Documentazione progetto RL 2022

Ronzani Marco – c.p. 10669641 – mat. 934552 – Politecnico di Milano

Indice

1 Specifiche di progetto

È richiesta l'implementazione di un modulo che applichi il codice convoluzionale ½ ad un flusso continuo di bit letti da una memoria con cui è necessario il modulo si interfacci.

Nel dettaglio si può dividere la specifica in tre parti, due di interfaccia ed una di elaborazione:

- L'interfaccia del modulo con la memoria si basa sui seguenti segnali:
  - `o_address 16bit - indirizzo della memoria attualmente letto e/o scritto, deve essere controllato adeguatamente dal modulo`
  - `o_en - segnale di abilitazione della memoria`
  - `o_we - segnale di abilitazione della scrittura sulla memoria`
  - `o_data 8bit - dati da scrivere sulla memoria se o_we è alzato`
  - `i_data 8bit - dati letti dalla memoria`

  Il modulo deve dunque controllare adeguatamente l'indirizzo della memoria sul quale sta operando, leggendo innanzitutto da `0x0000` la quantità di byte da processare e procedendo poi in ordine a leggere tale quantità di byte partendo da `0x0001`, scrivendo invece i risultati dall'indirizzo `0x03E8` in avanti.

  È di rilievo il fatto che la memoria presenti, come indicato dalla documentazione Xilinx per una `Single-Port Block RAM Write-First Mode` (https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synth), un ritardo in lettura di 2ns e nessun ritardo in scrittura.

  La costruzione del flusso di singoli bit richiesto per l'elaborazione deve essere fatta sempre partendo dal bit più significativo (big-endian) di `i_data`, e lo stesso byte order deve essere usato per la scrittura su `o_data` dopo la convoluzione.

- L'interfaccia del modulo con l'esterno si basa sui segnali:
  - `o_done - indicatore di operazione completata`
  - `i_clk - clock fornito al modulo`
  - `i_rst - reset fornito al modulo`
  - `i_start - segnale di richiesta di inizio operazione`

  Il protocollo che i precedenti segnali devono rispettare è:

  Prima della prima operazione è sempre fornito un `reset` mentre ogni altro segnale è 0, dopo il quale può venire alzato il segnale di `start`. Il segnale di `start` non verrà abbassato finché il modulo non porterà `done` ad 1, solo dopo che `done` è stato alzato, `start` potrà venire abbassato. A seguito di ciò anche `done` dovrà essere abbassato. Tornati in questa configurazione potrà ripetersi il tutto, ma senza necessità dell'iniziale segnale di `reset`.

- L'elaborazione che è richiesta al modulo è una convoluzione ½ di una sequenza di bit, ovvero produrre in uscita per ogni bit in ingresso una coppia di bit dipendenti sia dal bit in ingresso che dagli ultimi 2 bit processati (si assumano questi inizialmente 0).

Sia U_k il k-esimo bit ingresso e siano $P1_k$ e $P2_k$ il due bit prodotti da esso, allora:

$P1_k = U_k \text{ xor } U_{k-2}$

$P2_k = U_k \text{ xor } U_{k-1} \text{ xor } U_{k-2}$

Una rappresentazione del convolutore è la seguente:

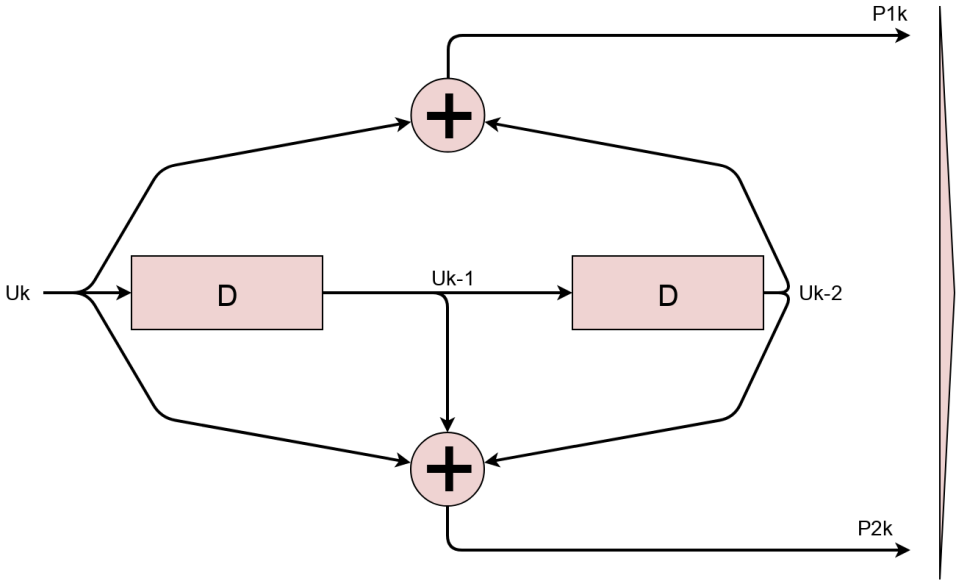


Figura 1 - rappresentazione del convolutore

Questa operazione può essere svolta da una FSM (finite-state machine) come la seguente:

