Prova finale di Reti Logiche

A.A 2021 – 2022

prof. Salice Fabio

Lorenzo Franze' 10679981 - 935221

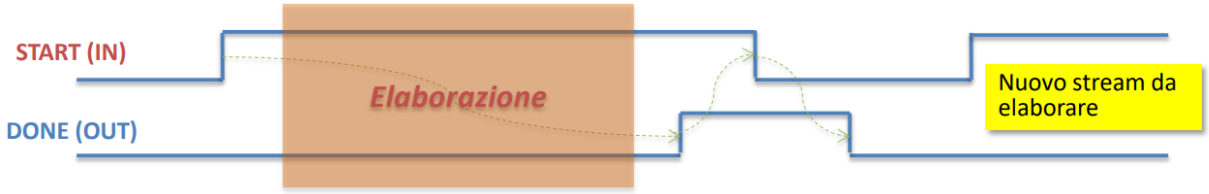
1 Introduzione e descrizione del progetto

La specifica richiede di implementare un modulo hardware in linguaggio VHDL e sintetizzarlo attraverso il tool Vivado.

Il modulo dovrà leggere da una memoria RAM con indirizzamento al byte prima all'indirizzo 0 il numero di parole da elaborare e in seguito le parole a partire dall'indirizzo 1 della memoria, per un massimo di 255 parole. Si suppongono le parole composte da 1 byte come la memoria.
A partire da ogni parola dovranno essere generate 2 parole attraverso l'uso di un altro componente: il convolutore (verrà spiegato in seguito). Infine, il risultato dovrà essere nuovamente scritto in memoria a partire dall'indirizzo 1000. In totale il numero di parole ottenute è il doppio delle precedenti.

Il componente da implementare dovrà poi essere in grado di codificare più flussi uno dopo l'altro, cioè finita l'elaborazione sulla memoria ogni volta che viene alzato un segnale detto START, il componente dovrà rieseguire l'elaborazione da capo.
Il segnale DONE viene usto per segnalare la fine della codifica.

Rappresentazione grafica:

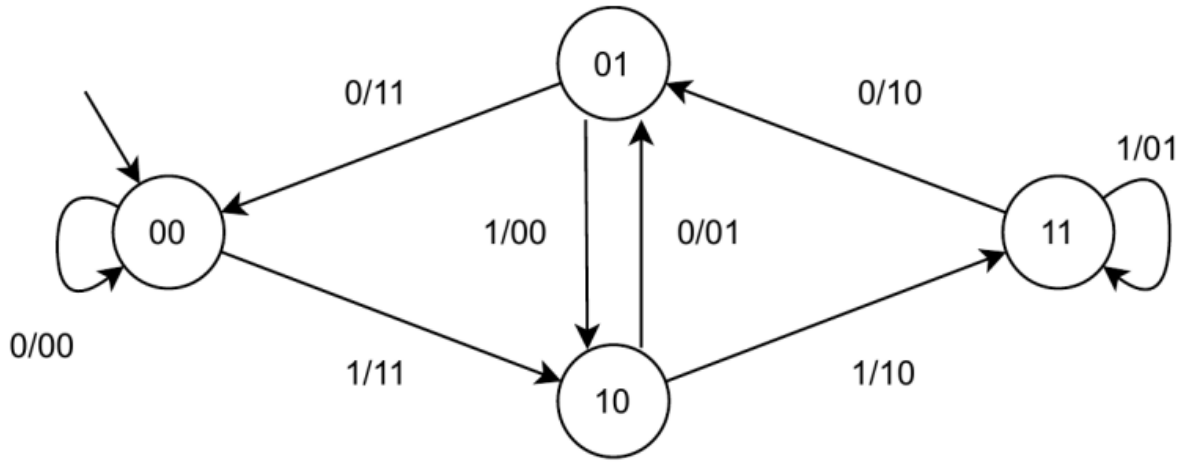


È infine necessario gestire anche un segnale di RESET.

CONVOLUTORE

Il convolutore è un codificatore che nel caso specifico applica una codifica con tasso di trasmissione ½ (ogni bit viene codificato con 2 bit):

Il funzionamento del convolutore è rappresentabile attraverso la seguente macchina di Mealy:



Il componente è pertanto dotato di memoria: il risultato di una codifica dipende anche dalla codifica precedente (stato in cui si trova il convolutore).
Il suo stato di reset è 00.

