PROVA FINALE

Progetto di Reti Logiche

Agostino Contemi	**Federico Consorte**
986325	982973
10863645	10825301

Anno Accademico 2023/2024

Professore: William Fornaciari

Sommario

1. Introduzione

a. Specifica

Si vuole descrivere, attraverso il linguaggio VHDL, un componente hardware che operando sulla memoria legge dei valori agli indirizzi pari e gli assegna un valore di credibilità all'indirizzo successivo.
Se il valore letto è '0' questo viene interpretato come valore non specificato, quindi viene sovrascritto con l'ultimo valore letto.
La credibilità di ogni valore indica la distanza di questo dall'ultimo valore letto assegnato.
Quindi ogni valore assegnato avrà credibilità pari a '31', successivamente per ogni valore non assegnato la credibilità sarà decrementata di 1.
La credibilità non può scendere sotto lo '0', dunque i valori non assegnati con distanza maggiore di 31 avranno tutti credibilità '0'. Nel caso in cui il primo valore letto non sia specificato si sottintende come ultimo valore assegnato letto lo ‘0’.

b. Controllo del componente

Prima di ricevere il segnale di reset il comportamento del componente non è specificato.
Ricevuto il segnare di reset il componente attende il segnale di start per dare inizio alla elaborazione.
Alla fine di questa il componente restituisce un segnale done che indica il termine delle operazioni.
Fintanto ché il segnale di done e alto il componente ignora altri segnali di start, quindi prima di iniziare
nuovamente si deve o dare un segnale di reset o si deve attendere che il segnale di done venga abbassato

c. Organizzazione della memoria

Si considera punto di inizio della memoria l'indirizzo fornito al componente (i_add). Questo viene considerato "indirizzo pari".

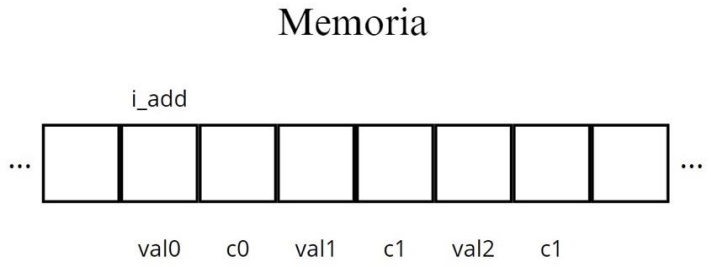


Figure 1: La memoria legge i valori val e aggiorna il valore di credibilità nella casella successiva

d. Esempio

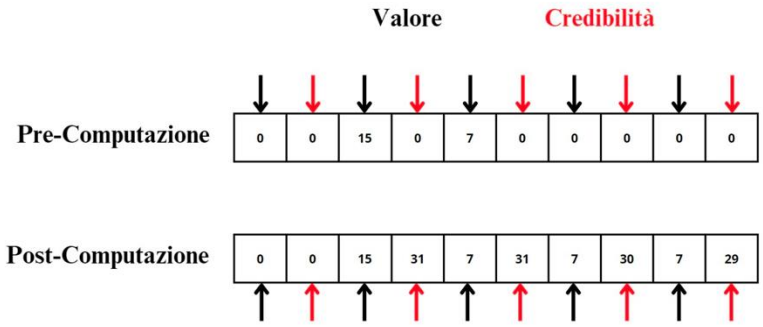
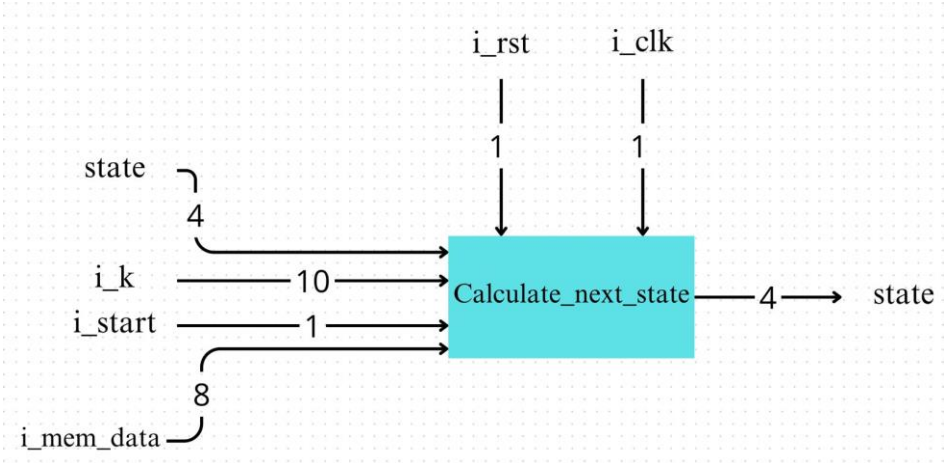


