Verifica e validazione del software embedded mediante piattaforme virtuali per applicazioni automobilistiche

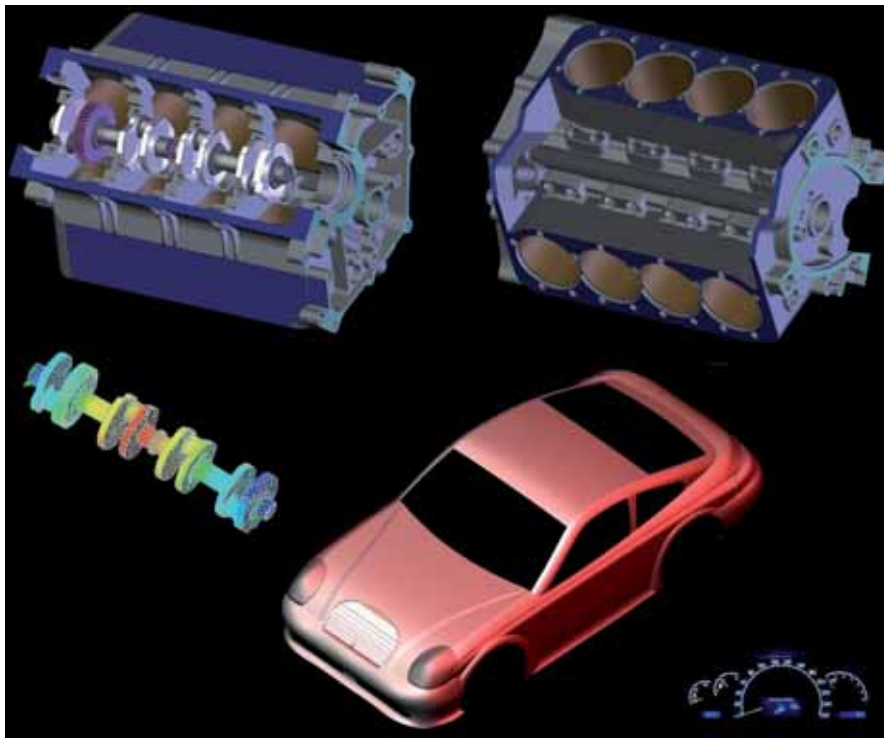
P. Cuenot
Siemens VDO Automotive SAS
a Continental Corp. Company

B. Tavernier
J.M. Talbot
VaST Systems Technology

I parte

Nel campo dei software embedded sviluppati per i sistemi di propulsione utilizzati in campo automobilistico si assiste a un notevole incremento della complessità imputabile all'aumento del grado di ottimizzazione delle funzioni di controllo/comando. L'utilizzo di piattaforme virtuali che abbinano precisione e velocità semplificano le fasi iniziali dello sviluppo software e garantiscono un collaudo più accurato del sistema. Nella prima parte dell'articolo viene spiegata la metodologia adottata per la modellazione hardware

Nel campo del software embedded sviluppato per i sistemi di propulsione utilizzati in campo automobilistico si assiste a un notevole incremento della complessità imputabile all'aumento del grado di ottimizzazione delle funzioni di controllo/comando. Il livello di complessità del controllo varia in funzione delle applicazioni: da quelle meno sofisticate come i sistemi a iniezione indiretta (Port Fuel Injection) a quelle più sofisticate come motori diesel ad alta pressione o controllo della gestione della potenza di veicoli ibridi. Gli algoritmi utilizzati aumentano a loro volta di complessità e richiedono più risorse per poter essere eseguiti con il livello di affidabilità richiesto dai sistemi safety-critical. In parallelo, l'introduzione di nuove funzionalità per il comfort dei passeggeri, l'aggiunta di nuovi sensori e aziona-



