Microcontrollori

Prof. Franco Zappa

Lenzi Francesco

AA 2020-2021

Indice

1 EMBEDDED SYSTEMS

Questo capitolo introduce il concetto di **Sistemi Embedded** e **Microcontrollori**, insieme ad alcuni concetti correlati, come schede di sviluppo, esigenze di debug e strumenti di compilazione e linker. Fornisce inoltre alcuni suggerimenti sui flussi di programma basati su polling e interrupt e su sistemi multi-tasking e real-time. Vengono inoltre menzionati i *benchmark* appropriati per il confronto delle prestazioni.

1.1 ASIC, FPGA e Microcontrollori

I sistemi *embedded* sono progettati e realizzati per eseguire alcune attività specifiche, piuttosto che essere un computer generico per più attività. Alcuni hanno anche vincoli di prestazioni in tempo reale che devono essere rispettati, per ragioni quali sicurezza e usabilità; altri possono avere requisiti di prestazioni bassi o nulli, consentendo di semplificare notevolmente l'hardware del sistema per ridurre i costi.
I dispositivi *embedded* semplici utilizzano pulsanti, LED, LCD grafici o a caratteri con un semplice sistema di menu. Dispositivi più sofisticati, che utilizzano uno schermo grafico con sensori tattili o pulsanti sul bordo dello schermo, offrono flessibilità riducendo al minimo lo spazio utilizzato: il significato dei pulsanti può cambiare con lo schermo e la selezione implica il comportamento naturale di puntare a ciò che si desidera.
Le proprietà tipiche che un sistema *embedded* deve fornire, se confrontato con le controparti per uso generale, sono il basso consumo energetico, le dimensioni ridotte, i range operativi robusti e il basso costo per unità. Ciò avviene al prezzo di risorse di elaborazione limitate, che le rendono significativamente più difficili da programmare. Il vantaggio è che l'hardware è standard ed è disponibile una vasta gamma di software, inclusi i sistemi operativi. D'altra parte, i sistemi sono grandi e consumano molta energia. La loro affidabilità può anche essere discutibile.

Tra questi estremi, si possono adottare tre approcci generali per sistemi embedded ragionevolmente complessi:

- **ASIC**: (Application-Specific Integrated Circuits) appositamente progettati per una particolare applicazione come suggerisce il nome, forniscono le migliori prestazioni ma sono estremamente costosi da progettare, produrre e testare. Questo li limita ad applicazioni di volume molto grande o dove le prestazioni devono essere acquistate a qualsiasi prezzo.

- **FPGA**: (Field-Programmable Gate Array) e Programmable Logic Devices (PLD), essenzialmente un circuito integrato confezionato con una serie di porte e flip-flop, che possono essere collegati programmando il dispositivo per produrre la funzione desiderata. Questo viene specificato utilizzando un linguaggio di descrizione dell'hardware come VHDL o Verilog. I dispositivi logici programmabili più vecchi contengono una serie di flip-flop, i cui ingressi provengono da un array di porte AND e OR. Sono spesso utilizzati per fornire la logica *"colla"* necessaria per supportare un processore di grandi dimensioni. Gli FPGA hanno una struttura più versatile e possono essere enormi, con oltre un miliardo di transistor.

- **µc**: (Microcontroller), che hardware quasi fisso costruito attorno a un'unità di elaborazione centrale (CPU). La CPU controlla una gamma di periferiche, che possono fornire funzioni sia digitali che analogiche come timer e convertitori analogico-digitale. I dispositivi di piccole dimensioni di solito includono memoria volatile e non volatile sul chip, ma i processori più grandi potrebbero richiedere una memoria separata. Il loro funzionamento è solitamente programmato utilizzando un linguaggio macchina come Assembly o un linguaggio di alto livello come C.

Nel seguito, considereremo come sistema embedded una scheda elettronica a circuito elettronico che contiene un microprocessore (µP) o un microcontrollore (µC) e altri dispositivi e circuiti integrati (IC), che esegue un programma software e viene utilizzata per uno scopo specifico piuttosto che di uso generale.
Un microprocessore è un chip di silicio contenente un'unità di elaborazione centrale (CPU) avanzata che preleva le istruzioni del programma da una memoria esterna e le esegue.
Un microcontrollore contiene invece un microprocessore e una serie di dispositivi periferici interni come timer, porte seriali, pin di input/output (I/O) generici, contatori, convertitori analogico-digitale (ADC), memoria di programma, memoria dati, ecc... tutto all'interno di un singolo chip di silicio. I microcontrollori sono i possibili candidati per i sistemi embedded, perché le loro periferiche e la loro memoria su chip consentono al progettista di sistemi embedded di risparmiare spazio sul circuito e dimensioni complessive, non dovendo aggiungere queste cose sulla scheda.

Il software deve essere scritto in un linguaggio di alto livello adatto all'uomo invece del codice digitale a livello di bit che un microcontrollore può comprendere ed eseguire. A tal fine esistono i compilatori, ovvero dei software che consentono al progettista di scrivere il programma in modo testuale e quindi di tradurlo negli stessi op-code da conservare all'interno della memoria programmi.

1.2 Microcontrollori

Le caratteristiche essenziali di un microcontrollore sono le seguenti:

- **Central Processing Unit**: include una **Arithmetic Logic Unit** (ALU) per i calcoli; **registri** per i dati della CPU, e uno *status register* (SR); altri registri temporanei; ***instruction decoder*** e altri logiche per controllare la CPU, gestire i **resets**, e **interrupts**, e così via.

- **Program Memory**: non-volatile (*Read Only Memory*, ROM), ovvero che mantiene i dati una volta che l'alimentazione viene rimossa.

- **Data Memory**: volatile (*Random Acces Memory*, RAM), il che significa che perde il suo contenuto quando l'alimentazione è spenta; è utilizzata per immagazzinare e trasferire velocemente dati temporanei fra la CPU e i programmi del microcontrollori.

- **Input e Output ports**: per fornire la comunicazione digitale con il mondo esterno.

- **Address and data buses**: per collegare questi sottosistemi per trasferire dati e istruzioni.

- **Clock**: per stare al passo con l'intero sistema sincronizzato; può essere generato internamente o ottenuto da un oscillatore a cristallo o da una sorgente esterna.

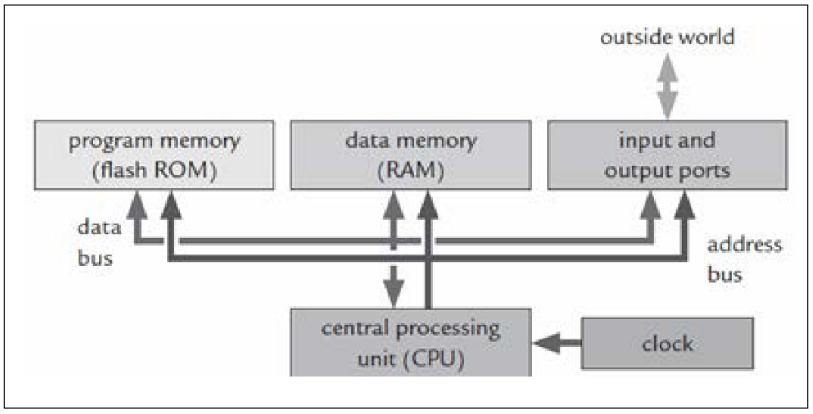


Figura 1: Tipica architettura di un generico Microcontrollore

1.2.1 Periferiche Interne

In un tipico progetto di microcontrollore, il processore occupa solo una piccola parte dell'area di silicio. Le altre aree sono occupate da memorie, generazione di clock (es. PLL) e logica di distribuzione, bus di sistema e periferiche (unità hardware come unità di interfaccia I/O, interfaccia di comunicazione, timer, ADC, DAC, ecc.) come mostrato nella seguente figura.

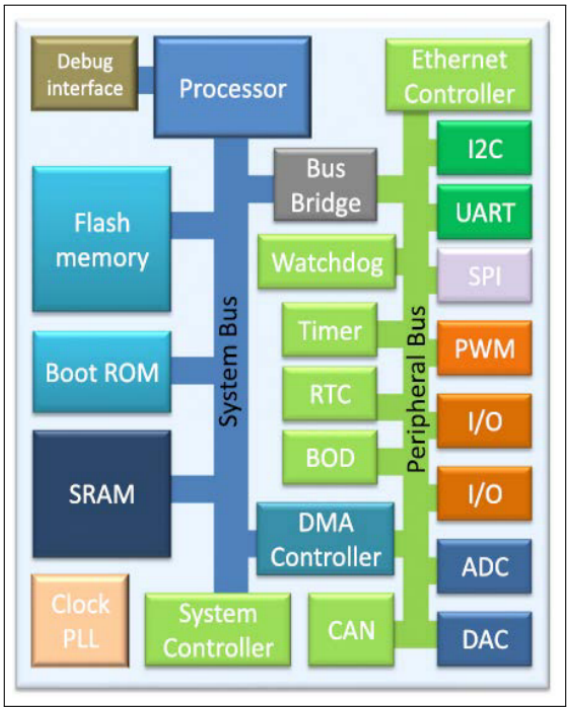


Figura 2: Tipico Diagramma a Blocchi di un microcontrollore

1.2.2 Memorie

La memoria è al centro di qualsiasi computer. Può essere immaginato come una pila di cassetti. Ogni posizione può tipicamente memorizzare 1 byte di dati ed è spesso chiamata *registro*, sebbene questo termine sia a volte riservato alle memorie all'interno della CPU.
La memoria è collegata alla CPU tramite bus per dati, indirizzo e controllo come mostrato in *Figura 1.* L'accesso al bus deve essere controllato, per definire quando i dati sono validi e garantire che due componenti non provino a scrivere contemporaneamente . Il numero di fili in un bus definisce la sua larghezza e i processori sono comunemente caratterizzati dalla larghezza del bus dati, che è generalmente uguale alla dimensione dei dati che possono essere elaborati dalla CPU.
Le memorie possono essere classificate come:

- **Volatile**: perde il suo contenuto quando viene spento. Di solito è chiamata memoria ad accesso casuale o RAM, ma il nome è fuorviante perché l'accesso alla maggior parte degli altri tipi di memoria è ugualmente casuale. La caratteristica fondamentale è che i dati possono essere letti o scritti con la stessa facilità. La memoria volatile viene utilizzata per i dati e i piccoli microcontrollori spesso hanno pochissima RAM, a volte solo poche decine di byte. La memoria è solitamente RAM statica, il che significa che conserva i suoi dati anche se il *clock* viene fermato. La RAM quindi occupa una vasta area di silicio, il che la rende costosa. La maggior parte della memoria in un computer desktop è RAM dinamica.

- **Non-Volatile**: conserva il suo contenuto allo spegnimento ed è quindi utilizzato per il programma e i dati costanti. Di solito è chiamata memoria di sola lettura o ROM, ma ancora una volta questo nome tradizionale è diventato fuorviante. La maggior parte dei moderni microcontrollori può scrivere sulla propria memoria non volatile, ma è molto più lenta e complicata rispetto alla scrittura sulla RAM.
  Esistono molti tipi di memoria non volatile in uso:

  - **Masked ROM**: I dati vengono codificati in una delle maschere utilizzate per la fotolitografia e scritti nell'IC durante la fabbricazione. Questa memoria è davvero di sola lettura. Viene utilizzato per la produzione ad alto volume di prodotti stabili perché qualsiasi modifica ai dati richiede la produzione di una nuova maschera con grandi spese.
  - **EPROM**: essendo una *Electrically Programmable ROM*, può essere programmata elettricamente, ma non cancellata. I dispositivi devono essere esposti alla luce ultravioletta (UV) per circa dieci minuti per cancellarli. Il solito incapsulamento epossidico nero è opaco, quindi i dispositivi cancellabili richiedono pacchetti speciali con finestre al quarzo, che sono costosi. Questi sono stati ampiamente utilizzati per lo sviluppo prima che la memoria flash fosse ampiamente disponibile.
  - **OTP**: (*one-time programmable memory*) questa è solo EPROM in un pacchetto normale senza finestra, il che significa che non può essere cancellata.
  - **Memorie Flash**: può essere sia programmata che cancellata elettricamente ed è di gran lunga la memoria più comune, avendo sostituito la ROM programmabile e cancellabile elettricamente (EEPROM). La differenza pratica è che i singoli byte di EEPROM possono essere cancellati ma la flash può essere cancellata solo a blocchi.

1.2.3 Architetture Harvard e Von Neumann

I due tipi di memoria che abbiamo appena esaminato, volatile e non volatile, possono essere trattati nei due modi generali illustrati. I processori generici utilizzano quasi esclusivamente l'architettura di von Neumann, ma entrambi sono utilizzati nei microcontrollori.

- Architettura Harvard
  Le memorie volatile (*data*) e non volatile (*program*) sono trattate come sistemi separati, ciascuno con il proprio indirizzo e bus dati. Il vantaggio principale è l'efficienza, perchè consente l'accesso simultaneo al programma e alle memorie dati. Ad esempio, la CPU può leggere un operando dalla memoria dati mentre legge l'istruzione successiva dalla memoria di programma. I due sistemi possono essere ottimizzati separatamente. Un problema con l'architettura di Harvard è che i dati costanti devono essere archiviati nella memoria programmi perché non è volatile. Ciò significa che le costanti non possono essere lette allo stesso modo dei valori volatili dalla memoria dati. Quindi devono essere fornite speciali istruzioni di "lettura della tabella" o parte della memoria programmi è mappata nella memoria dei dati.

