)

Tutte queste variabili si trovano nel registro **T0CON: TIMER 0 CONTROL REGISTER**

11.1 Register Definitions: Timer0 Control

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON**: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS<2:0>**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

Il segnale viene poi sincronizzato (completa con il perché) e poi va al counter TMR0L.

Se il timer0 va in overflow scateniamo un interrupt e il bit TM0IF del registro INTCON viene settato a uno.
Il tempo di overflow in ms può essere calcolato con la formula:

$$T_{TMR0IF} = \left(\frac{F_{OSC}}{4}\right)^{-1} \cdot PR \cdot (256 - TMR0L)$$

$$f_{osc} = 8MHz \rightarrow \left(\frac{f_{osc}}{4}\right)^{-1} = 0{,}5\ ms$$

TMR0L è un registro a 8 bit in cui carico il valore iniziale di timer 0

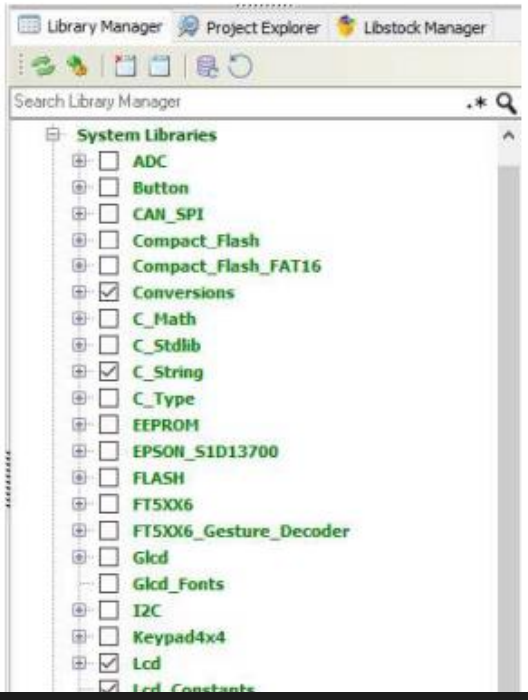
Quando gestisco più interrupt usa if…. Else if….. else if….. non if…if….if…. perchè occupa meno codice assembly

(ALTRI SPONI CHE NON HO GUARDATO LEZIONE 3 MINUTO 12)

DISPLAY LCD

Per il display LCD ci sono delle funzioni apposta che scrivo sotto, comunque il display usa le porte B (quindi non puoi usare portB e il dsplay contemporaneamente). Per utilizzare tali funzioni devo inserire le apposite librerie spuntandole in micro C. inoltre devo specificare le connessioni

Enable the library



Specify the LCD position to the library

```
·  // Lcd module connections
·  sbit LCD_RS at LATB4_bit;
·  sbit LCD_EN at LATB5_bit;
·  sbit LCD_D4 at LATB0_bit;
30 sbit LCD_D5 at LATB1_bit;
·  sbit LCD_D6 at LATB2_bit;
·  sbit LCD_D7 at LATB3_bit;
·
·  sbit LCD_RS_Direction at TRISB4_bit;
-  sbit LCD_EN_Direction at TRISB5_bit;
·  sbit LCD_D4_Direction at TRISB0_bit;
·  sbit LCD_D5_Direction at TRISB1_bit;
·  sbit LCD_D6_Direction at TRISB2_bit;
·  sbit LCD_D7_Direction at TRISB3_bit;
40 // End Lcd module connections
```

```
1  // Lcd module connections
2  sbit LCD_RS at LATB4_bit;
3  sbit LCD_EN at LATB5_bit;
4  sbit LCD_D4 at LATB0_bit;
5  sbit LCD_D5 at LATB1_bit;
6  sbit LCD_D6 at LATB2_bit;
7  sbit LCD_D7 at LATB3_bit;
8
9  sbit LCD_RS_Direction at TRISB4_bit;
10 sbit LCD_EN_Direction at TRISB5_bit;
11 sbit LCD_D4_Direction at TRISB0_bit;
12 sbit LCD_D5_Direction at TRISB1_bit;
13 sbit LCD_D6_Direction at TRISB2_bit;
14 sbit LCD_D7_Direction at TRISB3_bit;
15 // End Lcd module connections
16
17 void main() {
18
19     int a = 10; // varibaile numerica
20     char numtxt[7]; // variabile char da 7 caratteri
21
22     char txt[17]; // variabile char da 17 caratteri
23
24     Lcd_Init(); // inizzializza LCD
25     Lcd_Cmd(_LCD_CLEAR);// metti il cursore in posizione uno
26     Lcd_Cmd(_LCD_CURSOR_OFF);// nacondi il cursore
27
28     strcpy(txt,"Ciao"); // per le variabili string non posso fare txt="ciao" ma devo usare la funzione string copy
29     Lcd_Out(1,4,txt); //alla riga 1 della colonna 4 stampa txt
30
31     IntToStr(a,numtxt); // se devo stampare un numero lo devo convertire in una stringa con questa funzione
32     strcat(txt,numtxt); //attacca numtxt a txt e lo salva in txt
33     // Se provo a scrivere su una parte di LCD in cui ho già scritto qualcosa il vecchio testo viene sovrascritto
34     // se scrivo in due sezioni diverse invece la vecchia stringa non viene cancellata
35     //ES 1:
36     Lcd_Out(1,1,"ciao");
37     Lcd_Out(1,8,"ciao");  // RISULTATO : "ciao   ciao"
38     // ES 2:
39     Lcd_Out(1,1,"ciao");
40     Lcd_Cmd(_LCD_CLEAR);
41     Lcd_Out(1,8,"ciao");  // RISULTATO : "       ciao"
42 }
```

