Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – Anno 2023/2024

Juan Marco Patagoc (Codice Persona 10767244 – Matricola 982655)

Indice

1 Introduzione

1.1 Scopo del progetto

l'intento del progetto è quello di implementare un modulo hardware descritto in VHDL in grado di interagire con la memoria e di elaborare correttamente i dati in ingresso, seguendo la specifica descritta di seguito.

1.2 Specifica generale

Il sistema riceve un messaggio costituito da una sequenza di K parole, il cui valore è compreso tra 0 e 255.
In realtà il valore 0 dentro la sequenza deve essere considerato non come un valore di per sé ma come informazione: "il valore non è specificato".
La sequenza di K parole è memorizzata a partire da un indirizzo specificato ADD andando per ogni 2 byte.
La sequenza di parole in ingresso deve essere modificata secondo la regola che segue.

1.3 Regola

Il modulo ha il compito di completare la sequenza sostituendo gli 0 laddove sia presente e di riempire i byte mancanti inserendo il valore di credibilità caratterizzante di ogni parola della sequenza, in modo dettagliato:

- gli 0 devono essere sostituiti con l'ultimo valore non nullo letto precedentemente e appartenente alla sequenza.
- il valore della credibilità è sempre resettata al valore 31 ogni volta che il valore della sequenza è non nullo, mentre viene decrementato ogni volta che viene letto uno zero; la credibilità non deve scendere sotto il valore 0.

Caso particolare
Quando la sequenza inizia con 0, la parola in uscita e la sua corrispettiva credibilità sono anch'essi degli zero. Tale valore permane fintanto che non si legge nella sequenza un valore non nullo.