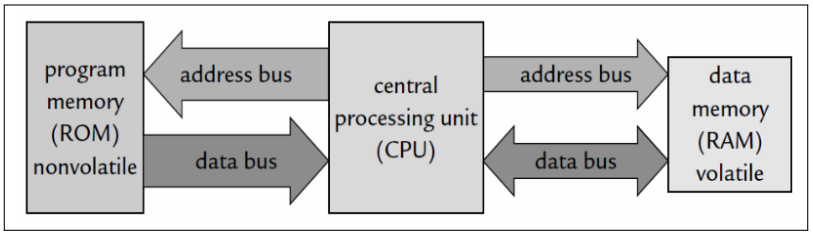


Figura 3: Architettura Harvard

- Architettura Von Neumann
  Esiste un solo sistema di memoria nell'architettura von Neumann o Princeton, quindi solo un insieme di indirizzi copre sia le memorie volatili che non volatili. Particolare importanza assume la mappa della memoria, che mostra gli indirizzi ai quali si trova ciascun tipo di memoria. L'architettura è intrinsecamente meno efficiente perché potrebbero essere necessari diversi cicli di memoria per estrarre un'istruzione completa dalla memoria.

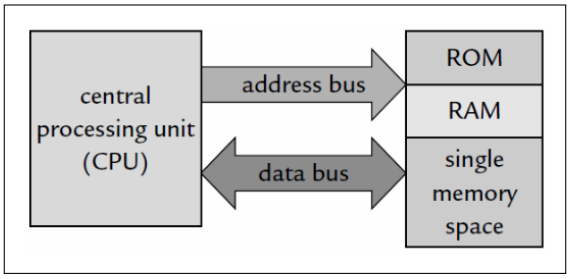


Figura 4: Achitettura Von Neumann

1.2.4 Instruction Set

CISC (*Complex Instruction Set Computer*)

- Calcoli complessi in hardware, meno complessi a livello software;
- Istruzioni complesse o molte diverse modalità di indirizzamento, tutte all'interno di singole istruzioni;
- Istruzioni altamente codificate per migliorare la densità del codice, ridurre le dimensioni del programma e richiedere meno accessi alla memoria principale;
- Grande vantaggio quando le memorie e l'accesso al disco erano costosi e bassi

RISC (*Reduces Instruction Set Computer*)

- Istruzioni molto semplici, che vengono eseguite all'interno di un singolo ciclo, ad alta velocità di clock. Ridotta complessità delle istruzioni eseguite in hardware.
- Intelligenza e flessibilità passano al livello software superiore.
- Un hardware molto veloce e un firmware molto efficiente sono un must

1.3 Program *Flow*

1.3.1 Polling

Per applicazioni molto semplici, il processore può attendere finché non ci sono dati pronti per l'elaborazione, elaborarli e quindi attendere di nuovo. Questo è molto facile da configurare e funziona bene per compiti semplici. Nella maggior parte dei casi, un microcontrollore dovrà servire più interfacce e quindi processi. Il metodo del flusso del programma di polling può essere ampliato per supportare facilmente più processi. Questa disposizione è talvolta chiamata "super-loop". Il metodo di polling funziona bene per applicazioni semplici, ma presenta diversi svantaggi.
Ad esempio, quando l'applicazione diventa più complessa, la progettazione del ciclo di polling potrebbe diventare molto difficile da mantenere. Inoltre, è difficile definire le priorità tra i diversi servizi utilizzando il polling e potresti finire con una scarsa reattività, in cui un servizio che richiede una periferica potrebbe dover attendere molto tempo mentre il processore sta gestendo attività meno importanti. Un altro principale svantaggio del metodo di polling è che non è efficiente dal punto di vista energetico.

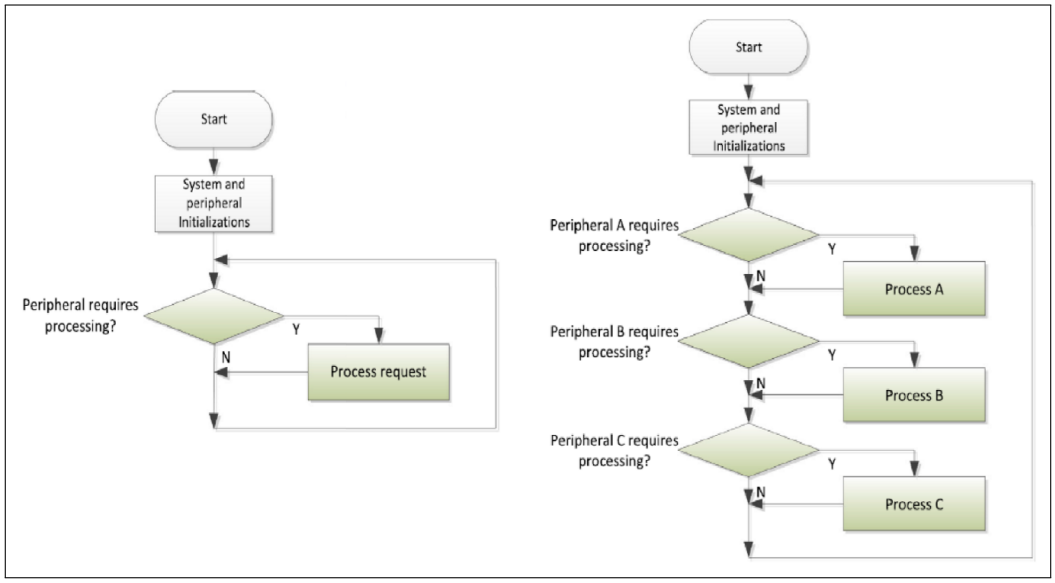


Figura 5: Polling Semplice (SX) e Polling con device multipli (DX)

1.3.2 Interrupt

Per risolvere questo problema, quasi tutti i microcontrollori hanno una sorta di supporto per la modalità di sospensione per ridurre la potenza, in cui la periferica può riattivare il processore quando richiede un servizio. Questo è comunemente noto come un'applicazione guidata da *interrupt*, in cui gli interrupt da diverse periferiche possono essere assegnati con diversi livelli di priorità di interrupt. Ad esempio, le periferiche importanti/critiche possono essere assegnate con un livello di priorità più alto in modo che, quando l'interrupt arriva, l'esecuzione del servizio di interrupt a priorità inferiore viene sospesa, consentendo l'avvio immediato del servizio di interrupt a priorità più alta. Questa disposizione consente una reattività migliore.
In alcuni casi, il trattamento dei dati dei servizi periferici può essere suddiviso in due parti: la prima parte deve essere eseguita rapidamente e la seconda può essere eseguita un po' più tardi. In una situazione del genere, possiamo utilizzare una combinazione di metodi basati su interrupt e di polling per costruire il programma.
Quando una periferica richiede un servizio, attiva una richiesta di interrupt come in un'applicazione basata su interruzioni. Una volta eseguita la prima parte del servizio di interrupt, aggiorna alcune variabili software in modo che la seconda parte del servizio possa essere eseguita nel codice dell'applicazione basata su polling.
Utilizzando questa disposizione, possiamo ridurre la durata dei gestori di interrupt ad alta priorità in modo che i servizi di interrupt con priorità inferiore possano essere serviti più rapidamente. Allo stesso tempo, il processore può ancora entrare in modalità di sospensione per risparmiare energia quando non è necessaria alcuna manutenzione.

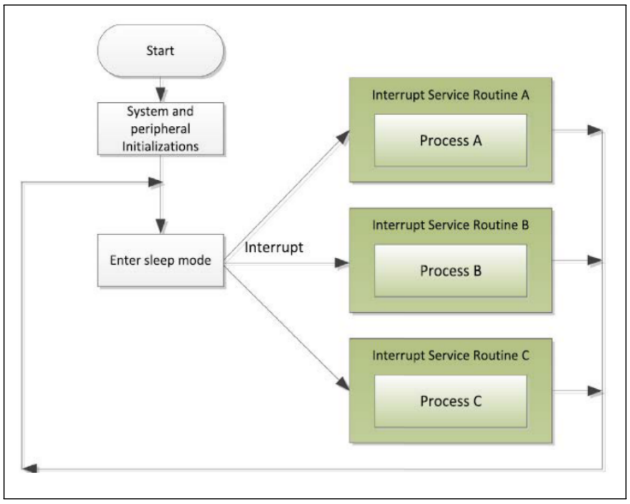


Figura 6: Semplice applicazione guidata da Interrupt

1.3.3 Benchmarks

In informatica, un benchmark è l'atto di eseguire un programma per computer, un insieme di programmi o altre operazioni, al fine di valutare le prestazioni relative di un oggetto, normalmente eseguendo una serie di test e prove standard contro di esso. Questo termine è anche utilizzato principalmente ai fini degli stessi programmi di benchmarking elaborati. Il benchmarking è solitamente associato alla valutazione delle caratteristiche prestazionali dell'hardware del computer, ad esempio le prestazioni operative in *floating-point* di una CPU, ma ci sono circostanze in cui la tecnica è applicabile anche al software.

- **MIPS** (Mega Instructions Per Second)
  La **velocità del processore** del computer (tassi di esecuzione di picco su istruzioni con pochi rami).
  Un carico di lavoro realistico in genere riduce i MIPS effettivi.
  I MIPS sono utili quando si confrontano le prestazioni tra µC con architetture simili, ma non è giusto confrontare tra diverse architetture di CPU.

- **FLOPS** (Floating-Point Operations Per Second)
  Prestazioni del computer utili nei campi dei calcoli scientifici che fanno un uso massiccio di calcoli in virgola mobile, misura più accurata rispetto alle istruzioni generiche al secondo.

- **Drystone**
  Benchmark di calcolo sintetico, che non contiene operazioni in virgola mobile.
  È il numero di iterazioni del ciclo di codice principale al secondo.

- **CoreMark**
  Prende di mira il core della CPU, simile a Dhrystone.
  CoreMark utilizza algoritmi reali piuttosto che essere sintetici.
  Entrambi sono gratuiti ed eseguiti su qualsiasi processore, compresi i piccoli microcontrollori.

2 MICROCONTROLLORI A 8-BIT

2.1 Memoria Programmi

In un µC, la memoria programmi contiene il firmware che il µC eseguirà, quindi questa memoria non dovrebbe essere volatile. Infatti, la memoria programmi può essere:

- ROM (cioè "programmata a maschera", per volumi superiori alle migliaia di pezzi);
- OTP (conveniente per produzioni di piccoli volumi, inferiori a poche centinaia di pezzi);
- EPROM (per modificare rapidamente il codice, quando è necessario eseguire il debug del codice o abilitare gli aggiornamenti del firmware, tramite In-System Programming).

La memoria programmi in genere non è molto ampia, da 1k a 4k circa, con parole di 8 bit (come nei µC della STMicroelectronics) o di 14 bit (come nei PIC di Microchip). Solitamente un op-code più lungo consente di scrivere operazioni più complesse in un unico op-code, cioè in una singola cella di memoria, anche se non si traduce direttamente in una esecuzione più rapida delle istruzioni rispetto alla soluzione a 8 bit, che invece spesso ha bisogno di due celle di memoria per memorizzare il codice operativo.

L'indirizzo della cella di memoria contenente l'istruzione da eseguire è memorizzata nel ***Program Counter*** (PC), che è un contatore in avanti che viene incrementato dopo che ogni istruzione è stata scaricata, ma che può anche essere caricato con un valore arbitrario, ad esempio nel caso di un salto o di una chiamata a un sottoprogramma. Rispetto ad un salto, la chiamata ad una subroutine richiede di memorizzare il valore attuale del PC, prima di caricare l'indirizzo della subroutine da eseguire, perché una volta completata la routine il µC deve tornare all'indirizzo dove era stata interrotta. La memoria utilizzata a tale scopo, per memorizzare e recuperare il contenuto del PC è detta *stack* ed è una memoria RAM.

Solitamente, la prima cella della memoria programmi viene utilizzata per definire l'indirizzo iniziale della prima riga effettiva del firmware: l'indirizzo `0000` è chiamato vettore di *Reset* e conterrà un salto alla riga della prima istruzione. Un'altra regione della memoria programmi è chiamata vettore di *interrupt* e contiene il salto alla routine che il µC deve eseguire quando viene attivato un evento asincrono (un interrupt). Infatti, ogni volta che viene attivato un interrupt all'interno del µC, l'istruzione in corso viene interrotta (di solito è completata), quindi il Program Counter viene salvato nello stack e viene eseguito un salto alla cella del vettore di interrupt corrispondente per eseguire l'istruzione. Una volta completato, il core µC recupera il valore memorizzato nello stack e lo ricopia sul PC, recuperando così il flusso del programma dal punto in cui è stato interrotto.

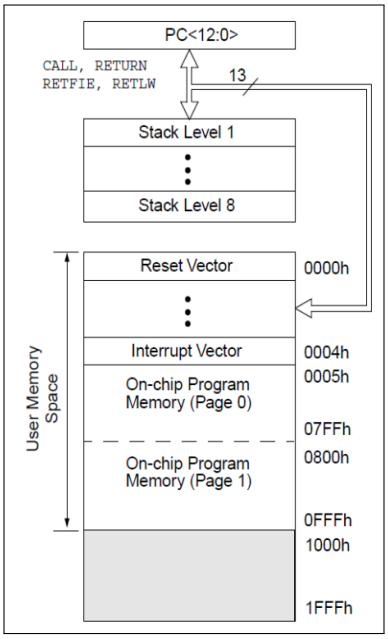


Figura 7: Memoria Programmi e Stack

Ogni cella contiene un codice operativo, largo 14 bit per la famiglia **PIC16Cxx**. Pertanto, un singolo codice operativo non può contenere un salto a qualsiasi altra cella nell'intervallo 8k, poiché ciò richiederà 13 bit dei 14 bit disponibili solo per specificare l'indirizzo desiderato, senza lasciare spazio per codificare il tipo di istruzione. Al fine di risparmiare area di silicio e, eventualmente, ridurre il costo del chip µC, Microchip ha deciso di limitare la larghezza del codice operativo a 14 bit e di fornire la memoria programmi proprio come "pagine" di 2k celle ciascuna. In questo modo sono sufficienti 11 bit per indirizzare una cella all'interno di una pagina, in modo che le istruzioni di salto possano occupare un solo op-code e anche il PC possa essere più corto.

