Progetto di Reti Logiche

Leonardo Evi | 10800288 | a.a. 2023-2024

1. Introduzione

Il componente da realizzare deve interfacciarsi con una memoria ed operare su di una sequenza di coppie di interi come descritto di seguito. Per ogni coppia, il secondo valore è da interpretarsi come un indice di “credibilità” o “affidabilità” del primo.

Ricevuti come input la lunghezza di tale sequenza e l’indirizzo di memoria da cui partire (la sequenza è memorizzata su celle adiacenti), il componente deve imporre il valore di credibilità a 31, per ogni valore “valido” (ovvero diverso da zero) e sostituire quelli non validi con l’ultimo valore valido incontrato. Il valore di credibilità dei sostituti dovrà decrescere volta per volta di una unità, senza mai andare sotto zero e tornando a 31 solo dopo aver incontrato un valore “valido”.

Risulta importante notare come tutti i valori non validi posti all’inizio della sequenza non possano essere sostituiti, essi dunque non devono essere cambiati, e il relativo valore di credibilità deve essere posto a zero.

La memoria è indirizzata al byte. Ogni valore (di credibilità e non) è rappresentato come un intero senza segno su 8 bit.

Di seguito alcuni esempi di sequenze pre e post elaborazione da parte del componente.

[**0**, 0, **0**, 0, **72**, 0, **198**, 0, **50**, 0, **65**, 0, **0**, 0, **0**, 0, **0**, 0]
[
0, 0, **0**, 0, **72**, 31, **198**, 31, **50**, 31, **65**, 31, **65**, 30, **65**, 29, **65**, 28]

—

[**8**, 0, **0**, 0, **0**, 0, **0**, 0, **0**, 0, **0**, 0, **0**, 0, **0**, 0, **0**, 0]
[
8, 31, **8**, 30, **8**, 29, **8**, 28, **8**, 27, **8**, 26, **8**, 25, **8**, 24, **8**, 23]

Nonché il codice python che le genera, in accordo con la specifica fornita.

```
from numpy import random


def generate_sequence():
    max_len = 1024
    max_val = 2**8

    # randomly generating parameters
    leading_zeros = random.randint(max_len)
    trailing_zeros = random.randint(max_len - leading_zeros)
    remaining_vals = random.randint(max_len - leading_zeros - trailing_zeros)

    # for good measure
    if leading_zeros + trailing_zeros + remaining_vals > max_len:
        print("SEQUENCE TOO LONG")
        return []

    sequence = []
    # fill in the sequence
    sequence += [0, 0]*leading_zeros
    for _ in range(remaining_vals):
        sequence.append(random.randint(max_val - 1))
```

```
        sequence.append(0)
    sequence += [0, 0]*trailing_zeros

    return sequence
```

```
def component_simulator(sequence):
    i = 0
    last_valid_value = 0
    credibility = 0

    # skipping initial zeros and setting credibility to 0
    while i < len(sequence) and sequence[i] == 0:
        sequence[i+1] = 0
        i += 2

    while i < len(sequence):
        if sequence[i] != 0:
                        # valid value
            last_valid_value = sequence[i]  # update last valid value
            credibility = 31                # reset credibility value
        else:
                        # invalid value
            sequence[i] = last_valid_value  # replace invalid value

        sequence[i+1] = credibility         # set credibility
        if credibility > 0:
            credibility -= 1                # decrease credibility for next value

                i += 2                              # go to next value

    # print(sequence)
    return sequence
```

2. Architettura

Il componente si serve di cinque moduli principali:

- **Sequence Left**

  memorizza la lunghezza della sequenza ancora da analizzare.

