POLITECNICO
MILANO 1863

RELAZIONE PROGETTUALE DELLA PROVA FINALE DI *RETI LOGICHE*

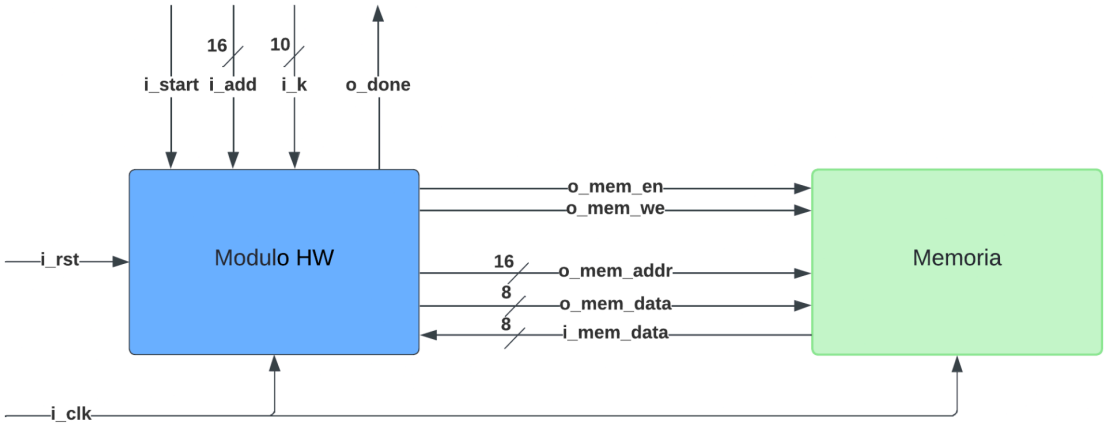
A.A 2023-2024

A cura di:

Tommaso Morganti - 10716563
Elia Falzoni - 10763955

INTRODUZIONE

La specifica progettuale richiede lo sviluppo di un componente hardware in grado di operare su una sequenza di numeri interi contenuta in memoria (la quale può contenere elementi da “0” a “255”), eventualmente modificandone i valori e fornire e memorizzare, subito successivamente ad ogni singolo elemento della successione, un “valore di credibilità”, il tutto in maniera coerente alle istruzioni fornite.

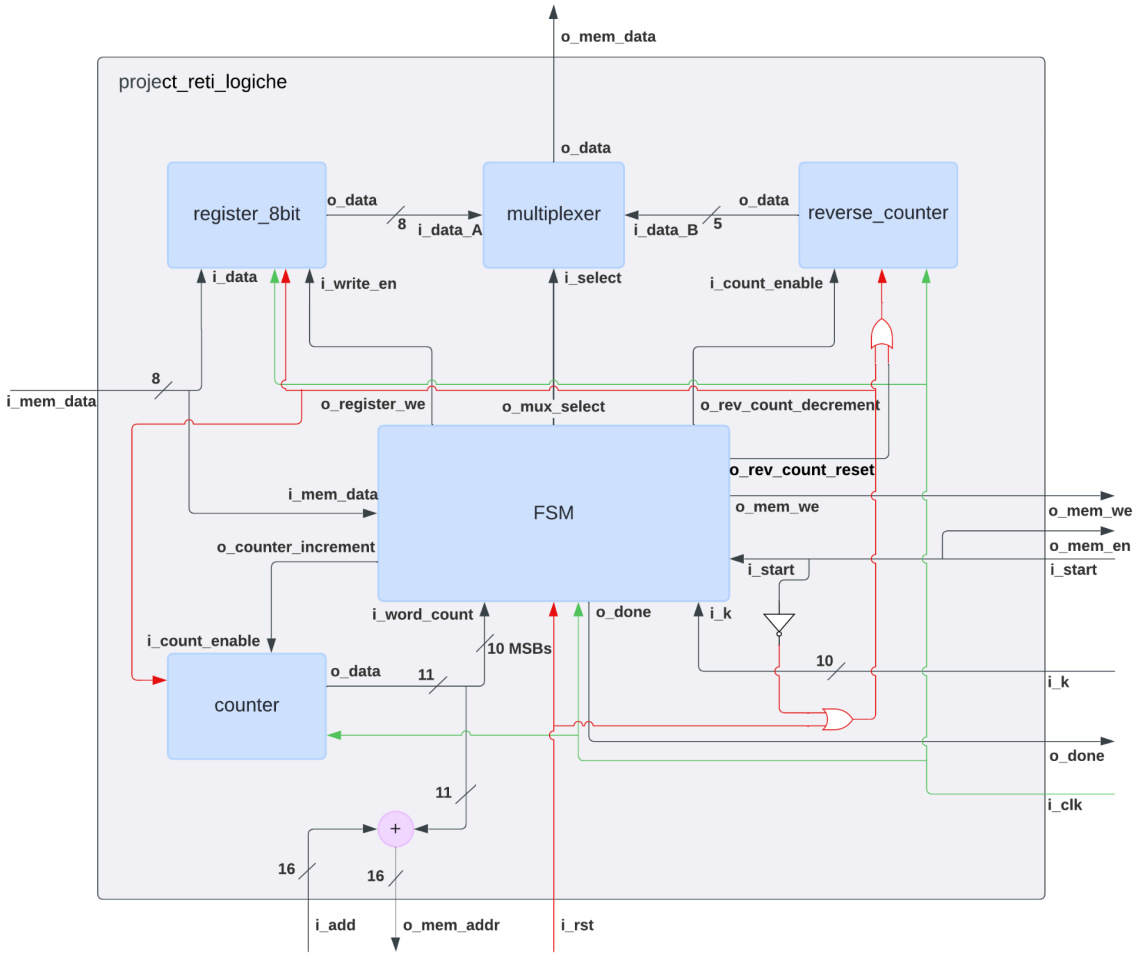


Una sintetica rappresentazione dell’intero apparato. In azzurro, il modulo del quale è richiesta la progettazione.

