# Progetto di Reti Logiche
# 2023/2024

Prof. Fabio Salice

Luigi Inguaggiato

Matricola: 981760

Codice Persone: 10809587

POLITECNICO
MILANO 1863

# Sommario

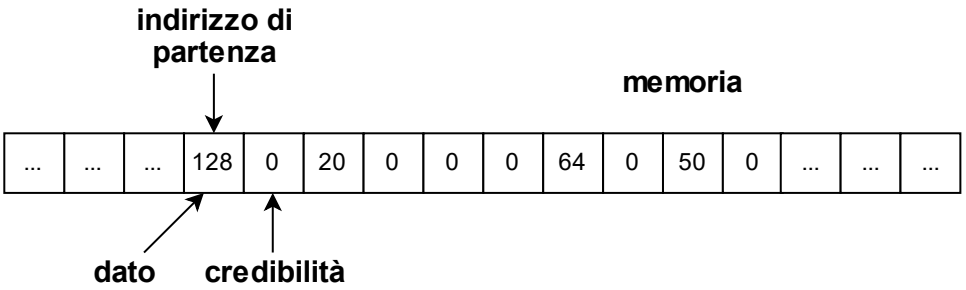
# Introduzione

## Obiettivo del progetto

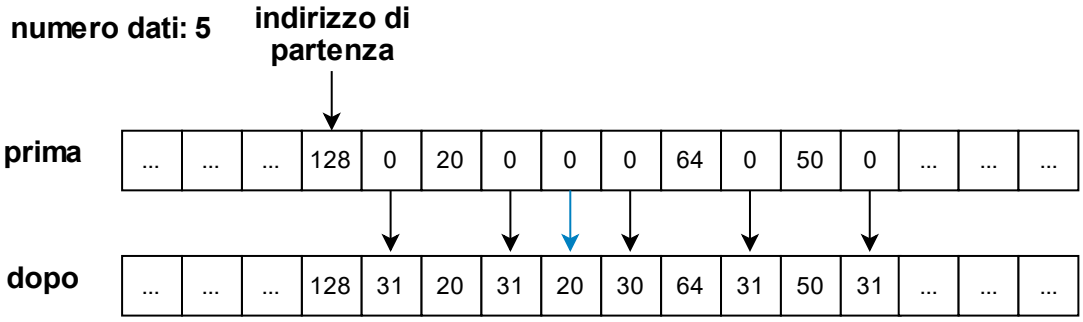
Il progetto mira a sviluppare un componente hardware in VHDL in grado di analizzare, correggere e valutare l'affidabilità dei dati provenienti da un sensore. L'output finale sarà una sequenza di dati tutti validi, con i dati mancanti approssimati al valore precedente e accompagnati da una valutazione della loro affidabilità.

## Specifiche generali

Il componente riceve in ingresso l'indirizzo di partenza dei dati campionati dal sensore e il numero di dati. Ogni dato è composto da due byte: il primo rappresenta il valore del dato, mentre il secondo indica la credibilità associata a quel valore.

