Modellazione in SystemC TLM di un Moltiplicatore Floating-point Single Precision ed integrazione in una Virtual Platform

Nicola Serlonghi - VR445270

***Sommario*—Questo documento presenta l'implementazione di un sistema per il calcolo della moltiplicazione in virgola mobile a singola precisione implementato nei diversi livelli di astrazione di SystemC TLM e su una Virtual Platform.**

I. Introduzione

Il seguente documento illustra il modo in cui è stato implementato e simulato un sistema per il calcolo della moltiplicazione in virgola mobile nei diversi livelli di astrazione di SystemC TLM come AT, LT e UT, il tutto è stato comparato con il sistema SystemC RTL sviluppato nel progetto precedente ed opportunamente modificato in modo tale che eseguisse una sola moltiplicazione. L'obbiettivo di tutto ciò è osservare come la differenza d'astrazione migliora o peggiora il tempo di simulazione.

In una seconda fase, il moltiplicatore realizzato in VHDL e Verilog, illustrato nella relazione precedente, è stato connesso ad una virtual platform già esistente (COM6502-Splatters) verificandone la funzionalità attraverso la simulazione Vivado. Lo scopo di questa implementazione è osservare come un sistema interagisce con l'architettura sottostante.

Nel seguito del documento descriverò le varie implementazioni e le scelte progettuali fatte.

II. Background

A. SystemC TLM

SystemC TLM è un una versione di SystemC che supporta la descrizione al livello TLM. I casi d'uso che portano ad optare per questo tipo di soluzione sono: lo sviluppo di un software più veloce, un'analisi delle prestazioni architetturali e la verifica dell'hardware. SystemC TLM offre anche la possibilità di modellare le funzionalità di un sistema ad un livello più astratto, quindi con meno dettagli d'implementazione, rispetto all'RTL, dando più importanza alla transazione dei dati. Il sistema è stato implementato in tre diversi stili di codifica:

- $AT(Approximately - timed)$ in cui ogni transazione viene sincronizzata in più punti. Nella versione base sono quattro: inizio della richiesta, fine della richiesta, inizio della riposta e fine della risposta.
- $LT(Loosely - timed)$ in cui ogni transazione viene temporizzata solo all'inizio e alla fine della transazione.
- $UT(untimed)$ dove il concetto di temporizzazione di una transazione non è considerato.

Le transazioni sopra citate si svolgono tra due entità: l'iniziatore, colui che inizia la transazione, e il target, colui con cui l'iniziatore vuole interagire. Durante la comunicazione essi si scambiano un payload generico contenente i dati di controllo; lo standard prevede la possibilità di estendere quest'ultimo per adattarlo ai diversi protocolli che possono essere utilizzati.

B. Virtual Platform

Le Virtual Platform sono una rappresentazione astratta dell'architettura sulla quale un sistema embedded dovrà essere eseguito. In questo caso è stata utilizzata una piattaforma preesistente: COM6502-Splatters, che ci ha permesso di sfruttare la facilità di estensione e modifica, caratteristica delle Virtual Platform, permettendo cosi di anticipare la fase di sviluppo del sistema ed avere pieno controllo dell'hardware sul quale è stato implementato il tutto.

III. Metodologia applicata

A. SystemC TLM

Come già detto in precedenza il moltiplicatore è stato implementato nelle tre diverse versioni di TLM: UT, LT e AT4. La struttura utilizzata dopo aver analizzato il problema è univoca in tutte e tre le versioni e presenta un moltiplicatore, che rappresenta il target, ed una testbench, che funge da iniziatore. Il payload inviato nelle transazioni è stato esteso con un puntatore ad una struttura che presenta al suo interno: i due operandi (input) e il risultato della moltiplicazione (output).

- $Untimed$: Non abbiamo il concetto di temporizzazione di una transazione quindi, esse avvengono senza il passaggio di riferimenti temporali. Per garantire comunque la sincronizzazione si chiama la primitiva $b\_transport$ con interfaccia bloccante. Il flusso di funzionamento è il seguente: una testbench richiama la funzione allegando il payload. Il target riceve poi la chiamata, esegue la moltiplicazione tra gli operandi e ritorna il risultato.
- $Loosely - timed$: La comunicazione tra le varie componenti avviene, come per UT, attraverso la primitiva $b_transport$, ma in questa implementazione abbiamo la nozione di tempo per la sincronizzazione. La particolarità di questa codifica è il Temporal Decoupling che consiste nella capacità della testbench di proseguire la sua esecuzione al di fuori del tempo di simulazione, finché non raggiunge un punto di sincronizzazione o non termina il tempo a sua disposizione.
- $Approximately - timed4fasi$: Tutto inizia dalla testbench che chiama la primitiva $nb_transport_fw$ (inizio

della richiesta) e si mette in attesa di una riposta di avvenuta ricezione da parte del target (fine della richiesta). A questo punto il moltiplicatore calcola la moltiplicazione e chiama la primitiva $nb\_transport\_bw$ (inizio della risposta), ed in seguito l'iniziatore può continuare la sua esecuzione (fine della risposta).