menti per l'adeguamento alle norme anti-inquinamento e il collegamento con i drive di comunicazione delle reti presenti all'interno di un veicolo contribuiscono a incrementare la complessità delle componenti hardware e software, con conseguente aumento della difficoltà di collaudo su un classico test bench.
Per affrontare con successo queste problematiche è necessario il ricorso a nuove metodologie e tool innovativi. In particolar modo l'utilizzo di piattaforme virtuali che abbinano precisione e velocità semplifica le fasi iniziali dello sviluppo software e garantisce un collaudo più accurato del sistema. Un altro problema è rappresentato dal progetto hardware seguito dallo sviluppo del software, che spesso si rivela un fattore critico durante il percorso di pianificazione di un progetto. Le piattaforme virtuali permettono, in linea generale, di procedere allo sviluppo del software già nelle prime fasi e, più in particolare, del software dipendente dall'hardware (come per esempio i driver dei dispositivi). Le piattaforme virtuali, inoltre, semplificano la distribuzione delle architetture hardware di riferimento a gruppi di sviluppo dislocati in aree geograficamente distanti tra loro. Le piattaforme virtuali mettono a disposizione dell'industria automobilistica strumenti aggiuntivi per affrontare le problematiche di natura economica in quanto semplificano l'ottimizzazione del Design-to-Cost, la metodologia che permette di conseguire importanti traguardi sia in termini di soddisfazione del cliente, velocità di sviluppo, reattività ai cambiamenti del mercato, sia in termini di efficienza e quindi di riduzione dei costi globali di prodotto/processo. I migliori compromessi tra le componenti hardware e software e gli effetti della configurazione hardware possono essere valutati mediante una prototipazione della piattaforma anticipata e più veloce. Risulta inoltre possibile la verifica di nuove architetture hardware che contengono circuiti ASIC al fine di ottimizzare le prestazioni.
Il problema principale di questa tecnologia è la disponibilità di modelli hardware e la loro conformità alle rispettive realizzazioni su silicio. La simulazione delle prestazioni e la standardizzazione dello stile di modellazione hardware sono elementi vincolanti per l'uso industriale nei sistemi embedded per applicazioni automotive. La modellazione basata su transazioni (TLM – Transaction Level Modelling – ovvero rappresentazioni astratte dei modi di comunicazione) supportata dall'iniziativa systemC OSCI [1] è adeguata per il conseguimento di tali obiettivi. Nel corso di questo articolo viene proposta una piattaforma virtuale basata su una versione semplificata del microcontrollore MPC5554 di Freescale, che comprende un modello di ASIC esterno sviluppato in SystemC da Continental. Questa piattaforma è stata sviluppata utilizzando il tool CoMET di VaST Systems Technology. In particolare sarà mostrato in che modo questa piattaforma può semplificare lo sviluppo di software nelle prime fasi e il collaudo di non regressione senza la necessità di disporre dell'hardware fisico. Nella prima parte viene presentata la metodologia impiegata per la modellazione hardware utilizzando l'approccio TLM, mentre nella seconda viene proposto il "case study" di utilizzo e fornita una descrizione generale della piattaforma. Nella terza parte vengono forniti i risultati tecnici e confrontati con gli obiettivi iniziali. Nella parte conclusiva sono evidenziati i vantaggi di tale approccio e le possibili estensioni ad applicazioni industriali.

Tecnologia di modellazione

Velocità e precisione sono le due caratteristiche salienti di una piattaforma virtuale che consenta lo sviluppo del software per applicazioni nel campo dei sistemi di propulsione (o più in generale nelle applicazioni safety critical). Sfruttando un approccio TLM, i modelli di core VaST come e200z6 sono in grado di far girare i classici benchmark come l'algoritmo di Viterbi a una velocità superiore a 80 MIPS (con un'accuratezza a livello di ciclo superiore al 99%) utilizzando un sistema equipaggiato con processore Pentium 4 operante a 2 GHz con 1 Gb di RAM. Il mantenimento di velocità di questo tipo a livello di piattaforma può essere ottenuto mediante l'astrazione dei segnali e del comportamento a un livello superiore rispetto a quello RTL, senza per questo compromettere l'accuratezza della temporizzazione a livello di sistema.
Di seguito vengono proposti alcuni principi di modellazione per ottimizzare la velocità di simulazione nella piattaforma. Dapprima si descriverà la modellazione dei segnali di clock e in che modo sia possibile astrarre a livello TLM canali di comunicazione quali un collegamento seriale, un bus interno o un bus CAN. In seguito sono presentati i principi generali per ridurre il numero di eventi in un modello di periferiche scritto in SystemC, come ad esempio un timer. Si vedrà che questo modello può essere impiegato per l'acquisizione degli ingressi di frequenza (come l'acquisizione degli eventi in un'applicazione di un sistema di propulsione) in modo veloce e preciso.