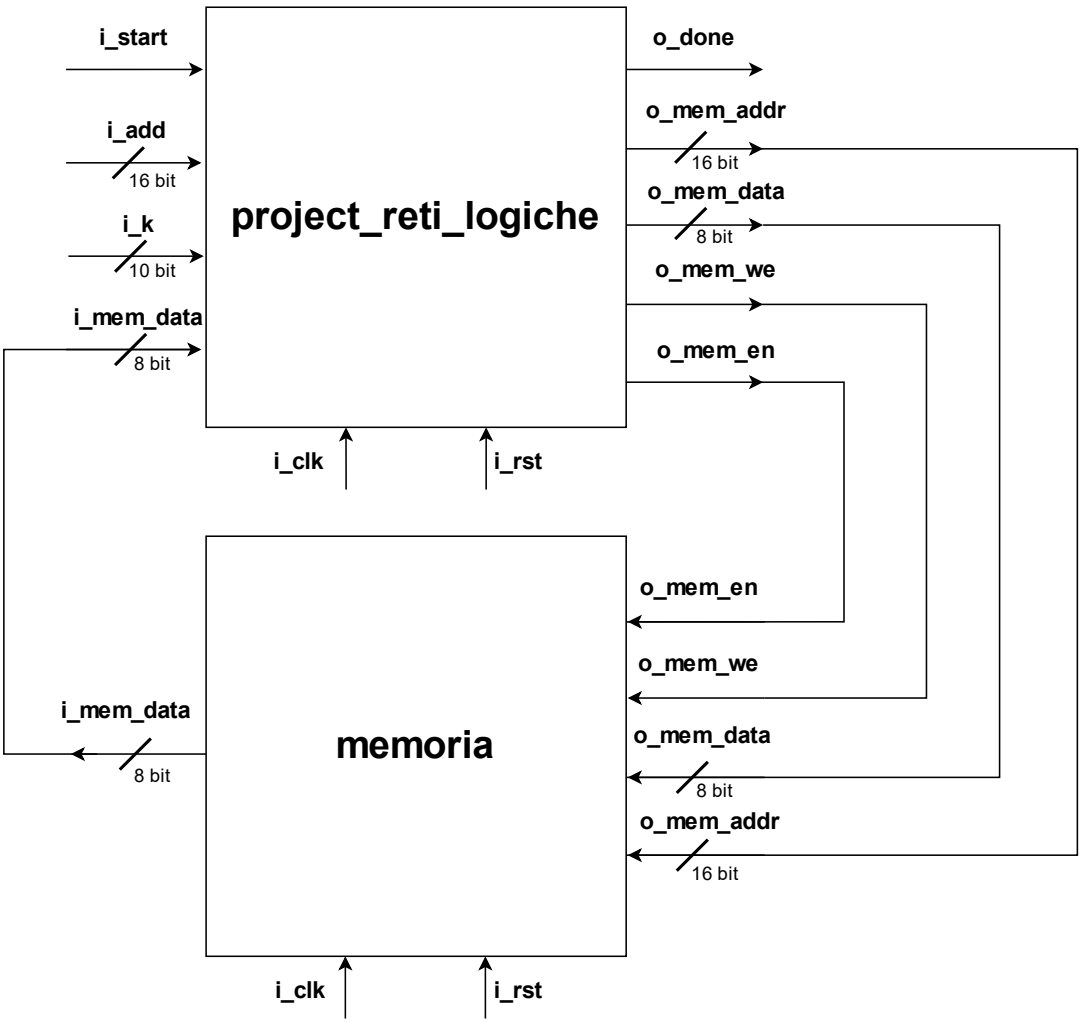



Tutti i dati non validi, identificati con il valore 0, vengono sostituiti con l'ultimo valore valido e ne viene aggiornata la credibilità. La credibilità iniziale per tutti i dati validi è 31 e diminuisce di 1 per ogni dato non valido consecutivo.



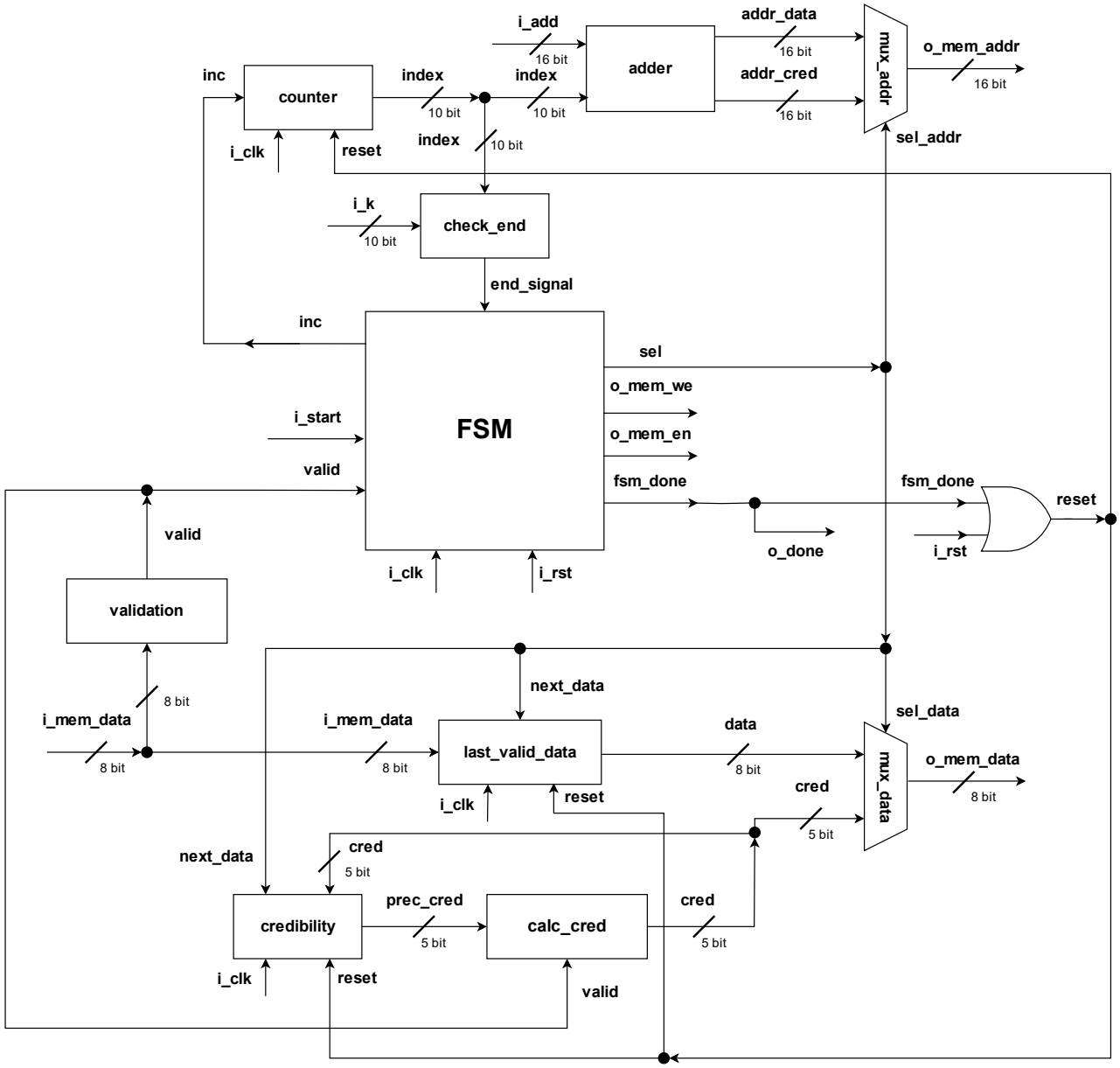

Per esempio, nell'immagine fornita, le frecce indicano i dati che sono stati modificati. Si osserva che a ogni dato è stata assegnata una credibilità di 31, ad eccezione del terzo dato (indicato dalla freccia azzurra) che è stato valutato come non valido (cioè pari a 0). Pertanto, è stato sostituito con il dato valido precedente, pari a 20, e la sua credibilità è stata ridotta a 30 (31 meno 1).