Figure 2: Memoria prima e dopo la computazione

2. Architettura

a. Calculate Next State



I numeri sulle frecce indicano quanti bit vengono trasmessi

Calculate Next State è il processo che occupa della gestione dello stato della macchina. Questa è composta da 9 stati.

- RESET: Dopo aver ricevuto il segnale di reset la macchina resetta tutti i valori a 0.
- INIT: Dopo aver ricevuto il segnale di start la macchina inizializza tutti i valori prendendo i dati da quelli forniti e si prepara alla prima lettura.
- PREREAD: La macchina sincronizza i tempi per la lettura della memoria con gli altri processi.
- READ: La macchina ha letto il dato ed ora sceglie se deve sovrascriverlo (andando nello stato PREFWRITE) o passare direttamente alla scrittura della credibilità (andando nello stato PRESWRITE).
- PREFWRITE: La macchina prepara per la sovrascrittura del valore letto.
- FWRITE: La macchina ha sovrascritto il valore in memoria
- PRESWRITE: La macchina prepara per la scrittura della credibilità.
- SWRITE: La macchina ha scritto il valore di credibilità in memoria e sceglie se terminare oppure ripetere per i prossimi valori.
- FINISH: La macchina termina la elaborazione, può ripartire solo se riceve il segnale di reset.

