Modellazione TLM e su Virtual Platform di un Moltiplicatore Floating-point Singola Precisione

Mirco De Marchi - VR445319

***Sommario*—Gli obiettivi del progetto sono modellare un moltiplicatore floating-point singola precisione IEEE754 a livello trasazionale con diversi strumenti, metodologie e protocolli per poi misurarne le prestazioni. Il primo scopo è quello di integrare la virtual platform COM6502-Splatters con un nuovo modulo per la moltiplicazione floating-point RTL e un wrapper che si intefacci come slave al Advanced Peripheral Bus. Come seconda parte si usa SystemC TLM per implementare diverse comunicazioni e stili di sincronizzazioni a livello transazionale tra un initiator e un targer che esegue la moltiplicazione floating-point a livello algoritmico. Gli obiettivi di questo report sono descrivere ed argomentare i risultati ottenuti della simulazione a livello transazionale di un moltiplicatore floating-point.**

I. Introduzione

La modellazione a livello transazionale di un moltiplicatore floating-point singola precisione è stata descritta in 3 linguaggi: Verilog, VHDL e SystemC TLM.

In Verilog e VHDL è stato integrato il moltiplicatore floating-point dentro la piattaforma virtuale COM6502-Splatters, come modulo slave. In particolare è stato collegato al AMBA Advanced Peripheral Bus il moltiplicatore floating-point scritto in Verilog e il moltiplicatore floating-point scritto in VHDL, con i relativi APB wrapper scritti rispettivamente in Verilog e VHDL.

La piattaforma virtuale COM6502-Splatters è composta dai seguenti componenti:

- *CPU MOS 6502 (1975)*: CPU dotata di indirizzamento a 16 bit e dati trasferiti in blocchi da 8 bit;
- *Memoria*: ROM da 16KB composta da un singolo blocco e una RAM da 16KB composta da 8 blocchi per permettere letture e scritture in parallelo;
- *Clock Divider*: gestisce operazioni MMIO tra periferiche e memoria, gestisce scritture multiple sullo stesso blocco di memoria RAM tra MMIO e CPU;
- *BUS ARM APB*: BUS con comunicazione standard AMBA che in questa piattaforma supporta fino a 8 periferiche;
- *Modulo I/O*: richiesta e invio dati al di fuori della piattaforma;
- *Moltiplicatore di interi*: periferica aggiuntiva per effettuare una moltipicazione tra due interi utilizzando il protocollo bus AMBA.

La virtual platform COM6502-Splatters è stata estesa con due ulteriori periferiche: un moltiplicatore floating-point IEEE754 scritto in Verilog e un altro identico scritto in VHDL. Il moltiplicatore ad interi è la perfiferica numero 1, il modulo I/O è la numero 2, il moltiplicatore IEEE754 in verilog il numero 3 e quello in VHDL il numero 4. Ogni moltiplicatore ha il suo rispettivo APB wrapper scritto nel medesimo linguaggio di descrizione dell'hardware. L'APB wrapper dei moltiplicatori floating-point è rappresentato da una EFSM che gestisce la sequenzializzazione di due operandi a 32 bit con il protocollo bus AMBA. Al modulo I/O è stato collegato il testbench della virtual platform, così da poter testare qualche valore in lettura e in scrittura.

Lato software sono state implementate tre routines: una per la periferica 3, una per la 4, ovvero per i due moltiplicatori floating-point, e una per simulare un Floating-point multiplicator HW Accelerator. Il programma software del sistema embedded si interfaccia al bus AMBA come master, mentre tutte le periferiche della piattaforma virtuale sono gli slave. Le tre routines gestiscono la comunicazione dei due operandi e del risultato con i due moltiplicatori floating point, poi per test si verifica che i due risultati delle due moltiplicazioni in verilog e vhdl siano uguali e si va a scrivere 1 sulla periferica di I/O (in questo caso il testbench) se i risultati coincidono, altrimenti si scrive 0. In fine si legge da I/O un valore trasmesso dal testbench e si riprova a fare delle moltiplicazioni.

Per provare questa estensione della virtual platform COM6502-Splatters con il relativo software embedded con le routines di test, bisogna prima di tutto compilare il cross-compilatore *cc65*, fare un `extern` del path della cartella in cui è contenuto tutto il progetto del cross-compilatore *cc65*, spostarsi nella cartella in cui è contenuto l'applicazione e compilare con *make* il software embedded. Verrà generato dentro la cartella bin un file *rom.mem* che rappresenta il file binario dell'applicazione eseguibile dalla COM6502-Splatters. Per eseguirlo, bisognerà caricare su Vivado tutta la piattaforma virtuale COM6502-Splatters con i nuovi moltiplicatori aggiunti e inserire anche come file di simulazione il file *rom.mem*. Una volta lanciata la simulazione si vedrà in console una stringa con il risultato del test di uguaglianza della moltiplicazione floating-point fatta dalla periferica in VHDL e Verilog stampata due volte.