## Interfaccia del componente




- `i_clk`: Segnale di CLOCK in ingresso generato dal Test Bench.
- `i_rst`: Segnale di RESET che inizializza la macchina per ricevere il primo segnale di START.
- `i_start`: Segnale di START generato dal Test Bench.
- `i_k`: Segnale (vettore) W generato dal Test Bench che indica la lunghezza della sequenza.
- `i_add`: Segnale (vettore) ADD generato dal Test Bench che rappresenta l'indirizzo di partenza della sequenza da elaborare.
- `o_done`: Segnale di uscita DONE che indica la fine dell'elaborazione.
- `o_mem_addr`: Segnale (vettore) di uscita che trasmette l'indirizzo alla memoria.
- `i_mem_data`: Segnale (vettore) in ingresso dalla memoria che contiene il dato letto.
- `o_mem_data`: Segnale (vettore) in uscita verso la memoria che contiene il dato da scrivere.
- `o_mem_en`: Segnale di ENABLE da inviare alla memoria per la comunicazione (lettura e scrittura).
- `o_mem_we`: Segnale di WRITE ENABLE da inviare alla memoria (=1) per consentire la scrittura. Deve essere 0 per la lettura dalla memoria.

# Architettura




## Scelte progettuali

Il componente `project_reti_logiche` è stato progettato in modo modulare. La suddivisione in moduli consente di isolare e gestire in modo efficiente le diverse funzionalità.

Le parti sincrone del componente sono state separate dalle altre per creare moduli dedicati al salvataggio dei dati quali: ultimo valore valido, della credibilità del dato precedente, del conteggio dei dati e dello stato della macchina. Questi moduli, denominati `last_valid_data`, `credibility`, `counter` e `FSM`, svolgono principalmente operazioni di memorizzazione tranne `FSM` che è più articolata ed ulteriormente divisa in due processi uno sequenziale per il cambio di stato e un combinatorio per i segnali di controllo (segnali di output della `FSM`).

Per consentire al componente di eseguire più elaborazioni corrette dopo un singolo reset (`i_rst`), i moduli `counter`, `credibility` e `last_valid_data` vengono resettati quando viene rilevato un segnale di fine elaborazione (`fsm_done`). Per fare ciò, sono stati messi in OR `fsm_done` e `i_rst`, in

modo che l'uscita dell'OR, denominata `reset`, faccia resettare i 3 moduli sia quando il segnale di reset (`i_rst`) viene chiamato, sia quando l'elaborazione è finita.

Il componente inizia l'elaborazione quando il segnale `i_start` è attivo. Durante l'elaborazione, viene letto un dato alla volta dalla memoria, la sua credibilità viene calcolata e aggiornata, e eventualmente i dati non validi vengono sostituiti con l'ultimo valore valido. Alla fine dell'elaborazione, il segnale `o_done` viene alzato per segnalare il completamento. Il segnale `o_done` rimane attivo fino a quando `i_start` non viene deselezionato.

Si osserva che il segnale `o_done` è equivalente al segnale `fsm_done`. Due nomi distinti sono stati attribuiti allo stesso segnale per distinguere tra i segnali interni del componente, come `fsm_done`, e quelli che vengono trasmessi verso l'esterno, come `o_done`.

# Descrizione dei moduli

## Modulo `counter`

- **Ingressi**:
  - `i_clk`
  - `reset`
  - `inc`
- **Uscite**:
  - `index` (10 bit)

Il modulo `counter` è sincrono e incrementa l'indice (`index`) quando il segnale `inc` è attivo sul fronte di salita del clock. Il segnale di reset (`reset`) reimposta l'indice a 0.

## Modulo `check_end`

- **Ingressi**:
  - `i_k` (10 bit)
  - `index` (10 bit)
- **Uscite**:
  - `end_signal`

Il modulo `check_end` determina se l'elaborazione è completata confrontando l'indice (`index`) con il numero totale di dati (`i_k`).

## Modulo `validation`

- **Ingressi**:
  - `i_mem_data` (10 bit)
- **Uscite**:
  - `valid`

Il modulo `validation` determina se il dato letto dalla memoria è valido (diverso da 0) e segnala la validità attraverso il segnale `valid`.

## Modulo `adder`

- **Ingressi**:
  - `i_add` (16 bit)
  - `index` (10 bit)
- **Uscite**:
  - `addr_data` (16 bit) = `i_add` + 2 * `index`
  - `addr_cred` (16 bit) = `i_add` + 2 * `index` + 1

Il modulo `adder` calcola gli indirizzi del dato da elaborare (`addr_data`) e della sua credibilità (`addr_cred`). Poiché i dati sono memorizzati in blocchi di 2 byte (dato + credibilità), l'indice viene moltiplicato per 2 (shiftato a sinistra di 1) per ottenere l'indirizzo corretto.

## Modulo `mux_addr`

- **Ingressi**:
  - `sel_addr`
  - `addr_data` (16 bit)
  - `addr_cred` (16 bit)
- **Uscite**:
  - `o_mem_addr` (16 bit)

Il modulo `mux_addr` è un multiplexer che seleziona l'indirizzo del dato o della sua credibilità in base al segnale di selezione `sel_addr`.

`sel_adrr = 0` per `addr_data`

`sel_adrr = 0` per `addr_cred`

## Modulo `mux_data`

- **Ingressi**:
  - `sel_data`
  - `data` (8 bit)
  - `cred` (5 bit)
- **Uscite**:
  - `o_mem_data` (8 bit)

Il modulo `mux_data` è un multiplexer che seleziona il dato o la credibilità estesa di segno in base al segnale di selezione `sel_data`.

`sel_data = 0` per `data`

`sel_data = 1` per `cred`

## Modulo `last_valid_data`

- **Ingressi**:
  - `i_clk`
  - `reset`
  - `i_mem_data` (10 bit)
  - `next_data`
- **Uscite**:
  - `data`

Il modulo `last_valid_data` è sincrono e salva l'ultimo dato valido. Ignora tutti i dati in ingresso tranne quando il segnale `next_data` è attivo sul fronte di salita del clock. Il segnale di reset reimposta l'ultimo dato valido a 0.

## Modulo `credibility`

- **Ingressi**:
  - `i_clk`
  - `reset`
  - `cred` (5 bit)
  - `next_data`
- **Uscite**:
  - `prec_cred` (5 bit)

Il modulo `credibility` è sincrono e salva la credibilità (`cred`) quando il segnale `next_data` è attivo e il clock è alto. In uscita (`prec_cred`) mette la credibilità salvata. Il segnale di reset reimposta la credibilità a 0.