- **Memory Address**

  tiene traccia dell’indirizzo di memoria a cui accedere

- **Last Valid Value**

  memorizza l’ultimo valore valido incontrato

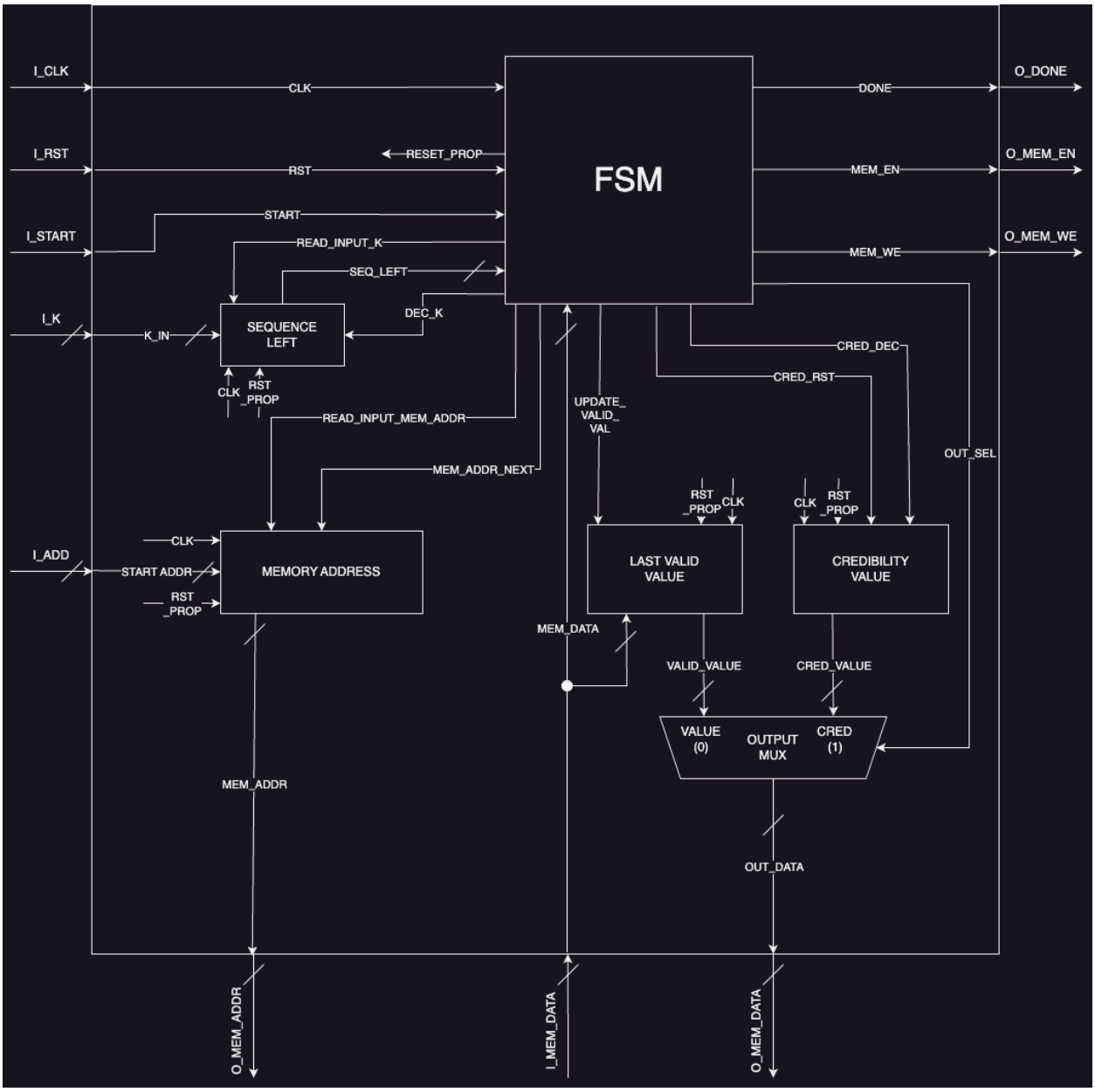
- **Credibility Value**

  tiene traccia dell’ultimo valore di credibilità impostato

- **FSM**

  gestisce lo stato di avanzamento del processo e coordina le operazioni di accesso alla memoria

Di seguito un diagramma che rappresenta tali componenti e i segnali che li connettono.



Ovviamente tutti i moduli presentati ricevono in input il segnale di clock (CLK), lo stesso che proviene dall'esterno del componente, e il segnale di reset (RST_PROP) che viene propagato dalla FSM quando questa riceve dall'esterno il segnale di RST.

I collegamenti di tali segnali sono stati omessi per non appesantire il diagramma, ma sono facilmente deducibili.