Su SystemC TLM è stata implementata la moltiplicazione floating point algoritmica su diversi livelli di astrazione: UT (untimed), LT (loosely timed) e AT (approximately timed). In aggiunta all'implementazione algortimica della moltiplicazione IEEE754 è stata inserita un'implementazione RTL in SystemC. Di queste 4 implementazioni si è fatto un test dell'efficienza iterando ogni implementazione per 10000000 di volte.

Le tecnologie utilizzate si sono rese utili per poter modellare una piattaforma embedded in modo astratto, usando SystemC TLM, e con diversi livelli di raffinamento, così da capire come si complica la comunicazione tra initiator e target abbassandosi di livello, e in fine in modo tangibile, estendendo una vera e propria piattaforma embedded e provando la comunicazione

su un bus AMBA.

L'attività che ha reso possibile la realizzazione di questo assignment si è svolta nei seguenti passi:

- Studio di SystemC TLM, dei vari livelli di raffinamento e su come implementarli;
- Scrittura in SystemC TLM dei 3 modelli UT, LT e AT per la moltiplicazione floating-point;
- Test del tempo di esecuzione delle 3 implementazioni del moltiplicatore floating-point in SystemC TLM e quella in SystemC a livello RT su 10000000 iterazioni;
- Studio della virtual platform COM6502-Splatters, del protocollo bus AMBA per il controllo del master lato software e per l'implementazione del APB wrapper dei moltiplicatori IEEE754 in Verilog e VHDL;
- Progettazione della EFSM degli APB wrappers dei moltiplicatori floating-point;
- Scrittura modelli degli APB wrappers in Verilog e VHDL secondo la EFSM fatta;
- Collegamento porte tra APB wrappers e moltiplicatori floating-point, e tra APB wrappers e top level della piattaforma;
- Implementazione software embedded per testare la piattoforma con i nuovi moduli;
- Test della piattaforma virtuale;
- Scrittura della relazione;

I risultati prodotti in termini di tempo delle implementazioni del moltiplicatore floating-point su SystemC TLM sui vari livelli di raffinamento si sono dimostrati coerenti con le aspettative. Inoltre SystemC TLM si è dimostrato un sistema molto veloce per provare la comunicazione tra due periferiche, tuttavia anche l'implementazione più raffinata di SystemC TLM approximately timed è ancora a un livello troppo astratto rispetto a una vera e propria implementazione su piattaforma.

II. Background

La modellazione di una piattaforma embedded richiede diverse fasi di progettazione. Prima di tutto bisogna partire dalle specifiche, poi partizionare il sistemi in tanti moduli che poi verranno raffinati in componenti software e componenti hardware.

Per verificare tali funzionalità e descriverle ad alto livello si usa il Transition Level Modeling (TLM). Il Transition Level Modeling è un livello di progettazione di piattaforme embedded e di modelli hardware più astratto rispetto al RTL, ma più specifico rispetto al livello algoritmico.

Il TLM definisce un protocollo di comunicazione tra due o più moduli: initiator e target. L'initiator è il modulo che inizializza una transazione puntata a un target, il target poi gestisce la richiesta, la elabora e fornisce al initiator una risposta. L'initiator comunica con il target sulla forward path e se l'interfaccia è non bloccante il target risponde al initiator sulla backward path. Dal punto di vista di un'architettura client-server, l'initiator è il client e il target è il server. Dal punto di vista di una progettazione di una piattaforma embedded, l'initiator è il testbanch, mentre il target è il top level da interrogare.

Un'implementazione del TLM è SystemC TLM, che permette di rappresentare la comunicazione tra initiator e target con 3 stili, in cui viene definito come il tempo e il dato sono in realzione in base alla percezione del progettista:

- Loosely-Timed (LT): la comunicazione avviene su due punti di sincronizzazione: inizio e fine della transazione. L'interfaccia implementata è bloccante, dunque viene usato solo il forward path, poichè l'initiator aspetta che il target risponda. Può accadere che l'initiator vada in avanti nell'esecuizione al massimo di un quanto di tempo rispetto al tempo di simulazione, definito nel quantum keeper;
- Approximately-Timed (AT): la comunicazione avviene su quattro punti di sincronizzazione: inizio richiesta, fine richiesta, inizio risposta e fine risposta. L'interfaccia implementata è quella non bloccante, dunque viene usato sia il forward che il backward path;
- Untimed (UT): c'è un solo punto di sincronizzazione, senza traccia del tempo;

L'implementazione in SystemC TLM mi permette di fare un'implementazione e un'analisi veloce della comunicazione di due moduli initiator e target, ma che poi devono essere rappresentati a livello RT sulla virtual platform COM6502-Splatters come comunicazione tra master e slave con in specifico il protocollo bus APB AMBA.