2 Architettura e scelte implementative

Il componente principale è *project_reti_logiche*, implementato con una macchina di Moore, utilizza per semplicità due moduli interni:

- *convolutore* : implementa il componente **convolutore** precedentemente descritto con delle leggere modifiche.
- *operations* : modulo che svolge una serie di operazioni: determina quando terminare la codifica, determina i bit di ingresso al convolutore e salva dei valori utili al funzionamento di *project_reti_logiche*.

convolutore

La macchina di Mealy precedente è implementata attraverso due process:

- Il primo process determina l'avanzamento sincrono della macchina e l'eventuale reset, il nuovo stato viene aggiornato solo se il segnale *conv_on* è attivo altrimenti la macchina rimane bloccata nello stato precedente. Ad ogni aggiornamento, inoltre, il segnale *variaz* viene aggiornato con l'inverso del suo valore precedente (motivo spiegato in seguito).
- Il secondo assegna un valore ai segnali *conv_P1k* e *conv_P2k* in base allo stato corrente e al segnale di ingresso *conv_Uk*; il process ha la particolarità di essere sensibile alla variazione di *conv_Uk* in modo tale da aggiornare immediatamente le uscite.

Il modulo *convolutore* riceve in ingresso i segnali:

- *i_clk*
- *conv_rst* : porta la macchina nello stato S0.
- *conv_on* : permette di far avanzare la macchina solo quando è alto.
- *conv_Uk* : bit da elaborare.

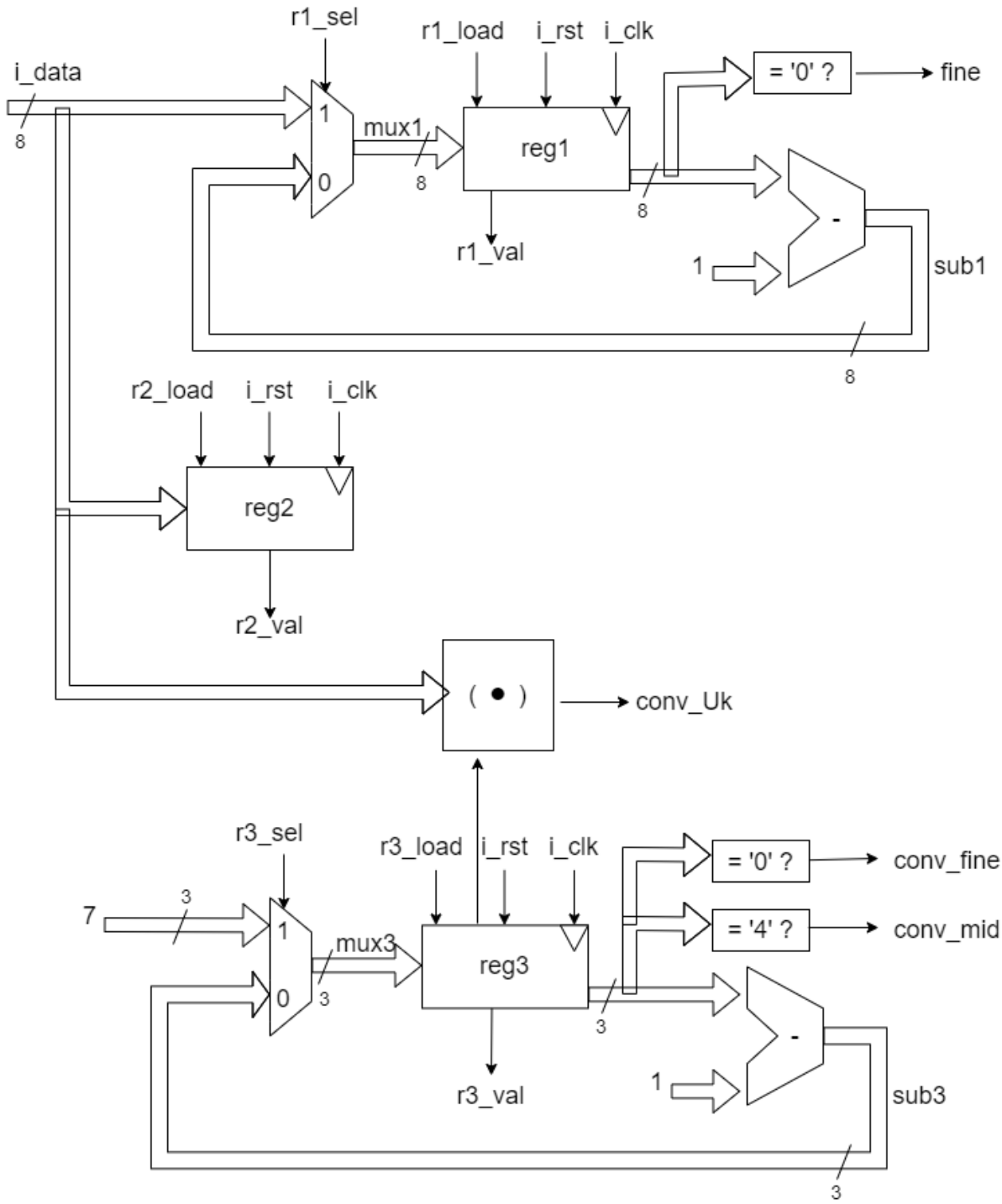
In uscita:

- *conv_P1k e conv_P2k* : bit risultati dell'elaborazione.
- *variaz :* segnale che indica a un process di *project_reti_logiche* che è avvenuta una elaborazione, il segnale è necessario per quei casi in cui né *conv_P1k* né *conv_P2k* variano valore; senza di esso il process di *project_reti_logiche* non si "risveglia".

Operations

Il componente si interfaccia direttamente con i dati letti da memoria (*i_data*) e a partire da essi svolge una serie di operazioni che ad alto livello sono:

- componenti intorno al registro *reg1* : nel registro viene salvato il numero di parole da elaborare (dall'indirizzo 0 della memoria) e per ogni parola elaborata si sottrae un'unità; quando il segnale *fine* è a zero allora tutte le parole sono state elaborate dal *convolutore*
- *reg2 :* nel registro viene solo salvato e mantenuto per tutto il tempo il primo valore letto ossia il numero di parole da elaborare; il valore viene utilizzato da *project_reti_logiche* per determinare gli indirizzi di memoria in cui scrivere i risultati
- componenti intorno al registro *reg3 :* un semplice contatore da 7 a 0 che segnala quando il valore è 4 e quando è 0 a *project_reti_logiche;* Il contenuto del registro viene però utilizzato dal componente successivo per selezionare i singoli bit delle parole
- componente (•) : seleziona da *i_data* il bit indicato dal registro *reg3*



Di seguito sono descritti i segnali che meritano maggiore attenzione (tutti segnali di output) :

- *r1_val, r2_val, r3_val* : valori dei rispettivi registri portati in output da *operations*
- *fine* : segnala che sono state elaborate tutte le parole indicate all'indirizzo 0 della memoria
- *conv_fine :* segnala che tutti i bit della parola sono stati analizzati
- *conv_mid :* indica che è stata raggiunta la metà della parola
- *conv_Uk :* bit di ingresso al modulo *convolutore*

project_reti_logiche

La macchina di Moore definita prima "dirige" tutte le operazioni che svolgono gli altri due componenti, ha S0 come stato di reset ed è rappresentata nella pagina successiva;

Il componente è implementato con tre process:

