Moltiplicatore IEEE 754 virgola mobile a singola precisione: integrazione all'interno della COM6502-Splatters Platform e modellazione TLM

Luigi Capogrosso - VR445456

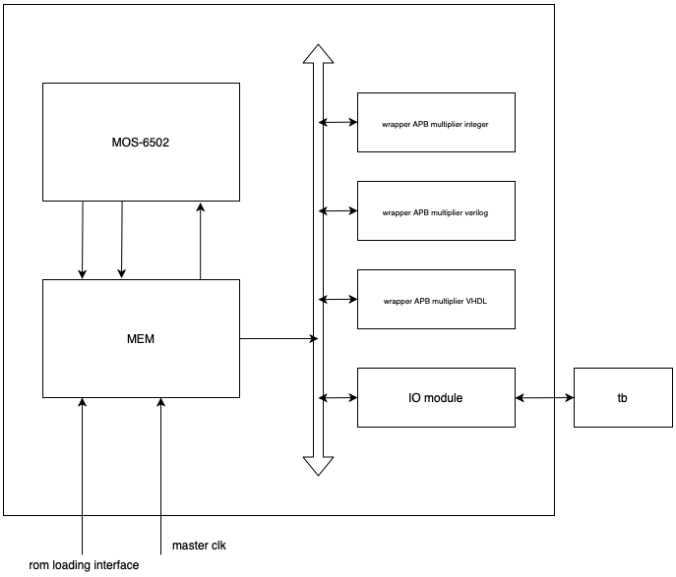

Figura 1. Organizzazione del sistema.

***Sommario*—All'interno del seguente report, in primo luogo, è presentato l'argomento: *virtual platform - modeling and simulation*. Questo, è trattato, attraverso l'integrazione di due moltiplicatori, implementati nei linguaggi Verilog e VHDL secondo lo standard IEEE for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) a 32 bit, all'intreno della piattaforma virtuale COM6502-Splatters.**

In secondo luogo, poi, è presentata la *modellazione TLM* dello standard IEEE for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) a 32 bit attaverso lo sviluppo di diversi moduli SystemC a differenti livelli di astrazione.

I. Introduzione

Nella prima parte del progetto ci si è concentrati sullo sviluppo basato su piattaforme. Nello specifico, in realtà, non si è fatto utilizzo di una piattafoma fisica, bensì, si è fatto uso di una *piattaforma virtuale*. In particolare, l'integrazione all'interno della `COM6502-Splatters` dei componenti `multiplier754_vhd` e `multiplier754_v` (Figura 1) è organizzata nei seguenti moduli:

- **CPU MOS 6502** (1975)
  - Indirizzamento a 16 bit (16KB ROM, 16KB RAM)
  - Larghezza dati: 8 bit
- **Memoria**
  - ROM in un unico banco
  - RAM suddivisa in 8 diversi blocchi per consentire operazioni di lettura/scrittura multiple
- **BUS ARM APB**
  - Supporta fino a 8 periferiche
- **Moltiplicatore di numeri interi**
- **Moltiplicatore IEEE-754 Verilog**
  - Si occupa di eseguire la moltiplicazione di due numeri secondo lo standard IEEE for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) a 32 bit, implementato facendo uso del linguaggio Verilog
- **Moltiplicatore IEEE-754 VHDL**
  - Si occupa di eseguire la moltiplicazione di due numeri secondo lo standard IEEE for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) a 32 bit, implementato facendo uso del linguaggio VHDL
- **Modulo IO**
  - Utilizzato per richiedere o inviare dati dalla piattaforma

Nello specifico, il moltiplicatore per numeri interi a 16 bit è la periferica numero 1, il modulo di IO è la periferica numero 2, il moltiplicatore IEEE-754 sviluppato per mezzo del linguaggio Verilog è la periferica numero 3, ed, infine, il moltiplicatore IEEE-754 sviluppato per mezzo del linguaggio VHDL è la periferica numero 4.

