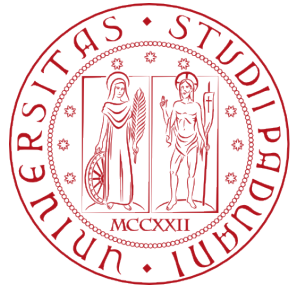
Co-Simulazione di un sistema controllato da microprocessori

Diego Casella, Fabio Marcuzzi, Paolo Martin

February 24, 2015



Università degli Studi di Padova

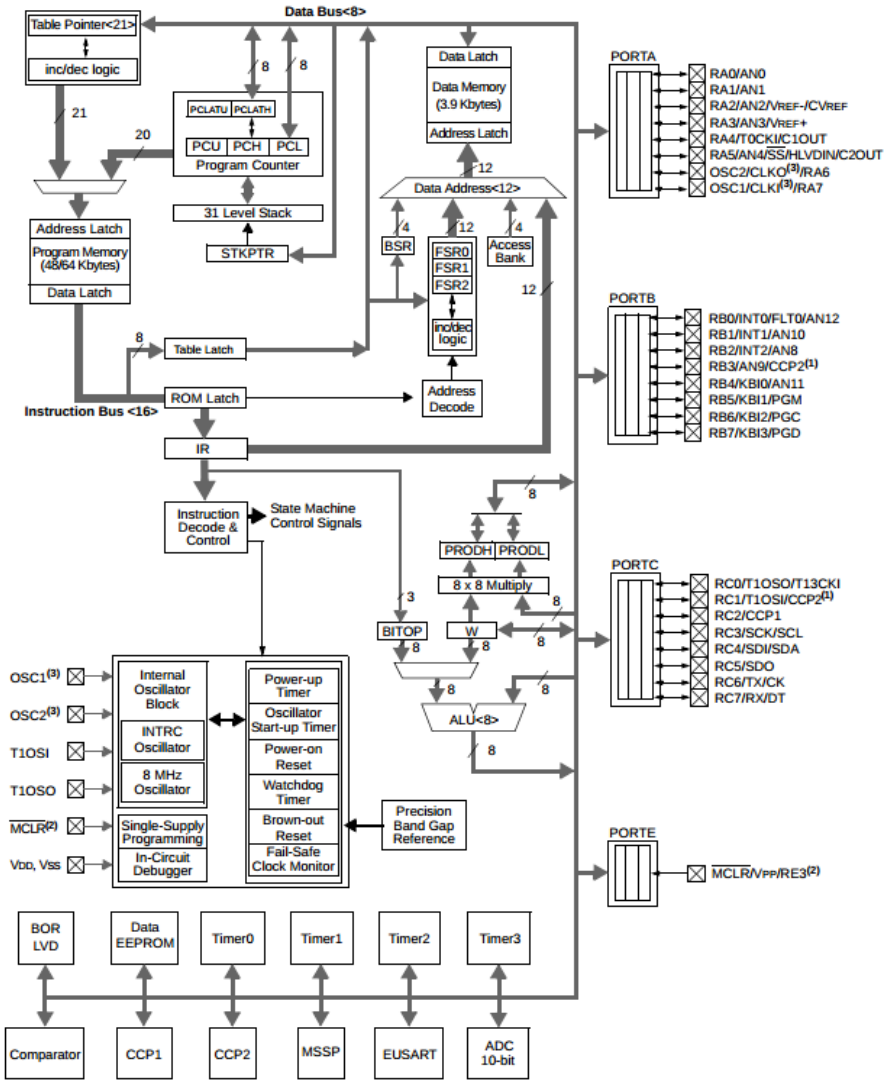
Contenuti

Introduzione

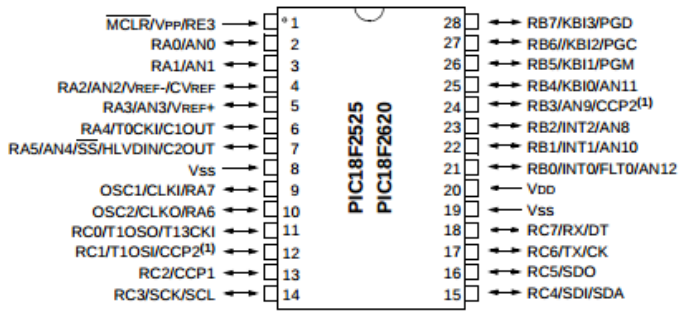
Com'é accaduto ormai da parecchio tempo in vari settori dell'ingegneria (civile, aerospaziale, meccanica), anche per l'ingegneria dei sistemi di controllo la prototipazione virtuale é diventata una metodologia possibile e desiderabile, quando non addirittura necessaria.