Noto l’indirizzo di memoria dal quale la sequenza ha inizio e l’offset tra i valori della stessa, è possibile identificare le celle di memoria dedicate agli elementi della successione e quelle destinate ai valori di credibilità.
Letto un valore della sequenza, se pari a “0”, il componente sovrascrive al presente il primo elemento della successione diverso da “0” immediatamente precedente a questo. Altrimenti, nessuna operazione viene effettuata sulla cella di memoria contenente l’intero della sequenza.
Il corretto valore di credibilità, il quale verrà memorizzato in corrispondenza del byte subito successivo, è individuato osservando i valori della successione: se il valore della successione relativo ad esso è diverso da “0”, esso sia pari a”31”, altrimenti, sia esso decrementato di un’unità rispetto al valore di credibilità immediatamente

precedente ad esso in memoria. Si noti che il valore di credibilità non può essere decrementato al di sotto di “0”.
Sono analizzati successivamente, nella presente relazione progettuale, gli opportuni comportamenti derivanti da edge-case individuabili a partire dalla specifica di progetto.

ARCHITETTURA DEL PROGETTO



Rappresentazione dettagliata dell’hardware progettato.

ASPETTI GENERALI

La soluzione progettuale proposta è caratterizzata da cinque componenti: quattro di questi sono di natura sequenziale e uno (il multiplexer) di natura combinatoria. Il segnale di clock è condiviso tra tutti i componenti sequenziali.

Particolarmente meritevoli di attenzione, i meccanismi di *reset* dei componenti sequenziali:

1. Tutti quanti i quattro moduli possono essere resettati dal segnale *i_rst* esterno in maniera asincrona,
2. I componenti sequenziali, ad eccezione della FSM, vengono resettati quando *i_start* è pari a "0". Infatti, quando quest'ultima condizione risulta vera, non è in atto alcuna computazione. Dunque, ciò permette ai sopracitati moduli di poter ricominciare un "ciclo" di lavoro senza dipendere dal segnale *i_rst*.
3. Il componente *reverse_counter* può essere resettato (eventualmente, più volte) dalla FSM in determinate condizioni durante la computazione.

Un altro peculiare aspetto dell'architettura progettata è da notare nel fatto che il segnale *o_mem_en* sia collegato in maniera diretta al segnale *i_start*: questo, in quanto è necessario che la memoria possa essere accessibile per l'intera durata della computazione.

MODULO I: register_8bit

"register_8bit" è un registro contenente, appunto, un valore di 8 bit: in particolare, l'ultimo elemento della sequenza diverso da '0'. Quando è attivo il segnale *i_rst*, i bit memorizzati all'interno del registro assumono tutti valore '0'.
Se in corrispondenza del *rising edge* di *i_clk* il segnale *i_write_en* (controllato dalla *FSM*) è attivo, il registro memorizza l'input da *i_mem_data* (*i_data*, nel modulo). Il segnale *o_data* assume il valore memorizzato nel registro, il quale non presenta alcun segnale di *enable* del segnale di output.

MODULO II: counter

"counter" è un modulo deputato al tenere traccia dell'indirizzo di memoria sul quale operare: ad ogni *rising edge* del *clock*, se il segnale *i_count_enable* (controllato dalla *FSM*) è attivo, il valore memorizzato all'interno di questo componente viene incrementato di un'unità, tenendo dunque conto dell'offset a partire da *i_add* al fine di individuare la corretta cella di memoria da leggere o sulla quale scrivere.
Se il segnale *i_rst* è attivo, i bit memorizzati in questo componente assumono tutti valore '0'.
Si noti che lo *stored value* di questo modulo è costituito da 11 bit e non da 10 come *i_k*: infatti, avendo la necessità, per scelta di design, di dover "contare" un numero di volte pari al doppio del valore di quest'ultimo segnale, risulta chiara questa particolarità.
Il segnale *o_data* assume il valore memorizzato nel modulo, il quale non presenta alcun segnale di *enable* del segnale di output.
Infine, è importante notare che il segnale *o_data* di "counter", sommato al valore del segnale *i_add*, costituisca *o_mem_addr*, appunto, l'indirizzo di memoria sul quale operare nel determinato momento dell'esecuzione.

