Prova Finale di Reti Logiche
Anno Accademico 2023-2024

Filippo Raimondi
Codice Persona: 10809051

Indice

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è sviluppare un modulo hardware descritto in VHDL, in grado di elaborare una sequenza di dati memorizzata in una memoria esterna. L'obiettivo principale è gestire i dati incompleti della sequenza, sostituendo i valori non specificati (pari a 0) con l'ultimo valore valido letto, e calcolando un livello di credibilità associato a ciascun dato. Il modulo, inoltre, deve segnalare il completamento della computazione e permettere l'elaborazione di nuove sequenze senza richiedere un reset esplicito.

1.2 Specifica generale

Il sistema analizza una sequenza di K parole W, ciascuna con un valore compreso tra 0 e 255. All'interno della sequenza, il valore 0 non rappresenta un dato valido, ma indica semplicemente che "il valore non è specificato". La sequenza di parole è memorizzata a partire da un indirizzo iniziale, chiamato ADD, con ogni parola posizionata a intervalli di 2 byte. Questo significa che la prima parola si trova all'indirizzo ADD, la seconda a ADD+2, la terza a ADD+4, e così via con incrementi regolari di 2.
Il compito del sistema è completare la sequenza, sostituendo i valori pari a 0 con l'ultimo valore letto diverso da 0. Inoltre, per ogni parola W, il sistema calcola un valore aggiuntivo chiamato "credibilità" C, che viene memorizzato nel byte successivo alla parola (ad esempio: se W è in ADD, C sarà memorizzato in ADD+1).
Le regole per il calcolo e la gestione di C sono le seguenti:

1. Ogni volta che si incontra un valore W diverso da 0, C viene impostato a 31.
2. Ogni volta che si incontra un valore W pari a 0, C viene decrementato rispetto al valore precedente, fino a un minimo di 0.
3. Se C raggiunge il valore 0, rimane stabile e non viene ulteriormente decrementato.

Se la sequenza inizia con uno o più valori pari a 0, questi rimangono invariati e il valore di C associato viene impostato a 0, fino a quando non si incontra il primo valore W diverso da 0.

