Prova Finale (Progetto di Reti Logiche)

Prof. Fabio Salice – Anno 2023/2024

Daniele Kopyshevskiy	Codice Persona: 10819304	Codice Matricola: 987418
Christian Malaman	Codice Persona: 10766966	Codice Matricola: 982228

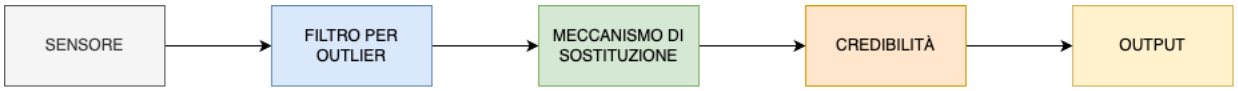
Indice

1. Introduzione

Nel contesto dello sviluppo e dell'implementazione di sistemi sensoriali, la capacità di filtrare e gestire dati non affidabili, in particolare outlier, è cruciale per garantire l'integrità e l'utilità delle informazioni raccolte. Il progetto in questione si concentra sulla creazione di un modulo capace di elaborare i dati provenienti da un sensore, il quale registra valori compresi tra 1 e 255. Tuttavia, dati anomali o valori fuori dal range previsto, classificati come outlier, vengono considerati non affidabili e, pertanto, sostituiti con zero, indicando l'assenza di una lettura valida.

Il sistema implementa un meccanismo di filtraggio che, in presenza di un valore zero, interviene sostituendo tale dato con l'ultimo valore valido rilevato, tenendo conto di una misura di "credibilità" di tale sostituzione. Inizialmente, la credibilità è alta (valore 31), ma con il susseguirsi dei valori zero consecutivi, la sua intensità diminuisce gradualmente fino a stabilizzarsi a zero, indicando la perdita progressiva della fiducia nel valore sostitutivo.

Il fulcro di questa metodologia è quindi non solo la rilevazione dei dati in sé, ma soprattutto la gestione intelligente delle anomalie.



2. Architettura

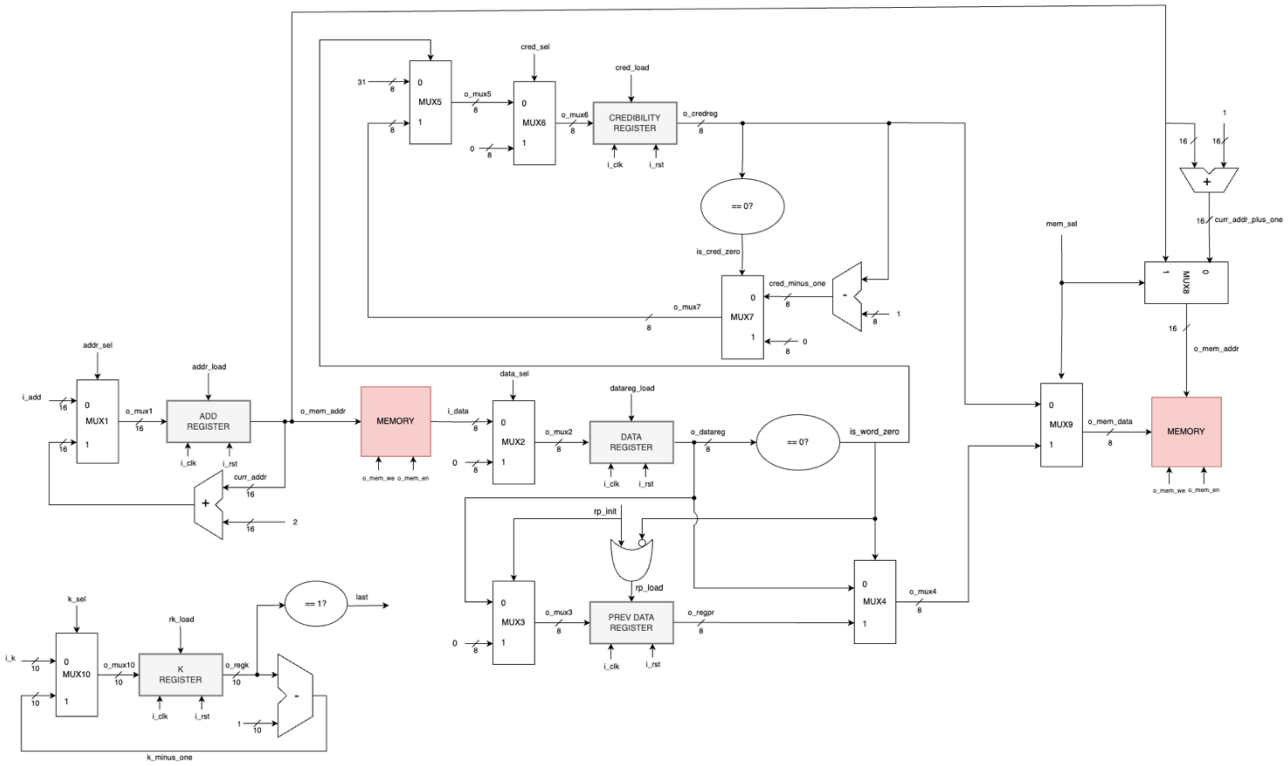
Nel nostro progetto, l'approccio adottato per lo sviluppo del codice VHDL si è focalizzato inizialmente sulla definizione e implementazione del datapath, seguito dalla progettazione e integrazione della macchina a stati finiti (Moore).