LAB 5: ATOMIC OPERATION E OTTIMIZZAZIONE CONTI CON ADC

ATOMIC OPERATIO

Un operazione atomica è un operazione che viene eseguita dal inizio alla fine senza interruzioni, non tutte le operazioni sono così: se ad esempio sommo due numeri da 16 bit dovrò eseguirlo in due cicli macchina (perché ho celle di memoria da 8 bit).

Ammettiamo di stare facendo un a serie di operazioni e arriva un interrupt in un momento sfigato in cui sto ancora lavorando sulla mia variabile a 16 bit, nel ISR faccio delle operazioni basate su quella variabile. Siccome l'interrupt è arrivato mentre ci stavo lavorando avrò un comportamento indesiderato. Per evitare questo comportamento indesiderato devo rendere l'operazione atomica (cioè farla senza interruzioni); per fare ciò disattivo il global interrupt faccio l'operazione e poi lo riattivo. Non è bellissimo ma è quello che abbiamo.
Non disattivare l'interrupt peer troppe righe di codice però

ATTRIBUTO VOLATILE

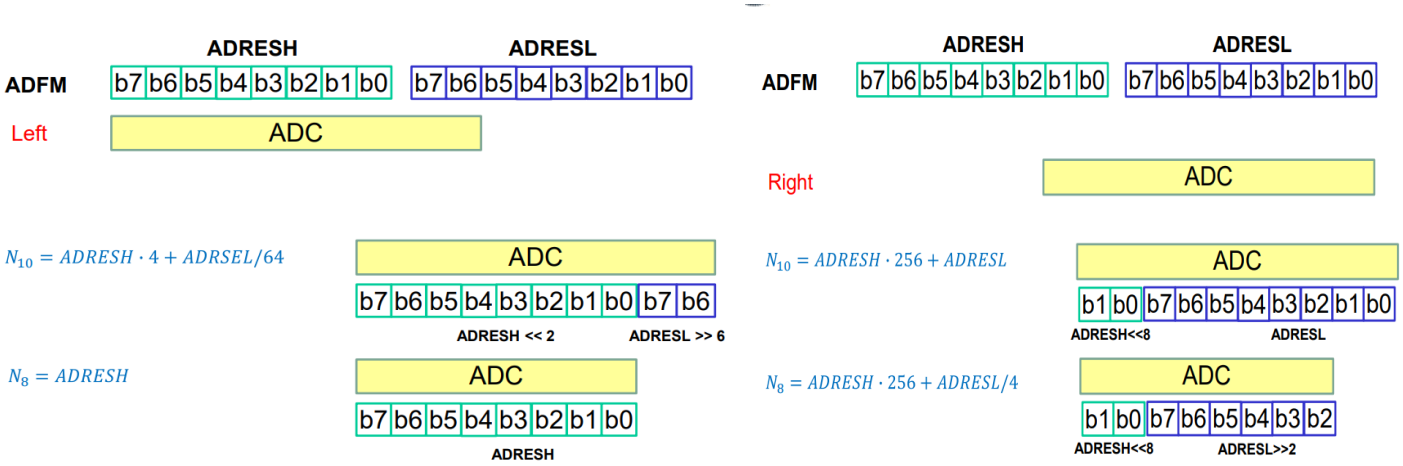
Quando usi una variabile globale dichiarala con volatile, perché? Non farti troppe domande e fallo

```
volatile int foo;
```

ADC

L'ADC del pic lavora a 10 bit, quindi il risultato va salvato in due variabili ADRESH : Analog Digital RESult High e ADRESL: Analog Digital RESult Low.

Oppure posso salvarlo in 8 bit arrotondando (tengo solo la parte alta). A questo punto ho due opzioni:



I due MSB di ADRESL costituiscono i due LSB del risultato
i due LSB di ADRESH costituiscono i due MSF del risultato
A seconda di quello che scelgo la conversione avviene in modo diverso.

