Prova Finale (Progetto di Reti Logiche)

Prof. William Fornaciari – Anno 2023/2024

Giacomo Bossi (Codice Persona 10766073)

Apollonio Marco (Codice Persona 10764083)

Sommario

[Introduzione 2](#_heading=h.gjdgxs)

[Scopo del progetto 2](#_heading=h.30j0zll)

[Design componente realizzato 4](#_heading=h.1fob9te)

[Descrizioni dei componenti sequenziali 5](#_heading=h.3znysh7)

[Controller (1) 5](#_heading=h.2et92p0)

[LastValidReading (LVR) (2) 10](#_heading=h.3dy6vkm)

[Credibility (3) 10](#_heading=h.1t3h5sf)

[CounterADD (4) 10](#_heading=h.4d34og8)

[Report sulla Sintesi dei componenti 12](#_heading=h.2s8eyo1)

[Credibility 12](#_heading=h.17dp8vu)

[LastValidReading 14](#_heading=h.3rdcrjn)

[CounterAdd 17](#_heading=h.26in1rg)

[CounterK 18](#_heading=h.lnxbz9)

[Controller 20](#_heading=h.35nkun2)

[Risultati sui Test Bench 22](#_heading=h.1ksv4uv)

[Test Bench D’esempio 22](#_heading=h.44sinio)

[Caso W = 0 nelle prime 2 posizioni 22](#_heading=h.2jxsxqh)

[Esecuzioni Multiple 23](#_heading=h.z337ya)

[Credibilità raggiunge lo 0 24](#_heading=h.3j2qqm3)

[Reset Asincrono 25](#_heading=h.1y810tw)

[Conclusioni 26](#_heading=h.1pxezwc)

# Introduzione

## Scopo del progetto

L’obiettivo è di realizzare un componente in grado di interfacciarsi con una memoria RAM e agire su di essa come richiesto dalla specifica.

Il componente deve accedere ad un insieme di indirizzi determinato da alcuni parametri che riceve in input e modificarli in modo opportuno.

La memoria RAM è strutturata con indirizzamento al Byte, le operazioni di lettura e scrittura avvengono sul fronte di salita.

La memoria ha un’ interfaccia composta da 5 segnali:

-o\_mem\_en : se il segnale è alto sul fronte di salita, allora avverrà la lettura del byte all’indirizzo corrispondente.

-o\_mem\_we : se sia o\_mem\_en sia questo segnale sono alti sul fronte di salita, allora il byte all’indirizzo indicato verrà sovrascritto con la parola presente su o\_mem\_data.

-o\_mem\_data: è un vettore di 8 bit che contiene la parola da scrivere

-i\_mem\_data : è un vettore di 8 bit che contiene il valore dei byte letti, è stabile con un ritardo di 2 ns rispetto al fronte, rimane stabile anche nei cicli di clock successivi se non vengono effettuate altre letture.

-o\_mem\_addr: è un vettore di 16 bit che contiene l’indirizzo al quale accedere durante le operazioni di lettura/scrittura.

I byte della memoria vanno interpretati a coppie, il primo dei due contiene un dato numerico (successivamente verrà indicato come WordByte), il byte successivo contiene un valore associato al WordByte che indica la sua “credibilità”.

Gli indirizzi sul quale agisce il componente devono essere specificati in input prima dell’inizio dell’elaborazione.

Tramite l’input ADD si fornisce l’indirizzo base da cui iniziare e tramite K si indica il numero di WordByte da analizzare.

Il componente si muove in automatico sugli indirizzi da ADD a ADD+ 2\*K -1.

L’obiettivo del componente è di rimuovere eventuali “buchi” nella memoria e riempire i byte di credibilità.

I “buchi” sono quei WordByte che contengono zero, il componente deve sovrascrivere agli zeri l’ ultimo valore diverso da zero incontrato.

I Valori di credibilità sono tutti inizializzati a 0.

Successivamente ci si riferirà a WordByte che non contengono zero come “validi”.

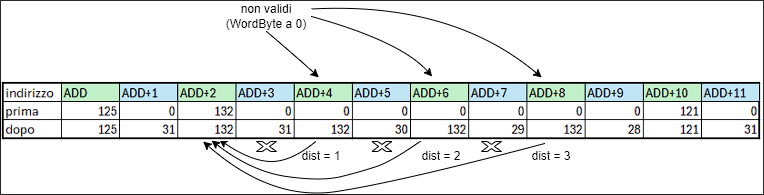
La credibilità associata ai WordByte è definita come segue:

31 se il WordByte è valido

31 – Dist se il WordByte non è valido e Dist ≤ 31

0 altrimenti

Con Dist si indica la distanza del WordByte non valido dal precedente WordByte valido in numero di byte di credibilità tra i due.



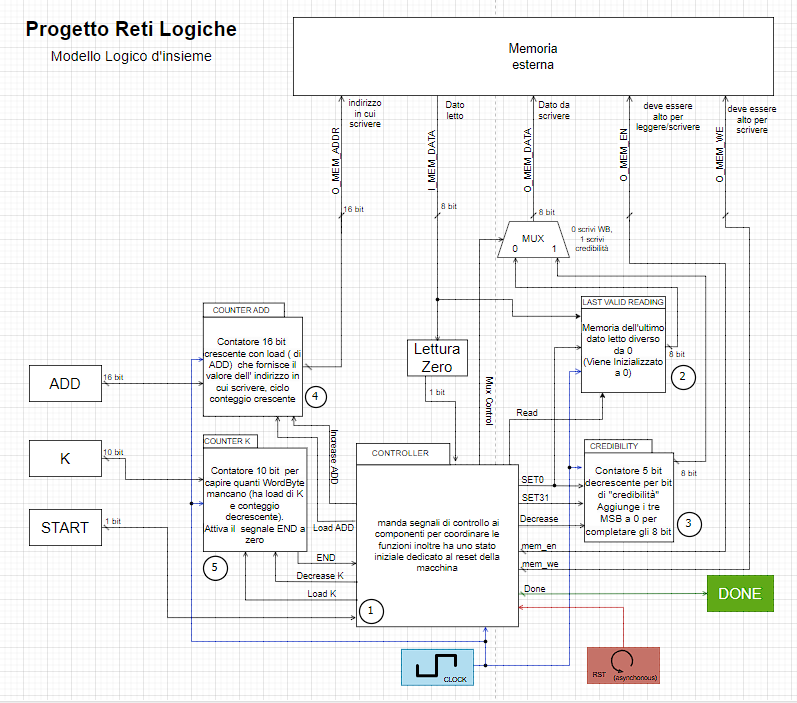
Un semplice esempio di funzionamento del componente, in verde sono i WordByte in azzurro i byte di credibilità.

Sono presenti segnali di interfaccia per l’utilizzo corretto del componente, questi sono:

* Reset: è un segnale asincrono che serve a resettare il componente al primo utilizzo.
* Done: è un segnale di output del componente e viene alzato soltanto a elaborazione terminata.
* Start: serve a iniziare l’elaborazione, va alzato solo dopo che gli input ADD e K sono stabili e mantenuto alto finché non viene alzato Done.

Il corretto utilizzo dei segnali di interfaccia deve essere garantito dall’utilizzatore esterno.

## Design componente realizzato



La struttura di alto livello si compone di 5 grandi sotto-componenti sequenziali (indicati sul disegno dell’architettura tramite numeri) e due semplici sotto-componenti combinatori.

I componenti combinatori sono un multiplexer che serve per scegliere cosa scrivere nella RAM (Word oppure credibilità), l’altro componente combinatorio è LetturaZero che tramite il NOR fra tutti i bit della parola letta dalla RAM segnala al controllore se è stato letto uno WordByte non valido (tutti zeri).

In ogni stato che non comprende una lettura dalla memoria il segnale LetturaZero viene ignorato.

## Descrizioni dei componenti sequenziali

### Controller (1)

Il controller è il componente più importante dell'intera architettura perché gestisce tutti gli altri componenti e coordina le operazioni che essi compiono.

Il controllore è realizzato come macchina a stati di Moore nella quale ad ogni stato è assegnata una “operazione”. In un certo stato i suoi segnali di uscita sono attivi/spenti in modo da realizzare l’operazione associata allo stato eccitando opportunamente sia gli altri componenti sequenziali sia la memoria RAM esterna.

Il controllore cambia stato in base a 4 ingressi differenti di cui 2 sono forniti dall’esterno, gli altri 2 sono interni e trasparenti dal un punto di vista esterno:

Essi sono:

* Reset : è l’unico ingresso asincrono, serve per inizializzare correttamente il componente riportandolo ad uno stato default.
* Start : serve a segnalare che si vuole iniziare una elaborazione, va mantenuto alto fino a quando viene segnalata la fine dell’elaborazione tramite il segnale Done.
* END: segnale interno, viene alzato da CounterK quando sono state elaborate K coppie di bit, segnalando così di dover fermare l’elaborazione. Tramite END il controllore si sposta in uno stato di terminazione dove alza l’uscita Done e si prepara ad una eventuale elaborazione successiva.
* LetturaZero: segnale interno, la sua generazione è già stata discussa in precedenza. Tramite il segnale LetturaZero il controllore può prendere decisioni sul flusso di operazioni da eseguire nei cicli di clock successivi ad una lettura della RAM. Permette di decidere se sovrascrivere il WordByte o meno e se la parola letta va salvata come ultima valida.