2.1 Datapath

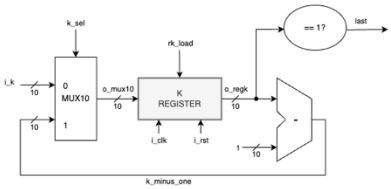
Il datapath è stato progettato per gestire efficacemente il flusso dei dati all'interno del modulo, suddividendolo in diverse sottoreti che trattano specificamente le fasi di acquisizione, elaborazione e scrittura dei dati:

1. Acquisizione dell'Input: In questa fase, abbiamo implementato le sottoreti necessarie per acquisire e memorizzare i segnali di input, preparando il sistema per le fasi successive di elaborazione.
2. Elaborazione dell'Input: Successivamente, i dati acquisiti sono stati analizzati ed elaborati per determinare le operazioni necessarie, utilizzando logiche combinatorie e sequenziali per manipolare e preparare i dati per la scrittura in memoria.
3. Scrittura in Memoria: Infine, le sottoreti dedicate alla scrittura in memoria si occupano di determinare cosa scrivere e dove, assicurando che i dati elaborati vengano correttamente memorizzati nelle locazioni di memoria desiderate.

Di seguito, viene allegato uno schema, sviluppatosi durante la fase di progettazione del modulo hardware, che mette in evidenza come la rete è stata sviluppata.

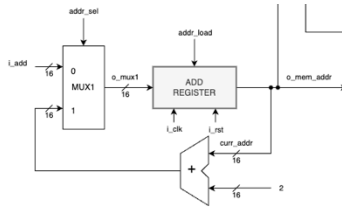


SOTTORETE 1



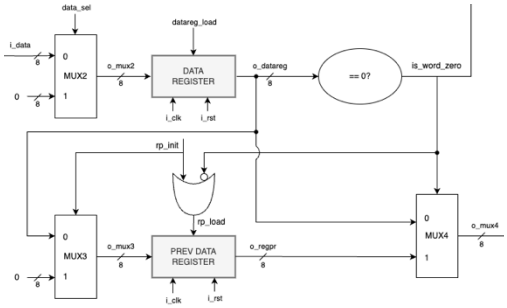
La seguente sottorete funge da contatore: essa ha lo scopo di decrementare il valore di k, ricevuto inizialmente in ingresso, ogniqualvolta che o il dato w o la credibilità associata ad esso viene scritta in memoria. Non appena k = 1, viene alzato il segnale last al fine di indicare che il modulo entra nella sua fase finale di elaborazione relativo alla sequenza di dati ricevuta in ingresso, la quale consiste nell'elaborazione dell'ultimo dato w e della credibilità associata ad esso.