## Modulo `calc_cred`

- **Ingressi**:
  - `prec_cred` (5 bit)
  - `valid`
- **Uscite**:
  - `Cred` (5 bit)

Il modulo `calc_cred` calcola la credibilità del dato corrente. Se il dato è valido (`valid` = 1), la credibilità viene impostata a "11111"; altrimenti viene decrementata di 1 rispetto alla credibilità precedente (`prec_cred`), evitando che scenda al di sotto di 0.

## Modulo `FSM`

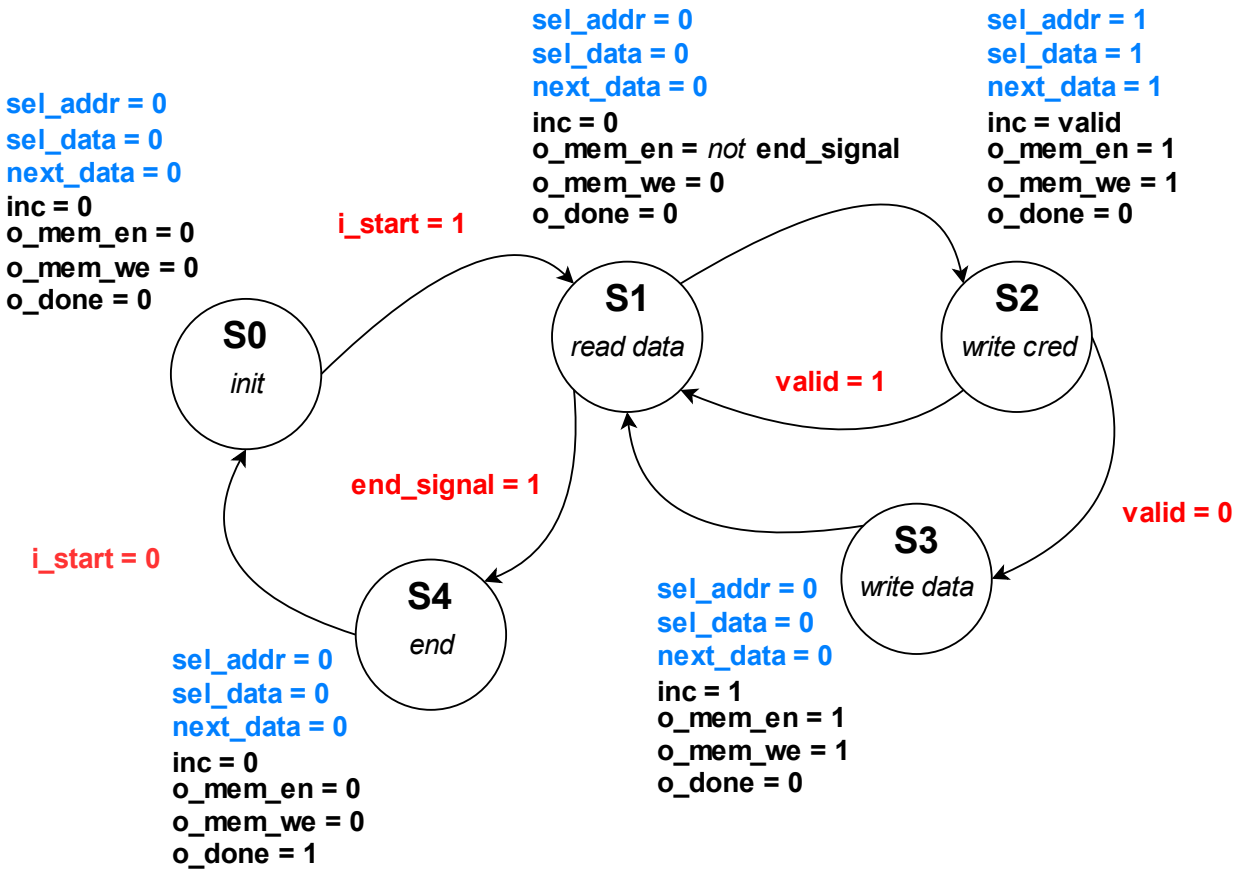
- **Ingressi**:
  - `i_clk`
  - `i_rst`
  - `i_start`
  - `valid`
  - `end_signal`
- **Uscite**:
  - `sel`
  - `inc`
  - `o_mem_en`
  - `o_mem_we`
  - `fsm_done`

Il modulo `FSM` è una macchina a stati finiti che controlla il flusso di elaborazione dei dati.

