Documentazione progetto RL 2022

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

## Specifiche di progetto - introduzione

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 alzerà done, solo dopo che done è 1, start verrà abbassato ed a seguito di ciò anche done dovrà essere abbassato. Tornati in questa configurazione potrà ripetersi il tutto, ma senza l’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 Uk il k-esimo bit ingresso e siano P1k e P2k il due bit prodotti da esso, allora:

P1k = Uk xor Uk-2

P2k = Uk xor Uk-1 xor Uk-2

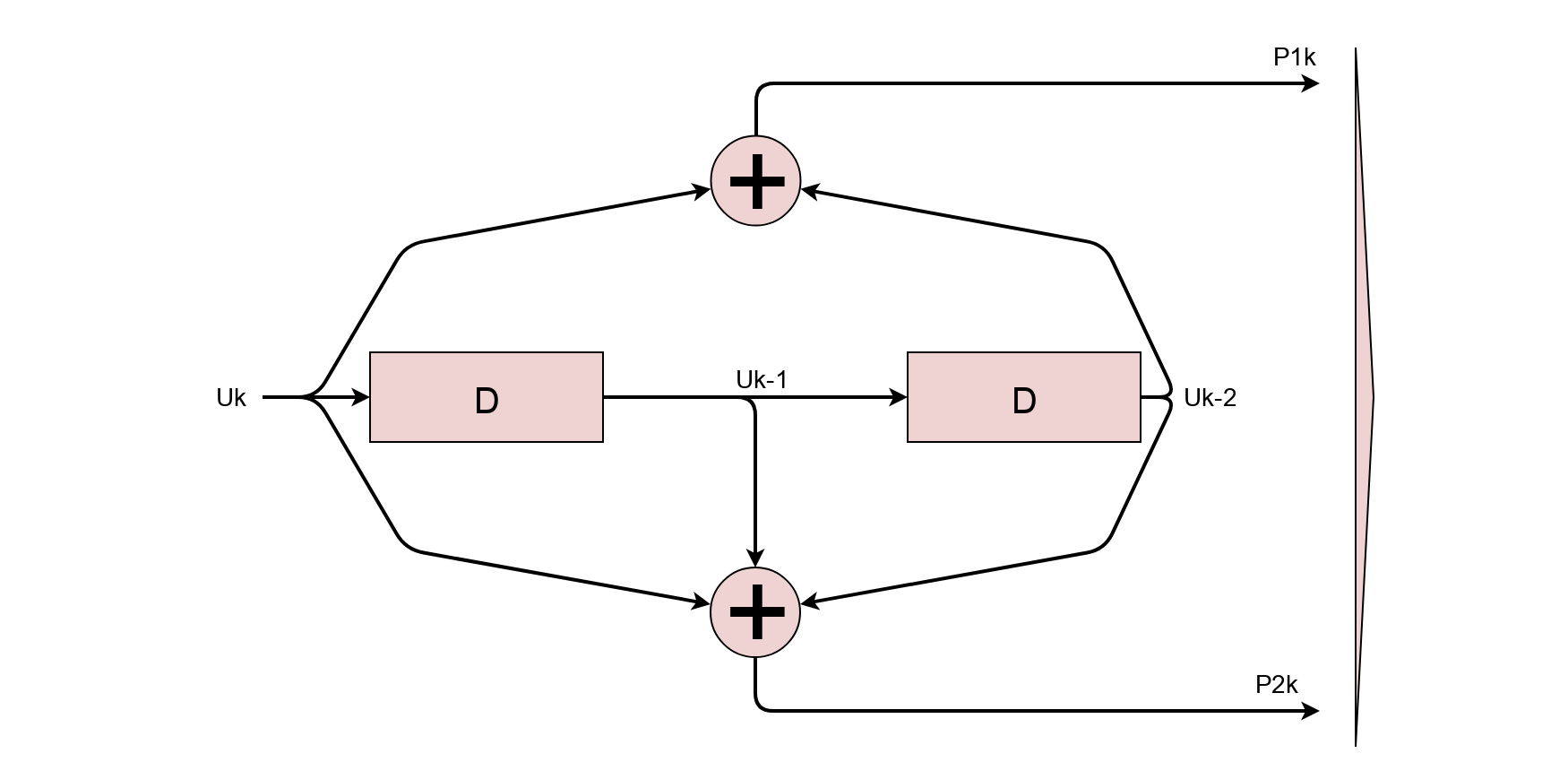


Figura - rappresentazione del convolutore

Una rappresentazione del convolutore è la seguente:

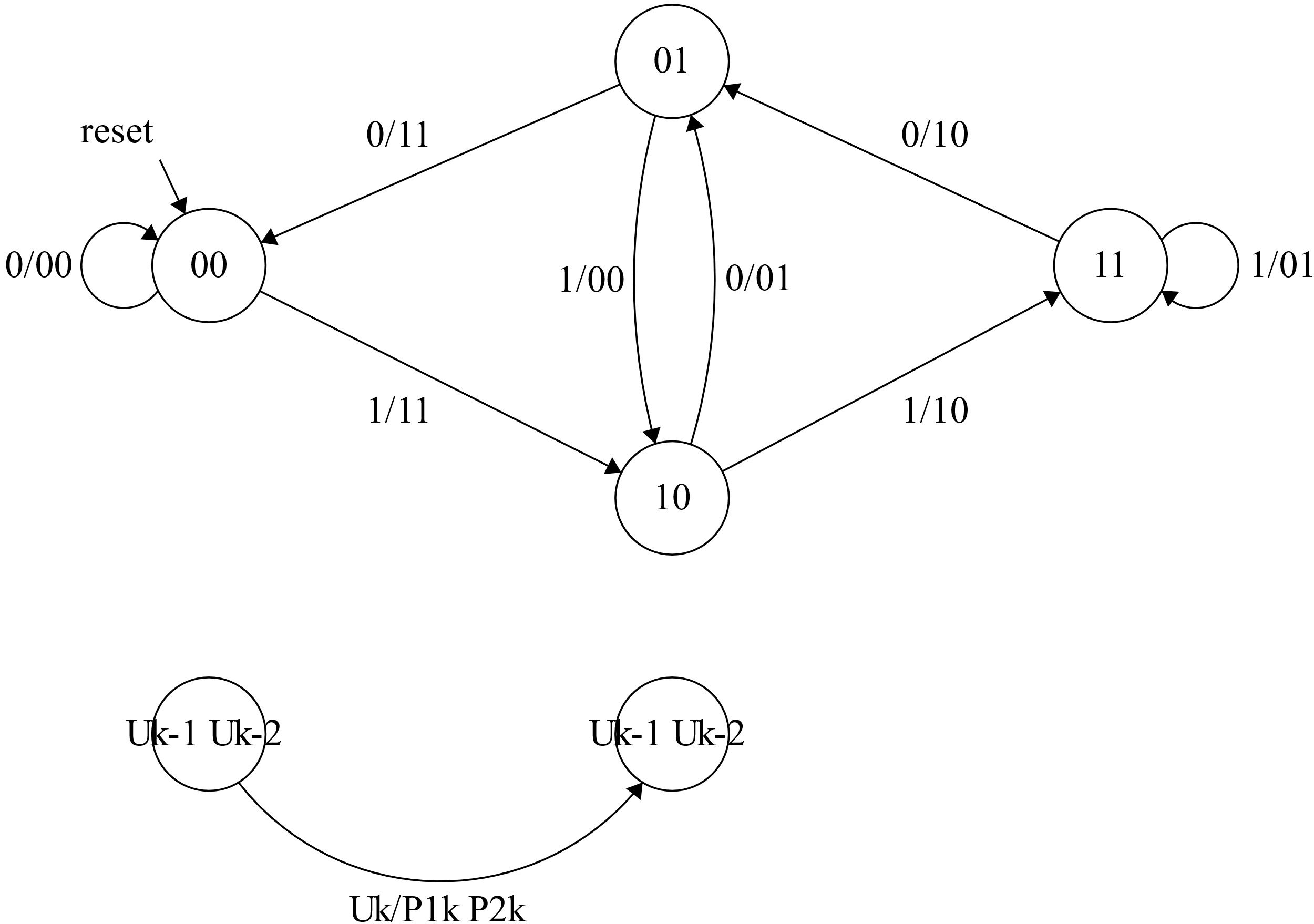
 Questa operazione può essere svolta da una FSM (finite-state machine) come la seguente:

Figura - esempio di FSM svolgente la convoluzione 1/2

In conclusione un esempio di elaborazione, dato in ingresso il byte 10100010 con i due bit precedenti inizializzati a 0:

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.