2.2 Memoria Dati

Questa memoria viene solitamente utilizzata per contenere i risultati temporanei dei calcoli eseguiti dal µC, o valori numerici acquisiti dalle porte di ingresso, o forniti dalle periferiche interne, come l'ADC, il timer interno, le interfacce di comunicazione seriale, ecc...
Di solito i µC di fascia bassa forniscono una memoria dati molto piccola, di poche decine o centinaia di byte. La memoria dati deve essere letta e scritta molto spesso e molto rapidamente, quindi si basa sulla RAM, sia statica che dinamica. La RAM statica occupa un'area di silicio più ampia, quindi il µC è più costoso, ma il suo accesso è veloce (lettura e scrittura richiedono poche decine di ns) e conserva i dati anche se il clock dell'µC si ferma.
Invece, la RAM dinamica occupa meno area, quindi il µC può essere più economico o può contenere più memoria dati per la stessa area di silicio e lo stesso costo, ma è più lento e richiede un aggiornamento periodico. Questo compito è svolto dal core µC, in background, quindi è completamente trasparente per i progettisti di firmware e hardware. Inoltre, molti µC offrono la possibilità di entrare in uno stato di "sleep", in cui quasi tutte le periferiche on chip e l'elettronica ausiliaria vengono spente per ridurre al minimo il consumo di corrente. In questo caso, un µC

con RAM dinamica consumerebbe più energia, poiché dovrebbe mantenere in funzione il circuito di aggiornamento.

Nei microcontrollori PIC, la memoria dati è suddivisa in **banchi**, che contengono i ***General Purpose Register*** e gli ***Special Function Register***. I bit RP1 e RPO sono i bit di selezione del *Banco*, e sono rispettivamente il bit6 e il bit5 dello `STATUS` *register*: le combinazioni possibili sono `00`-Bank0, `01`-Bank1; `10`-Bank2; `11`-Bank3.
Ogni banco si estende fino a $7F_H$ (128 byte). Le posizioni inferiori di ciascun banco sono riservati agli *Special Function Registers* (SFRS). Alcuni SFRS "ad alto utilizzo" di una banca possono essere replicati in un altro *Banco* per la riduzione del codice e un accesso più rapido.

Si noti che solo i registri generici sono implementati come una vera memoria RAM, disposta in un'area ben definita e localizzata del chip µC. Invece gli *Special Function Register* sono insiemi individuali di 8 flip-flop (solitamente) spazialmente adiacenti l'uno all'altro o addirittura sparsi nel layout del chip , ma indirizzate come se fossero un insieme contiguo di celle. Questa implementazione riduce efficacemente il numero di istruzioni e semplifica il nucleo dell'µC, perché è necessaria una sola istruzione per indirizzare una cella generica o il contenuto di un timer, il risultato dell'ADC, il contenuto di qualsiasi porta I/O, ecc., semplicemente specificando il suo indirizzo di file (come se fossero tutte celle di memoria RAM contigue), invece di utilizzare un'istruzione dedicata per periferica e un'altra istruzione per la memoria RAM.

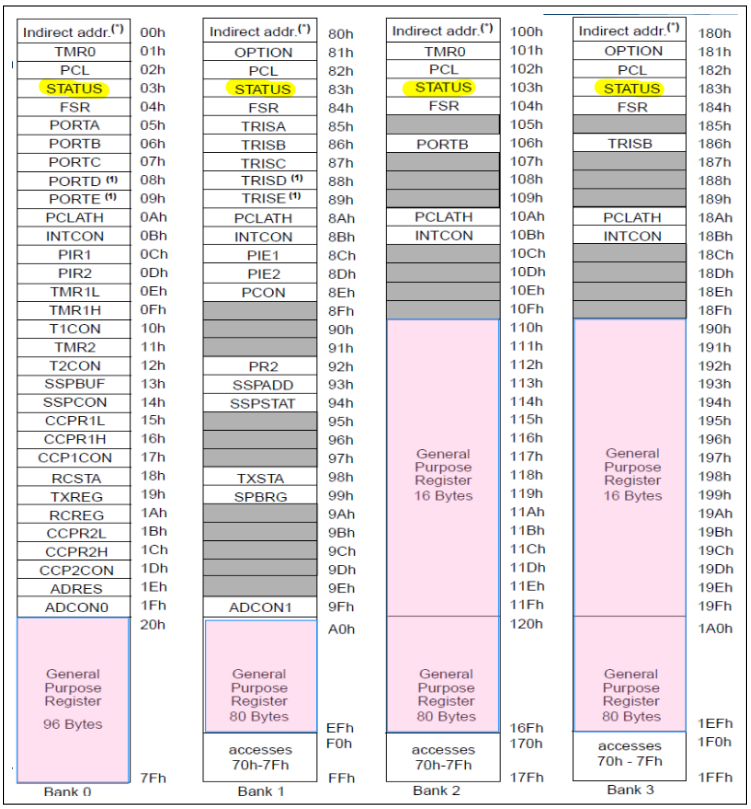


Figura 8: Memoria Dati

Come già fatto per le pagine in memoria programma, anche per la memoria dati il Costrut-

tore µC può decidere di implementare solo uno o due o qualunque numero di banchi possa essere necessario, a seconda delle esigenze del cliente. Dal punto di vista del progettista microelettronico, deve solo posizionare il numero desiderato di banchi RAM, quindi abilitarne solo uno alla volta, semplicemente decodificando il contenuto dei bit di `Bank Select`, se non abbastanza deve aggiungere un altro bit di selezione del banco e quindi indirizzare le linee di abilitazione ad altri *SPRS* della periferica su chip, se necessario.

2.2.1 Special Function Registers

Il **General Purpose Register file** è una piccola quantità di spazio di archiviazione che può essere accessibile più rapidamente di qualsiasi altra memoria. È possibile accedere al file di registro direttamente o indirettamente tramite il File Select Register FSR.

Gli **Special Function Registers** sono anche registri di memoria utilizzati per speciali funzioni dedicate. Questi registri svolgono varie funzioni dedicate all'interno del chip PIC: ogni funzione speciale all'interno di questo chip PIC è controllata utilizzando questi registri, poiché quasi ogni bit di questi registri è utilizzato dalla CPU e dai moduli periferici per controllare il funzionamento desiderato del µC complessivo. Tutti questi registri sono implementati come RAM statica.

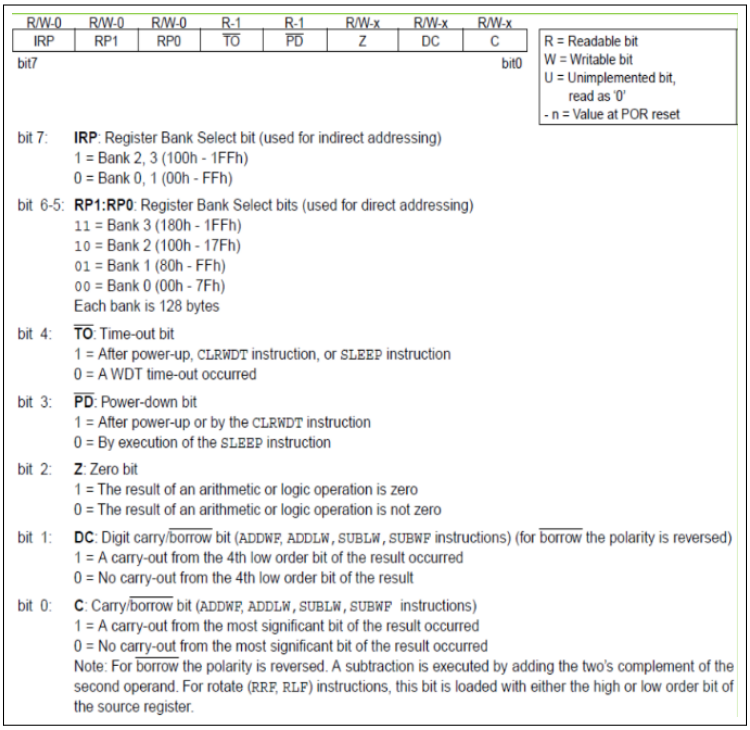
`STATUS` register

Lo STATUS register contiene lo stato aritmetico dell'ALU, lo stato di RESET e i bit di selezione del banco per la memoria dati. Ecco una legenda:

- R = bit leggibile;
- W = bit scrivibile;
- U = bit non implementato(letto come '0');
- -n = valore al ripristino dell'accensione;
- '1' = bit è impostato;
- '0' = bit viene cancellato;
- x = bit sconosciuto.

Ad esempio, "R/W-0" significa che il bit può essere letto o scritto e, dopo il ripristino, il suo contenuto viene reimpostato a 0.

Lo STATUS register può essere la destinazione di qualsiasi istruzione, come di qualsiasi altro registro. Tuttavia, essendo un registro che contiene alcuni bit interessati dall'ALU, se il registro STATUS è la destinazione di un'istruzione che interessa i bit Z, DC o C, allora la scrittura su questi tre bit è disabilitata. Questi bit vengono impostati o cancellati in base alla logica del dispositivo. Inoltre, i bit TO e PD non sono scrivibili, pertanto il risultato di un'istruzione con il registro STATUS come destinazione potrebbe essere diverso da quello previsto.



R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4: **$\overline{TO}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3: **$\overline{PD}$**: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Figura 9: STATUS register

`OPTION` register

L'`OPTION` register è leggibile e scrivibile. Contiene vari bit di controllo per configurare il prescaler TIMER 0 (TMR0) e il postscaler WATCHDOG (WDT) (registro singolo assegnabile noto anche come prescaler), le transizioni di tipo External INT Interrupt in grado di generare l'interrupt, e la presenza o meno dei deboli pull-up interni su PORTB. Si noti che per ottenere un'assegnazione del prescaler 1:1 per il registro TMR0, il prescaler deve essere assegnato al Watchdog.

2.3 Istruzioni in Assembler

A questo punto conviene vedere le istruzioni disponibili in un µC a 8 bit, come nella famiglia PIC16Cxx considerata finora. Come vedremo, è un set di istruzioni molto semplice, ma abbastanza potente ed efficiente per controllare la maggior parte delle applicazioni destinate a questo tipo di µC. Ogni set di istruzioni PIC16CXX ha solo 35 istruzioni di 14 bit ciascuna, suddivise in un **OPCODE**, che specifica il tipo di istruzione, e uno o più operandi, che specificano ulteriormente il funzionamento dell'istruzione. Il riepilogo del set di istruzioni PIC16CXX nella prossima figura elenca le operazioni orientate ai byte, ai bit, letterali e di controllo. Tutte le istruzioni vengono eseguite all'interno di un singolo ciclo di istruzioni, a meno che un test condizionale non sia vero o il Program Counter non venga modificato a seguito di un'istruzione. In questo caso, l'esecuzione richiede due cicli di istruzione con il secondo ciclo eseguito come NOP.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected
				MSb			LSb	
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
MOVWF	f	Move W to f	1	00	0000	lfff	ffff	
NOP	-	No Operation	1	00	0000	0xx0	0000	
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	
LITERAL AND CONTROL OPERATIONS								
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk	
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk	
RETFIE	-	Return from interrupt	2	00	0000	0000	1001	
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk	
RETURN	-	Return from Subroutine	2	00	0000	0000	1000	
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z

Figura 10: Instruction Set Assembly

2.3.1 Modalità di Indirizzamento

Per indirizzare una determinata posizione nella RAM è possibile specificare l'indirizzo all'interno dell'*op-code*: questo si chiama **indirizzamento diretto**. Tuttavia, se si dovesse scansionare un'area di RAM senza conoscerne a priori l'estensione o il punto di partenza o di arrivo, sarebbe utile disporre di un modo diverso di indirizzamento, chiamato modalità di **indirizzamento indiretto**.
Ad esempio, questo è importante quando si desidera applicare la stessa elaborazione dati a celle diverse nella RAM, quindi non si desidera scrivere una parte del codice per un blocco dati specifico (es. `MOVLW 0x22; ADD...`) e poi lo stesso codice per un altro blocco dati, dopo aver cambiato tutti gli indirizzi (es. `MOVLW 0x29; ADD...`), e così via. La soluzione migliore dovrebbe essere quella di avere un registro contenente la cella di interesse attuale (ad es. `0x22`) ed eseguire il codice, quindi modificare il valore del registro (ad es. `0x29`) ed eseguire nuovamente lo stesso codice e così via.

Nella maggior parte dei microcontrollori ci sono registri specifici per questo scopo. Ad esempio, i PIC µC hanno un registro speciale chiamato File Select Register (FSR), che può essere scritto, letto, aumentato o diminuito a piacimento, il cui indirizzo di memoria è 04_H, indipendentemente dal Banco.
Per accedere alla locazione di memoria puntata da questo registro è sufficiente fare riferimento al registro fittizio `INDF`. Richiamandolo, l'µC utilizzerà il contenuto di FSR per accedere alla posizione nel file di registro. Infatti il registro `INDF` non è un registro fisico: l'indirizzamento dell'`INDF` provocherà l'**indirizzamento indiretto**. Pertanto, qualsiasi istruzione che utilizza il registro `INDF` accede effettivamente al registro puntato da FSR.