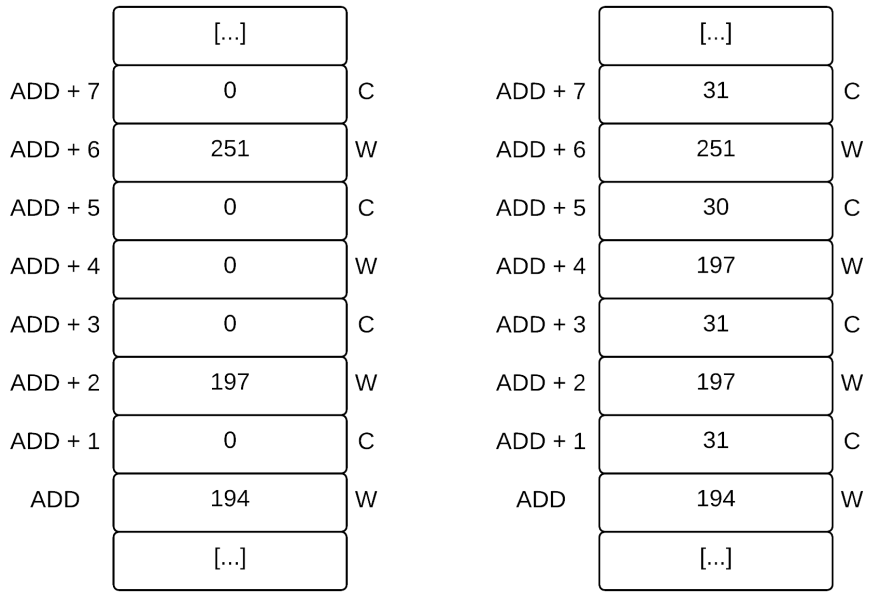

Figura 1: Esempio di sequenza prima e dopo l'elaborazione del sistema

1.3 Interfaccia del componente

Il componente da descrivere ha la seguente interfaccia:

```
entity project_reti_logiche is
       port (
               i_clk         : in  std_logic;
               i_rst         : in  std_logic;
               i_start       : in  std_logic;
               i_add         : in  std_logic_vector(15 downto 0);
               i_k           : in  std_logic_vector(9 downto 0);
               o_done        : out std_logic;
               o_mem_addr    : out std_logic_vector(15 downto 0);
               i_mem_data    : in  std_logic_vector(7 downto 0);
               o_mem_data    : out std_logic_vector(7 downto 0);
               o_mem_we      : out std_logic;
               o_mem_en      : out std_logic
       );
end project_reti_logiche;
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal test bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_start è il segnale di START generato dal test bench;
- i_add è il segnale (vettore) ADD generato dal test bench che rappresenta l'indirizzo dal quale parte la sequenza da elaborare;
- i_k è il segnale (vettore) K generato dal test bench rappresentante la lunghezza della sequenza;
- o_done è il segnale DONE di uscita che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- i_mem_data è il segnale (vettore) che arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura;
- o_mem_data è il segnale (vettore) che va verso la memoria e contiene il dato che verrà successivamente scritto;
- o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura).

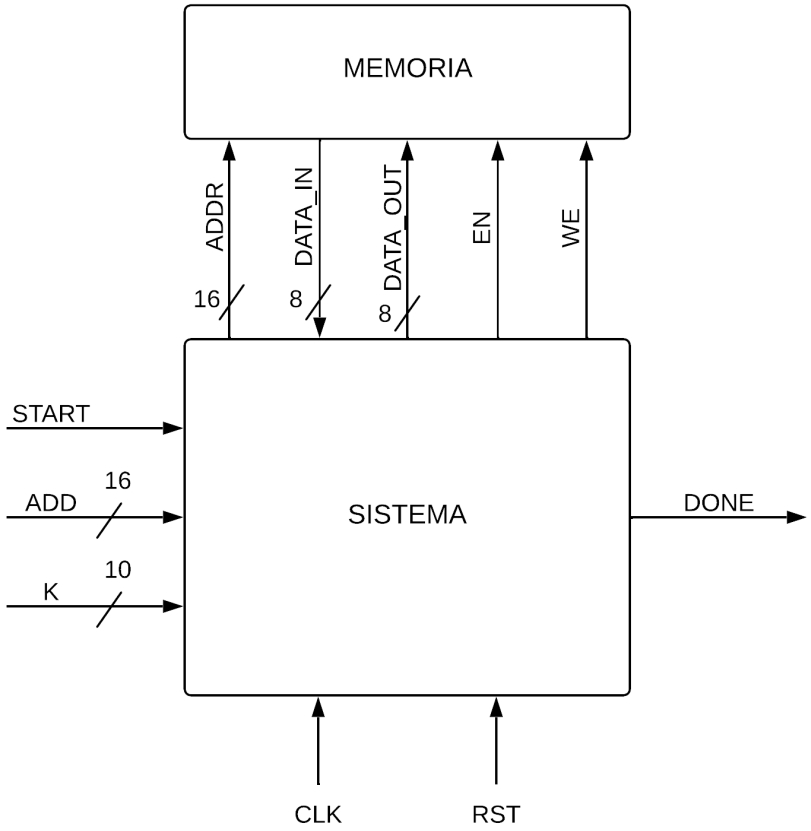

Figura 2: Interfaccia del componente

1.4 Memoria

La memoria implementata nel progetto è un modulo RAM a singola porta che opera in modalità "write-first". Essa è caratterizzata da una larghezza dell'indirizzo di 16 bit, che consente l'accesso a 65.536 locazioni, ciascuna delle quali può contenere dati rappresentati su 8 bit.
Il modulo ha la seguente interfaccia:

```
entity rams_sp_wf is
port (
    clk  : in std_logic;
    we   : in std_logic;
    en   : in std_logic;
    addr : in std_logic_vector(15 downto 0);
    di   : in std_logic_vector(7 downto 0);
    do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;
```

Il funzionamento della memoria è sincronizzato sul fronte di salita del clock (CLOCK) ed è controllato dai segnali di abilitazione (EN) e di abilitazione alla scrittura (WE). Quando EN = 1, la memoria è accessibile.

- Se, durante questo periodo, WE = 1, viene effettuata un'operazione di scrittura: il dato in ingresso (DI) viene scritto nell'indirizzo specificato (ADDR) e, dopo un ritardo di 2 ns, lo stesso dato viene reso disponibile in uscita (DO).
- Se, invece, WE = 0 mentre EN = 1, viene effettuata un'operazione di lettura: il dato presente all'indirizzo specificato viene letto dalla memoria e reso disponibile in uscita (DO) dopo un ritardo di 2 ns.

Questa memoria è già istanziata all'interno del test bench e non necessita di essere sintetizzata.

2 Architettura

2.1 Funzionamento

All'avvio del sistema, il modulo deve impostare il segnale di uscita DONE a 0. Dopo che il segnale RESET è stato riportato a 0, l'elaborazione può iniziare solo quando il segnale di ingresso START viene portato a 1. Durante l'elaborazione, START rimane alto fino a quando il segnale DONE non diventa 1, segnalando la fine del processo. Una volta completata la computazione e salvato il risultato, DONE resta alto finché START non torna a 0, impedendo una nuova elaborazione finché DONE non viene azzerato.
Prima del primo utilizzo, è necessario impostare il segnale RESET a 1 per configurare correttamente il modulo. Dopo questo reset iniziale, non sono richiesti ulteriori reset per elaborazioni successive, a meno che non venga esplicitamente fornito un nuovo segnale RESET, che re-inizializza completamente il modulo.

2.2 FSM

La computazione del modulo è controllata da una macchina a stati finiti (FSM). Ogni stato rappresenta una fase specifica del processo di elaborazione, con transizioni condizionate da segnali di ingresso e valori intermedi. La FSM garantisce un flusso ordinato e sequenziale delle operazioni, dalla fase iniziale di idle (IDLE) fino alla segnalazione del completamento (DONE).