I componenti `multiplier754_vhd` e `multiplier754_v` sono collegati alla piattaforma virtuale come moduli *slave*. Per connettere i moltiplicatori all'AMBA Advanced Peripheral Bus è stato necessario implementare i rispettivi APB wrapper. Da componente *master*, invece, funge l'ESW sviluppato per la piattaforma. La comunicazione tra il modulo master ed il modulo slave avviene secondo il *Sequence Diagram* in Figura 2. Lato SW, quindi, è stato necessario implementare due routines per la gestione della comunicazione: una per la periferica numero 3 ed una per la periferica numero 4.

Il secondo scopo del progetto è stato quello di sviluppare per mezzo della libreria TLM del linguaggio SystemC la moltiplicazione floating-point in modo algoritmico a diversi livelli di astrazione, nello specifico: UT, LT ed AT4. Inoltre, è stata aggiunta anche una implementazione a livello RT. Poi, di seguito, sono stati effettuati poi test di efficienza provando ogni implementazione per 10000000 di casi.

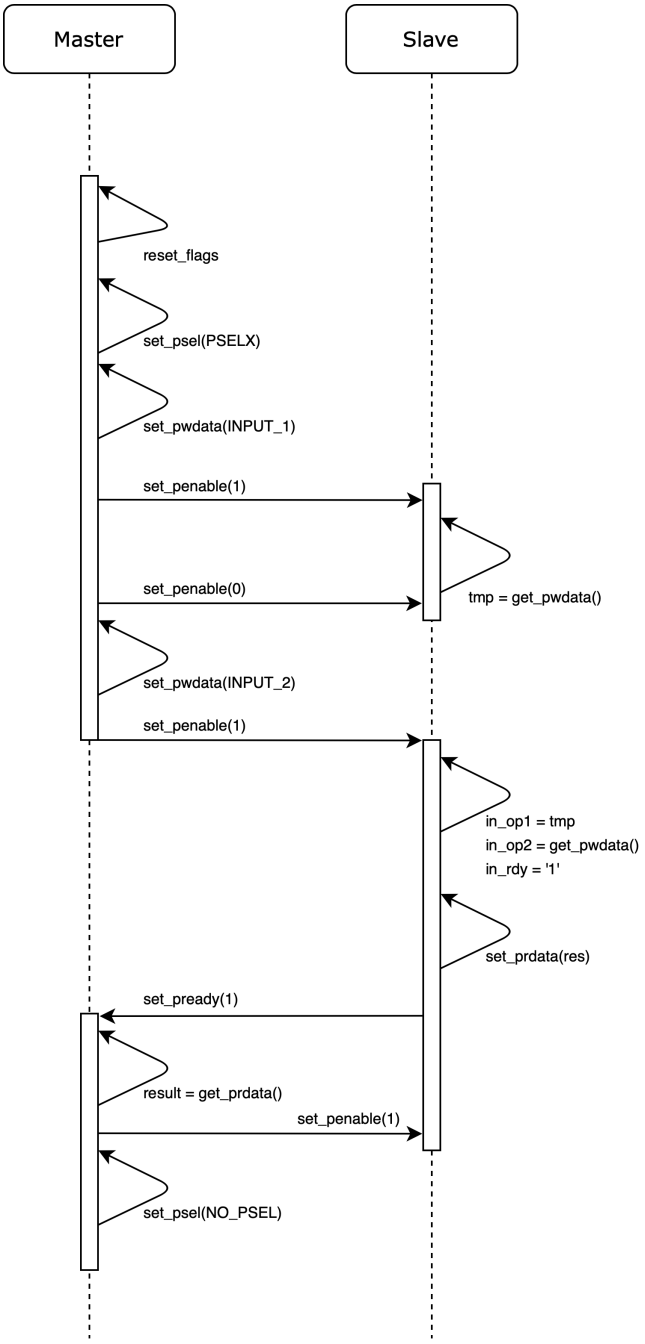

Figura 2. Comunicazione tra il modulo master e il modulo slave.

II. Background

Una **piattaforma virtuale** è un sistema basato su software che rispecchia tutte le funzionalità di un sistema basato su chip. Queste piattaforme combinano simulatori di processore e modelli di alto livello per fornire una rappresentazione astratta ed eseguibile dell'hardware. La virtual platform utilizzata è la `COM6502-Splatters`, piattaforma descritta attraverso i linguaggi VHDL e Verilog.

