Prova finale di Reti Logiche

A.A. 2023-24

Docente: Fabio Salice

Giuseppe Laguardia

Matricola: 984638

Codice Persona: 10773518

Lorenzo Meroi

Matricola: 984250

Codice Persona: 10788126

Indice

1. Introduzione

1.1 Requisiti del progetto

La Prova Finale (Progetto di Reti Logiche) dell'anno accademico 2023/24 richiede l'implementazione di un modulo hardware che si interfacci con una memoria e rispetti determinate indicazioni.

Il modulo deve leggere un messaggio costituito da una sequenza di *K* parole *W*. Ogni parola è memorizzata in due byte consecutivi di memoria, e può assumere valori compresi tra 255 e 0, dove quest'ultimo è da intendersi come *"il valore non è specificato"*. La sequenza di parole è memorizzata a partire da un indirizzo specificato *ADD*. Quindi le parole W si troveranno agli indirizzi: *ADD, ADD+2, ADD+4, …, ADD+2*(K-1)*.

Il byte mancante dovrà essere completato inserendo un valore di Credibilità *C* per ogni valore della sequenza. Quindi questi valori saranno inseriti agli indirizzi: *ADD+1, ADD+3, ADD+5, …, ADD+(2*(K-1))+1*.

Il modulo ha il compito di completare la sequenza seguendo queste direttive:

- Qualora il primo byte di una parola sia zero, questo dovrà essere sostituito con l'ultimo valore letto diverso da zero.
- Il valore di credibilità *C* deve essere posto a 31 ogni volta che il primo byte della parola corrispondente non è zero, altrimenti viene decrementato rispetto alla credibilità della parola precedente.
- Quando *C* raggiunge il valore zero non viene ulteriormente decrementato.
- Se il primo dato della sequenza è pari a zero, il suo valore non viene sostituito e la credibilità ad esso associato deve essere posta a zero. Lo stesso succede fino al primo valore della sequenza con valore diverso da zero.

Infine, tutti i segnali del modulo devono essere sincroni e devono essere interpretati sul fronte di salita del clock, avente periodo di 20 ns con duty cycle del 50%. L'unico segnale asincrono è il segnale di RESET.

1.2 Esempi

Forniamo qui sotto degli esempi volti a comprendere meglio il funzionamento del modulo. Sono evidenziati i valori W della sequenza.

<u>Esempio 1:</u>

- Sequenza di partenza:
  128 0 **64** 0 **0** 0 **0** 0 **0** 0 **0** 0 **0** 0 **100** 0 **1** 0 **0** 0 **5** 0 **23** 0 **200** 0 **0** 0
- Sequenza finale:
  128 31 **64** 31 **64** 30 **64** 39 **64** 28 **64** 27 **64** 26 **100** 31 **1** 31 **1** 30 **5** 31 **23** 31 **200** 31 **20**

Esempio 2:

- Sequenza di partenza:
  0 0 **0** 0 **0** 0 **103** 0 **0** 0 **0** 0 **83** 0 **1** 0
- Sequenza finale:
  0 0 **0** 0 **0** 0 **103** 31 **103** 30 **103** 29 **83** 31 **1** 31

Esempio 3:

- Sequenza di partenza:
  222 0 **34** 0 **189** 0 **95** 0 **71** 0 **127** 0 **0** 0 **33** 0 **228** 0 **29** 0 **245** 0
- Sequenza finale:
  222 31 **34** 31 **189** 31 **95** 31 **71** 31 **127** 31 **127** 30 **33** 31 **228** 31 **29** 31 **245** 31

1.3 Ipotesi Progettuali

Durante lo sviluppo del modulo si sono supposti i seguenti fatti:

- Tutti i byte nei quali doveva essere inserito il valore di credibilità C fossero uguali a zero.
- I valori W non superassero mai il valore di 255 e che la sequenza fosse ben composta.
- I segnali dati in ingresso dal test bench sono sempre ben inizializzati.
- Nel caso della computazione di più sequenze senza reinizializzazione della memoria, le sequenze prese in considerazione non si intersecano tra di loro.