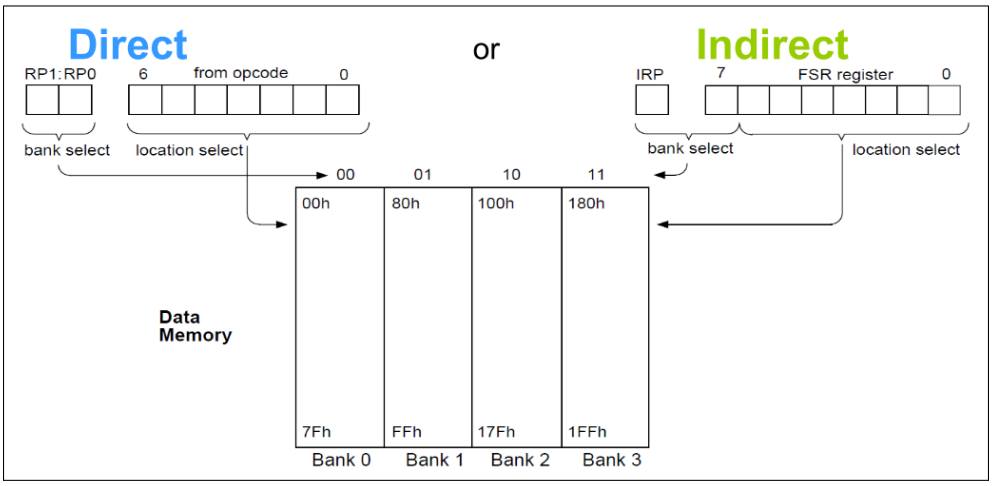


Figura 11: Modalità di indirizzamento Diretto e Indiretto

2.3.2 Ciclo delle istruzioni

Le operazioni del microcontrollore sono temporizzate da un clock, poiché si tratta di una macchina microprogrammata e, in definitiva, di un processore sequenziale. Non è vero che per ogni periodo di clock viene completata un'intera operazione. Ad esempio, nel PIC, in ogni periodo di clock viene eseguita solo una fase di un'operazione. L'ingresso del clock (da *OSC1*) è diviso internamente per quattro per generare quattro clock in quadratura non sovrapposti, ovvero Q1, Q2, Q3 e Q4. Internamente, il Program Counter (PC) viene incrementato ogni Q1, l'istruzione viene prelevata dalla memoria programmi e bloccata nel registro delle istruzioni. Quindi, l'istruzione viene decodificata ed eseguita.
Le istruzioni di recupero ed esecuzione sono condotte in modo tale che il recupero richieda un ciclo di istruzioni mentre la decodifica e l'esecuzione richiedono un altro ciclo di istruzioni. Tuttavia, a causa del *pipelining*, ogni istruzione viene effettivamente eseguita in un ciclo (ovvero 4 impulsi di clock). Un ciclo di recupero inizia con l'incremento del Program Counter (PC) in Q1. Nel ciclo di esecuzione, l'istruzione prelevata viene agganciata al "Registro istruzioni" nel ciclo Q1. Questa istruzione viene quindi decodificata ed eseguita durante i cicli Q2, Q3 e Q4. La memoria dati viene letta durante Q2 (lettura operando) e scritta durante Q4 (scrittura destinazione).

Se un'istruzione provoca la modifica del Program Counter (es. GOTO) allora sono necessari due cicli per completare l'istruzione. Infatti, tutte le istruzioni sono a ciclo singolo, ad eccezione di eventuali rami di programma. Questi richiedono due cicli poiché l'istruzione di recupero viene "svuotata" dalla pipeline mentre la nuova istruzione viene recuperata e quindi eseguita.

Questa funzionalità di pipeline è possibile anche grazie all'architettura Harvard, che fornisce una struttura a doppio bus affinché il core comunichi solo con la memoria programmi, durante il recupero, e solo con la memoria dei dati, durante l'esecuzione. I due bus distinti consentono di accedere contemporaneamente alle istruzioni (Memoria Programmi) e agli operandi o alle periferiche (Registers file).

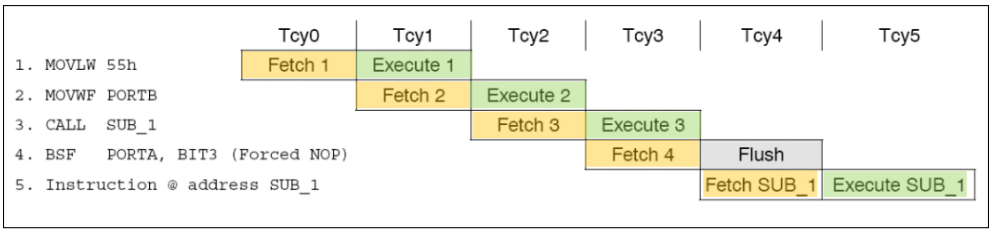


Figura 12: Esecuzione del flusso di istruzioni

2.4 Porte di I/O

Le porte Input/Output sono l'esempio più semplice di interfaccia verso il mondo esterno. Il singolo pin I/O può essere un ingresso (I) o un'uscita (O) del µC a seconda di come il firmware imposta la porta e può essere configurato e utilizzato in diversi modi, riassunti come segue:

- Ingresso Digitale: può essere un normale ingresso CMOS o un trigger Schmidt (che toglie l'incertezza nella commutazione e aumenta il Noise Margin). Alcuni µC offrono la possibilità di avere una resistenza di pull-up interna, permettendo così di lasciare il pin flottante all'esterno.

- Interrupt: simile a un semplice input di dati digitali, ma può anche essere abilitato per generare un interrupt al core µC.

- Ingresso analogico: consente di acquisire segnali analogici e inviarli al convertitore analogico-digitale (ADC) su chip all'interno del µC, per un'ulteriore elaborazione digitale sulla versione numerica del segnale analogico esterno.

- Digital Output: solitamente a livelli CMOS, a volte open-drain (con la sola uscita pull down n-MOS), per abilitare connessioni wired-AND; può anche andare ad alta impedenza rimanendo così flottante.

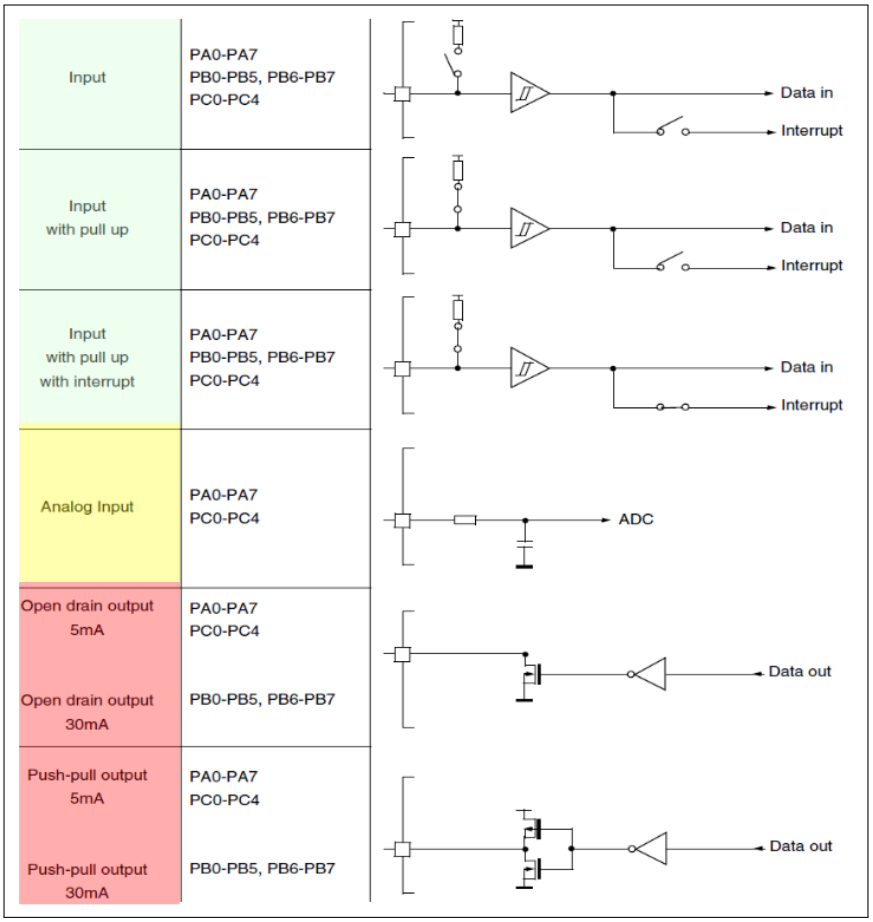


Figura 13: Possibili configurazioni dei pin di I/O

Tutti i registri I/O possono essere letti o scritti come qualsiasi altra posizione RAM nello spazio dati, quindi non sono necessarie celle RAM aggiuntive per l'archiviazione e la manipolazione dei dati della porta. Durante l'inizializzazione del µC, tutti i registri I/O vengono cancellati e viene selezionata la modalità di input con pull-up e nessuna generazione di interrupt per tutti i pin, evitando così conflitti di pin. Sono possibili operazioni a bit singolo sui registri I/0, ma è necessaria attenzione perché la lettura in modalità input viene eseguita dai pin I/O mentre la scrittura influirà direttamente sul registro dati della porta causando una modifica indesiderata della configurazione dell'ingresso.

Un modo semplice per configurare un pin come Input o Output è mostrato nella seguente figura:

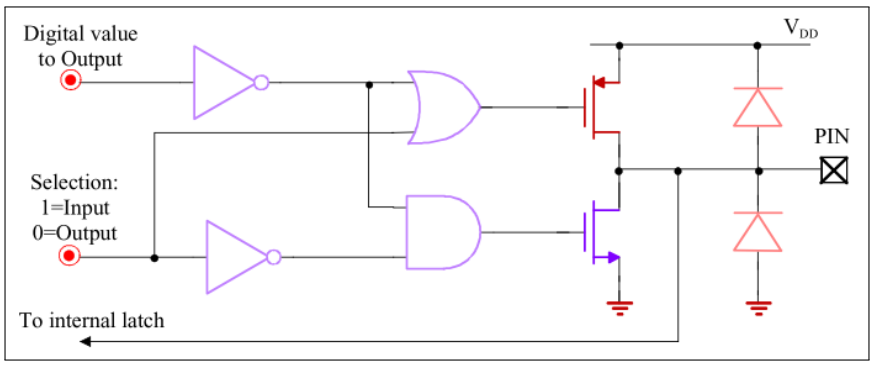


Figura 14: Diagramma di un pin di I/O

Se il *Direction control* è impostato su 1, entrambi i transistor MOS sono spenti, indipendentemente dal valore dell'input, quindi il pin è ad **alta impedenza**(Z) e può essere impostato dall'esterno: il pin può fungere da **ingresso**. Viceversa, se il controllo di Direzione viene mantenuto basso (0), le porte sono trasparenti al valore di ingresso applicato alla porta dal bus dati: il pin funge da **uscita**.

Nei PIC µC di Microchip, la scelta di come utilizzare il pin I/O viene effettuata tramite lo *Special Function Register* chiamato `TRIS`. Se 1 è memorizzato nel latch `TRIS` (ovvero $Q = 1$ e $\bar{Q} = 0$), i transistor MOS di uscita sono spenti e il pin va in tri-state, quindi può fungere da ingresso. Scrivendo invece uno 0 nel latch `TRIS`, il contenuto del latch DATA deve raggiungere il pin, fungendo così da output. Ciascun registro `TRIS` a 8 bit guiderà di conseguenza 8 pin di una porta.

2.5 Interrupts

Quando si verifica un evento asincrono, può essere attivata una Interrupt Request (IRQ) al core del µC, che dovrebbe quindi interrompere l'esecuzione del programma principale e saltare invece ad eseguire una routine dedicata, per accertare quale intervento periferico o utente ha causato l'evento e gestirlo di conseguenza. Le richieste di interrupt possono provenire da più sorgenti:

- da pin di qualsiasi porta I/O, se adeguatamente abilitata;
- dal pin privilegiato Non-Maskable Interrupt (NMI), proveniente dall'esterno del µC;
- da periferiche interne (es. da *ADC*, per segnalare la fine della conversione e la disponibilità dei dati convertiti, o da *timer* per spuntare la fine del conteggio).

Per gestire un interrupt, il µC deve conoscere l'indirizzo nella memoria programmi in cui è memorizzata la routine di interrupt, che può essere diverso a seconda della sorgente che ha attivato l'interrupt. Tale gestione di routine può essere implementata in due modi differenti, tramite un vettore di interrupt o tramite polling.

Vettore di Interrupt

I µC di fascia alta hanno un'area di memoria nella memoria programmi non volatile composta da diverse celle vicine, ognuna delle quali contiene l'indirizzo di una routine di interrupt all'interno della memoria programmi. A seconda della sorgente che attiva la chiamata di interrupt, il nucleo del µC leggerà l'indirizzo a cui saltare, da una specifica posizione del vettore di interrupt, riservata a quell'interrupt. Ogni possibile interrupt o gruppo di interrupts può avere la sua routine firmware specifica in diverse posizioni della memoria programmi che il progettista del firmware può scegliere liberamente. Per evitare che il µC venga interrotto troppo spesso dal verificarsi di diversi interrupt non sovrapposti, solitamente l'architettura interna del µC consente di "mascherare" quelli a bassa priorità. Ad esempio, quando viene eseguita un'attività importante, il firmware può disabilitare le periferiche più lente o meno importanti per attivare il loro interrupt e riattivarle quando il µC torna a compiti meno importanti. Il progettista del firmware ha la possibilità di decidere se mascherare o meno alcune sorgenti di interrupt e anche di assegnare un ordine di priorità tra loro e una certa soglia al di sopra della quale viene servita la sorgente di interrupt.