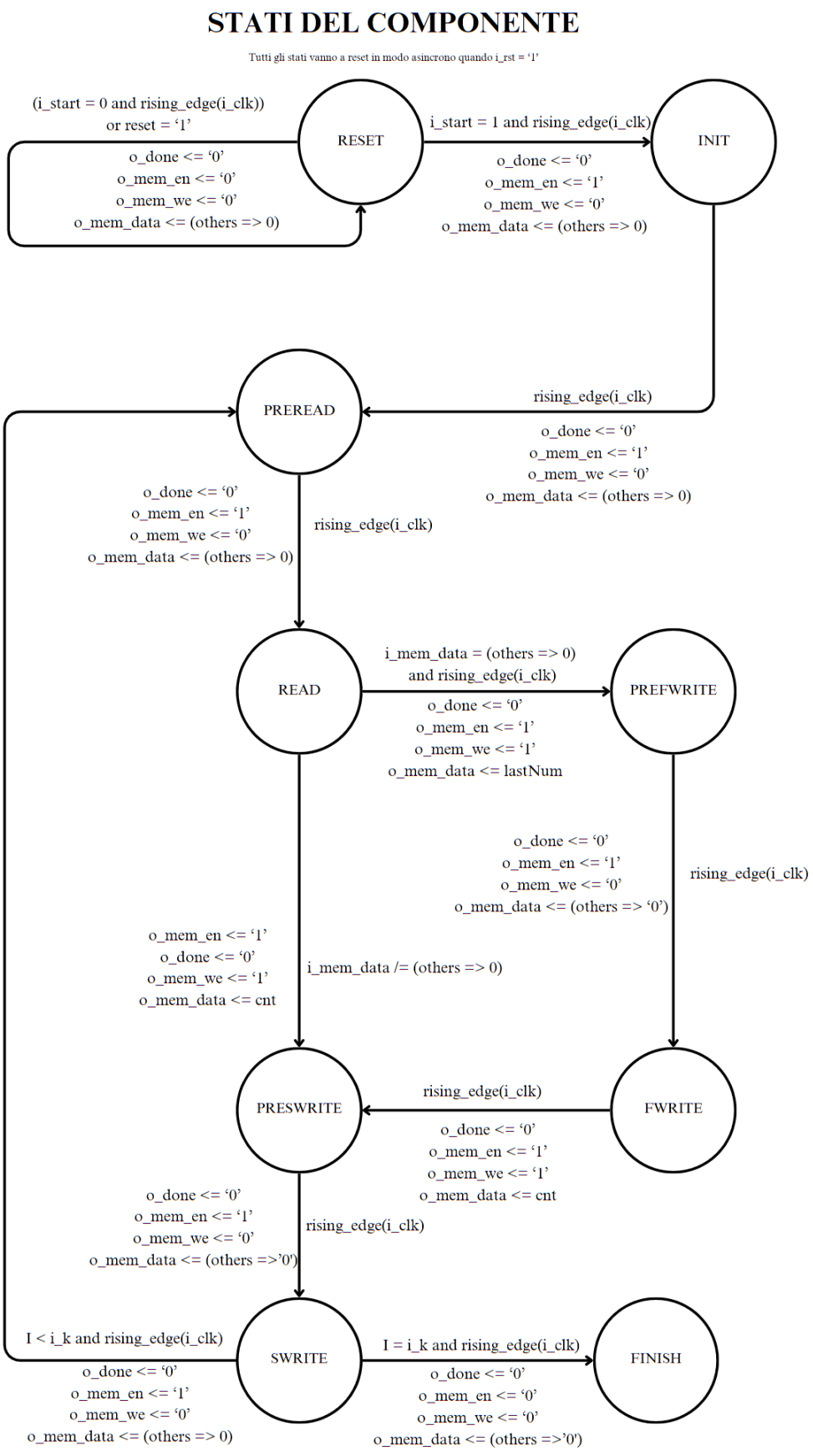
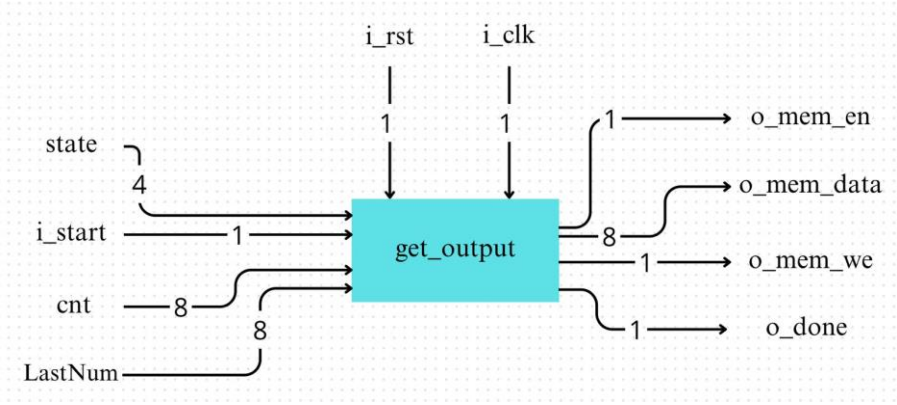


Figure 3: Diagramma degli stati

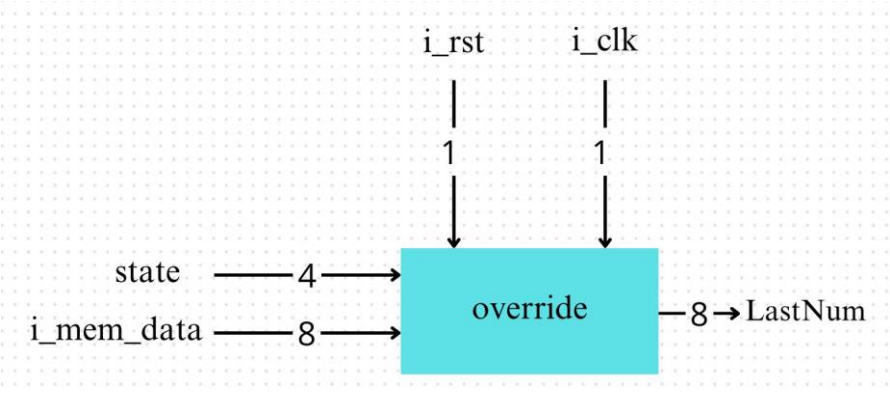
b. Get Output



Get Output è il processo che si occupa di impostare i segnali di output o_mem_en, o_mem_we, o_mem_data ed o_done.
La modifica dipende dallo stato in cui si trova la macchina.
Usa il segnale di start per impostare o_done a 0.
Prende come dati lastNum e cnt, rispettivamente l'ultimo numero valido letto e il valore di credibilità da scrivere.
Prende lo stato per scegliere se in scrittura (o_mem_data) ci sarà lastNum o cnt.