Modellazione del clock

Per ottenere una piattaforma virtuale che sia al contempo veloce e precisa alcuni tra i segnali più critici da astrarre sono quelli di clock. Poiché i kernel di simulazione per la modellazione di piattaforme virtuali sono di tipo event-driven, un clock di tipo impulsivo genera un evento su ogni fronte, anche se molti di questi eventi non rivestono alcuna utilità pratica. Per esempio, se una periferica è in uno stato inattivo ma riceve un impulso di clock con un ingresso per l'innesco di una funzione di callback (per esempio un SC_METHOD sensibile ai fronti positivi), il kernel di simulazione eseguirà del codice, anche se ciò non avrà alcuna influenza sul comportamento della piattaforma. Per migliorare la velocità di simulazione, è possibile eseguire l'astrazione dei segnali di clock dal loro periodo, conseguendo numerosi vantaggi tra cui:
- un evento è generato solamente quando cambia il periodo del segnale di clock. Il verificarsi di questo genere di evento è molto meno frequente rispetto all'impulso di clock;
- è possibile mantenere la precisione dell'intero ciclo sostituendo il comportamento pilotato dal clock con calcoli del clock e annotazione della temporizzazione e generare solo gli eventi osservabili;

- con un'astrazione di questo tipo, si ottiene la temporizzazione dinamica nei modelli della periferica.
Con la sua API di modellazione, i tool di simulazione di VaST forniscono un insieme di funzioni utili per la ricostruzione dei segnali di clock a partire da una rappresentazione del clock basato sul periodo, fattore questo che contribuisce ad aumentare il livello di astrazione della modellazione. Questa API, inoltre, semplifica l'esecuzione di operazioni standard per la modellazione delle periferiche come ad esempio la programmazione di un callback in un numero definito di clock tick (ovvero la più piccola unità di tempo durante la quale può essere eseguita un'elaborazione).
Nella modellazione SystemC, gli oggetti della primitiva sc_clock sono modelli di clock impulsivo. Essi dovrebbero essere evitati e sostituiti da un segnale intero che modella il periodo di clock. Questo tipo di astrazione richiederà più calcoli rispetto a quelli necessari utilizzando l'API di VaST: ciò può essere gestito in maniera molto semplice per mezzo di una classe che si occupa di tutte le conversioni tra il numero di tick di clock e il tempo assoluto (come richiesto da SystemC).

Canali di comunicazione

Per effettuare l'astrazione delle funzioni di comunicazione su un canale specifico si seguono le medesime linee guida:
- i frame dei dati sono trasferite dal "produttore" al "consumatore" in un passaggio;
- solo le parti osservabili del protocollo vengono modellate e (se richiesto) associate agli eventi. Se il produttore (o il consumatore) non richiede questi eventi esterni (come fasi parziali di una transazione di tipo burst), gli eventi sono eliminati e ciò contribuisce a conferire una velocità superiore al protocollo;
- i canali con arbitrazione sono modellati da un "engine del protocollo" specifico che si occupa dello scheduling di ogni transazione e assicura l'accesso al bus a ogni master.

Protocollo seriale

L'astrazione del protocollo seriale viene eseguita in modo tale da consentire solamente la trasmissione delle informazioni di interesse dal punto di vista del sistema. Nei casi più semplici, a livello di sistema, le informazioni sono le seguenti:
- velocità di trasferimento;
- timestamp della fine del trasferimento;
- dati trasferiti.
Invece di modellare ciascun trasferimento di bit a ogni tick di clock, è possibile effettuare l'astrazione di tutti questi eventi per mezzo di una singola transazione trasferendo i dati in un singolo evento: per il calcolo della temporizzazione si moltiplica il periodo della velocità di trasferimento per il numero dei bit trasferiti.

Astrazione del bus

Le transazioni del bus sono astratte per mezzo di eventi osservabili dal punto di vista di un master e di uno slave. Per esempio, in una transazione del bus interna, solo i seguenti eventi sono osservabili da un master che richiede la transazione:
- ottenere l'accesso al bus (evento di assegnazione);
- completamento, se richiesto, delle fasi intermedie della transazione, come nel caso di un accesso burst (evento di completamento parziale);
- completamento della transazione (evento completo).
Indipendentemente da questi eventi associati con il protocollo del bus, i dati possono essere trasferiti in un singolo passo.
Il medesimo tipo di astrazione è usato nel modello del bus CAN dove un modello di "engine del protocollo" specifico ha il compito di risolvere le questioni di priorità.

Solamente gli eventi osservabili sono modellati: questo si verifica ad esempio quando un frame è accettato o quando un frame viene ricevuto con successo.