Polling

I µC di fascia bassa hanno un vettore di interrupt composto dalla sola locazione di memoria programmi, che contiene l'unica prima istruzione da eseguire quando viene attivato un evento di interrupt. Questa istruzione è solitamente un salto all'effettiva prima riga di codice dell'*Interrupt Service Routine* all'interno della memoria programmi, in una posizione scelta dal progettista del firmware, dove è implementata l'unica routine di interrupt.

Per identificare il dispositivo chiamante e quindi servirlo correttamente eseguendo una sottoparte della routine di interrupt globale, il firmware deve verificare quale FLAG è stato generato tra i tanti.

Al fine di consentire la selezione di quale sorgente potrebbe attivare un interrupt, la logica di interrupt consente al progettista del firmware di abilitare o disabilitare ciascuna sorgente di interrupt tramite un bit di abilitazione (etichettato da una "E" finale, lettera che significa "ENABLE"). Inoltre, esiste anche un bit di abilitazione dell'interruzione globale (GIE), che può disabilitare definitivamente tutti i possibili interrupt contemporaneamente. Come si può notare, questa modalità non permette di impostare una priorità primaria tra le sorgenti, ma semplicemente di abilitarle o disabilitarle. Quindi, la priorità di manutenzione può essere gestita eseguendo il polling in un ordine specifico, a seconda di quale può essere posticipato rispetto ad altre. Inoltre, la procedura di polling è decisamente più costosa in termini di tempo di esecuzione e latenza di interrupt rispetto all'approccio del vettore di interrupt, dove non era necessario il polling, perché il core del µC sapeva in anticipo quale routine eseguire a seconda della sorgente chiamante.

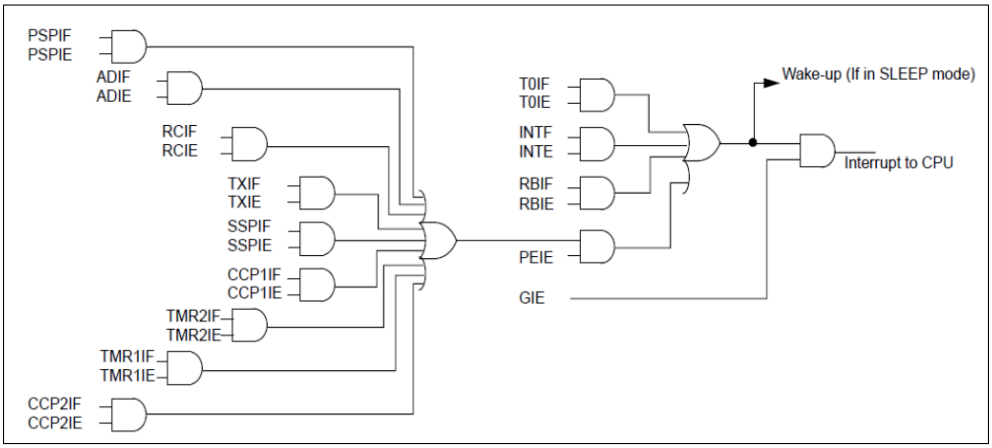


Figura 15: Logica di Interrupt per abilitare (ENABLE) le possibili richieste di interrupt

2.6 Analog-to-Digital Converter ADC

Molti microcontrollori di fascia media hanno un convertitore analogico-digitale interno (di solito con uscita digitale a 8 bit), un multiplexer analogico per selezionare uno di alcuni (solitamente da 4 a 10) pin di ingresso analogico, un Sample&Hold e un selettore per impostare il riferimento dell'intervallo di fondo scala desiderato (V_{REF}).

L'ingresso analogico da acquisire viene selezionato tramite i bit `CHSO:CHS2` tramite switch implementati in hardware da pass-transistor MOS. A volte, anche la tensione di riferimento V_{REF} per il **Fondo Scala** (FSR) dell'ADC può essere impostata dall'esterno, tramite uno degli otto ingressi analogici disponibili.

Il convertitore stesso è solitamente un'architettura di Registro ad Approssimazione Successiva. Dopo aver selezionato l'ingresso e la tensione di riferimento, e aver atteso il tempo di acquisizione, il firmware può avviare la conversione (*Start of Conversion*, SoC) impostando un apposito bit all'interno del µC (il bit `GO/`$\overline{\texttt{DONE}}$). L'ADC SAR raggiunge i risultati di conversione in circa n+1 cicli di clock (dove n è il numero di bit dell'ADC). Si noti, tuttavia, che la frequenza di clock dell'ADC non è la frequenza di clock principale del µC, ma solo un sottomultiplo. Ciò è necessario per garantire il tempo di stabilizzazione appropriato dei circuiti analogici, per ridurre al minimo gli errori di conversione e possibili problemi.

Il modulo ADC ha tre registri. Il registro `ADCON0` controlla il funzionamento del modulo ADC. Il registro `ADCON1` configura le funzioni dei pin della porta, come ingressi analogici (RA3 può anche essere un riferimento di tensione V_{REF}) o come I/O digitali. Il registro `ADRES` contiene il risultato della conversione ADC.

Quando la conversione ADC è completa, il risultato viene caricato nel registro `ADRES`, il bit `GO/`$\overline{\texttt{DONE}}$ viene cancellato e il bit `ADIF` del flag di interrupt ADC viene impostato. Dopo che il modulo ADC è stato configurato come desiderato, il canale selezionato deve essere acquisito prima di avviare la conversione. I canali di ingresso analogico devono avere i bit `TRIS` corrispondenti selezionati come input.

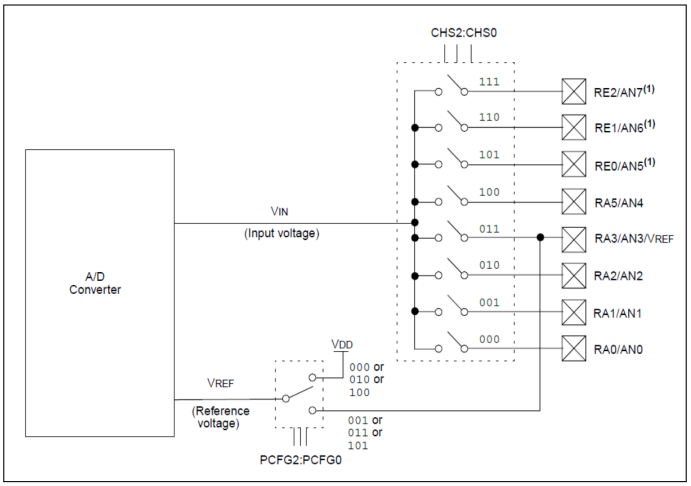


Figura 16: Diagramma a blocchi dell'ADC

2.6.1 Tempo di Acquisizione dell'ADC

Affinché il convertitore ADC soddisfi la precisione specificata, è necessario consentire al condensatore di mantenimento della carica (C_{HOLD}) di caricarsi completamente al livello di tensione del canale di ingresso. L'impedenza della sorgente (R_s) e l'impedenza dell'interruttore di campionamento interno (R_{ss}) influenzano direttamente il tempo necessario per caricare il condensatore C_{HOLD}. Notare che l'impedenza dell'interruttore di campionamento (R_{ss}) varia in base alla tensione del dispositivo (V_{DD}). Inoltre, l'impedenza della sorgente influenza la tensione di offset all'ingresso analogico, a causa della corrente di dispersione del pin $I_{leakage}$. Per calcolare il tempo minimo di acquisizione, può essere utilizzata la seguente equazione:

$$t_{acq} \geq \tau \cdot ln\left(\frac{V_{REF}}{1/2 \cdot LSB}\right) = (R_S + R_{IC} + R_{SS}) \cdot C_{HOLD} \cdot ln2^{n+1}$$

Deriva dalla tipica carica esponenziale di C_{HOLD} attraverso la resistenza in serie totale dalla sorgente di ingresso all'ingresso ADC, prendendo un errore massimo all'interno di LSB per soddisfare la precisione specificata dall'ADC.

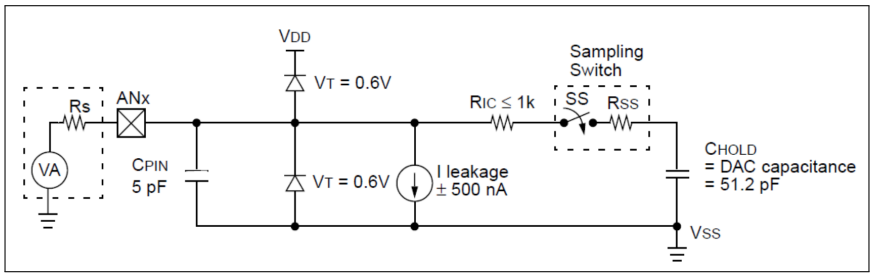


Figura 17: Modello dell'Analog Input dell'ADC

Tempo di Conversione ADC

Il tempo di conversione ADC per bit è definito come T_{AD}. La conversione dell'ADC richiede $9,5 \cdot T_{AD}$ per conversione a 8 bit. La sorgente del clock di conversione dell'ADC è selezionabile dal firmware tra quattro possibili opzioni: 2 T_{osc}; 8 T_{osc}; 32 T_{osc}; e l'oscillatore RC interno.

2.7 Watchdog o Comparatore Analogico

Alcuni µC hanno dei pin (uno o due) che possono essere utilizzati per eseguire un confronto tra una tensione analogica esterna e una soglia analogica interna. Questa soglia può essere impostata tramite firmware, accendendo e spegnendo una rete di resistori. Quando tale soglia viene superata, il comparatore attiva una richiesta di *interrupt* al nucleo del µC. Questo dispositivo è molto utile quando il µC deve monitorare un livello di tensione, da un sensore o più comunemente dalla batteria che alimenta il µC. In questi casi, non è necessario eseguire una conversione ADC accurata, ma semplicemente verificare se il livello di tensione in ingresso è sufficientemente alto. Risparmiare risorse e tempo può portare a un firmware e un sistema embedded molto più efficienti e intelligenti. A volte è previsto un doppio comparatore, con soglia sia inferiore che superiore, regolabile via firmware, per effettuare una selettività a "finestra" se l'ingresso analogico rientra in due valori definiti.

Un **Watchdog analogico** è simile al comparatore, ma l'unica differenza è che quando la tensione sul pin dedicato scende al di sotto di un valore di soglia, invece di un interrupt, viene generato un reset hardware del µC. Il Watchdog agisce infatti come un vero e proprio *"Cane da Guardia"* che forza un riavvio del µC ogni volta che qualcosa (in questo caso un livello di tensione) scende al di fuori del range corretto.

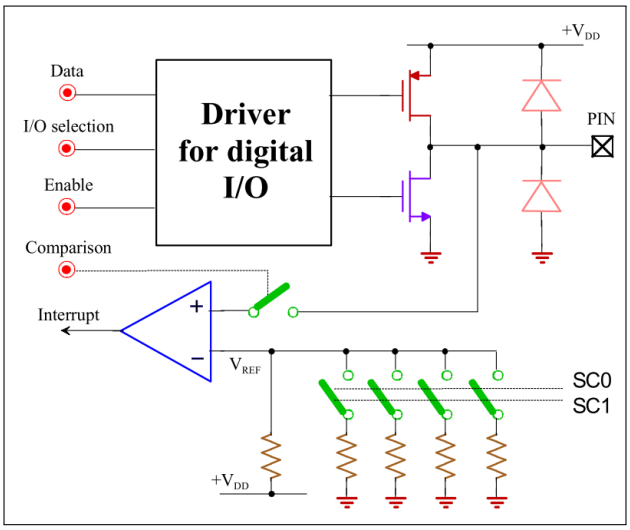


Figura 18: Comparatore Analogico periferico di un Microcontrollore

2.8 Timer e Contatore

Molti sistemi embedded devono essere in grado di contare gli eventi. Molte altre applicazioni richiedono, invece, di misurare il tempo assoluto da un evento di attivazione, o il tempo trascorso tra gli impulsi digitali (ad esempio uno Start e uno Stop), o semplicemente di attendere un tempo di ritardo ben definito (programmabile) prima di eseguire un task. Tutti i µC hanno una o più periferiche Timer/Contatore per svolgere tutte queste attività. Il cuore di queste periferiche è un contatore a n bit (solitamente 8 o 16 bit), il cui clock può essere pilotato da un pin digitale esterno dallo stesso oscillatore di clock principale del µC. Ogni periferica Timer/Contatore all'interno di un µC può generare il proprio *interrupt* in caso di overflow (cioè all'impulso di clock dopo il conteggio $\texttt{FF}_\texttt{H}$, che fa ripiegare il contatore in avanti a $\texttt{00}_\texttt{H}$) o di underflow (viceversa).

Si noti che non è necessario attendere che l'intervallo di fondo scala 2^n del contatore generi un interrupt. Ad esempio, per contare fino a 29 eventi prima di attivare un interrupt, è possibile precaricare il contatore con 28 e forzare il conteggio indietro. Se il contatore può solo contare in avanti, allora è possibile precaricarlo con 256-29 e attendere l'overflow al 29-esimo impulso. Lo stesso concetto si applica ai Timer.

2.8.1 TIMER0

I µC PIC16Cxx hanno tre moduli timer, ognuno dei quali può generare un interrupt per indicare che si è verificato un evento. Il modulo `TIMER0` ha le seguenti caratteristiche:

- timer/contatore a 8 bit; leggibile e scrivibile;
- prescaler programmabile software a 8 bit;
- selezione del clock interno o esterno;
- interrupt su overflow da $\texttt{FF}_\texttt{H}$ a $\texttt{00}_\texttt{H}$;
- edge select per clock esterno.