SOTTORETE 2



La seguente sottorete si occupa inizialmente di ricevere l'indirizzo di memoria iniziale i_add, memorizzarlo nel registro add_register, e incrementare il valore curr_addr di due ogni volta che si scrive in memoria la credibilità associata alla parola. Quindi, ad esempio, se leggiamo il dato w, allora scriveremo w in memoria all'indirizzo curr_addr, la credibilità associata a w in curr_addr + 1 ed infine andremo a pescare il dato successivo all'indirizzo curr_addr + 2.
Osserviamo che l'operazione curr_addr + 1 viene gestita in una sottorete diversa.

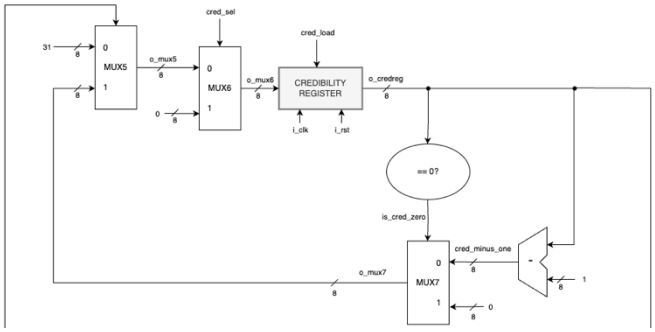
SOTTORETE 3



La seguente sottorete si occupa di analizzare la parola w.
Inizialmente, il contenuto del registro data_register e del registro prev_data_register è inizializzato a zero. Nel caso di data_register, si pone il segnale data_sel = '0' e datareg_load = '1'. Invece, nel caso di prev_data_register, si pone solo il segnale rp_init = '1', in quanto funge sia da segnale di selezione del mux3 sia segnale di enable del registro prev_data_register. Quest'ultima affermazione è dettata dal fatto che il "vero" segnale di enable del registro prev_data_register è rp_load, la cui espressione analitica è la seguente: rp_load = rp_init OR NOT(is_word_zero). Essendo che però is_word_zero è inizialmente zero, l'operando rp_init è l'unico a pilotare il valore del segnale rp_load.
Terminata questa fase iniziale, si pone rp_init = '0' e si verifica, con un comparatore, se il dato memorizzato in data_register è zero o meno. L'uscita del comparatore is_word_zero è un segnale che in questa fase assume un ruolo estremamente importante. Anzitutto, diventa il segnale di enable del registro prev_data_register (il ragionamento è analogo a quanto detto sopra): infatti, nel caso in cui il

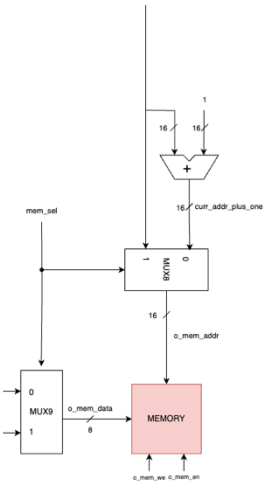
dato memorizzato in data_register sia non uguale a zero, tale dato viene memorizzato nel registro prev_data_register; altrimenti si tiene il valore precedente (osserviamo che se la prima parola della sequenza è zero, allora il contenuto del registro prev_data_register è comunque zero, grazie all'inizializzazione iniziale). Inoltre, is_word_zero è il segnale di selezione del mux4: se il dato memorizzato in data_register è non zero, allora si riscrive tale dato in memoria; altrimenti si scrive il dato memorizzato in prev_data_register. Ed infine, è anche il segnale di selezione del mux5, la cui funzionalità verrà analizzata in seguito poiché mux5 appartiene ad un'altra sottorete.