Per stampare il risultato della conversione dobbiamo moltiplicare il valore della conversione per il passo, (diciamo l'unità di misura) es l'ADC può leggere un valore da 0 a 5V=5000mv e ha 1024 livelli (se considero 10 bit)

$$V_{ADC,10} = N_{10} \cdot LSB_{10}$$

$$LSB_{10} = \frac{FSR}{2^{10}}$$

L'unico problema è che è molto dispendioso per il PIC fare 5000/1024, ci conviene arrotondare 1024 a 1000 in modo da ottenere solo una moltiplicazione per 5, avremmo un errore ma vabbe

$$LSB_{10} = \frac{5000\ mV}{1024} \cong \frac{5000\ mV}{1000} = 5\ mV$$

$$\frac{5000\ mV}{1024} = 4{,}8828125\ mV$$

$$err_{10} = \frac{5\ mV - 4{,}8828125\ mV}{5\ mV} = 2{,}34\%$$

Se volessimo ottimizzare la precisione e performance possiamo farci furbi:

$$\frac{5000}{1024} = \frac{5^4 \cdot 2^3}{2^{10}} = \frac{5^4}{2^7}$$

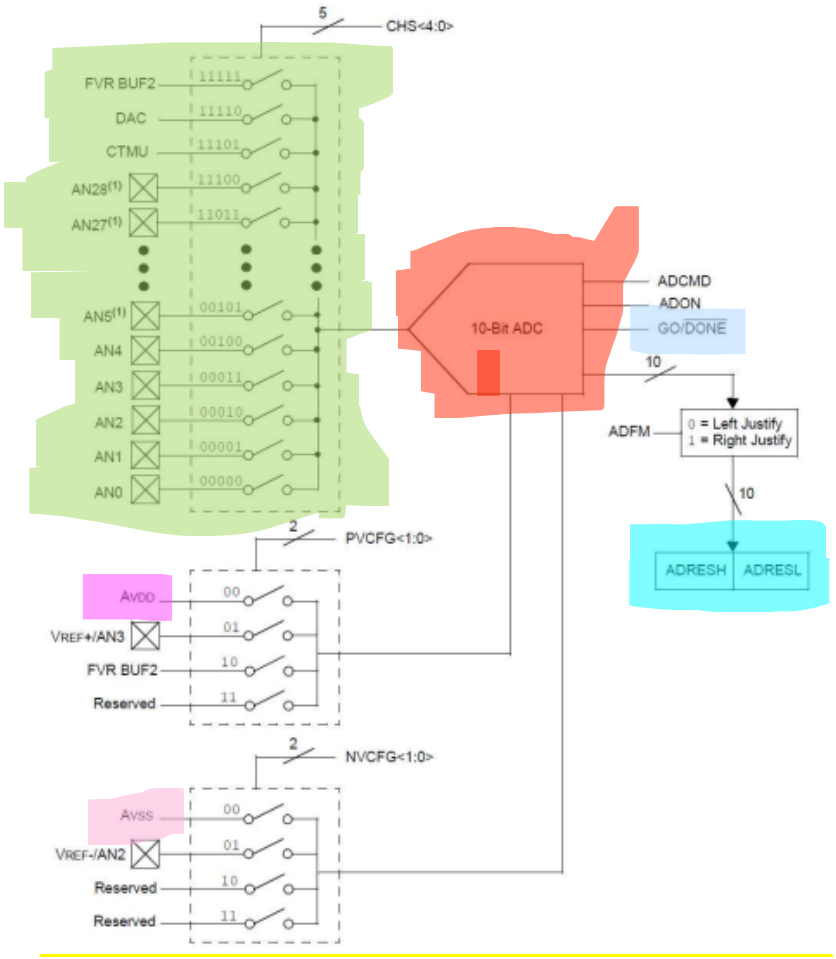
$$V_{ADC,10} = N_{10} \cdot \frac{5^4}{2^7} = \frac{N_{10} \cdot 5 \cdot 5 \cdot 5 \cdot 5}{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}$$

In questo modo dobbiamo fare 4 moltiplicazioni per 5 e 7 shift a destra che è più rapido di 5000/1024.

Dobbiamo però stare attenti a problemi di overflow: se faccio N10*5000 e poi divido il risultato per 1024 andre in overflow alla prima operazione! Quindi devo alternare moltiplicazioni e divisioni in modo inteliggenti

In generale nel microcontrollore si evitano operazioni con floating point, si usano sempre interi e per le divisioni ci si riconduce a degli shift e si fanno arrotondamenti e approssimazioni