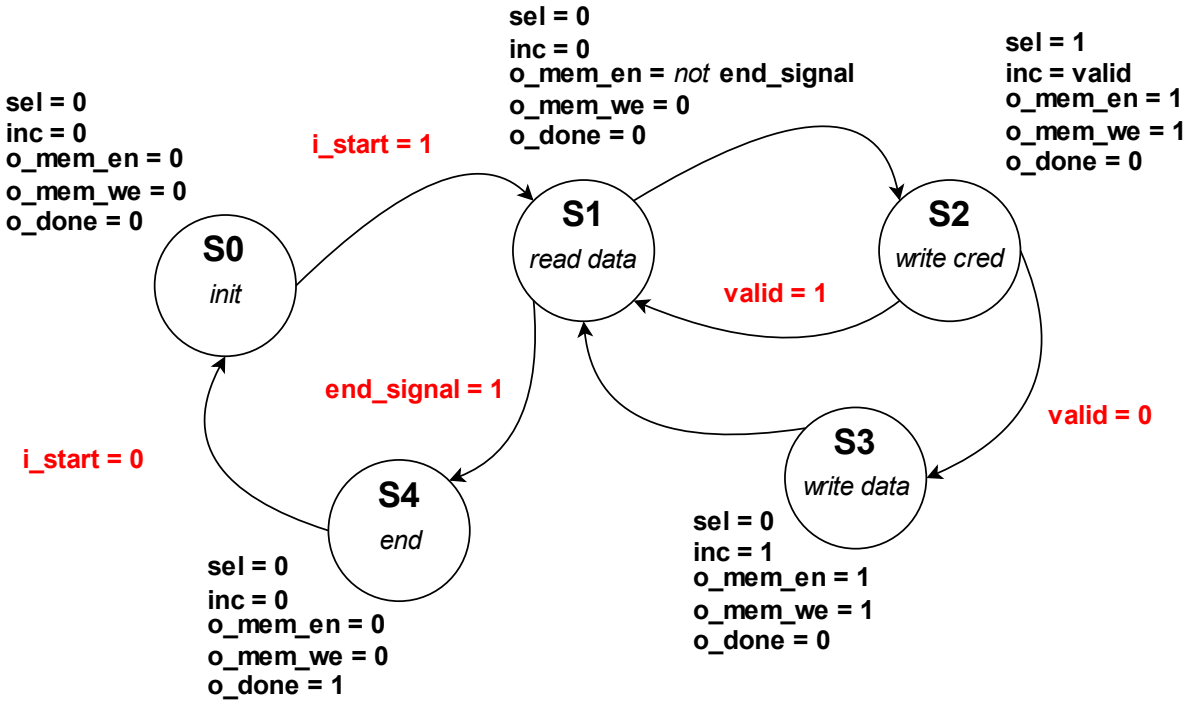
Si compone di 5 stati:

- **S0**: Stato iniziale prima dell'inizio di ogni elaborazione.
- **S1**: Legge il dato dalla memoria.
- **S2**: Scrive la credibilità del dato in memoria ed incrementa il contatore se il dato è valido.
- **S3**: Sostituisce il dato non valido con l'ultimo valido.
- **S4**: Elaborazione completata, in attesa che `i_start` scenda.

I segnali di controllo `sel_addr`, `sel_data` e `next_data` assumono sempre gli stessi valori come si vede nella seguente figura



quindi sono stati condensati in un unico segnale `sel`




Il segnale di reset (`i_rst`) riporta la macchina allo stato S0. Quando `i_start` diventa attivo, la macchina passa attraverso gli stati S1-S3 per elaborare i dati.

In S1 vengono impostati i segnali per leggere il dato dalla memoria.

Al successivo fronte di salita del clock, la macchina passa allo stato S2, dove viene scritta la credibilità del dato. Se il dato letto è valido, la macchina ritorna allo stato S1 e il contatore viene incrementato per leggere il prossimo dato. Se il dato non è valido è necessario sovrascriverlo quindi il prossimo stato diventa S3, senza incrementare il contatore.

Nello stato S3, il dato viene sovrascritto con l'ultimo valore valido e il contatore viene incrementato per tornare all'esecuzione nello stato S1.

In S1 se `end_signal` è attivo (tutti i dati sono stati elaborati), la macchina passa allo stato S4 e segnala il completamento dell'elaborazione tramite `fsm_done`.

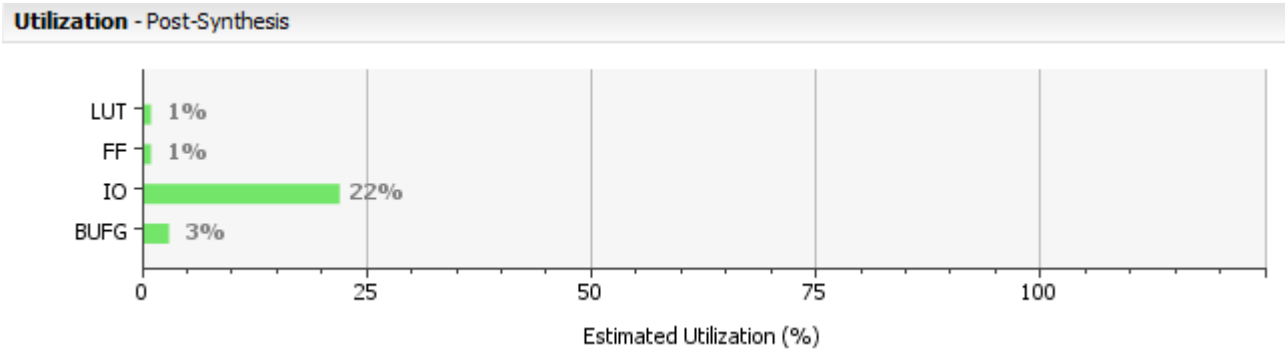
# Risultati Sperimentali

## Sintesi

Il componente è stato sintetizzato con successo utilizzando l'Artix-7 FPGA xc7a200tfbg484-1. Durante la sintesi, è stato verificato che il componente soddisfacesse correttamente i test forniti, oltre ad essere stato sottoposto a una serie di test aggiuntivi per valutarne le prestazioni in modo più approfondito.

### Utilization Report

Il report di utilizzo mostra che il componente sintetizzato utilizza complessivamente 26 flip-flop e non sono stati generati latch.




| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 70 | 0 | 134600 | 0.05 |
| LUT as Logic | 70 | 0 | 134600 | 0.05 |
| LUT as Memory | 0 | 0 | 46200 | 0.00 |
| Slice Registers | 26 | 0 | 269200 | <0.01 |
| Register as Flip Flop | 26 | 0 | 269200 | <0.01 |
| Register as Latch | 0 | 0 | 269200 | 0.00 |
| F7 Muxes | 0 | 0 | 67300 | 0.00 |
| F8 Muxes | 0 | 0 | 33650 | 0.00 |

### Timing Report

Il componente ha superato i requisiti temporali imposti dal clock di periodo 20 ns. Lo slack temporale, che rappresenta la differenza tra il tempo richiesto e il tempo effettivamente impiegato, è risultato essere di 16.854 ns.

