Modellazione di un Moltiplicatore Floating-point Single Precision in SystemC TLM ed Integrazione in una Virtual Plaform

Elena Ramon

***Sommario*—Questo documento presenta l'implementazione di un sistema per il calcolo della moltiplicazione in virgola mobile nei diversi stili di codifica di SystemC TLM e presenta un'implementazione su Virtual Platform.**

I. Introduzione

I sistemi descritti nel documento rappresentano il calcolo della moltiplicazione in virgola mobile nei diversi stili di codifica di SystemC TLM e su una Virtual Platform.

A. Virtual Platform

L'obiettivo è quello di osservare come un sistema interagisce con l'architettura sottostante.

Si è cercato di integrare il sistema sviluppato nel progetto precedente in una piattaforma virtuale dove sono presenti anche altri componenti.

I moduli principali che sono stati aggiunti nella directory platform sono:

- il wrapper del modulo moltiplicatore in virgola mobile, il quale si occupa di definire:
  - in quale modo vengono recepiti gli input dal bus e quando passarli al modulo;
  - in quale modo vengono recepiti gli output del modulo e quando passarli al bus.
- il moltiplicatore stesso, con alcune modifiche rispetto alla precedente implementazione.

B. SystemC TLM

L'obiettivo è quello di osservare le caratteristiche delle diverse implementazioni anche a confronto tra loro.

SystemC TLM viene solitamente utilizzato per una progettazione veloce con poca attenzione ai dettagli di implementazione, quindi più astratta. Ogni stile di codifica è composto da un testbench, che funge da iniziatore della transazione e da un moltiplicatore, che rappresenta il target. Per svilupparli è stato preso come riferimento l'esempio presentato in laboratorio, adattandolo al caso specifico in esame.

I risultati finali, che verranno in seguito analizzati, mostrano come, nonostante la medesima rappresentazione astratta, gli stili di codifica abbiano dei tempi di simulazione differenti, dovuti alle diverse tipologie di relazione tra dati e tempo (la sincronizzazione tra target ed iniziatore). Per eseguire un confronto con il sistema sviluppato a livello RTL nel progetto precedente, quest'ultimo è stato modificato in modo tale che venisse eseguita una sola moltiplicazione.

II. Background

A. Virutal Platform

Le Virtual Platform sono una rappresentazione astratta dell'architettura su cui un sistema embedded dovrà essere eseguito. Utilizzare una Virtual Platform, per la progettazione del moltiplicatore in virgola mobile, ha permesso di avere pieno controllo dell'hardware su cui il sistema embedded veniva eseguito, consentendo la risoluzione più rapida di errori attraverso la riproduzione consecutiva dei medesimi passi in fase di test. Un'altra caratteristica fondamentale, che ha permesso di anticipare la fase di sviluppo del sistema in esame, è la facilità di estensione e modifica di Virtual Platform pre-esistenti, in questo caso è stata utilizzata la piattaforma COM6502-Splatters [1].

B. SystemC TLM

SystemC TLM (Transaction Level Modeling) [2] permette di modellare le funzionalità di un sistema ad un livello più astratto, quindi con meno dettagli di implementazione, rispetto all'RTL (Register Transfer Level) [3], dando più importanza alla transazione dei dati (l'interazione tra i moduli). Il sistema è stato implementato nei tre diversi stili di codifica previsti dallo standard:

Untimed dove il concetto di temporizzazione di una transazione non è considerato;

Loosely-timed in cui ogni transazione viene temporizzata solo all'inizio e alla fine della transazione;

Approximately-timed in cui ogni transazione viene sincronizzata in più punti. Nella versione base sono quattro: inzio della richiesta, fine della richiesta, inizio della risposta e fine della risposta.

I soggetti che interagiscono in una transazione sono due, l'iniziatore (colui che inizia la transazione) e il target (il punto finale della transazione, colui con cui l'iniziatore vuole interagire). Questi durante la comunicazione si scambiano un payload contenente i dati su cui effettuare operazioni e informazioni di controllo. Lo standard prevede la possibilità di estendere il payload generico per adattarlo ai diversi protocolli che possono essere utilizzati, come avviene nel sistema in esame.