MODULO III: multiplexer

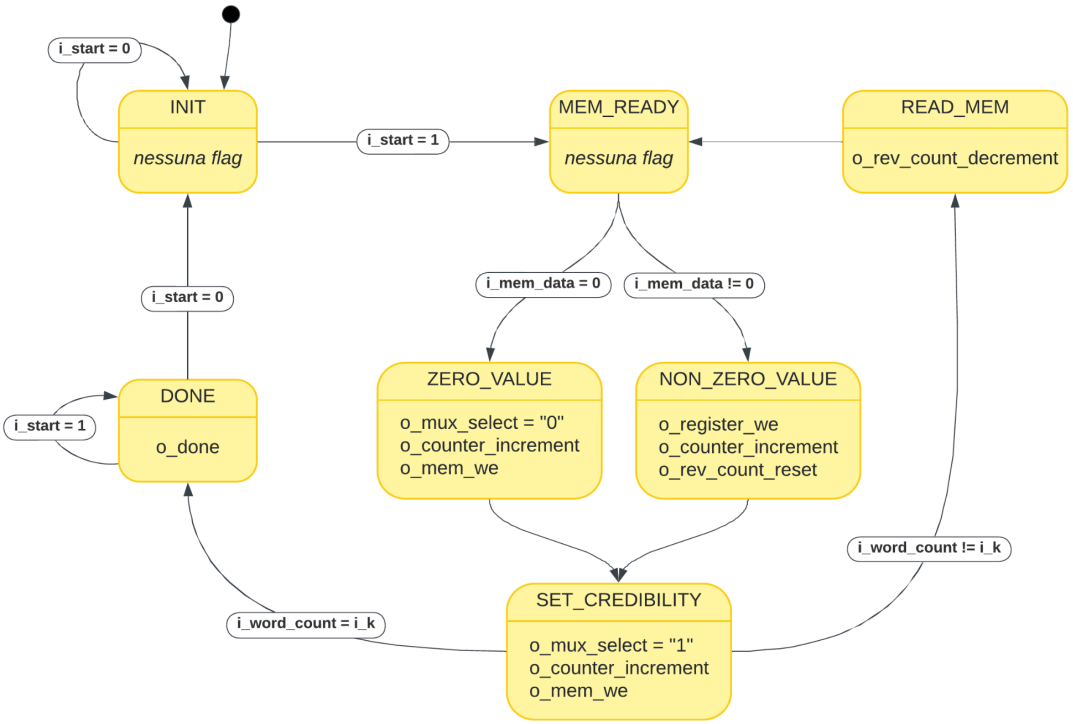
Il multiplexer è il componente dedicato alla selezione del corretto valore da trasferire in memoria quando è richiesta un'operazione di scrittura: grazie al segnale *i_select*, proveniente dalla *FSM*, è infatti possibile effettuare la selezione tra lo *stored value* di *register_8bit* e quello di *reverse_counter* (estendendo quest'ultimo ponendo tre bit uguali a '0' in posizioni più significative).
E' importante notare come, al fine di perseguire le istruzioni della specifica di progetto nell'eventualità di una sequenza i cui primi due valori siano 'zero', il presente modulo, nel caso in cui il segnale *i_data_A* (ovvero, il valore memorizzato all'interno di *register_8bit*) sia costituito da bit tutti quanti posti a '0' (in quanto questa condizione è implicata dal fatto che la sequenza non abbia ancora presentato alcun valore non nullo nel determinato momento di esecuzione),

ponga *o_data* anch'esso pari a zero. Dunque, questo particolare *path* di esecuzione fa sì che il valore di credibilità relativo all'elemento uguale a 'zero' sia anch'esso 'zero'.

MODULO IV: reverse_counter

Il presente modulo ha la funzione di tenere traccia del corretto valore di credibilità da memorizzare per ogni elemento della sequenza. In corrispondenza di ogni *rising edge* del *clock*, se lo *stored value* del presente componente è diverso da 'zero' (al fine di non permettere underflow) e il segnale *i_count_enable* (controllato dalla *FSM*) è attivo, quest'ultimo viene decrementato di un'unità.
"reverse_counter" è soggetto a reset in corrispondenza dell'attivazione del segnale *i_rst* (il quale, nell'ambito di questo specifico modulo, è controllato anche dall'*FSM*): la procedura consiste nel porre il valore memorizzato a '31' (ovvero, porre tutti quanti i bit a '1').

MODULO V: finite-state machine (FSM)



La FSM è il componente di maggior importanza del progetto: è deputata a coordinare tutti quanti i restanti moduli, regolandone l'attività grazie all'ausilio dei segnali uscenti da essa.
In alto, una schematica rappresentazione degli stati del componente e dei comportamenti ad essi correlati dal punto di vista dei segnali in uscita (si noti che, per ogni stato, sono specificate esclusivamente le *flag* poste "attive" e che, nel passaggio da uno stato ad un altro, è implicita la loro "inattivazione"). Di seguito, una dettagliata descrizione di ognuno di essi:

- INIT: è lo stato di reset della FSM, il quale "attende" il segnale *i_start* = 1 al fine di effettuare il cambio di stato a MEM_READY. Il perdurare di *i_start* = 0 implica la permanenza nello stato INIT.
- MEM_READY: durante la persistenza in questo stato, è certo il fatto che il valore del segnale *i_mem_data* sia effettivamente l'elemento letto nella cella di memoria all'indirizzo *o_mem_addr*. Se *i_mem_data* è 'zero', lo stato successivo sarà ZERO_VALUE, altrimenti, NON_ZERO_VALUE.
- NON_ZERO_VALUE: questo stato si riferisce alla lettura dalla memoria di un valore della sequenza diverso da 'zero'. Di conseguenza, la FSM rende possibile la scrittura del sopracitato intero nello *stored value* di "register_8bit" tramite la flag *o_register_we*, grazie a *o_counter_increment* viene permesso l'aggiornamento del valore memorizzato in "counter" e, tramite *o_rev_count_reset*, viene ripristinato a '31' il valore all'interno di "reverse_counter".
- ZERO_VALUE: questo stato si riferisce alla lettura dalla memoria di un valore della sequenza uguale a 'zero'. Dunque, avviene la selezione, tramite *o_mux_select*, del corretto *stored value* da riportare nella presente cella di memoria, ovvero, quello di "register_8bit", la cui scrittura viene "autorizzata" grazie al segnale *o_mem_we* e, infine, è permesso l'aggiornamento del valore memorizzato in "counter" grazie a *o_counter_increment*.
- SET_CREDIBILITY: nell'ambito di questo stato, viene effettuata la scrittura in memoria del valore di credibilità relativo al valore della sequenza immediatamente precedente. Viene selezionato il corretto *stored value* da riportare in memoria (ovvero, quello di "reverse_counter") grazie a *o_mux_select*, avviene