Tutti i segnali sono sincroni, quelli esterni vengono interpretati sul fronte di salita del clock, quelli interni sul fronte di discesa. Questa scelta permette alla macchina di vedere il risultato dei comandi imposti ai vari componenti entro il successivo fronte di salita, velocizzando l'intero processo ed evitando di dover aggiungere degli stati di semplice attesa.

I segnali di reset sono invece asincroni, portano la macchina nello stato iniziale e i valori memorizzati dai vari moduli a zero.

Segue una semplice spiegazione dei singoli moduli e del loro funzionamento.

Modulo Sequence Left

Questo modulo è caratterizzato da **due** elementi di memoria.

Il **primo** memorizza il valore in ingresso K_IN ad ogni fronte di salita del clock.

Il **secondo** assume il valore memorizzato dal primo sul fronte di discesa del clock solo quando il segnale READ_INPUT_K, proveniente dalla FSM, è alto (ovvero quando il valore presente nel primo è valido).

Questa scelta può sembrare una eccessiva complicazione dal momento che il componente potrebbe semplicemente memorizzare il segnale in input. Tuttavia, per quanto già detto, è necessario che i componenti interni operino sul fronte di discesa del clock, momento in cui, secondo la specifica, i valori in input (I_K o I_ADD) potrebbero non avere senso.

Il segnale di output SEQ_LEFT porta l'informazione memorizzata (corrispondente alla lunghezza della sequenza ancora da analizzare) dal secondo **elemento** di memoria verso la **FSM**.

Il segnale di input DEC_K, proveniente dalla FSM, porta il valore SEQ_LEFT a decrementare di una unità sul fronte di discesa del clock, quando posto a '1'.

Modulo Memory Address

Questo modulo è caratterizzato da due elementi di memoria.

Il primo memorizza il valore in ingresso START_ADDR ad ogni fronte di salita del clock.

Il secondo assume il valore memorizzato dal primo sul fronte di discesa del clock solo quando il segnale READ_INPUT_MEM_ADDR, proveniente dalla FSM, è alto (ovvero quando il valore presente nel primo è valido).

Anche per questo modulo valgono le stesse considerazioni fatte per il precedente.

Il segnale di output MEM_ADDR è direttamente connesso alla memoria e corrisponde all'indirizzo memorizzato sul quale si vuole operare.

Il segnale di input MEM_ADDR_NEXT, proveniente dalla FSM, porta il valore memorizzato ad aumentare di una unità sul fronte di discesa del clock, quando posto a '1'.

Modulo Last Valid Value

Questo modulo è caratterizzato da due elementi di memoria.

Il primo memorizza il valore in ingresso MEM_DATA ad ogni fronte di salita del clock.

Il secondo assume il valore memorizzato dal primo sul fronte di discesa del clock solo quando il segnale UPDATE_VALID_VAL, proveniente dalla FSM, è alto (ovvero quando il valore presente nel primo è valido).

Anche per questo modulo valgono le stesse considerazioni fatte per il precedente.

Il segnale di output VALID_VALUE contiene l'informazione memorizzata (corrispondente all'ultimo valore valido incontrato).

Modulo Credibility Value

Questo modulo memorizza l'ultimo valore di credibilità impostato.

Il segnale di input CRED_RST impone, quando posto a '1', il valore memorizzato a "31" sul fronte di discesa del clock.

Il segnale di input CRED_DEC, quando posto a '1', porta il valore memorizzato a diminuire di una unità sul fronte di discesa del clock, non ha alcun effetto se il valore memorizzato è zero.

Il segnale di output CRED_VALUE contiene l'informazione memorizzata (corrispondente all'ultimo valore di credibilità impostato).

Modulo Output Mux

Questo modulo permette alla FSM di selezionare quale tra i valori di VALID_VALUE e CRED_VALUE debba andare in output (OUT_DATA) per essere scritto in memoria, tramite il segnale di scelta OUT_SEL ('0' per VALID_VALUE, '1' per CRED_VALUE)

Tra tutti, è l'unico modulo puramente combinatorio.

Modulo FSM

Questo modulo gestisce le operazioni da eseguire sui componenti interni e sulla memoria grazie ai vari segnali di output.

Si tratta di una **FSM di Moore**, le cui uscite dipendono solo dallo stato in cui si trova.