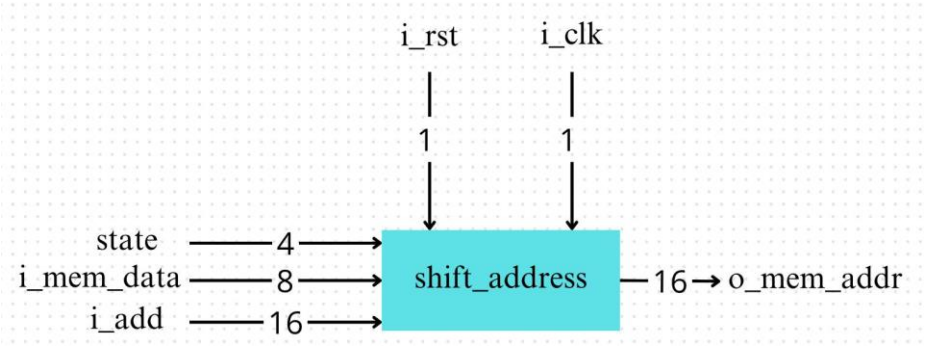
Se lo stato è FINISH allora imposta o_done a 1, dopo che il segnale di start è stato portato a 0.

c. Override



Prende in input lo stato e i_mem_data
Override è il processo che si occupa di gestire il segnale lastNum che contiene l'ultimo valore valido letto. Durante lo stato INIT, inizializza il segnale a 0. Mentre nello stato READ se ha letto un valore assegnato aggiorna lastNum con i_mem_data.

d. Shift Address



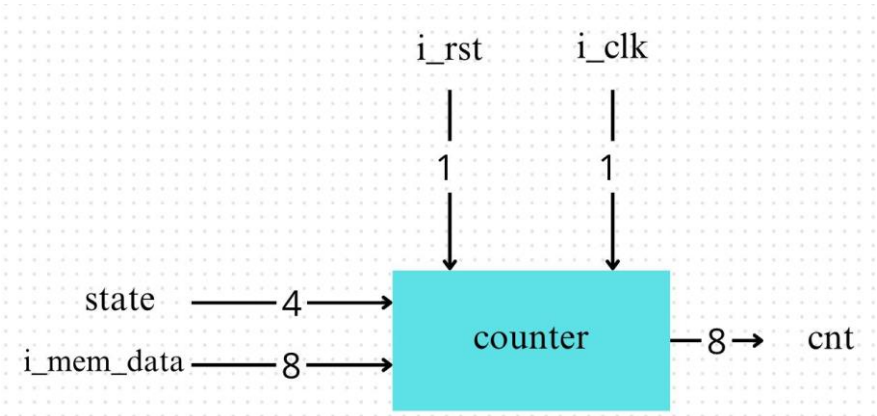
Prende come input i_add, i_mem_data e lo stato.

Questo componente si occupa di gestire il punto in memoria in cui lavoriamo. I_add serve per sapere da dove iniziamo a lavorare, viene copiato quando lo stato è INIT.

Se lo stato è FWRITE o SWRITE, incrementa il valore salvato di 1, perché nel passo successivo la macchina opererà sulla cella di memoria successiva.