Di particolare interesse nella piattaforma `COM6502-Splatters` troviamo la tipologia di bus secondo il protocollo Advanced Peripheral Bus AMBA, che, prevede i seguenti segnali:

- `psel`: segnale di clock
- `presetn`: segnale di reset. Quando questo vale 1, la piattaforma è riportata allo stato iniziale
- `paddr`: indirizzo a 32 bit
- `penable`: Quando questo vale 1, indica che è disponibile per il modulo slave un dato di `pwdata`
- `psel N`: bit di selezione per la specifica periferica N. Quando questo vale 1 indica che un trasferimento dati dello specifico modulo N è richiesto
- `pwrite`: bit per la specifica di una operazione di scrittura (quando vale 1), o di scrittura (quando vale 0)
- `pwdata`: per il trasferimento dati da master a slave (32 bit)
- `pready`: quando questo vale 1, indica che uno slave ha il dato pronto per il master
- `prdata`: per il trasferimento dati da slave a master (32 bit)

Una implementazione a **livello transazionale**, invece, può essere implementata facendo uso della libreria TLM del linguaggio SystemC. La progettazione di questo linguaggio si basa su moduli (*initiatior* e *target*) che comunicano fra loro per mezzo di socket. In particolare, la comunicazione fra questi può essere descritta a diversi livelli di astrazione attraverso i quali si definisce come il tempo e il dato sono in realzione fra loro. Abbiamo quindi:

- **SystemC TLM AT4** (acronimo di *Aproximately Timed 4 phases*) è la versione basata su chiamate *non* bloccanti. La sincronizzazione avviene secondo il protocollo handshake a 4 fasi, dove, in partiolare: il modulo initiator invia i dati al modulo target mettendosi, successivamente, in attesa dell'acknowledgement (fasi `BEGIN_REQ`, `END_REQ`). Il target una volta ricevuta la transazione elabora i dati e ne salva il risultato, per poi, successivamente, ritornare l'acknowledgement alla controparte sbloccandone l'esecuzione (fasi `BEGIN_RESP`, `END_RESP`).
- **SystemC TLM LT** (acronimo di *Loosely Time*) è la versione basata su una sincronizzazione per mezzo di chiamate bloccanti. Qui è l'initiator che chiama la funzione `b_transport` implementata nel target, aggiungendo alla transazione anche l'informazione relativa al tempo. Questa viene utilizzata per sfruttare la tecnica del *temporal decoupling*, ovvero, per permettere ai processi (nel nostro caso il testbench) di continuare la propria esecuzione senza però incrementare il tempo di simulazione. La sincronizzazione avviene poi in modo esplicito in punti definiti, oppure, allo scadere del *quanto* di tempo che gli è stato associato dal kernel di simulazione, ottenendo così una simulazione estremamente rapida.
- **TLM UT** è una versione simile alla precedente in cui, però, non è specificata l'informazione relativa al tempo. Non richiede, dunque, di una sincronizzazione esplicita, poiché, questa viene garantita dalla chiamata bloccante, permettendo una classica sincronizzazione dei processi.

III. Metodologia applicata

Per quanto concerne l'integrazione dei componenti `multiplier754_vhd` e `multiplier754_v` all'interno

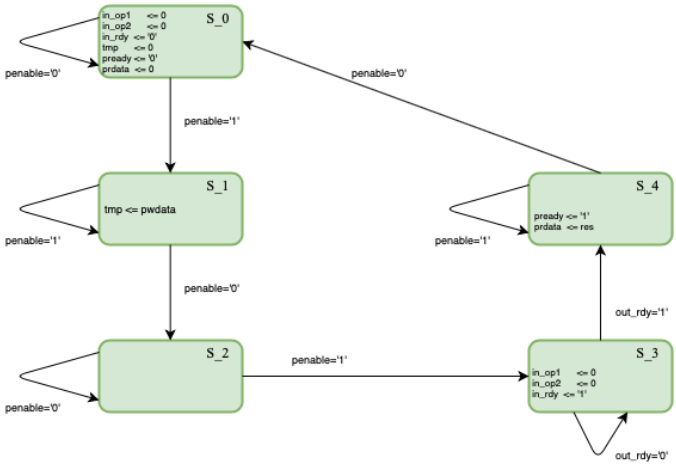

Figura 3. EFSM per gli APB wrapper.