È ora importante soffermarsi sulla sequenza di eventi che caratterizzano l'esecuzione di una determinata istruzione:

- la macchina giunge in un nuovo stato ad ogni fronte di salita del clock
- come conseguenza i segnali di output della FSM possono variare (es. DEC_K = '1')
- poco dopo il fronte di discesa del clock "le istruzioni vengono eseguite" (es. SEQ_LEFT passa da "1" a "0")
- in questo modo, entro il successivo fronte di salita, la macchina ha tutte le informazioni necessarie per effettuare la corretta transizione (es. porsi nello stato di SEQUENCE_COMPLETED)

Fatta questa doverosa premessa, è possibile elencare i vari segnali di input, i diversi stati della macchina e i corrispondenti segnali di output che vengono posti a '1', gli altri sono implicitamente posti a '0'.

Segnali di input:

- CLK

  segnale di clock, la macchina esegue una transizione sul suo fronte di salita.
- RST

  segnale di reset, asincrono, inizializza la FSM.
- START

  segnale di start, proviene dall'esterno del componente, permette alla FSM di cominciare l'elaborazione della sequenza.
- MEM_DATA

  segnale che porta l'informazione proveniente dalla memoria, su 8 bit, interpretato come un intero senza segno.

  Proviene dal modulo MEMORY ADDRESS.
- SEQ_LEFT

  segnale che indica la lunghezza della sequenza che resta da analizzare, su 10 bit, interpretato come un intero senza segno.

  Proviene dal modulo SEQUENCE LEFT.

Segnali di output:

- RESET_PROP

  propaga il segnale di reset al resto dei moduli, inizializzandoli.
- DONE

  indica all'esterno del componente che l'elaborazione è terminata.
- MEM_EN

  diretto verso la memoria esterna, abilita le operazioni di lettura/scrittura.
- MEM_WE

  diretto verso la memoria esterna, indica se l'operazione da svolgere è di lettura (quando basso) o di scrittura (quando alto)

Segnali di output, per il controllo del modulo SEQUENCE LEFT:

- READ_INPUT_K
- DEC_K

Segnali di output, per il controllo del modulo MEMORY ADDRESS:

- READ_INPUT_MEM_ADDR

- MEM_ADDR_NEXT

Segnale di output, per il controllo del modulo LAST VALID VALUE:

- UPDATE_VALID_VAL

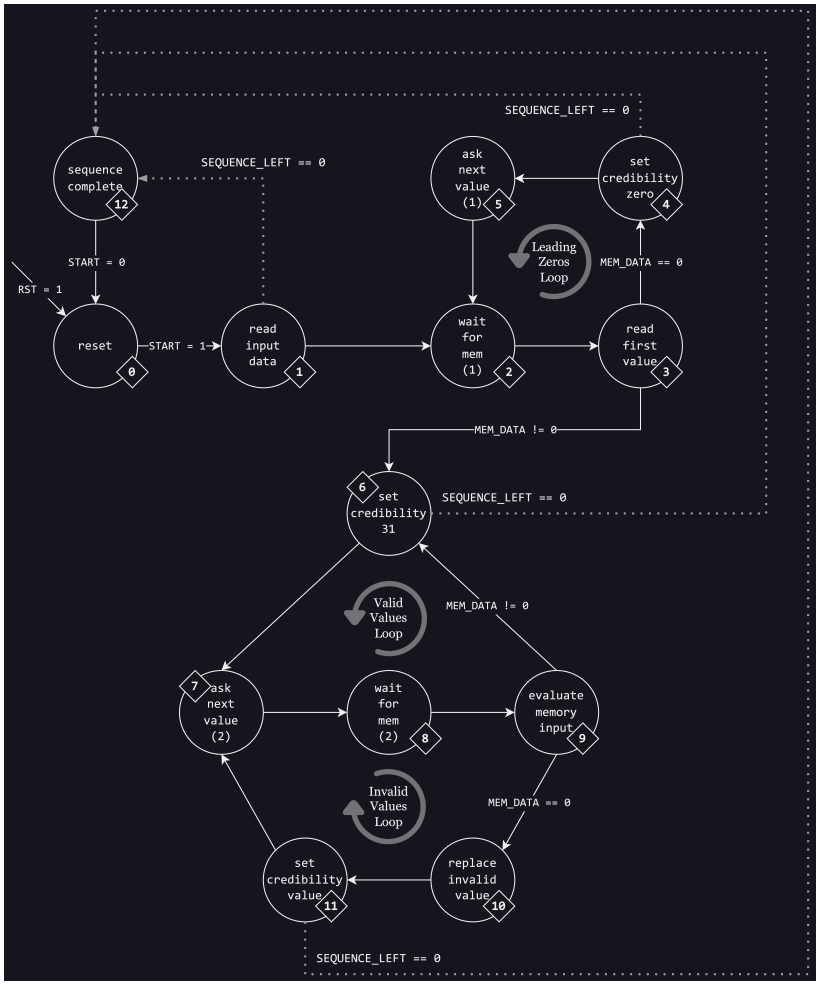
Segnali di output, per il controllo del modulo CREDIBILITY VALUE:

- CRED_RST
- CRED_DEC

Segnale di output, per il controllo del modulo OUTPUT MUX:

- OUT_SEL

Diagramma degli stati:



Dovrebbe essere evidenti le analogie tra il diagramma e il codice python riportato all'inizio.

Il primo while è implementato dal ciclo "leading zeros loop", il secondo dai due cicli "valid" e "invalid" "values loop" che corrispondono ai due rami dell'if-else.

Si noti come le transizioni rappresentate da un tratto discontinuo (per semplici motivi di leggibilità del grafico) sono transizioni a tutti gli effetti e, per come la FSM è stata implementata, hanno precedenza sulle altre. Ovvero quando la sequenza da analizzare è finita, e la macchina si trova in uno tra gli stati "1", "4" e "11", il successivo stato sarà il "12".

Segue una descrizione degli stati, e dei corrispondenti valori di output alti.

▼ (0) Reset

- RESET_PROP

  In questo stato la macchina è in attesa del segnale di START, propaga inoltre il segnale di reset al resto dei moduli.

▼ (1) Read Input Data

- READ_INPUT_K
- READ_INPUT_MEM_ADDR
- MEM_EN

  La macchina impone ai moduli dedicati di memorizzare le informazioni fornite in input, ossia la lunghezza e l'indirizzo di inizio della sequenza.

  Da notare come quest'ultimo venga immediatamente propagato alla memoria che, essendo abilitata in lettura, ne fornirà il primo valore.

▼ (2) e (8) Wait For Mem

Nessun segnale alto, si attende che la memoria fornisca i dati richiesti.

▼ (3) Read First Value

- UPDATE_VALID_VAL

  In questo stato la macchina ha le informazione necessarie (dato in input della memoria) per decidere che strada percorrere. Il segnale viene alzato per assicurarsi che, nel caso il valore sia diverso da zero, questo venga memorizzato dall'apposito modulo.

▼ (4) Set Credibility Zero

- MEM_ADDR_NEXT
- DEC_K
- MEM_EN
- MEM_WE

  È stato letto un valore non valido all'inizio della sequenza.

  L'indirizzo viene incrementato e la memoria abilitata in scrittura per imporre il valore di credibilità a zero.

  La selezione dell'output è indifferente, entrambi i segnali sono posti a zero.

  Viene decrementata la lunghezza della sequenza da analizzare.

▼ (5) Ask Next Value 1

- MEM_ADDR_NEXT
- MEM_EN

  L'indirizzo da cui leggere viene incrementato e la memoria viene abilitata in lettura.

▼ (6) Set Credibility 31

- CRED_RST
- OUT_SEL
- MEM_ADDR_NEXT

- DEC_K
- MEM_EN
- MEM_WE

  È stato letto un valore valido.

  La memoria viene abilitata in scrittura sul successivo indirizzo.

  Il valore di credibilità memorizzato viene reimpostato a “31”.

  Il valore selezionato per essere scritto è quello di credibilità.

  Viene decrementata la lunghezza della sequenza da analizzare.

▼ (7) Ask Next Value 2

- CCRED_DEC
- MEM_ADDR_NEXT
- MEM_EN

  L’indirizzo da cui leggere viene incrementato e la memoria viene abilitata in lettura.

  Il valore di credibilità memorizzato, da impostare per il prossimo valore valido, viene decrementato.