Se si trova nello stato di lettura, a seconda del valore letto (i_mem_data) sceglie se avanzare o meno. Se il valore letto è 0 non avanza poiché dopo ci sarà una scrittura in quel punto altrimenti avanza perché si dovrà scrivere il valore di credibilità.

e. Counter

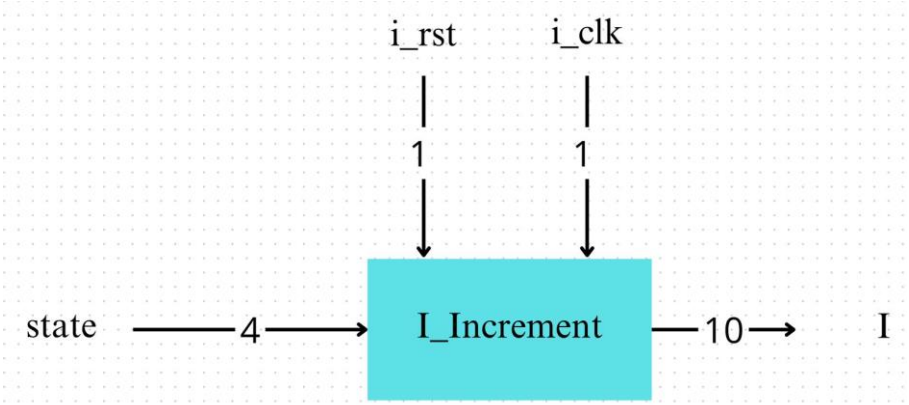


Questo blocco tiene salvato il valore di credibilità. Si comporta come un contatore con un segnale di reset.

Prende in input lo stato, se questo è INIT allora inizializza il contatore a 0, altrimenti nello stato READ “attiva” il componente. In questo modo anche se leggiamo un valore sbagliato (dato che teniamo la memoria sempre accesa) il componente ignora il valore.

Prende in input il valore letto e capisce se deve decrementare o resettare a 31.

f. I Increment



I_Increment è il processo che conta il numero di parole lette per segnalare quando il sistema deve terminare.
Prende in input lo stato, per scegliere quale azione compiere.
In INIT viene inizializzato a 0 e viene incrementato di 1 in READ.
In output espone il valore del contatore, rappresentato nel codice come il segnale I.