La concentrazione maggiore di segnali si trova sulle uscite del controllore che sono ben 11 e servono per interfacciarsi con la RAM esterna e per attivare le funzioni degli altri componenti.

Le uscite del controllore sono le seguenti:

* Load K / Load ADD : segnalano ai contatori K e ADD che i valori di K e ADD sono stabili e possono essere caricati sui rispettivi contatori. Sono rappresentati come segnali separati nel disegno dell'architettura ma nella sintesi sono stati unificati perché sono attivi sempre insieme.
* Decrease K : dopo aver completato l’elaborazione di ogni coppia di Byte (Word e credibilità) viene alzato il segnale Decrease K che fa da enable per il conteggio del CounterK decrementando così il valore salvato di 1.
* Increase ADD: è un segnale simile a Decrease K perché anche Increase ADD fa da enable per un contatore (CounterADD) ma viene usato in momenti diversi. Viene attivato in due momenti, per spostarsi dall’ indirizzo del WordByte all’indirizzo del Byte di credibilità ad esso associato e per spostarsi dal Byte di credibilità al successivo WordByte.
* Mux Control: intuitivamente ha il ruolo di segnale di controllo del Multiplexer. Serve per permettere la scrittura della RAM da due sorgenti diverse , Credibility e LastValidReading (LVR). Per il valore 0 del selettore è stato assegnato l’ultimo valore letto diverso da 0 contenuto nel LastValidReading, mentre per il valore del selettore 1 ho il valore di Credibilità.
* O\_mem\_en : è uno dei due segnali che si interfacciano direttamente con la RAM e permettono di usarla. È attivo in tutti gli stati che richiedono di leggere o scrivere sulla memoria.
* O\_mem\_we: è il secondo segnale di interfaccia con la RAM. Questo segnale alto, insieme a O\_mem\_en altrettanto alto, segnala alla memoria l’intenzione di effettuare una scrittura. Se non serve scrivere questo segnale è basso.
* SET0 : è un segnale che serve al reset dei componenti Credibilità e LVR e viene alzato sempre e solo prima di ogni elaborazione quando il controllore è nello stato di attesa di START. Permette di inizializzare Credibility e LVR a 0.
* SET31 : nonostante la somiglianza con il segnale SET0 serve soltanto al componente Credibility. Viene attivato ogni volta che viene letto un WordByte valido per resettare il valore della credibilità a 31.Il segnale viene attivato ogni lettura valida indipendentemente dal fatto che ci possa essere già salvato 31.
* Decrease : è un segnale complementare a SET31, infatti serve solo a Credibility. Viene attivato quando viene letto un WordByte non valido per abbassare il valore della credibilità ed esso associato. Funziona in modo analogo a DecreaseK e Increase ADD nel senso che fa da enable per il ciclo di conteggio del contatore. Viene attivato ogni lettura non valida indipendentemente dal fatto che Credibility possa contenere 0, è credibility stesso che blocca il suo conteggio quando si trova a 0.
* Read : serve alla LVR per memorizzare I WordByte validi letti, come altri segnali fa da enable alla funzione del componente. In questo caso non si tratta di conteggio ma di salvataggio. Per motivi legati alla sincronizzazione la memorizzazione avviene nei cicli di clock successivi ai cicli di lettura, solo in caso di lettura valida.
* Done : è l’unica uscita dal contatore ed essere osservabile dall’esterno, infatti serve a segnalare la fine della elaborazione.

Logica di funzionamento del Controller

Il controllore è dotato di 10 stati (da S0 a S9).

Lo stato s0 serve per il reset, ogni stato può cambiare in qualunque momento e passare a s0 se viene fornito il segnale di reset (frecce omesse nel diagramma perché transizioni asincrone).

La macchina rimane in s0 fino alla fine del ciclo di clock per poi passare in s1 dove alza il segnale SET0 e rimane in s1 finché non viene dato il segnale start per iniziare.

Quando inizia l'elaborazione per prima cosa la macchina passa per lo stato s2 che attiva LoadK e LoadADD.

Se K è zero si termina subito andando in S9 altrimenti si passa a S3.

In S3 viene letto un WordByte dalla memoria e tramite LetturaZero si prosegue o in S4 o in S5.

Se la macchina si trova in S4 allora è stato letto un dato non valido, quindi sovrascrivo il dato con il valore memorizzato nella LVR e decrementato la credibilità .

Il ciclo successivo la macchina passa in S6 dove viene aumentato ADD per poter successivamente scrivere la credibilità.

Se invece da S3 la macchina passa in S5 allora è stato letto un valore valido, il dato non va sovrascritto ma viene resettata la credibilità a 31 e aumentato ADD. Inoltre, viene salvato il dato letto nella LVR.

Il salvataggio avviene nel ciclo successivo alla lettura perché, non leggendo né scrivendo sulla RAM, il dato letto al ciclo precedente rimane sull’uscita.

Dopo S5 e S6 i flussi si ricongiungono in S7 dove viene scritto il Byte di credibilità e decrementato K. Se K si azzera allora l’elaborazione deve terminare e si va in S9, altrimenti si passa a S8 che incrementa ADD per puntare al WordByte successivo e poi ricominciare da S3.

In S9 la macchina alza l’uscita Done per segnalare l’elaborazione terminata e aspetta che si abbassi START.

Dopo che Start si abbassa la macchina va in S1 dove tramite SET0 resetta la LVR e la credibilità e aspetta una successiva elaborazione.

Da notare che tramite la transizione da S2 a s9 la macchina gestisce anche il caso non richiesto K = 0, questo perché abbiamo ritenuto che fosse meglio lasciare il minor numero di casi non definiti per non causare side effect sui dati della memoria.

Dopo aver scritto l’intera macchina a stati abbiamo rinominato gli stati in modo da dare una descrizione di cosa fa lo stato.

Gli stati rinominati sono :

S0 → Status\_Init

S1 → Status\_Start

S2 → Status\_Load

S3 → Status\_Read\_W

S4 → Status\_Read\_0

S5 → Status\_Read\_Not\_0

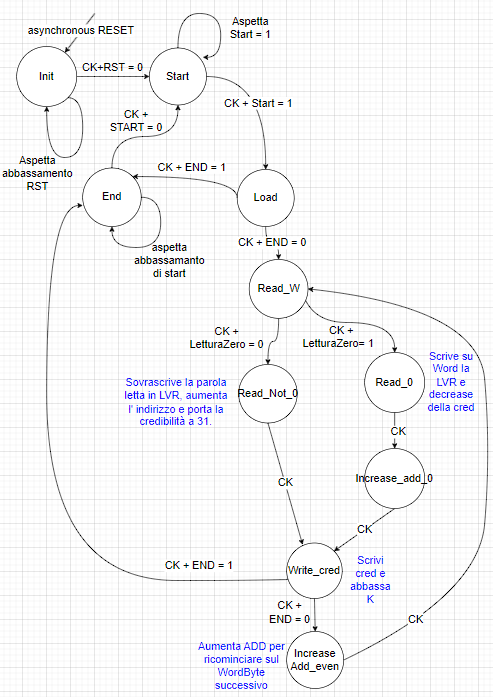
S6 → Status\_Increase\_Add\_0

S7 → Status\_Write\_Credibility

S8 → Status\_Increase\_Add\_Even

S9 →Status\_End

Il diagramma degli stati corrispondente è:



Il caso in cui si presentano dei WordByte non validi all’inizio è gestito dalla macchina senza aggiungere stati dedicati. Questo è possibile perché sia LVR che la credibilità sono inizializzate a 0 e rimangono tali finché non avviene una lettura valida.

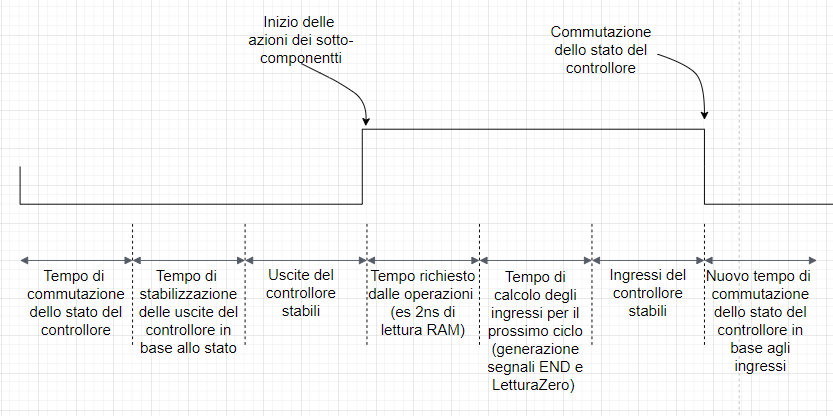
In questo modo per ogni lettura non valida iniziale viene preso il ramo con Read\_0 nel quale il Wordbyte (che contiene zero) viene sovrascritto con LVR (anche essa a 0), la credibilità viene decrementata ma è già a zero per cui non cambia.

Alla prima lettura valida la macchina comincia a funzionare nel modo “standard” andando in Read\_not\_0, salvando la parola nella LVR e portando la credibilità a 31.

#### Sincronizzazione tra i componenti interni e il controllore

Per garantire la corretta sincronizzazione tra controllore, memoria RAM e altri componenti abbiamo deciso di realizzare tutti i componenti secondari in modo che commutino stato sul fronte di salita uniformandoli con la RAM.

