Progettazione di sistemi embedded a basso consumo: dal silicio al software

Keith Odland
Director of marketing
Microcontroller products
Silicon Laboratories

II parte – Il ruolo del software

La progettazione di sistemi a basso consumo è un processo olistico che prende in considerazione silicio, software e relativi tool di sviluppo. Una corretta gestione delle relazioni fra questi tre elementi permette ai progettisti di sviluppare sistemi embedded caratterizzati da migliori prestazioni e da una maggiore efficienza energetica

Per realizzare applicazioni embedded efficienti dal punto di vista energetico è necessario un progetto software che utilizzi nella maniera più appropriata le risorse hardware. Per determinare ciò che è appropriato non bisogna considerare solo l'applicazione ma anche l'implementazione hardware. Allo stesso modo, tanto più è flessibile l'hardware – a livello di CPU, clock, tensione e utilizzo di memoria – tanto maggiori saranno i risparmi energetici che si potranno conseguire. I tool software di tipo "hardware-aware" permettono ai progettisti di sistemi embedded di poter valutare con maggior precisione gli ulteriori risparmi ottenibili.
Una possibile opzione prevede il ricorso alla tecnica DVS (Dynamic Voltage Scaling – variazione dinamica della tensione). Per poter utilizzare questa metodologia sono necessari convertitori dc-dc on-chip e circuiti di monitoraggio delle prestazioni che permettono di ridurre la tensioni di alimentazione quando l'applicazione non richiede l'esecuzione delle istruzioni alla massima velocità. In queste condizioni il sistema opera con consumi di potenza ridotti. I vantaggi che si possono ottenere sono funzione della tensione di ingresso e possono variare nel corso della vita del prodotto. Nelle figure 3 e 4 sono riportate le differenze nelle seguenti condizioni: nessuna variazione di tensione (VDD fissa), variazione statica della tensione (SVS – Static Voltage Scaling) e variazione attiva della tensione (AVS).
A questo punto è interessante segnalare che la strategia AVS può cambiare in base alla tensione di ingresso del sistema. Nell'esempio preso in considerazione, nel caso la tensione di ingresso sia pari a 3,6V, è più efficiente alimentare la logica interna e la memoria flash con un convertitore dc-dc ad alta efficienza. Nel momento in cui la tensione di ingresso diminuisce (cioè la batteria si scarica nel corso del ciclo di vita del prodotto), è invece più efficiente alimentare il sottosistema di memoria flash direttamente con la tensione di ingresso poiché la logica interna può operare a tensioni inferiori rispetto alla memoria. Per esempio la famiglia di MCU i3L1xx di Silicon Labs è caratterizzata da un'architettura di alimentazione dinamica che prevede sei domini di alimentazione separati e variabili che consentono di effettuare questa ottimizzazione di natura dinamica.
Solitamente i circuiti logici CMOS operano molto più lentamente con tensioni di valore ridotto. Nel caso l'applicazione possa operare senza problemi con prestazioni ridotte (come accade nel caso di protocolli di comunicazione che prevedono un invio dei dati a una velocità non superiore a quella di una frequenza

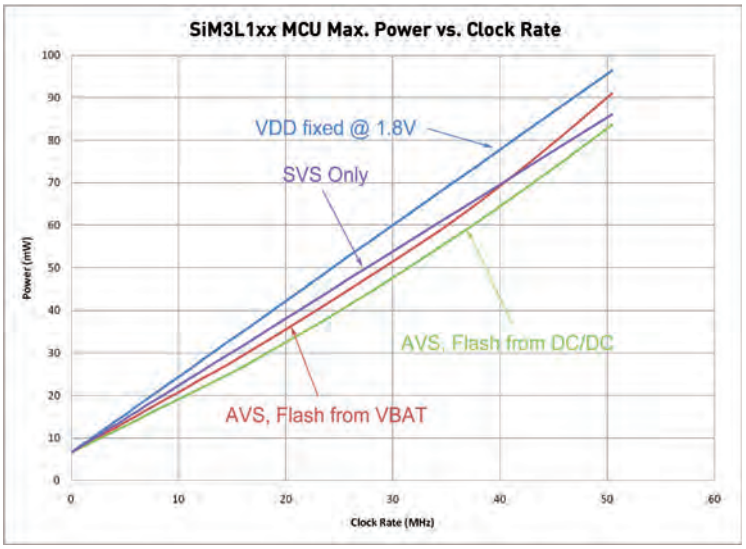

Fig. 3 – Effetti della variazione di tensione con VBAT = 3,6V

standardizzata), la riduzione di natura quadratica del consumo di energia ottenibile con tensioni di valore inferiore può comportare significativi risparmi energetici.