1.4 Interfaccia del componente

Il componente è caratterizzato dalla 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;
```

Figura 1: Interfaccia del componente

I segnali rappresentano:

- 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 START;
- i_start è il segnale START generato dal Test Bench;
- i_k è il segnale (vettore) K generato dal Test Bench rappresentante la lunghezza della sequenza;
- i_add è il segnale (vettore) ADD generato dal Test Bench che rappresenta l'indirizzo dal quale parte la sequenza da elaborare;
- 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_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- 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.

1.5 Funzionamento

Il modulo ha 3 ingressi primari: START da 1 bit, ADD da 16 bit e K da 10 bit.
Ha un'uscita primaria DONE da 1 bit.
Tutti i segnali sono sincroni e interpretati sul fronte di salita; l'unica eccezione è RESET che è asincrono.

All'istante iniziale, quando il RESET è 1, DONE deve essere 0.
Quando RESET torna a 0 il modulo inizierà l'elaborazione quando START è uguale a 1 e rimarrà tale fino a quando DONE sarà uguale a 1.

Al termine della computazione, DONE dovrà essere portato a 1 (notifica la fine dell'elaborazione).
DONE rimane a 1 fino a quando START non sarà stato riportato a 0.

Un nuovo segnale di START è dato solo quando DONE sarà stato riportato a 0.

Nota importante
Si considera che prima della prima codifica verrà SEMPRE dato il reset del modulo.
Alla successiva elaborazione, non si attende il reset del modulo.
Il modulo è progettato per elaborare più sequenze

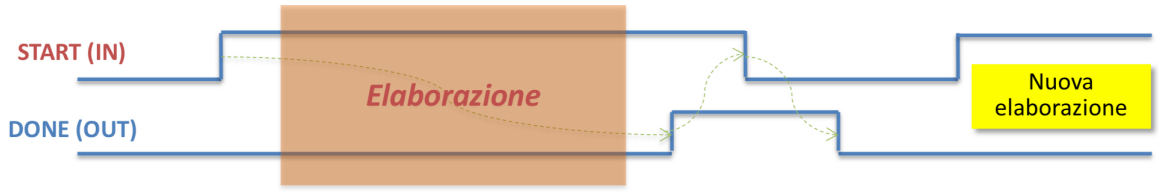

Figura 2: Segnali *START* e *DONE* durante elaborazioni multiple

2 Architettura

Il modulo è costituito da segnali interni ed è implementato basandosi su un automa a stati finiti.

2.1 Segnali interni

I segnali interni rappresentano:

- *state* e *next_state*: memorizzano lo stato della FSM;
- *o_done_next*: memorizza il valore prossimo di o_done;
- *o_mem_en_next*: memorizza il valore prossimo di o_mem_en;
- *o_mem_we_next*: memorizza il valore prossimo di o_mem_we;
- *o_mem_addr_next*: memorizza il valore prossimo di o_mem_addr;
- *o_mem_data_next*: memorizza il valore prossimo di o_mem_data;
- *temp* e *temp_next*: memorizzano i valori della sequenza man mano elaborati da mettere in uscita;
- *address* e *address_next*: memorizzano l'indirizzo da cui recuperare il dato richiesto e successivamente scriverne il dato ottenuto dall'elaborazione;
- *input* e *input_next*: memorizzano il valore in ingresso dato dalla RAM, specificatamente i_mem_data;
- *cred* e *cred_next*: memorizza la credibilità del dato in analisi;
- *k* e *k_next*: segnale in cui inizialmente si mette il numero delle parole della sequenza K. Ogni volta che viene elaborata una parola e la sua associata credibilità si decrementa il valore del segnale
- *k_fixed* e *k_fixed_next*: memorizza il numero delle parole della sequenza K dell'elaborazione in corso.

2.2 Macchina a Stati Finiti

L'automa a stati finiti è costituito da 13 stati:

- *START*: stato iniziale della macchina, il modulo riceve i_add e i_k e salva i valori negli appositi segnali
- *FETCH_DATA_RAM*: stato in cui il modulo chiede alla memoria RAM il dato. Viene chiesta l'attivazione della memoria RAM (o_mem_en_next = 1) in modalità lettura (o_mem_we_next = 0) e viene mandato l'indirizzo su cui ricavare il dato necessario
- *WAIT_RAM*: stato dove si attende la risposta della memoria RAM che è in modalità lettura

- *GET_WORD*: stato in cui la memoria RAM ha in i_mem_data il dato su cui si andrà ad elaborare, che poi è inviato al modulo. Allo stesso tempo la RAM viene disattivata