L'automa è definito da 11 stati.

2.2.1 IDLE state

Lo stato iniziale in cui il modulo attende che il segnale i_start sia impostato a 1. Se i_k (dimensione della sequenza) è diverso da 0, il modulo inizia l'elaborazione, altrimenti passa direttamente allo stato DONE.

2.2.2 CHECK_ADDR state

Controlla se l'indirizzo corrente (calcolato con i_add e counter_reg) è valido e rientra nei limiti della memoria. A seconda della condizione, può passare a leggere una parola (SET_READ_W), scrivere il valore di credibilità (SET_WRITE_C), o terminare (DONE) se l'indirizzo non è valido.

2.2.3 SET_READ_W state

Prepara il modulo per leggere un valore W dalla memoria (i_mem_data), impostando l'indirizzo di memoria corretto su o_mem_addr e abilitando il segnale di lettura (o_mem_en).

2.2.4 READ_W state

Legge il valore dalla memoria (i_mem_data) per utilizzarlo nella fase successiva (PROCESS_W).

2.2.5 PROCESS_W state

Esegue il controllo e l'elaborazione del valore letto. Se il dato letto è valido (diverso da 0), lo salva come ultimo valore valido (last_valid_w_reg) e aggiorna il valore di credibilità a 31. Se il valore è nullo, diminuisce il valore di credibilità (se non è già 0).

2.2.6 SET_WRITE_W state

Prepara il modulo a scrivere l'ultimo valore valido di W in memoria. Imposta l'indirizzo corretto su o_mem_addr e abilita i segnali di scrittura (o_mem_we e o_mem_en).

2.2.7 WRITE_W state

Scrive in memoria il valore valido calcolato nella fase precedente (last_valid_w_reg). Dopo la scrittura, passa a controllare l'indirizzo per il valore di credibilità (CHECK_ADDR).

2.2.8 SET_WRITE_C state

Prepara il modulo a scrivere il valore di credibilità in memoria. Calcola l'indirizzo di memoria corretto per il valore di credibilità e lo imposta su o_mem_addr. Abilita i segnali di scrittura (o_mem_we e o_mem_en).

2.2.9 WRITE_C state

Scrive in memoria il valore di credibilità calcolato (credibility_reg). Passa poi al controllo del contatore (CHECK_COUNTER) per verificare se sono stati elaborati tutti i dati della sequenza.