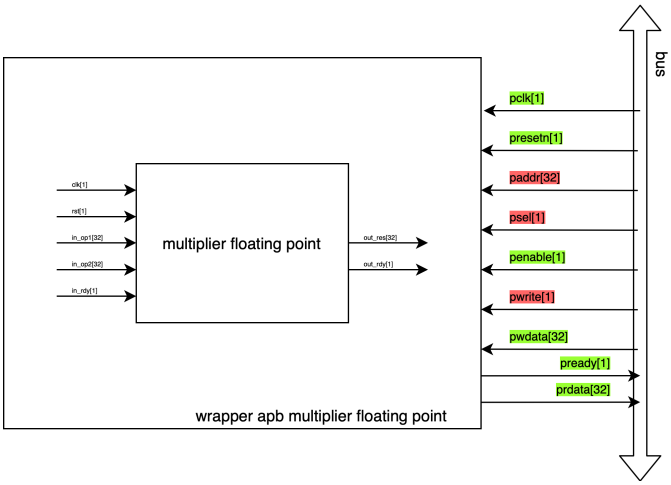

Figura 4. Segnali utilizzati del bus AMBA.

della virtual platform vediamo che è stato necessario tenere conto di alcuni vincoli progettuali.

Il primo riguarda il bus della `COM6502-Splatters` che permette di trasferire dati di grandezza massima di 32 bit. Ciò implica, quindi, che i due operandi della moltiplicazione devono essere inviati in tempi differenti e, poi, essere fra loro sincronizzati. Per fare ciò è stata progettata la EFSM per gli APB wrapper mostrata in Figura 3.

La EFSM rappresentata è appunto concepita per la gestione della serializzazione degli input e, la propagazione del risultato sul bus. Lo stato iniziale della EFSM è `S_1`, stato all'interno del quale sono inizializzati tutti i segnali. Si è fermi in questo finché il bit `penable` rimane a 0. Quando il segnale `penable` diventa 1 il primo operando è pronto sulla porta `pwdata` e viene salvato all'interno di un registro temporaneo. A questo punto la EFSM rimane "ferma" fino a quando `penable` non ritorna a 0, così da rieffettuare con il medesimo procedimento il trasferimento del secondo operando. Appena `penable` torna nuovamente ad 1 il secondo operando e il registro temporaneo vengono inseriti nel moltiplicatore, viene quindi avviata l'operazione effettuata dal sottomodulo. Per concludere, quando il risultato della moltiplicazione è pronto questo è riportato sul segnale `prdata` e, infine, si ritorna allo stato iniziale.

Per quanto riguarda, invece, l'utilizzo dei segnali del bus AMBA nei diversi APB wrappers, vediamo che questi sono i seguenti (Figura 4):

- `pclk`
- `presetn`
- `penable`
- `pwdata`
- `pready`
- `prdata`

Nello specifco notiamo quindi che i segnali `pwrite` e `paddr` non sono utilzzati. Il segnale `psel`, invece, non è utilizzato perché già gestito dal master. In particolare, quando il bit di `psel` è posto su una specifica periferica slave, tutti gli altri segnali sono inoltrati su quella periferica.

Nota progettuale: l'implementazione dell'accelleratore hardware per la moltiplicazione di numeri in virgola mobile a singola precisione, a dispetto dell'implementazione svolta nel *first assignment*, non è stata gestita per mezzo di un modulo HW. Ho deciso di applicare tale scelta progettuale, poiché, mi sembrava essere la più conforme a quello che è lo "spirito" di effettuare una progettazione basata su piattaforme, quindi, lo svolgere una rapida implementazione per raggiungere più velocemente il time-to-market. Tale descrizione è quindi gestita a livello di SW dato che risulta essere di più semplice e, soprattutto, più rapida implementazione: difatti, è possibile inserire i due operandi, attendere che l'esecuzione parallela dei due moltiplicatori sia terminata e, infine, ricevere i due risultati serializzati.