2. Architettura

2.1 Architettura del progetto

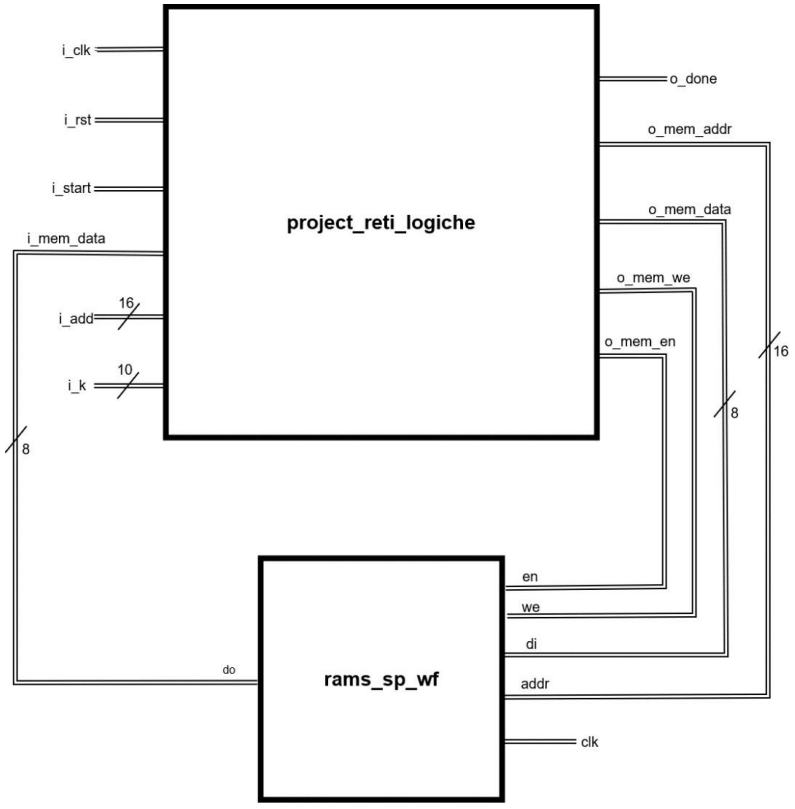
Il modulo hardware da implementare si interfaccia con una memoria e possiede sei ingressi:

- i_clk (1 bit): segnale di clock.
- i_rst (1 bit): segnale di reset asincrono.
- i_start (1 bit): segna quando il modulo dovrà partire nell'elaborazione della sequenza.
- i_add (16 bit): indica l'indirizzo di memoria dal quale parte la sequenza di parole.
- i_k (10 bit): indica il numero di parole nella sequenza da analizzare.
- i_mem_data (8 bit): arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura.

L'interfaccia del modulo comprende anche cinque segnali di uscita:

- o_done (1 bit): segnala la fine dell'elaborazione.
- o_mem_addr (16 bit): segnale indirizzato alla memoria che indica l'indirizzo al quale si vuole accedere.
- o_mem_data (8 bit): segnale indirizzato alla memoria che contiene il dato che verrà successivamente scritto.
- o_mem_we (1 bit): segnale di *Write Enable* da dover mandare alla memoria per poter scrivere. Per leggere deve essere posto a zero.
- o_mem_en (1 bit): segnale di *Enable* per poter comunicare con la memoria, sia in lettura che in scrittura.

Viene qui riportato uno schema che rappresenta l'architettura del progetto.



Prima del primo *START = 1* deve essere sempre dato il segnale di *RESET*, mentre una seconda elaborazione non dovrà aspettare il reset del modulo.

All'istante iniziale di reset, *DONE* deve essere a zero e rimarrà tale fino al termine della computazione, quando sarà portato a uno. Il segnale di *START* rimarrà alto fino a quando il segnale *DONE* verrà portato alto; quest'ultimo rimarrà in questo stato fino a quando quello di *START* non sarà portato a zero.

Una seconda elaborazione può essere richiesta dopo che il segnale di *DONE* sarà riportato a zero.

Quando il segnale di *START* viene posto a uno, gli ingressi *ADD* e *K* vengono inizializzati, e conservano il loro valore fintanto che *START* rimane alto.