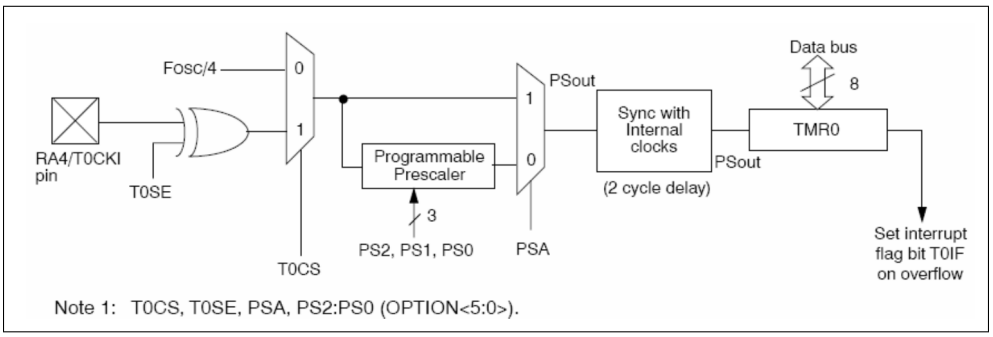


Figura 19: Schema a blocchi del TIMER0

La modalità timer si seleziona azzerando il bit T0CS: il contatore TMR0 incrementerà ogni ciclo di istruzione (senza prescaler). Se viene scritto il registro TMR0, l'incremento viene inibito

per i successivi due cicli di istruzione. Tuttavia, l'utente può aggirare questo problema scrivendo un valore regolato nel registro TMR0. Ciò significa che quando si avvia il timer, c'è sempre un ritardo prima di ottenere il valore corretto del conteggio.

2.8.2 TIMER1

La periferica TIMER1 è un timer/contatore a 16 bit composto da due registri a 8 bit (`TMR1H` e `TMR1L`) leggibili e scrivibili. La coppia di registri `TMR1` (`TMRIH:TMRIL`) aumenta da 0000_H a $FFFF_H$ e passa a 0000_H. L'interrupt TMR1, se abilitato, viene generato in caso di overflow che viene latchato nel bit del flag di interrupt `TMR1IF` (`PIR1`<0). Questo interrupt può essere abilitato/disabilitato impostando/cancellando l'abilitazione dell'interrupt TMR1 `TMR1IE`. La modalità operativa del TIMER1 è determinata dal bit di selezione del clock, `TMR1CS`: impostarlo a 1 per la modalità contatore e azzerarlo a 0 per la modalità timer.

In modalità timer, TIMER1 incrementa ad ogni ciclo di istruzione. In modalità contatore, aumenta ad ogni fronte di salita dell'ingresso del clock esterno.
TIMER1 può essere abilitato/disabilitato impostando l'azzeramento del bit di controllo `TMR1ON`.

La modalità timer viene selezionata cancellando il bit `TMR1CS`. In questa modalità, il clock in ingresso al timer è FOSC/4.

Sono necessarie precauzioni nel firmware per leggere/scrivere il timer perché, durante la lettura del timer a 16 bit in due valori a 8 bit, il timer potrebbe traboccare tra le letture. Per le scritture, si consiglia all'utente di arrestare semplicemente il timer e scrivere i valori desiderati, il motivo è che potrebbe verificarsi un conflitto di scrittura scrivendo nei registri del timer mentre il registro sta incrementando, il che potrebbe produrre un valore imprevedibile nel registro del timer.

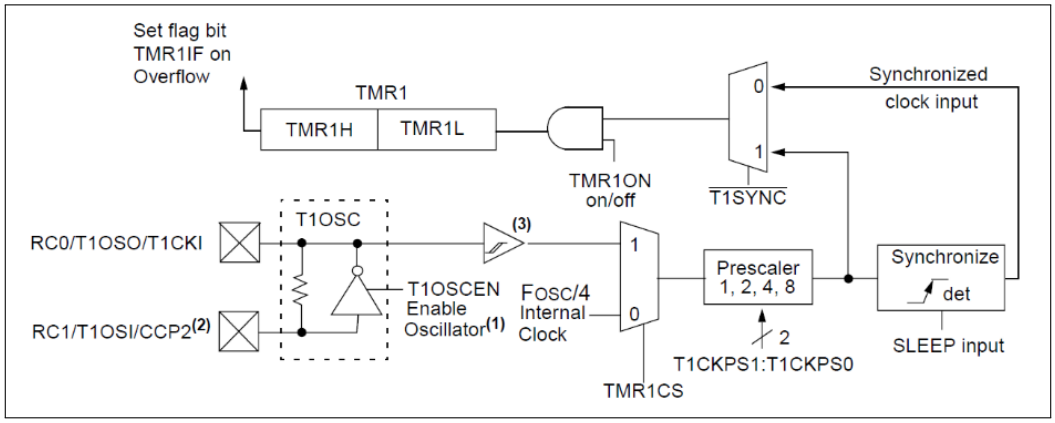


Figura 20: Schema a blocchi del TIMER1

2.8.3 TIMER2

TIMER2 è un timer a 8 bit con un prescaler e un postscaler. Può essere utilizzato come base dei tempi del PWM.
Il registro `TMR2` è leggibile e scrivibile e viene cancellato a qualsiasi ripristino del dispositivo. Il clock di ingresso (FOSC/4) ha un'opzione di prescaler di `1:1`, `1:4` o `1:16`, selezionata dai bit di controllo `T2CON<1:0>`.

La periferica TIMER2 dispone di un registro di periodo `PR2` a 8 bit: il timer incrementa da 00_H fino a coincidere con PR2 per poi tornare a 00_H al successivo ciclo di incremento. PR2 è un registro leggibile e scrivibile, inizializzato a FF_H al ripristino. TIMER2 può essere spento cancellando il bit di controllo `TMR2ON` per ridurre al minimo il consumo di energia.

L'output di corrispondenza di TMR2 passa attraverso un postscaler a 4 bit per generare un interrupt, latchato nel bit di flag `TMR2IF`. I contatori prescaler e postscaler vengono azzerati quando si verifica una delle seguenti condizioni:

- una scrittura nel registro `TMR2`;
- una scrittura nel registro `T2CON`;
- qualsiasi ripristino del dispositivo

Invece, TMR2 non viene cancellato quando viene scritto `T2CON`.

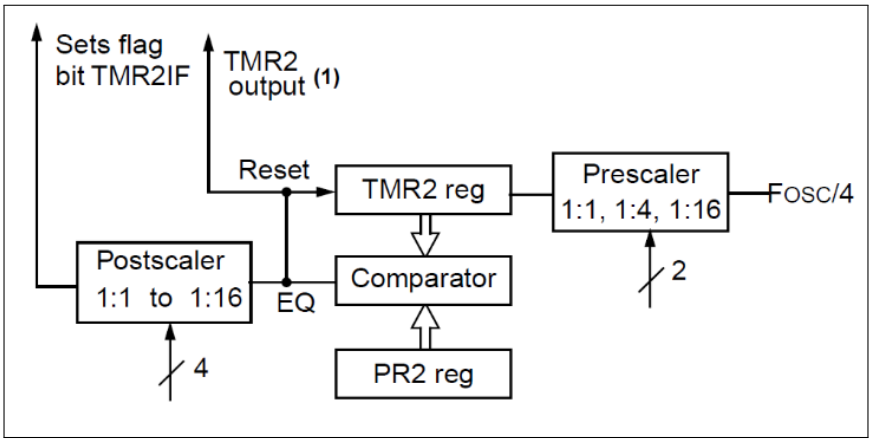


Figura 21: Schema a blocchi del TIMER2

2.9 Capture/Compare/PWM

Il modulo Capture/Compare/PWM (**CCP**) contiene un registro a 16 bit, che può funzionare come registro di *cattura* a 16 bit, come registro di *confronto* a 16 bit o come Registro *master/-slave PWM Duty-Cycle* (Pulse-Width Modulation). Ci sono due periferiche CCP nella famiglia PIC16Cxx: entrambe le periferiche CCP1 e CCP2 sono identiche nel funzionamento (con una piccola eccezione).

Per entrambi i moduli CCPx, il registro `CCPRx` è composto da due registri a 8 bit: `CCPRxL` (low byte) e `CCPRxH` (high byte). Il registro `CCPxCON` controlla il funzionamento di CCPx. Tutti sono leggibili e scrivibili. Affinché il modulo CCP funzioni, le risorse timer devono essere utilizzate insieme al modulo CCP. La modalità operativa CCP desiderata determina quali risorse del timer sono necessarie:

- la modalità di acquisizione e la modalità di confronto richiedono TIMER1;
- La modalità PWM richiede TIMER2.

Entrambe le modalità Capture e Compare richiedono che TIMER1 funzioni in modalità timer o in modalità contatore sincronizzato.

Modalità *Capture*

In modalità *Capture*, `CCPR1H:CCPR1L` cattura il valore a 16 bit del registro TMR1 quando si verifica un evento sul pin `RC2/CCP1`. Un evento è definito come:

- ogni fronte di discesa;
- ogni fronte di salita;
- ogni 4 fronte di salita;
- ogni 16 fronti di salita.

Un evento viene selezionato dai bit di controllo `CCP1CON<3:0>`. Quando viene effettuata una cattura, viene impostato il flag di richiesta di interrupt `CCP1IF` e quindi deve essere cancellato nel firmware. Se si verifica un'altra cattura prima che venga letto il valore nel registro CCPR1, il vecchio valore catturato andrà perso.

In modalità Capture, il pin RC2/CCP1 deve essere configurato come ingresso impostando il bit TRISC<2>. Se invece RC2/CCP1 è configurato come uscita, una scrittura sulla porta può causare una condizione di cattura.

Modalità *Compare*

In modalità di confronto, il valore del registro CCPR1 a 16 bit viene costantemente confrontato con il valore della coppia di registri TMR1: quando si verifica una corrispondenza, a seconda del valore dei bit di controllo `CCP1CON<3:0>`, il pin RC2/CCP1 è guidato alto, guidato basso, rimane invariato. Allo stesso tempo, viene impostato il bit di flag di interruzione CCP1IF. L'utente deve configurare il pin RC2/CCP1 come uscita azzerando il bit `TRISC<2>`.

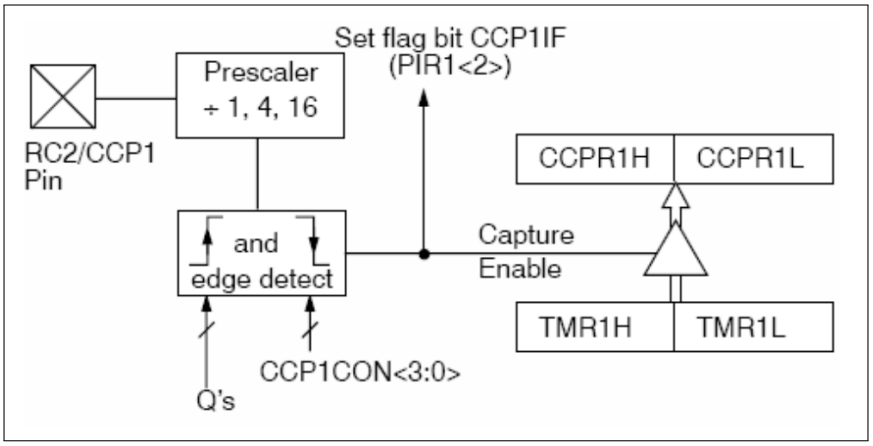


Figura 22: Diagramma a blocchi della modalità Capture

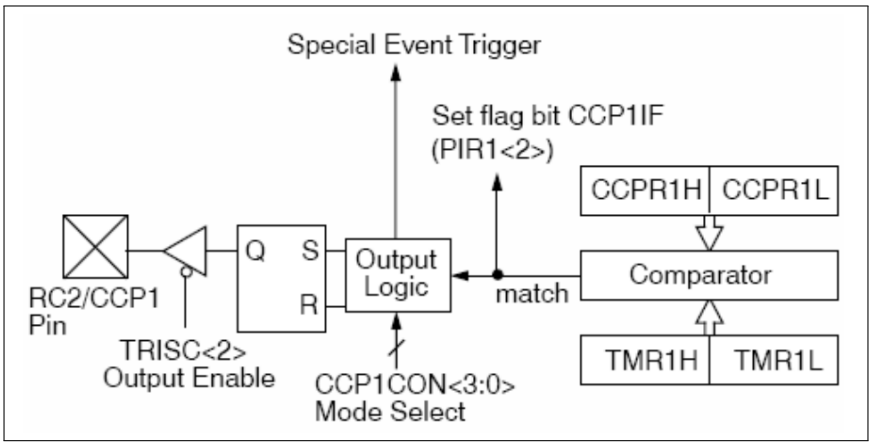


Figura 23: Diagramma a blocchi della modalità Compare

Modalità *PWM*

In modalità *Pulse-Width Modulation* (PWM), il pin CCPx produce un'uscita PWM con risoluzione fino a 10 bit. Una forma d'onda di uscita PWM è un segnale che ha una base temporale (periodo) e un tempo in cui l'uscita rimane alta (duty cycle). Il periodo è la durata dopo la quale il fronte di salita PWM si ripete. La risoluzione dell'uscita PWM è la granularità con cui è possibile variare il duty cycle. Ciascun modulo CCP può supportare un segnale di uscita PWM, con un sovraccarico del firmware minimo. Questo segnale PWM può raggiungere una risoluzione fino a 10 bit, dal modulo TIMER2 a 8 bit; questo fornisce 1024 passi di varianza da un contatore di overflow a 8 bit e una precisione massima pari a T_{osc}. Quando il TIMER2 va in overflow (ovvero il registro del periodo di temporizzazione), il valore nei registri del duty cycle (`CCPRxL:CCPRx-CON<5:4>`) si aggancia al latch dello slave a 10 bit. Un nuovo valore del duty cycle può essere caricato nei registri del duty cycle in qualsiasi momento, ma viene memorizzato nel latch dello slave solo quando TIMER2 = PR2. Il periodo di TIMER2 è determinato dalla frequenza del dispositivo, dal valore del prescaler TIMER2 (1, 4 o 16) e dal Registro di periodo TIMER2 (PR2).