2.2.10 CHECK_COUNTER state

Controlla se il contatore (counter_reg) ha raggiunto il valore massimo specificato da i_k. Se non ha ancora completato tutti i dati, incrementa il contatore e torna a leggere nuovi dati (CHECK_ADDR). Se il contatore ha raggiunto il valore massimo, passa a DONE.

2.2.11 DONE state

Segnala che l'elaborazione è terminata alzando il segnale o_done. Il modulo rimane in questo stato finché i_start non torna a 0, dopodiché si reimposta e ritorna nello stato IDLE.

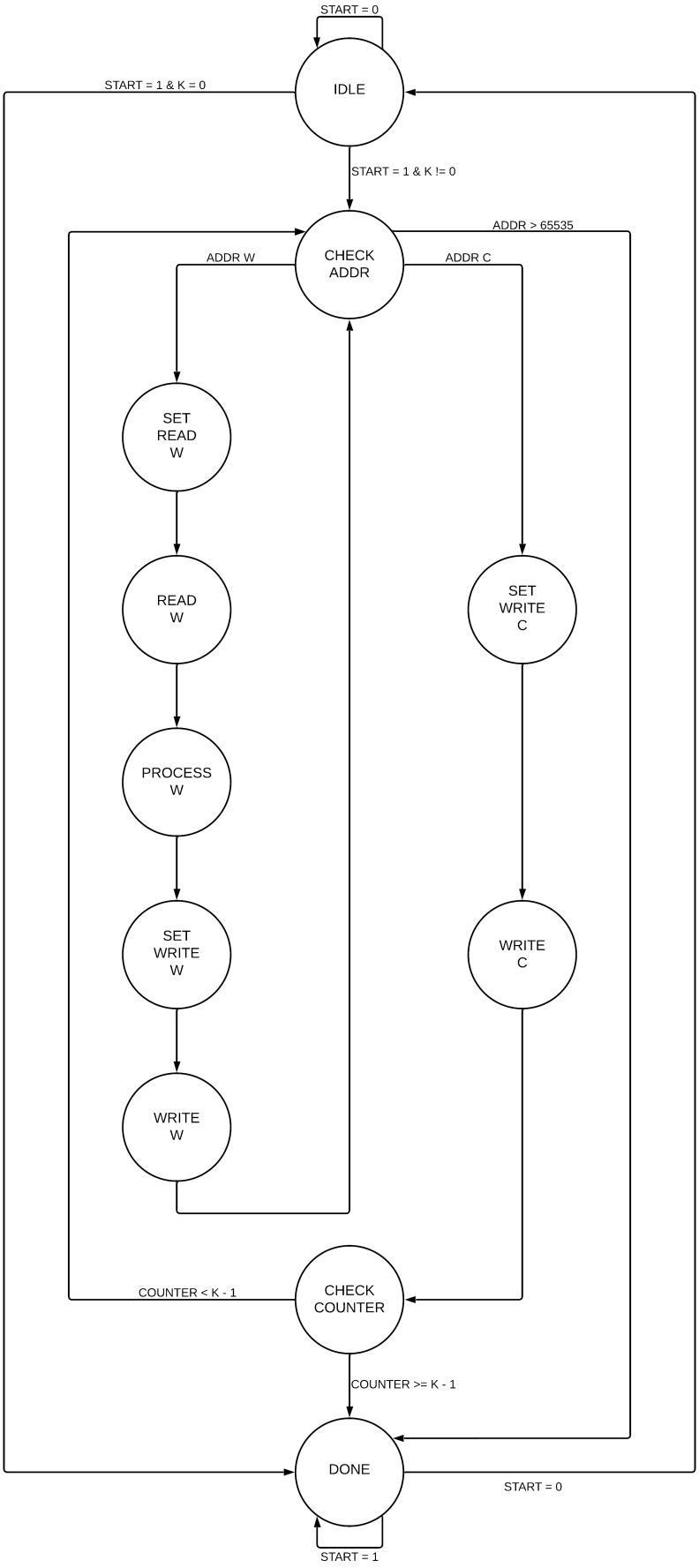

Figura 3: Diagramma degli stati della FSM

2.3 Scelte progettuali

2.3.1 Struttura dei processi

La descrizione del componente è suddivisa in due processi distinti:

1. Processo Sequenziale (Register Transfer Level - RTL):
   Questo processo si occupa di gestire la parte sequenziale del componente. Include la sincronizzazione al fronte di salita del CLOCK e al segnale di RESET, con l'obiettivo di aggiornare lo stato corrente della FSM e i registri interni, come last_valid_w_reg, counter_reg e credibility_reg. La sua funzione principale è quella di garantire la corretta memorizzazione e il trasferimento dei dati attraverso i vari stati della macchina, permettendo il funzionamento ordinato e sequenziale del sistema.

2. Processo Combinatorio (Finite State Machine - FSM):
   Questo processo rappresenta la logica combinatoria della FSM. Analizza i segnali di ingresso e lo stato corrente per determinare le transizioni di stato, le uscite appropriate e le operazioni da eseguire sui dati.

Questa divisione permette di separare chiaramente la gestione della parte sequenziale da quella combinatoria.

2.3.2 Operazioni logiche e aritmetiche

Il componente utilizza operazioni aritmetiche e confronti logici per elaborare i dati e calcolare valori. Addizione e sottrazione vengono impiegate per calcolare gli indirizzi di memoria e aggiornare i contatori, come counter_reg e credibility_reg. La moltiplicazione viene utilizzata per calcolare gli offset di memoria.