## Scelte progettuali

Il design si costituisce di un singolo modulo contenente un processo e 5 registri interni, dei quali uno contenente lo stato corrente della macchina. 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 dunque una FSM(D) (finite-state machine with datapath), che è la seguente:

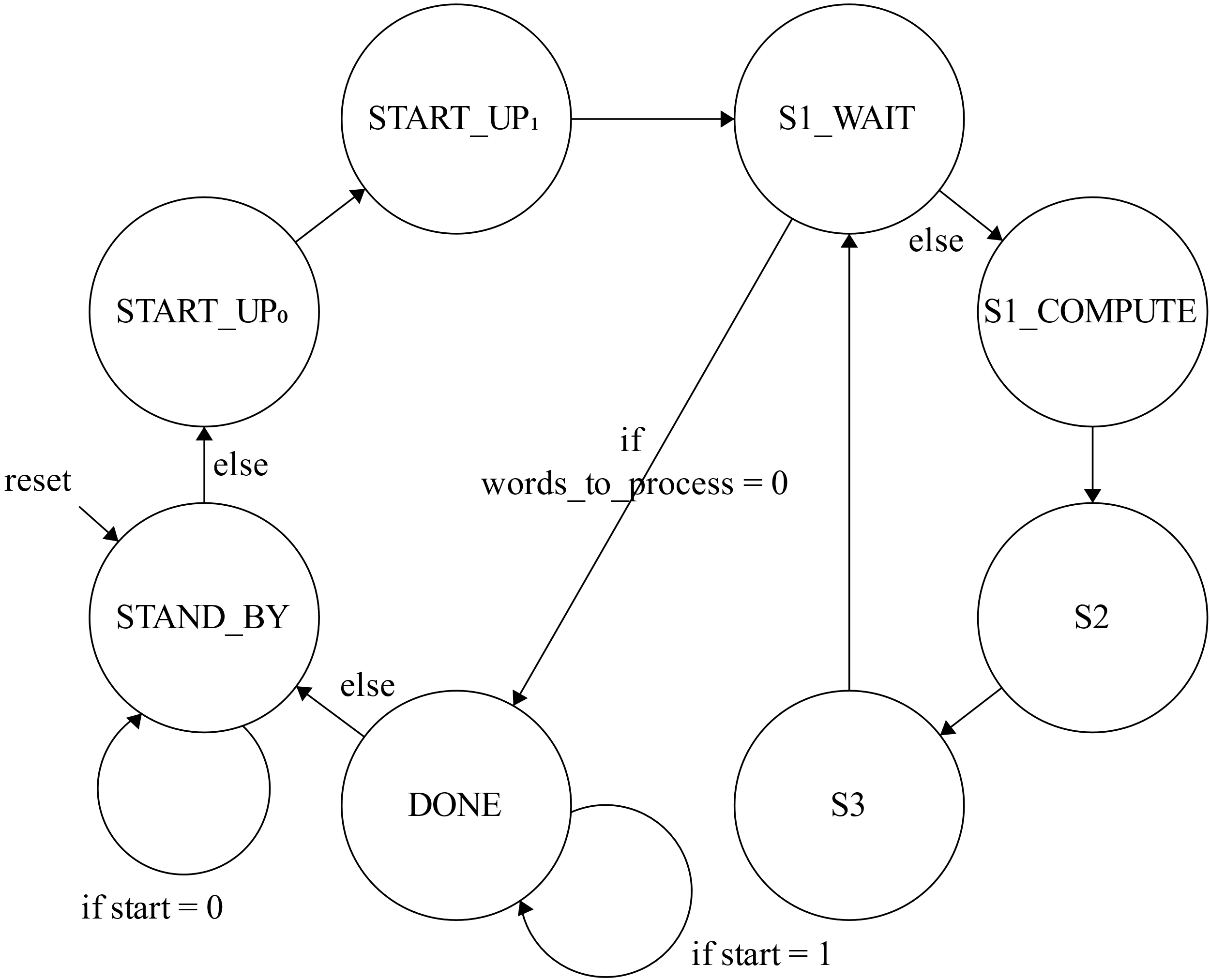
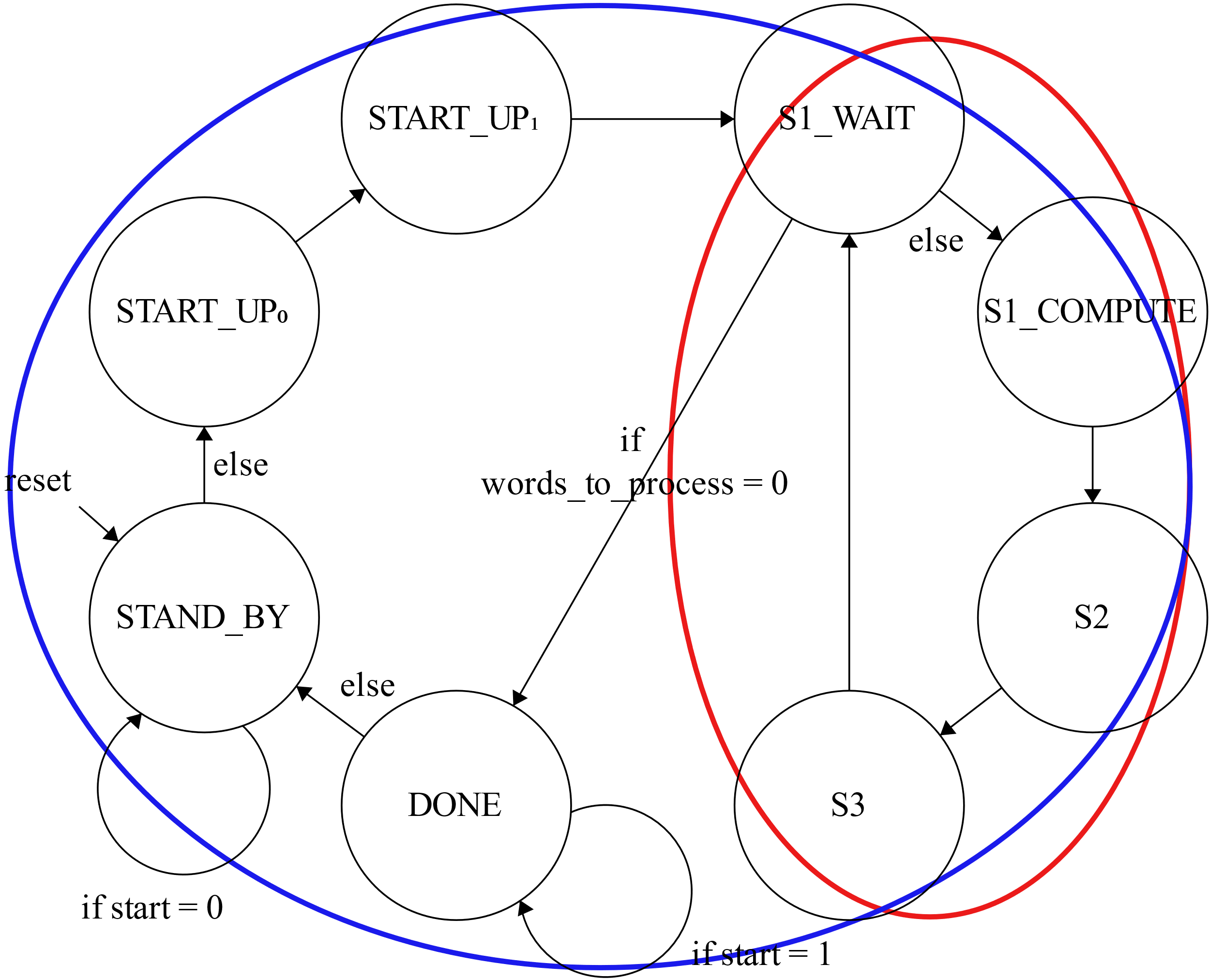
<IMMAGINE E DESCRIZIONE FSM(D)>