LAB 6: ADC



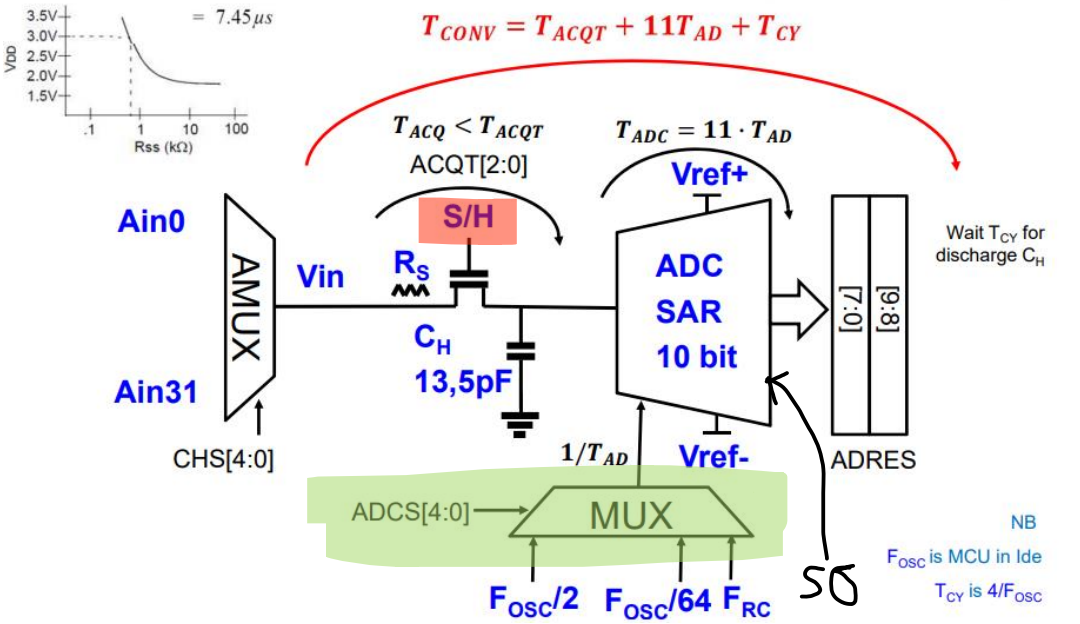
- Nel Pic 18F45k22 abbiamo una sola ADC a 10 livelli
- ci sono 32 ingressi che facendo riferimento ad un'unica ADC sono multiplexati col registro CHS
- funziona alla frequenza di clock f_{osc} (non il ciclo macchina $f_{osc}/4$)
- Possiamo scegliere il riferimento di tensione positiva e negativa noi useremo sempre 0 e 5 V
- Il risultato della conversione viene salvato in ADRESH e ADRESL (guarda lab precedente per capire come viene salvato)
- La conversione si avvia settato il bit GO/DONE a 1 e quando bisogna iniziare la conversione e viene settato a 0 dal ADC quando la conversione è finita, inoltre l'ADC scatena un interrupt quando finisce la conversione

T_{ACQ} = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient

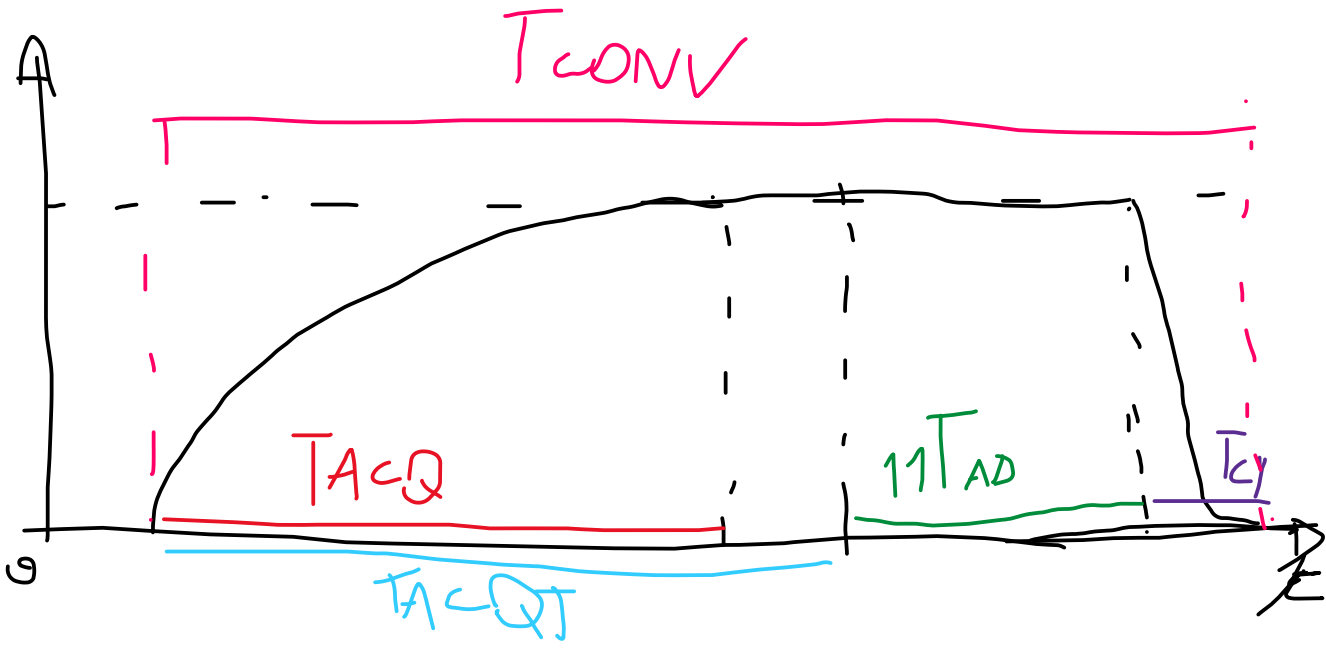
$= T_{AMP} + T_C + T_{COFF}$

$= 5\mu s + T_C + [(Temperature - 25°C)(0.05\mu s/°C)] = 5\mu s + 1.20\mu s + [(50°C - 25°C)(0.05\mu s/°C)]$