Il controllore invece commuta stato sul fronte di discesa.



Questa scelta permette di rimuovere vari stati “cuscinetto” che servono solo a far passare cicli di clock per garantire che certe azioni finiscano prima di usare i loro risultati. Visto il generoso tempo di clock (20 ns), tutte le operazioni e il calcolo dei segnali riescono ad essere completati facilmente entro un semi periodo di clock.

### LastValidReading (LVR) (2)

È il sotto-componente più semplice di tutta la macchina, si tratta infatti di un semplice registro di memoria che memorizza la parola in ingresso sul fronte di salita se è attivo il segnale Read.

La sua uscita è la parola memorizzata (8 bit). È dotata inoltre di un segnale sincrono di reset, SET0, che forza la parola memorizzata a 0.

In caso siano attivi sia Read che SET0 viene data la precedenza a SET0.

Il controllore garantiva già la mutua esclusività tra i due segnali, la precedenza è stata comunque introdotta per rendere il componente più riutilizzabile.

### Credibility (3)

Si occupa di tenere traccia del valore corrente di credibilità. Viene realizzato come un contatore modulo 32 (5 bit) a conteggio decrescente. La parola salvata viene decrementata se è attivo il segnale decrease.

Sono presenti anche due segnali, SET31 e SET0 che caricano rispettivamente le parole 31 e 0.

Se la parola salvata è zero allora il conteggio viene bloccato e il segnale decrease viene ignorato fino ad un successivo SET31.

La parola in uscita è di 8 bit e viene costruita a partire dai 5 bit dello stato del contatore aggiungendo tre MSB fissi a zero.

La priorità implementata tra i segnali è SET31 -> SET0 -> Decrease,dove SET31 ha la priorità massima.

Come nel caso della LVR il controllore garantisce già la mutua esclusività fra i tre ingressi, la priorità è stata comunque implementata per motivi di robustezza e riutilizzo.

### CounterADD (4)

Si occupa di fornire alla memoria l’indirizzo al quale leggere / scrivere. È in grado di svolgere due funzioni, caricare l’indirizzo base da cui iniziare e incrementare tale indirizzo.

Se è attivo il segnale LoadADD carica il valore presente sull’ingresso ADD che è l’indirizzo base.

Ogni volta che viene attivato IncreaseADD l’indirizzo salvato viene incrementato di 1.

Per non creare possibili conflitti con la RAM il controllore garantisce che increaseADD è attivo solo in cicli in cui non avvengono né letture né scritture.

Per la gestione del valore di add è abbiamo usato la libreria std\_logic\_unsigned che implementa le somme in automatico.

CounterK (5)

Tiene traccia di quante coppie di Byte ci sono ancora da elaborare. All‘inizio dell’elaborazione tramite LoadK memorizza il valore K fornito in input.

Ogni volta che riceve il segnale DecreaseK il valore salvato viene abbassato di 1.

Quando arriva a zero attiva la sua uscita END che segnala che non ci sono più Byte da elaborare e si può terminare.

Anche CounterK come CounterADD usa la libreria std\_logic\_unsigned per le sue operazioni sul valore di K.

# Report sulla Sintesi dei componenti

Nel costruire l’intera macchina in VHDL abbiamo seguito un approccio di tipo Bottom-Up cioè realizzare prima i componenti slave e testarli separatamente.

Questo ci ha permesso, una volta costruito il controllore, di testarlo subito con tutti i componenti già realizzati e testati.

## Credibility

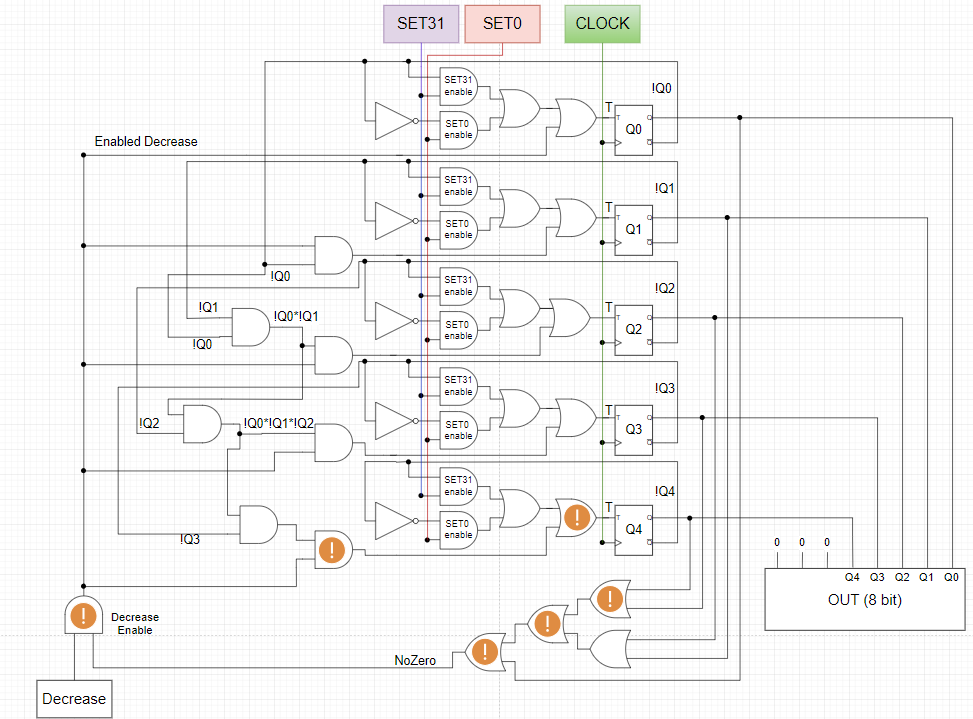
Credibility è un contatore 5 bit con tre ingressi oltre al clock e tre funzioni sul conteggio, decremento, setta a 31 e setta a 0. Abbiamo deciso di usare flip flop T perché le funzioni di settaggio risultano estremamente semplici, e il conteggio ha implementazioni standard.

Per forzare il valore salvato a 0 basta fornire in ingresso ad ogni flip flop T il proprio stato.

Infatti se è salvato un 1 esso verrà cambiato in 0 , se è già presente uno 0 rimane tale.

Analogamente per forzare la parola a 31 basta fornire in ingresso ad ogni flip flop T il proprio stato negato.

Le tre funzioni vengono realizzate in parallelo e vengono attivate dagli ingressi, tramite porte AND che fanno da enable.

Progetto di Credibility sulla carta:

La funzione di conteggio è stata realizzata con l’implementazione serie perchè utilizza meno porte logiche rispetto all’implementazione parallela e non rallenta il tempo totale di commutazione del contatore perchè il percorso critico è dato dalla funzione che verifica se è memorizzato zero.

Il percorso critico è quello segnalato in arancione.

Inoltre le porte NOT sono superflue perché realizzano !!Q = Q cioè l’ uscita dai flip-flop, sono state messe nel disegno per ridurre il numero di cavi e rendere più leggibile il disegno.

Per quanto riguarda il codice VHDL, il componente è stato realizzato con un singolo process che si attiva sul fronte di salita e fornisce ai flip flop i segnali che realizzano la funzione attivata. È stata aggiunta, rispetto alla carta, la precedenza tra i segnali.

I flip flop T sono stati realizzati separatamente e importati come component.

Abbiamo però notato che Vivado usa solo flip-flop D, infatti il T è stato sintetizzato con un D e una LookUpTable in ingresso.

Immagine che contiene testo, diagramma, linea, Carattere

Descrizione generata automaticamente

Il risultato della sintesi dell’intero componente è la seguente:

Immagine che contiene testo, diagramma, linea, Piano

Descrizione generata automaticamente

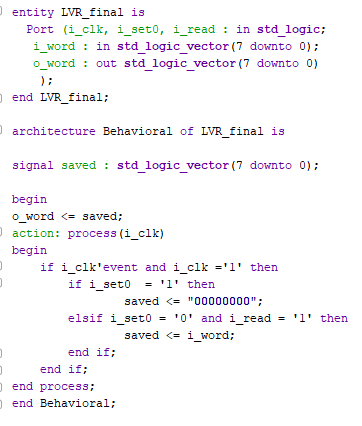
Da notare che i 5 Flip-Flop T hanno un numero di ingressi diversi, questo significa che Vivado li ha sintetizzati in modi diversi tra loro e da quello base mostrato sopra (es LUT diverse a più ingressi), probabilmente per ottimizzazioni fatte dal sintetizzatore.

Osservando i risultati della sintesi abbiamo capito che per gli altri componenti è più conveniente realizzare un progetto più semplice di alto livello e poi lasciar fare le ottimizzazioni di basso livello al sintetizzatore.

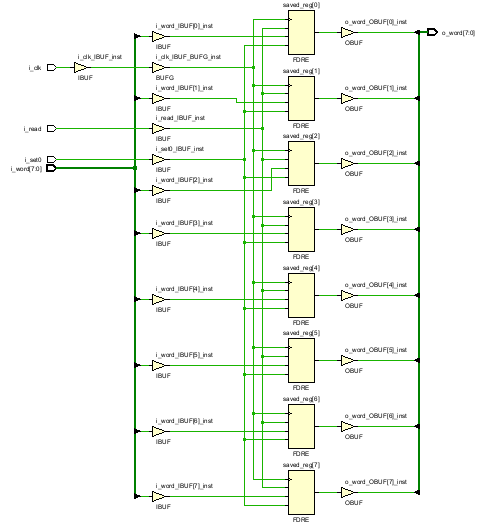
## LastValidReading

Trattandosi di un semplice registro di memoria il suo Codice VHDL risulta molto semplice.