Figura - 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 | | |

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.
* La FSM del modulo si compone di due “anelli” di stati, uno interno ed uno esterno. 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 di un byte letto dalla memoria per ogni iterazione dell’anello, mentre l’anello esterno viene percorso dopo un reset o tra diverse fasi di lavoro sull’anello interno, poiché esso gestisce l’interfaccia dei segnali start, done, reset e memory enable.

<IMMAGINE FSM CON ANELLI EVIDENZIATI>

* 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, ovvero una per byte.

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 <àncora>) e ciò consente di risolvere le due scritture in soli 2 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.

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 atto a memorizzare e trasportare 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 se stesso e 998 o 999 per produrre gli indirizzi di memoria ove scrivere.

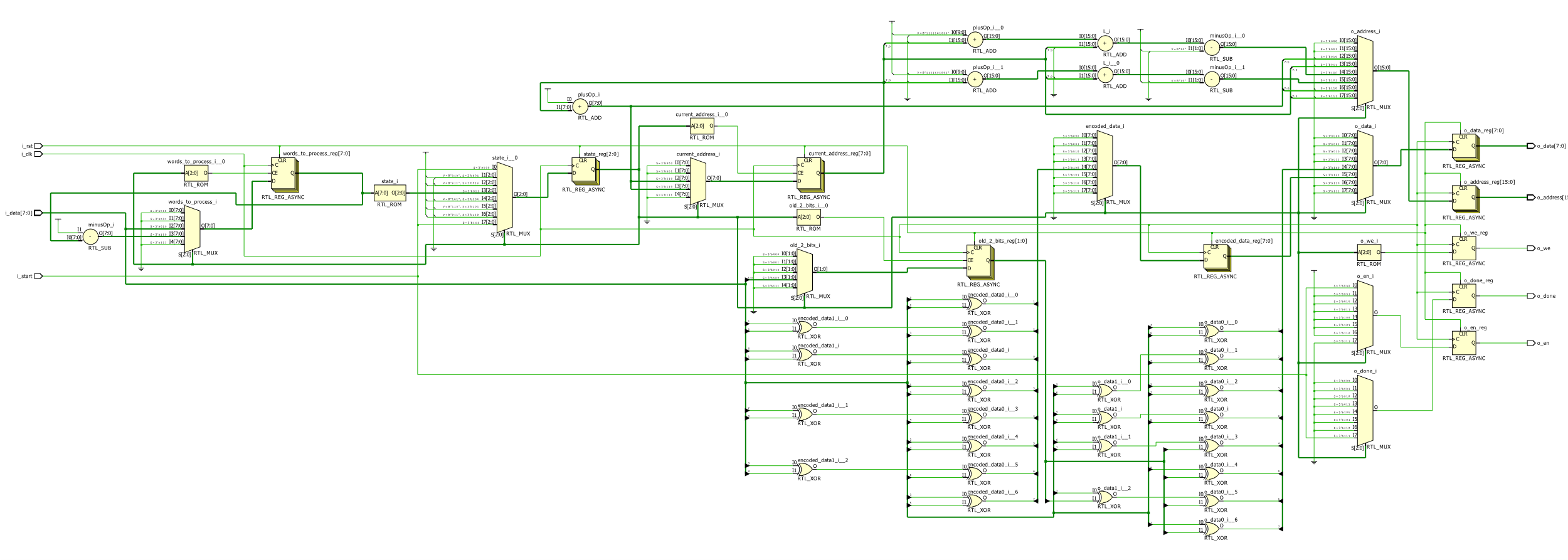
* signal words\_to\_process : std\_logic\_vector (7 downto 0);

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. Quanto il suo valore raggiunge 0 la macchina esce dall’anello interno e passa allo stato di DONE.

STATI:

<BLA BLA SU OGNI STATO DELLA MACCHINA>

<OGNI REGISTRO IL CUI VALORE NON È DIVERSAMENTE SPECIFICATO NEI VARI STATI SI ASSUMA SIA IMPOSTATO A TUTTI 0>

In conclusione questa è la schematic del desing:

## Risultati dei test fatti e le ragioni di tali test - motivare le scelte –

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 pressoché la totalità delle situazioni limite. Seguono le brevi descrizioni dei test con i rispettivi esiti delle post-synthesis functional simulations effettuate con Vivado 2016.X.X. 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.

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

TESTS:

* Il primo test svolto è stata la testbench fornita insieme alle specifiche, che termina correttamente.

<FOTO TESTBENCH D’ESEMPIO>

* Test con numero di parole da processare pari a zero.
* Test con numero di parole da processare massimo, 255.
* Test di tre sequenze diverse da processare una dopo l’altra, senza reset in mezzo. (sequenze corte)
* Test di tre sequenze diverse da processare una dopo l’altra, senza reset in mezzo. (sequenze max)
* Test di reset asincrono durante la fase operativa e successivo completamento di una diversa sequenza da processare.
* Test di reset asincrono durante la fase operativa e successivo completamento della precedente sequenza da processare.
* Test con ritardo nullo della memoria e ritardo aumentato a 5ns della memoria.

<BLA BLA SCOPO DI OGNI TEST>

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 test, poi letti dalla testbench tramite le librerie ieee.std\_logic\_textio e STD.textio. Ogni test si è concluso con successo.

NOTA: Tutti i test qui riporati sono stati svolti solo nelle condizioni permesse dalle specifiche, quindi non vi sono 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.

## Risultati della sintesi

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.

<report\_utilization, report\_timing>