▼ (9) Evaluate Memory Input

- UPDATE_VALID_VAL

  L’ultimo valore valido viene aggiornato (se diverso da zero).

  La macchina deve decidere quale transizione seguire, in base a tale valore.

▼ (10) Replace Invalid Value

- MEM_EN
- MEM_WE

  È stato letto un valore non valido.

  La memoria viene abilitata in scrittura per rimpiazzare il precedente valore non valido.

  La selezione dell’output posta a ‘0’ permette di scrivere in memoria l’ultimo valore valido incontrato.

▼ (11) Set Credibility Value

- OUT_SEL
- MEM_ADDR_NEXT
- DEC_K
- MEM_EN
- MEM_WE

  La memoria viene abilitata in scrittura sul successivo indirizzo.

  Il valore selezionato per essere scritto è quello di credibilità.

  Viene decrementata la lunghezza della sequenza da analizzare.

▼ (12) Sequence Complete

- DONE

Sequenza terminata, la FSM comunica all’esterno tale informazione e attende che il segnale di START torni a zero.

3. Risultati Sperimentali

Sintesi

La sintesi del componente richiede l’impiego di 77 Flip Flop e non genera nessun latch.

Ecco il report di utilizzo (in versione sintetica) generato da Vivado:

```
+-------------------------+------+
|        Site Type        | Used |
+-------------------------+------+
| Slice Registers         |   77 |
|   Register as Flip Flop |   77 |
|   Register as Latch     |    0 |
+-------------------------+------+
```

Simulazioni

Il componente è stato testato post-sintesi su alcune sequenze che costituiscono casi limite rispetto alla specifica e su altre generate casualmente dal codice sopra riportato.

Di seguito, i casi limite testati:

- Sequenza di lunghezza 0

  Questo test fornisce un valore in input I_K pari a zero. Serve ad assicurarsi che il componente non abbia problemi a gestire sequenze di questo tipo, e che non modifichi erroneamente alcun valore in memoria.

  (Transizione SEQ_LEFT == 0 a partire dallo stato “read input data”)

- Sequenza composta da soli valori non validi

  Questo test verifica che la macchina completi l’elaborazione della sequenza, riportandosi nello stato di reset, nonostante non abbia incontrato alcun valore valido.

  (Transizione SEQ_LEFT == 0 a partire dallo stato “set credibility zero”)

- Sequenza con molti valori non validi in testa

  Questo test verifica che la macchina gestisca correttamente la presenza di molti valori non validi in testa alla sequenza, per poi elaborarne correttamente il resto.

- Sequenza con molti valori non validi, preceduti da dei valori validi

  Questo test verifica che il valore di credibilità impostato per la lunga sequenza di valori non validi sia zero (dal trentunesimo in poi) e non vada sotto zero (tornando a 31).

- Sequenza di lunghezza massima

  Questo test, effettuato su diverse sequenze di lunghezza massima, verifica la corretta funzionalità del componente SEQ_LEFT anche nel caso in cui sia inizializzato al valore massimo.

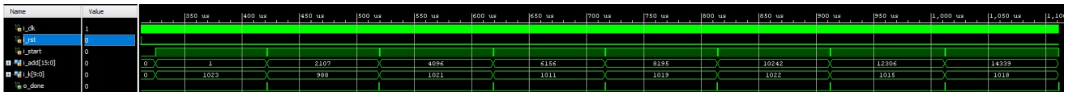
- Sequenza interrotta dal segnale di reset

Questo test verifica che il componente, dopo essere stato interrotto del segnale di reset, sia in grado di riprendere l'elaborazione della sequenza successiva.

Sono stati inoltre testati alcuni casi costituiti da numerose sequenze, alcune delle quali corrispondenti ai casi limite di cui sopra, sottoposte in serie al componente.

In certi casi si è trattato di 32 sequenze di lunghezza 1024. 32*1024*2 = 65536.In questi casi il componente ha elaborato correttamente l'intera memoria messagli a disposizione.

Senza il bisogno di alzare il segnale di reset tra una sequenza e l'altra, il componente le ha correttamente elaborate.