- *SETUP*: stato dove il dato che la RAM ha inviato al modulo viene salvato in input_next, pronto per essere elaborato nel prossimo stato
- *ELAB_WORD*: elaborazione del dato e della sua relativa credibilità, secondo la logica descritto nelle regole *(vedi paragrafo 1.3 e paragrafo 2.3.2)*
- *WRITE_WORD*: stato dove viene chiesta l'attivazione della memoria RAM (o_mem_en_next = 1) in modalità scrittura (o_mem_we_next = 1) e viene mandato il dato appena elaborato con l'indirizzo su cui scrivere; nel frattempo il segnale contenente l'indirizzo viene incrementato di uno, in quanto viene successivamente scritto la credibilità
- *WRITE_CRED*: stato in cui la RAM rimane in modalità scrittura e viene scritta la credibilità del dato
- *DONE_WRITING*: stato dove la memoria RAM viene disattivata
- *NEXT_STEP*: una volta elaborato e scritto il dato e la sua credibilità, viene decrementato k
- *VERIFY*: se non ci sono più parole rimaste da elaborare (k = 0) viene chiesta di sollevare il segnale o_done (o_done_next = '1') e passa nel suo stato di fine, altrimenti si continua con l'elaborazione
- *DONE*: stato in cui si attende che il Test Bench abbassi i_start così da poter abbassare o_done e nel caso poter fare un'altra elaborazione
- *DONE_WAIT*: se i_start è uguale a 0 tutti i segnali interni vengono resettati per poter quindi fare un'altra elaborazione

2.3 Scelte progettuali

2.3.1 Processi Utilizzati

Il modulo è costituito da due processi:

1. Processo di tipo sequenziale dedicato alla gestione dei segnali, quindi al reset e all'aggiornamento dei valori dei segnali al fronte di salita.
2. Processo che implementa il funzionamento della FSM: a seconda dei segnali interni del modulo, i segnali di ingresso e lo stato corrente della macchina, determina lo stato successivo del modulo.

2.3.2 Algoritmo di Elaborazione

Il modulo utilizza un algoritmo di elaborazione del dato in ingresso spiegato qui sotto.