La fase di progettazione a livello transazionale, invece, segue diverse fasi prima del suo completamento nella versione finale.

A. AT4 implementation

Lo stile di codifica AT usa interfacce non bloccanti. Questo stile permette di avere più dati e informazioni per perfezionare il progetto. L'interfaccia di tipo non bloccante significa che, il modulo dopo aver chiamato la primitiva, invece di bloccarsi, continuerà. Sulla base di questo concetto abbiamo quindi che, il modulo `multiplier754_AT4_tb` (che è l'initiator) usa la primitiva `nb_transport_fw` per chiamare il target, e, poi chiama la primitiva `wait(event)` per "lasciare" continua esecuzione e, notificare i dati per mezzo della `nb_transport_bw`. Dall'altro lato, il target, quando la `nb_transport_fw` viene triggerata, chiamerà l'`ioprocess()`, sbloccandolo con la primitiva `notify(t_event)`, che, dopo il processo di calcolo, richiamerà l'initiator con la primitiva `nb_transport_bw`. L'AT4 è un protocollo di handshaking a 4 fasi tra l'initiator e il target. Questo significa che supporterà una sequenza più dettagliata di interazioni tra i due moduli, ma, richiede un tempo di simulazione più lento.

B. LT implementation

Lo stile di codifica LT fa utilizzo di interfacce bloccanti a due punti di sincronizzazione (invocazione e ritorno)

e, la tecnica del temporally decupled. Questo stile rende le simulazioni più veloci rispetto alla simulazione RTL. Il modulo `multiplier754_LT_tb` (che è l'initiator) usa la primitiva `b_transport` con la nozione di tempo per la sincronizzazione: quando la `b_transport` del modulo target (`multiplier754_LT`) viene triggerata, crea una variabile `SC_ZERO_TIME` prima del calcolo e, in fase di lettura, restituisce questa all'initiator. In questo momento, l'initiator è in attesa della risposta del target.

C. UT implementation

Limplementazione UT presenta la totale assenza della nozione di tempo. Il tipo di interfaccia utilizzata è quella bloccante, e, ci sarà un punto di sincronizzazione predefinito. Il modulo `multiplier754_UT_tb`, che, funge da initiator, comunica per mezzo della interfaccia `b_transport` con il modulo `multiplier754_UT`, che è il target. Per il modulo target, quando la primitiva `b_transport`è in modalità scrittura, essa calcolerà la funzionalità; quando la primitiva è in modalità lettura, invece, invierà il risultato del calcolo.

D. RTL implementation

L'implementazione SystemC a livello TL del moltiplicatore IEEE-754 definisce una modellazione a "basso livello", dunque più vicina all'harware. Prevede uno sviluppo dei moduli suddividendo la parte logica (FSM) dalla parte di calcolo (datapath).

IV. Risultati

Per quanto riguarda i risultati della simulazione della virtual platform, questi possono essere osservati in Figura 5, in Figura 6 e in Figura 7. Nello specifico:

- Figura 5: in questa è possibile osservare le onde dei segnali del modulo di IO attraverso il quale si interagisce con il testbench. Come viene mostrato nelle onde, anzitutto è notificato al testbench se il risultato dell'operazione di moltiplicazione svolta dai due moduli Verilog e VHDL è corretta (in tal caso viene inviato 1). Di seguito abbiamo che il testbench, su richiesta del master, invia il valore 0x3F800000 (1.0).
- Figura 6: in questa è possibile osservare le onde dei segnali per la moltiplicazione che avviene per mezzo del moltiplicatore Verilog.
- Figura7: in questa è possibile osservare le onde dei segnali per la moltiplicazione che avviene per mezzo del moltiplicatore VHDL.

I risultati, in termini temporali, ottenuti dall'esecuzione dei diversi moduli in SystemC TLM sono, invece, i seguenti:

	UT	LT	AT4	RTL
Real time	$3,16s$	$5,89s$	$18,72s$	$594,17s$
User time	$1,09s$	$5,18$	$17,65s$	$571,96s$
System time	$0,04s$	$0,06s$	$0,13s$	$4,52s$

Nello specifico, i risultati ottenuti sono conformi alle aspettative, ovvero, dalla tabella si può evincere che, più il livello di astrazione è basso, più il tempo di simulazione è lungo (ma la rapprasentazione è più accurata).