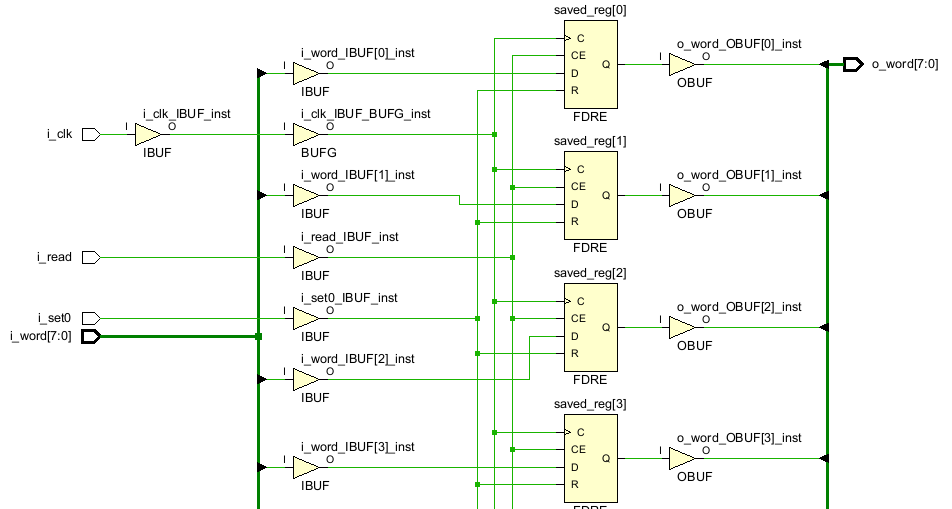
Basta un singolo processo che ogni fronte di salita, se è attivo set0 memorizza 0, altrimenti se è attivo il segnale Read memorizza la parola in ingresso.



La versione sintetizzata che ne segue, come ci si aspettava, contiene semplicemente 8 flip flop D che hanno come ingresso la parola in ingresso e SET0 che fa da reset ai flip flop.

Schematica sintetizzata:

Qui mostriamo uno zoom su 4 flip flop per leggibilità dei segnali (gli altri 4 sono collegati in modo analogo).



## CounterAdd

Il componente CounterAdd è stato pensato come una macchina a stati che evolve rispetto ai segnali di ingresso forniti dal controller.

I segnali sono:

* LoadAdd: che indica al componente che dovrà leggere il valore in ingresso i\_add
* IncreaseAdd: che indica al componente che dovrà incrementare di 1 il valore memorizzato

La macchina è strutturata in 4 stati.Immagine che contiene diagramma, testo, linea

Descrizione generata automaticamente

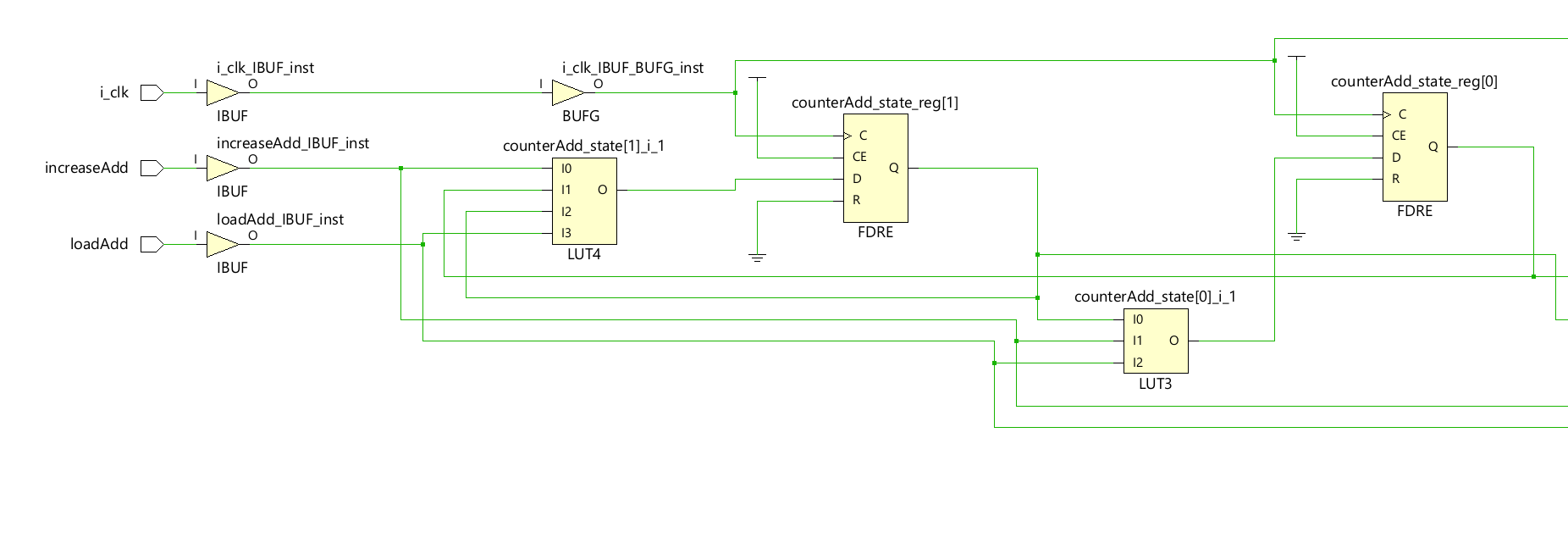
Status\_Init è lo stato di inizializzazione, nel caso in cui si verifica un errore si torna in questo stato. Appena arriva il segnale di LoadAdd si va nello stato Status\_Read\_Add.

Status\_Read\_Add memorizza il valore in ingresso i\_add in modo da venir utilizzato per eseguire gli incrementi e per associarlo al valore in uscita.

Status\_Idle è uno stato cuscinetto per quando i segnali di Load e di Increase sono a 0.

Quando arriva il segnale di IncreseAdd mi sposto in Status\_Increase\_Add eseguo l’incremento grazie alla libreria std\_logic\_unsigned semplicemente scrivendo + 1 al segnale interno.

In questo Screenshot vengono raffigurati i segnali in ingresso e i flip flop di tipo D che servono per memorizzare lo stato corrente della macchina a stati



In questo screenshot vengono rappresentati alcuni dei bit del segnale interno count che serve per memorizzare il valore i\_add ricevuto in ingresso ed eseguirne gli eventuali incrementi quando richiesto attraverso il segnale di IncreaseAdd.

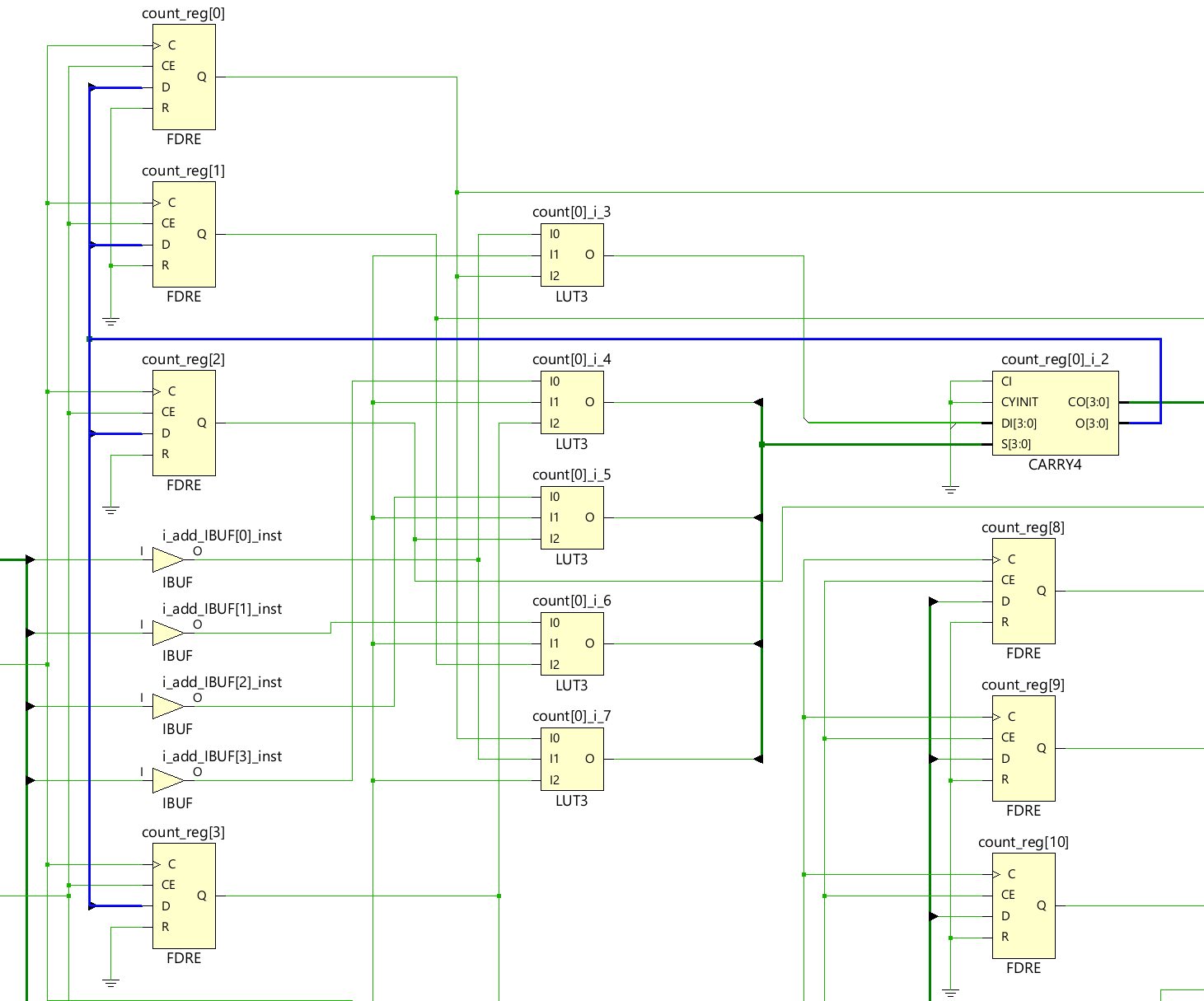


Immagine che contiene diagramma, testo, linea

Descrizione generata automaticamente

## CounterK

Il componente CounterK come CounterAdd è stato pensato come una macchina di Moore che evolve rispetto ai segnali che ha in ingresso forniti dal Controller.