$= 7.45\mu s$



- Quando diamo il go chiudiamo il sample and hold
- Affinché la tensione all'ingresso del ADC raggiunga V_{in} dobbiamo aspettare un tempo T_{ACQ} (ACQ=aquisition) la cui formula è riportata sopra si noti che dipende dalla temperatura, dalla tensione di alimentazione e dalla V_{in}, con i nostri dati sono 7,45 us
- Una volta che in ingresso al ADC arriva V_{in} (dopo T_{ACQ}) dobbiamo dare lo start conversion (SOC), lo start conversion non viene direttamente settato a livello di firmware; ma viene settato dopo quanto tempo dopo aver dato il go (quindi aver chiuso il sample and hold) la conversione può iniziare settando il registro ACQT (aquisition temporization). In altre parole: settiamo il tempo T_{ACQT} che l'ADC deve aspettare per convertire dopo aver dato il go, $T_{ACQT} > T_{ACQ}$
- L'ADC (che è di tipo SAR: ad approssimazioni successive) necessita di 10 colpi di clock per finire la conversione più uno per salvare il risultato. Il periodo del clock è T_{AD} che è multiplexato tra diverse possibili frequenze. Il multiplexer è controllato dal registro ADCS : AD clock selection
- T_{CY} è il periodo della macchina T cycle=4/f_{osc}, alla fine della conversione dobbiamo aspettare un T_{CY} perché la capacità di sample and hold (C_H) deve scaricarsi per tornare al valore inziale e poter prendere un nuovo dato.



Registri da impostare per usare l'ADC

- Set analog input

  ADCON0.CHS<4:0>

- Set TAD

  ADCON2.ADCS<2:0> **C**lock **S**election

- Configure voltage references

  ADCON1.PVCFG<1:0>

  ADCON1.NVCFG<1:0>

- Select ACQT

  ADCON2.ACQT<2:0>

- Set justification

  ADCON2.ADFM Come suddivere il dato tra ADRESH E ADRESL

- Turn on the ADC

  ADCON0.ADON Per spegnere l'ADC in modo da consumare meno potenza

- Start A/D conversion

  ADCON0.GO/DONE = 1

- Interrupt/Polling

  Interrupt PIR1.ADIF

  Polling ADCON0.GO/DONE

- Stop A/D conversion (GO/DONE = 0)
- 10 bits result after Tconv

  ADRESH

  ADRESL

Interrupt

- ADC Interrupt Priority (IPR1.ADIP), not used
- Set ADC Interrupt Enable (PIE1.ADIE=1) **P**eripheral **I**nterrupt **E**nable: **AD I**nterrupt **E**nable
- Clear ADC Interrupt Flag (PIR1.ADIF=0) **P**eripheral **I**nterrupt **R**eset: **AD I**nterrupt **F**lag
- Enable Peripheral interrupt (INTCON.PEIE=1) **P**eripheral **I**nterrupt **E**nable: dobbiamo autorizzare l'enable per le periferiche
- Set Global Interrupt Enable (INTCON.GIE = 1)
- Wait interrupt in ISR
- Check ADC Interrupt Flag
- Clear ADC Interrupt Flag (PIR1.ADIF=0), exit ISR

Polling

- Polling GO/DONE = 0

IMPORTANTE: l'ADC ha una frequenza di lavoro minima e massima

TABLE 27-22: A/D CONVERSION REQUIREMENTS PIC18(L)F2X/4XK22

Standard Operating Conditions (unless otherwise stated)
Operating temperature Tested at +25°C

Param. No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
130	TAD	A/D Clock Period	1	—	25	µs	-40°C to +85°C
			1	—	4	µs	+85°C to +125°C
131	TCNV	Conversion Time (not including acquisition time) **(Note 1)**	11	—	11	TAD	
132	TACQ	Acquisition Time **(Note 2)**	1.4	—	—	µs	VDD = 3V, Rs = 50Ω
135	TSWC	Switching Time from Convert → Sample	—	—	**(Note 3)**		
136	TDIS	Discharge Time	1	—	1	TCY	

Note 1: ADRES register may be read on the following TCY cycle.
2: The time for the holding capacitor to acquire the "New" input voltage when the voltage changes full scale after the conversion (VDD to VSS or VSS to VDD). The source impedance (*Rs*) on the input channels is 50 Ω.
3: On the following cycle of the device clock.