SOTTORETE 4



La seguente sottorete si occupa di analizzare la credibilità associata al dato w.
Inizialmente, il contenuto del registro credibility_register è inizializzato a zero. Come detto in precedenza, is_word_zero è il segnale di selezione del mux5: infatti, quando il dato w non è uguale a zero, allora la credibilità associata ad esso è 31; in caso contrario, il contenuto del registro credibility_register viene decrementato, grazie al sottrattore, di 1 fin tanto che non si raggiunge il valore zero, in quanto in tal caso non si deve più decrementare.

SOTTORETE 5

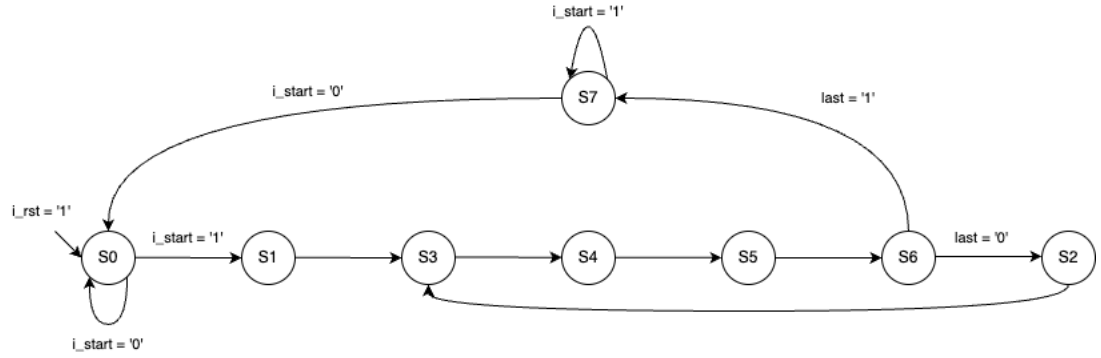


Una volta computati correttamente i valori da scrivere in memoria, bisogna accertarsi di scrivere tali valori nella zona di memoria corretta. La seguente sottorete si occupa di scegliere cosa scrivere in memoria ed in quale indirizzo di memoria scrivere quanto di interesse. In particolare, se mem_sel = 0 allora la scrittura verrà fatta all'indirizzo curr_addr (mux8) e quanto verrà scritto in memoria sarà proprio il dato w (mux9); se mem_sel = 1 allora la scrittura verrà fatta all'indirizzo curr_addr + 1 (mux8), grazie alla presenza del sommatore, e quanto scritto in memoria sarà la credibilità associata al dato w (mux9).

2.2 FSM

Una volta completato e verificato il funzionamento del datapath, abbiamo proceduto con la progettazione della macchina a stati finiti di tipo Moore. Questa macchina a stati è stata integrata per coordinare le operazioni del datapath, gestendo il flusso di controllo e assicurando che le transizioni tra le diverse fasi di elaborazione avvenissero in modo preciso e deterministico.

In figura, una rappresentazione della FSM.



STATO S0

La macchina a stati inizia dallo stato S0, dove rimane in attesa di ricevere il segnale di start. Non appena il segnale i_start diventa '1', il modulo transita allo stato S1, segnando l'inizio dell'elaborazione.

STATO S1

Nello stato S1, il modulo procede con la configurazione iniziale. In particolare, si ha la

- memorizzazione del valore di k (k_sel = ‘0’ e rk_load = ‘1’);
- memorizzazione dell'indirizzo i_add (addr_sel = ‘0’ e addr_load = ‘1’);
- inizializzazione del registro credibility_register (cred_sel = ‘1’ e cred_load = ‘1’);
- inizializzazione del registro prev_data_register (rp_init = ‘1’).

Successivamente, il modulo transita nello stato S3.

STATO S3

Nello stato S3, vengono effettuate le seguenti operazioni:

- setting dell’indirizzo di memoria da cui leggere il dato (o_mem_addr = curr_addr);
- abilitazione della lettura in memoria (en = ‘1’).

Successivamente, il modulo transita nello stato S4.

STATO S4

Nello stato S4, i dati letti dalla memoria vengono caricati nel registro dei dati (datareg_load = ‘1’)
Successivamente, il modulo transita nello stato S5.