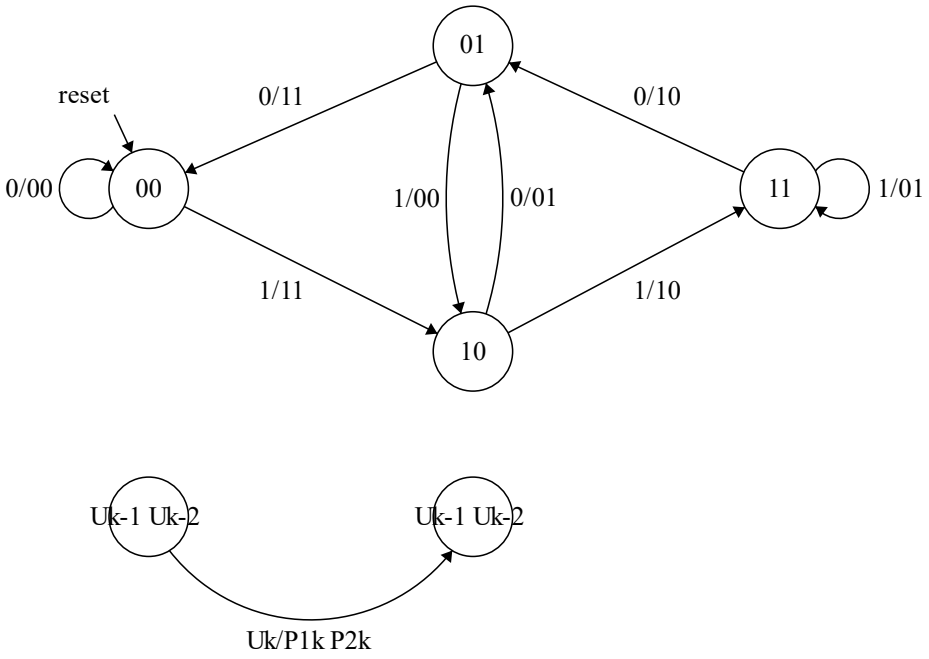


Figura 2 - esempio di FSM svolgente la convoluzione 1/2

Concludendo l'analisi delle specifiche, un esempio di elaborazione: dato in ingresso il byte 10100010 con i due bit precedenti inizializzati a 0 il risultato deve essere il seguente:

Tempo	0	1	2	3	4	5	6	7
Uk	1	0	1	0	0	0	1	0
P1k	1	0	0	0	1	0	1	0
P2k	1	1	0	1	1	0	1	1

Byte in uscita: 11010001 e 11001101.

2 Scelte progettuali

Il design si costituisce di un singolo modulo composto da un processo e 5 registri interni, sincronizzato interamente sul fronte di salita del clock. Il processo è risvegliato da cambiamenti sia in `i_rst` che in `i_clk`, quando `i_rst` è portato ad 1 lo stato diviene `STAND_BY` e ogni altro registro interno viene riportato a 0. Alternativamente, ad ogni ciclo di clock, se non vi è `i_rst` a 1, lo stato è aggiornato e ogni operazione pertinente allo stato corrente viene svolta. I registri sono volti a memorizzare informazioni utili nei diversi stati della macchina che ora saranno discussi nel dettaglio.

In ragione di quanto detto il componente implementa una FSM(D) (finite-state machine with datapath), che è la seguente:

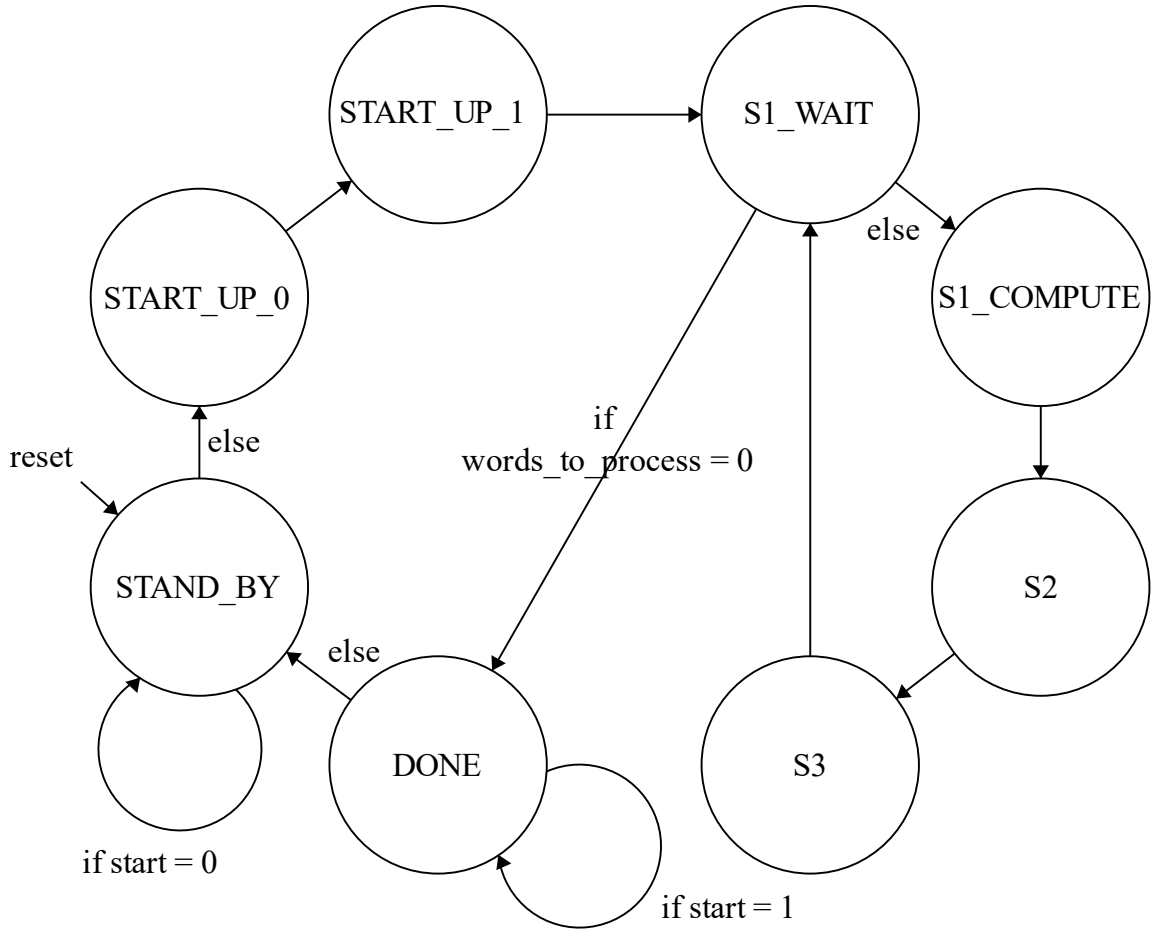


Figura 3 - FSM del componente progettato

Stato Corrente	**Stato Prossimo**	**Controlli e Azioni Datapath**
	Condizione, Stato	**Condizione, Azione**
STAND_BY	i_start = 0, STAND_BY i_start = 1, START_UP_0	i_start = 0, o_en <= 0 i_start = 1, o_en <= 1
START_UP_0	START_UP_1	<attesa memoria>
START_UP_1	S1_WAIT	words_to_process <= i_data current_address <= 1 + current_address o_address <= 1 + current_address o_en <= 1
S1_WAIT	words_to_process = 0, DONE words_to_process != 0, S1_COMPUTE	old_2_bits <= old_2_bits words_to_process <= words_to_process current_address <= current_address o_address <= current_address o_en <= 1 <attesa memoria>
S1_COMPUTE	S2	encoded_data <= (i_data(3) xor i_data(5)) & (i_data(3) xor i_data(4) xor i_data(5)) & […] (i_data(0) xor i_data(1) xor i_data(2)) o_data <= (i_data(7) xor old_2_bits(1)) & (i_data(7) xor old_2_bits(0) xor old_2_bits(1)) & […] (i_data(4) xor i_data(5) xor i_data(6)) old_2_bits <= i_data(1) & i_data(0) words_to_process <= words_to_process current_address <= current_address o_address <= current_address*2 + 998 o_en <= 1 o_we <= 1
S2	S3	o_data <= encoded_data old_2_bits <= old_2_bits words_to_process <= words_to_process current_address <= current_address o_address <= current_address*2 + 999 o_en <= 1 o_we <= 1
S3	S1_WAIT	old_2_bits <= old_2_bits words_to_process <= words_to_process - 1 current_address <= current_address + 1 o_address <= current_address + 1 o_en <= 1
DONE	i_start = 1, DONE i_start = 0, STAND_BY	i_start = 1, o_done <= 1 i_start = 0, o_done <= 0

NOTA: si assuma che ogni segnale o registro non citato venga sempre assegnato a 0.

NOTA: non è riportato l'intero calcolo di `encoded_data` e `i_data` per ragioni di leggibilità, esso è comunque deducibile dalla parte presente.

2.1 Scelte di design principali

- La computazione della convoluzione non è fatta bit per bit, ma in parallelo per 8 bit, ovvero l'intero byte letto dalla memoria è processato e i 16 bit derivanti dalla convoluzione sono computati tutti insieme.
- Il grafo orientato della FSM presenta due anelli, uno interno (rosso nel diagramma) ed uno esterno (blu nel diagramma). L'anello esterno include quello interno e si compone degli stati di `DONE`, `STAND_BY`, `START_UP_1` e `START_UP_2`, seguiti dall'anello interno. L'anello interno itera invece sugli stati di `S1_WAIT`, `S1_COMPUTE`, `S2` ed `S3`. Nell'anello interno viene svolta la convoluzione e scrittura di un byte letto dalla memoria per ogni iterazione completa, mentre l'anello esterno viene percorso dopo un reset o tra diverse fasi di lavoro sull'anello interno, poiché esso gestisce i segnali di interfaccia `start`, `done`, `reset`, ed `enable`.