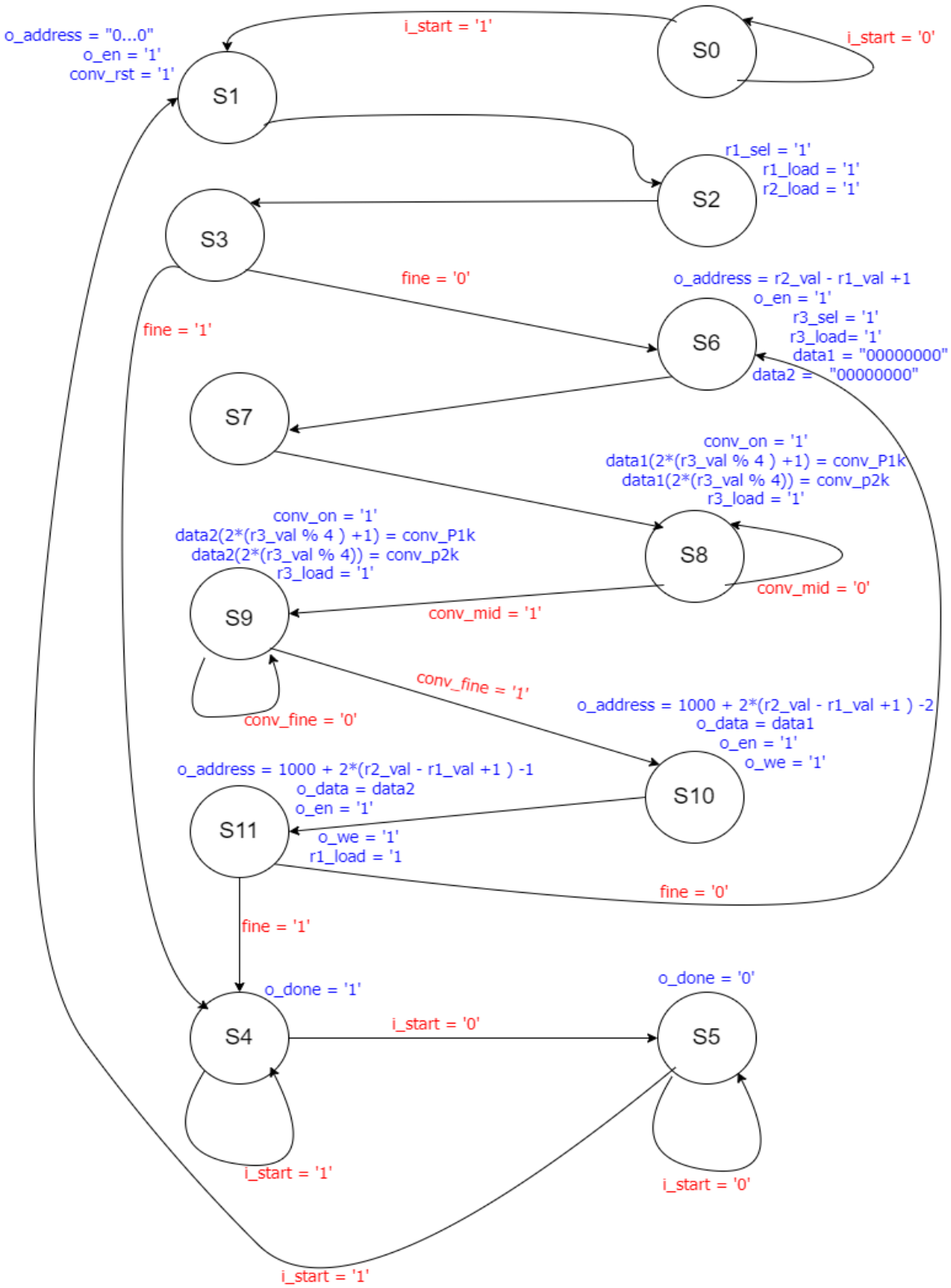
- Il primo determina l'avanzamento degli stati della macchina su ogni fronte di salita del clock e gestisce il reset asincrono
- Il secondo implementa la funzione *delta* e contiene nella lista di sensitività oltre a *cur_state* anche *i_start*, *fine*, *conv_fine* e *conv_mid* in questo modo gli stati vengono aggiornati non appena c'è una variazione dei precedenti segnali
- Il terzo implementa la funzione di uscita e contiene nella lista di sensitività oltre a cur_state anche *conv_P1k*, *conv_P2k* e *variaz* in questo modo il process si "attiva" e aggiorna le uscite non appena il modulo *convolutore* aggiorna *conv_P1k* o *conv_P2k,* per quei casi in cui né *conv_P1k* né *conv_P2k* variano valore allora il process si attiva perchè *variaz* cambia valore.

Descrizione degli stati della macchina:

- S0: stato di reset in cui rimane la macchina finché *i_start* non viene alzato
- S1: richiesta di lettura del valore all'indirizzo 0 della memoria
- S2: abilita i registri *reg1* e *reg2* al salvataggio del precedente valore
- S3: stato di attesa per i valori da memoria a causa dei 2 ns di ritardo: non avviene nulla; in totale essendo la memoria sincrona si attendono 2 cicli di clock
- S4: elaborazione completata: *o_done* viene alzato.
- S5: si attende una eventuale nuova elaborazione.
- S6: richiesta di lettura dell'indirizzo N = *r2_val* – *r1_val* – 1 ricordando che *r2_val* contiene sempre il numero di parole totali da elaborare e *r1_val* contiene il numero di parole ancora da elaborare (terminata l'elaborazione della parola il registro viene decrementato).
- S7: stato di attesa.
- S8: viene abilitato il *convolutore* che produce le uscite *conv_P1k* e *conv_P2k* salvandone i valori immediatamente in data1; i valori dell'indice di *data1* sono calcolati attraverso *r3_val* (in cui è salvato l'indice del bit di *conv_Uk* ottenuto da *i_data*) e dalle operazioni aritmetiche; l'operazione è svolta per i primi 4 bit della parola.
- S9: operazione analoga a S8 con la differenza che essendo stata superata la metà della parola ora i dati vengono salvati in *data2*.
- S10: salvataggio in memoria di *data1* all'indirizzo: 1000 + 2*N -2 (stesso valore di N precedente).
- S11: salvataggio in memoria di *data2* all'indirizzo: 1000 + 2*N -1 e aggiornamento di N che aumenta di un'unità.

