1

Integrazione su Virtual Platform e modellazione con SystemC-TLM di un Moltiplicatore Floating-point Single Precision

Enrico Sgarbanti - VR446095

Sommario—Questo documento mostra l'integrazione di un modulo, che realizza due moltiplicazioni a virgola mobile singola precisione secondo lo standard IEEE754[1], nella virtual platform COM6502-SPlatters e la modellazione in SystemC[2] TLM.

I. Introduzione

II. BACKGROUND

Nel classico flusso di progettazione di un sistema ciberfisico si parte a sviluppare software solo dopo aver finito la progettazione hardware. Manca però una visione concretamente utilizzabile all'interno del sistema prima della fase di tapeout, e ciò porta spesso a dover modificare il codice e quindi ad avere diversi rallentamenti. La progettazione basata su piattaforma è la creazione di un'architettura stabile basata su microprocessore che può essere rapidamente estesa, personalizzata per diverse applicazioni e consegnata ai clienti per una rapida implementazione. (J.M. Chateau-STMicroelectornics). Essa permette di fare verifica funzionale, stime di tempo per analisi di perfomance, partizionale hardware e software, porta ad un incremento della velocità e permette modularità e riuso. La modellazione a livello di transazione (TLM) è un tipo di progettazione che sta tra il livello algoritmico e quello RT. I dettagli di implementazione vengono astratti preservando però gli aspetti comportamentali del sistema, permettendo quindi una simulazione di quella RTL e di avere una piattaforma dove si può iniziare velocemente a sviluppare software. Permette quindi di iniziare a sviluppare software molto prima rispetto al classico flusso di sviluppo.

In SystemC-TLM la comunicazione tra componenti si ottiene dallo scambio di pacchetti tra un modulo initiator e un modulo target attraverso una Socket: L'initiator fa partire una transizione (cioè un trasferimento di dati da un modulo ad un altro) verso il target attraverso il forward-path, poi il target elaborerà i dati e se l'interfaccia non è bloccante allora risponderà attraverso il backward-path.

Ci sono poi tre principali sistemi che definiscono la relazione tra tempo e dati e permettono al progettista di descrivere il sistema con livello più o meno astratto:

• Approximately timed: le transazioni sono divise in quattro fasi: inizio richiesta, fine richiesta, inizio risposta, fine risposta. L'interfaccia è non bloccante quindi viene usato sia il forward-path che il backward-path. Esso è indicato per l'esplorazione architetturale e l'analisi delle performance.

- Loosely timed: le transazioni sono divise in due fasi: inizio transizione, fine transizione. L'interfaccia è bloccante quindi viene usato solo il forward-path poichè l'initiator aspetta la risposta del target. Esso rappresenta i dettagli di temporizzazione sufficienti per avviare un sistema operativo ed eseguire sistemi multi-core.
- Untimed: la nozione di tempo non è necessaria e quindi non viene presa in considerazione.

La virtual platform utilizzata è COM6502-Splatters che include:

- **CPU** MOS 6502 (1975) con indirizzamento a 16 bit e gestione di dati in 8 bit.
- ROM da 16KB in un singolo blocco.
- RAM da 16KB divisa in 8 blocchi per permettere operazioni multiple di lettura/scrittura.
- BUS ARM APB (advanced peripherical Bus) che supporta fino a 8 periferiche.
- IO Module usato per richiedere e inviare dati alla piattaforma.
- Multiplier usato per eseguire moltiplicazioni fra interi.

Essa deve essere compilata col crosscompilatore cc65[3] (checkout commit: 582aa41f2a702ff477a00a5d69a794390a13b544) AMBA

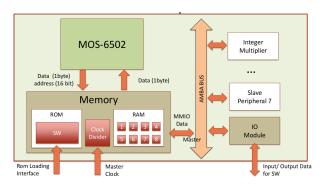


Figura 1: COM6502-Splatters

(Advanced Microcontroller Bus Architecture) è uno standard open-source di ARM per la connessione e la gestione di blocchi funzionali nei progetti di system-on-a-chip. In APB (Advanced peripherical bus) ci sono due attori: **Master** che controlla le perifiriche; **Slave** periferica controllata dal master. I segnali utilizzati in questo protocollo sono:

- pclk: segnale di clock della periferica.
- preset: segnale di reset della periferica.

- **paddr:** indirizzo.
- **psel:** segnale che indica se la periferica è stata selezionata.
- penable: segnale che indica se la periferica è stata abilitata.
- **pwrite:** segnale che indica operazioni di scrittura (1) o lettura (0) sulla periferica.
- pwdata: dati sulla periferica da parte del Master.
- **pready:** segnale che indica che i dati per il Master sono pronti.
- prdata: dati sulla periferica per il Master.