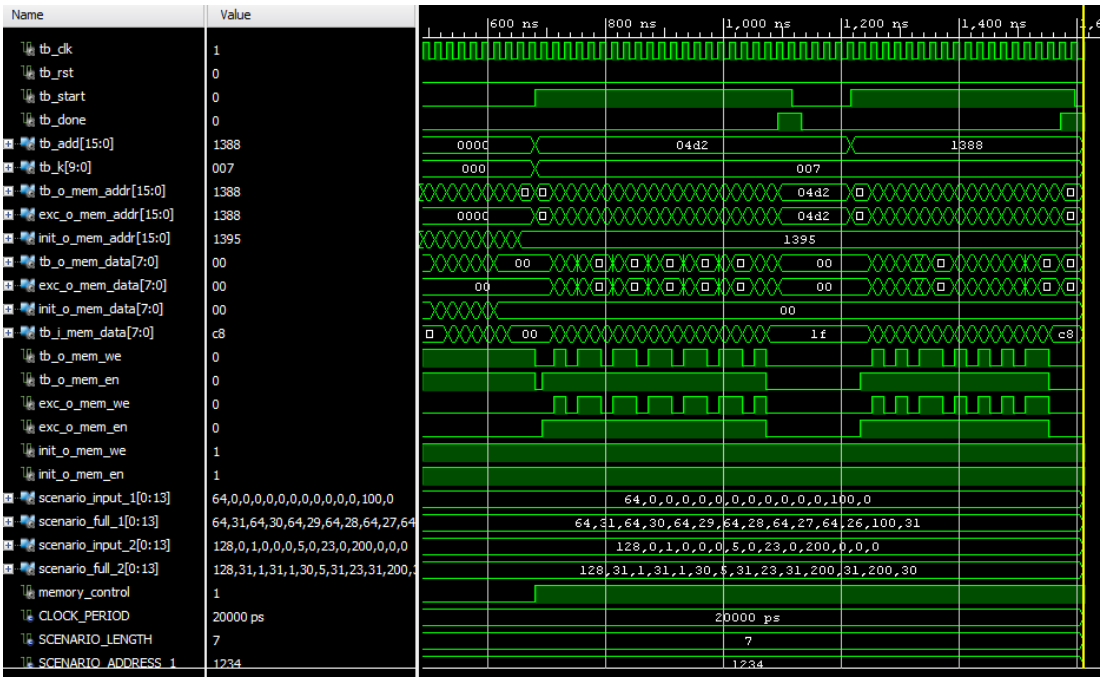
```
Timing Report

Slack (MET) :             16.854ns  (required time - arrival time)
 From Clock:  clock
   To Clock:  clock

Setup :            0  Failing Endpoints,  Worst Slack       16.854ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.144ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        9.500ns,  Total Violation        0.000ns
```

## Test Bench

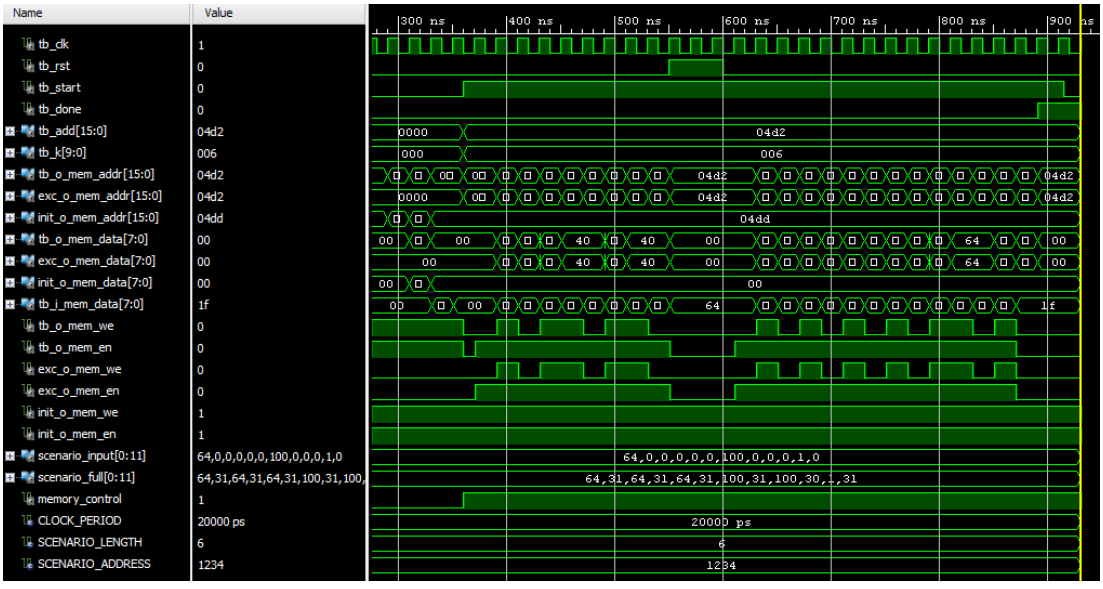
Per garantire il corretto funzionamento del componente in situazioni critiche e in generale, sono stati sviluppati numerosi test, oltre a quelli forniti.

Ogni test viene attentamente condotto per verificare che il componente modifichi correttamente la sequenza di dati in memoria, evitando l'accesso non autorizzato durante le fasi di elaborazione e garantendo che non vi siano modifiche o letture di parti della memoria non destinate a essere accedute.