III. Metodologia applicata

A. Virtual Platform

Come detto nella sezione precedente il moltiplicatore in virgola mobile è stato integrato su una Virtual Platform già esistente, per cui sono state aggiunte funzioni e collegamenti.

1) Application: Sono stati modificati i seguenti file:

- main.c: è stato introdotta la moltiplicazione in virgola mobile, in particolare vengono dichiarate quattro variabili di tipo uint32_t:
  - *VHDL_in_1* e *VHDL_in_2*: gli operandi per il moltiplicatore in VHDL;
  - *verilog_in_1* e *verilog_in_2*: gli operandi per il moltiplicatore in verilog;

  il valore di tre di esse (*VHDL_in_1*, *verilog_in_1* e *VHDL_in_2*) viene definito nel main.c, mentre il valore della quarta (*verilog_in_2*) deriva dall'iomodule. Vi sono poi le due variabili, sempre di tipo uint32_t, per la memorizzazione del risultato.
  Il main esegue una sola chiamata a *floating point multiplication* la funzione definita in routines.c, a cui passa i quattro input e i puntatori ai due output. Una volta ottenuti i risultati chiama la funzione *io_write()*.
- routines.c (con il corrispondente routines.h): viene definita la funzione "floating_point_multiplication". La dimensione massima dei dati trasportabili sul bus è di 32 bit e la lunghezza degli operandi della moltiplicazione in virgola mobile è di 32 bit. Per tale motivo si è reso necessario fare più invii consecutivi alla piattaforma. Gli input vengono inviati in modo tale da corrispondere al medesimo ordine del progetto implementato nella precedente consegna (quindi *VHDL_in_1*, *verilog_in_1*, *VHDL_in_2*, *verilog_in_2*). Dopo aver inviato gli input la funzione attende che i risultati siano pronti, quindi li assegna ai puntatori *VHDL_result* e *verilog_result*.

2) Platform: Sono stati modificati i seguenti file:

- top_level.v: è stata aggiunta la mappatura al modulo per la moltiplicazione in virgola mobile;
- tb.v: è stato cambiato il valore che viene passato alla richiesta iomodule.

Sono stati aggiunti i seguenti file:

- floating_point_multiplier_apb_wrapper.vhd: rappresenta l'interfaccia che permette al modulo che esegue la moltiplicazione in virgola mobile di interagire con il bus a cui è collegato sulla porta numero 3. L'applicazione e la piattaforma non sono sincronizzate tra loro, per cui per fare in modo che il modulo riceva gli operatori nel modo corretto è stata implementata una EFSM (Extended Finite State Machine), rappresentata in figura 1. L'applicazione e la piattaforma si sincronizzano sul valore di *penable*, come mostrato nel diagramma di sequenza 2, e vengono utilizzate tre variabili temporanee, tutte di tipo unsigned su 32 bit, per contenere gli input in fase di acquisizione, in modo tale da passarli al modulo nel modo e al momento corretto, e per contenere gli output, in modo tale da acquisirli dal modulo e inviarli all'applicazione in modo corretto. Il wrapper si occupa anche di eseguire il cast dei valori in ingresso e in uscita dal modulo.
  I segnali che vengono definiti nel wrapper sono:
  - *current_state* e *next_state*: per rappresentare gli stati della EFSM;
  - *result*: unsigned su 32 bit;
  - VHDL_in e verilog_in: unsigned su 32 bit per passare i dati al modulo;
  - *temp*, *temp1*, *temp2*: unsigned 32 bit;
  - *done* e *ready*: bit usati per sincronizzare il wrapper e il top level del modulo.

  Il mapping tra le porte del wrapper e del moltiplicatore è il seguente:
  - *clk* nel moltiplicatore corrisponde al *pclk* del wrapper (rispetto alla versione implementata nel precedente progetto, è stato cambiato il tipo di dato del clock, da bit a std_logic);
  - *ready* nel moltiplicatore corrisponde al *ready* del wrapper;
  - *VHDL_in* nel moltiplicatore corrisponde al *VHDL_in* del wrapper;
  - *verilog_in* nel moltiplicatore corrisponde al *verilog_in* del wrapper;
  - *eccezione_out* nel moltiplicatore corrisponde ad *eccezione_out* del wrapper;
  - *done* nel moltiplicatore corrisponde al *done* del wrapper;
  - *result* nel moltiplicatore corrisponde al *result* del wrapper;
  - *rst* nel moltiplicatore corrisponde al *presetn* del wrapper (come per il clock è stato cambiato il tipo di dato rispetto al precedente progetto).
- I file relativi al modulo per le moltiplicazioni in virgola mobile del precedente progetto:
  - top level, definito come floating_point_multiplier.vhd;
  - multiplier vhdl;
  - multiplier verilog.