l'aggiornamento del valore memorizzato in "counter" grazie a *o_counter_increment* e l'operazione di scrittura viene autorizzata tramite il segnale *o_mem_we*. E' fondamentale notare che la scrittura del valore in memoria avvenga prima che l'incremento del valore memorizzato in "counter" causi il mutare dell'indirizzo di memoria *o_mem_addr*, a causa dei tempi di propagazione dell'hardware. Nell'istante di tempo immediatamente precedente ad un nuovo cambio di stato il valore di *o_mem_addr* risulterà aggiornato, dunque, pronto per essere fruito per una nuova operazione.
Al fine di proseguire l'esecuzione, viene selezionato lo stato successivo comparando *i_word_count* (segnale costituito dai 10 bit più significativi dello *stored value* di "counter") a *i_k*: se uguali, si procede nello stato DONE, altrimenti, se diversi, si proceda in READ_MEM. Infatti, essendo il valore memorizzato in "counter" incrementato due volte per ogni elemento della sequenza, i 10 bit più significativi di esso rappresentano il numero di interi elaborati fino ad un dato istante.

- READ_MEM: in questo stato, viene atteso un ciclo di clock al fine di poter garantire la correttezza del valore del segnale *i_mem_data* una volta raggiunto lo stato MEM_READY. Inoltre, viene decrementato, grazie a *o_rev_count_decrement*, il valore memorizzato in "reverse_counter", in prospettiva di una successiva lettura.
- DONE: si giunge nel presente stato in corrispondenza del termine dell'elaborazione di una sequenza. Dunque, il segnale *o_done* viene posto ad '1'. La FSM permane nel presente stato fintantoché *i_start* = 1.

MODULO VI: project_reti_logiche

Questo modulo contiene tutti gli altri componenti e, all'interno dell'architettura, sono specificati i collegamenti diretti tra segnali dei diversi sottomoduli. L'entity del presente componente è definita dalla specifica progettuale e si interfaccia direttamente con la memoria e i test-benches.

Tra i collegamenti definiti in questo modulo, sono degni di nota i segnali di reset e *o_mem_en* (come presentati nel paragrafo “aspetti generali”) e il segnale *o_mem_addr*, il quale è il risultato della somma tra l’output *o_data* di “counter” e *i_add*.

RISULTATI SPERIMENTALI

SINTESI PROGETTUALE

Al fine di sintetizzare il progetto, abbiamo utilizzato il software Xilinx Vivado 2023.2 e lavorato con un FPGA target di tipologia Artix-7 FPGA xc7a200tfbg484-1.
Analizzando i report prodotti in post-sintesi da Vivado grazie ai comandi *report_utilization* e *report_timing*, si evince come il componente progettato rispetti in ampia maniera i requisiti della specifica progettuale: in particolare, si noti l’assenza di *slice registers* di tipo *latch* ed il valore di *slack time,* pari a 16.509 nanosecondi.
Di seguito, una parziale rappresentazione dell’output del comando *report_utilization*, in particolare, la tabella “Slice logic”.

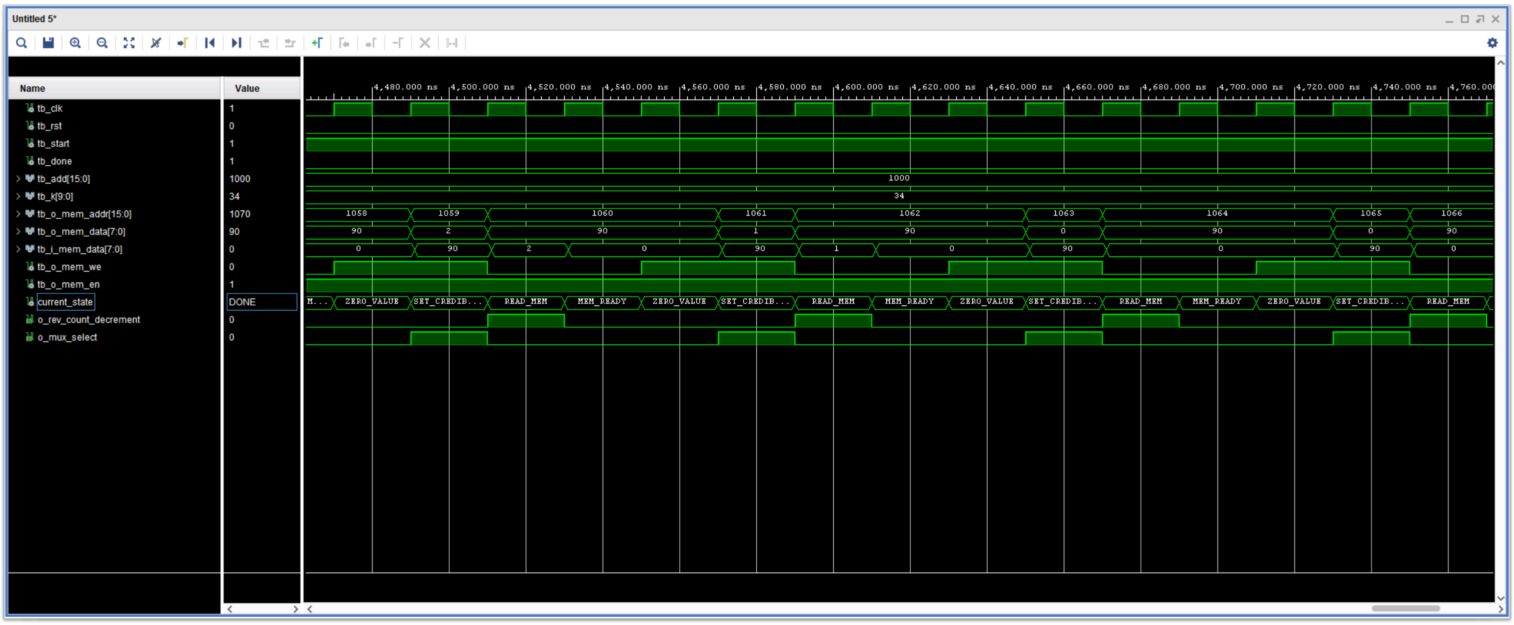
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	45	0	0	134600	0.03
LUT as Logic	45	0	0	134600	0.03
LUT as Memory	0	0	0	43200	0.00
Slice Registers	27	0	0	269200	0.01
Register as Flip Flop	27	0	0	269200	0.01
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