Il codice che genera tali test è disponibile sul mio profilo github: https://github.com/leonardoevi

Il requisito di tempo è stato soddisfatto.

```
Timing Report

Slack (MET) :             2.857ns  (required time - arrival time)
```

4. Conclusioni

Il componente è stato progettato con l'intento di evidenziarne la modularità, separando ogni modulo in base alla funzionalità svolta. Si è inoltre cercato di rendere quanto più semplice e chiara possibile l'architettura dalla macchina a stati, evitando di inserire un numero eccessivo di stati di attesa.

Così facendo è stato necessario complicare leggermente il funzionamento degli altri moduli, beneficiando tuttavia di un tempo di elaborazione inferiore.

Infatti per processare una generica sequenza, di lunghezza non nulla, la macchina impiega un numero di transizioni pari a:

3 (per raggiungere lo stato "read first value") +

4 * numero_di_zeri_iniziali (nzi, per compiere il "leading zeros loop") +

4 * numero_di_valori_validi (nvv, per compiere il "valid values loop") +

5 * numero_di_valori_non_validi (nvnv, per compiere l' "invalid values loop") -

1 (dovuto al fatto che, per tornare nello stato di reset, occorre una transizione in meno rispetto a completare uno qualunque dei tre cicli)

Semplificando:

2 + 4*(nzi + nvv) + 5*(nvnv) tranzizioni per sequenze di lunghezza > 0

3 transizioni altrimenti.

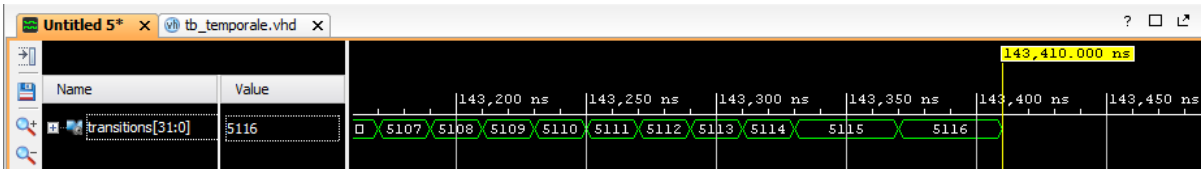
Considerando il **caso pessimo** in cui:

La sequenza ha lunghezza massima, 1023 valori (ed altrettanti di credibilità).

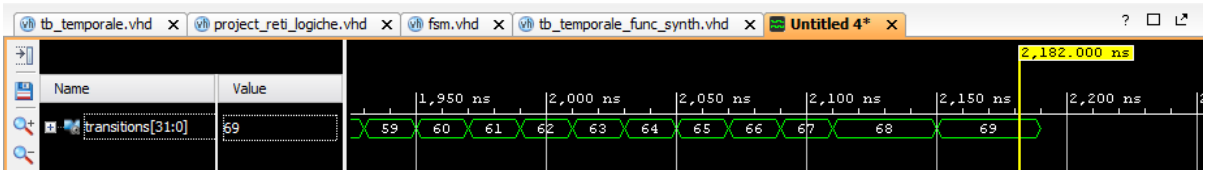
È costituita da un unico valore valido posto all'inizio.

Ovvero si ha nzi = 0, nvv = 1, nvnv = 1022

La macchina impiegherà 5116 transizioni per completare l'elaborazione della sequenza, corrispondenti a poco più di 0,1 milli secondi (dato che il periodo del clock è di 20 ns).



Nel caso in cui la sequenza abbia nzi = 5, nvv = 8, nvnv = 3, la macchina impiega 69 transizioni per riportarsi nello stato di reset.



Il segnale che memorizza il numero di transizioni è stato inserito solo in fase di debug e non ha nessuna utilità in termini funzionali.

Tutto questo ha un elevato costo in termini di elementi di memoria utilizzati. Si sarebbe infatti potuto sacrificare qualche transizione in più, al prezzo di uno o due flip flop per memorizzare il numero maggiore di stati della FSM, per ridurre della metà i flip flop impiegati nei tre moduli SEQUENCE_LEFT, MEMORY_ADDRESS e LAST_VALID_VALUE.

Dal momento che la specifica non imponeva limiti nè sul tempo di computazione totale, nè sul numero di elementi di memoria utilizzati, entrambe le scelte erano ugualmente percorribili.