B. SystemC TLM

Come detto nella sezione precedente il moltiplicatore in virgola mobile è stato implementato nelle tre diverse versioni di TLM: approximately-timed, loosely-timed e untimed. Tutte presentano la stessa struttura:

- un testbench che funge da iniziatore
- un moltiplicatore che rappresenta il target

Nel sistema in esame il payload che viene inviato nelle transazioni è stato esteso con un puntatore alla struttura iostruct, la quale contiene le informazioni sui dati in input (i due operandi) e output (il risultato).

1) Untimed: Le transazioni avvengono senza il passaggio di riferimenti temporali, la sincronizzazione viene garantita dalla chiamata della primitiva *b_transport* con interfaccia bloccante. Il testbench richiama la funzione, agganciando il payload, il target riceve la chiamata, esegue la moltiplicazione tra gli operandi e ritorna il risultato.

	AT4	LT	UT	RTL
Real time	38m21,518s	17m27,345s	15m4,200s	21m20,212s
User time	0m33,111s	0m19,793s	0m14,260s	1m37,117s
System time	9m43,919s	5m9,430s	3m49,961s	19m41,661s

Tabella I: Tempi di esecuzione su 10000000 moltiplicazioni

2) Loosely-timed: La comunicazione tra le componenti avviene attraverso la primitiva *b_transport*, come per "Untime", la particolarità di questo stile di codifica è il Temporal Decoupling, cioè la capacità del testbench di proseguire la sua esecuzione "al di fuori" del tempo di simulazione, quindi senza produrre effetti su di esso, finché non raggiunge un punto di sincronizzazione o non termina il tempo a sua disposizione (ogni processo in SystemC TLM viene eseguito per un quanto di tempo).

3) Approximately-timed 4 fasi: È stata sviluppata la versione base la quale prevede 4 fasi: inizio e fine della richiesta, inizio e fine della risposta. Il testbench chiama la primitiva *nb_transport_fw* (inizio della richiesta) per la quale attende una risposta di ricezione da parte del moltiplicatore (fine della richiesta). Quindi il target calcola la moltiplicazione, chiama la primitiva *nb_transport_bw* (inizio della risposta) in seguito alla quale l'iniziatore continua la sua esecuzione (fine della risposta).

Le tre versioni implementano la stessa funzionalità, la moltiplicazioni di due fattori in virgola mobile, la differenza si trova quindi nel tempo di simulazione impiegato da ciascuna. I tempi di simulazione sono presentati nella tabella I.

Per ottenere dei dati rilevanti in ogni tipologia sono state eseguite 10000000 moltiplicazioni. Il modulo RTL presente nella tabella è una versione composta da un solo moltiplicatore, in modo da poter essere confrontabile con le versioni in SystemC TLM (quindi rispetto al modulo SystemC RTL presentato nel precedente progetto non è presente il top level, in quanto non più necessario, e viene creata una sola istanza del moltiplicatore la quale comunica direttamente con il testbench).

I dati della tabella rendono evidente che maggiore è il controllo del tempo e la sincronizzazione dei processi più il tempo di simulazione è lungo. Nello specifico il dato real time rappresenta il tempo di esecuzione effettivamente impiegato dal calcolatore per terminare l'esecuzione del sistema, il suo valore è quindi influenzato da fattori esterni al sistema stesso (include anche il tempo utilizzato da altri processi). Il dato user time rappresenta il tempo impiegato dalla CPU per l'esecuzione effettiva del sistema. Il dato sys time rappresenta il tempo impiegato per l'esecuzione da parte della CPU di system call per il processo.