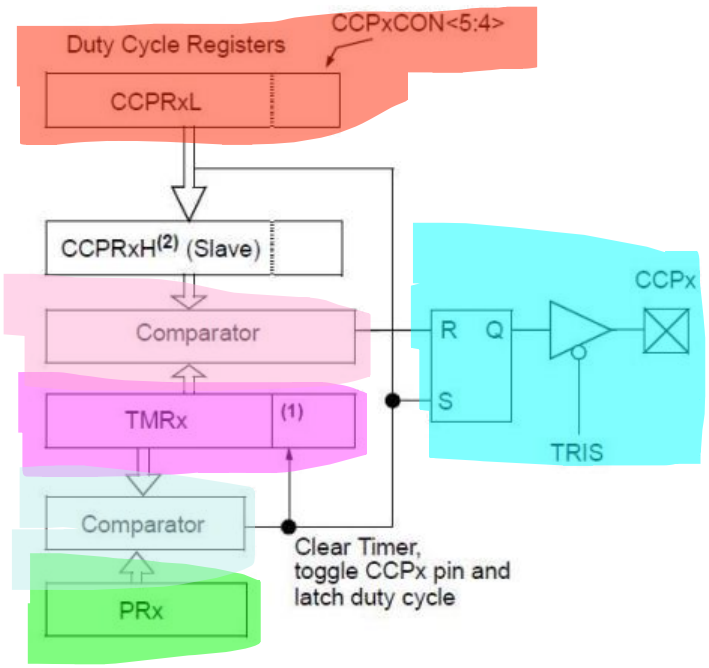
In condizioni standard min 1 us massimo 24 us

LAB 7: PWM

Capitolo 12-13-14 manuale PIC
La modalità capture and compare si appoggia al timer 1/3/5
La modalità PWM si appoggia al timer 2/4/6

Generazione di un segnale PWM con T fissato e T_{up} deciso da noi a seconda del valore analogico in uscita che vogliamo.
Col il PWM generiamo un seganle analogico con valore $A\frac{T_{up}}{T}$

Architettura



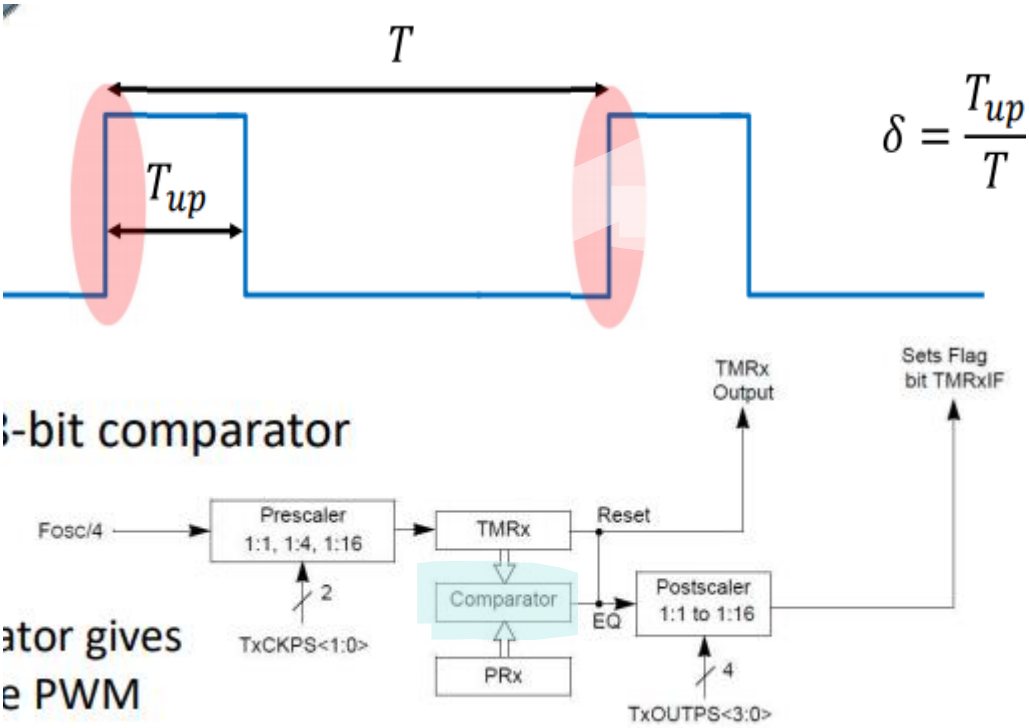
I Pin di pwm sono indicati con CCPRx (x=nome pin), ovviamente TRISx acceso.

TRMx (x=numero del timer quindi 2/4/6) il timer serve per dare la temporizzazione è può essere prescalato col registro TMRxPR . Per fissare il valore di T settiamo il valore del registro PRx.

$T = (PRx + 1) \cdot TMRxPS \cdot \frac{4}{f_{osc}}$
Per settare T_{up} invece dobbiamo agire sul registro CCPRxL adesso vediamo meglio.