STATO S5

Nello stato S5, vengono effettuate le seguenti operazioni:

- si calcola se il dato w è diverso da zero e, in caso affermativo, lo si memorizza nell’apposito registro (dipende dal segnale is_word_zero);

- in base al valore di is_word_zero, si determina se scrivere in memoria il contenuto del registro data_register o del prev_data_register e, successivamente, si scrive quanto di interesse in memoria (en = '1' e we = '1') nell'apposito indirizzo di memoria (o_mem_addr = curr_addr) (si osservi che mem_sel = '0');
- si calcola il valore di credibilità e lo si memorizza nell'apposito registro (cred_load = '1').

Successivamente, il modulo transita nello stato S5.

STATO S6

Nello stato S6, vengono effettuate le seguenti operazioni:

- scrittura in memoria della credibilità associata al dato (mem_sel = '1');
- aggiornamento dell'indirizzo di memoria in cui scrivere la credibilità (o_mem_addr = curr_addr + 1);
- scrittura effettiva in memoria (en = '1' e we = '1');
- aggiornamento del valore di k (rk_load = '1').

A questo punto, se il segnale di last è uguale ad 1, si transita nello stato S7; altrimenti, se si passa nello stato S2.

STATO S2

Nello stato S2, si ha aggiornamento dell'indirizzo di memoria da cui leggere il nuovo dato della sequenza per poi passare nello stato S3.

STATO S7

Nello stato S7 vi si rimane fin tanto che i_start = '1'. Non appena i_start = '0', si ritorna nello stato iniziale S0.

