Steve Blackbird (TardoFreak)

LE INTERRUPT. TEORIA "SOFT" PER I NEOFITI

22 June 2013

Introduzione

Mi capita abbastanza di sovente vedere che molti neofiti aspiranti microcontrollisti hanno qualche difficoltà a comprendere, o non conoscono, le interrupt. In questo breve articolo tenterò timidamente di spiegare cosa sono, come e perché si usano e quando è bene utilizzarle e quando no. E' una trattazione molto leggera e all' acqua di rose, niente di ultra tecnico, vuole solo essere un primo aiuto per la comprensione della cosa.

Cosa sono

Per capire bene cos'è una interrupt partiamo dal suo significato letterale: **interrompere**. Per capire ancora meglio immaginiamo di essere dei microcontrollori umani che, durante la giornata, svolgono il loro compito: vivere. Quindi mi sveglio, mi lavo, mi vesto, mi reco al lavoro, mangio pranzo e tutto il resto.

Ma la vita non è solo l' esecuzione di una sequenza di operazioni perché, di tanto in tanto, qualche evento interrompe il mio pacato fare: lo squillo del telefono ad esempio. Ecco, quella è una **interrupt**! E' quindi un **evento** che interrompe (magari "rompe" anche) quello che in quel momento sto facendo. A questo punto potrei fare due cose: o fregarmene del telefonino che squilla oppure smettere momentaneamente di fare quello che sto facendo e rispondere al telefono. Se rispondo al telefono sto, in pratica, eseguendo la **funzione/routine di servizio** dell' interrupt. Alla fine della telefonata faccio mente locale per ricordarmi cosa stavo facendo e riprendo a fare il lavoro di prima.

E' facile immaginare che lo svolgimento della nostra vita è piena di interrupt: una telefonata, il corriere che suona al citofono, la pipì quando scappa, il capo o il collega che ti chiede qualcosa, la moglie/fidanzata che si mette a farti domande proprio mentre stai cercando di guardare un programma in TV. Sono tutti eventi che generano interrupt ed ognuno di loro viene gestito in modo differente. Se un pirla mi taglia la strada (evento automobilista_pirla_taglia_strada) la ruotine di servizio sarà ovviamente diversa da quella che userei quando arriva l' interrupt della pipì che scappa (evento me_la_sto_facendo_addosso). Il primo caso attiverà una funzione che mi fa frenare e mandare a quel paese l' automobilista, nel secondo caso attiverà una funzione che mi farà alzare e correre in bagno.

Quindi ogni tipo di evento in grado di generare un' interruzione ha la sua funzione di servizio.
Come nei microcontrollori ho anche la possibilità di ignorare un' interruzione, cioè la posso "mascherare" oppure non abilitarla proprio. Per esempio posso decidere di non rispondere al cellulare ed ho due modi per farlo: fregarmene di lui e lasciarlo squillare (o metterlo in silenzioso) oppure spegnere il telefono. Nel primo caso la maschero (magari momentaneamente) nel secondo caso la disabilito completamente.
Le nostre interrupt umane hanno anche una priorità che dipende dalla loro importanza, per esempio se devo decidere se correre al bagno o rispondere al telefono di casa (che non mi posso portare al bagno) ovviamente correre al bagno ha priorità più elevata.
Bene, nei microcontrollori succede esattamente la stessa cosa, ma prima di parlare dei nostri amici di silicio analizziamo come funziona un' interruzione.

Come funzionano

Continuiamo con l' esempio umano perché è perfetto ed analizziamo passo per passo quando scatta l' evento telefonata. Supponiamo che di essere dove siamo adesso e di fare quello che stiamo facendo: leggere un articolo su EY.