V. Conclusioni

In conclusione, riprendo sulla base di quanto detto nello scorso elaborato:
Il modo migliore di procedere, dunque, per costruire un circuito digitale risulta essere quello percorso in questo documento, ovvero passare da una versione VHDL/Verilog più dettagliata, verificando una iniziale correttezza del componente prodotto con primi test diretti su hardware programmabile, poi, prima di un eventuale impiego su larga scala, sviluppando un progetto ad alto livello in SystemC/C++ verificare l'hardware in modo efficiente.

In realtà, al termine di questo progetto risulta chiaro come il modo migliore di procedere per costruire un circuito digitale risulta essere quello di partire con lo sviluppo per mezzo di linguaggi ad alto livello come SystemC/C++,in modo tale da modellare velocemente una soluzione, analizzarne le performance, valutarne la fattibilità e, capire se la descrizione corrisponde alle specifiche date, per poi, solo dopo eventualmente concentrarsi sullo sviluppo HW.

In conclusione, possiamo inoltre dire che, lo sviluppo per mezzo di piattaforme virtuali presenta i seguenti vantaggi:

- Una VP consente lo sviluppo in anticipo rispetto alla disponibilità di silicio;
- Una VP fornisce una elevata controllabilità: l'HW fisico non può, di solito, essere "fermato". Una VP, essendo un programma SW può essere invece completamente gestito e personalizzato.
- Una VP fornisce piena visibilità: in qualsiasi momento, un utente può ottenere informazioni relative a qualsiasi parte del sistema (processore, bus, periferiche, eccetera…). Così, le informazioni possono essere analizzate e gestite in qualsiasi modo dall'utente finale.

