Relazione del Progetto di Reti Logiche

Prof. Salice - 2022-2023
Stefano Morano (Codice Persona 10737463 – Matricola 959671)
Alessandro Mancini (Codice Persona 10733387 – Matricola 957908)

Sommario

Introduzione

1.1 Scopo del progetto

Il progetto consiste nell'implementazione di un modulo Hardware in VHDL. Tale modulo deve essere capace di ricevere in input un numero di bit variabile, leggere valori contenuti in memoria e riportarli in output. Il compito della macchina consiste nella lettura di un indirizzo di memoria in input e nella stampa del contenuto della memoria a tale indirizzo. Quest'ultima deve avvenire su una delle 4 uscite scelte in base all'input. Questo compito deve essere svolto molteplici volte, fino alla terminazione della lettura dei bit ricevuti.

1.2 Specifiche generali

Il modulo HW si serve di diverse porte di input e di output.

- Oltre ai segnali di clock e reset, la macchina ha bisogno di un segnale che le comunichi quando inizia e finisce la lettura dei dati per ciascuna sua esecuzione. Questo segnale è detto *i_start* ed avrà valore '1' in corrispondenza dell'acquisizione dei bit.
  La lettura dei bit ricevuti in input avviene attraverso un segnale di ingresso detto *i_w*. Ogni lettura di quest'ultimo comprende dai 2 ai 18 bit: i primi 2 indicano l'uscita sulla quale stampare l'output, mentre i seguenti bit rappresentano l'indirizzo di memoria da cui prendere il valore, eventualmente esteso a 16 bit.
  Infine, per ricevere il dato contenuto nella cella di memoria scelta, viene usato un segnale di input chiamato *i_mem_data*.

- La macchina possiede 8 segnali di output: 4 segnali di uscita dove riportare i valori letti dalla memoria, 2 per gestire lettura e scrittura in memoria (*o_mem_en* e *o_mem_we*), uno per comunicare alla memoria l'indirizzo dal quale estrarre il dato (*o_mem_addr*) e uno che notifica il ciclo di clock in cui avviene la stampa dei valori sulle uscite (*o_done*). Facciamo notare che per quanto riguarda la specifica di questo progetto, non avremo mai bisogno di scrivere in memoria.

Esempio:

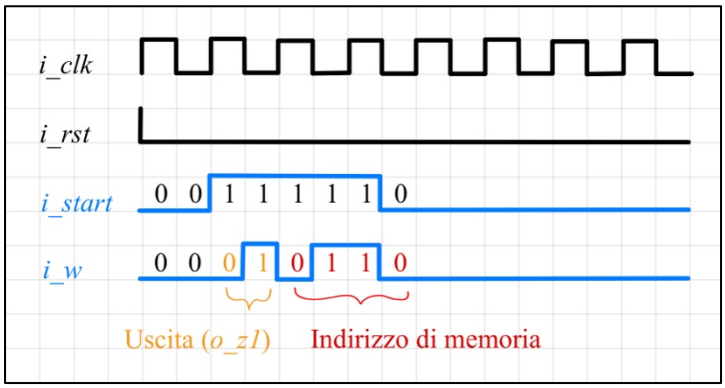

Immagine 1.1: Esempio di segnali di input

1) Quando *i_start* = 1: leggiamo *i_w*.

2) I primi 2 bit si riferiscono all'uscita:

00	o_z0
01	o_z1
10	o_z2
11	o_z3

3) I seguenti bit si riferiscono all'indirizzo di memoria, il quale andrà esteso a 16 bit.

L'indirizzo esteso verrà usato per reperire il dato dalla memoria, in corrispondenza del segnale *o_mem_en = 1*. Quando *o_done = 1*, Il dato ottenuto verrà stampato sull'uscita *o_z1*, mentre le altre uscite mostreranno l'ultimo dato memorizzato.