- Suona il telefono (**evento telefonata**)
- **Interrompo** la lettura dell' articolo.
- Mentalmente prendo nota di cosa stavo leggendo, l' argomento, la riga. In linguaggio tecnico si dice **salvo il contesto**. Dove lo salva il micro lo vedremo più avanti, io me lo salvo da qualche parte nella mente.
- Afferro il telefono e rispondo, cioè **eseguo la funzione di servizio** dell' interrupt generata dall' evento telefonata.
- La telefonata termina (fine della funzione di servizio)
- Rovisto nella mia mente per ricordarmi cosa stavo facendo prima dell' interruzione. Tecnicamente **recupero il contesto**.
- **Riprendo** a leggere l' articolo su EY.

Un' ultima nota: se mentre sto parlando al telefono mi scappa la pipì (evento me_la_sto_facendo_addosso) che è un evento di priorità più elevata della telefonata, interrompo la telefonata, salvo il contesto della telefonata ecc. ecc. Quindi al mio ritorno dovrò recuperare il contesto precedente (quello della telefonata) e, finita la telefonata recuperare il contesto ancora precedente (leggere l' articolo su EY).

A cosa servono

A questo punto pare evidente che **le interrupt servono per rispondere a dei determinati eventi**. Non servono per fare un qualcosa di specifico, per fare un lavoro o per fermarne un' altro. No, sono azioni più o meno complesse che fanno da

contorno a quello che è lo svolgimento del compito primario.
Il paragone umano, a mio avviso, è quanto di più rispondente a quello che succede con i microcontrollori. Ricordarsi di cosa sono le interrupt umane significa avere sempre chiaro cosa sono e a cosa servono.

Come funzionano nei micro

Bene, ora dovrebbe essere chiaro cosa sono le interrupt. Passiamo quindi a vedere cosa c' entrano con i micro e come questi rispondono alle interruzioni.

Chi genera gli eventi

Gli eventi che attivano le interurpt possono essere i più disparati, ne elenco alcuni:

- Un pin del micro passa da stato logico 0 a stato logico 1
- Un timer arriva al termine del conteggio
- La EUSART ha ricevuto un carattere dalla seriale
- La SPI ha finito di inviare un byte

Chiaramente le interrupt possono essere abilitate o meno. I vari dispositivi generano comunque gli eventi, se l' interrupt per un tale evento è abilitata allora verrà generata un' interruzione.
Ci sono alcune periferiche interne dei micro che possono generare più eventi di interruzione. Ad esempio la EUSART può generare un' interruzione quando riceve un carattere e/o quando ha appena finito di mandarne uno. Nel primo caso la funzione di servizio andrà a leggere questo carattere per farci qualcosa, nel secondo caso invierà il carattere successivo che è eventualmente in attesa di essere trasmesso. Sono comunque funzioni diverse, ognuna per servire una specifica interruzione.
Il timer può benissimo averne una sola: il raggiungimento dell' overflow (o del conteggio impostato). Ovviamente la funzione che servirà questa interruzione servirà solo questa.

Cosa fa il micro quando viene interrotto

Ed eccoci al punto cruciale di tutto l' ambaradan! Supponiamo che il micro stia felicemente ed allegramente eseguendo il suo programma principale, ha un' interrupt abilitata (supponiamo quella del timer) e ... zacchete, questa arriva. Bene lui:

- finisce di eseguire l' istruzione che sta eseguendo
- **Salva il contesto**. Il contesto non è altro che lo stato dei registri della MPU (ce ne possono essere diversi) e del program counter. In poche parole fa una specie di fotografia istantanea dello stato in cui è e la mette nello stack.
- Salta all' indirizzo di partenza della **funzione di servizio**.
- Esegue la funzione

- **Recupera il contesto** dallo stack
- **Riprende l' esecuzione** del programma principale esattamente dal punto in cui è stato interrotto.

in pratica fa la stessa cosa che facciamo noi esseri umani ma lui è una macchina e tutto il processo è ben determinato e regolato.