Mediante la simulazione numerica del sistema controllato é possibile studiare il funzionamento del sistema con una risoluzione temporale impensabile nella sperimentazione fisica, ed analizzare il sistema dal punto di vista:

- dell'interazione tra firmware e sistema fisico;
- dell'analisi numerica degli algoritmi implementati nel firmware.



(a) Schema interno



(b) Pinout fisico

Microcontrollori/DSP

- Esistono moltissime varianti di questi dispositivi: diciamo almeno 10 produttori con circa 1000 dispositivi ciascuno.
- Relativamente pochi dispositivi (alcune decine) condividono la stessa CPU (8-16-32 bit, vari set di istruzioni, es. DSP).
- Pochissimi dispositivi (al massimo poche decine) condividono le stesse periferiche, che per numerositá variano comunque da dispositivo a dispositivo.
- Il progetto del sistema di controllo é legato alle specifiche periferiche (tipologia e numerositá) presenti nel dispositivo.
- Il firmware interagisce direttamente con le periferiche (i sistemi operativi generalisti non sono quasi mai usati nei microcontrollori) e puó dunque girare quasi esclusivamente sul singolo dispositivo scelto: la portabilitá del firmware é molto bassa.

$\rightarrow$ **la co-simulazione richiede il simulatore dello specifico dispositivo utilizzato**.

Ogni microcontrollore é una macchina complessa: il suo reale funzionamento dipende dai valori programmati in una moltitudine di registri che governano le periferiche.

$\rightarrow$ **Il simulatore é un software complesso ed impegnativo da debuggare**.

Il funzionamento fisico interno del microcontrollore non é oggetto della co-simulazione, viene dato per buono; viene simulato il funzionamento logico.

Il modello del sistema di controllo e del sistema multifisico controllato hanno invece significato fisico e quindi viene simulato il loro funzionamento fisico. Il confine tra modello logico e fisico é il pinout del microcontrollore, dove i livelli logici diventano livelli di tensione elettrica.

La velocitá di funzionamento é una questione fondamentale: sono macchine RISC con clock 5-40 MHz e quindi la complessitá computazionale del simulatore é elevata.

Non é possibile parallelizzare una singola esecuzione del firmware.

Co-simulazione

Queste criticitá fanno sí che realizzare un simulatore performante é uno sforzo notevole e gestire la diversificazione lo é ancor di piú.

Inoltre, un simulatore ha senso se puó essere usato in modo friendly: deve essere ben integrato con il progetto firmware e quindi con il suo ambiente di sviluppo, ed avere una buona GUI.

$\rightarrow$ **Questi ultimi aspetti fanno preferire Java come linguaggio per lo sviluppo di simulatori, mentre questioni di performance indicherebbero il C come ottimale**.

Co-simulazione

Java difatti permette una rapida realizzazione dell'idea di simulatore che si ha in mente, permettendo allo sviluppatore di focalizzarsi sull'architettura dell'implementazione.

$\rightarrow$ Ció non esclude il fatto che, una volta realizzato il simulatore e delineata la sua struttura funzionale, esso non possa essere reimplementato in linguaggi piú performanti come C/C++.

Co-simulazione

Un altro requisito che la co-simulazione impone é il controllo fine del simulatore: non é sufficiente avviare il simulatore ed attendere l'ottenimento dell'andamento dei pin a simulazione conclusa, bensí:

- conoscere i valori assunti dai registri/memorie (debugging, tracing variabili)

Co-simulazione

Un altro requisito che la co-simulazione impone é il controllo fine del simulatore: non é sufficiente avviare il simulatore ed attendere l'ottenimento dell'andamento dei pin a simulazione conclusa, bensí:

- modificare il flusso del programma originario (in maniera da testare differenti implementazioni per una determinata routine, al fine da determinarne la piú efficiente o performante)
- controllare il flusso dell'esecuzione firmware (breakpoints)

Co-simulazione

Un altro requisito che la co-simulazione impone é il controllo fine del simulatore: non é sufficiente avviare il simulatore ed attendere l'ottenimento dell'andamento dei pin a simulazione conclusa, bensí:

- fornire utility di conversione di segnali dai/ai pins (adapters)