3. Risultati sperimentali

3.1 Report di Sintesi

Nell'**immagine 3.1** si può vedere che lo **Slack** è pari a **15,137 ns** in un periodo di clock di 20 ns. Ciò significa che l'implementazione del progetto usa **4,863 ns** per compiere tutte le operazioni necessarie e finire l'elaborazione; perciò, si potrebbe ridurre il periodo di clock fino a questo valore senza causare problemi di timing, riuscendo ad arrivare ad una **frequenza max** di funzionamento di circa **205 MHz**.

```
Timing Report

Slack (MET) :             15.137ns  (required time - arrival time)
  Source:                 FSM_sequential_cur_state_reg[2]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            curr_addr_reg[13]/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)
  Data Path Delay:        4.880ns  (logic 1.446ns (29.629%)  route 3.434ns (70.371%))
  Logic Levels:           3  (CARRY4=2 LUT5=1)
  Clock Path Skew:        -0.009ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.415ns = ( 24.415 - 20.000 )
    Source Clock Delay      (SCD):    4.763ns
    Clock Pessimism Removal (CPR):    0.339ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Immagine 3.1.1: Timing Report

Nell'**immagine 3.2** è raffigurato il report utilization. Da essa si può vedere come il modulo progettato non faccia uso di latches, mentre sono presenti **54 flip flops** e **105 LUTs** (LookUp Tables).

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	105	0	63400	0.17
LUT as Logic	105	0	63400	0.17
LUT as Memory	0	0	19000	0.00
Slice Registers	54	0	126800	0.04
Register as Flip Flop	54	0	126800	0.04
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Immagine 3.1.2: Utilization Report, parte di Slice Logic

3.2 Simulazioni

Per verificare il corretto funzionamento del modulo progettato abbiamo eseguito dei test per verificare i **corner cases** sotto discussi.

3.2.1 K pari a zero

In questo caso c'è un comparatore iniziale che controlla se i_k sia pari a zero. In caso di esito positivo, la macchina a stati passa direttamente da S1 a S7, impostando done (e di conseguenza o_done) ad 1 ed aspettando il ritorno di i_start a 0 per poter ritornare allo stato S0.

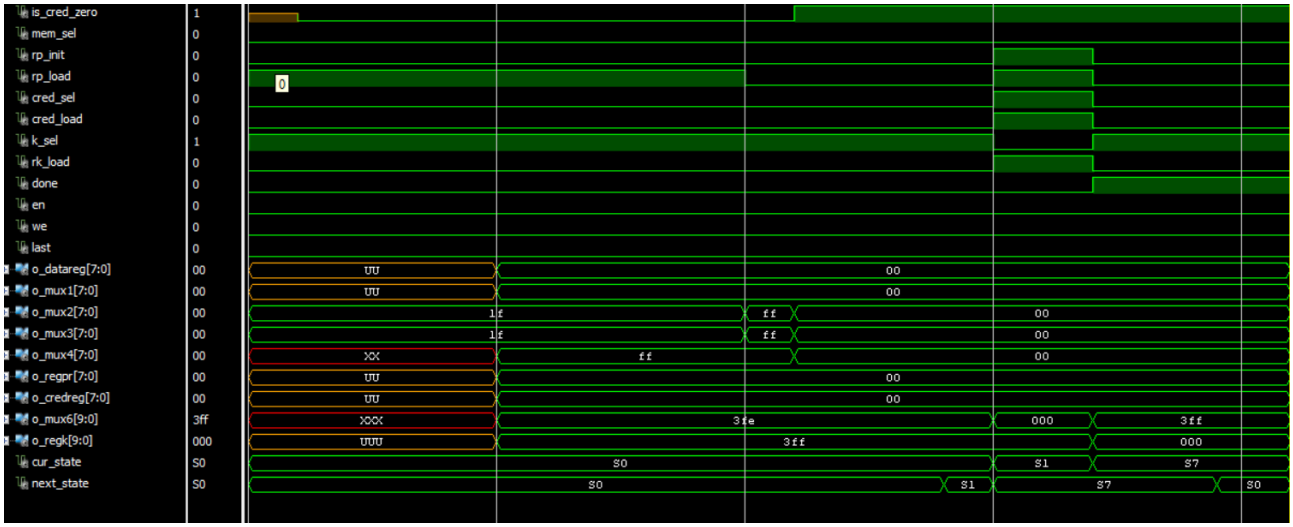


Immagine 3.2.1.1: Test quando i_k = 0

3.2.2 Reset durante l'esecuzione

In questo caso, appena viene attivato il reset, la macchina torna allo stato iniziale, ovvero S0 e tutti i segnali ritornano al loro valore di default. Questo succede appena viene impostato ad uno il valore del reset che, essendo un segnale asincrono, non ha bisogno di aspettare il ciclo di clock successivo per modificare l'elaborazione corrente.

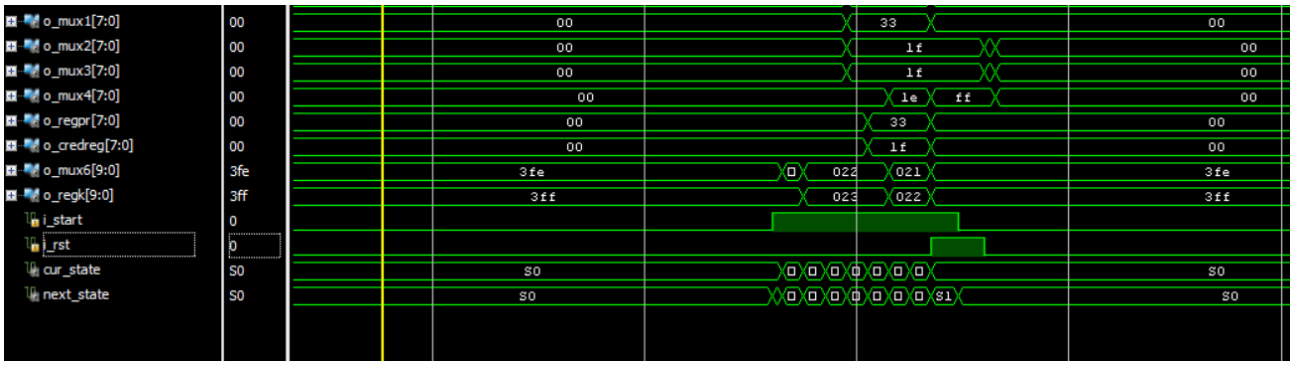


Immagine 3.2.2.1: Test con reset durante un elaborazione

3.2.3 Elaborazione multipla