I Segnali in ingresso che sono gestiti dal Controller sono LoadK e DecreaseK. Le operazioni vengono fatte sul fronte di salita del Clock.

Status\_Init è lo stato di inizializzazione, nel caso in cui si verifica un errore si torna in questo stato. Appena arriva il segnale di LoadK si va nello stato Status\_Read\_K.

Status\_Read\_Add assegna il valore in ingresso i\_k ad un segnale interno che verrà utilizzato per eseguire i decrementi fino ad arrivare a 0.

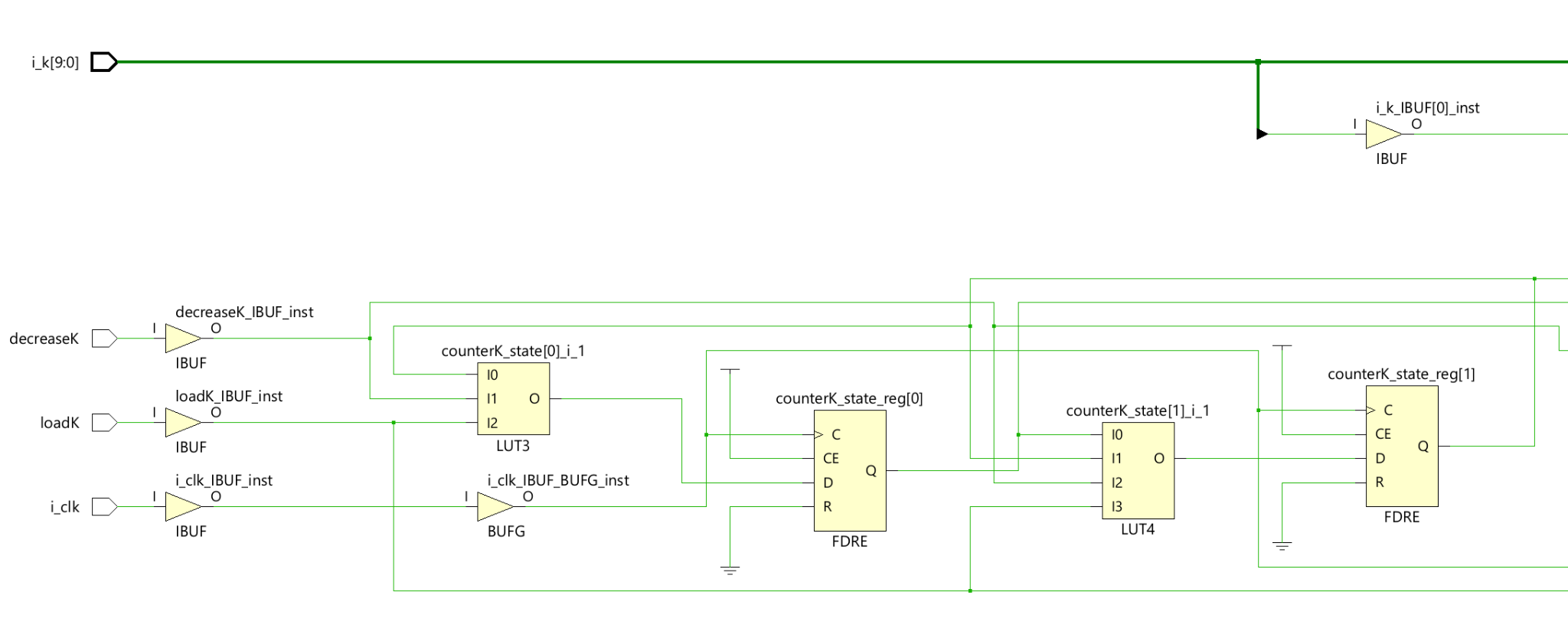
Status\_Idle è uno stato cuscinetto in cui entrambi i segnali di Load e di Decrease sono a 0 .

Si entra nello stato Status\_Decrease\_K quando ho il segnale di DecreaseK ad 1 e grazie alla libreria std\_logic\_unsigned eseguo il comando di decremento ( -1) di K.

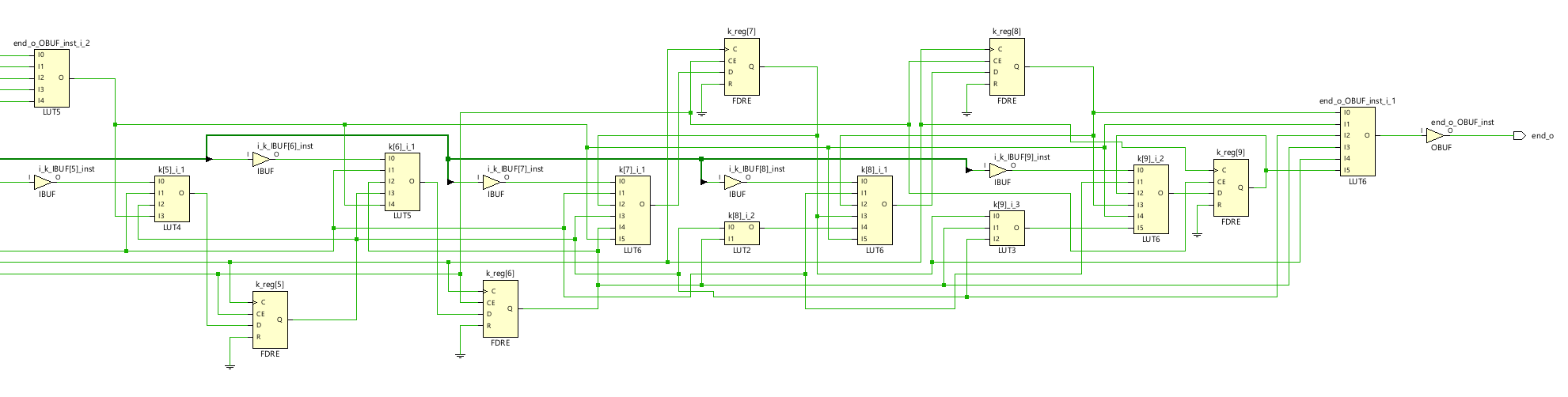
In modo asincrono controllo che la variabile interna sia diversa da 0, appena diventa 0, in qualunque stato in cui mi trovo, mi sposto in Status\_Rise\_End ed alzo il segnale in uscita end\_o. Rimango in questo stato fino a quando non mi viene chiesto di caricare nuovamente un dato nella memoria interna.

Il controllo asincrono di K non è rilevante negli stati Status\_Init e Status\_Idle visto che non eseguono operazioni che modificano il dato e quindi non vengono riportate nella macchina a stati. Ad inizio esecuzione per evitare conflitti la variabile interna viene inizializzata con tutti i bit a 1 così che non entro accidentalmente in Status\_Rise\_End dall’inizio dell’esecuzione.