Il protocollo Advanced Peripheral Bus AMBA, prevede i seguenti segnali:

- `pclk`: segnale di clock per sincronizzare la comunicazione;
- `presetn`: segnale di reset che quando sale a 1 riporta tutta la piattaforma allo stato iniziale;
- `paddr`: indirizzo a 32 bit;
- `pselX`: bit di selezione di una specifica periferica slave X. Ne esiste uno per ogni slave. Quando viene alzato indica che il modulo slave è selezionato e che un trasferimento di dati è richiesto;
- `pwrite`: bit che se a 1 indica un'operazione di scrittura, altrimenti di lettura;
- `penable`: bit che se a 1 indica che un master ha pronto un dato per lo slave e che le operazioni dello slave possono iniziare;
- `pwdata`: dato trasferito da master a slave a 32 bit;
- `pready`: bit che se a 1 indica che uno slave ha pronto un dato per il master;
- `prdata`: dato trasferito da slave a master a 32 bit;

III. Metodologia Applicata

La COM6502-Splatters è una piattaforma virtuale descritta in VHDL e Verilog dal gruppo di ricerca del laboratorio ESD dell'Università degli Studi di Verona. Il compito da portare a termine è stato quello di estendere questa virtual platform in modo che supportasse la moltiplicazione floating-point. Per risolvere questo requisito sono stati implementati due componenti (figura 2), uno in Verilog, l'altro in VHDL, che andassero ad eseguire ognuno la moltiplicazione floating-point IEEE754, assegnando allo slave 3 la versione in Verilog e allo slave 4 la versione in VHDL.

Per poter mettere in comunicazione i moltiplicatori floating-point con il bus AMBA come moduli slave sono stati scritti

due APB wrapper, rispettivamente in Verilog per il moltiplicatore in Verilog e in VHDL per il moltiplicatore in VHDL. Il problema però è che il bus della COM6502-Splatters permette di trasferire dati di grandezza massima di 32 bit, ciò significa che i due operandi della moltiplicazione dovevano essere inviati su due tempi diversi e in quache modo sincronizzati. Per questo è stato progettata una EFSM per gli APB wrapper (figura 1). I segnali utilizzati del bus AMBA nei APB wrappers sono i seguenti (4):

- `pclk`: segnale di clock;
- `presetn`: segnale di reset;
- `penable`: dato pronto su `pwdata` per lo slave;
- `pwdata`: dato trasferito da master a slave a 32 bit, usato per gli operandi;
- `pready`: dato pronto su `prdata` per il master;
- `prdata`: dato trasferito da slave a master a 32 bit, udato per il risultato;

I segnali `pwrite` e `paddr` non vengono utilzzati. Il segnale `psel` invece non viene utilizzato perchè è già gestito dal master, ovvero quando il bit di `psel` è posto su una specifica periferica slave, che in questo caso è la numero 3 per il modulo Verilog e la numero 4 per il modulo VHDL, tutti gli altri segnali vengono inoltrati su quella periferica.

La EFSM rappresentata nella figura 1, gestisce la serializzazione degli input con uno specifico pattern di comunicazione e la propagazione del risultato sul bus. La EFSM inizia in uno stato iniziale e ci rimane finchè il bit `penable` rimane a 0, in cui inizializza tutti i segnali. Dopodichè quando il segnale `penable` si alza ad 1, allora vuol dire che il primo operando è pronto sulla porta `pwdata` e viene salvato in un registro temporaneo. A questo punto la macchina a stati rimane ferma fino a quando `penable` ritorna a 0, così da ricomiciare allo stesso modo per il trasferimento del secondo operando. Appena dunque `penable` ritorna a 1 il secondo operando e il registro temporaneo vengono inseriti nel moltiplicatore floating-point e viene così avviata l'operazione effettuata dal sottomodulo. Quando il risultato della moltiplicazione è pronto questo viene riportato sul segnale `prdata` e si ritorna allo stato iniziale.

L'implementazione di due moltiplicatori floating-point in Verilog e VHDL su un bus con protocollo di comunicazione APB AMBA sulla piattaforma virtuale COM6502-Splatters permette anche di rappresentare un Floating-point multiplication HW Accelerator, ovvero eseguire la molitplicazione floating-point nei due moduli contemporaneamente. Questo è possibile selezionando il primo moltiplicatore, impostando correttamente lo slave selector, passargli gli operandi secondo il protocollo descritto dalla EFSM 1, poi cambiare periferica con lo slave selector al secondo moltiplicatore e passare gli operandi allo stesso modo, lasciare che le due macchine in parallelo elaborino il risultato della moltiplicazione floating-point riselezionando la prima periferica e aspettando che il risultato sia pronto e in fine aspettare anche il secondo risultato cambiando slave selector.