Le perdite rappresentano il limite inferiore della variazione di tensione. Se ogni operazione richiede tempi troppo lunghi, la dispersione inizierà a rappresentare il fattore predominante dell'equazione dell'energia, contribuendo a incrementare il consumo di energia complessivo. Per tale ragione può essere utile eseguire una funzione nel più breve tempo possibile e portare il processore in modalità "sleep" per minimizzare il fattore legato appunto alla dispersione.

Si consideri ad esempio un'applicazione che preveda l'uso di un sensore wireless che debba effettuare una notevole mole di operazioni DSP (Digital Signal Processing), come accade nel caso di un rilevatore di rottura vetri. In questo caso l'applicazione usa l'algoritmo FFT (trasformata di Fourier veloce) per analizzare le vibrazioni acquisite da un sensore audio relative alle frequenza caratteristiche generate dalla rottura del vetro. L'algoritmo FFT è relativamente complesso per cui la sua esecuzione a una frequenza più bassa (imposta dalla tensione ridotta) provocherà un sensibile incremento della dispersione anche utilizzando tecnologie di processo più datate. In questo caso l'approccio migliore è eseguire l'algoritmo a una velocità prossima a quella massima e tornare nello stato di "sleep" finché non arriva il momento di riportare qualche risultato al nodo host.

Il codice del protocollo wireless, dal canto suo, impone requisiti differenti. I protocolli radio sono caratterizzati da temporizzazioni fisse per gli eventi. In questi casi i protocolli possono essere gestiti interamente in hardware. La riduzione della tensione del core del processore è un'operazione utile. Pertanto il codice necessario per l'assemblaggio dei pacchetti e le trasmissioni viene eseguito a una velocità compatibile con il protocollo wireless.

L'aggiunta di blocchi hardware come ad esempio un blocco DMA (Direct Memory Access) "intelligente" può contribuire a modificare i compromessi in termini di energia. Parecchi controllori DMA, come ad esempio quello disponibili sul processore ARM Cortex-M3 nativo, richiede frequenti interventi da parte del processore. Controllori DMA più "intelligenti" che supportano una combinazione di messa in sequenza e concatenazione (chaining) consentono al processore di calcolare l'intestazione del pacchetto, cifrare i dati, assemblare i pacchetti e quindi trasferire l'operazione di passaggio dei pacchetti a intervalli appropriati ai buffer di memoria utilizzati dal front end radio. Per la maggior parte del tempo che il link radio è attivo il processore si trova nello stato di "sleep", con significativi risparmi in termini energetici.

Utilizzo della memoria

Con le attuali MCU a 32 bit, i progettisti possono utilizzare n maniera molto flessibile i blocchi di memoria. Solitamente la MCU metterà a disposizione un mix di memoria flash non volatile per memorizzare il codice e i dati unitamente a una memoria SRAM (Static Random Access Memory) per immagazzinare i dati temporanei. In molti casi il consumo di potenza durante gli accessi alla memoria flash sarà maggiore di quello richiesto per gli accessi alla memoria SRAM. Durante il normale utilizzo le letture della memoria flash sono molto più frequenti (di un fattore pari a tre) rispetto alle letture della memoria SRAM. Le operazioni di scrittura della memoria flash, che richiedono la cancellazione di interi blocchi e la successiva riscrittura mediante una lunga sequenza di impulsi di tensione di valore relativamente elevato, consumano una potenza ancora maggiore. In ogni caso, nella maggior parte delle applicazioni, le operazioni di scrittura della flash non sono frequenti e non influenzano quindi il consumo medio di potenza. Un ulteriore elemento da

tenere in considerazione nella valutazione del consumo di potenza di una memoria flash è la modalità di distribuzione degli accessi del processore.
All'interno di ogni blocco di memoria flash vi sono molteplici pagine, ciascuna delle quali può avere una dimensione massima di 4 kb.
Per supportare qualsiasi accesso, ciascuna pagina deve essere alimentata mentre ogni pagina non utilizzata può essere mantenuta in uno stato a basso consumo.
Nel caso una sezione del codice alla quale si accede su base regolare si trovi ai confini tra due pagine della memoria flash piuttosto che essere contenuta in una sola pagina, l'energia associata alla lettura delle istruzioni aumenta.
La riallocazione della memoria effettuata in modo tale che le sezione del codice e i dati a cui si ha accesso più di frequente siano contenuti all'interno di pagine singole può comportare significativi risparmi in termini di consumi nel corso della durata di una carica della batteria senza necessità di apportare modifiche hardware.
Spesso è utile copiare le funzioni di uso più frequente nella SRAM presente a bordo del chip e leggere le relative istruzioni da questa memoria piuttosto che dalla flash, anche se ciò può apparire come un modo meno efficiente di utilizzo della capacità di memoria.
I benefici che si ottengono in termini di durata della batteria compensano il "consumo di memoria" leggermente superiore.