Come scrivere le funzioni di servizio

Non esiste un metodo generale perché dipende dal micro che si sta utilizzando vediamone alcuni:

- **PIC16**. Esiste un' unica funzione/routine per il servizio dell' interrupt, una sola per tutte. Per capire quale dispositivo ha generato l' interrupt bisogna testare manualmente i bit di richiesta di interrupt di tutti i dispositivi. Ovviamente si vanno a guardare solo quelli che interessano, cioè quelli per cui si è deciso di far generare un' interruzione. Sarebbe inutile andare a vedere se, chessò, la seriale ha generato un' interruzione se questa non interessa e non è abilitata. Inpratica si risolve con qualche bit-test (in assembly) o con qualche if (in C).
- **PIC18** Stessa cosa del PIC16 ma di funzioni di servizio ce ne sono due: una per le interrupt a priorità alta ed una per quelle a priorità bassa.
- **ATmega** Ogni evento ha il suo vettore di interrupt e quindi ogni evento ha la sua funzione di servizio. La priorità delle interruzioni è fissa, decisa dalla Atmel.
- **ARM, PIC32** etc. Non solo ogni evento ha la sua funzione specifica ma per ogni evento è possibile impostare la priorità e la sub-priorità. Non solo, ma alcuni eventi interni (come un errore sul bus o una divisione per zero e molto altro) generano sempre e comunque delle interrupt che devono per forza essere gestite e servono per aumentare l' affidabilità del sistema. Questi oggetti sono molto flessibili ma anche molto complicati.

Quando utilizzarle e quando no

Anche qui non esiste una regola generale che non sia quella del buon senso. C'è da dire che le interrupt sono state implementate per **rispondere più velocemente possibile** a gli eventi. Si usano quindi quando questa velocità di risposta è richiesta. Prendiamo ad esempio la nostra buca delle lettere. Non avrebbe senso metterci un sensore per sapere esattamente quando il postino ci infila una lettera o una cartolina perché nessuno si sognerebbe di interrompere quello che sta facendo (magari sta lavorando o pranzando) per correre a gambe levate per prendere la posta. per carità, si potrebbe anche fare, ma sarebbe uno spreco di energie. Così come non ha senso collegare un pulsante azionato a mano ad un pin del micro che genera una interrupt

in quanto i tempi umani sono così lunghi che basta andare a testare il pin di tanto in tanto.
Questa tecnica è chiamata **polling** e, in effetti la parola "poll" significa sondaggio. Si tratta infatti di andare a sondare lo stato di un qualcosa di tanto in tanto, ad esempio un pulsante o comunque un qualcosa che non necessita di reazione rapida.
Ha invece senso utilizzare una interrupt per la linea seriale per correre a leggere il carattere ricevuto, metterlo da qualche parte in memoria (in un buffer per esempio) *per non correre il rischio di ignorare un carattere ricevuto*. oppure quando un timer va in overflow *per ricaricarlo nel tempo più breve possibile* e fare quello che si deve fare quando va in overflow. Si usano, ad esempio, per misurare la lunghezza di un impulso se il micro non è provvisto di una periferica per la cattura degli impulsi. Insomma le interrupt non servono per tutto, non sono la soluzione di tutti i problemi, **non servono per fare una qualche elaborazione ma solo per rispondere a gli eventi**. E non bisogna abusarne per non caricare il micro di lavoro inutile, magari a scapito della velocità di risposta che alcuni eventi meriterebbero di avere.

Conclusioni

Bene, spero di aver dato un piccolo contributo alla comprensione delle interrupt. Chiaramente l' argomento è vasto e non era nelle mie intenzioni fare un trattato completo ma solo quello di dare una prima infarinatura ai neofiti in modo da facilitarne la comprensione.
Tempo permettendo (per questo non ho scritto "Parte 1" nel titolo eh eh eh) scriverò un articolo che illustri qualche esempio pratico di implementazione delle interrupt. Spero di riuscirci nel minor tempo possibile.