In fine i due APB wrappers progettati dovranno essere collegati al top level della piattaforma virtuale e al relativo istanza del bus. In questo modo tutti i componenti sono al posto giusto per poter funzionare.

Il software embedded della piattaforma virtuale COM6502-Splatters è scritto in C ed è cross-compilabile con *cc65*. Il risultato della cross-compilazione è un file binario che corrisponde ad un'immagine della memoria ROM della piattaforma virtuale con le istruzioni che la CPU MOS 6502 poi sa interpretare. Questo file eseguibile dalla COM6502-Splatter potrà poi essere caricato su Vivado insieme a tutto il progetto della piattaforma virtuale e l'implementazione dei due moltiplicatori floating-point, così da poterla simulare correttamente.

Il software embedded in sostanza controlla il modulo master della piattaforma virtuale. Il codice implementato per i due moltiplicatori in sostanza segue i passi descritti per la EFSM, in aggiunta però deve essere selezionato il giusto bit di `psel`, così da abilitare la periferica giusta (rappresentato in figura 3). Ad alto livello invece, come software utente, viene scritta un'implementazione di test, che esegue in sequenza due moltiplicazioni floating-point, una usando il modulo Verilog e l'altra usando il modulo VHDL, dopodichè viene verificato se i risultati delle due operazioni di moltiplicazione corrispondono e in tal caso di scrive nel modulo I/O il valore 1, altrimenti si scrive 0. All'esterno della piattaforma viene collegato il testbench con il modulo I/O che è in attesa di un valore di cui ne farà poi la display. Dopodichè il testbench attende che arrivi una richiesta di lettura, che se attivata, risponde con il valore $0x3F800000$, cioè $1,0$ in floating-point. Questo valore è usato poi dal software per eseguire una doppia moltiplicazione floating-point in parallelo con i moduli Verilog e VHDL come avviene in un Floating-point multiplication HW Accelerator.

In SystemC TLM invece è stato descritto a livello più astratto come può avvenire la moltiplicazione floating-point attraverso un mezzo di comunicazione. L'idea è che l'initiator rappresentasse il master della piattaforma virtuale e che il target, il processo che esegue la moltiplicazione floating-point, fosse lo slave. Il processo initiator dovrà preparare dei dati, impacchettarli in un oggetto transazione, richiedere al target di fare la moltiplicazione floating-point e poi gestire la risposta. Il processo target dovrà eseguire il calcolo dei dati inglobati nella transazione ricevuta e spedire indietro la risposta.

Questa comunicazione avviene con diversi livelli di raffinamento che si ottengono con gli stili untimed (UT), loosely-timed (LT) e approximately-timed (AT4). Con lo stile untimed, avviene esattamente ciò che viene descritto sopra: l'initiator invia i dati al target con una transazione e il target ritorna il risultato. Non c'è alcun tipo di sincronizzazione di processi o di annotazione tempolare.

Con lo stile loosely-timed invece il processo initiator può continuare la propria esecuzione di al massimo un quanto di tempo rispetto al tempo di simulazione. In questo caso dunque viene implementata l'interfaccia bloccante di SystemC TLM e viene tenuta traccia in una variabile globale del `local_time`, gestita dal *quantum keeper* che ad ogni quanto di tempo viene riportata a 0. Il `local_time` viene passato come riferimento al target insieme all'oggetto transazione e il target incrementa il `local_time` di un valore di tempo che corrisponde al tempo che il progettista presuppone che il target impieghi a compiere l'operazione di moltiplicazione floating-point.

In questo caso è stato valutato che un singolo moltiplicatore

floating-point impiegasse $6ns$ a compiere tale operazione. Questo valore è stato scelto dal report temporale ottenuto dall'high level synthesis della moltiplicazione algoritmica tra due float effettuata per il primo assignment. Il valore esatto del report sul clock è di $5,702ns$, che approssimato poi è stato messo a $6ns$.

Con lo stile approximately-timed viene implementata l'interfaccia non bloccante. L'initiator implementa la backward path, ovvero la callback che il target poi dovrà chiamare per notificare al initiator che il risultato è pronto, mentre il target implementa la forward path, ovvero la callback che l'initiator deve chiamare per richiedere che il target esegua la moltiplicazione floating-point. I punti di sincronizzazione sono 4 (per questo la sigla AT4): l'inizio della richiesta, la fine della richiesta, l'inizio della risposta e la fine della risposta. L'inititor prepara la transazione con i dati incapsulati, invia la richiesta attraverso la forward path e sveglia il processo target. Mentre l'initiator attende una notifica di risposta, il target esegue l'operazione di moltiplicazione e termina la richiesta con una risposta sulla backward path della stessa transazione. A questo punto il processo initiator viene svegliato ed elabora il risultato nella risposta.