IV. Risultati

Il testbench utilizzato in entrambi i progetti è di tipo automatico con verifica lasciata al programmatore. In particolare in Virtual Platform le due moltiplicazioni vengono fatte sugli stessi operandi, mentre in SystemC TLM gli operandi sono tutti diversi.

A. Virtual Platform

Il sistema è stato prima implementato aggiungendo alla piattaforma pre-esistente un solo moltiplicatore con il suo wrapper, questo ha permesso di verificare quali accorgimenti fosse necessario prendere al fine di far funzionare il tutto. Infatti il flusso di esecuzione da seguire in un sistema con un solo moltiplicatore è più piccolo, di conseguenza è stato possibile sistemare eventuali errori più velocemente. Dopo aver ottimizzato il sistema con un solo moltiplicatore è stato aggiunto tutto il modulo e verificato il corretto funzionamento. Le figure 3, 4, 5 mostrano le waveform della memoria, dell'iomodule e del moltiplicatore, le figure 6 e 7 sono un ingrandimento delle waveform originali.

B. SystemC TLM

Il sistema è stato implementato nelle tre diverse versioni e verificato singolarmente per ciascuna, prima attraverso l'esecuzione di una sola moltiplicazione uguale per tutti gli stili di codifica, quindi con la generazione random dei valori. La verifica della corretta sincronizzazione è stata effettuata attraverso l'ordine dei messaggi mostrati durante la simulazione.

Il confronto tra i tempi di simulazione dei diversi stili di codifica di SystemC TLM e SystemC RTL conferma il divario tra astrattezza, quindi tempi di simulazione più brevi (SystemC TLM), e precisione temporale con maggiori dettagli di implementazione, quindi una simulazione più lenta (SystemC RTL). Nel sistema in esame ritengo che un'implementazione ad un livello di astrazione più basso possa essere migliore, anche se più lenta, dato che nella versione RTL sono state prese decisioni riguardo la tipologia di arrotondamento e sono state gestite le eccezioni, mentre nella versione ad alto livello viene eseguita la moltiplicazione semplice, senza valutare tutti questi aspetti che nel caso della virgola mobile sono certamente importanti. Naturalmente se il sistema fosse rivolto ad un mercato con un elevato numero di concorrenti scegliere l'implementazione in SystemC TLM avvantaggerebbe sul TTM (Time To Market), dato che è risultato più veloce e semplice sviluppare tutte e tre le versioni in SystemC TLM piuttosto che quella in SystemC RTL. Un altro fattore che potrebbe influenzare la scelta tra i due livelli di astrazione è la reattività richiesta al sistema, se il tempo è un elemento fondamentale per il sistema richiesto la versione più adeguata è quella di SystemC TLM, altrimenti SystemC RTL.

V. Conclusioni

Lo sviluppo del moltiplicatore in virgola mobile, nei diversi stili di codifica di SystemC TLM, ha permesso di comprendere quali fattori influenzino la scelta tra i diversi livelli di astrazione.

L'integrazione del modulo implementato nel progetto precedente in una Virtual Platform ha permesso di verificare quale sia l'interazione del sistema con l'architettura sottostante e di esplorare l'architettura durante l'esecuzione.

Riferimenti bibliografici

[1] "cc65 - a freeware c compiler for 6502 based systems." [Online]. Available: https://www.cc65.org/

[2] "Ieee standard for standard systemc language reference manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, Jan 2012.
[3] Synopsys, "Describing synthesizable rtl in systemc."