Per ottenere dei dati rilevanti, ogni tipologia ha eseguito 10000000 moltiplicazioni ed il modulo RTL è stato privato del secondo moltiplicatore e di tutta la logica annessa per gestirlo. In modo da avere dei dati confrontabili con le versioni in SystemC TLM.

Osservando i dati dei tempi di simulazione raccolti, riportati nella tabella I, è possibile notare che più il tempo di simulazione è lungo, maggiore è il controllo del tempo e della sincronizzazione dei processi. In tabella I troviamo riportati 3 tipologie di tempo differenti:

- $Realtime$: rappresenta il tempo di esecuzione effettivamente impiegato dal calcolatore per terminare l'esecuzione. Il suo valore è quindi influenzato da fattori esterni al sistema.
- $Usertime$: rappresenta il tempo impiegato dalla CPU per l'esecuzione effettiva del sistema.
- $System\ time$: rappresenta il tempo impiegato dalla CPU per l'esecuzione di system call per il processo.

	AT4	LT	UT	RTL
Real time	36m 42,481s	15m 58,214s	16m 1,320s	21m 51,346s
User time	0m 34,012s	0m 20,129s	0m 13,879s	1m 39,121s
System time	9m 70,153s	6m 1,058s	3m 40,931s	19m 31,355s

Tabella I
TEMPI DI ESECUZIONE SU 10000000 DELLE VARIE IMPLEMENTAZIONI DI SYSTEMC.

B. Virtual Platform

Il moltiplicatore in virgola mobile è stato integrato in una Virtual Platform già esistente. Per cui sono state aggiunte funzioni e collegamenti. Si è scelto di non collegare il moltiplicatore creato nel report precedente, ma di collegare direttamente sulla piattaforma i due moltiplicatori Verilog e VHDL come periferiche. Questa scelta è stata fatta per testare i moltiplicatori VHDL e Verilog in modo indipendente, come se fossero due slave collegati al BUS, utilizzando dall'altra parte di esso il software cross compilato con routine le quali forniscono ai moltiplicatori ingressi uguali o diversi utilizzano il modulo I/O per verificare che il risultato da essi ottenuto sia corretto.

- $Application$: contiene i sorgenti del software cross-compilato. Qui sono stati modificati due file, $routines.c$, in cui sono state aggiunte due routine, una per il moltiplicatore Verilog ed una per il moltiplicatore VHDL. Quest'ultime accettano come parametri due operandi uint_32_t. Il modulo Verilog è stato connesso alla periferica 3 e il modulo VHDL alla periferica 4. L'altro file modificato è $main.c$, che opera come testbench utilizzando il modulo I/O per leggere e scrivere dati. Il main chiama le due routine dei moltiplicatori passandogli gli stessi dati come input. Se entrambi ritornano lo stesso risultato vuol dire che entrambi i moduli operano correttamente.
- $Platform$: contiene il progetto di Vivado usato per la simulazione. Qui è stato creato un wrapper APB per ciascun moltiplicatore, uno in VHDL ed uno in Verilog. Il wrapper ha due funzioni: la prima è quella d'interagire con il BUS, mentre la seconda è quella di operare come top level per i moltiplicatori. L'applicazione e la piattaforma non sono sincronizzate tra loro, per cui per far sì che modo che il modulo riceva gli operandi nel modo corretto è stata implementata una EFSM, rappresentata in figura 1.
  - ST_0 : Stato di inizializzazione e reset.
  - ST_1 : Memorizza il primo operando in una variabile temporanea.
  - ST_2 : Attende l'arrivo del secondo operando.
  - ST_3 : Invia al modulo i due operandi e rimane in attesa del risultato.
  - ST_4 : Setta $prdata$ con i risultati ottenuti.