g. Interconnessione tra processi

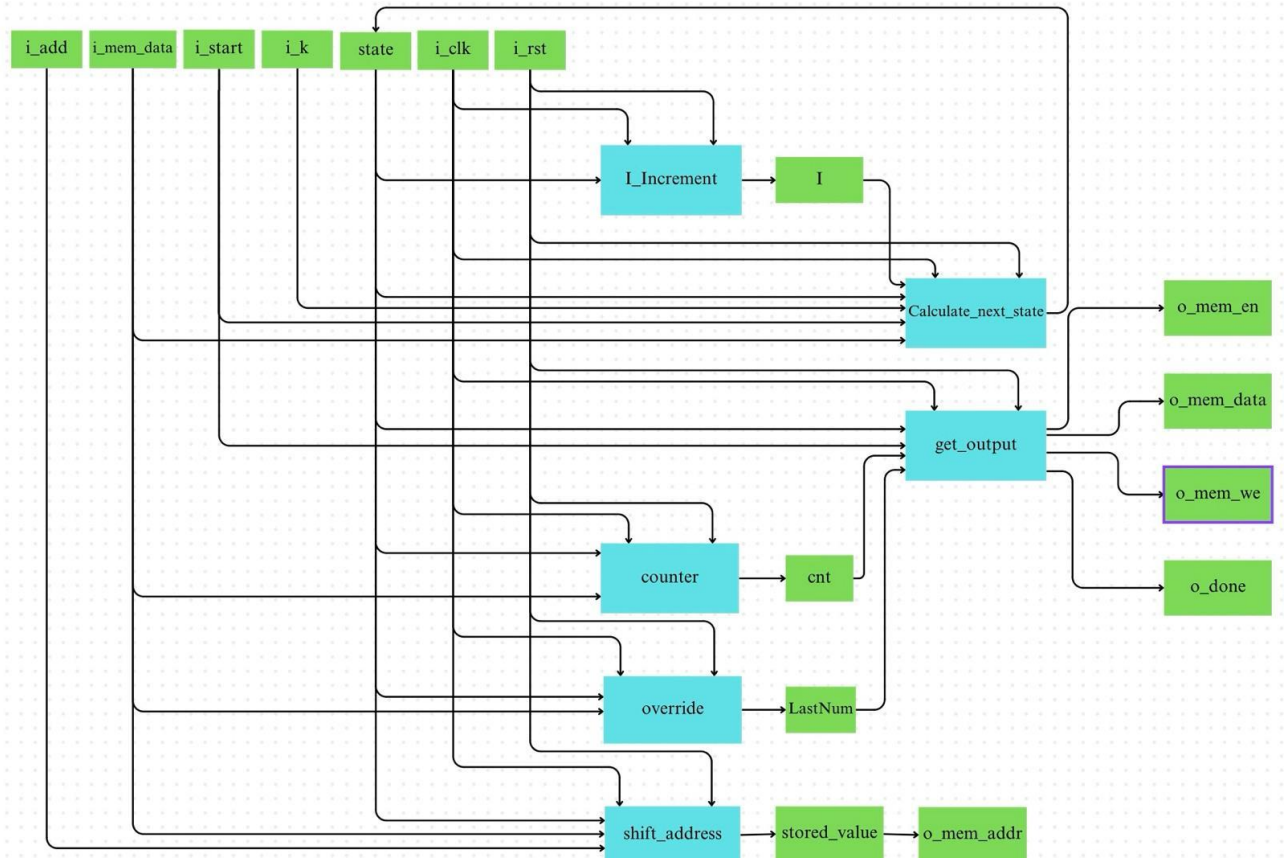


Figure 4: Schematizzazione dei segnali

I blocchi verdi in alto a sinistra (tranne state) sono i segnali che vengono ricevuti in input.
I blocchi verdi sulla parte destra sono invece i segnali che vengono dati in output per la gestione della memoria e la segnalazione di terminazione esecuzione.
Distinguiamo chiaramente i componenti funzionali del progetto, indicandoli con il colore celeste.

Essendo tutti i blocchi sincroni prendo in input il segnale di clock. Dato che i valori salvati dopo il reset sono sempre gli stessi, i componenti prendono anche il segnale di reset.

3. Risultati sperimentali

a. Risultati di sintesi

State	New Encoding	Previous Encoding
reset	0000	0000
init	0001	0001
preread	0010	0010
read	0011	0011
prefwrite	0100	0100
fwrite	0101	0101
preswrite	0110	0110
swrite	0111	0111
iSTATE	1000	1000

Figure 5.1: Codifica degli stati

Mappatura degli stati in formato binario. Notiamo che la sintesi non ha cambiato la codifica binaria di ogni stato. Questo perché il componente Calculate_next_state è equivalente ad un contatore preceduto da un blocco logico che genera il segnale di incremento soltanto se sono verificate le condizioni necessarie.

```
Detailed RTL Component Info :
+---Adders :
           2 Input   16 Bit       Adders := 1
           2 Input   10 Bit       Adders := 1
           2 Input    8 Bit       Adders := 1
+---Registers :
                     16 Bit    Registers := 1
                     10 Bit    Registers := 1
                      8 Bit    Registers := 3
                      1 Bit    Registers := 3
+---Muxes :
           5 Input   16 Bit        Muxes := 1
           2 Input    8 Bit        Muxes := 1
           9 Input    8 Bit        Muxes := 1
           3 Input    8 Bit        Muxes := 1
           8 Input    4 Bit        Muxes := 1
           2 Input    4 Bit        Muxes := 2
           2 Input    1 Bit        Muxes := 4
           9 Input    1 Bit        Muxes := 4
           3 Input    1 Bit        Muxes := 1
           5 Input    1 Bit        Muxes := 1
```

```
Report BlackBoxes:
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+

Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     5|
|3     |LUT1   |     2|
|4     |LUT2   |     3|
|5     |LUT3   |     5|
|6     |LUT4   |    29|
|7     |LUT5   |    11|
|8     |LUT6   |    17|
|9     |FDCE   |    12|
|10    |FDRE   |    45|
|11    |IBUF   |    37|
|12    |OBUF   |    27|
+------+-------+------+
```

Figure 5.2: Report e utilizzazione della sintesi

Notiamo che non viene usata nessuna blackbox, questo perché l'intero progetto è sviluppato in un singolo componente, i componenti vengono simulati con i processi. Non sono usati componenti da librerie esterne. Indica quindi, che si ha piena visibilità e controllo di ogni parte del progetto. In questo caso si può fare una migliore ottimizzazione del codice, dato che appunto non ci sono parti nascoste.