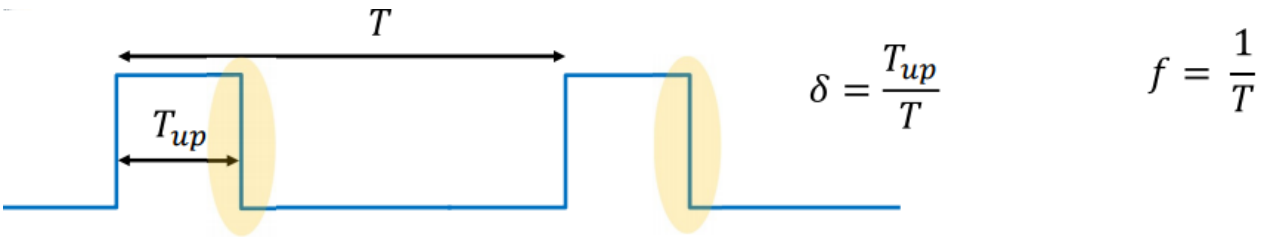
I due comparatori servono uno per fare la parte di rising del segnale e uno quelle dalla falling

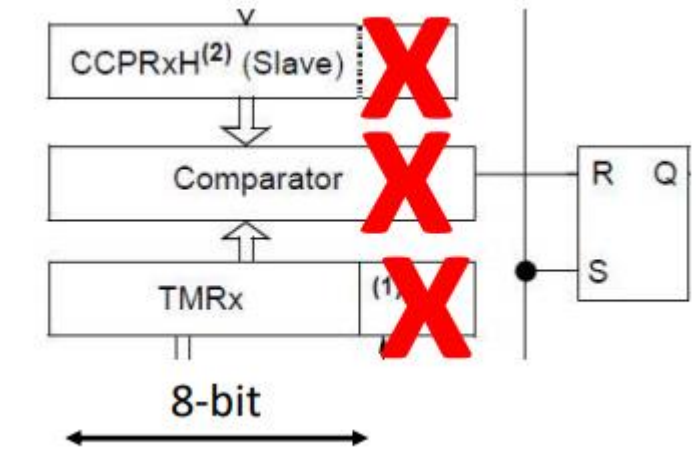
Parte di set: rising edge



Quando il valore del timer x è uguale a PRx verrà impostato a 1 il reset del latch, dando così 1 in uscita dal pin e generando il rising edge. Il timerx è un timer a 8 bit con solo tre possibili valori di prescaler TMRxPS. Inoltre verra generato un interrupt cge può essere postscalato e usato per altre cose ma non ci interessa.

Parte di reset: falling edge



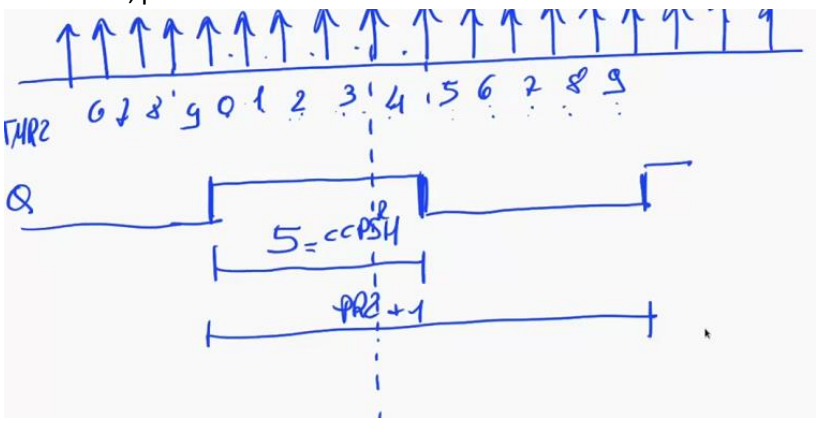


Quando il registro CCPRxH è uguale a TMRx viene scatenato il reset portando l'output a zero. Per ora ignoriamo gli altri due bit con le X rosse sopra.
ATTENZIONE: la x di CCPRxH non è la stessa di TMRx. Infatti per il timer x=2/4/6 invece per CCPR è il numero del pin che sis tà utilizzando.
Con questa struttura

$$T_{up} = CCPxH \cdot TMRxPS \cdot \frac{4}{f_{osc}}$$

In realtà anche dalla prima immagine si osserva che prima di CCPxH ho CCPRxL. Il registro CCPRxL è stato aggiunto come registro intermedio per evitare cicli spuri quando si vuole cambiare il valore di CCPxH

Supponiamo
PR2=9
CCP5H=5
quindi il duty cycle sarà 50%: quando il timer2 arriva a 10 (PR2+1) viene generato il set e il timer viene riazzerato, poi a 5 viene effettuato il reset



Cosa succede se mentre siamo a livello logio alto CCPH cambia? Da 5 a 2 per esempio, succede che al ciclo succesivo perdo il primo valore del PWM. Per ovviare a questo problema uso un doppio registro: io modifico CCPRL in modo tale che CCPRH venga cambiato in modo sincrono quando ho il prossimo set.