  Come si può vedere nel diagramma di sequenza, l'applicazione e la piattaforma si sincronizzano sul valore di $penable$. Viene quindi utilizzata una variabile temporanea per memorizzare il primo dei due operandi che verrà poi passato, insieme al secondo, al modulo nel modo e al momento corretto. Un operazione simile viene poi fatta anche per l'output, in modo tale da acquisirli dal modulo e inviarli all'applicazione nel modo corretto. Per integrare correttamente i moltiplicatori e i wrapper sono state necessarie delle modifiche al top level e alla testbench della piattaforma. In particolare nel primo è stata aggiunta la mappatura ai moduli per la moltiplicazione in virgola mobile, mentre al secondo è stato cambiato il valore che viene passato alla richiesta iomodule.

IV. RISULTATI

A. SystemC TLM

La testbench utilizzato in entrambi i progetti è di tipo automatico con verifica lasciata al programmatore. Il sistema è stato prima verificato singolarmente per ciascuna della 3 versioni implementate, attraverso l'esecuzione di moltiplicazioni singole inserendo a mano i valori e andando così a testare i casi limite. Dopo di che si è passati alla verifica automatica generando i valori in modo casuale e ripetendo più volte l'operazione. La verifica della corretta sincronizzazione è stata fatta controllando l'ordine dei messaggi mostrati durante la simulazione.

Confrontando i tempi di simulazione dei diversi stili di codifica di SystemC TLM e SystemC RTL si nota in modo marcato la differenza: nel primo, più astratto, abbiamo tempi di simulazione più brevi; mentre nel secondo, che ha precisione temporale con maggiori dettagli di implementazione, abbiamo una simulazione più lenta.

Nel sistema preso in esame sono emerse queste differenze, rendendo impossibile non notare i tempi d'implementazione molto più dilatati per la versione RTL che però dal canto suo, offre un sistema più completo nelle gestione delle eccezioni e degli arrotondamenti e, vista l'operazione da svolgere sono

certamente importanti. La versione TLM sarebbe invece più vantaggiosa nel caso in cui l'arrivo sul mercato debba essere il più rapido possibile (TTM) oppure se c'è la necessità di verificare il modello scelto prima di imbarcarsi nella progettazione RTL.

B. Virtual Platform

Per la simulazione della piattaforma il software viene cross-compilato con il cc65, il quale produce un file rom.mem che viene importato in Vivado come sorgente di simulazione. A questo punto, fissato un valore per il primo operando ed uno per il secondo, viene eseguita la simulazione. Andando a monitorare la variazione dei segnali e il risultato dato in output, si può verificare il corretto funzionamento del primo moltiplicatore in Verilog e successivamente, del secondo in VHDL.

Infine, dopo l'esecuzione della simulazione, il software cross-compilato controlla che i risultati siano corretti andando a verificarne l'uguaglianza, stampando poi sulla console TLC il risultato e scrivendo il valore di controllo letto dalla testbench sul modulo IO.

V. Conclusioni

Valutando i risultati ottenuti da questo elaborato, si può dire che SystemC TLM ha punti di forza e punti deboli, che sono stati toccati con mano, e questo ha permesso di aumentare la consapevolezza sulle differenze presenti tra i vari livelli di astrazione.

L'integrazione del modulo sviluppato nel progetto precedente in una Virtual Platform ha fatto emergere quali sono le operazioni necessarie per il riuso, l'adattamento o l'integrazione di piattaforme già esistenti e i vantaggi che queste operazioni portano. Infine è stato possibile terminare il ciclo iniziato con il progetto precedente andando a comprendere come questo vada ad interagire con l'architettura che lo ospita.