Adder:

Viene usato un Adder a 2 input da 16 bit ciascuno per il calcolo di o_mem_addr in Shift_Address.

Viene un Adder a 2 input da 10 bit ciascuno per calcolare se si è arrivati alla fine dell'input di I_increment. Infatti, il segnale K è da 10 bit!

Viene usato un Adder a 2 input da 8 bit ciascuno per calcolare il valore della parola da scrivere in Override che esegue il calcolo è da il valore al blocco Get_output che al momento corretto lo scriverà in memoria.

Registri:

1 registro da 16 bit per salvare l'indirizzo su cui si sta operando.

1 registro da 10 bit per salvare su quante parole si è già operato.

3 registri da 8 bit, 1 per salvare l'ultimo numero letto con credibilità maggiore di 0, 1 per tenere pronto il valore da scrivere in memoria e l'ultimo per salvare il valore di credibilità da scrivere.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	55	0	0	134600	0.04
LUT as Logic	55	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	57	0	0	269200	0.02
Register as Flip Flop	57	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figure 5.3: Report e utilizzazione della sintesi

Tutte le Look Up Table sono usate per la logica.
I registri usati sono tutti di tipo Flip Flop. Questo perché tutta la memoria è sincrona al clock e quindi non sorge la necessità di usare Latch.

b. Test: default
Caso di test fornito nella documentazione.