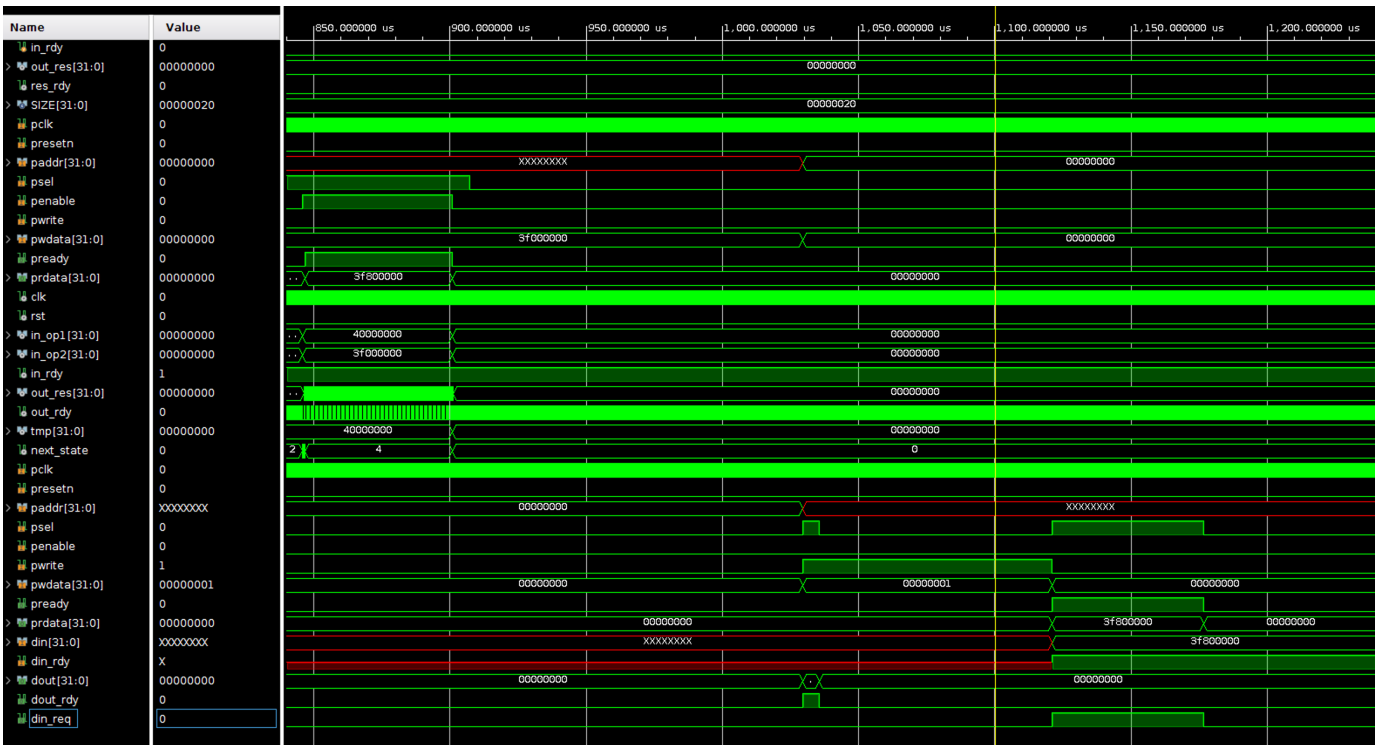

Figura 5. Grafico dei segnali della comunicazione con il modulo IO

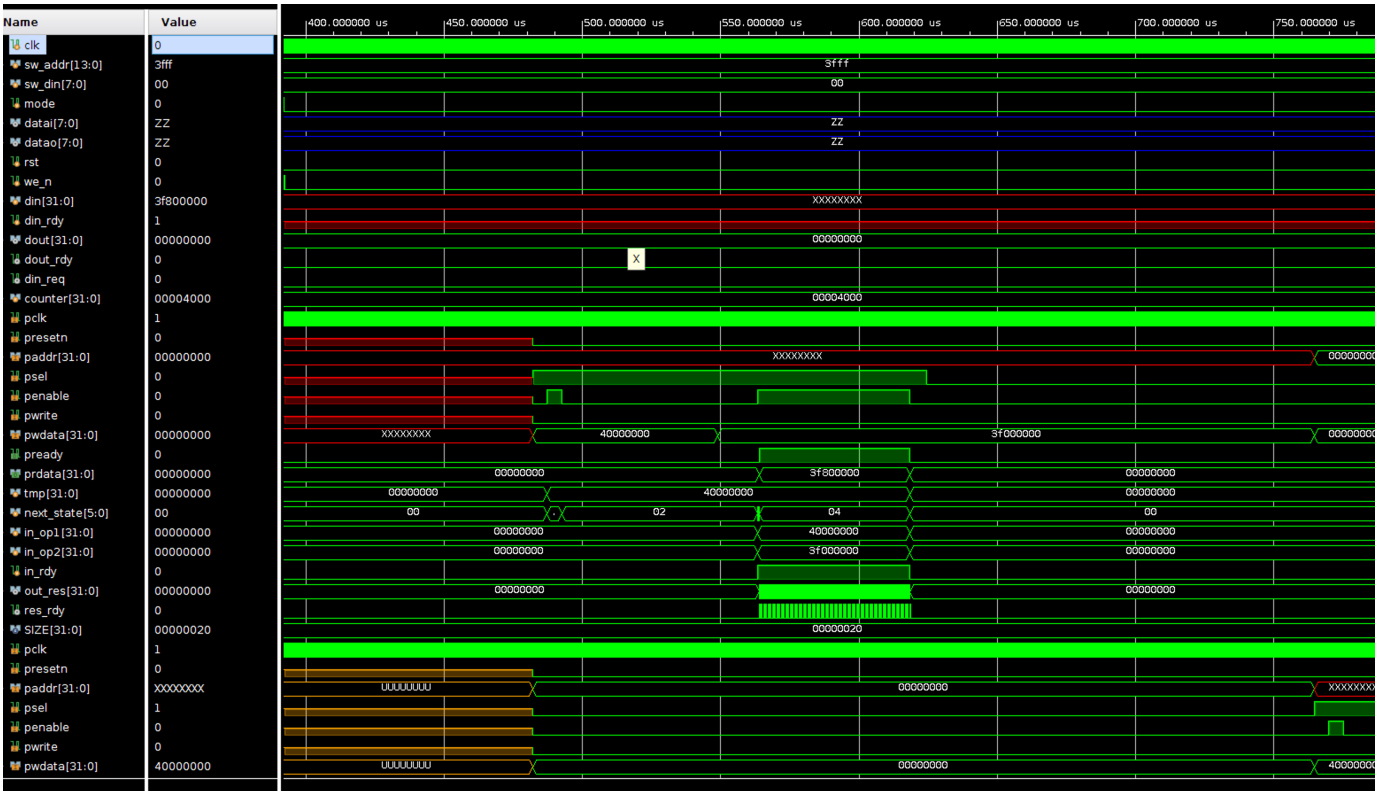