Appendice

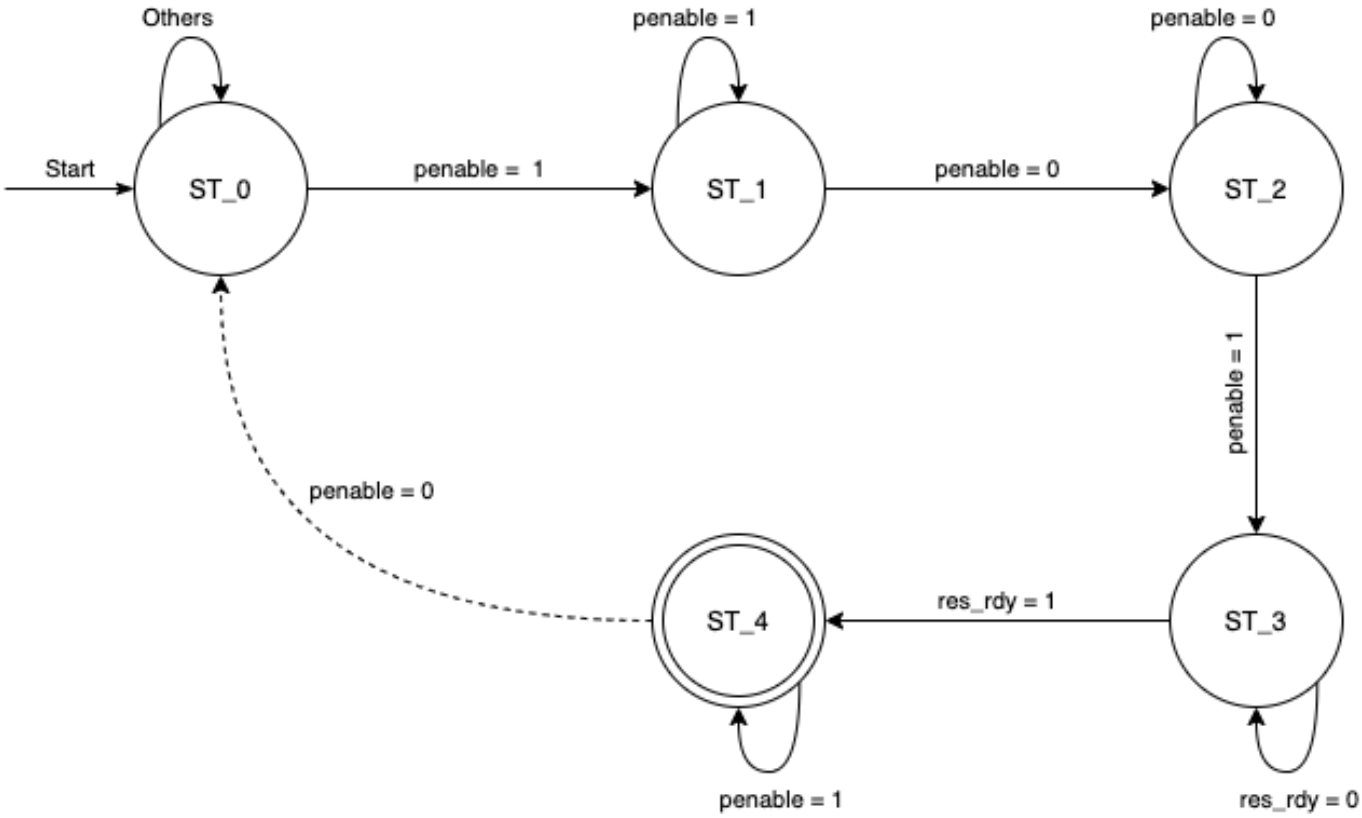

Figura 1. EFSM del wrapper del moltiplicatore.