Modellazione efficiente di SystemC

Poiché lo stile di modellazione SystemC è di tipo event-driven, l'efficacia nell'implementazione di una periferica si ottiene limitando il numero di eventi generati dal modello. Per far ciò la prassi comune è quella di combinare tecniche di modellazione come:
- previsione dell'evento. Ciò significa utilizzare il contesto attuale per prevedere un evento che si verificherà in futuro. Naturalmente un cambiamento del contesto nella periferica (per esempio una modifica nei registri della periferica o un mutamento del clock di riferimento) deve essere preso in considerazione, con conseguente riprogrammazione degli eventi futuri;
- elaborazione e memorizzazione del contesto. Ciò significa memorizzare nel modello della periferica ogni informazione necessaria per aggiornare lo stato della periferica quando richiesto (ovvero quando l'utente lo deve utilizzare). Nell'esempio del timer si vedrà come la memorizzazione di un timestamp permetta di evitare centinaia di eventi senza penalizzazioni in termini di precisione.

Un altro problema di grande importanza, più legato all'implementazione SystemC che non al principio di astrazione, è il seguente: poiché un SC_THREAD è implementato come thread nel kernel OSCI, questo genere di costrutto implica un overhead dovuto alla commutazione tra processi (task switching). In molti casi ciò può essere evitato utilizzando un approccio di modellazione del callback che preveda l'abbinamento di un SC_METHOD e un evento interno usato per innescare il callback.

Un semplice modello di timer per il conteggio degli eventi

Di seguito si vedrà come i principi generali appena esposti possano essere implementati in un semplice timer e come questa metodologia metta a disposizione "a titolo gratuito" funzionalità di temporizzazione dinamica. Questa funzionalità può essere impiegata per conteggiare e misurare eventi, dove l'ingresso potrebbe essere un segnale di clock con una frequenza variabile (come accade comunemente nelle applicazioni automobilistiche).

Interfaccia esterna del timer

Una semplice interfaccia del timer sarà composta da:
- un'interfaccia bus formata da una porta del bus e una porta del clock del bus. L'interfaccia del bus sarà collegata al bus della periferica accessibile dalle porte del bus del core e200z6;
- un ingresso di "reset";
- un ingresso "TimerClock" usato per fornire il riferimento di clock usato dal timer;
- un'uscita "TimerInt". Questa porta è un'uscita logica utilizzata per richiedere un interrupt. Questa uscita è solitamente collegata al controllore dell'interrupt.

Registri del timer

È importante sottolineare il fatto che non si parlerà del protocollo TLM usato per l'accesso ai registri interni. I tool di simulazione come quelli sviluppati da VaST supportano parecchi protocolli di bus e aggiungono automaticamente un transactor (ovvero un adattatore) del bus accurato a livello di ciclo per supportare modelli untimed (ovvero non presuppongono nessuna nozione di tempo) PV (programmers View). In questo contesto si farà l'ipotesi che il protocollo TML consenta un accesso al bus per la scrittura o la lettura di registri interni. Per ulteriori dettagli è possibile consultare. Il timer preso in considerazione utilizzerà cinque registri:
- GTR (Global Time Register) che contiene il numero dei clock tick trascorsi;
- OS_PERIOD contiene i valori target dei clock tick per la generazione di un interrupt;
- INT_ENABLE utilizzato per abilitare gli interrupt quando GTR = OS_PERIOD;
- INT_FLAG riflette lo stato dell'uscita "TimerInt". Questo registro è usato per cancellare l'interrupt;
- TIMER_ENABLE utilizzato per abilitare/disabilitare il timer.

Modellazione del comportamento del timer

Di seguito sono riportate le specifiche del timer preso in considerazione:
- quando si verifica un reset, tutti i valori dei registri sono posti a zero così come l'uscita TimerInt;
- quando il timer è abilitato, il registro GTR può essere letto dall'interfaccia del bus per ottenere il numero di fronti dall'ultimo interrupt;
- quando GTR = OS_PERIOD e INT_ENABLE = 1 viene azionato un interrupt: ciò vuole dire scrivere un 1 sull'uscita logica TimerInt e nel registro INT_FLAG;
- l'uscita TimerInt può essere azzerata scrivendo zero nel registro INT_FLAG;
- la temporizzazione dinamica deve essere tenuta in considerazione.