Estensione a 10 bit
Per essere più precisi col valore di T_{up} in modo che possa avere più duty cycle possibili e quindi più precisione, più granularità. Per fare ciò vengono aggiunti due bit al TRMx che oscillano però a 1/fosc. E aggiungo due bit a CCPRxL chiamati CCPxCOM <4:5>. Con questa truttura

$$T_{up} = CCPxL \cdot TMRxPS \cdot \frac{4}{f_{osc}} + CCPxCOM < 4:5 > \cdot TMRxPS \cdot \frac{1}{f_{osc}}$$

Essendo

$$T = (PRx + 1) \cdot TMRxPS \cdot \frac{4}{f_{osc}}$$

Si ottiene

$$\delta = \frac{T_{up}}{T} = \frac{4CCPxL + CCPxCOM < 4:5 >}{4(PRx + 1)}$$

Impostare PWM:

1. Spegner Buffer d'uscita del pin che vogliamo usare. Perché mentre il PWM viene settato possono succedere delle commutazioni spurie (meglio evitare azioni meccaniche esterne indesiderate che possono causare situazioni pericolo per l'utente o per l'ambiente)
2. Selezionare il timer sorgente con il registro CCPTMRSx
3. Caricare nel registro PRx il valore del prescalere
4. Impostare la modalità PWM (infattia abbiamo anche le modalità cmpture and compare) con registro CCPxCON
5. Scegliere T impostando CCPRxL

6. Istruzioni per impostare l'interrupt del timer in modo da evitare configurazioni spurie al accensione: se abbasso il tris del pin di output in modo non sincrono al PWM il primo ciclo avrà un duty cycle diverso, per evitare configurazione spurie devo accendere il TRIS sul SET del PWM, in laboratorio possiamo anche trascurarlo
7. Riaccendere buffer d'uscita pin

```c
void main() {
    //1-spegnere buffer d'uscita della porta che usiamo come PWM: la CCP5
    TRISE.RE2 = 1;

    //2-connettere timer 2 a PWM di CCP5
    CCPTMRS1.C5TSEL0 = 0;
    CCPTMRS1.C5TSEL1 = 0;

    //3-imposto prescaler timer 2
    T2CON = 0b00000111;

    //4-imposto CCP5 in modalità PWM
    CCP5CON.CCP5M3 = 1;
    CCP5CON.CCP5M2 = 1;
    // CCP5CON = 0b00001100;

    //5-imposto Ton e T
    CCPR5L=128;
    PR2=255; // quindi duty cycle del 50%

    //6-lo balziamo

    //7-accendo CCP5
    TRISE.RE2=0;

    while (1){
        delay_ms(100);
        CCPR5L++; // aumenta duty cycle ogni MS
    }
}
```

La struttura del timer 1/3/5 ha una struttura molto più complicata dei timer già visti ma non ci interessa capirla .
modalità … usiamo un solo PWM per pilotare una struttura a HALF-BRIDGE che consente di invertire il valore del PWM in uscita in modo da avere segnali "sia negativi che positivi" (far girare il motore in avanti o indietro). Questo circuito non è integrato nel microcontrollore ma

REGISTRI UTILIE

CCXCON : CCPxM selezione modalità (noi useremo sempre PWM)
CCPTMES0 (timer selection)

In tabella 3 del PIC vedo pin che possono essere usati per il PWM
Per aprire l'osclilloscopio debug>digital oscilloscope

REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/DONE	ADON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6-2 **CHS<4:0>: Analog Channel Select bits**
00000 = AN0
00001 = AN1
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
10101 = AN21(1)
10110 = AN22(1)
10111 = AN23(1)
11000 = AN24(1)
11001 = AN25(1)
11010 = AN26(1)
11011 = AN27(1)
11100 = Reserved
11101 = CTMU
11110 = DAC
11111 = FVR BUF2 (1.024V/2.048V/2.096V Volt Fixed Voltage Reference)(2)

bit 1 **GO/DONE:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

Note 1: Available on PIC18(L)F4XK22 devices only.
2: Allow greater than 15 µs acquisition time when measuring the Fixed Voltage Reference.

REGISTER 17-2: ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
TRIGSEL	—	—	—	PVCFG<1:0>		NVCFG<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TRIGSEL**: Special Trigger Select bit
1 = Selects the special trigger from CTMU
0 = Selects the special trigger from CCP5

bit 6-4 **Unimplemented:** Read as '0'

bit 3-2 **PVCFG<1:0>:** Positive Voltage Reference Configuration bits
00 = A/D VREF+ connected to internal signal, AVDD
01 = A/D VREF+ connected to external pin, VREF+
10 = A/D VREF+ connected to internal signal, FVR BUF2
11 = Reserved (by default, A/D VREF+ connected to internal signal, AVDD)

bit 1-0 **NVCFG<1:0>:** Negative Voltage Reference Configuration bits
00 = A/D VREF- connected to internal signal, AVSS
01 = A/D VREF- connected to external pin, VREF-
10 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)
11 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **ADFM:** A/D Conversion Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 **Unimplemented:** Read as '0'

bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/DONE bit is set until conversions begins.
000 = 0(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)

Note 1: When the A/D clock source is selected as FRC then the start of conversion is delayed by one instruction cycle after the GO/DONE bit is set to allow the SLEEP instruction to be executed.