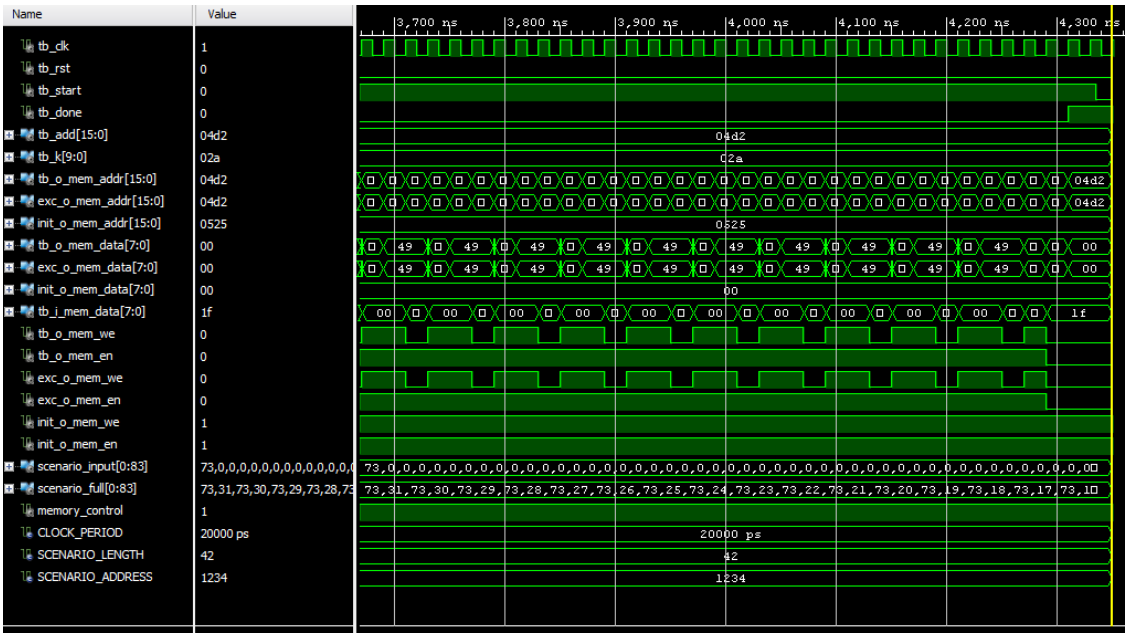
1. **Test multiple elaborazioni**: Questo test verifica che il componente sia in grado di eseguire correttamente molteplici elaborazioni consecutive dopo aver ricevuto il segnale di reset.



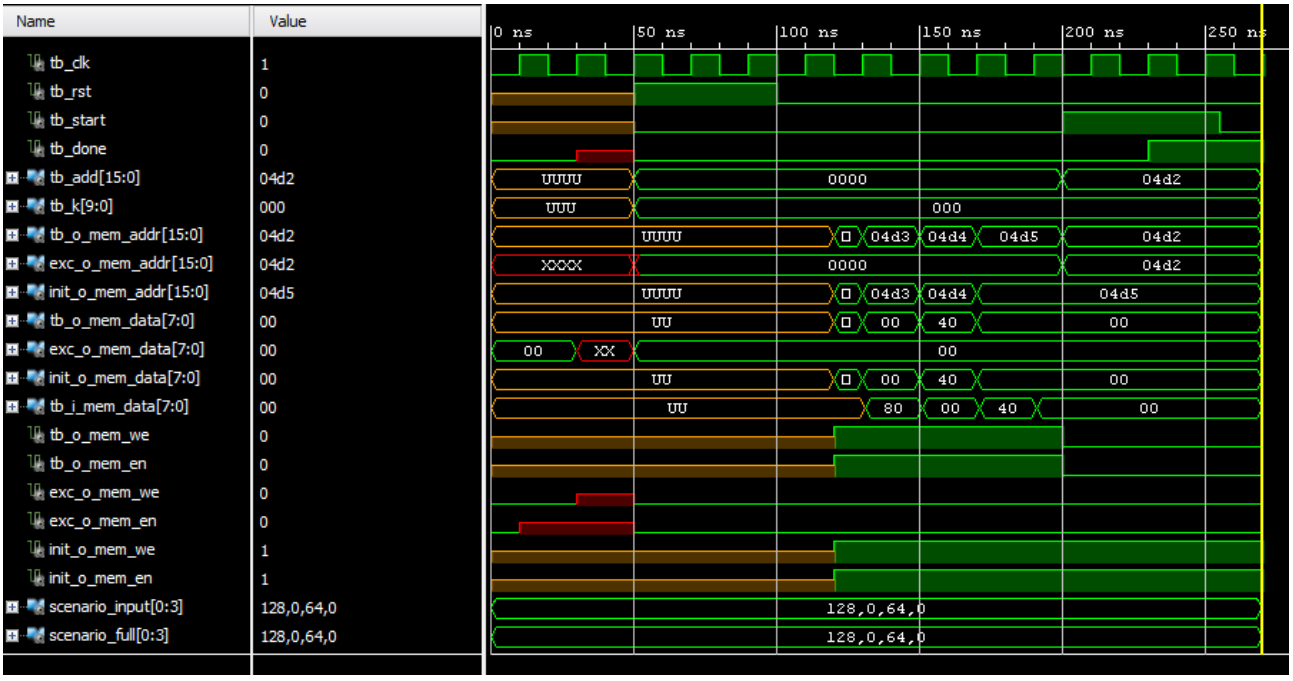

2. **Test segnale di reset durante l'elaborazione**: Verifica che il componente, se riceve un segnale di reset durante un'elaborazione, ritorni nello stato iniziale e riprenda l'esecuzione senza errori.