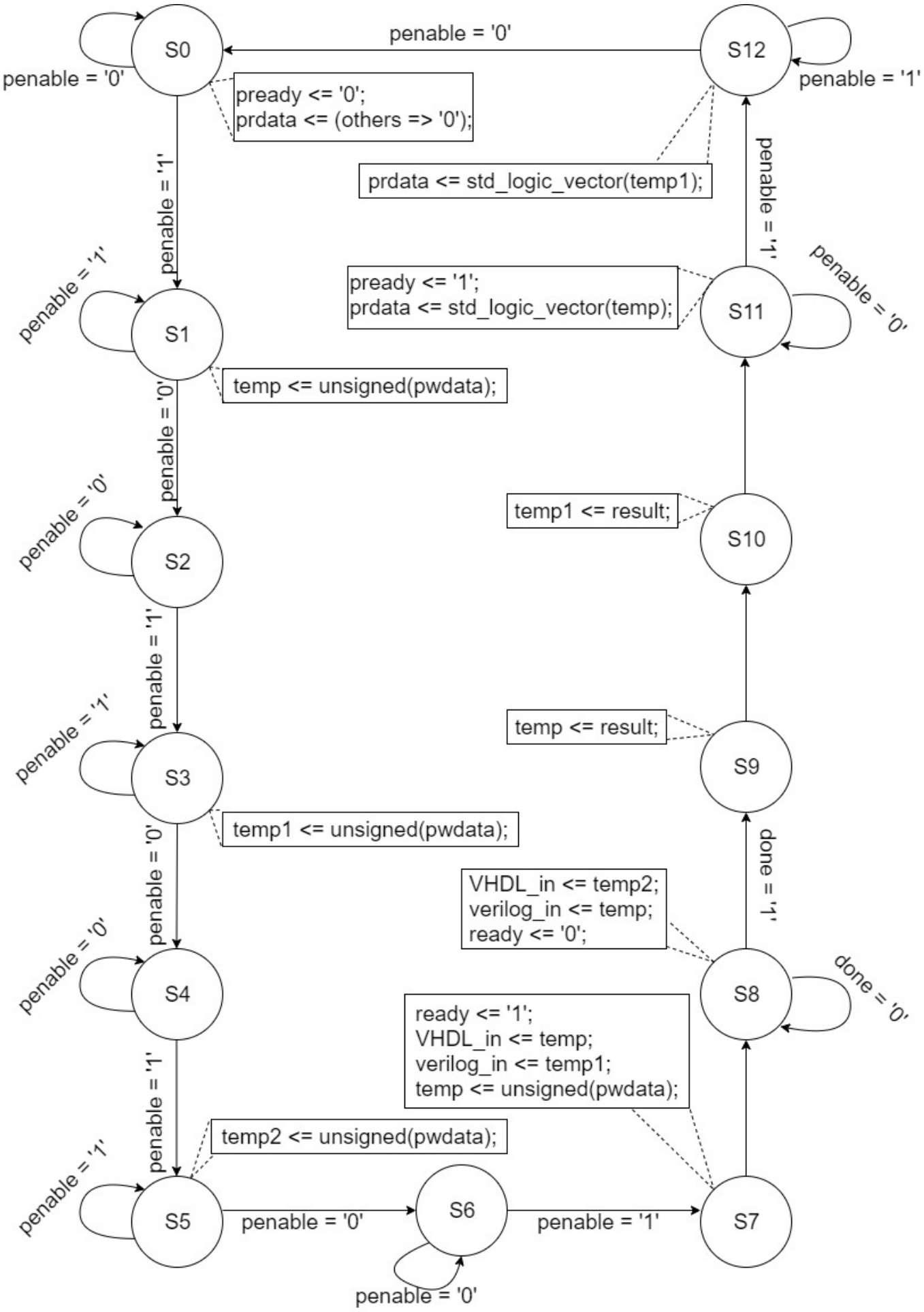

Figura 1: EFSM Floating Point Multiplier Wrapper

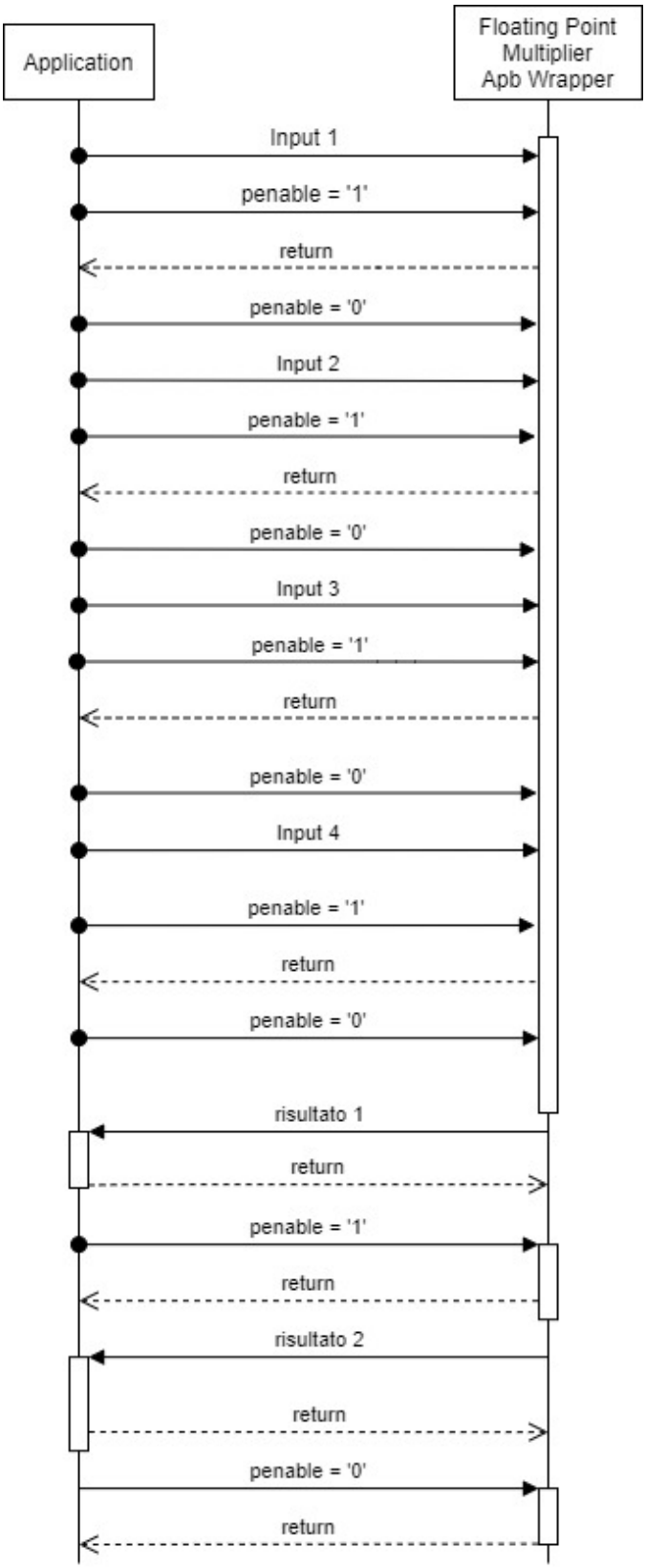

Figura 2: Diagramma di sequenza del protocollo di comunicazione