III. METODOLOGIA APPLICATA

A. Struttura progetto

Virtual_Platform/

- application/ cartella contenente il codice sorgente dell'applicazione.
- platform/ cartella contenente il codice sorgente di Splatters, del modulo double_multiplier e il testbench.

TLM/

- UT/ progetto con modellazione TLM Untimed.
- LT/ progetto con modellazione TLM Loosely Timed.
- AT4/ progetto con modellazione Approximately Timed.
- RTL/ progetto con modellazione a livello RT. Questa versione è funzionalmente equivalente a quella dell'altro report, ma col testbench adattato per essere coerente con quello usato per le modellazioni TLM.
- script.sh piccolo script per eseguire in automatico in tutte le cartelle i comandi make, make clean e l'esecuzione con time.
- Ogni progetto presenta la seguente struttura:
 - * Makefile: tool per la compilazione automatica del progetto. Richiede che la variabile d'ambiente SYSTEMC_HOME contenga il path alla libreria di SystemC.
 - * include: contiene gli headers del progetto.
 - * src: contiene i file sorgenti del progetto.
 - * **bin:** contiene l'eseguibile generato dopo la compilazione.
 - * **obj:** contiene i file oggetto generati dopo la compilazione.

B. Virtual Platform

1) Procedimento: Per prendere dimestichezza con la piattaforma è stato prima integrato il modulo di moltiplicazione IEEE754 scritto in verilog sulla periferica 3. Per fare ciò è stato creato un wrapper in hardware con l'interfaccia APB slave per poterlo fare comunicare con il resto della piattaforma e un driver per poterlo utilizzare a livello software.

Poi è stato integrato il modulo d'interesse cioè double_multiplier sulla periferica 4.

Entrambi i codici sono statai testati eseguendo due semplici moltiplicazioni dove un operando è stato letto da input

- 2) Wrapper double_multiplier: I segnali del bus APB sono stati collegati nel seguente modo al double_multiplier:
 - pclk: collegato a clk.
 - preset: collegato a reset.
 - paddr: non utilizzato.
 - psel: non utilizzato.
 - penable: utilizzato nella EFSM.
 - pwrite: non utilizzato.
 - pwdata: utilizzato nella EFSM per prelevare gli operandi.
 - **pready:** utilizzato nella EFSM per indicare che su **prdata** è presente un risultato.
 - prdata: utilizzato nella EFSM per inviare il risultato al master.

Sono stati inoltre usati i seguenti segnali intermedi:

- op1, op2: collegati alle porte op1 e op2 del double_multiplier e utilizzati per inviare gli operandi.
- res: collegato alla porta res del double_multiplier e utilizzato per ricevere il risultato delle moltiplicazioni.
- op1_tmp, op2_tmp, op3_tmp, op4_tmp: utilizzati per memorizzare i valori degli operandi letti dal bus e poi inviarli a op1 e op2.
- res_tmp: utilizzato per memorizzare i valore del secondo risultato da res e inviarlo al momento giusto sul bus.
- **ready, done:** utilizzati per il protocollo di handshake col double_multiplier
- STATE, NEXT_STATE: utilizzati per rappresentare lo stato presente e lo stato prossimo della FSMD.

Avendo scelto di leggere gli operandi (e scrivere i risultati) su cicli di clock consecutivi si è stati costretti ad utilizzare molti registri per memorizzare i valori temporanei. Si può migliorare questo aspetto utilizzando *ready* e *done* diversi per le due moltiplicazioni all'interno di double_multiplier

Il wrapper è descritto grazie alla EFSM [Figura 2] la quale è formata da 14 stati:

- **ST_WAIT1:** stato di partenza. Qui vengono resettati i segnali interni e gli output a zero. In caso di segnale *preset* a 1 si torna in questo stato. In caso di segnale *penable* a 1, il master avrà pubblicato il valore del primo input in *pwdata* e quindi si passa a *ST_READ1*.
- **ST_READ1:** qui si salva il valore di *pwdata* in *op1_tmp*. In caso di segnale *penable* a 0 si passa a *ST_WAIT*2.
- **ST_WAIT1:** qui si attende che venga inviato l'operando successivo. In caso di segnale *penable* a 1, il master avrà pubblicato il valore del secondo input in *pwdata* e quindi si passa a *ST_READ2*.
- ST_READ2: qui si salva il valore di *pwdata* in *op2_tmp*. In caso di segnale *penable* a 0 si passa a *ST_WAIT3*.
- **ST_WAIT3:** qui si attende che venga inviato l'operando successivo. In caso di segnale *penable* a 1, il master avrà pubblicato il valore del terzo input in *pwdata* e quindi si passa a *ST READ3*.
- ST_READ3: qui si salva il valore di *pwdata* in *op3_tmp*. In caso di segnale *penable* a 0 si passa a *ST_WAIT4*.
- **ST_WAIT4** qui si attende che venga inviato l'operando successivo. In caso di segnale *penable* a 1, il master avrà pubblicato il valore del quarto input in *pwdata* e quindi si passa a *ST_READ4*.