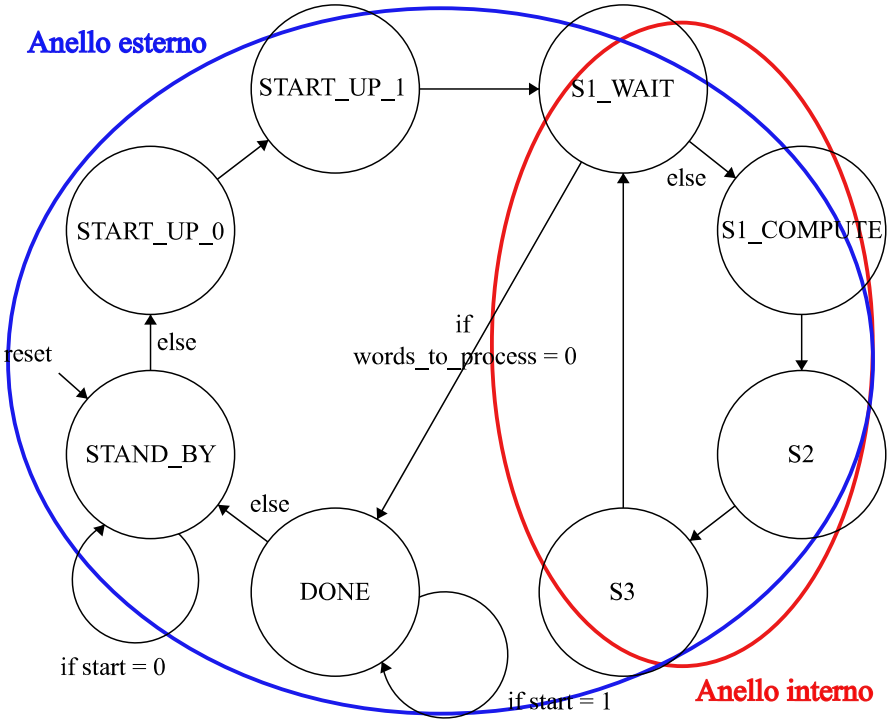


Figura 4 - FSM del componente con evidenziati gli anelli

- Ogni 4 cicli di clock, ovvero ogni iterazione completa dei 4 stati dell'anello interno, il modulo legge dalla memoria un byte, ne calcola la convoluzione ottenendo 2 byte e scrive questi risultati in memoria, il costo in cicli di clock è di 2 per la singola lettura dalla memoria e 1 per ogni scrittura necessaria, indi 2 complessivamente.
  Il motivo dei 2 cicli in lettura è che, dopo aver impostato l'indirizzo da cui leggere nel primo ciclo di clock, occorre attendere il successivo prima di leggere l'output della memoria, questo è dovuto al fatto che la memoria presenta ritardi in lettura che non possono garantire l'immediata disponibilità del dato richiesto. Al contrario la memoria non presenta ritardi in scrittura (vedi specifica) e ciò consente di risolvere le due scritture in 2 soli cicli.

- Il reset del modulo è asincrono, mentre ogni altra operazione è sincronizzata sulla rising edge del clock.
- Ad ogni ciclo di clock, per ogni stato interno, e durante il reset è sempre assegnato ogni output ed ogni registro interno, grazie a questo l'intero modulo è sintetizzabile senza uso di latch.

2.2 Registri e tipi

- `type state_type is (STAND_BY, START_UP_0, START_UP_1, S1_WAIT, S1_COMPUTE, S2, S3, DONE)`
  Tipo enumerazione degli 8 stati di cui si costituisce la macchina.
- `signal state : state_type`
  Vettore memorizzato in un registro a 3 bit, contenente il corrente stato della macchina.
- `signal encoded_data : std_logic_vector (7 downto 0)`
  Vettore che memorizza temporaneamente gli ultimi 8 bit dei 16 prodotti da una convoluzione di 8 bit letti dalla memoria. Usato tra gli stati di `S1_COMPUTE` e `S2`.
- `signal old_2_bits : std_logic_vector (1 downto 0)`
  Vettore volto a preservare gli ultimi 2 bit (i 2 meno significativi) del byte letto dalla memoria attraverso le diverse iterazione dei 4 stati dell'anello interno.
- `signal current_address : std_logic_vector (7 downto 0)`
  Indirizzo dal quale la macchina sta attualmente leggendo, viene incrementato di 1 ad ogni iterazione dell'anello interno. Esso viene sempre esteso con 8 zeri al fine di essere usato come `o_address` ed eventualmente viene sommato a sé stesso e 998 o 999 per produrre gli indirizzi di memoria ove scrivere.
- `signal words_to_process : std_logic_vector (7 downto 0)`
  Contatore dei byte (words) che è richiesto il modulo processi, il suo valore è quello letto dall'indirizzo 0x0000 della memoria e viene decrementato di 1 ad ogni iterazione dell'anello interno. Quando il suo valore raggiunge 0 la macchina esce dall'anello interno e passa allo stato di `DONE`.

2.3 Stati

- `STAND_BY`: stato iniziale della macchina, raggiunto dopo un reset o alla fine di una fase operativa, in attesa della successiva. In questo stato il componente è in idle, in attesa del segnale di start. In questo stato ogni registro interno è resettato a 0.
- `START_UP_0`: primo stato di una nuova fase operativa del componente, in esso si alza il segnale di enable dalla memora e si imposta l'indirizzo di lettura su 0x0000, per leggere il numero dei byte da processare.
- `START_UP_1`: secondo stato di una nuova fase operativa del componente, in esso si legge il contenuto dalla memoria richiesto in `START_UP_0`, in numero di byte da processare, e lo si salva in `words_to_process`, dopodiché si incrementa l'indirizzo corrente di lavoro sulla memoria, `current_address`, a 0x0001.

- `S1_WAIT`: primo stato dell'anello interno, esso permette l'uscita dall'anello interno qualora `words_to_process` fosse 0, alternativamente prepara la memoria per una lettura dal corrente indirizzo di lavoro salvato in `current_address`.
- `S1_COMPUTE`: secondo stato dell'anello interno, legge il byte da processare dalla memoria e ne produce i due byte derivanti dalla convoluzione ½. Salva il secondo byte in `encoded_data` e pone invece il primo byte già su `o_data` affinché venga scritto all'indirizzo corretto, impostato in questo stato come `current_address` + `current_address` + 998. Infine, in questo stato sono anche aggiornati i due bit che occorre memorizzare per la prossima convoluzione, `old_2_bits`. Write enable è alzato.
- `S2`: terzo stato dell'anello interno, in esso viene posto `encoded_data` su `o_data` e l'indirizzo di scrittura è posto a `current_address` + `current_address` + 999. Write enable rimane alzato.
- `S3`: quarto e ultimo stato dell'anello interno, in esso si incrementa `current_address` di 1, si decrementa di 1 `words_to_process` e si prepara la lettura dalla memoria del successivo byte da processare. Write enable viene abbassato.
- `DONE`: stato di uscita dall'anello interno, vi si arriva da `S1_WAIT` dopo che `words_to_process` raggiunge 0, si rimane in questo stato fino a quando il segnale di start non è abbassato, dopodiché si torna in `STAND_BY`. In questo stato ogni registro interno è resettato a 0, l'enable dalla memoria è abbassato e `done` è 1.