SIMULAZIONI

Abbiamo testato il componente progettato sia grazie al test-bench ufficiale fornito in concomitanza alla specifica, sia sviluppando autonomamente alcuni casi di test particolarmente interessanti, di seguito riportati.

TEST-BENCH I: più di 31 “zeri” consecutivi in sequenza

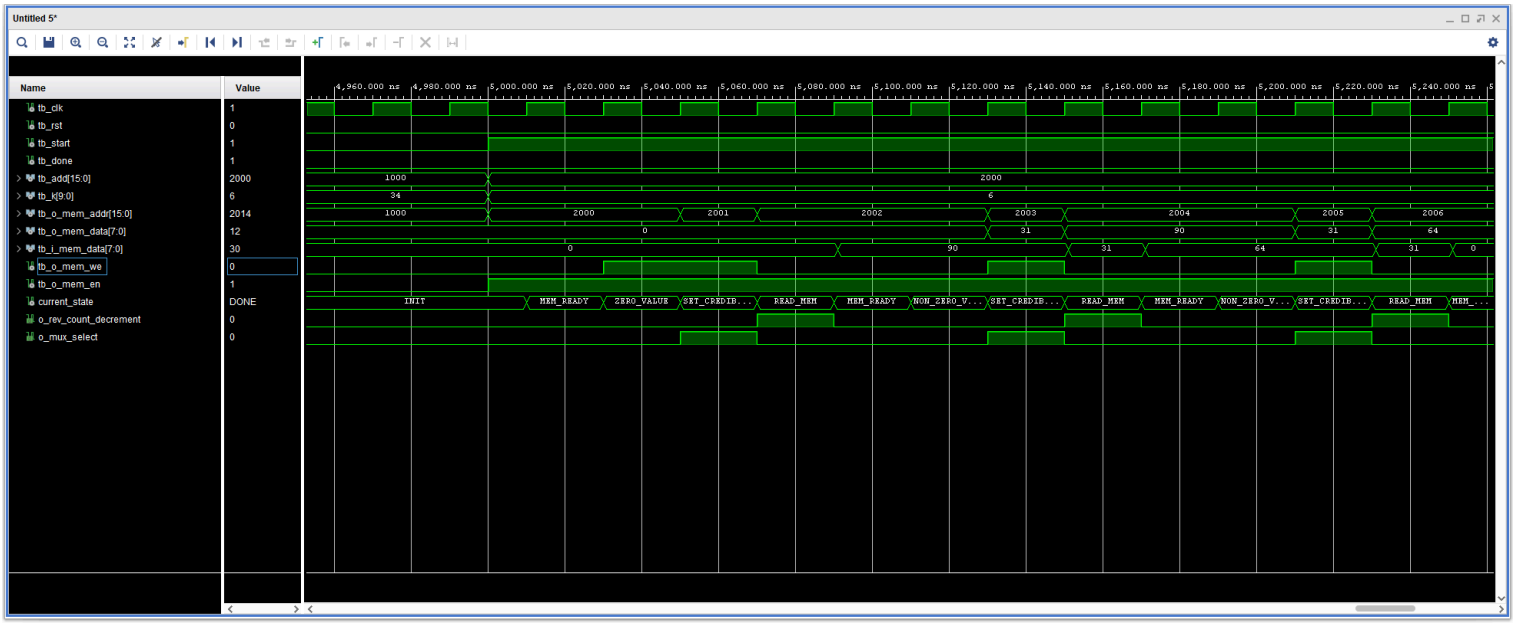
-INPUT: 90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
-OUTPUT: 90, 31, 90, 30, 90, 29, 90, 28, 90, 27, 90, 26, 90, 25, 90, 24, 90, 23, 90, 22, 90, 21, 90, 20, 90, 19, 90, 18, 90, 17, 90, 16, 90, 15, 90, 14, 90, 13, 90, 12, 90, 11, 90, 10, 90, 9, 90, 8, 90, 7, 90, 6, 90, 5, 90, 4, 90, 3, 90, 2, 90, 1, 90, 0, 90, 0, 90, 0



Questo test è stato sviluppato al fine di verificare il corretto assegnamento del valore di credibilità ad ogni elemento della sequenza. In particolare, è necessario che il modulo “reverse_counter” non vada in underflow. Tutti quanti i valori di credibilità a seguito del trentunesimo devono essere posti a ‘0’.