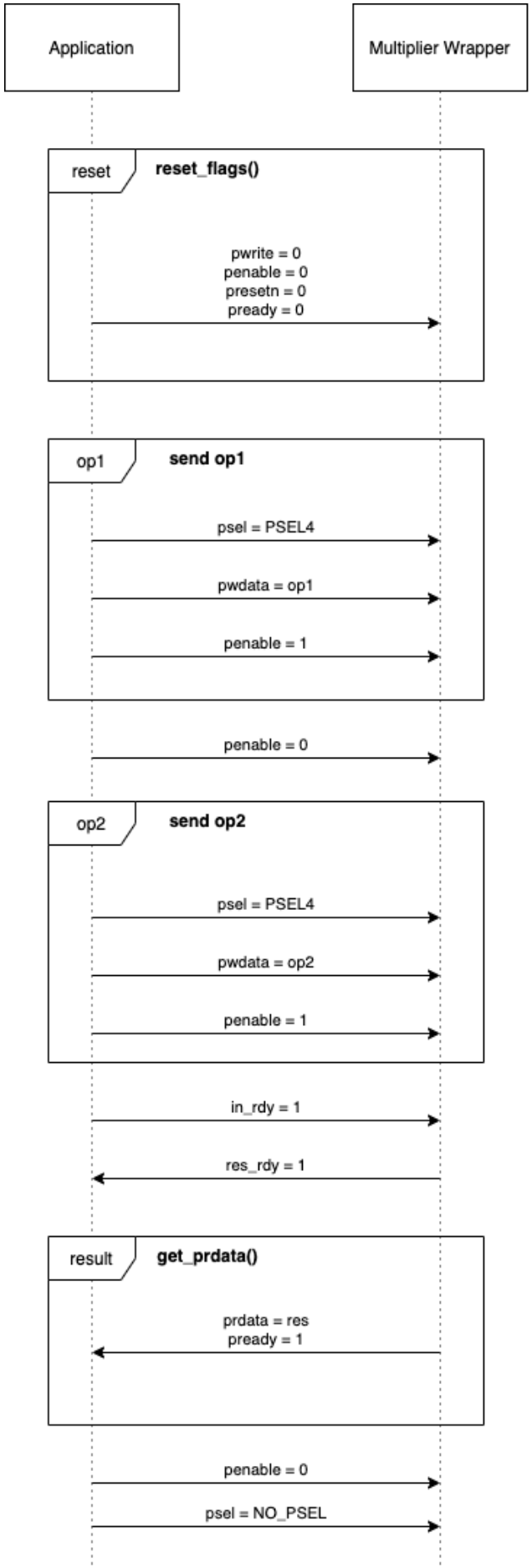

Figura 2. Diagramma di sequenza del protocollo di comunicazione.

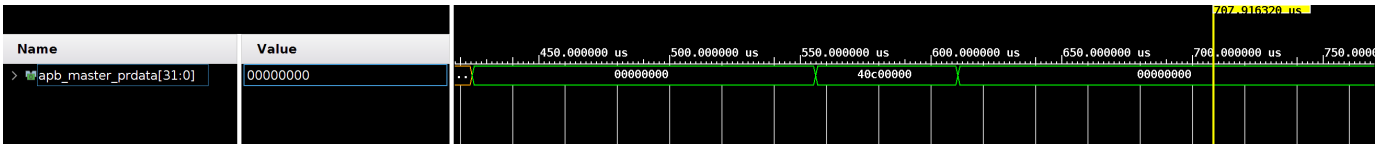

Figura 3. Waveform output della memoria.

Figura 4. Waveform modulo I/O.

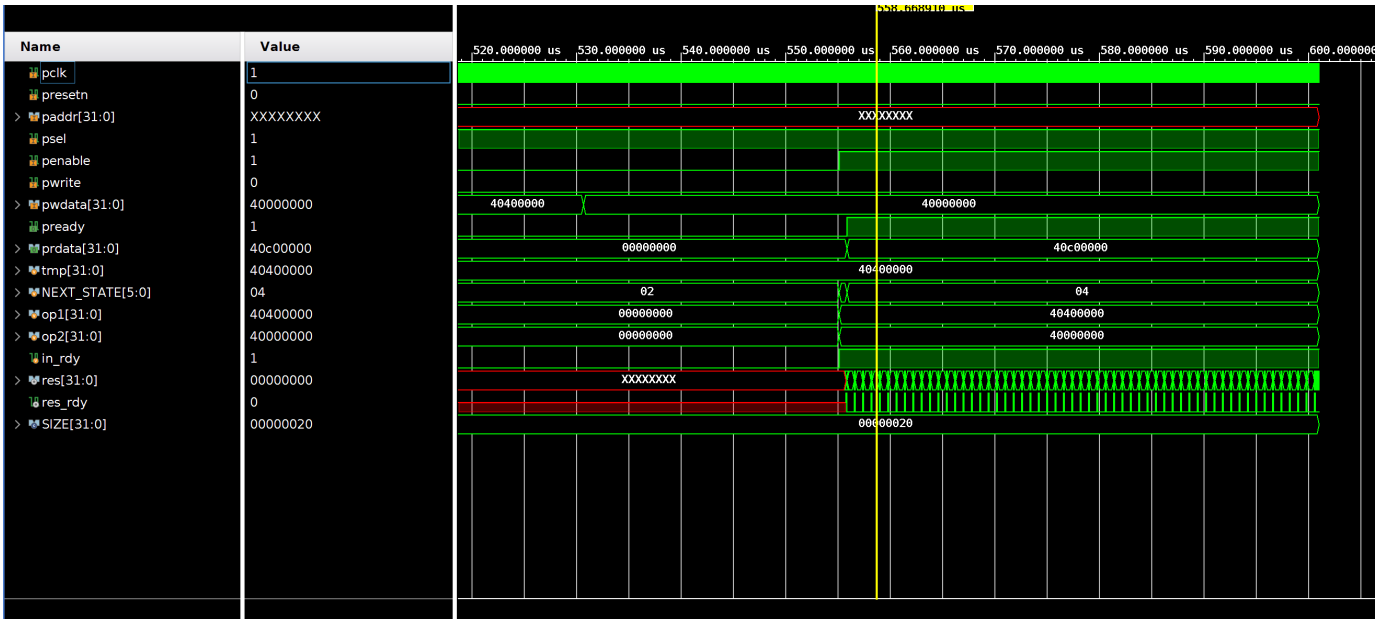

Figura 5. Waveform modulo moltiplicatore.