Stefano Busnelli (IlGuru)

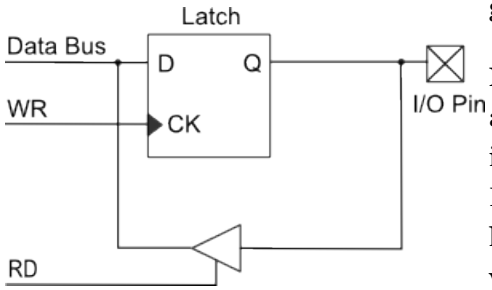
I PIC E IL PROBLEMA DEL READ MODIFY WRITE

14 September 2015

Abstract

Led che erano accesi che si spengono inspiegabilmente, comandi inviati ad un dispositivo che non fanno quello che devono, livelli di segnali che cambiano senza che il codice abbia detto di farlo ecc... Il PIC è posseduto? Serve un esorcista? Forse no, forse si tratta di un problema dovuto alla struttura circuitale di alcune famiglie di PIC che implementano la logica RMW, quella che ci fa esclamere WTF!

Struttura di un PIN generico

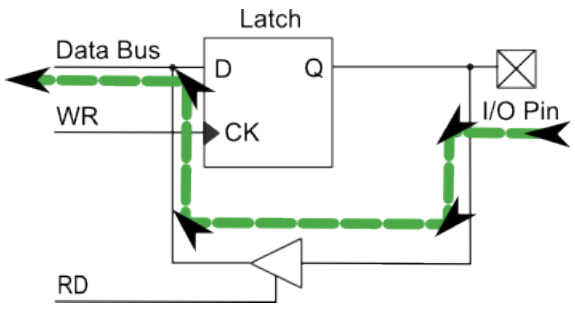


L'immagine mostra la versione semplificata di un generico pin digitale di I/O di un PIC ad 8 bit.

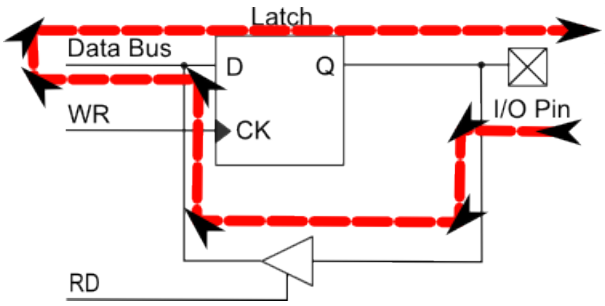
Nell' immagine non compaiono i vari blocchi relativi ai registri TRISx di direzione dei dati, il MOS che implementa il pull-up programmabile ecc. Il registro PORTB (o PORTA) è rappresentato dal latch, il valore logico che il programma legge o scrive su Data Bus, di volta in volta viene memorizzato nel latch e portato sul pin corrispondente.

Il Read Modify Write

A proposito di PORTB (ma il discorso è identico per PORTA), il datasheet del PIC16F628A recita: *Reading PORTB register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch*. Il datasheet non parla di singoli pin, ma di pins, ovvero dell' intera porta.



Quando si tratta della lettura di un pin, viene letto ciascun pin della porta ed il valore copiato in un registro interno del PIC.

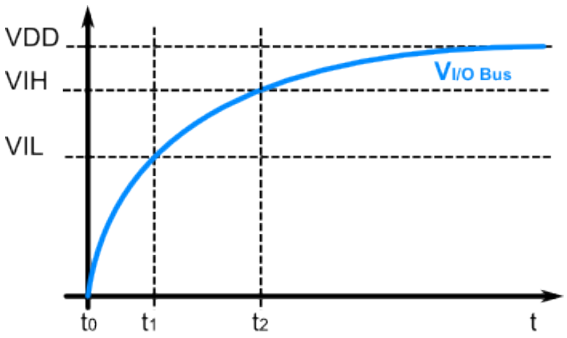


Ma quando si tratta di modificare lo stato logico di un pin ad esempio RB0, in certe circostanze circuitali si cela un'insidia.

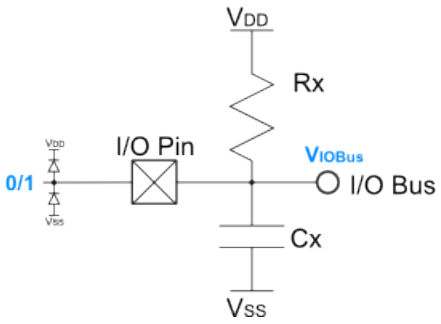
Infatti il PIC prima legge lo stato logico nel pin, ne modifica il contenuto e poi memorizza il valore nel registro PORTB.

I guai sono sempre nei dettagli..

Nei PIC con core base-line o mid-range, la lettura di un registro PORTx e quindi anche la lettura che viene fatta per la modifica, significa sempre leggere il livello logico dei pin.