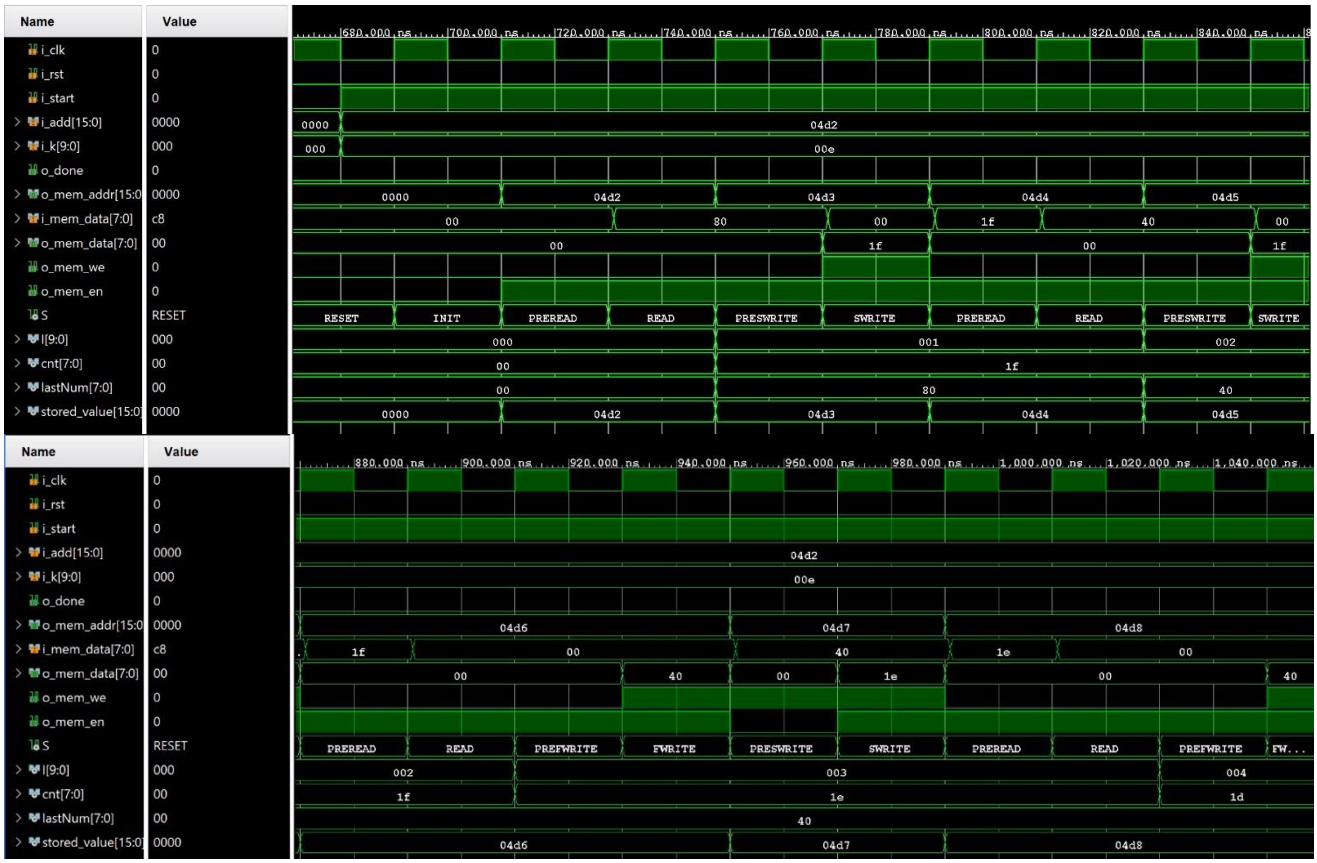


Figure 6 e 7: Analisi dei segnali durante la simulazione

Dalla visione dei segnali vediamo che tutti i processi, avvengono sul fronte di salita del clock come richiesto dalla specifica.
Notiamo che i_mem_data cambia il valore sempre poco dopo il segnale di clock, questo perché anche la memoria è sincrona con il clock e restituisce il valore in uscita dopo un breve lasso di tempo.
Il componente progettato non è affetto dal suddetto comportamento poiché la memoria viene impostata per leggere nello stato PREREAD, ma la lettura di i_mem_data avviene al ciclo successivo nello stato READ.
Spostando l'attenzione su o_mem_en si può notare che da dopo lo stato di inizializzazione il suo valore è sempre 1. Questo perché, quando la memoria è in scrittura ha sempre il valore corretto da scrivere, mentre il resto del tempo è in lettura. Se la memoria ci sta restituendo un valore non utile alla computazione questo viene ignorato.

c. Test: begin with zero
Caso di test nel quale la prima parola letta è 0, con questo test viene assicurato la corretta inizializzazione di lastNum e cnt. (Test passato)

d. Test: not zero in cred
Caso di test in cui nel valore di credibilità non è presente il valore 0, con questo caso viene testato che il valore di credibilità non viene letto. (Test passato)

e. Test: double value
Caso di test nel quale sono presenti 2 parole identiche consecutive, con questo test viene assicurato che la seconda parola identica venga interpretata come una nuova parola e non come la ripetizione della precedente. (Test passato)

f. Test: only 1 word

Caso di test nel quale è presente una sola parola, con questo test viene assicurato il corretto funzionamento del meccanismo di terminazione della computazione. (Test passato)

4. Conclusioni

Il progetto è stato sviluppato tutto in un unico componente, rendendo i componenti dei semplici processi all'interno del componente.
Sarebbe comunque sempre possibile incapsulare ogni processo in un componente a sé e poi importarle nel componente principale.

Periodo di clock: 20ns.
La simulazione pre sintesi con il test-bench fornito impiega 2130ns prima di portare il segnale di o_done a 1.
La simulazione funzionale post sintesi con il test-bench fornito impiega 2130,1ns prima di portare il segnale di o_done a 1.

NB: Quando il programma viene eseguito, se l'input è sufficientemente lungo, la simulazione viene interrotta prima che il componente abbia effettivamente terminato. Per ovviare a questo problema dalla console eseguivamo il comando "run all", per eseguire il test fino alla fine.

Le ottimizzazioni eseguite sono relative agli stati. In prima battitura, infatti, la macchina contava molti più stati con il compito di sincronizzare e spegnere ed accendere la memoria. Abbiamo notato che i tempi di sincronizzazione potevano essere accorpati in stati già presenti, e che la memoria poteva restare sempre accesa a patto di fare attenzione che non scrivesse quando non doveva e di ignorare i valori letti se non servivano.