Ogni implementazione di SystemC TLM e quella SystemC RTL è testata con un testbench che itera per 10^7 volte con degli operandi random. Questo ha permesso di fare delle analisi delle tempistiche in relazione al livello di raffinamento con cui rappresento la comunicazione. Si prevede che più la rappresentazione è raffinata, più l'accuratezza in termini di tempi di comunicazione è precisa, ma più la simulazione sarà lenta.

Il codice SystemC della moltiplicazione floating-point è suddiviso in cartella a seconda dello stile rappresentato: UT, LT, AT4 e RTL. Per ogni cartella c'è un *CMakeList.txt* per la compilazione del codice, inoltre nella cartella padre c'è un *CMakeList.txt*, che, dopo essere configurato dentro una cartella *build/* si può dare il comando `make time` che eseguirà in sequanza tutti gli stili implementati in SystemC e ne darà un'analisi dei tempi di esecuzione.

IV. Risultati

I risultati ottenuti delle performance in termini di tempo dell'esecuzione in SystemC TLM della comunicazione tra initiator e target con gli stili di modellazione transazionali rispettivamente untimed (UT), loosely-timed (LT), approximately-timed (AT4) e in fine con la rappresentazione in SystemC a livello RT (RTL) interate 10^7 volte, sono i seguenti:

	UT	LT	AT4	RTL
Real time	$1,85s$	$1,80s$	$10,27s$	$587,17s$
User time	$1,09s$	$1,60$	$9,86s$	$545,96s$
System time	$0,00s$	$0,00s$	$0,01s$	$3,32s$

I risultati ottenuti sono quelli aspettati, ovvero, da come si può osservare dalla tabella, più è basso il livello di astrazione della moltiplicazione floating-point in SystemC più il tempo di simulazione è lungo, ma nello tempo otteniamo una rappresentazione più accurata del modello embedeed che vogliamo rappresentare.

Come si può vedere dalle onde dei segnali dei moltiplicatori floating-point in Verilog e VHDL nelle figure 5 e 6 , viene eseguita correttamente la moltiplicazione di $0x3F000000$ e $0x40000000$, corrispondente all'operazione in floating-point di $2.0 * 0.5$, con risultato $0x3F800000$, che in floating-point è 1.0. Dove in questo caso la moltiplicazione eseguita dai due moduli è effettuata in sequenza.

Invece nella figura 7 si vedono le onde dei segnali dell'interazione con il modulo di I/O, attraverso il quale si interagisce con il testbench. Come viene mostrato nelle onde dei segnali, prima di tutto viene comunicato al testbench se il risultato dell'operazione di moltiplicazione floating-point fatta precedentemente dai due moduli Verilog e VHDL è corretto e in tal caso viene inviato 1. Dopodichè il testbench, su richiesta del master, invia il valore $0x3F800000$, ovvero il valore floating-point 1.0, che verrà poi utizzato come operando per le successive moltiplicazioni.

L'operazioni di moltiplicazione successive sono di simulazione di un Floating-point multiplication HW Accelerator, ovvero di un modulo che serializza 4 operandi in ingresso, li inoltra a due a due al moltiplicatore floating-point in Verilog e a quello in VHDL, li esegue in parallelo e serializza poi i risultati. Nella figura 8 si possono vedere le onde generate per passare gli operandi al primo e poi al secondo moltiplicatore. Poi i due moltiplicatori floating-point vengono lasciati ad elaborare il risultato in parallelo, dopodichè, si serializza il risultato come rappresentato con le onde dei segnali della figura 9.

Un'implementazione alternativa del Floating-point multiplication HW Accelerator poteva essere essere fatta totalmente HW, ovvero con un top level che gestisse la serializzazione degli operandi in input e dei risultati in output dei due moltiplicatori floating-point e che poi venisse collegato al bus come singola periferica slave che si interfacciasse al bus con un APB wrapper. Effettivamente in questo modo avremmo risparmiato un posto per una periferica nel bus, tuttavia visto che la piattaforma già supporta un bus con un protocollo specifico e molto efficiente, sarebbe stato inutile implementare la serializzazione degli operandi con un top level fatto ad hoc. Inoltre l'APB wrapper che ne avrebbe gestito la comunicazione dei 4 operandi (2 per moltiplicatore) sarebbe risultato molto più complicato rispetto alla soluzione qui implementata. Dunque senza dubbio è molto più efficiente e semplice utilizzare il bus APB AMBA per serializzare gli operandi su due moduli di moltiplicazione floating-point separati ed implementare così lato embedded software tutte le operazioni di serializzazione e di attesa del risultato.