Il numero degli stati della macchina è minimizzato, in quanto 4 cicli sono il minimo per l'anello interno, visti i requisiti di 2 cicli per la lettura e 1 per ogni scrittura. Allo stesso modo 4 stati addizionali sono il minimo per l'anello esterno, visto che 2 sono necessari per la lettura del numero di parole da processare e gli stati di `STAND_BY` e `DONE` sono necessari per soddisfare la specifica.

Questa è la schematica del design prodotta da Vivado:

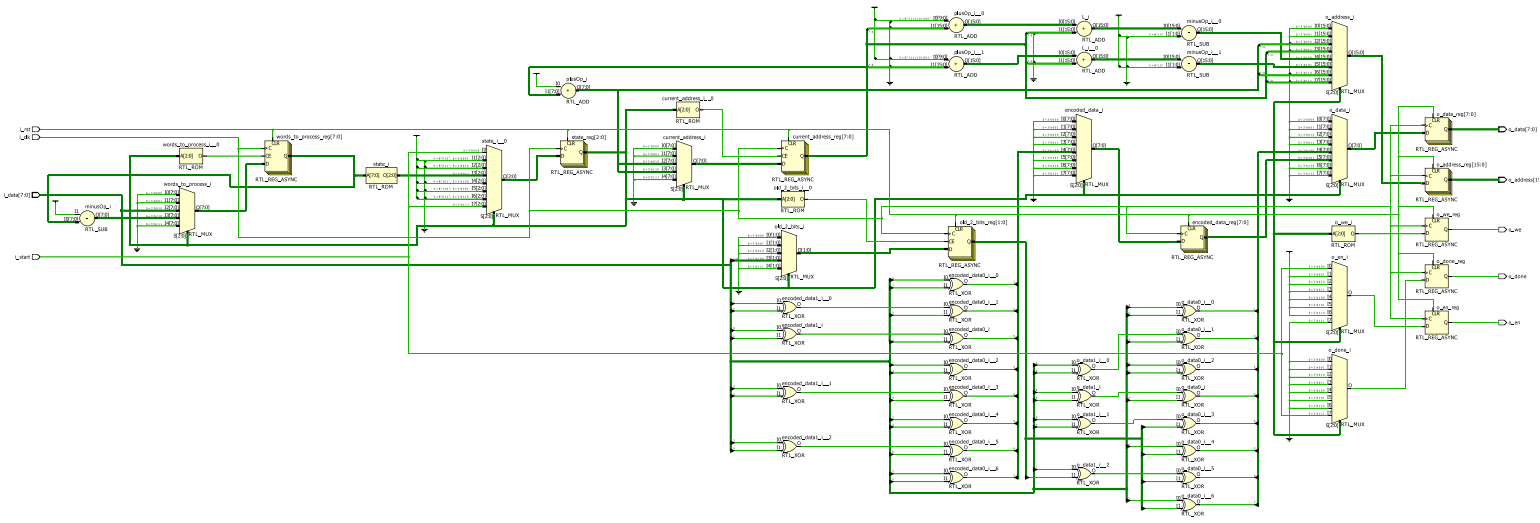


Figura 5 - schematica del design

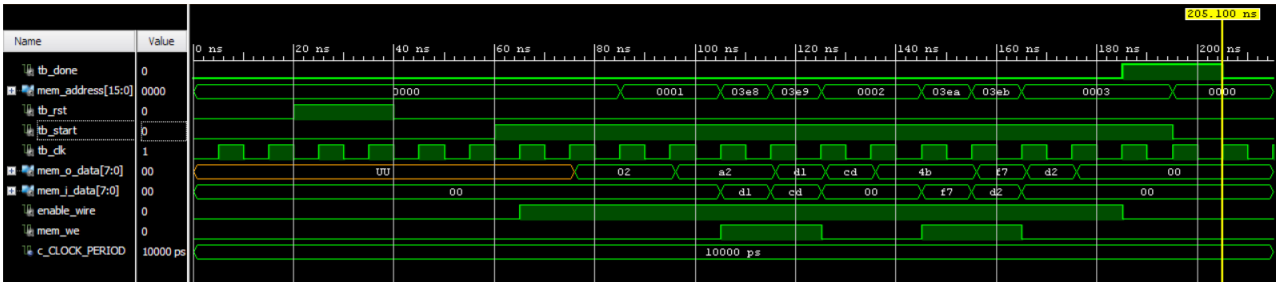
3 Risultati dei test

Al fine di verificare il corretto funzionamento del modulo sono stati svolti test sia inerenti a varie condizioni di normale funzionamento che al fine di coprire molteplici situazioni limite.
Seguono le brevi descrizioni dei test con i rispettivi esiti delle post-synthesis functional simulations effettuate con Vivado 2016.4. Per ogni test il corretto funzionamento è stato testato anche tramite una post-synthesis timing simulation.