Figura 6. Grafico dei segnali del moltiplicatore Verilog

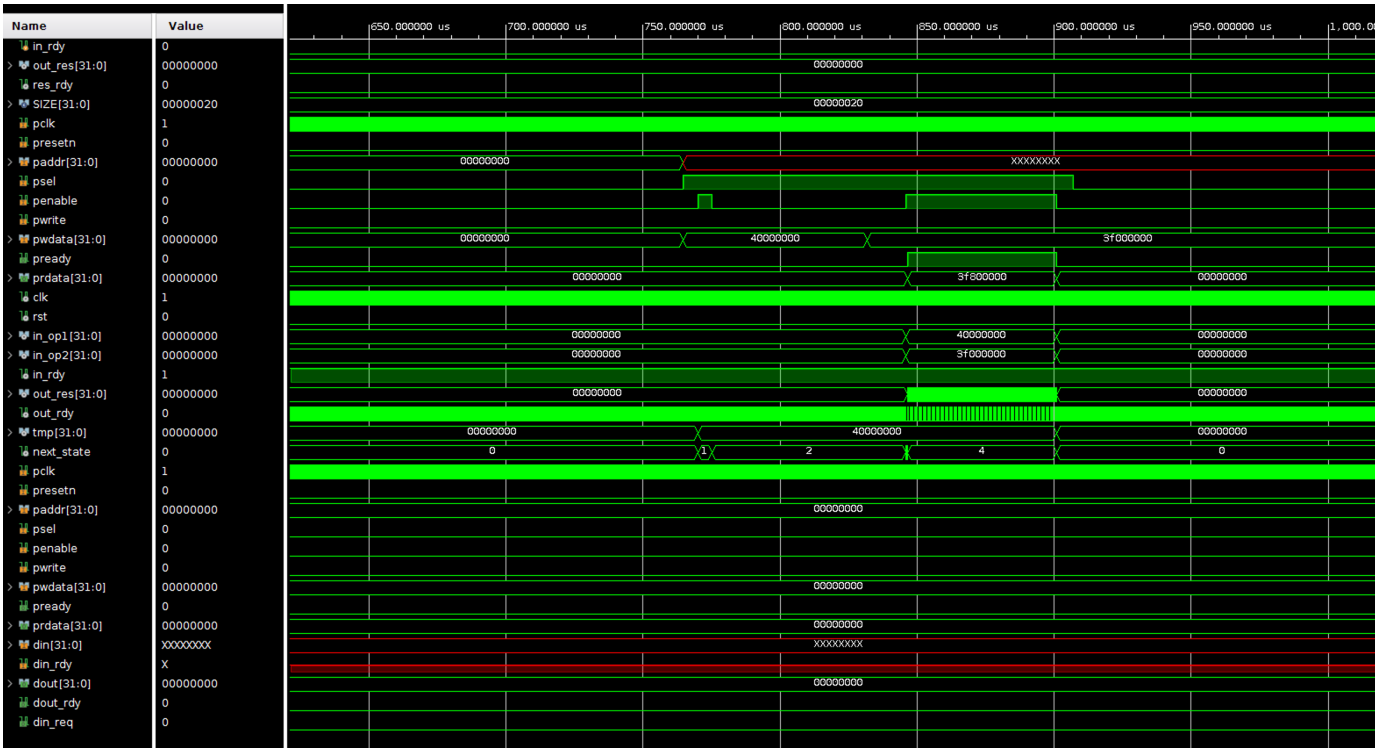

Figura 7. Grafico dei segnali del moltiplicatore VHDL