Se il pin è collegato ad un componente passivo non ci sono problemi, ma se il pin è collegato ad un bus (es. I2C) o ad un altro dispositivo (es. il gate di un FET) che presentando delle capacità parassite rappresentano un carico di tipo capacitivo, la sistuazione si complica.

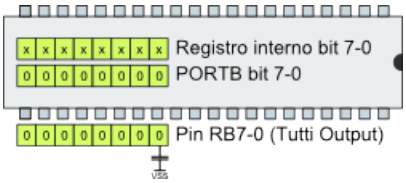


Se nel'istante t_0 abbiamo cambiato lo stato da 0 ad 1, dovremmo attendere fino a t_2 prima che il valore logico letto sul bus corrisponda allo stato 1, ma se avviene un RMW prima di t_1 ecco che il valore che verrà letto sarà uno 0, che verrà riscritto nel latch della porta. Questo problema si presenta tanto più in questi casi:

- quando la costante τ di tempo di carica e scarica del condensatore parassita è grande.

- quando lavoriamo con frequenze di clock elevate
- quando la frequenza con cui cambia lo stato di un pin è elevata, come ad esempio nelle trasmissioni di segnali.

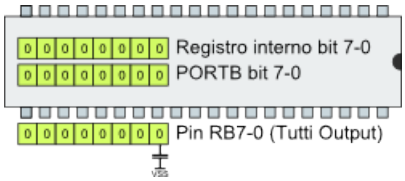
Vediamo un esempio del dettaglio:



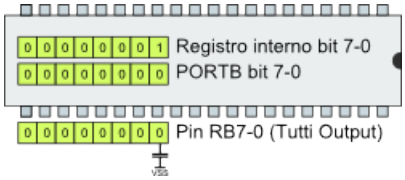
Partiamo dalla condizione in cui i pin di PORTB sono tutti configurati come Output, sono tutti stabili ed a livello logico basso.

In questa condizione portiamo il livello logico di RB0 ad 1:

```
PORTBbits.RB0 = 1;
```

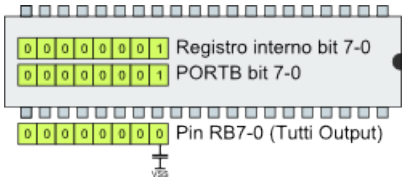


Fase READ del Read-Modify-Write, il valore dei pin RB7-RB0 viene letto e memorizzato in un registro interno del pic.

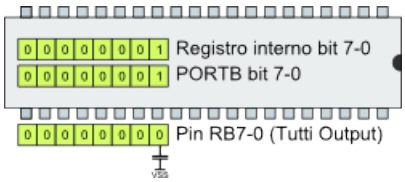


Fase MODIFY del Read-Modify-Write, il bit 0 del registro interno viene modificato.

Fase WRITE del Read-Modify-Write, il registro interno viene copiato sul registro PORTB.



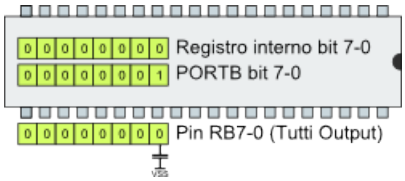
A questo punto RB0 in un caso normale, andrebbe immediatamente ad 1 (ignorando i ritardo di propagazione del segnale interno al circuito del PIC che comunque è ordini di volte inferiore al periodo del clock). In questo caso però il pin che ha una certa resistenza di uscita non riesce a caricare istantaneamente la capacità parassita del bus, la quale inizia a caricarsi con una certa costante di tempo τ.



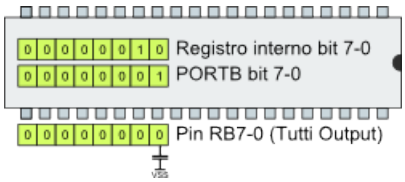
Subito dopo questa istruzione, quindi in tempi dell'ordine del µs modifichiamo un altro pin:

```
PORTBbits.RB1 = 1;
```

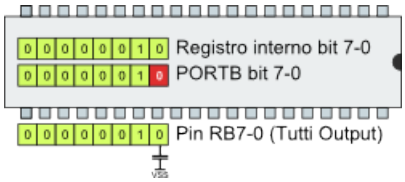
La nostra capacità parassita è ancora li che si sta caricando...



Fase READ del Read-Modify-Write, il valore dei pin RB7-RB0 viene letto e memorizzato in un registro interno del pic. Quello che viene letto però, non è quello che c'è nel registro PORTB perchè la tensione ai capi della capacità parassita è ancora sotto il livello di soglia VIH, ed al posto del pin RB0=1 viene letto RB0=0.



Fase MODIFY del Read-Modify-Write, il bit 1 del registro interno viene modificato. Il bit 0 rimane quello che è stato letto dal pin, quindi 0.