2.2 Architettura dei componenti

Per la realizzazione del componente hardware (*project_reti_logiche*) si è optato per la realizzazione di due moduli che interagiscono tra loro:

- Macchina a stati finiti: si occupa dell'elaborazione e completamento della sequenza, gestendo anche i segnali diretti alla memoria. È stata realizzata con un'architettura behavioural.
- Contatore: gestisce l'offset di memoria da applicare ad *ADD*, controllando di non superare le *K* parole *W*. Dato che le parole si trovano solo sugli indirizzi di offset pari rispetto all'indirizzo iniziale, il contatore dovrà essere in grado di contare fino a $2*K_{MAX}$, rendendolo un contatore a 11 bit. Anche questo modulo è stato realizzato con architettura behavioural.

Di seguito sono riportati i segnali di ingresso e uscita della macchina a stati finiti (gli ingressi sono caratterizzati dal prefisso *i_* e le uscite da *o_*). Si è provato a mantenere le denominazioni dei segnali dell'architettura top level del progetto. Per la spiegazione di quei segnali si rimanda al paragrafo della suddetta architettura:

- i_clk (1 bit).
- i_rst (1 bit).
- i_start (1 bit).
- i_add (16 bit).
- i_k (10 bit).
- i_mem_data (8 bit).
- i_j (11 bit): segnale in arrivo dal contatore; indica l'offset di memoria dall'indirizzo iniziale *ADD*.
- o_done (1 bit).
- o_mem_addr (16 bit).
- o_mem_data (8 bit).
- o_mem_we (1 bit).
- o_mem_en (1 bit).
- o_ec (1 bit): segnale di *ENABLE COUNTER* diretto al contatore. Quando è posto ad uno, viene messo in *AND* con il clock, indica quando il modulo può incrementare il conteggio di uno.

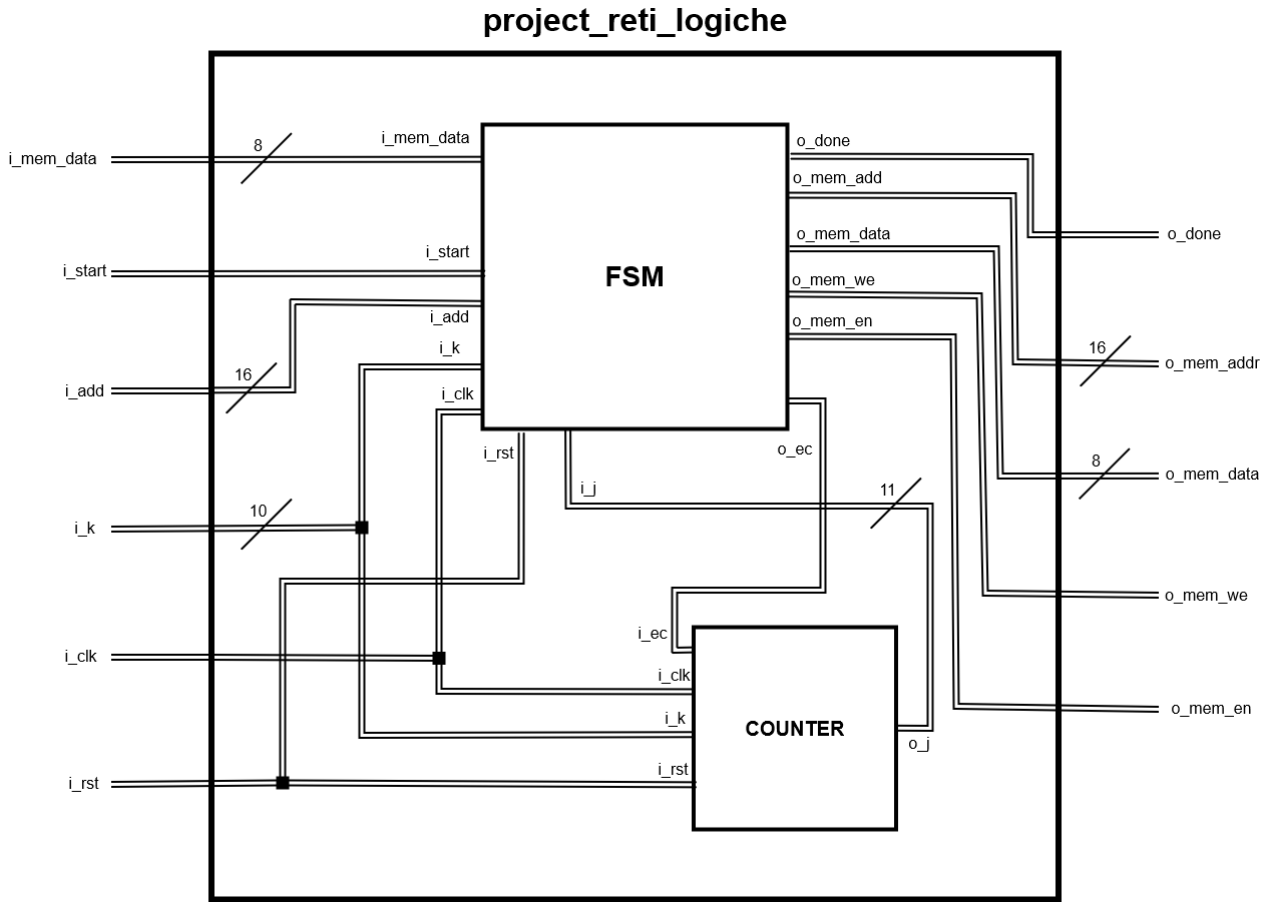
Ora vengono riportati i segnali relativi al contatore, si usano le stesse convenzioni usate per la macchina a stati finiti:

- i_clk (1 bit).
- i_rst (1 bit).
- i_k (10 bit).
- i_ec (1 bit): segnale che viene messo in *AND* con il clock indica quando il contatore può incrementare il conteggio di uno.
- o_j (11 bit): segnale diretto alla macchina a stati finiti, indica l'offset di memoria dall'indirizzo iniziale *ADD*.

Il contatore utilizza anche due segnali interni di supporto alla sua elaborazione:

- cnt (11 bit): segnale che indica lo stato attuale del contatore, *o_j* viene posto uguale a *cnt* alla fine del *process*.
- temp (11 bit): segnale che indica lo stato precedente del contatore; viene usato di supporto a *cnt* per calcolare il nuovo stato.

Viene qui riportato uno schema che rappresenta le architetture appena descritte



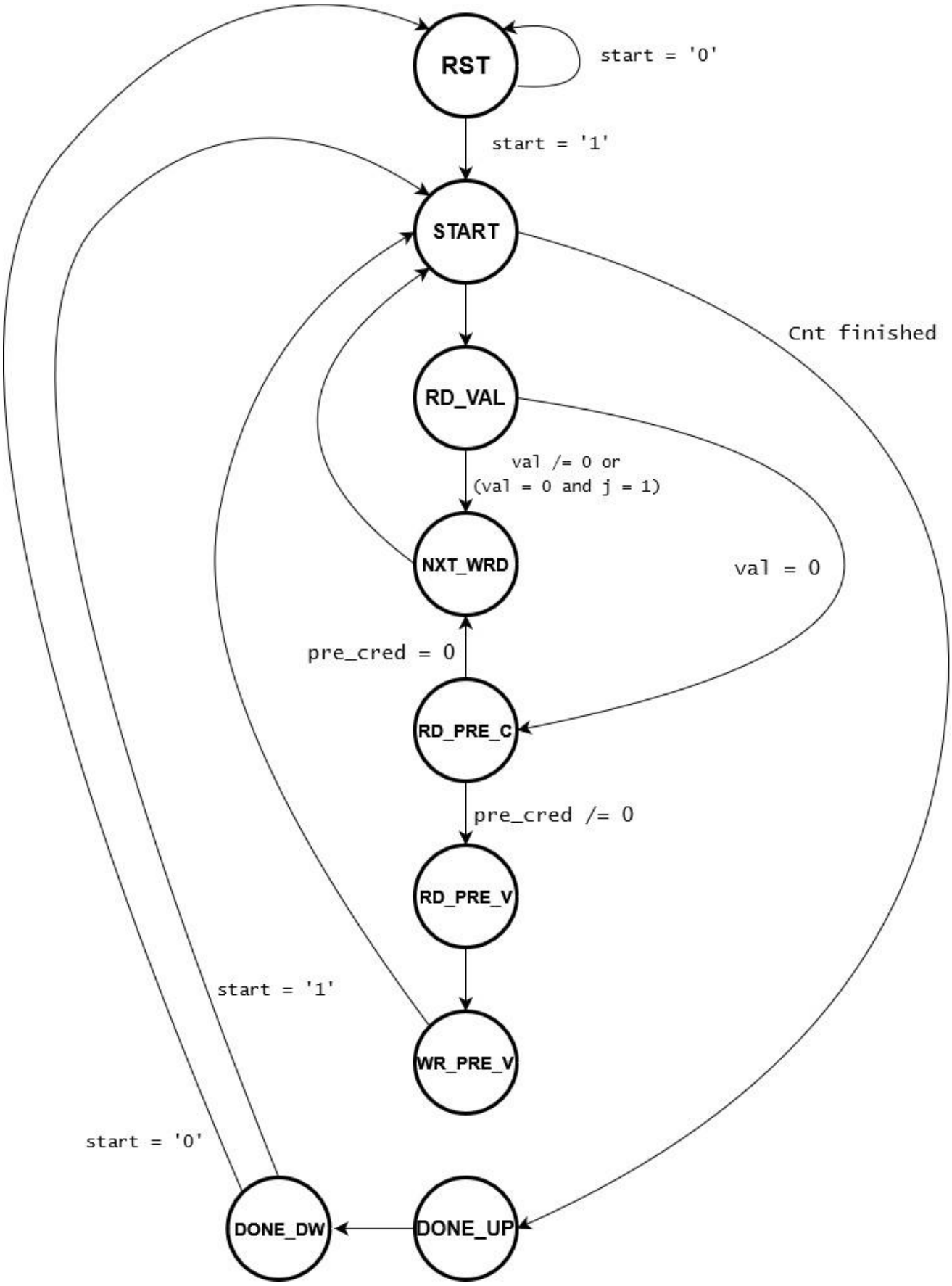
2.3 Macchina a stati finiti

La macchina a stati finiti realizzata conta nove stati. Segue una loro descrizione formale:

- **RESET (RST):** stato in cui viene inserita la macchina all'arrivo del segnale di *RESET* o all'inizio della computazione. È in attesa di ricevere il segnale di *START = 1*.
- **START (START):** stato di inizio della lettura di una parola. Si possono verificare due condizioni:
    - Il segnale *i_j* è uguale a *2*i_k*; quindi, la computazione è finita e si entra nel ciclo di terminazione ponendo il segnale di *DONE* a uno.
    - La computazione non è ancora terminata e si procede alla lettura del valore *W*.
- **READ_VALUE (RD_VAL):** stato nel quale si analizza il valore ricevuto dalla memoria:
    - Valore zero (*"il valore non è specificato")*: se è la prima iterazione della computazione (*i_j = 0*) allora il valore letto non verrà sostituito e la credibilità verrà posta a zero. Altrimenti si procede alla lettura della credibilità della parola precedente.
    - Valore diverso da zero: la sua credibilità viene posta a 31.
- **NEXT_WORD (NXT_WRD):** stato nel quale il contatore viene alzato per aumentare l'offset e andare a leggere il prossimo valore *W*.
- **READ_PREVIOUS_CREDIBILITY (RD_PRE_C):** stato nel quale si analizza la credibilità associata alla parola precedente:
    - Se essa è zero allora viene lasciata a zero la credibilità del valore corrente.
    - Se è diversa da zero, viene copiata e decrementata di uno.
- **READ_PREVIOUS_VALUE (RD_PRE_V):** stato nel quale la macchina viene posta se il valore *W* è zero e non è il primo valore. In questo stato leggo l'ultimo valore valido.
- **WRITE_PREVIOUS_VALUE (WR_PRE_V):** segue allo stato di RD_PRE_V e sovrascrive il valore zero con l'ultimo valore valido.
- **DONE_UP (DONE_UP):** stato di inizio del ciclo di terminazione, il segnale di *START* è stato abbassato e viene mantenuto per un altro ciclo di clock quello di *DONE* alto.
- **DONE_DOWN (DONE_DW):** stato nel quale viene abbassato il valore del segnale *DONE*. L'esecuzione è terminata e si possono verificare due condizioni:
    - *START = 0*: la macchina viene posta nello stato RESET
    - *START = 1*: ricomincio l'esecuzione tornando allo stato di BEGIN