1st Step: Elaborazione del dato in ingresso

- Se il dato in ingresso è diverso da 0 il segnale *temp* prende il valore in ingresso.
- Se il dato in ingresso è 0 e corrisponde alla prima parola della sequenza (*k = k_fixed*) *temp* ha valore 0.
- Se il dato in ingresso è 0 e non è la prima parola della sequenza *temp* rimane all'ultimo valore non nullo salvato.

2st Step: Elaborazione della corrispettiva credibilità

- Se il dato in ingresso è diverso da 0 il segnale *cred* si resetta al valore 31.
- Se il dato in ingresso è 0 e corrisponde alla prima parola della sequenza il segnale *cred* ha anch'esso valore 0.
- Se il dato in ingresso è 0 e non è la prima parola della sequenza, se *cred* ha precedentemente valore nullo rimane a 0 altrimenti viene decrementato di 1.

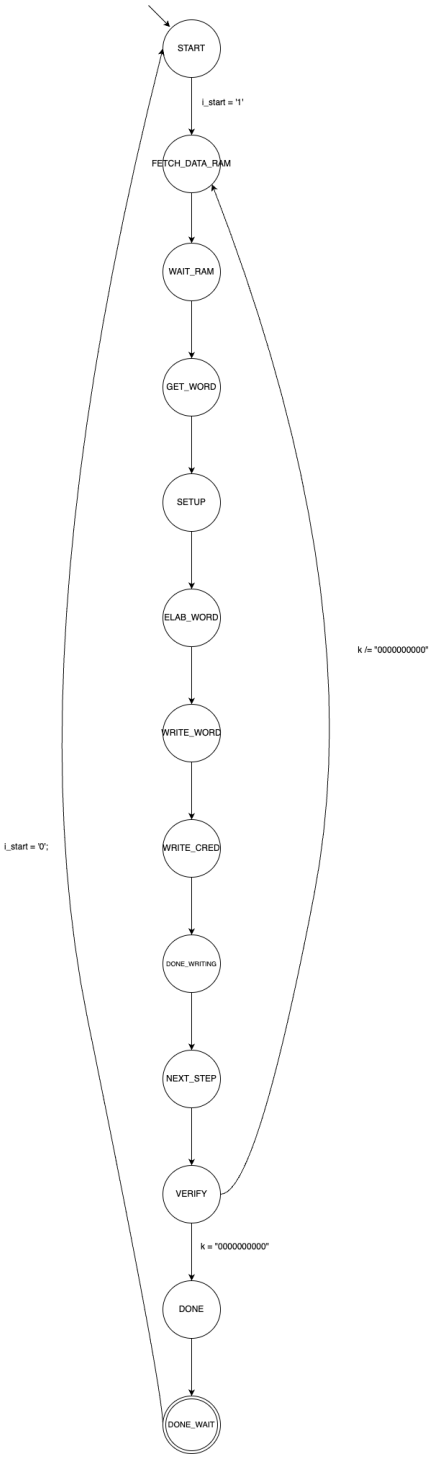

Figura 3: Macchina a Stati

3 Risultati Sperimentali

3.1 Report di sintesi

Sono riportate i risultati della sintesi

Resource	Utilization	Available	Utilization %
LUT	92	8000	1.15
FF	100	16000	0.63
IO	64	150	42.67

Slice Logic
- Slice LUTs (1%)
  - LUT as Logic (1%)
- Slice Registers (1%)
  - Register as Flip Flop (1%)

Figura 4 e 5: Risorse utilizzate

Il modulo sintetizzato non presenta alcun Latch.

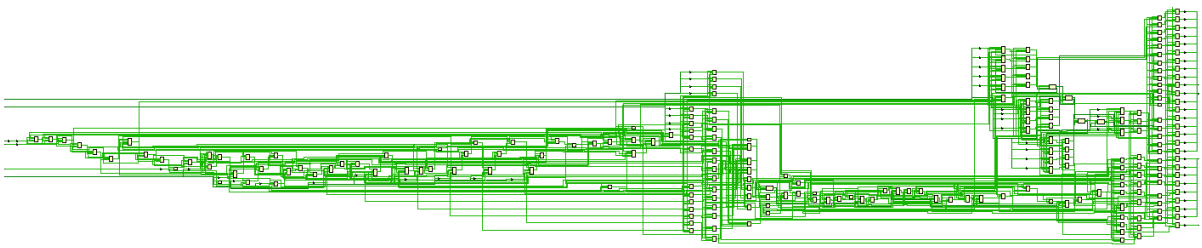

Figura 6: Schematico del modulo sintetizzato

3.2 Risultati dei Test

Per verificare il corretto funzionamento del modulo sono stati condotti dei TestBench che oltre a quello dato dal docente, controllano il comportamento del modulo anche nei possibili casi limite.
Il modulo viene simulato in due modi: Behavioral Simulation prima della sintesi e Functional Simulation dopo la sintesi.
In seguito sono presentati i test simulati nei due modi.

3.2.1 One scenario

TestBench offerto dal docente, verifica un semplice scenario.

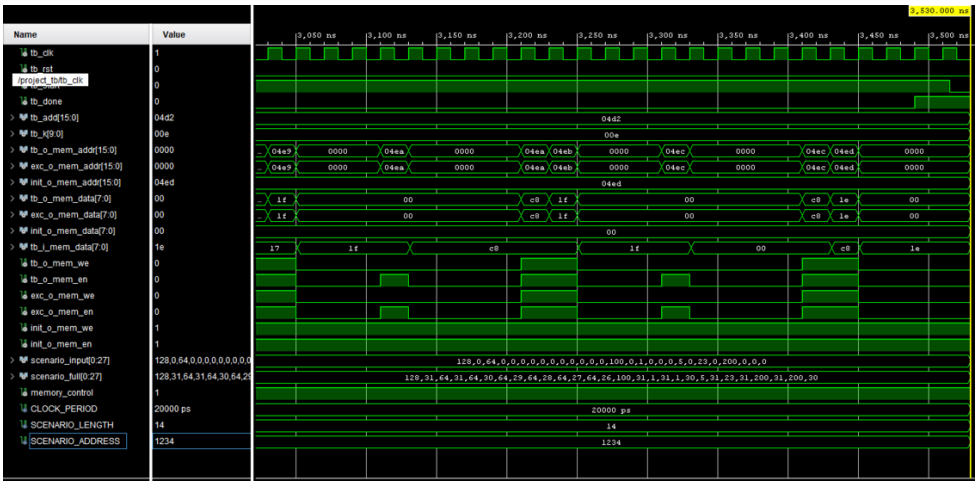

Figura 7: Simulazione “One Scenario”

3.2.2 Credibility to 0