SystemC TLM è un'estensione di SystemC utile a capire il grado di fattibilità dei requisiti. Inoltre permette di rappresentare la soluzione a diversi livelli di raffinamento così da capire meglio, di livello in livello, su quali punti della comunicazione si complica e fare un'analisi approssimata delle tempistice. Quando invece c'è da rappresentare il sistema per approcciarsi poi alla sintesi bisogna per forza passare a una descrizione della piattaforma embedded con VHDL e Verilog, dove qui

bisogna proprio implementare e descrivere il protocollo di comunicazione segnale per segnale.

V. Conclusione

Come risultato di questo assignment posso dirmi soddifatto dei risultati ottenuti nella simulazione della virtual platform e dell'implementazione della comunicazione fatta in SystemC TLM. Le previsioni fatte sull'analisi delle performance tra UT, LT, AT4 e RTL in SystemC si sono rivelate esatte, ovvero, come previsto, raffinando la soluzione, i tempi di simulazione si allungano. Inoltre le simulazioni delle routines scritte in C, cross-compilate sulla COM6502-Splatters sono risultate coerenti rispetto a quanto scritto lato software.

SystemC TLM si rivela molto utile quando si vuole modellare velocemente una soluzione, così da fare qualche analisi delle performance, valutarne poi la fattibilità e capire se le funzionalità corrispondono alle specifiche date. Inoltre SystemC TLM permette di valutare la comunicazione di due moduli su più livelli di raffinamento, fattore molto importante quando si vuole valutare la progettazione vera e propria del modello con un linguaggio di descrizione dell'hardware. Tuttavia pur dando la possibilità di raffinare un modello, SystemC TLM si mantiene a un livello tanto più astratto rispetto a una soluzione reale su piattaforma. Infatti SystemC permette di rappresentare l'architettura del sistema embedded, ma non permette di rappresentare in modo concreto il protocollo che ne gestisce la comunicazione. Per questo è stato utile passare a una rappresentazione in Verilog e VHDL di una piattaforma embedded, come la COM6502-Splatters, così da vedere un protocollo di comunicazione come quello per il bus APB AMBA.

In fine è stato fondamentale eseguire tutti i passaggi di cross-compilazione di un software embedded e il caricamento del file binario sulla piattaforma virtuale, così da capire meglio tutto il giro che c'è dietro lo sviluppo di una piattaforma embedded.

Appendice

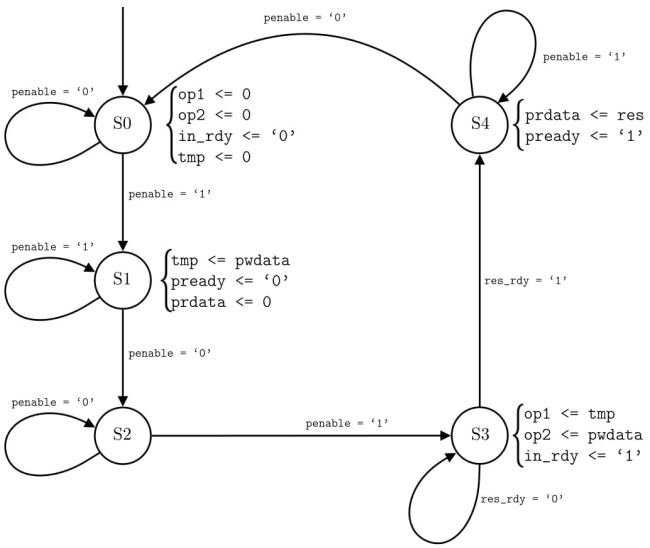

Figura 1. EFSM del Advaced Peripheral Bus wrapper per il moltiplicatore IEEE754 floating-point