I confronti logici verificano la validità delle parole W (i_mem_data $\neq 0$), controllano che il valore della credibilità non sia mai minore di 0 (credibility_reg > 0) e prevengono errori come l'overflow degli indirizzi.
Queste operazioni guidano le transizioni di stato e l'aggiornamento dei registri interni.

2.3.3 Gestione dei tipi di dato

Per garantire una gestione corretta degli indirizzi di memoria e dei dati, il sistema richiede la conversione tra diversi tipi di dato, come std_logic_vector, unsigned e integer. Tali conversioni, effettuate tramite funzioni come to_integer e to_unsigned, consentono di eseguire operazioni aritmetiche sui segnali e di adattarli alle necessità del modulo.

3 Test bench

Sono stati eseguiti test bench per verificare il corretto funzionamento del componente in una varietà di condizioni operative. I test hanno incluso sia scenari ottimali che situazioni limite, per garantire che il sistema risponda correttamente anche in circostanze eccezionali o con segnali asincroni.

3.1 Test bench eseguiti in condizioni ottimali

- Sequenze normali: Il modulo è stato testato con diverse sequenze di dati rappresentative di condizioni standard per verificare il corretto funzionamento di tutte le funzionalità principali.

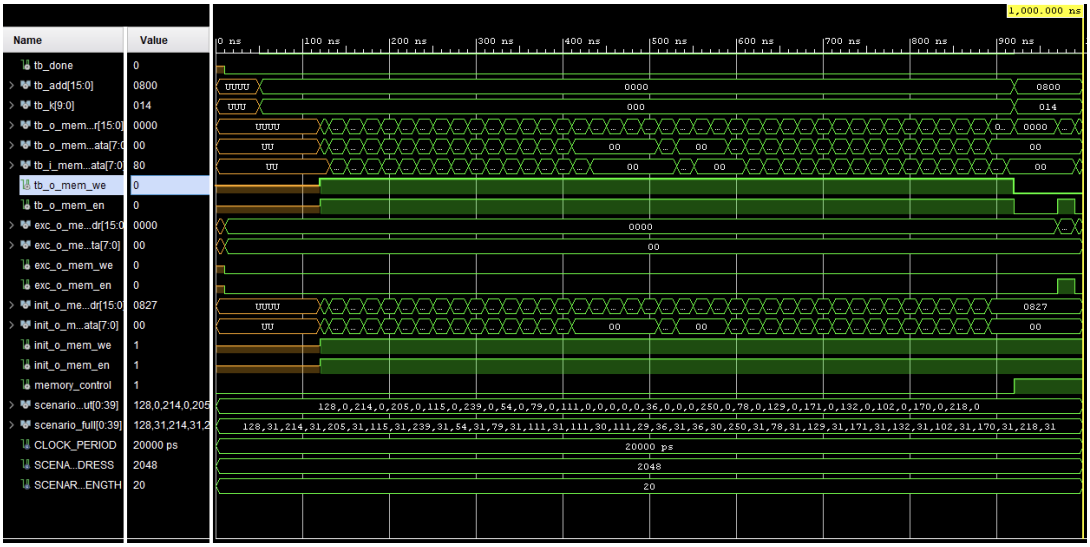

Figura 4: Waveform di simulazione: sequenza normale

3.2 Test bench dei casi limite

- K = 0: Il componente è stato testato con K=0 per valutare il comportamento nei casi di minima dimensione della sequenza.
- K = 1023: È stato controllato il funzionamento utilizzando il massimo valore consentito per K, per verificare il rispetto delle specifiche.
- Overflow nel calcolo dell'indirizzo: Sono stati creati scenari in cui gli indirizzi calcolati per W e C superavano i limiti di memoria per valutare il comportamento in caso di overflow.

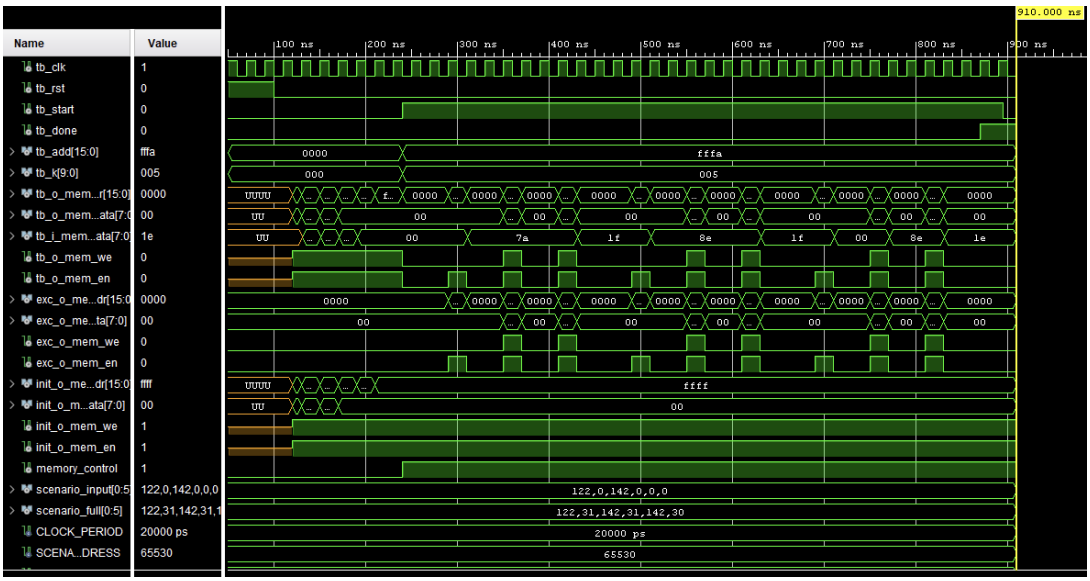

Figura 5: Waveform di simulazione: overflow nel calcolo dell'indirizzo

- Sequenza di tutti zeri: Il test ha verificato la capacità del componente di gestire correttamente una sequenza composta esclusivamente da zeri.
- Sequenza che inizia con un po' di zeri: È stato valutato se il componente riesce a processare correttamente sequenze che iniziano con dati nulli prima di incontrare valori validi.