TEST-BENCH II: “zeri” all’inizio della sequenza

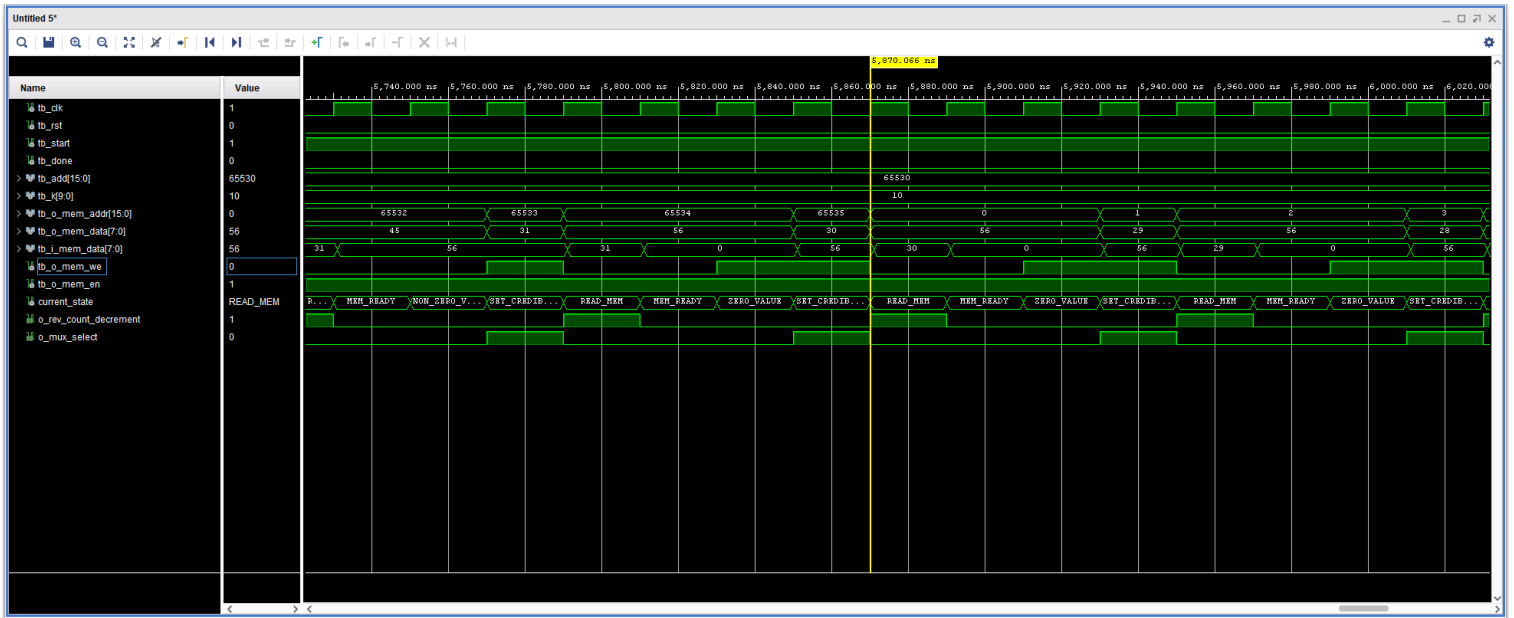
-INPUT: 0, 0, 90, 0, 64, 0, 0, 0, 56, 0, 12, 0
-OUTPUT: 0, 0, 90, 31, 64, 31, 64, 30, 56, 31, 12, 31



Questo test è stato sviluppato al fine di verificare il corretto funzionamento del sistema nella casistica in cui, all’inizio della sequenza numerica, sia presente un valore uguale a ‘0’: dunque, è previsto che il relativo valore di credibilità sia posto anch’esso a ‘0’.

TEST-BENCH III: overflow dell’indirizzo di memoria

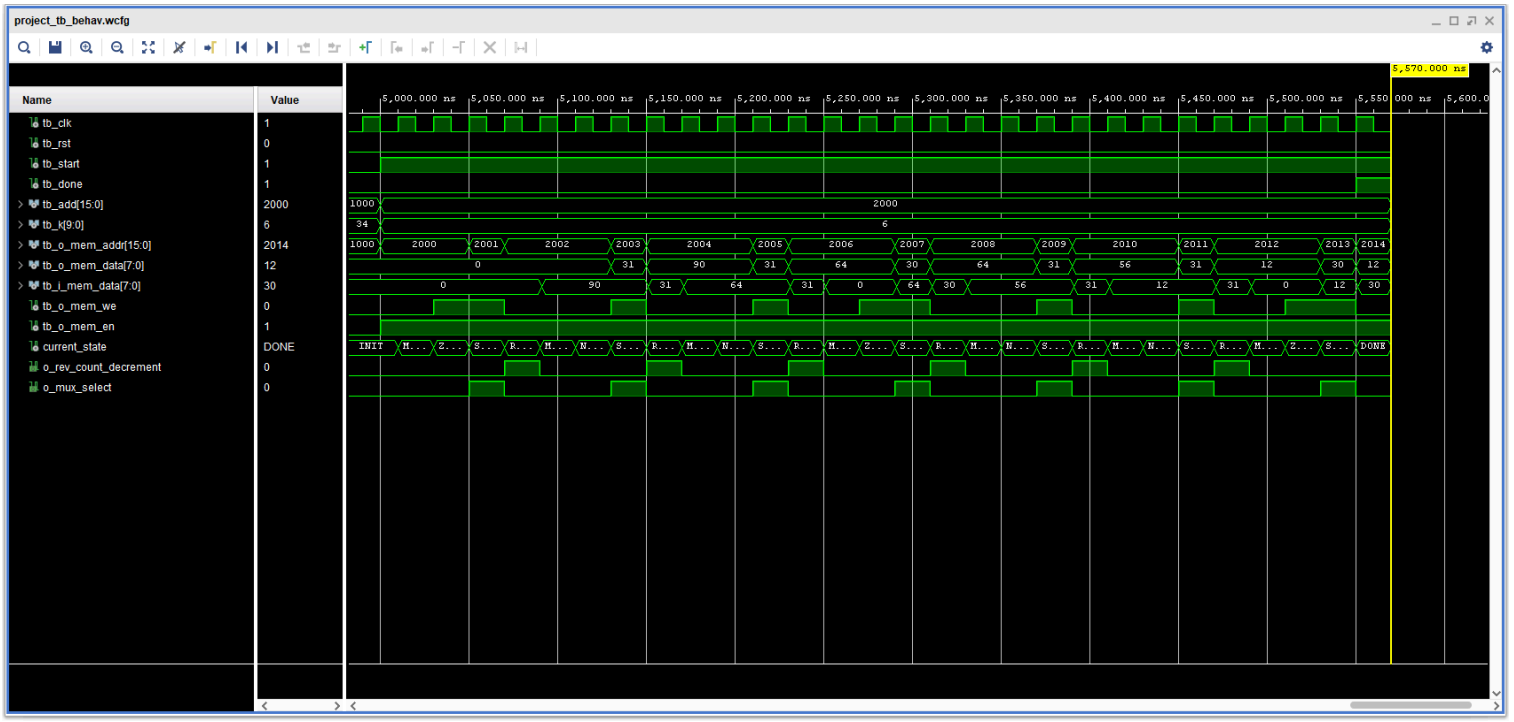
-INPUT: 45, 0, 56, 0, 0, 0, 0, 0, 0, 0, 12, 0, 128, 0, 128, 0, 31, 0, 0, 0
-OUTPUT: 45, 31, 56, 31, 56, 30, 56, 29, 56, 28, 12, 31, 128, 31, 128, 31, 31, 31, 31, 30
-INDIRIZZO DI MEMORIA DI PARTENZA: 65530; ULTIMO INDIRIZZO: 65535