1.3 Interfaccia del componente e segnali

I segnali di input sono:

- I segnali di **CLOCK** (*i_clk*) e di **RESET** (*i_rst*).
- Il segnale di **WRITE** (*i_w*): indica la stringa di bit in cui sono compresi il numero di uscita nella quale stampare il valore e l'indirizzo da cui prendere il valore nella memoria.
- Il segnale di **START** (*i_start*): gestisce l'inizio e la fine di ogni lettura del segnale *i_w*.
- Il segnale **MEMORY DATA** (*i_mem_data*): riceve il dato dalla memoria, composto da 8 bit.

I segnali di output sono:

- I segnali **Z0**, **Z1**, **Z2**, **Z3** (*i_z0, i_z1, i_z2, i_z3*): indicano le uscite, da 8 bit.
- Il segnale **DONE** (*o_done*): notifica quando viene stampato il valore delle 4 uscite.
- I segnali alla memoria: **MEMORY WRITE ENABLE** (*o_mem_we*) che abilita la scrittura su memoria, **MEMORY (READ) ENABLE** (*o_mem_en*) che abilita la lettura della memoria.
- Il segnale **MEMORY ADDRESS** (*o_mem_address*): comunica alla memoria l'indirizzo ricevuto da *i_w*.

Architettura

2.1 Circuito

Lo schema del circuito riportato in seguito è il prototipo del circuito in fase di pre-implementazione. I vari moduli sono gestiti tramite opportuni segnali di load e di reset, per avere una completa gestione della macchina a stati.

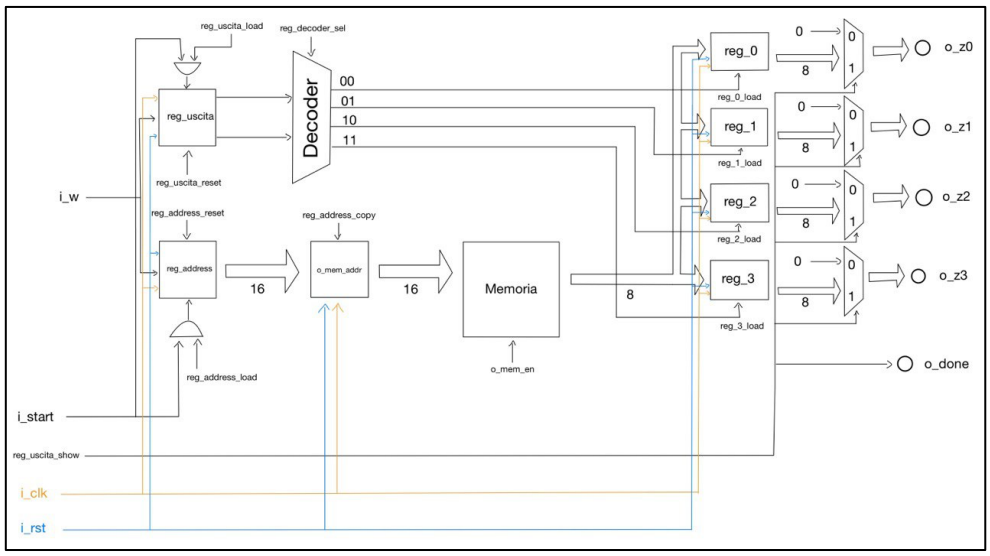

Immagine 2.1: Schema del progetto logico

Di seguito vengono spiegate le caratteristiche e funzionalità dei diversi moduli:

- Il **registro *reg_uscita*** (2 bit) viene usato per memorizzare i bit che indicano l'uscita selezionata. Riceve in input un singolo bit della *i_w* ad ogni ciclo di clock. I bit vengono salvati attraverso uno shift register, attivato dal segnale *reg_uscita_load* posto in and logico con *i_start*. Quest'ultimo è necessario per evitare di attivare lo shift register quando *i_start* = 0, ovvero quando non dobbiamo leggere da *i_w*. Il reset del registro è gestito dal segnale *reg_uscita_reset*. I segnali del modulo sono tutti sincroni: ogni azione viene compiuta ad ogni ciclo di clock.

- Il **registro *reg_address*** (16 bit) serve per memorizzare i 16 bit dell'indirizzo di memoria. Riceve in input un singolo bit della *i_w* ad ogni ciclo di clock. I bit vengono salvati attraverso uno shift register attivato dal segnale *reg_address_load* posto in and logico con *i_start*. Il reset del registro viene gestito dal segnale *reg_address_reset*. Il registro memorizza bit fino a quando *i_start* vale '1'. I segnali del modulo sono tutti sincroni: ogni azione viene compiuta ad ogni ciclo di clock.