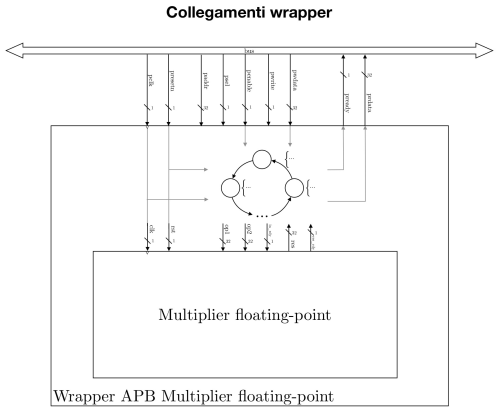

Figura 2. Schema generale della virtual platform COM6502-Splatters

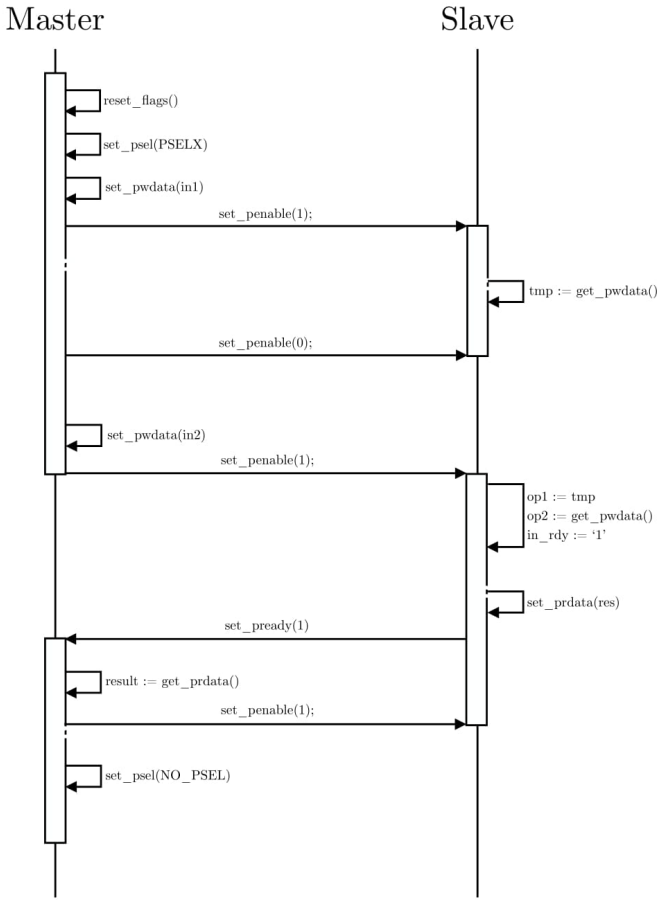

Figura 3. Sequence Diagram della comunicazione master e slave nella virtual platform COM6502-Splatters per gestire la comunicazione con uno dei moltiplicatori IEEE754 floating-point

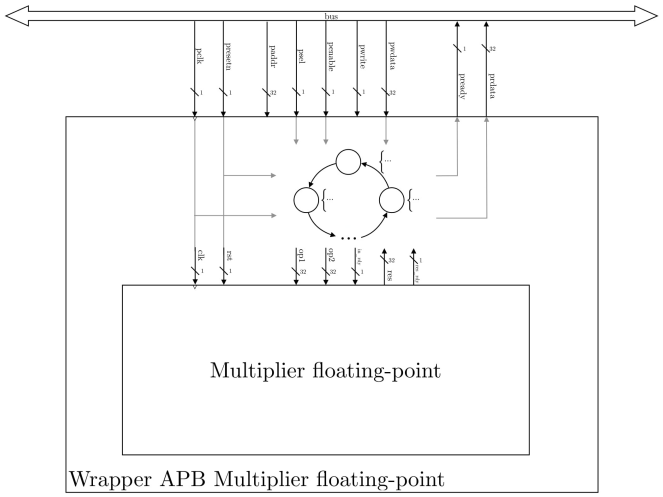

Figura 4. Collegamenti tra Advanced Peripheral Bus Wrapper e un moltiplicatore IEEE754 floating-point incluso in esso

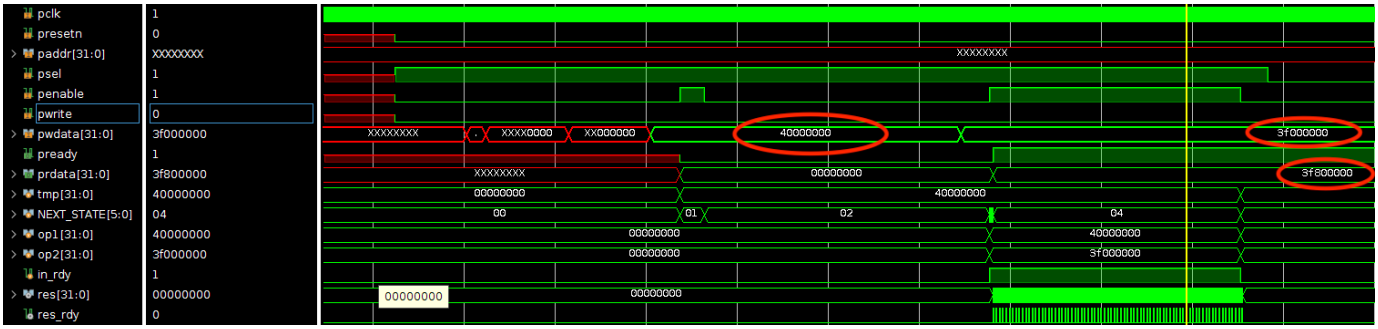

Figura 5. Grafico delle onde dei segnali del Moltiplicatore floating-point in Verilog