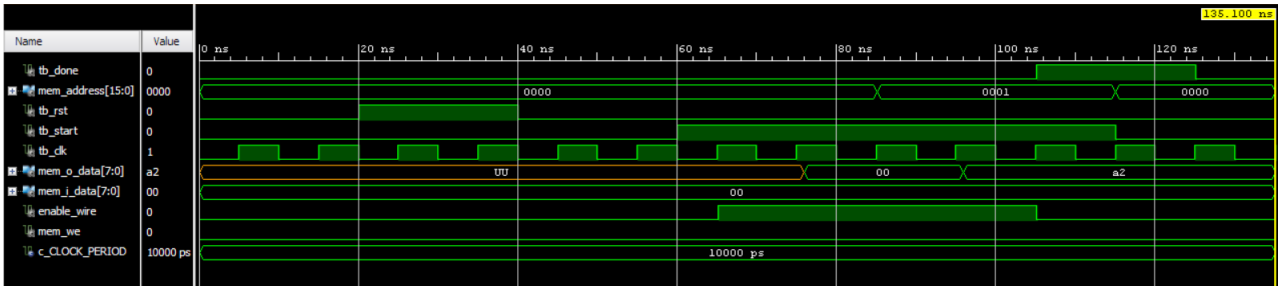
Ogni test è stato eseguito con un periodo di clock di 10ns.

3.1 Tests

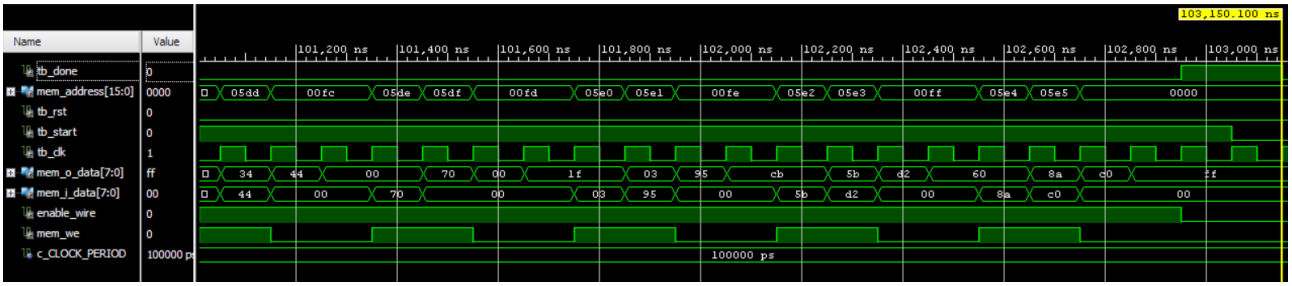
- Test svolto con la testbench fornita insieme alle specifiche.
  Scopo: preliminare verifica del corretto calcolo della convoluzione e la corretta risposta ai segnali di `start` e `reset`.



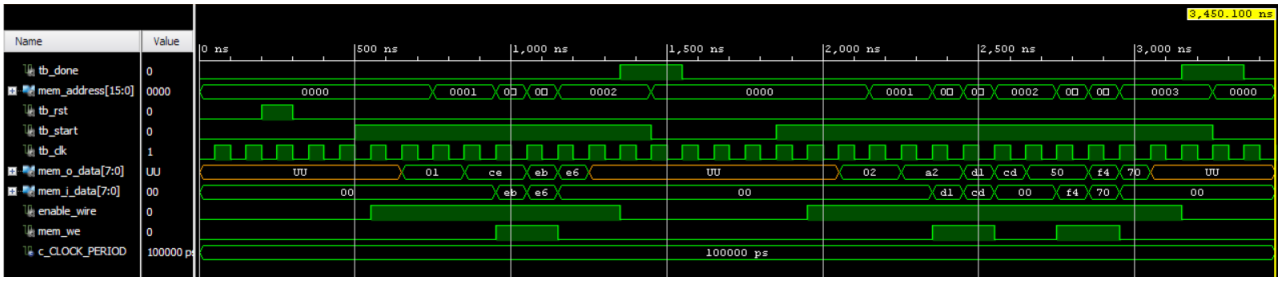
- Test con numero di parole da processare pari a zero.
  Scopo: verificare che il caso limite non abbia conseguenze sul modulo e che esso termini direttamente, senza svolgere scritture sulla memoria.



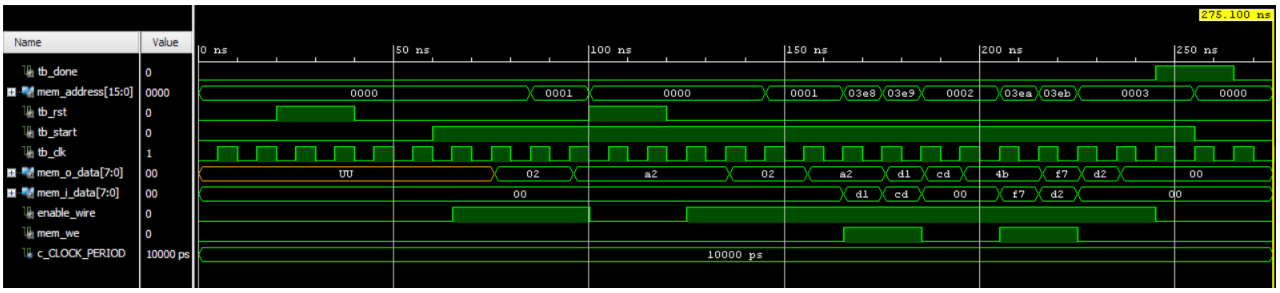
- Test con numero di parole da processare massimo, 255, trattandosi di un numero ad 8 bit.
  Scopo: verificare che il modulo possa soddisfare la specifica richiedente per una sequenza massima di almeno questa lunghezza.