Questo test è stato sviluppato al fine di testare il funzionamento del componente nel caso in cui, durante l'esecuzione, si verifichi un overflow dell'indirizzo di memoria sul quale svolgere operazioni di lettura o scrittura. In questa casistica il sistema continua a funzionare normalmente, spostandosi direttamente dall'indirizzo di memoria '65535' all'indirizzo '0'.

TEST-BENCH IV: valori "spazzatura" in celle di memoria dei byte di credibilità

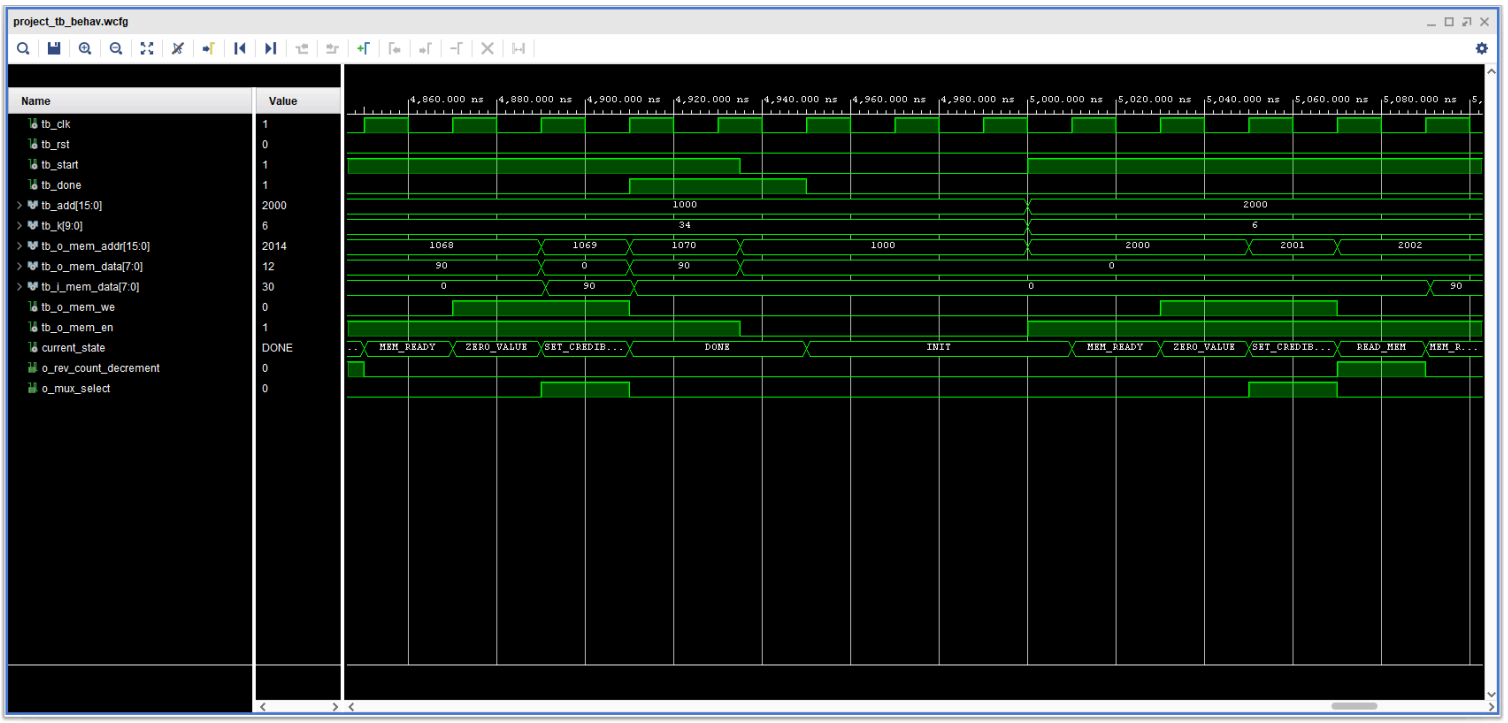
-**INPUT**: 0, 4, 90, 3, 64, 2, 0, 1, 56, 25, 12, 46
-**OUTPUT**: 0, 0, 90, 31, 64, 31, 64, 30, 56, 31, 12, 31



Questo test è una versione opportunamente modificata del test-bench II, sviluppato al fine di testare l'eventualità in cui nelle celle di memoria adibite ai valori di credibilità siano già presenti valori non coerenti con il relativo elemento della sequenza, verificando che vengano correttamente sovrascritti con il giusto valore di credibilità (anche nel caso in cui questo dovesse risultare essere 0).