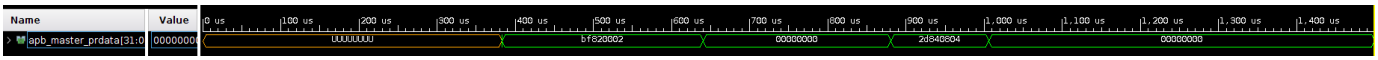

Figura 3: Waveform output della memoria

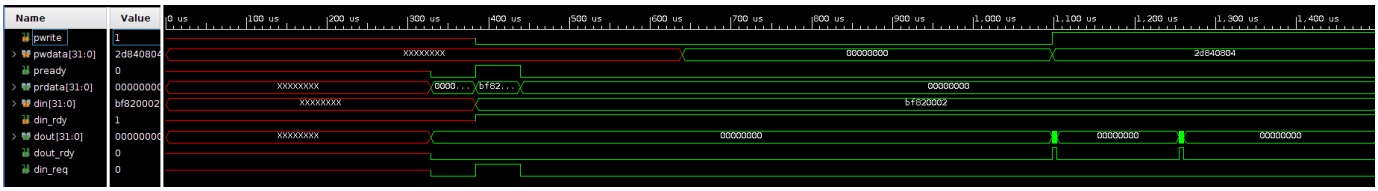

Figura 4: Waveform modulo moltiplicatore

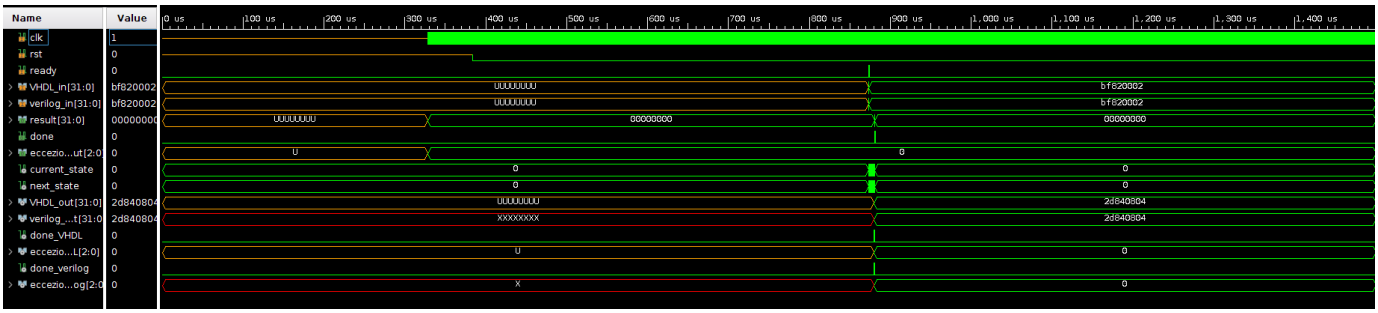