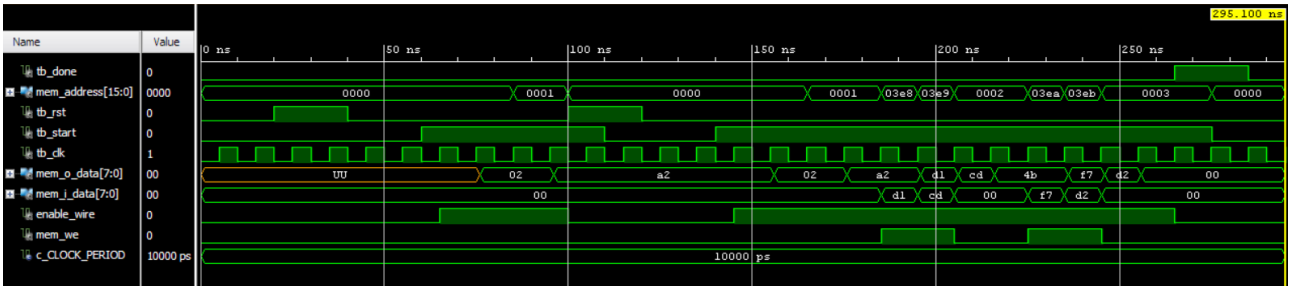
- Test di due sequenze diverse da processare una dopo l'altra, senza reset in mezzo.
  Scopo: verificare che il modulo possa processare più sequenze senza necessità di un reset tra esse, come da specifica.



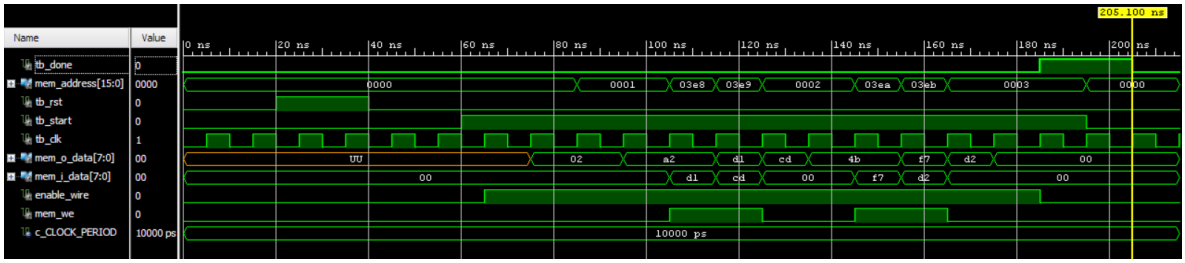
- Test di reset asincrono durante la fase operativa, con start che rimane alzato, e successivo completamento di una sequenza da processare.
  Scopo: verificare che l'uso del segnale di reset in istanti arbitrari non comprometta il funzionamento del modulo e che esso possa riprendere a funzionare immediatamente dopo, ripartendo dallo stato di `STAND_BY`.



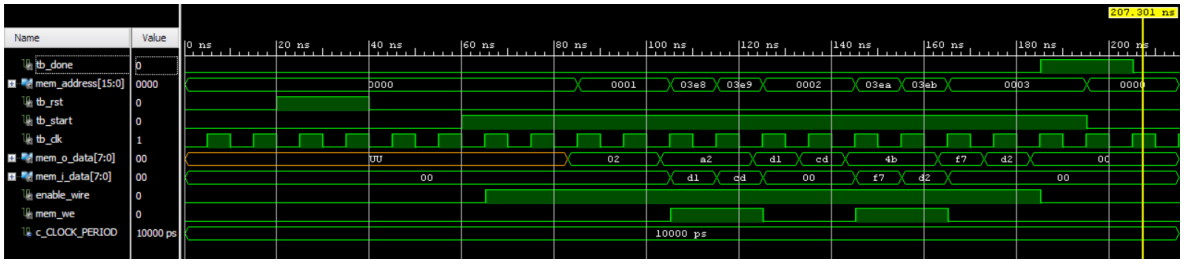
- Test di reset asincrono durante la fase operativa, con start che viene abbassato durante il reset, e successivo completamento di una sequenza da processare.
  Scopo: verificare che l'uso del segnale di reset in istanti arbitrari riporti il modulo nella sua condizione iniziale e che il modulo possa quindi da lì ripartire a seguito di un altro segnale di start.



- Test con differenti ritardi della memoria.
  Scopo: verificare la tolleranza del modulo a diversi ritardi della memoria, purché inferiori al suo periodo di clock.
  - ritardo nullo



  - ritardo a 8ns



- Test di ogni possibile convoluzione. Ovvero testare abbastanza sequenze da avere almeno una volta ogni byte preceduto da ogni possibile byte, per ottenere queste 256^2 coppie di byte da testare, il test è stato effettuato con 258 sequenze da 254 byte e una da 4 byte, nell'arco delle quali ogni possibile coppia di byte viene testata.
  Il motivo delle coppie è che il risultato della convoluzione dipende sia dal byte precedente che da quello corrente. Sarebbe corretto osservare che, poiché il byte precedente conta solo per gli ultimi 2 bit, sarebbero bastate meno sequenze, ma il risultato sarebbe stato equivalente.
  Scopo: dimostrare la correttezza dell'implementazione del convolutore.
  (Immagine omessa in quanto poco significativa a causa della lunghezza della simulazione)

Oltre ai test sopra elencati sono stati eseguiti 20 test generati casualmente, ognuno costituito da un numero casuale (tra 1 e 32) di sequenze da processare in successione, ognuna di lunghezza casuale. I casi di test sono stati generati con uno script python e scritti su file di testo, poi letti a runtime da una testbench apposita. Ogni test si è concluso con successo.