- La **memoria** prende in input il valore di *o_mem_address*, quando *o_mem_en* = 1. Essa manda in output il valore del dato corrispondente all'indirizzo ricevuto. Questo valore (8 bit) viene ricevuto dalla nostra macchina tramite l'input *i_mem_data*.

- Il **decoder** ha il compito di attivare il segnale di load corrispondente all'uscita indicata da *i_w*. Prende in ingresso *reg_uscita*, e pone a valore '1' un segnale tra: *reg_0_load*, *reg_1_load*, *reg_2_load*, *reg_3_load*. Ciò avviene solo quando reg_*decoder_sel* = 1. Il modulo è asincrono: la sua funzione è indipendente dai cicli di clock.

- I **registri reg_0, reg_1, reg_2, reg_3** (8 bit ciascuno) sono i registri che salvano i valori delle uscite. Essi ricevono il valore di *i_mem_data* nel momento in cui i rispettivi segnali di load (*reg_0_load*, *reg_1_load*, *reg_2_load*, *reg_3_load)* vengono settati a '1' dal decoder. I segnali del modulo sono tutti sincroni: ogni azione viene compiuta ad ogni ciclo di clock.

- Nel circuito sono presenti 4 **multiplexer**, uno per ciascun registro di uscita (reg_0, reg_1, reg_2, reg_3). L'output del multiplexer viene gestito dal segnale *reg_uscita_show*. Viene stampato '0' se *reg_uscita_show* vale '0', in caso contrario viene stampato il valore del registro di uscita corrispondente al multiplexer. L'output del multiplexer viene salvato nelle uscite della macchina *o_z0, o_z1, o_z2, o_z3*. Il modulo è asincrono: la sua funzione è indipendente dai cicli di clock. Il segnale *o_done* ha sempre lo stesso valore del segnale *reg_uscita_show*.

2.2 Macchina a stati

Per sviluppare il progetto abbiamo progettato una macchina a 9 stati: la seguente immagine rappresenta l'idealizzazione degli stati e delle funzionalità della macchina per ognuno di essi.

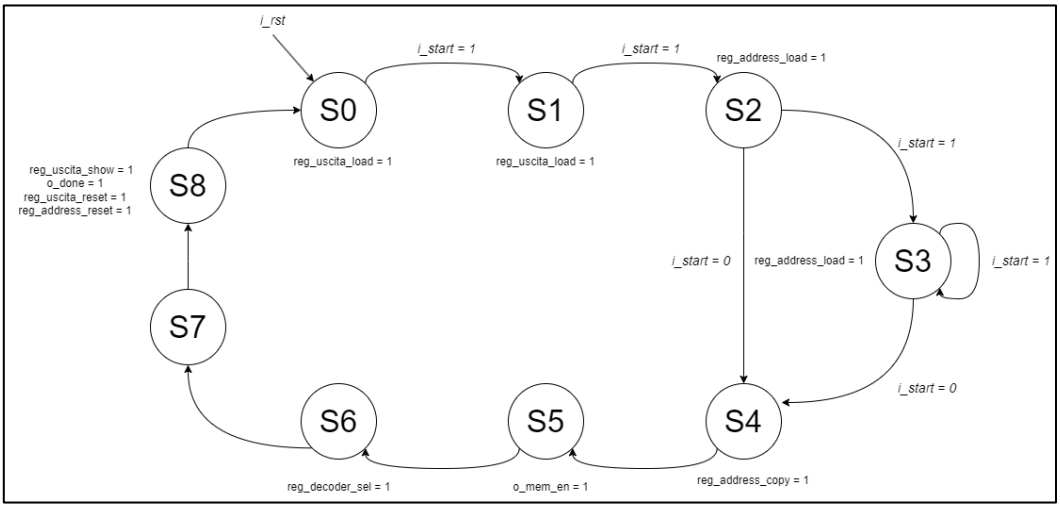

Immagine 2.2: Schema della macchina a stati

Segue la spiegazione dettagliata di ciò che avviene durante ciascuno stato:

- Lo stato **S0** è lo stato di reset. Se il segnale *i_rst* = 1, la macchina passa a questo stato e tutti i moduli vengono resettati. Il *reg_uscita_load* viene settato a '1', in attesa che anche *i_start* valga 1 per iniziare a salvare i bit di *i_w* su *reg_uscita*. Quando *i_start* = 1, la macchina passa allo stato S1.