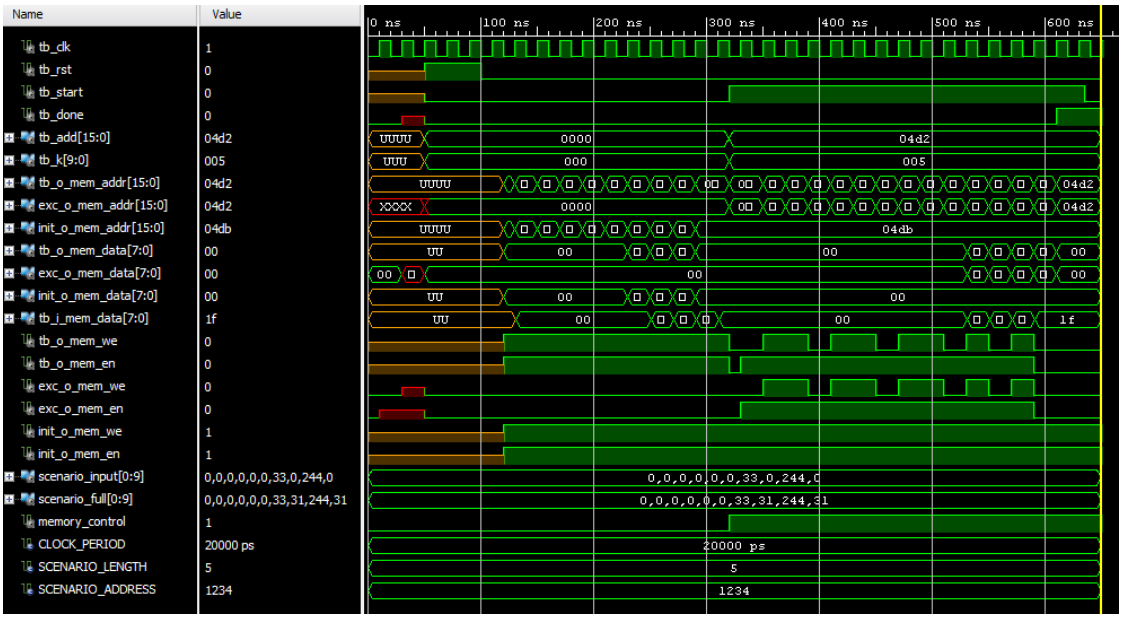
3. **Test con più di 31 zeri di fila**: Questo test verifica come il componente gestisce una sequenza con più di 31 zeri consecutivi. Si verifica che la credibilità degli zeri successivi sia correttamente impostata a 0 senza assumere valori negativi.



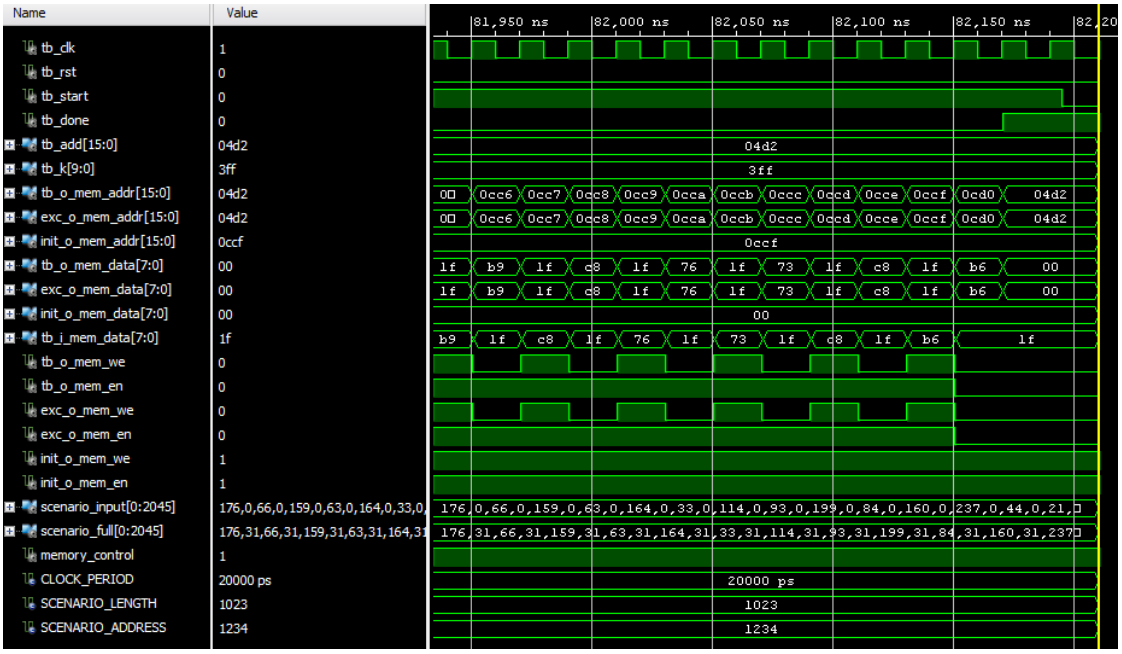

4. **Test 0 dati da elaborare**: Verifica che il componente termini correttamente l'elaborazione quando non ci sono dati da processare, senza effettuare scritture o letture dalla memoria.



5. **Test dato iniziale 0**: Si assicura che il componente gestisca correttamente un dato iniziale pari a 0, lasciandolo inalterato in memoria e associandogli una credibilità di 0.




6. **Test grandezza massima**: Verifica che il componente sia in grando di gestire correttamente 1023 dati (il numero di dati è un segnale di 10 bit quindi il massimo valore è 1023).




Oltre ai test sopraelencati che testano situazione specifiche, il componente è stato sottoposto a test di grandi dimensioni, generati con uno script python, per assicurare il suo funzionamento.

# Conclusione

Il componente progettato ha superato con successo una serie di test bench, dimostrando la sua affidabilità e robustezza nell'elaborazione dei dati e nel rispetto dei requisiti della specifica del progetto. La sintesi del componente è stata completata con successo, rispettando i limiti temporali imposti e utilizzando un numero limitato di flip-flop. I test progettati per valutare il comportamento del componente in vari scenari, inclusi casi limite, edge-case, hanno confermato la sua capacità di gestire situazioni complesse in modo accurato ed efficiente.

L'architettura modulare del componente non solo facilita la comprensione del suo funzionamento, ma offre anche una flessibilità che consente adattamenti semplici e veloci per soddisfare diverse esigenze. Questo approccio permette di apportare modifiche al sistema senza dover riprogettare l'intero componente da zero.