Nello Screenshot vengono raffigurati i segnali sintetizzati dal programma e i flip-flop D che memorizzano i valori dello stato corrente della macchina a stati



In questo screenshot si vedono i flip-flop D che memorizzano i bit del segnale K che verrà archiviato per poter eseguire i decrementi successivi richiesti tramite il segnale di decreaseK se sono nel corretto stato corrente. Inoltre si può vedere la look-up table che alzerà il segnale di uscita end\_o quando tutti i bit di K avranno come valore 0



## Controller

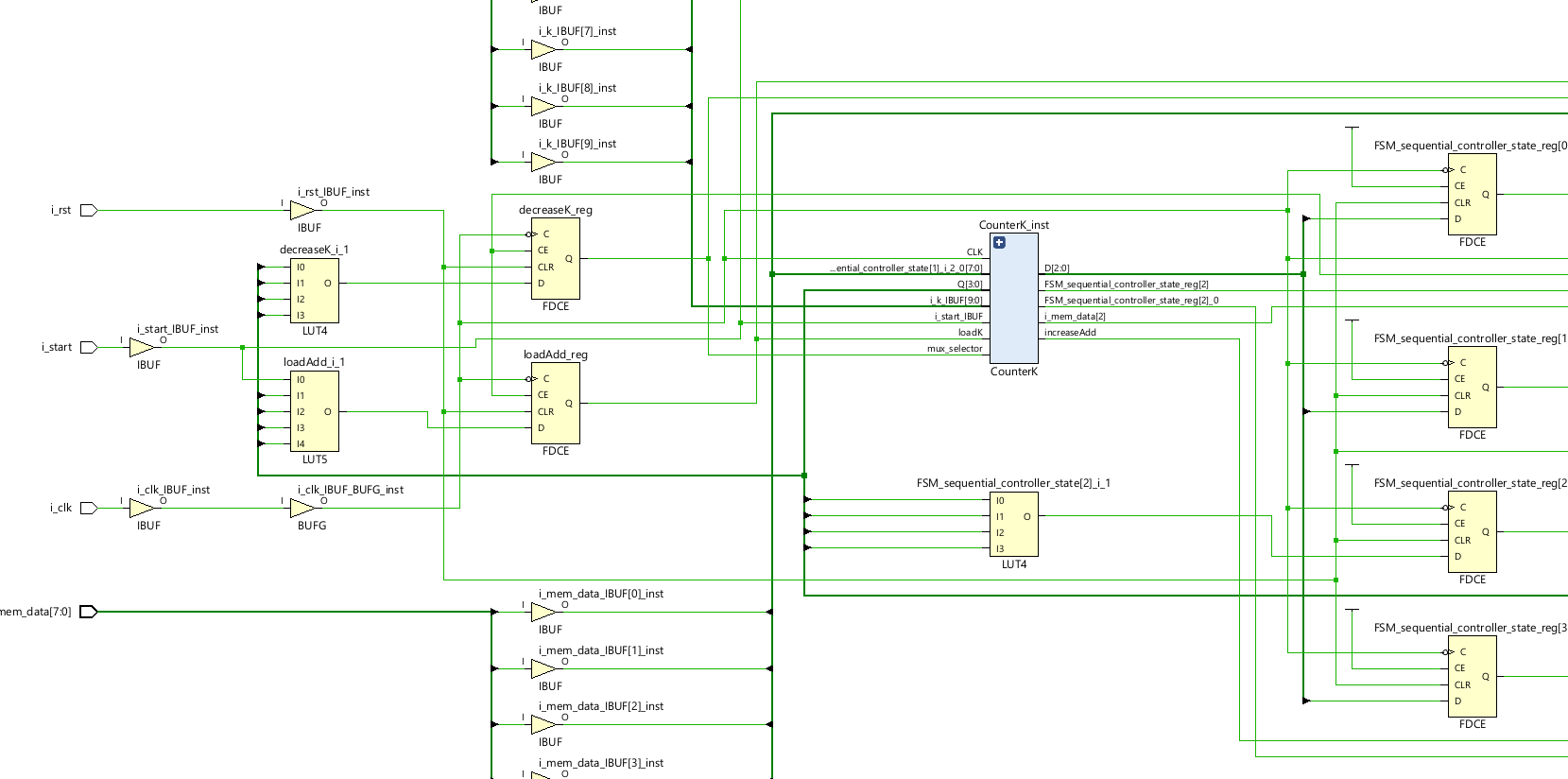
A livello di codice lo abbiamo realizzato con un singolo process che realizza la FSM precedentemente descritta tramite un costrutto case is.

Per semplicità di lettura gli stati sono stati definiti con un type S definito ad hoc.

Nello screenshot sottostante si possono vedere alcuni flip-flop D di segnali interni alla macchina per il controllo dei componenti.

Il sintetizzatore unisce il segnale di LoadAdd e quello di LoadK essendo alzati entrambi solo nello stato Load.

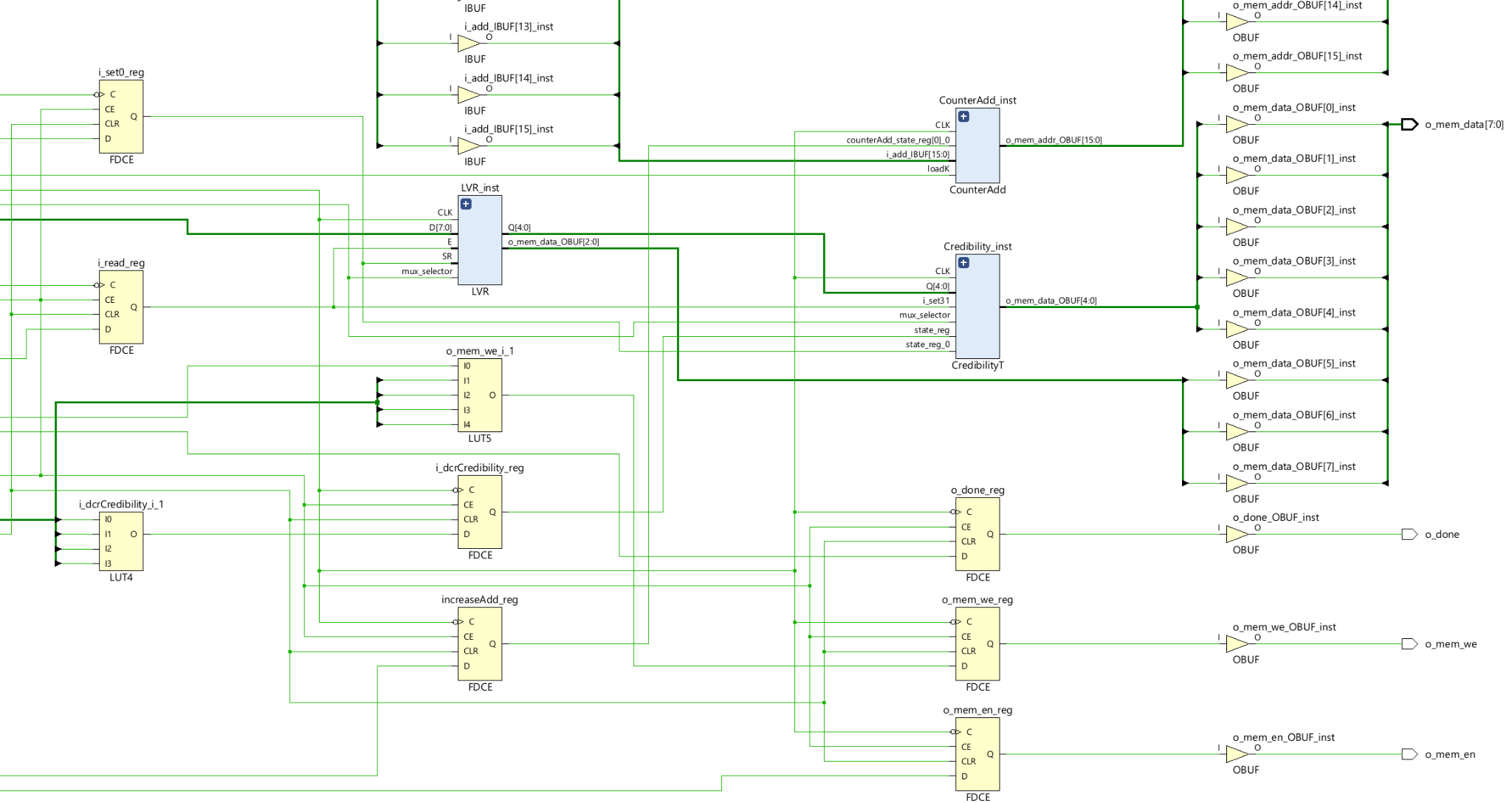
Inoltre in questo screenshot si vedono anche i flip-flop che memorizzano lo stato corrente del controllore.



In questo screenshot si possono vedere alcuni componenti interni visti in modalità black box e la loro interazione con i segnali generati dal controllore.

Non sono stati espansi perché sono già stati discussi precedentemente.

Un cosa da notare è che il MUX che seleziona cosa va scritto nella cella di memoria della RAM non viene sintetizzato ma viene realizzato con un segnale in ingresso ai componenti che sono di interesse per il MUX, cioè LVR e il Credibility. In più visto che i tre bit più significativi non sono influenzati dal credibility questi vengono gestiti da delle look-up table all’interno del LVR.



## 

# Risultati sui Test Bench

## Test Bench D’esempio

Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

Time: 1980100 ps Iteration:2 Process: /project\_tb/test\_routine File: C:/Users/marco/VivadoProjects/project\_reti\_logiche\_final/project\_reti\_logiche\_final.srcs/sim\_1/new/project\_tb\_2324.vhd

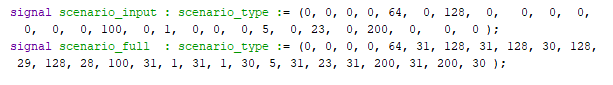
$finish called at time : 1980100 ps : File "C:/Users/marco/VivadoProjects/project\_reti\_logiche\_final/project\_reti\_logiche\_final.srcs/sim\_1/new/project\_tb\_2324.vhd" Line 187

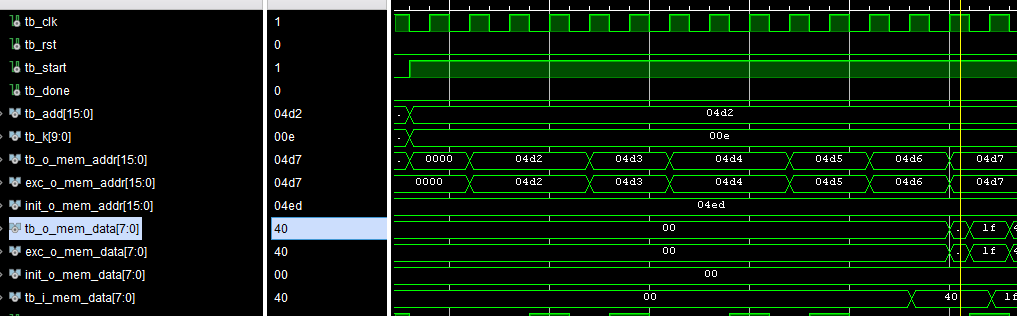
launch\_simulation: Time (s): cpu = 00:00:18 ; elapsed = 00:00:33 . Memory (MB): peak = 2576.988 ; gain = 1161.879

Questo Test Bench è stato fornito dal professore e ci è servito per vedere se per le prime esecuzioni il comportamento d’insieme della macchina e dei suoi sotto componenti.