- Lo stato **S1** è uno stato necessario per il salvataggio del primo bit in *reg_uscita*. Questo registro infatti è composto da 2 bit che vengono aggiornati se *reg_uscita_load* e *i_start* hanno valore 1. Durante S1 si ha il solo scopo di salvare nel registro il primo bit e di passare al salvataggio del secondo in S2, lasciando inalterati i segnali *reg_uscita_load* e *i_start*. Il passaggio allo stato S2 avviene senza condizioni aggiuntive.

- Nello stato **S2** avviene il salvataggio in *reg_uscita* del secondo e ultimo bit, preso da *i_w*. Come in S1, la permanenza in questo stato dura un ciclo di clock. Il segnale *reg_address_load* viene posto a '1', per far si che inizi il salvataggio dell'indirizzo su *reg_address* da *i_w* in caso si passi allo stato S3. Ciò avviene se *i_start* rimane uguale a '1', altrimenti lo stato successivo diventa S4.

- Nello stato **S3** avviene la memorizzazione dell'indirizzo di memoria in *reg_address* da *i_w*. La permanenza in questo stato sarà necessariamente di 16 cicli di clock al massimo e termina quando *i_start* cambia valore da '1' a '0'. Il segnale *reg_address_load* rimane a 1 per tutta la durata dello stato, per garantire il funzionamento dello shift register. Successivamente si passa a S4.

- Nello stato **S4** viene memorizzato l'ultimo bit dell'indirizzo nel *reg_address*, preso da *i_w*. Viene settato a '1' il segnale di *reg_address_load*, che attiverà la copia di *reg_address* sull'uscita

o_mem_addr, nello stato successivo. La macchina permane in questo stato per un ciclo di clock. Successivamente si passa allo stato S5.

- Nello stato **S5** il valore di *reg_address* viene caricato sull'uscita *o_mem_addr*. Il segnale di *o_mem_en* viene settato a '1' per abilitare la lettura della memoria nello stato successivo. La macchina permane in questo stato per un ciclo di clock. Successivamente si passa allo stato S6.

- Nello stato **S6** il valore dato in output dalla memoria viene caricato sull'ingresso *i_mem_data*. Il segnale reg_*decoder_sel* viene settato a '1' per attivare il decoder nello stato successivo. La macchina permane in questo stato per 1 ciclo di clock. Successivamente si passa allo stato S7.

- Nello stato **S7**, il decoder attiva un segnale tra: *reg_0_load*, *reg_1_load*, *reg_2_load*, *reg_3_load*. Il valore di *i_mem_data* viene caricato nell'uscita corrispondente al segnale attivato. Anche questo stato dura 1 ciclo di clock. Successivamente si passa allo stato S8.

- Nello stato **S8** i segnali *reg_uscita_show* e *o_done* vengono settati a 1. Le uscite *o_z0, o_z1, o_z2, o_z3*, prendono il valore dei rispettivi registri *reg_0, reg_1, reg_2, reg_3*. La macchina termina così l'esecuzione relativa all'input ricevuto. Vengono posti *reg_address_reset* e *reg_uscita_reset* a '1', in modo che i due registri *reg_uscita* e *reg_address* vengano resettati nel prossimo stato e siano pronti per la successiva esecuzione della macchina. Lo stato S8 dura un ciclo di clock, successivamente si ritorna allo stato S0.

2.3 Scelte progettuali

Il componente HW è stato sviluppato in linguaggio VHDL, tramite il software Xilinx Vivado, con FPGA target di tipo Artix-7 FPGA xc7a200tfbg484-1. Per gestire i moduli progettuali, abbiamo definito i seguenti processi.