Anche in presenza di una specifica molto semplice, uno stile di modellazione inefficace può essere causa di scarse prestazioni di simulazione. Per esempio, la modellazione di questo timer mediante un clock impulsivo utilizzato per incrementare il registro GTR potrebbe implicare una velocità di simulazione dell'ordine dei KIPS (Kilo Instruction Per Second) e non dei MIPS, anche se il core e200z6 può garantire velocità superiori a 60 MIPS. Inoltre, è inutile cercare di mantenere un valore continuamente aggiornato nel registro GTR specialmente se l'utilizzatore (ovvero il software) non lo legge.

Per modellare il comportamento si farà ricorso ai tre principi delineati in precedenza.
Nessun SC_THREAD. Il comportamento farà ricorso ai seguenti SC_METHOD:
- metodo di Reset, eseguito quando cambia la porta di ingresso di reset;
- metodo ClockChanged, eseguito quando cambia il valore della porta di clock;
- metodo MatchEvent. Esso verrà avviato utilizzato un evento interno usato per la predizione dello stesso.

Elaborazione e memorizzazione del contesto. Come spiegato in precedenza, non è necessario aggiornare il valore del timer attuale (registro GTR). La sola azione da compiere è memorizzare un timestamp associato al valore nel registro GTR.
Quando l'utente legge il registro è possibile calcolare il valore nel registro GTR appropriato aggiungendo il numero dei clock tick corrispondenti al tempo trascorso dal timestamp memorizzato nel GTR e il tempo attuale (Listato A).

```
void cTimer::UpdateCurrentTime(void) {
 if (mTIMER_ENABLE) {
 sc_time NowTime = sc_time_stamp();
 mGTR += (int)((NowTime - mTimeStamp)
/mClockPeriod);
 mTimeStamp = NowTime;
 }
}
```

Previsione dell'evento. Quanto il valore nel registro GTR è coerente, è facile prevedere quando è necessario azionare l'interrupt. Se il timer è abilitato, esso si verificherà nei tick compresi tra OS_PERIOD – GTR: utilizzando il periodo del clock, è possibile pianificare in modo semplice i futuri eventi di interrupt.
Nel caso di cambiamento del contesto (variazioni del periodo di clock, della configurazione del registro, e così via) questo evento deve essere annullato e pianificato nuovamente (Listato B).

```
void cTimer::SetupNextMatch(void) {
 mTimerEvent.cancel();
 if (TIMER_ENABLE) {
 intmatchTick = mOS_PERIOD-mGTR;
 if (matchTick>=0)
 mTimerEvent.notify(
(double)(matchTick*mClockPeriod.value()),
SC_PS
 );
 }
}
```

Se non si verifica alcun cambiamento nel contesto, l'evento mTimerEvent si verificherà nel tempo previsto. Ciò darà avvio al TimerEvent SC_METHOD, dove verrà scritto un 1 sulla porta di uscita TimerInt e nel registro TIMER_INT.

Comportamento della temporizzazione dinamica

Per la temporizzazione dinamica è necessario tenere presente il fatto che la frequenza del clock può cambiare. Quando ciò si verifica è necessario procedere come segue:
- aggiornare il valore attuale del registro GTR: Poiché ClockPeriod è utilizzato per calcolare il valore del registro GTR, esso deve essere costante dal momento in cui sono stati memorizzati per l'ultima volta il valore nel registro GTR e il timestamp;
- ripianificare l'evento di accoppiamento: la previsione è stata fatta con il valore del periodo precedente (listato C).

```
voidcTimer::ClockChanged(void) {
 UpdateCurrentTime();
 mClockPeriod=mTimerClockPort.read() *
sc_get_time_resolution();
 SetupNextMatch();
}
```

Conteggio dell'evento

Questo timer può ora essere utilizzato per generare interrupt a una frequenza multipla della frequenza dell'ingresso. Questo controllo è tipico delle applicazioni dei sistemi di propulsione per generare un clock di sistema interno sulla base delle informazioni provenienti dai sensori.
La parte mancante è la generazione del testbench, che fornisce lo scenario evolutivo del periodo di clock. Questa operazione può essere effettuata mediante tool per la modellazione a livello di sistema come Matlab/Simulink o Saber collegati alla piattaforma virtuale.
Questo testbench sarà campionato a una velocità definita dal periodo di sincronizzazione tra l'ambiente di simulazione del sistema e il kernel di simulazione della piattaforma virtuale.
La frequenza dell'evento di ingresso sarà convertita al periodo di clock in picosecondi (risoluzione del kernel VaST) e trasferita al timer definito in precedenza.

Siemens VDO Automotive SAS
(a Continental Corp. Company) www.conti-online.com

VaST Systems Technology www.vastsystems.com