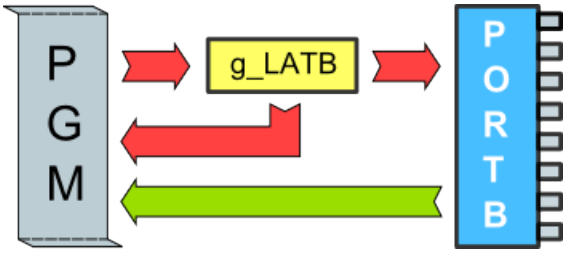


Fase WRITE del Read-Modify-Write, il registro interno viene copiato sul registro PORTB. Il pin RB1 va correttamente ad 1, ma l valore di RB0 che il programma aveva impostato nell' istruzione precedente viene sovrascritto. A questo punto il pin RB0 rimane a 0 e potremmo esserci persi un impulso di SCL su un bus I2C.

Saremmo li a cercare di capire perchè la memoria, il display o il DAC non vuole saperne di comunicare, debuggheremmo il codice eseguendolo passo-passo senza trovare nessun bug e saremmo bloccati. Il problema non salta fuori dal codice, perchè non è un bug del software.

Soluzione

Nei PIC a 16 bit, in quelli di fasce superiori come nella famiglia PIC18F questo problema non si pone, perchè sono dotati di un ulteriore registro LATCH interno. In questi dispositivi, per scrivere su una porta si può scriver sul registro LATx invece che su PORTx. La scrittura su LATx è equivalente alla scrittura su PORTx, ma la lettura da LATx ritorna il valore realmente memorizzato nel latch e non quello "falso" che potrebbe restituire la lettura del pin. Se volessimo proprio sapere lo stato del pin, si può sempre fare leggendo PORTx, operazione che però non influenza il contenuto del latch.



Possiamo adottare questo metodo anche con i PIC di fascia bassa, solo che questo ci costa la dichiarazione di un registro e qualche operazione in più mentre nei PIC "fighi" di tutte queste cose se ne occupa l'hardware.

Infatti dobbiamo riservare un indirizzo per memorizzare il dato che vogliamo venga inviato sulla porta d'uscita, chiamandolo ad esempio g_LATx. Ogni volta che vorremo leggere lo stato dei pin leggeremo normalmente il registro PORTx, le operazioni bitwise per impostare o invertire lo stato dei pin le effettueremo su g_LATx e quando vorremo modificare effettivamente lo stato dei pin, copieremo il contenuto di g_LATx nel registro PORTx.

Il problema del RMW deriva dall'illusione di operare su un singolo pin quando invece il dispositivo legge e scrive interamente la porta. Effettuare le operazioni sui bit di PORTx impica la riscrittura del registro con gli stati attualmente esistenti, per questo interporre un registro in cui memorizzare gli stati che vogliamo in uscita, ci permette di ripristinare la condizione che vorremmo ogni volta che copieremo il nostro "finto latch" sulla porta.

Esempio

Possiamo lavorare utlizzando dei registri Shadow, dichiarando delle variabili da utilizzare a questo scopo:

```
// A livello globale, ad esempio nel main.c dichiariamo una variabile che fa da latch:
unsigned char g_LATB    = 0x00;
// Dichiariamo una macro (o in alternativa una funzione) che copia questa variabile su
#define f_Write_LATB_PORTB()   PORTB = g_LATB
```

Poi alla bisogna:

```
unsigned char c_var    = 0x00;
// Se dobbiamo leggere un pin facciamo come al solito:
c_var = PORTB;
// Ed eventualmente, solo se è il caso ci memorizzimo il valore del bit di PORTB
//    che ci interessa nel latch shadow:
g_LATB |= ( c_var & 0b00000001 );                          // Copiamo solo il valore del
g_LATB |= ( PORTB & 0b00000001 );                          // Copiamo solo il valore del
// Se invece dobbiamo scrivere su port B, passiamo attraverso il registro shadow:
g_LATB = ((g_LATB & 0b11111110) | (c_var & 0b00000001));  // Memorizziamo il primo bit
// oppure
g_LATB &= 0b11111110;                                      // Impostiamo il primo bit a
```

```
// oppure
g_LATB |= 0b00000001;                                    // Impostiamo il primo bit a
// O varie ed eventuali altre operazioni bitwise...
// Poi richiamiamo la funzione di copia:
f_Write_LATB_PORTB();
```

In questo modo qualunque sia il contenuto di PORTB, viene sovrascritto da quello che abbiamo calcolato nel programma. Invece di lavorare come al solito così:

```
PORTBbits.RB0=1;
PORTBbits.RB1=1;
```

Scriveremo:

```
g_LATB |= 0b00000001;
f_Write_LATB_PORTB();
g_LATB |= 0b00000010;
f_Write_LATB_PORTB();
```

Se invece vogliamo settare i due bit contemporaneamente:

```
g_LATB |= 0b00000011;
f_Write_LATB_PORTB();
```

Poi si può scrivere una libreria un po' più sofisticata, ma il succo è questo.

Note

Versione del documento: 1.1

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Ilguru:i-pic-e-il-problema-del-read-modify-write"