- **Processo per la gestione del clock e del reset:** aggiorna lo stato ad ogni salita del clock, tramite l'istruzione *cur_state* = *next_state*. Inoltre, setta la macchina allo stato S0 in caso di *i_rst* = 1.

- **Processo per la gestione del cambio stato:** analizza il valore di *i_start* e in base ad esso cambia il valore di *next_state*.

- **Processo per la gestione dei segnali:** i vari segnali utilizzati in questo processo vengono posti a '1' o a '0' in base al valore di *cur_state*, ovvero allo stato in cui si trova la macchina. Questo processo gestisce tutti i segnali usati per attivare altri processi. Ad esempio, nello stato S4 viene attivato il segnale *reg_address_copy,* che indica al processo responsabile di svolgere la copia di *reg_address* in *o_mem_address.*

- **Processi per la gestione dei registri:** attivati dal fronte di salita del clock oppure dal segnale di reset. I registri *reg_0*, *reg_1*, *reg_2*, *reg_3*, *reg_uscita* e *reg_address* hanno ciascuno un proprio processo. Essi vengono caricati o meno in base al valore del segnale di load corrispondente, controllato ad ogni ciclo di clock. In caso di *i_rst* = 1, si resettano.

- **Processo per la gestione di *o_mem_address*:** si occupa della copia di *reg_address* in *o_mem_address,* nel caso in cui il segnale *reg_address_copy* sia uguale a '1'. In caso di *i_rst = 1,* si resetta.

- **Processo per la gestione del decoder:** in base al segnale *reg_decoder_sel,* il decoder svolge o meno la sua funzione (illustrata al paragrafo 2.1).

I 4 multiplexer sono gestiti dal segnale *reg_uscita_show*, in modo asincrono. Per questo motivo non appartengono ad un processo.

Risultati sperimentali

3.1 Sintesi

Il processo di sintesi del codice VHDL, effettuato dal software Vivado, ha esito positivo. L'HW vanta buoni parametri di efficienza, descritti da 2 tipi di report ottenuti tramite command line:

- Secondo il **Timing report**, la macchina ha un eccellente tempo d'esecuzione. La voce Slack-Timing indica il tempo rimanente, dopo l'esecuzione, al raggiungimento del limite di 100ns. Nel nostro caso lo Slack-Timing vale 97,311ns, perciò è ottimale.

```
Slack (MET) :             97.311ns  (required time - arrival time)
  Source:                 FSM_sequential_cur_state_reg[2]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Destination:            reg_0_reg[0]/CE
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.307ns  (logic 0.751ns (32.553%)  route 1.556ns (67.447%))
```

Immagine 3.1: Timing report

- L'**Utilization report** descrive il numero di moduli impiegati per la sintesi virtuale del componente HW. Quest'ultimo risulta molto robusto, in quanto sono stati utilizzati registri Latch. Questi registri vengono talvolta inseriti in post sintesi per mantenere in memoria il valore di segnali non specificati in alcune fasi del circuito. Nel nostro caso, invece, ogni registro e segnale ha sempre un valore assegnato ad ogni stato della macchina. Il numero di Flip Flop risulta coerente rispetto all'implementazione pre-sintesi della macchina, dimostrando l'effettivo funzionamento del sistema anche in post sintesi.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	51	0	134600	0.04
LUT as Logic	51	0	134600	0.04
LUT as Memory	0	0	46200	0.00
Slice Registers	70	0	269200	0.03
Register as Flip Flop	70	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Immagine 3.2: Utilization report

Il risultato della sintesi dell'HW è rappresentato dall'immagine seguente, sviluppata dal software Vivado sulla base del codice VHDL:

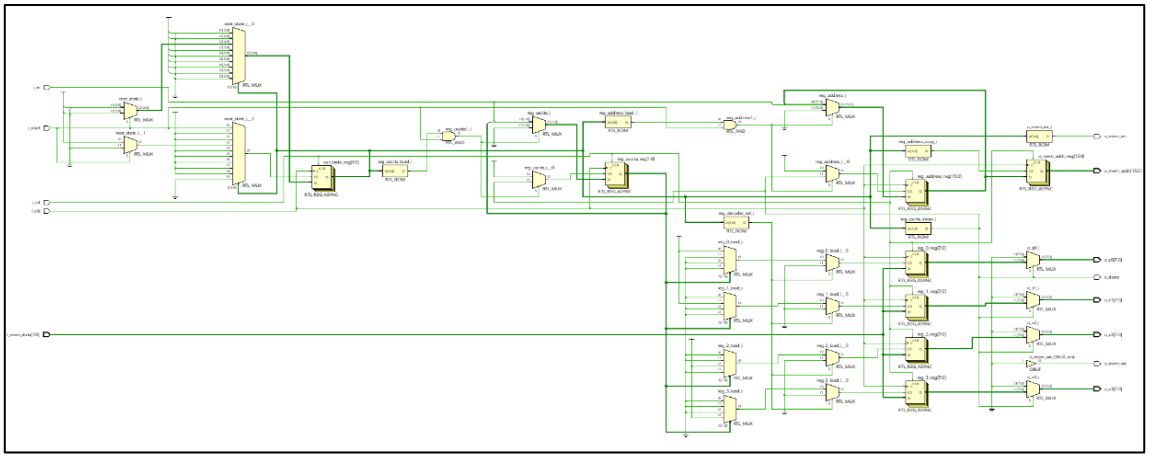

Immagine 3.3: Schema della sintesi dell'HW, stampato da Vivado

Il progetto logico iniziale è molto simile alla struttura della macchina osservabile nell'immagine. Questo dimostra una buona implementazione della scelta progettuale.

3.2 Simulazioni

Per verificare il corretto funzionamento della macchina, sono state svolte simulazioni dei 7 test bench d'esempio. Inoltre, il nostro progetto è stato sottoposto a 6 test aggiuntivi per controllare i casi limite e operazioni più complesse. Di seguito sono riportate le descrizioni di alcuni test, con relative immagini delle forme d'onda dei segnali durante la simulazione in post sintesi.

3.2.1 Test dell'output

1) Questo test è utile per testare la capacità della macchina di ricevere una stringa d'ingresso di lunghezza massimale, ovvero di 18 bit. Nella prima immagine la macchina riceve diciotto '0' dal segnale i_w. Ciò implica la scrittura sull'uscita *o_z0* del dato contenuto all'indirizzo di memoria formato da sedici '0'. Nella seconda invece si ricevono diciotto '1'. La macchina, quindi, scriverà in *o_z3* il dato contenuto all'indirizzo di memoria composto da sedici '1'.