- Credibilità a zero: È stato verificato che il valore di credibilità non scendesse mai sotto lo zero, anche quando continuamente decrementato, rispettando i vincoli del progetto.

3.3 Test bench dei segnali asincroni

- Reset asincrono: Il componente è stato sottoposto a un reset asincrono per verificarne il comportamento immediato e la capacità di riavviarsi correttamente.

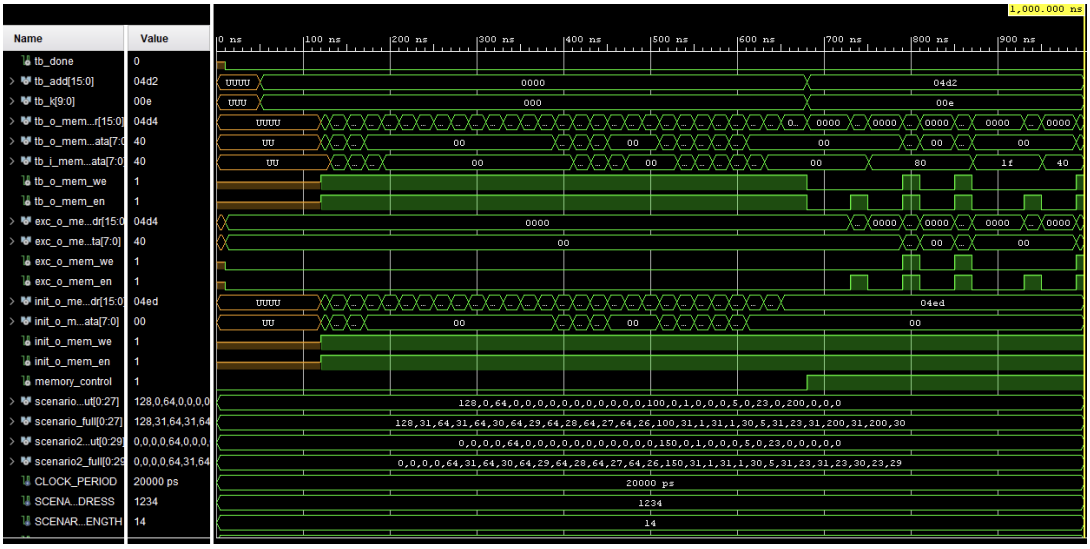

Figura 6: Waveform di simulazione: reset asincrono

- Due sequenze consecutive senza necessità di reset: Sono stati verificati la continuità operativa e il comportamento del sistema su più sequenze consecutive, evitando reset.

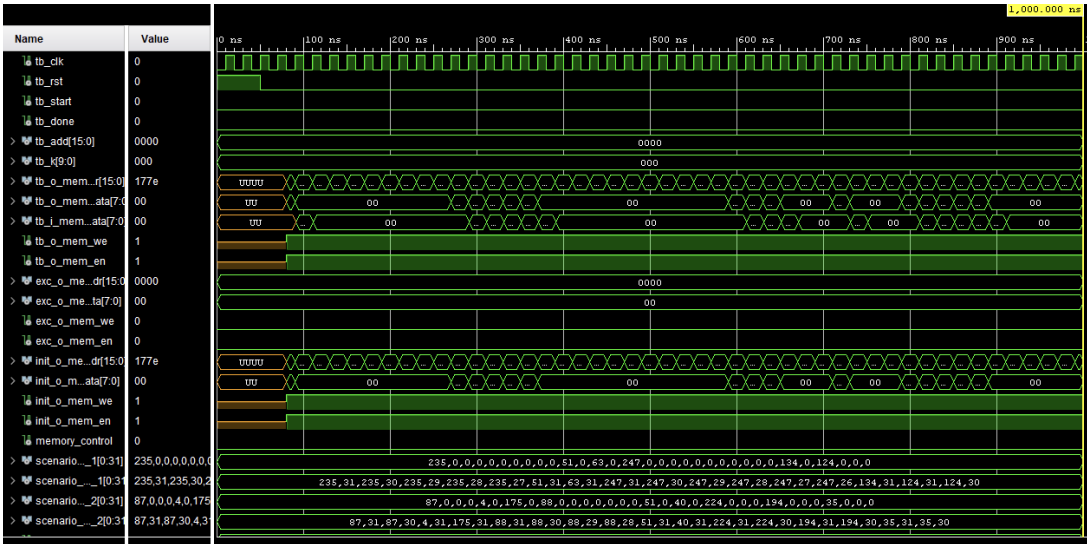

Figura 7: Waveform di simulazione: sequenze consecutive

4 Conclusioni

Il modulo hardware, descritto in VHDL, è stato simulato e sintetizzato con successo, dimostrando un funzionamento affidabile sia in fase pre-sintesi che post-sintesi.
L'implementazione proposta gestisce in modo efficace la sostituzione dei valori non validi e il calcolo del livello di credibilità, garantendo un'elaborazione corretta della sequenza di dati nel pieno rispetto dei requisiti definiti dalla specifica. Inoltre, il modulo è in grado di operare su sequenze multiple senza necessità di reset.
Infine, l'assenza di latch indesiderati e il rispetto dei requisiti temporali, con ampi margini di sicurezza sui tempi di setup, confermano la validità delle scelte progettuali adottate.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	101	0	134600	0.08
LUT as Logic	101	0	134600	0.08
LUT as Memory	0	0	46200	0.00
Slice Registers	56	0	269200	0.02
Register as Flip Flop	56	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

Figura 8: Report di utilizzo delle risorse hardware FPGA

```
Timing Report

Slack (MET) :             14.179ns  (required time - arrival time)
  Source:                 counter_reg_reg[0]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            o_done_reg/D
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
```

Figura 9: Report di analisi temporale