Si suppone che in ogni stato se non diversamente specificati i valori di default sono:

r1_sel = '0'
r1_load ='0'
r2_load = '0'
r3_sel = '0'
r3_load = '0'
conv_rst = '0'
conv_on = '0'
o_address = "0000000000000000"
o_data = "00000000"
o_done = '0'
o_en = '0'
o_we = '0'



considerazioni sull'implementazione

- Essendo il modulo *convolutore* una macchina di Mealy per cui a seconda del valore di *conv_Uk* sono già disponibili le uscite, si è deciso di aggiornare immediatamente *data1* e *data2* attraverso un process sensibile alle uscite del *convolutore: conv_P1k*, *conv_P2k* e *variaz,* senza dover aspettare un ciclo di clock in più.
- La sensibilità a *conv_P1k* e *conv_P2k* serve per i casi in cui almeno uno dei due valori cambia, quella a *variaz* per i casi in cui i segnali precedenti non cambiano ed è necessario attivare il process per scrivere in *data1* o *data2* gli stessi valori precedentemente calcolati.
- *data1* e *data2* per essere aggiornati immediatamente vengono implementanti come *latch* (visibile nel report di sintesi).
- Dato il basso ordine di grandezza dei valori si è preferito un approccio più algebrico con anche prodotti e operazioni modulari eseguiti fortunatamente con potenze del 2 (*2 e mod 4), utilizzando così i dati già a disposizione.

3 Risultati sperimentali

sintesi: report di sintesi

Il modulo è stato correttamente sintetizzato e dai *reports di sintesi* sono state riscontrati i seguenti aspetti (*board* utilizzata: xc7k160tfbv676-2L) :

- rispettato il vincolo di almeno 100 ns sul ciclo di clock, con uno *slack* di 98.193 ns e un *path delay* di 1,452 ns. Ad esempio, le simulazioni funzionano anche con una frequenza di clock di 50 volte quella data.
- vengono sintetizzati 81 LUT e 41 SLICE REGISTERS di cui 25 *flip flop* (8 per *reg1*, 8 per *reg2*, 3 per *reg3,* 2 per gli stati della macchina di Mealy e 4 per gli stati della macchina di Moore) e 16 *latch* (8 per *data1* e 8 per *data2*). Sono stati mantenuti i latch per semplicità e per non complicare ulteriormente l'architettura.
- la sintesi non inferisce direttamente le 2 *fsm*, vengono però sintetizzati 2 flip flop di tipo D per la macchina con 4 stati e 4 flip flop di tipo D per la macchina con 12 stati.
- vengono inferiti degli *adders* e dei *muxes* anche in *project_reti_logiche* probabilmente dovuti alle operazioni sugli indirizzi.

simulazioni e test fatti

Il componente è stato testato e simulato in pre-sintesi e post-sintesi sempre con risultati positivi.

Oltre ai test di esempio nella specifica sono stati provati i seguenti test e casi limite:

- memoria formata da tutti 0.
- memoria che impone di leggere 0 parole e con parole successive diverse da 0.
- memoria che impone di leggere il massimo numero possibile di parole: 255.
- simulazioni con segnali di reset.
- simulazioni che al termine dell'elaborazione la fanno ripartire altre volte sulla stessa memoria.
- simulazioni che al termine dell'elaborazione la fanno ripartire altre volte su memorie diverse.

Infine, un primo test che ha permesso di scoprire alcuni errori è stato l'Esempio 2 della specifica il cui output era tutto corretto eccetto il 37 all'indirizzo 1003; la simulazione dava un 36 perché non si consideravano i casi in cui né *conv_P1k* né *conv_P2k* variavano rispetto al valore precedente, per questo è stato inserito il segnale *variaz.*

4 Conclusioni

Si ritiene che l'architettura rispetti tutte le specifiche, il componente è stato testato numerose volte sia in pre-sintesi che post-sintesi.

Nella progettazione si è cercato di mantenere l'architettura il più semplice possibile utilizzando la divisione in moduli.

Infine, si è cercato di evitare la creazione di ulteriori moduli per operazioni semplici come il calcolo degli indirizzi di memoria o il salvataggio dei risultati intermedi dell'elaborazione prima di scrivere in memoria, preferendo svolgere tali operazioni in un'unica *entity* ed evitando così un codice più verboso e meno chiaro.