- **ST_READ4:** qui si salva il valore di *pwdata* in *op4_tmp*. Ora sono stati raccolti tutti gli operandi per *double_multiplier* quindi si passa direttamente a *ST_ELAB1*.
- ST_ELAB1: qui si passano i primi due operandi a double_multiplier e poi si passa a ST_ELAB2.
- **ST_ELAB2:** qui si passano gli altri due operandi a *double_multiplier* e si rimane in attena che *done* diventi 1 per poi passare a *ST_RETO*.
- **ST_RET0:** qui si inserisce su *prdata* il valore di *res* e si pone *pready1* a 1, per indicare al Master che è pronto il primo risultato. Poi si passa a *ST_RET1*.
- **ST_RET1:** qui si salva in *res_tmp* il risultato della seconda moltiplicazione ottenuto da *double_multiplier* e si resta in attesa che il master abbia letto il valore del primo risultato. Quando *penable* diventa 0 allora il Master avrà letto il primo risultato e si passa in *ST_WAIT5*.
- **ST_RET0:** qui si pone *pready* a 0 e si aspetta che il Master richieda il secondo risultato. Quando *penable* diventa 1 allora si passa in *ST_RET2*.
- **ST_RET1:** qui si inserisce su *prdata* il valore di *res* e si resta in attesa che il master abbia letto il valore del secondo risultato. Quando *penable* diventa 0 allora il Master avrà letto il primo risultato e si passa in *ST_WAIT1*.
- 3) Driver double_multiplier: Per utilizzare il double_multiplier è stata aggiunta una routine all'interno del file /application/src/routines.c chiamata double multiplier. La comunicazione tra master e slave è descritta dal sequence diagram in figura 3. Sostanzialmente il master invia uno alla volta gli operandi di 32 bit e poi resta in attesa che pready diventi 1. Lo slave nel frattempo salva gli operandi in registri, dopodichè li invia nel giusto ordine a double_multiplier e attende che done diventi 1. A questo punto invia al master il primo risultato e imposta *pready* a 1 e poi si salva il secondo in un registro. Il master si salva il valore del primo risultato e poi pone *penable* a 0 per dire allo slave che ha ricevuto il dato, il quale di conseguenza imposta pready a 0. Dopodichè il master imposta penable a 1 per dire allo slave che è pronto a ricevere il secondo risultato e si mette in attesa che pready diventi 1. Lo slave analogamente a prima inviarà il risultato e porrà pready a 1 sbloccando il master che si salverà il risultato e metterà pready a 0 permettendo così allo slave di ritornare allo stato iniziale.

C. SystemC TLM

IV. RISULTATI

1) Simulazione e testbench sulla VirtualPlatform: Il main del software legge un valore dal modulo I/O e chiama il driver di double_multiplier per eseguire la moltiplicazione con l'operando letto e altri 3 scelti arbitrariamente. Una volta ottenuto i risultati vengono poi trasmessi per essere letti in simulazione del testbench. (Nel main è anche presente la possibilità di utilizzare gli stessi operandi per eseguire due moltiplicazioni separate col driver float_multiplier).

Nel testbench scritto in verilog viene caricato il codice del software nella ROM, inviato un valore sul bus, che verrà poi utilizzato come operando e infine stampati i due risultati ottenuti. Nelle figure 45 è possibile guardare la simulazione.

2) Risultati TLM:

V. CONCLUSIONI

RIFERIMENTI BIBLIOGRAFICI

- I. C. Society, "Ieee standard 754 for binary floating-point arithmetic," Online, 1985.
- [2] Accellera Systems Initiative et al., "Systemc," Online, December, 2013.
- [3] "Cc65," https://github.com/cc65/cc65.

APPENDICE

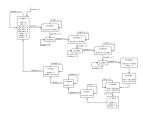


Figura 2: EFSM del wrapper di double_multiplier

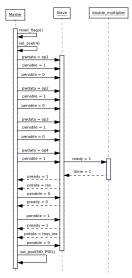


Figura 3: Sequence diagram della comunicazione tra Master Slave e double_multiplier



Figura 4: Simulazione virtual platform

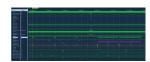


Figura 5: Simulazione con zoom virtual platform