Inoltre, in ogni stato dell'esecuzione qualora il segnale di *START* dovesse diventare zero, allora si tornerebbe allo stato di RESET.

Viene riportato uno schema che descrive il comportamento della macchina a stati finiti appena descritta.


RST
start = '0'
start = '1'
START
Cnt finished
RD_VAL
val /= 0 or
(val = 0 and j = 1)
NXT_WRD
val = 0
pre_cred = 0
RD_PRE_C
pre_cred /= 0
RD_PRE_V
start = '1'
WR_PRE_V
start = '0'
DONE_DW
DONE_UP

3. Risultati sperimentali

3.1 Report di sintesi

Per la sintesi del modulo si è usata la FPGA xc7a200tfbg484-1. Dal punto di vista dell'area, il report di sintesi riporta il seguente utilizzo dei componenti:

- LUT as Logic: 139 (0.10% del totale)
- FF: 14 (<0.01% del totale)

Si è fatta particolare cautela nella scrittura del codice per evitare l'utilizzo di Latch.

Inoltre, attraverso il *timing_report*, è possibile verificare che il modulo rispetta il periodo di clock di 20 ns richiesto. In particolare, lo *Slack Time* è pari a 15.880 ns.

3.2 Simulazioni

Per testare il corretto funzionamento del modulo hardware, esso è stato sottoposto ad una batteria di test comprendente il test bench fornito dai docenti.

Successivamente, prendendo ispirazione dal file di test fornito, è stato creato un generatore di test casuali in Python. Mediante questo codice sono stati creati 1000 file da sottoporre al modulo, e sono stati tutti passati con successo. Mediante questo script, le variabili che venivano generate casualmente erano:

- *K*, che poteva assumere un valore compreso tra 0 e 1023.
- I valori delle parole *W*, che potevano assumere valori compresi tra 0 e 255.

Visto il grande quantitativo di test generati, si è anche creato un software che permettesse l'automatizzazione delle simulazioni su Vivado.

A questo link si possono trovare i codici sopracitati:

https://github.com/Iugal23/testbench

3.3 Edge cases

Si è posta particolare attenzione a dei possibili casi limite che si potevano incontrare nella sequenza. Per testare il corretto funzionamento del modulo in presenza di queste determinate configurazioni, sono stati generati manualmente i corrispondenti casi di test. Sono stati considerati i casi in cui:

- Tutte le parole della sequenza hanno valore zero.
- Si hanno più sequenze una successiva all'altra.
- Viene inviato il segnale di *RESET* durante l'esecuzione.
- La sequenza si apre con valore zero.
- Si trova un valore diverso da zero ogni 31 parole

In tutti i casi, l'architettura ha gestito le sequenze nella maniera attesa, ritornando esiti positivi in ogni test.

4. Conclusioni

Si ritiene che il modulo progettato soddisfi le richieste fornite dalla specifica. Ciò è stato controllato mediante numerosi test, sia generati casualmente, sia scritti manualmente.

Inoltre, l'architettura evita l'utilizzo di Latch, che avrebbero potuto instaurare dei cicli infiniti.

Dal punto di vista del design, si è scelto di utilizzare una macchina a stati finiti e un contatore. In questa maniera si è separata la logica del completamento della sequenza dall'offset della memoria da considerare per interagirci. Non si è ritenuto opportuno aggiungere altri componenti per non complicare ulteriormente la comprensione e l'utilizzo del modulo hardware.