## Caso W = 0 nelle prime 2 posizioni

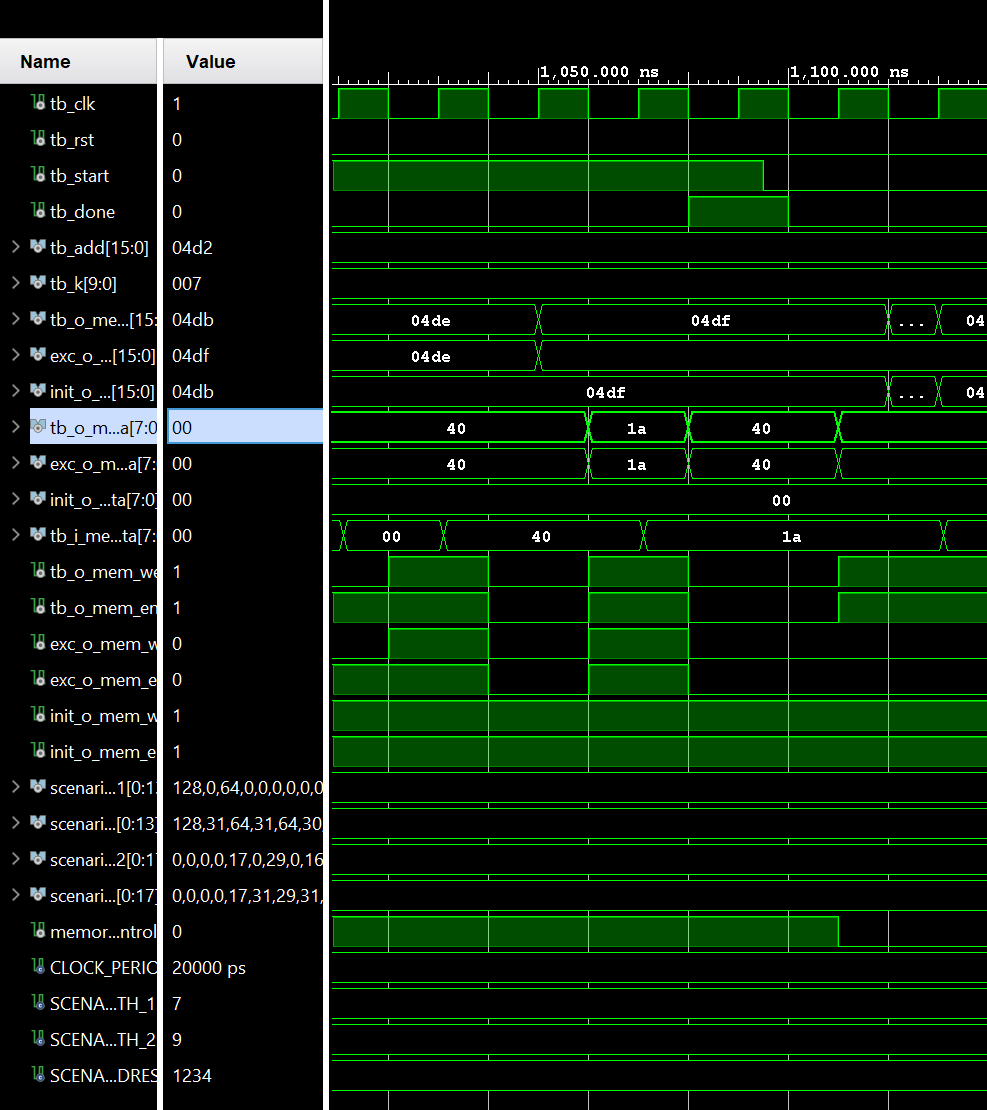
In questo Test Bench abbiamo simulato il comportamento della macchina quando i primi due WordByte sono pari a 0. Da specifica il comportamento è che se arrivano degli 0 come primi valori di W anche il credibility avrà come valore 0, fino a quando non leggo il primo W con valore diverso da zero.





Come si vede dallo screenshot nei primi quattro indirizzi della RAM vengono letti e scritti dei valori pari a zero, il quinto indirizzo contiene 64 (40 hex) e dopo essere stato letto viene modificato il segnale o\_mem\_data che cambia in 40 iniziando l’esecuzione normale.

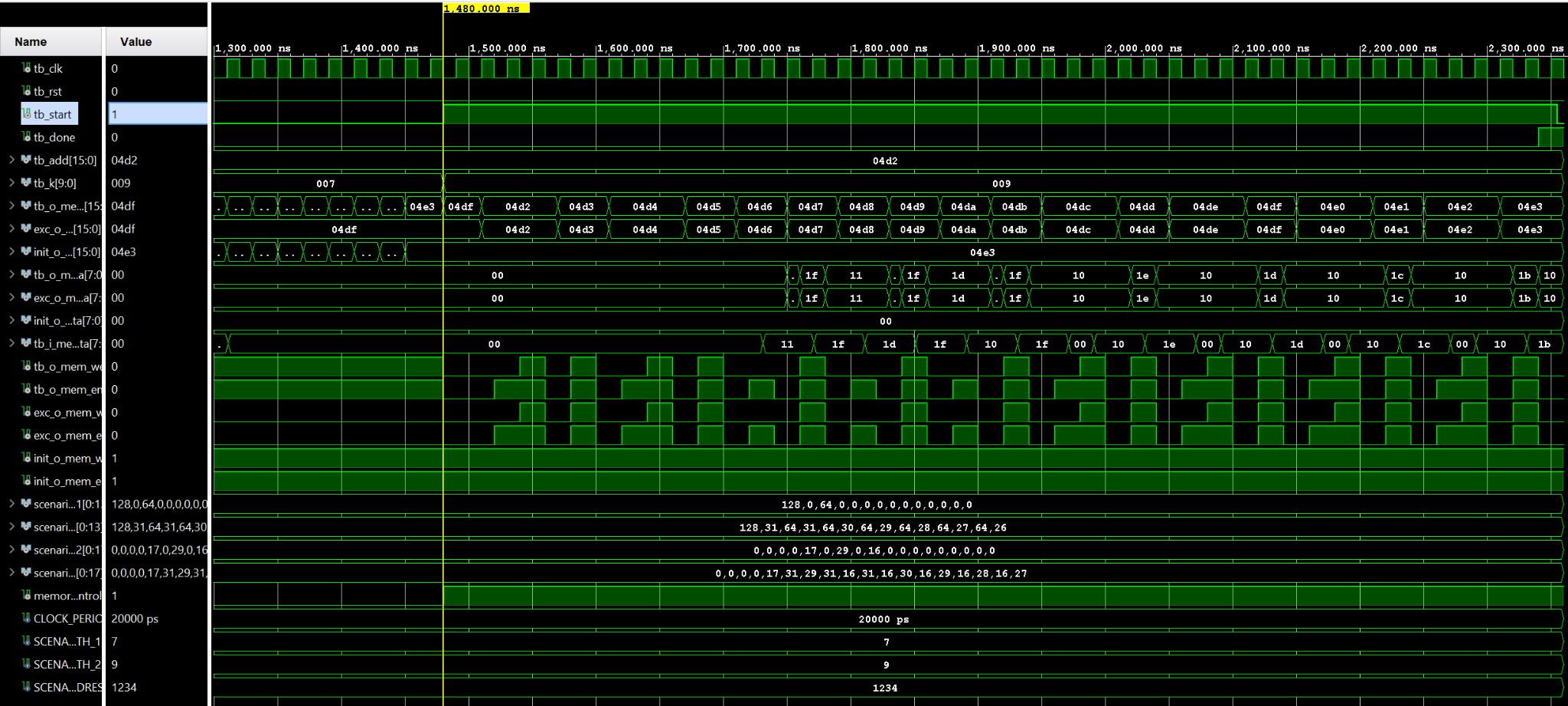
## Esecuzioni Multiple



In questo Test Bench viene simulata una esecuzione successiva di due elaborazioni che hanno K diversi tra di loro e WordByte diversi.

Invece abbiamo mantenuto lo stesso indirizzo della memoria RAM.

Nello screenshot si vede il termine della prima elaborazione. Tra la prima e la seconda esecuzione c’è un tempo di inattività, questo è dovuto al fatto che dobbiamo modificare i valori della RAM per poi iniziare con la seconda elaborazione.