Co-simulazione

Una tendenza recente é appoggiarsi a NetBeans come framework per far interagire un simulatore con un programma che lo pilota, come é il caso della co-simulazione. Le motivazioni sono le seguenti:

- Ambiente che fornisce un framework generico per realizzare il proprio IDE (interfaccia,editor,register view, debugger)

Co-simulazione

Una tendenza recente é appoggiarsi a NetBeans come framework per far interagire un simulatore con un programma che lo pilota, come é il caso della co-simulazione. Le motivazioni sono le seguenti:

- Svincola i programmatori dal peso di realizzare la GUI che lo gestisce e governa, permettendo loro di concentrarsi sulla realizzazione del solo simulatore

Co-simulazione

Una tendenza recente é appoggiarsi a NetBeans come framework per far interagire un simulatore con un programma che lo pilota, come é il caso della co-simulazione. Le motivazioni sono le seguenti:

- Fornisce pattern OOP (Lookup, Command pattern) ed utility (File handling trasparente) pronti all'uso

Prestazioni

Come ottimizzare questa architettura di co-simulazione ?

L'attuale stato dell'arte della co-simulazione prevede l'interazione tra un plugin apposito realizzato per NetBeans, che gestisce la simulazione del microcontrollore, il quale scambia e riceve dati col simulatore vero e proprio tramite chiamate RMI.

Prestazioni

Come ottimizzare questa architettura di co-simulazione ?

Con l'aumentare:

- Del numero dei registri che si vogliono tracciare/debuggare;
- Dei pin che sono connessi al simulatore;

Il traffico dati che RMI deve sostenere cresce considerevolmente

Esempio

Un microcontrollore operante a 10MHz ($T_{cy} = 4\ \mu$s), nessun pin utilizzato, un solo registro a 8bit tracciato, esegue un numero di operazioni pari a

$$(1/4\mu\text{s}) = 250000\ \text{op/s}$$

Dunque, volendo tracciare un registro ad 8 bit, generiamo un traffico dati pari a

$$250000*8 = 2\ \text{Mbit/s} = 0.25\ \text{MB/s}$$

Esempio (cont.)

Se ora vogliamo tracciare un pin, rappresentato con un double formato 64bit IEEE 754, otteniamo

$$250000*64 = 16 \text{ Mbit/s} = 2 \text{ MB/s}$$

Prestazioni

Le casistiche diventano molto piú articolate di quanto mostrato nel precedente esempio:

- i devices presenti in commercio possiedono configurazioni da 12, 14, 16, 18, 28, 40, 44 pin;
- molteplici architetture: 8, 16 o 32 bit;

Inoltre, ogni pin/registro necessita di essere identificato tramite una stringa univoca, che dunque deve essere anch'essa trasmessa tramite RMI al simulatore del controllore.

Prestazioni

Il traffico dati costantemente serializzato e de-serializzato tra applicativo e simulatore tramite RMI é consistente.

Soluzione addottata:

Raggruppare le (de)serializzazioni in una unica chiamata (getPin(name) vs getPins())

Prestazioni

Il traffico dati costantemente serializzato e de-serializzato tra applicativo e simulatore tramite RMI é consistente.

Possibile soluzione futura:

Rendere disponibile l'intero processo di co-simulazione all'interno dello stesso ambiente NetBeans

Prestazioni

Qualé il grado di concorrenza raggiungibile?

Il solutore della co-simulazione é implementato su di un singolo thread: si possono raggiungere prestazioni migliori dedicando idealmente un thread per componente, in modo tale da eseguire l'avanzamento di un passo di simulazione in maniera parallela invece che sequenziale.

Prestazioni

Qualé il grado di concorrenza raggiungibile?

Una ulteriore evoluzione del co-simulatore potrebbe includere uno scheduler che analizzi i tempi di esecuzione dei vari componenti, individuando quelli che presentano criticitá dal punto di vista dei tempi di esecuzione. In tale maniera é possibile porli in esecuzione su thread separati, cosí da eseguire nel contempo i restanti componenti, piú veloci.

Prestazioni

Come ottimizzare la API del simulatore per rendere piú veloce possibile la co-simulazione?

- migliorarne la velocitá (ad es. rispetto ai nostri simulatori scritti in C)
- dare la possibilitá al fruitore del simulatore di farsi notificare in maniera automatica dello stato dei registri e pin, registrandoli in fase di pre-simulazione, invece di fare delle query ad ogni ciclo di iterazione

Q&A