I precedenti test sono sufficienti a coprire ogni diverso comportamento desiderato dalla macchina secondo le specifiche, inoltre, essi portano la macchina in ogni stato e attraverso ogni transizione che essa possiede.

NOTA: Tutti i test qui riportati sono stati svolti solo nelle condizioni permesse dalla specifica, non vi sono dunque test riguardo situazioni impossibili rispetto a quest'ultima, come ad esempio l'abbassarsi del segnale `i_start` durante la fase operativa del circuito, prima che `o_done` venga alzato. Quei casi eccezionali sono stati considerati nel progetto, ma portano comunque a comportamenti del

componente diversi caso per caso. Ad esempio, nel caso precedentemente citato, il modulo continua a operare come se `i_start` fosse alto, fermandosi solo per un ciclo in `DONE` a fine elaborazione e tornando direttamene in `STAND_BY`.

4 Risultati della sintesi

L'FPGA usata per sintesi e implementazione è l'Artix-7 FPGA xc7a200tfbg484-1, come suggerito nella specifica.

La sintesi porta ad un design che utilizza 51 registri come flip flop e 67 LTU come porte logiche, ogni registro è sincrono col clock e resettabile asincronamente.

Output di "report utilization" e "report timing" per un clock di 100ns:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	67	0	134600	0.05
LUT as Logic	67	0	134600	0.05
LUT as Memory	0	0	46200	0.00
Slice Registers	51	0	269200	0.02
Register as Flip Flop	51	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Location	Delay type	Incr(ns)	Path(ns)		Netlist Resource(s)
	(clock clock rise edge)	0.000	0.000	r	
		0.000	0.000	r	i_clk (IN)
	net (fo=0)	0.000	0.000		i_clk
	IBUF (Prop_ibuf_I_O)	0.944	0.944	r	i_clk_IBUF_inst/O
	net (fo=1, unplaced)	0.800	1.744		i_clk_IBUF
	BUFG (Prop_bufg_I_O)	0.096	1.840	r	i_clk_IBUF_BUFG_inst/O
	net (fo=51, unplaced)	0.584	2.424		i_clk_IBUF_BUFG
	FDCE			r	o_address_reg[0]/C
	FDCE (Prop_fdce_C_Q)	0.456	2.880	r	o_address_reg[0]/Q
	net (fo=1, unplaced)	0.800	3.680		o_address_OBUF[0]
	OBUF (Prop_obuf_I_O)	2.782	6.461	r	o_address_OBUF[0]_inst/O
	net (fo=0)	0.000	6.461		o_address[0]
				r	o_address[0] (OUT)
	(clock clock rise edge)	100.000	100.000	r	
	clock pessimism	0.000	100.000		
	clock uncertainty	-0.035	99.965		
	output delay	-0.000	99.965		
	required time		99.965		
	arrival time		-6.461		
	slack		93.503		

Un indicatore di un corretto design post-sintesi è il "Worst Negative Slack" calcolato in questo caso per un clock di 100ns, esso rappresenta la parte di periodo di clock rimasta dopo che il segnale più lento ha raggiunto la sua destinazione. Il segnale più in ritardo nel design arriva dunque a destinazione dopo 6.461ns rispetto al fronte di salita del clock.

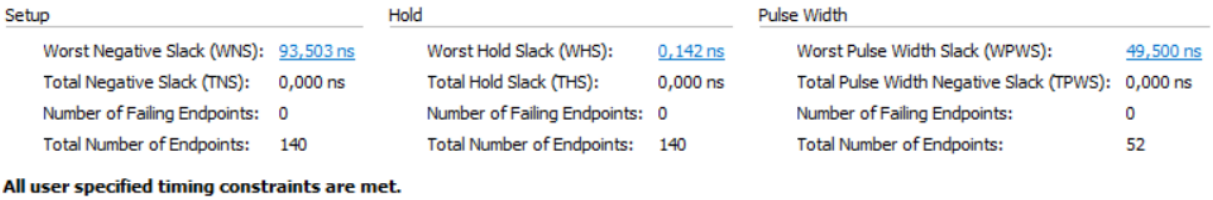


Figura 6 - Timing report summary

5 Conclusioni

Il componente realizzato rispetta completamente la specifica ed è in grado di venire utilizzato anche a periodi di clock significativamente minori di quello richiesto (10ns). Come mostrato dai test passati con successo, ogni possibile stato e transizione si comporta come desiderato.

Il numero degli stati della macchina è minimizzato, come precedentemente detto, poiché il ritardo in lettura dalla memoria richiede 2 stati per ogni lettura. Allo stesso modo, per soddisfare la specifica, in particolar modo l'interfaccia con l'esterno, servono gli stati di `STAND_BY` e `DONE`. Questo rende 8 stati necessari.

Il modulo risulta correttamente sintetizzabile ed anche in post sintesi mostra di passare ogni test nelle simulazioni.