Figura 5: Waveform modulo moltiplicatore

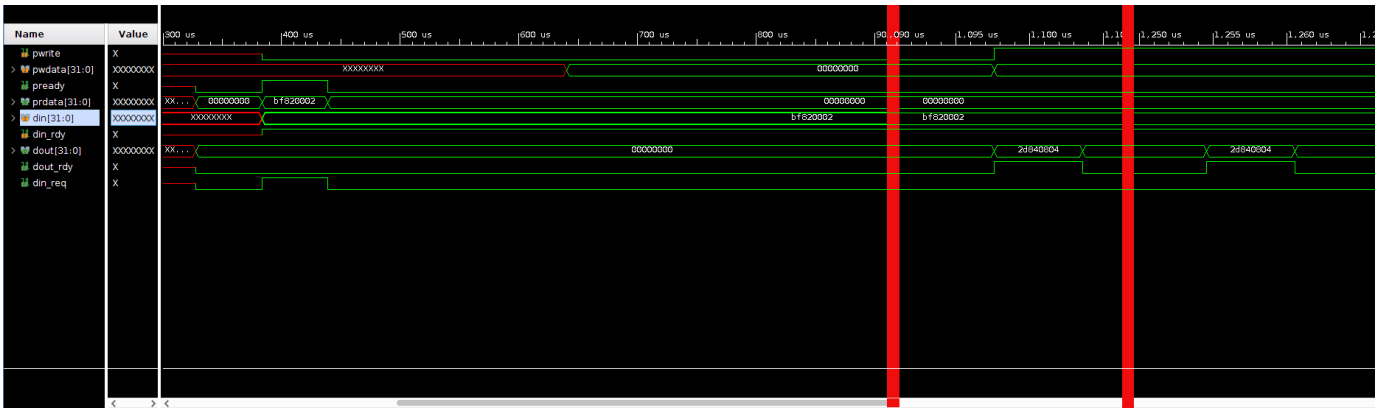

Figura 6: Ingrandimento waveform modulo I/O

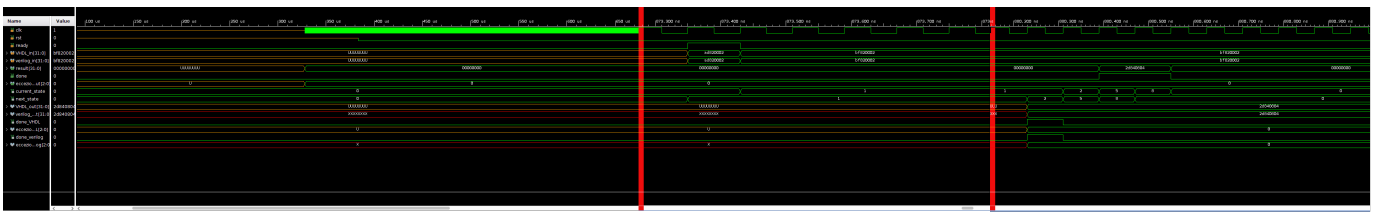

Figura 7: Ingrandimento waveform modulo moltiplicatore