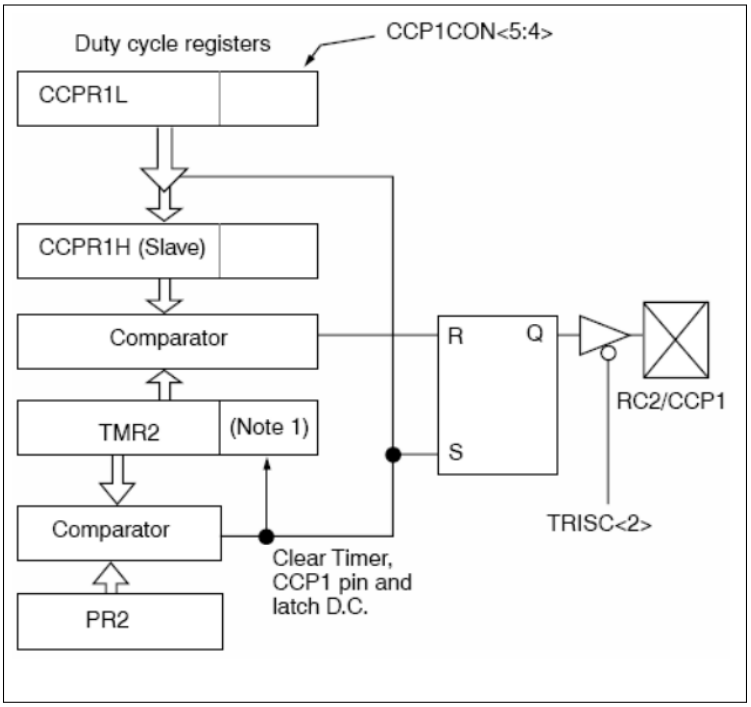


Figura 24: Diagramma a blocchi della modalità PWM

2.10 Watchdog

Analogamente a qualsiasi altra macchina a stati, anche i µC corrono il rischio di uscire dal corretto flusso di istruzioni e di andare in stallo o di iniziare a eseguire codice senza senso, a causa di errori del firmware, problemi hardware o disturbi transitori e picchi di tensione sull'alimentatore o qualsiasi altro pin di ingresso/uscita. Un modo molto semplice, ma efficace, per uscire da questo stato di *"impasse"* può essere realizzato da una periferica interna denominata Watchdog che, dopo comportamenti anomali del µC, lo ripristina tramite hardware, evitando così di continuare ad eseguire codice errato in modo incontrollato. Un watchdog è un timer digitale che non fa nulla per "capire" se tutto sta procedendo correttamente, ma semplicemente conta alla rovescia come una bomba. Durante il normale funzionamento, il codice C deve reimpostare regolarmente a $\texttt{FF}_\texttt{H}$ il timer del Watchdog per evitare che scada. Se, a causa di un guasto hardware o di un errore di programma, µC non riesce a ripristinare il watchdog, il timer scadrà e genererà un segnale di timeout. Il segnale di timeout viene utilizzato per avviare azioni correttive. Quello più semplice, sempre implementato nei dispositivi µC, consiste tipicamente in un reset hardware dell'intero µC. Naturalmente, una soluzione migliore sarebbe riportare il µC in uno stato sicuro e affidabile e non pericoloso, e quindi ripristinare il normale funzionamento del sistema. Tuttavia, quest'ultimo è molto più complesso da implementare, quindi il produttore di µC lascia semplicemente "esplodere la bomba" e inizia tutto (il codice) da zero!

I timer watchdog si trovano comunemente nei sistemi integrati e in altre apparecchiature controllate da computer in cui le persone non possono accedere facilmente all'apparecchiatura o

non sarebbero in grado di reagire tempestivamente ai guasti. Il watchdog può essere utilizzato anche per svegliare il µC da uno stato SLEEP, come sveglia pianificata in alternativa all'interrupt.

Nei µC di PIC16Cxx, il Watchdog Timer (WDT) è pilotato da un oscillatore RC on-chip a funzionamento libero che non non richiede alcun componente esterno. Questo oscillatore RC è separato dall'oscillatore RC del pin `OSC1/CLKIN`; ciò significa che il WDT verrà eseguito, anche se il clock sui pin `OSC1/CLKIN` e `OSC2/CLKOUT` del dispositivo è stato interrotto, ad esempio, mediante l'esecuzione di un'istruzione SLEEP.
Durante il normale funzionamento, un timeout WDT genera un RESET del dispositivo (*Watchdog Timer Reset*).
Se il dispositivo è in modalità SLEEP, un timeout WDT fa sì che il dispositivo si riattivi e continui con il normale funzionamento (*Watchdog Time Wake-up*).
Il WDT può essere disabilitato permanentemente cancellando il bit di configurazione WDIE durante la "masterizzazione" della memoria di programma del µC.

Il WDT ha un periodo di timeout nominale di **18 ms** senza prescaler. Se si desiderano periodi di timeout più lunghi, è possibile assegnare un prescaler con un rapporto di divisione fino a 1:128 al WDT sotto il controllo del firmware scrivendo nel registro `OPTION`. Pertanto, possono essere realizzati periodi di timeout fino a 2,3 secondi.
Nel caso più breve di soli 18 ms, il µC in esecuzione a un clock di 20 MHz può eseguire fino a 90.000 istruzioni prima di richiedere un reset del Watchdog per evitare l'*hard reset*.

Le istruzioni `CLRWDT` e `SLEEP` azzerano il WDT e il postscaler, se assegnati al WDT, e ne impediscono il timeout e la generazione di una condizione di RESET del dispositivo.

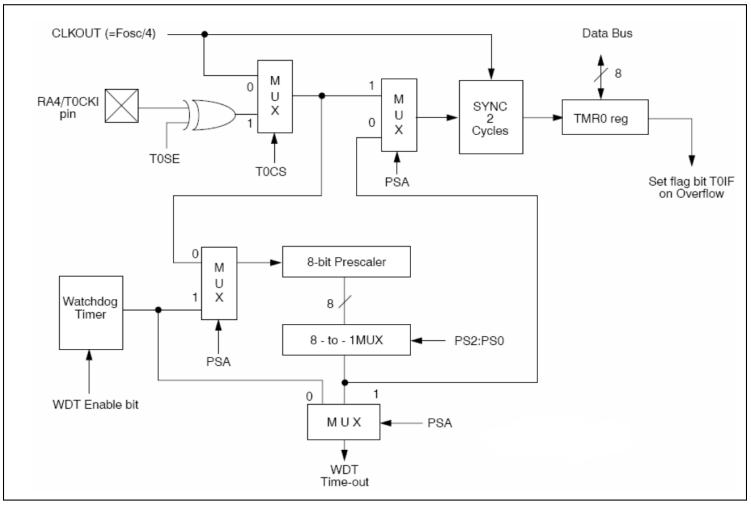


Figura 25: Diagramma a blocchi del Watchdog Timer **Interno**

La presenza di un watchdog è così importante in qualsiasi sistema embedded che, nel caso in cui il µC non ne abbia uno on-chip, è spesso una buona idea aggiungerne uno esterno. Un semplice watchdog esterno dovrebbe essere in grado di generare un reset senza dover testare tutti i bit della macchina digitale e senza bisogno di generatori di clock aggiuntivi. Il modo più semplice per capire se tutto è ok nel µC potrebbe essere quello di controllare uno dei suoi pin e vedere se c'è attività, come le transizioni del fronte di discesa e di salita entro un tempo massimo trascorso.

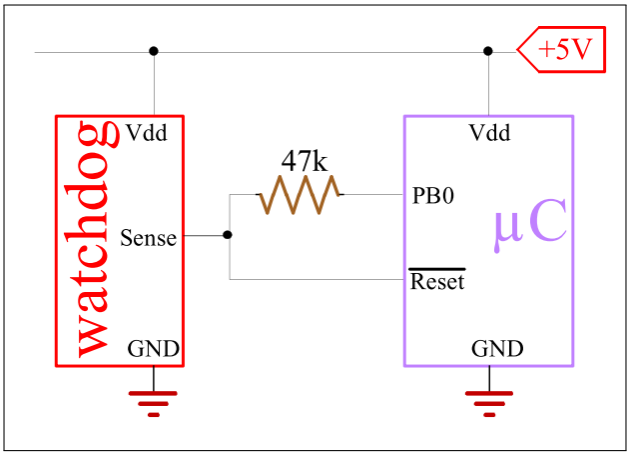


Figura 26: Diagramma a blocchi del Watchdog Timer **Esterno**

2.11 Dispositivi di Comunicazione Seriale

In informatica, una porta seriale è un'interfaccia di comunicazione seriale attraverso la quale le informazioni vengono trasferite in entrata o in uscita un bit alla volta (a differenza di una porta parallela). Per gran parte della storia dei personal computer, i dati sono stati trasferiti tramite porte seriali a dispositivi come modem, terminali e varie periferiche.

All'interno dei sistemi embedded, le porte seriali sono estremamente utili perché consentono al µC di dialogare e scambiare dati con altri dispositivi esterni, sia sulla stessa scheda (cioè all'interno del sistema embedded, come con EEPROM seriali, shift-register, display driver, ADC , ecc.) e su schede diverse (collegamenti ad altri sistemi embedded o ad un computer remoto o periferiche lontane). L'utilizzo di porte seriali anziché parallele, consente di risparmiare sia sul numero di pin utilizzati che sul numero di collegamenti, quindi sono molto utili sia per connessioni *intra-board* (chip-to-chip), sia per quelli *inter-board* (board-to-board). Esistono sostanzialmente tre diversi protocolli di base, diversi per il modo in cui trasferiscono i dati, sul numero di fili necessari e se consentono o meno di avere un solo master o forniscono accessi multipli a master, e se possono o meno gestire l'arbitraggio sul bus . I protocolli sono:

- Serial Peripheral Interface (**SPI**) di Motorola;
- Circuito Inter-Integrato (**I^2C**) di Philips;
- interfaccia di comunicazione seriale (**SCI**), meglio conosciuta come **USART**.

I primi due sono utilizzati principalmente per collegamenti intrascheda, mentre USART è più comune per collegamenti interscheda "lunghi". Microchip µC ha una periferica interna *Synchronous Serial Port* (SSP), che è un'interfaccia seriale utile per comunicare con altre periferiche o µC, che può funzionare in una delle due modalità: **SPI** o **I^2C**.

2.11.1 Protocollo SPI (*Serial Peripheral Interface*)

La *Serial Peripheral Interface* (SPI) è l'interfaccia seriale chip-to-chip più semplice. Ha il vantaggio di essere molto semplice, perché impiega solo tre fili per trasmettere un *Serial Data Output* (SDO) a 8 bit, per ricevere un *Serial Data Input* (SDI) a 8 bit e per alimentare un *Serial Clock* sincrono (SCK), rispettivamente. Ha lo svantaggio di consentire un solo master e ha una linea di clock unidirezionale (da master a slave). In caso di più slave, il master deve abilitarne solo uno alla volta. Il master è il µC che gestisce il clock e, in definitiva, il trasferimento dei dati; lo slave è il dispositivo esterno (o un altro µC, che rimarrà sempre uno slave).

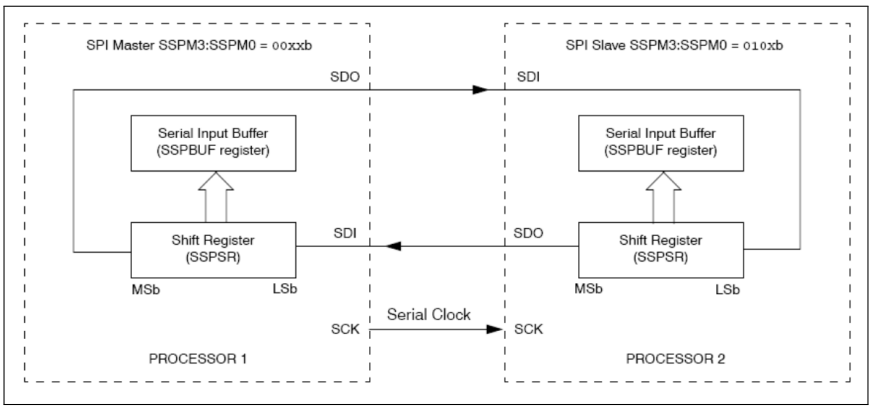


Figura 27: Connessioni SPI fra master(µC) e dispositivo slave

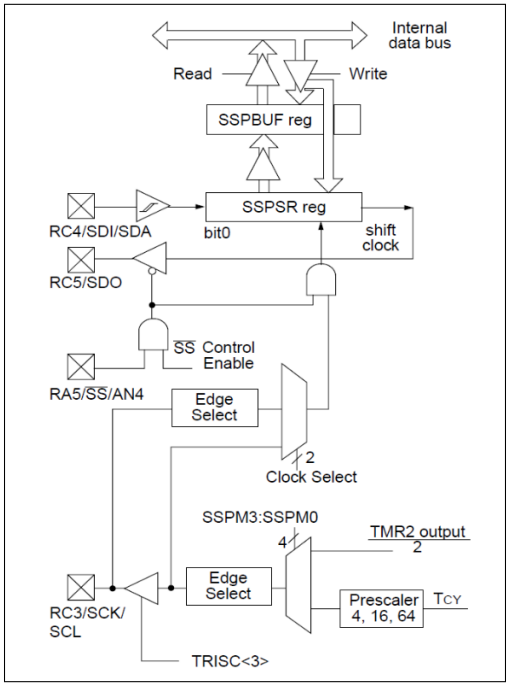


Figura 28: Diagramma a blocchi dell'SSP in modalità SPI

2.11.2 Protocollo I^{2}C (*Inter-Integrated Circuit*)

Il bus I^{2}C è un'interfaccia seriale a due fili sviluppata dalla Philips Corporation. La specifica originale, o modalità standard, prevedeva trasferimenti di dati fino a 100 kbps. Le specifiche avanzate (fast-mode) consentono di raggiungere i 400 kbps. Questo protocollo è più complesso dei precedenti, perché permette al µC di gestire l'arbitraggio e la sincronizzazione del bus seriale, utilizzando solo due fili: *Serial Clock* (SCL) e *Serial Data* (SDA). Quando non c'è trasferimento di dati, sia la linea di clock (SCL) che i dati (SDA) vengono portati a un livello alto dai resistori di pull-up esterni. Entrambe le linee sono bidirezionali perché ogni dispositivo può fungere da Master, occupando il bus e inviando l'indirizzo del dispositivo con cui vuole comunicare. A quel punto ogni altra richiesta di controllo del bus verrà respinta. Si tratta quindi di un sistema di comunicazione seriale adatto al collegamento di dispositivi "intelligenti", in grado di gestire il traffico sul bus e di riconoscere la chiamata al proprio indirizzo. Le linee chip-enable non sono più necessarie, poiché per il protocollo I^{2}C sono sufficienti due fili, qualunque sia il numero di dispositivi, contrariamente a quanto visto con l'interfaccia SPI.