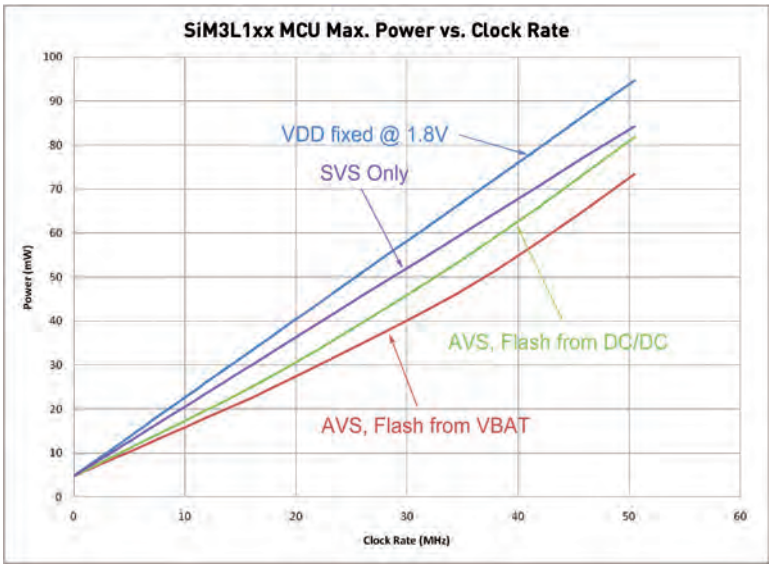

Fig. 4 – Effetti della variazione di tensione con VBAT = 2,4V

Ottimizzazione del codice

L'ottimizzazione del codice può anche avere un effetto dirompente sul tradizionale concetto di efficienza del codice. Per molti decenni i progettisti di sistemi embedded hanno concentrato la loro attenzione sull'ottimizzazione del codice in termini di occupazione di memoria, ad eccezione dei casi in cui le prestazioni risultano critiche. L'ottimizzazione energetica impone un insieme di criteri di valutazione completamente nuovo. Un aspetto da prendere in considerazione è l'utilizzo della cache on chip che è generalmente disponibile per le piattaforme a 32 bit.
L'ottimizzazione delle dimensioni del codice consente l'immagazzinamento della maggior parte dell'eseguibile nella cache, con conseguenti miglioramenti in termini di velocità e di consumi. Comunque le chiamate di funzione e le istruzioni di salto (branch) utilizzati per ridurre le dimensioni dell'applicazione attraverso il riutilizzo di codice comune possono dar luogo a conflitti imprevisti tra le sezioni del codice per la stessa linea della cache (i blocchi della cache sono chiamati linee). Ciò può dar luogo al fenomeno di cache thrashing (ovvero la CPU spende il proprio tempo ad allocare e disallocare aree di memoria senza più svolgere lavoro utile), dispendioso in termini energetici, nonché all'attivazione di un certo numero di pagine della flash quando le istruzioni devono essere prelevate dalla memoria principale. Nel caso di codice eseguito frequentemente nel corso della vita del prodotto, esso deve essere sufficientemente compatto da poter essere ospitato nella memoria cache ma senza salti o funzioni di chiamate.
Si consideri ad esempio un rilevatore di fumo: anche se viene innescato un allarme una volta alla settimana (a causa magari dell'eccessivo fumo prodotto dalle attività che vengono svolte nella cucina), si tratta di 520 eventi (invece degli oltre 315 milioni che si potrebbero verificare nel corso della vita del prodotto - stimata in dieci anni). Per la quasi totalità del tempo il codice acquisisce la lettura del sensore, controlla che non sia stata superata la soglia prefissata e quindi fa ritornare il core del processore nello stato di "sleep" finché questo non viene "svegliato" dal timer del sistema. Di tutte le letture del sensore acquisite dal rilevatore, meno dello 0,0002% darà luogo all'esecuzione di codice che genera un allarme. Il rimanente 99,9998% dell'esecuzione del codice sarà al di fuori del loop di lettura del sensore. Assicurarsi

che questo codice sia fatto girare in maniera sequenziale al di fuori della cache può essere la chiave per minimizzare il consumo di energia. Poiché esso gira così raramente, il codice rimanente può essere ottimizzato sfruttando tecniche più tradizionali.