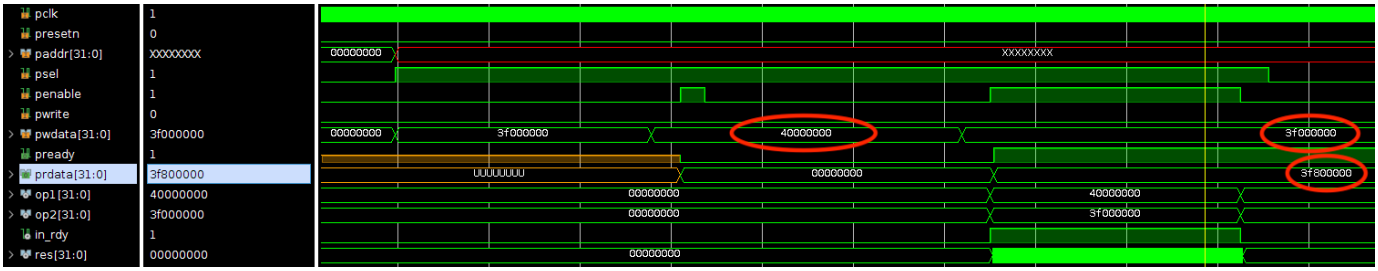

Figura 6. Grafico delle onde dei segnali del Moltiplicatore floating-point in VHDL

Figura 7. Grafico delle onde dei segnali della comunicazione con il modulo I/O per verificare se il risultato della moltiplicazione dei due precedenti moduli corrisponde e per passare alla piattaforma uno dei parametri della moltiplicazione simulando il comportamento di un Floating-point Multiplier HW Accelerator

Figura 8. Grafico delle onde dei segnali della comunicazione con i due moltiplicatori floating-point Verilog e VHDL simulando il comportamento di un Floating-point Multiplier HW Accelerator. Parte della serializzazione dei parametri.

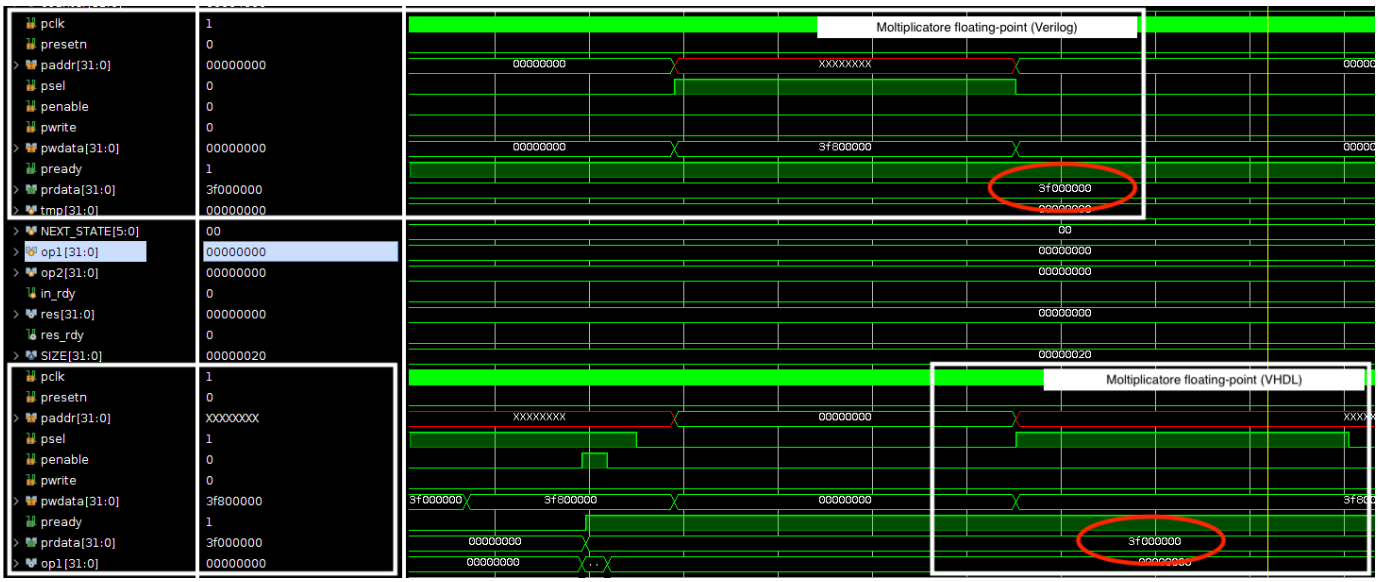

Figura 9. Grafico delle onde dei segnali della comunicazione con i due moltiplicatori floating-point Verilog e VHDL simulando il comportamento di un Floating-point Multiplier HW Accelerator. Parte della serializzazione dei risultati.