Entrambe le linee devono essere open-collector con una resistenza di pull-up esterna, il cui dimensionamento viene effettuato in base al numero di dispositivi (sia master che slave) collegati. In questo modo ogni pin collegato alla linea (sia SCL che SDA) può essere sia un'uscita (open-drain) che un ingresso, a seconda di quale dispositivo ha il controllo del bus.

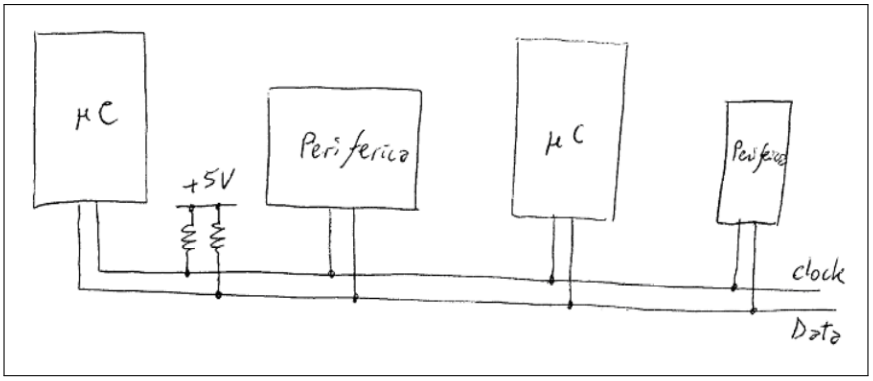


Figura 29: Comunicazione seriale I^{2}C

I problemi risiedono nella gestione dei protocolli, poiché ciò richiede un hardware complesso. Un µC di fascia bassa senza l'interfaccia per questo protocollo può comunicare solo tramite routine firmware creata ad hoc. Anche quelle periferiche che fungeranno sempre da slave devono essere intelligenti, per garantire la gestione dell'indirizzamento da parte del master e la corretta risposta al chiamante.

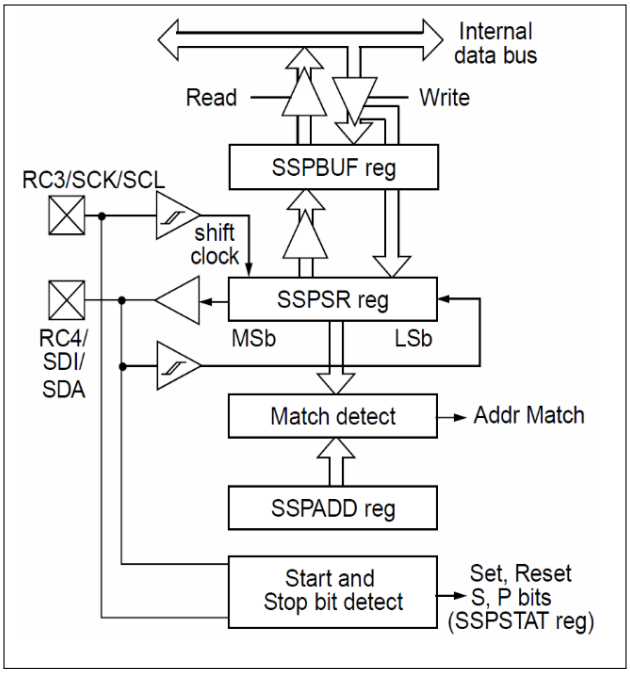


Figura 30: Diagramma a blocchi dell'SSP in modalità I^2C

2.12 Oscillatore

I circuiti necessari per realizzare il feedback positivo al fine di garantire l'oscillazione di un semplice circuito esterno sono solitamente già inclusi all'interno del µC. Tipicamente sono presenti due pin `OscIn` e `Oscout` dove collegare un cristallo di quarzo esterno in modo da ottenere la frequenza di oscillazione desiderata. Se si vuole garantire un'oscillazione stabile, si deve impiegare quarzo preciso e di alto valore (dell'ordine dei MHz), altrimenti per le basse frequenze è sufficiente utilizzare una semplice rete RC, perdendo ovviamente accuratezza e stabilità. Il vantaggio di una frequenza di clock inferiore è ovviamente una richiesta di potenza inferiore (che aumenta con l'aumentare della frequenza in ogni rete CMOS) e una minore generazione di disturbi di interferenza irradiati. Il PIC16CXX può essere utilizzato in quattro diverse modalità di oscillazione. L'utente può programmare due bit di configurazione (`FOSC1` e `FOSC0`) per selezionare una di queste quattro modalità:

1. Low Power Crystal **LP**;
2. Cristallo/Risonatore **XT**;
3. Cristallo/risonatore ad alta velocità **HS**;
4. Resistenza/condensatore **RC**.

Questi clock si estendono su tre bande: **LP** (applicazione a bassa potenza, fino a 200 kHz), **XT** (intermedia) e **HS** (alta velocità, fino a 20 MHz).

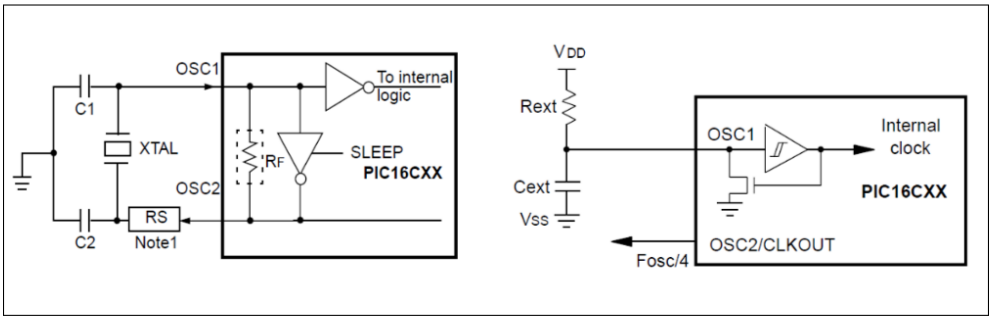


Figura 31: Oscillatore interno con cristallo esterno(SX) e con rete RC(DX)

Il clock può essere fornito anche esternamente, perché già disponibile sulla scheda o per garantire la sincronizzazione con altri circuiti. In questo caso utilizzeremo solo il pin `OSC1`, mentre la frequenza dell'oscillatore divisa per 4 è disponibile sul pin `OSC2/CLKOUT`. Diversamente, l'oscillatore esterno può essere nuovamente realizzato con un quarzo in risonanza sia in parallelo che in serie. Per applicazioni insensibili alla temporizzazione, l'opzione del dispositivo "RC" offre un ulteriore risparmio sui costi.

2.13 Reset

Il reset è necessario per una corretta inizializzazione della macchina sequenziale rappresentata dal core del µC. All'inizio, quando l'µC viene alimentato, deve partire dalla prima riga di codice e poi eseguire tutte le procedure di inizializzazione. Ci sono più sorgenti di reset, perché la richiesta può essere fornita dall'esterno o essere generata internamente nel µC stesso. Il PIC16CXX distingue vari tipi di reset:

- Master Clear (MCLR)
  Questo reset può verificarsi durante il normale funzionamento del firmware o durante lo SLEEP. Il progettista può collegare una semplice rete RC al pin dedicato a questo scopo per garantire il ripristino automatico all'accensione. Quando viene applicato VDD, il condensatore mantiene l'ingresso di ripristino MCLR attivamente basso per un tempo sufficiente. Quindi, durante la carica, la tensione ai capi del condensatore supera il livello di soglia digitale del pin e viene rilasciato il ripristino. Il diodo esterno viene utilizzato per aiutare il diodo di protezione all'interno del pin $\overline{MCLR}$: quando l'alimentazione viene tolta e VDD scende a 0V, il condensatore C carico è a una tensione superiore a VDD e anche rispetto all'alimentazione interna del µC, questo può eventualmente danneggiare il pin o la circuiteria interna. Invece, grazie al diodo D, il condensatore viene rapidamente cortocircuitato a VDD, che sta svanendo a massa. Infine, il resistore R1 mira a limitare la corrente nel pin del µC.

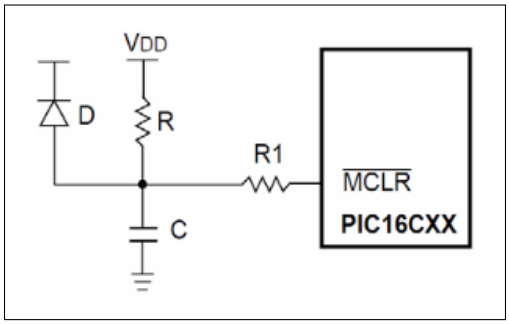


Figura 32: Utilizzo del Master Clear per resettare l'µC all'accensione

- Power-On Reset (POR)
  Oltre alla rete esterna (Punto precedente), i µC hanno normalmente una circuiteria interna per generare un reset ogni volta che viene applicata l'alimentazione, cioè all'accensione. Un impulso di Power-On Reset viene generato sul chip quando viene rilevato un aumento VDD (nell'intervallo 1,5V - 2,1V). Per sfruttare il POR, basta collegare il pin $\overline{MCLR}$ direttamente (o tramite un resistore) a VDD. Ciò eliminerà i componenti RC esterni solitamente necessari per creare un ripristino all'accensione. Anche se è stato imposto un corretto POR, quando il µC inizia il normale funzionamento, i parametri operativi del dispositivo (tensione, frequenza, temperatura, ecc.) devono essere soddisfatti per garantire il funzionamento. Se queste condizioni non sono soddisfatte, il dispositivo deve essere tenuto in reset fino al raggiungimento delle condizioni di funzionamento.

- Power-Up Timer (PWRT)
  Il timer di *Power-Up* fornisce un timeout nominale fisso di *72 ms* solo all'accensione, dal *Power-On Reset*. Il timer di Power-Up funziona su un oscillatore RC interno. Il chip viene mantenuto in reset finché il PWRT è attivo. Il ritardo del PWRT consente al VDD di raggiungere un livello accettabile. Viene fornito un bit di configurazione per abilitare/disabilitare il PWRT.

  NB: Il ritardo di accensione varia da chip a chip a causa di VDD, temperatura e variazione di processo.

- Start-Up Timer dell'oscillatore (OST)
  Il timer di avviamento dell'oscillatore fornisce un ritardo di 1024 cicli dell'oscillatore (dall'ingresso `OSC1`) al termine del ritardo del PWRT. Ciò garantisce che l'oscillatore al cristallo o il risonatore si sia avviato e stabilizzato. Il timeout OST viene invocato solo per le modalità XT, LP e HS e solo al Power-on Reset o al risveglio da SLEEP.

- Brown-Out Reset
  Con il termine *"brown-out"* si intende la momentanea fluttuazione di potenza che può essere seguita dal vero e proprio "black-out". Se l'alimentazione dovesse subire notevoli fluttuazioni (circa 1V), il power-on reset non interverrebbe, nonostante gli errori che il µC potrebbe aver subito nel firmware (es. a causa della mancata lettura o scrittura di alcuni bit), hardware (es. lettura da dispositivi esterni), o software (alcuni bit di registri o memoria di programma possono essere commutati). È quindi necessario disporre di un

circuito di rilevamento del brown-out che funzioni quando la tensione di alimentazione fluttua di un certo valore del valore nominale.

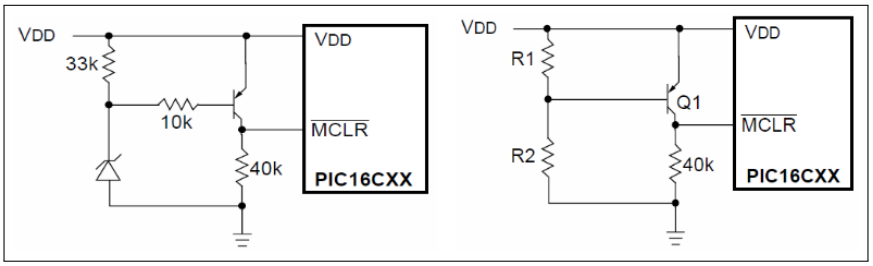


Figura 33: Circuito di protezione di Brown-Out Esterno