TestBench che verifica che la credibilità non scenda sotto il valore 0.

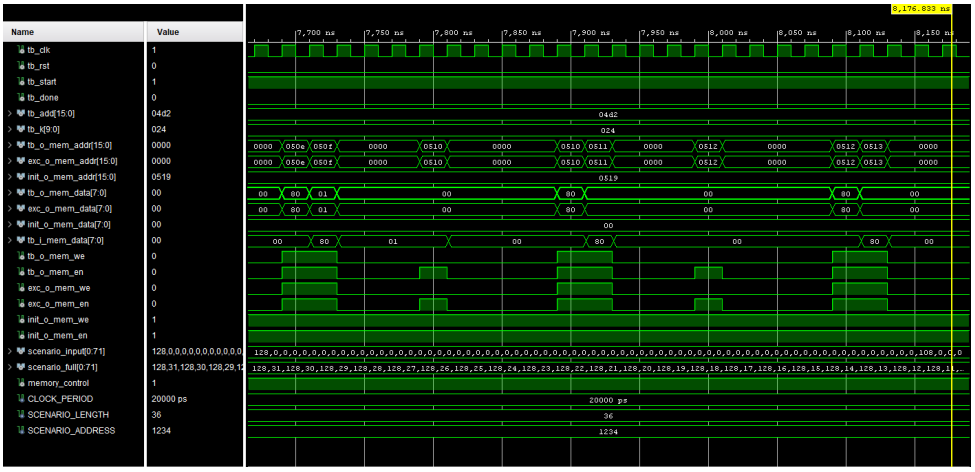

Figura 7: Simulazione “Credibility to 0”
La credibilità rimane a 0 fino a quando non incontro un valore non nullo

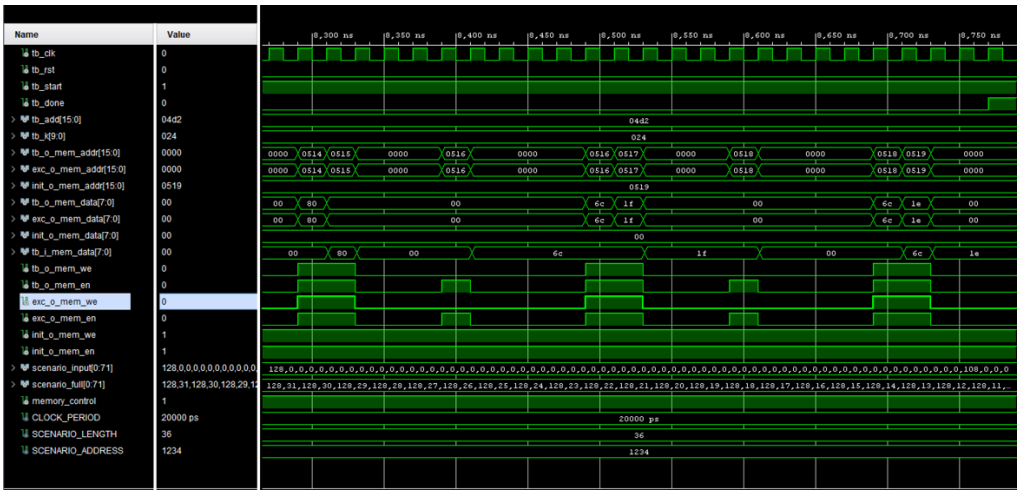

Figura 8: Simulazione “Credibility to 0”

La credibilità viene resettata al valore 31 appena si raggiunge un valore della sequenza non nullo

3.2.3 Multiple Starts (no reset)

TestBench che verifica il corretto funzionamento del modulo quando elabora più di una sequenza.

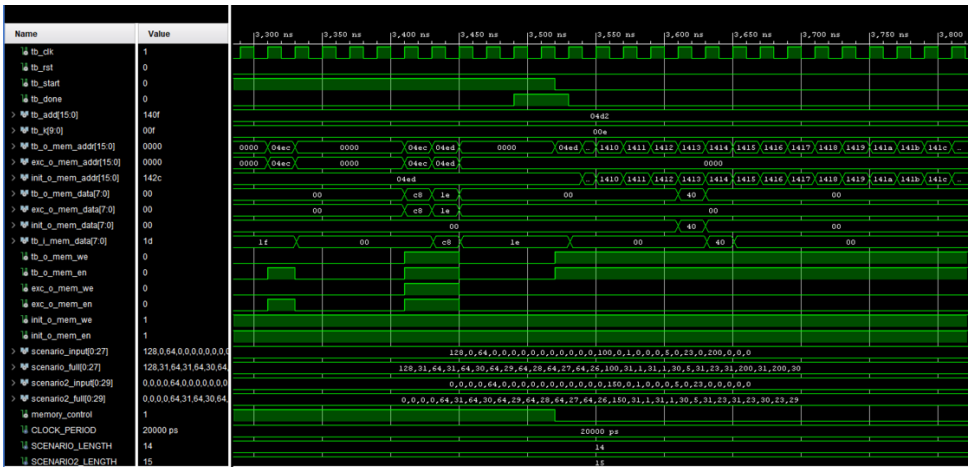

Figura 8: Simulazione “Multiple Starts”, fine prima elaborazione.

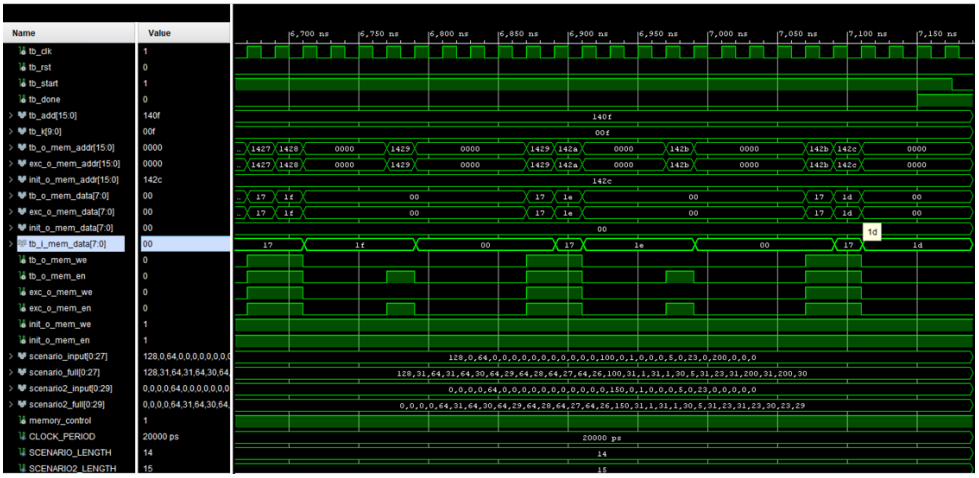

Figura 9: Simulazione “Multiple Starts”, fine seconda elaborazione.

3.2.4 Multiple Starts (mid reset)

TestBench che verifica il corretto funzionamento del modulo quando ,durante un'elaborazione, viene sollevato il segnale di reset.

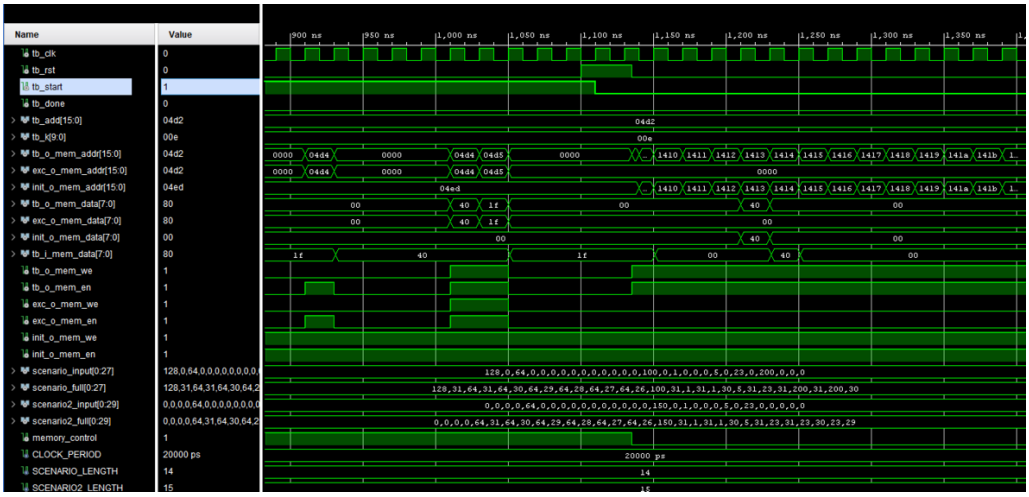

Figura 10: Simulazione "Multiple Starts (mid reset)"
Reset sollevato nel mezzo di un'elaborazione

3.2.5 Start from 0

TestBench che verifica che quando una sequenza inizia col valore 0, il valore iniziale in uscita e la sua credibilità hanno valore 0, come descritto nelle regole (*caso particolare nel paragrafo 1.3*).

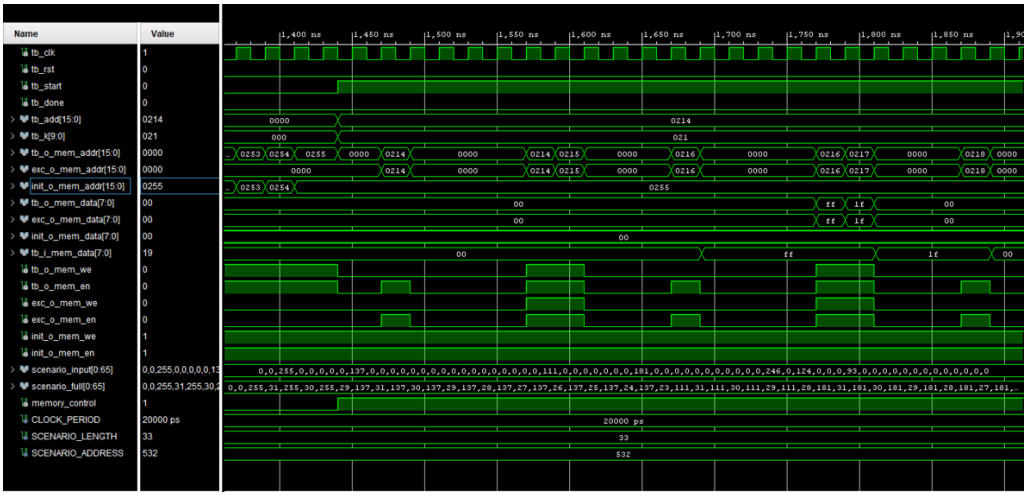

Figura 11: Simulazione "Start from 0"
La sequenza inizia con il valore 0

4 Conclusioni

Il modulo sintetizzato funziona correttamente e i test elencati sopra sono stati simulati nelle due modalità senza problemi (*Behavioral e Functional Simulation*).

Il modulo progettato è in grado di:

- Comunicare con la memoria RAM
- Accedere ad un indirizzo della RAM e ricavarne il dato
- Elaborare il dato e la sua credibilità rispettando le regole richieste
- Scrivere nella memoria RAM i dati in uscita
- Compiere elaborazioni multiple