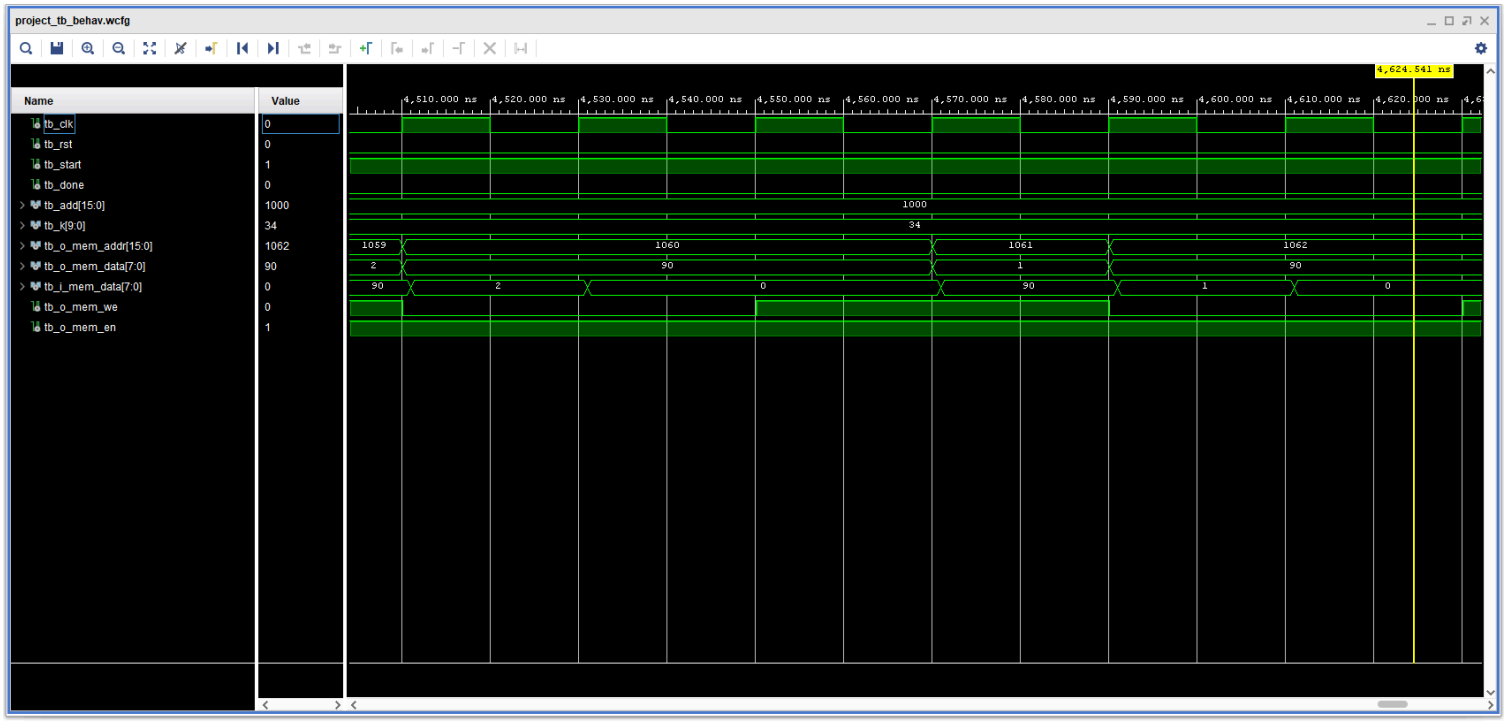
TEST-BENCH V: sequenze multiple

In quest'ultimo test-bench abbiamo voluto testare il progetto ponendo in input tutte quante le precedenti sequenze, senza fare uso del segnale di reset. Questo ci ha permesso di verificare che il componente fosse in grado di gestire in maniera indipendente sequenze in serie separate da un periodo nel quale il segnale *i_start* è posto a '0'.



NOTA GENERALE SULLE SIMULAZIONI POST-SINTESI

Nella *waveform* riportata di seguito è possibile notare chiaramente il *delay* tra il *rising edge* del *clock*, l'aggiornamento del segnale *o_mem_addr* (controllato da "counter") e il segnale *i_mem_data* (fornito dalla memoria modellata all'interno dei test-bench). A causa della presenza di questo *delay* è stato necessario adeguare la struttura della *FSM* aggiungendo lo stato READ_MEM, in modo tale da assicurare che nel ciclo di clock successivo (ovvero, una volta raggiunto lo stato MEM_READY) *i_mem_data* sia adeguatamente aggiornato, coerentemente all'indirizzo fornito tramite *o_mem_addr*.



CONCLUSIONI

Il progetto sviluppato è totalmente conforme alla specifica progettuale data e alle richieste fornite. Le prestazioni del componente rientrano ampiamente nei limiti posti e, come specificato, la sintesi del progetto non genera alcun *inferred latch*.
Le scelte di design hanno permesso di concretizzare un modulo hardware caratterizzato da componenti distinti e potenzialmente riutilizzabili: la logica implementativa risulta separata dai componenti e condensata nella *FSM*, coadiuvata da alcune precise ed oculate scelte di definizione dell'architettura del multiplexer.