Tool per l'efficienza energetica

Il supporto di tool idonei è un elemento di fondamentale importanza per garantire l'efficienza energetica della piattaforma MCU. La capacità di allocare le funzioni all'interno di singole pagine della memoria flash richiede la presenza di un linker in grado di "comprendere" la mappa di memoria dettagliata di ciascuna MCU target. Il linker può acquisire l'informazione fornita dallo sviluppatore relativa ai blocchi ai quali è consentito l'attraversamento dei confini delle pagine e generare un codice binario ottimizzato per garantire l'uso più efficiente in termini energetici della memoria non volatile. In linea di principio questo codice è anche usato per assicurare che funzioni e dati siano allocate in modo tale che quelle che vengono eseguite più frequentemente non entrino in conflitto con le linee delle cache.

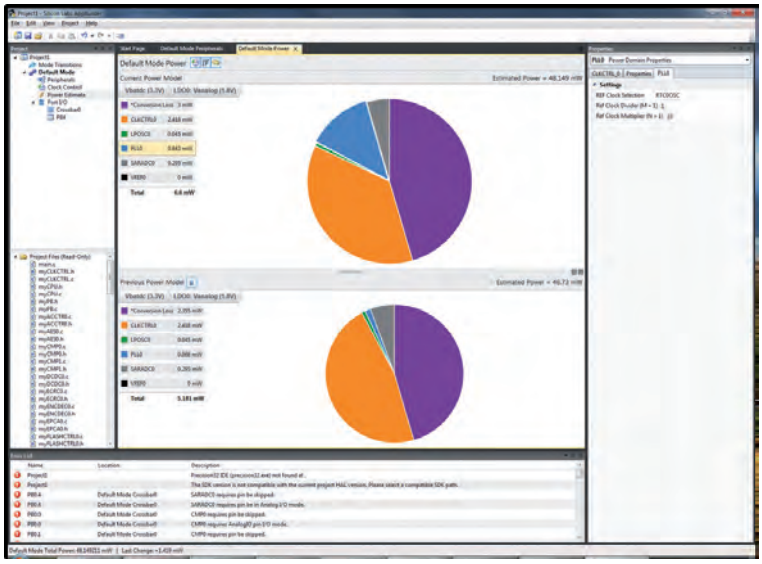

Fig. 5 – I tool di tipo "power-aware" consentiranno ai progettisti di ottimizzare i progetti in termini di consumi

Un tale livello di dettaglio può essere ottenuto con maggior facilità quando i tool sono forniti dal produttore della MCU, il quale conosce il layout della memoria e i requisiti di potenza di ogni piattaforma target. Un risultato di questo tipo è più difficilmente conseguibile con tool forniti da terze parti.La MCU ha anche una conoscenza completa e dettagliata dell'organizzazione delle diverse periferiche e dei bus on-chip. Tale conoscenza può essere utilizzata dai tool per fornire un valido ausilio ai progettisti per minimizzare i consumi. Un esempio tipico è rappresentato dall'ambiente AppBuilder realizzato da Silicon Labs (Fig. 5). Questo tool permette di definire la struttura per un'applicazione tramite semplici operazioni di "drag&drop" in un'opportuna area (canvas). AppBuilder può anche esaminare il setup delle periferiche e determinare se sono possibili modifiche per ottimizzare l'efficienza energetica. Per esempio, se un utilizzatore ha deciso di impiegare un UART per la sua applicazione e ha impostato la velocità a 9.600 baud, il tool vedrà il bus periferico dell'UART e determinerà l'impostazione adeguata. Il bus APB (ARM Peripheral Bus) utilizzato per ospitare blocchi quali UART e convertitori A/D può operare fino a 50 MHz. In questo caso la velocità è abbastanza alta (e consumerà più energia) rispetto a quanto necessario: il tool a questo punto chiederà all'utilizzatore se desidera ridurre la velocità di trasferimento dati dell'APB a un valore più appropriato.Il software AppBuilder fornisce al progettista altre informazioni sui consumi relativi a una specifica applicazione. Utilizzando una simulazione della MCU target (resa possibile da un'approfondita conoscenza delle caratteristiche del silicio), il tool può fornire un istogramma interattivo della corrente stimata non dell'intera applicazione, bensì del processore e di ciascuna periferica. I tool di sviluppo evolveranno al fine di diventare sempre più "power-aware". Solitamente le caratteristiche di debug come i breakpoint sono impostate sugli eventi (ad esempio letture/scritture della memoria). In futuro è plausibile che il supporto dei breakpoint evolverà in modo da gestire le problematiche legate ai consumi. Ad esempio se il consumo di potenza in un particolare punto oppure l'energia associata all'ultimo stato di sleep eccedono il target prefissato, il debugger interverrà mostrante quali parti dell'applicazione consumano una potenza maggiore di quella prevista. Consumi maggiori di quelli previsti e informazioni sulla posizione del codice nella mappa di memoria sono indizi vitali che aiutano gli sviluppatori software ad adottare le misure più idonee. ■

(La I parte dell'articolo è stata pubblicata sul numero 430 - settembre di Elettronica Oggi)