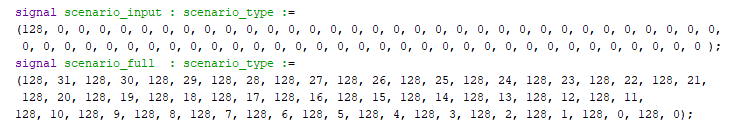


In questo secondo screenshot al tempo 1480 ns si vede il segnale di start ad alto e da quell’istante temporale inizia l’esecuzione della seconda elaborazione.

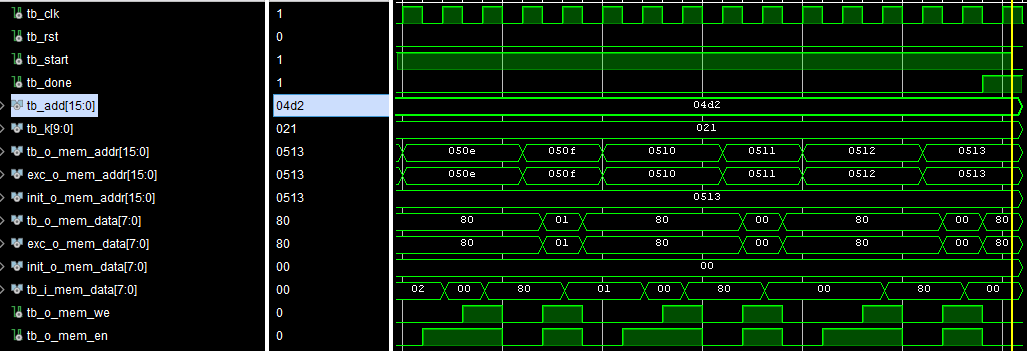
Entrambe le esecuzioni sono state eseguite con successo.

## Credibilità raggiunge lo 0

In questo Test Bench abbiamo testato il caso nel quale il valore della credibilità raggiunge lo zero e ci sono letture non valide successive. Per creare questo scenario abbiamo utilizzato un K pari a 34 e la memoria inizializzata come segue:



Il risultato del test è concorde con ciò che ci aspettavamo:



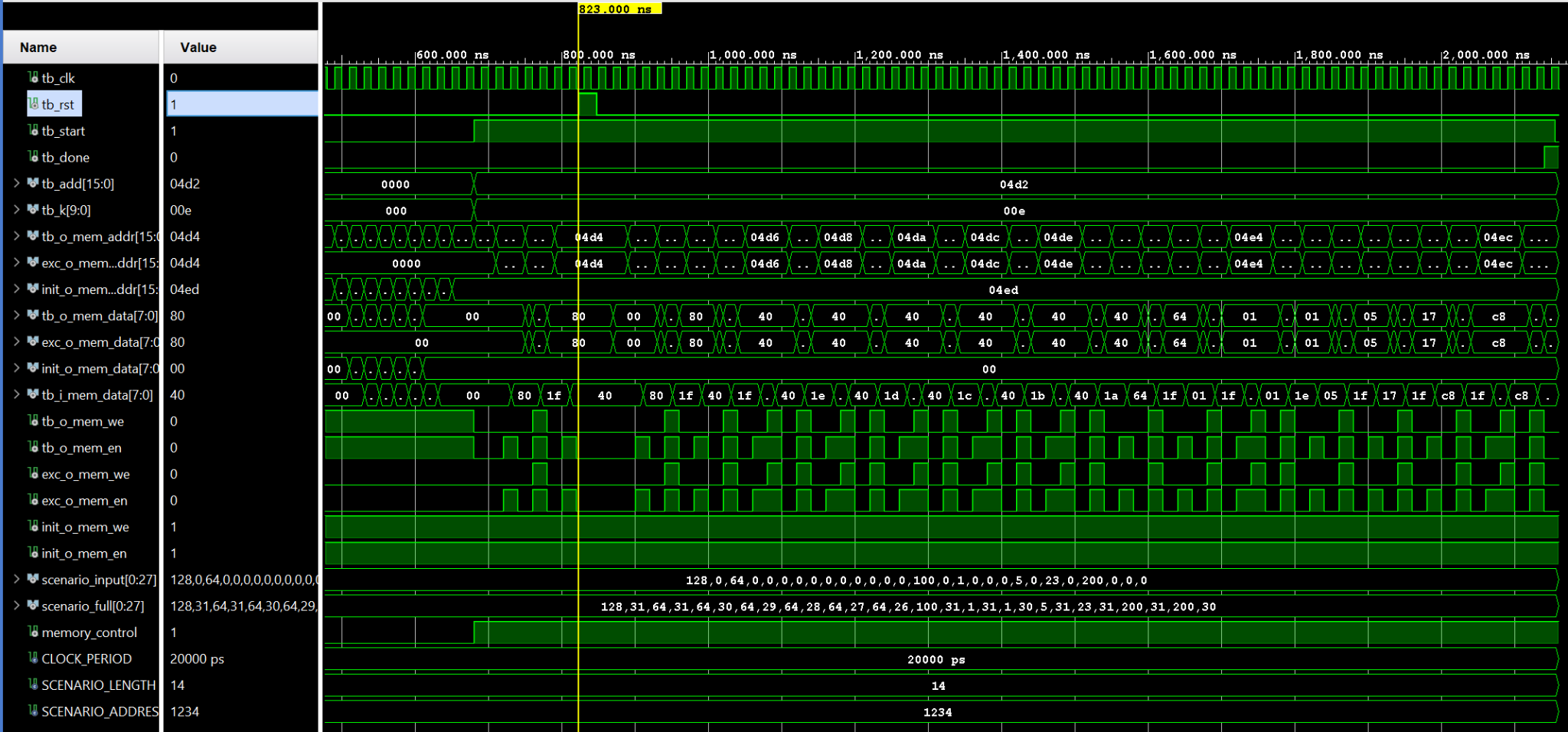
Le righe notevoli in questo test bench sono exc\_o\_mem\_data che contiene il dato passato alla RAM e

tb\_i\_mem\_data che contiene i dati letti dalla RAM.

In accordo con quanto ci aspettavamo tb\_i\_mem\_data assume solo due valori: 0 e 80 (128 in hex) che sono rispettivamente gli zeri del Testbench e il 128 che riempie le celle.

Il segnale exc\_o\_mem\_data invece invia alla RAM i valori 128 e la credibilità che decrementa fino ad arrivare a 0 e rimanere tale anche con letture non valide successive.

## Reset Asincrono



In questo Test Bench simuliamo il comportamento della macchina quando arriva un segnale di Reset asincrono rispetto al clock.

Quando arriva il segnale di reset la macchina torna nello stato di Init in cui reinizializza i diversi segnali in ingresso ai diversi componenti e i segnali in uscita alla macchina stessa.

Quando il segnale di Reset si abbassa, al ciclo di clock successivo inizia nuovamente, da capo la sua elaborazione fino al suo completamento.

## 

# Conclusioni

Il componente ha superato con successo tutti i test bench, sia in behavioral, sia in post synthesis, Functional e Timing.

Tramite la timing simulation abbiamo visto che i ritardi di commutazione raggiungono in media i 3,7 ns e lo slack time è > 6 ns, questo permetterebbe di raddoppiare la frequenza di clock in sicurezza senza incorrere a problemi di sincronizzazione (semiperiodo di 5 ns).

Ulteriori velocizzazioni sono rischiose perché il semiperiodo di clock si avvicina ai ritardi di commutazione.

Nel complessivo il tempo di elaborazione comprende 2 cicli di clock per l’inizializzazione dei componenti e 4 cicli per ogni parola valida , 5 se non è valida.

Il caso pessimo ha una complessità temporale di 2 + 5\*K cicli di clock.

Abbiamo ritenuto che operare su indirizzo e memoria nello stesso ciclo presenterebbe molti problemi di sincronizzazione (come ad esempio usare la memoria su un indirizzo che non è ancora del tutto stabile) quindi abbiamo evitato di fare cose del genere .

Seguendo questa scelta, non ci sono stati “inutili” perché ogni stato di elaborazione o fa uso della memoria o ne modifica l’indirizzo di lavoro.

Nella progettazione del componente abbiamo lavorato gestendo anche casi limite che sono poi stati dichiarati “fuori specifica" dopo aver terminato con il nostro lavoro.

Questi casi extra gestiti sono il caso in cui K = 0 e il caso in cui il componente debba leggere/scrivere oltre il limite della memoria.

Abbiamo lasciato le politiche di gestione di questi casi all’interno del componente anche se non richieste, questo perché aumentano la robustezza complessiva a casi limite e rimuovono side effect, tutto senza diminuire le prestazioni, infatti non richiedono segnali extra né stati extra né cicli di clock dedicati solo alla gestione di queste eccezioni (richiedono solo transizioni extra su stati già esistenti).

Il caso K=0 causa la terminazione dell’elaborazione il ciclo subito dopo aver caricato K tramite il segnale END che già serve per la terminazione al di fuori del caso eccezionale.

Il caso di indirizzi oltre il massimo della dimensione della memoria viene gestito con l’ overflow del contatore Counter ADD che ritorna a 0, di conseguenza l’ elaborazione continua ripartendo dall’ indirizzo 0x0000 della RAM come se essa fosse “ciclica”.

L’elaborazione termina comunque dopo K parole.

Dal momento che il segnale K è a 10 bit e ADD a 16 bit, non può presentarsi il caso in cui la ciclicità comporta il dover ripassare sui primi byte elaborati.