In questo test andiamo a verificare se due elaborazioni successive sono possibili anche senza un reset in mezzo ad esse. Dall'immagine si può vedere come finita la prima elaborazione la macchina torna in S0 e si mette in attesa della prossima.
Alla successiva richiesta di elaborazione (i_start impostato ad uno) la macchina riparte senza nessun problema passando allo stato S1 e ricominciando una nuova computazione.

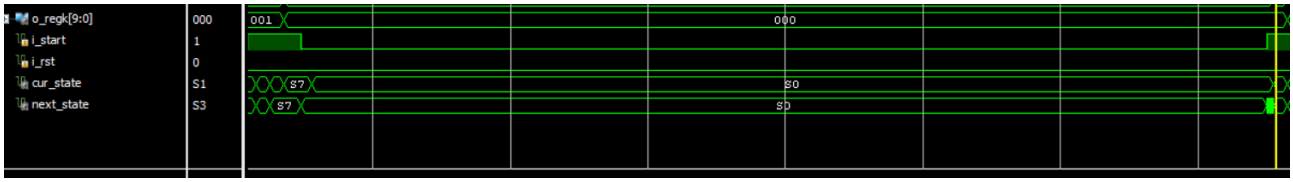


Immagine 3.2.3.1: Test di più elaborazione successive

3.2.4 Credibilità a zero

Questo test verifica il caso in cui ci sono sufficienti dati raccolti non validi (ovvero valore pari a zero) per far diminuire la credibilità fino al valore zero, per poi rimanerci (senza problemi di underflow).

Immagine 3.2.4.1: Appena viene avviato il processo la credibilità passa dal valore di default (0) al valore massimo (31), essendo il primo dato un numero diverso da 0. Successivamente essa continua a calare fino a raggiungere il valore zero e non scendendo ulteriormente.

3.2.5 Altri test

Oltre a quelli descritti sopra ci sono stati molti altri test in funzionamento normale e non della macchina, passati tutti senza alcun problema.
Per esempio, un caso non discusso precedentemente ma che viene testato è quando l'input della macchina comincia con una serie di 0, in questo caso il comportamento dato dalla specifica impone che venga copiato il dato sulla memoria (quindi 0) mettendo come byte di credibilità un altro 0.
Tutti i test fatti comprendevano dei reset durante l'elaborazione, o più elaborazioni successive, con anche input di grandi dimensioni e svariati scenari.

4. Conclusioni

Il progetto ha dimostrato la sua validità rispondendo in modo efficace e veloce a tutte le prove effettuate, sia nella fase di pre-sintesi che in quella di post-sintesi. Il sistema sviluppato si è rivelato robusto e capace di gestire situazioni complesse, mantenendo sempre un comportamento conforme alle specifiche. La progettazione accurata del datapath e della macchina a stati finiti di tipo Moore ha permesso di ottenere un modulo hardware che non solo soddisfa i requisiti funzionali, ma che è anche efficiente dal punto di vista temporale, con un margine di sicurezza di 15,137 ns su un periodo di clock di 20 ns, il che suggerisce la possibilità di operare a una frequenza massima di circa 205 MHz. Le simulazioni eseguite hanno coperto una vasta gamma di casi, inclusi corner cases come il reset durante l'esecuzione e l'elaborazione multipla, evidenziando l'affidabilità del sistema in condizioni operative reali. La gestione intelligente dei dati non validi, con il decremento progressivo della credibilità associata ai valori sostitutivi, rappresenta un ulteriore punto di forza del progetto, garantendo l'integrità dei dati elaborati. Complessivamente, il modulo ha dimostrato di essere una soluzione solida e ottimizzata per l'elaborazione dei dati sensoriali, mantenendosi affidabile anche con dati potenzialmente errati.