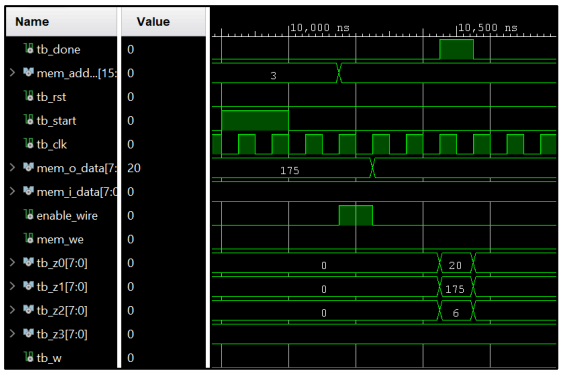

Immagine 3.4: Test 1, soli 0 in input

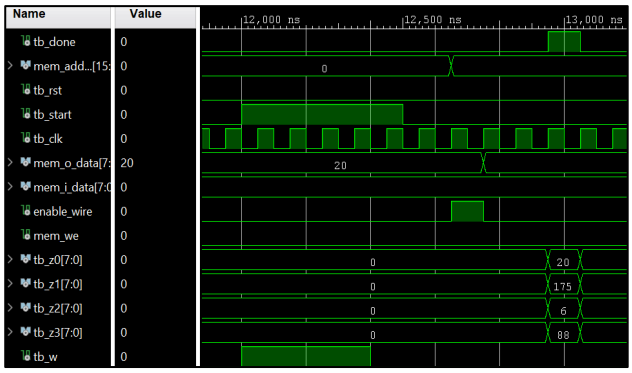

Immagine 3.5: Test 1, soli 1 in input

2) Simuliamo il caso in cui *i_w* è formata solo da due bit che valgono '01'. Questo test è utile per verificare il corretto funzionamento della macchina a stati. Dopo la lettura dei due bit, che si riferiscono all'uscita, ci si aspetterebbe la lettura dell'indirizzo di memoria. Quest'ultimo però non ci viene fornito, e la macchina riceve segnale *i_start* = 0. L'FSA in questo caso bypassa lo stato S3 invocando direttamente lo stato S4. La macchina continua a svolgere il suo funziomaneto considerando come default address quello formato da sedici '0'.

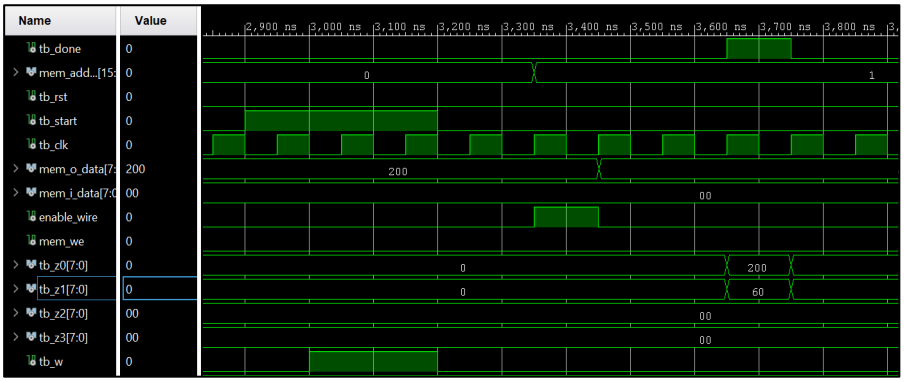

Immagine 3.6: Test 2, i_w = 01

3) In questa simulazione si testa la sovrascrittura dei dati nei registri delle uscite. Viene ripetuta la stampa di valori diversi sulla stessa uscita. Il registro associato all'uscita dovrà perciò sovrascrivere il proprio valore, affinché il dato stampato sia sempre il più recente.

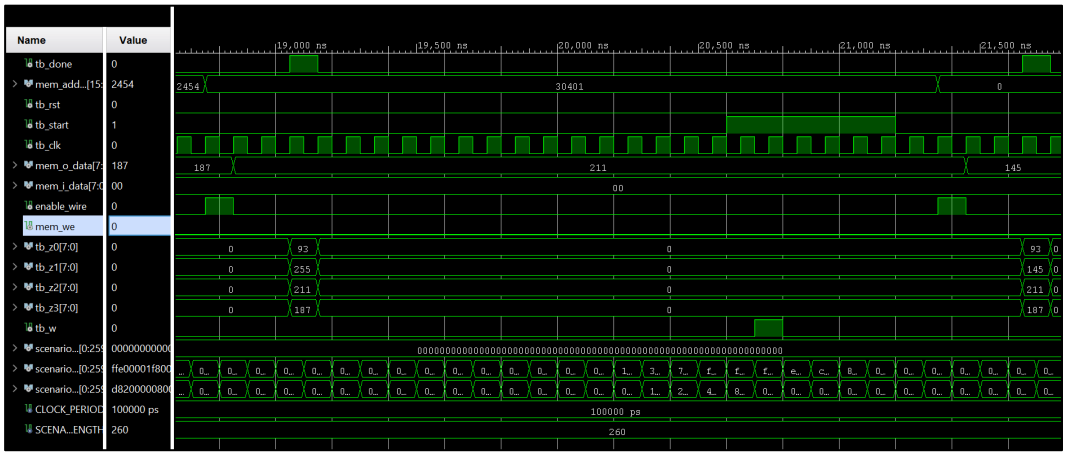

Immagine 3.7: Test 3, uscita o_z1 sovrascritta

3.2.2 Test dei segnali

1) In questa simulazione viene posto *i_rst* = 1 durante la fase di elaborazione della memoria. Il test verifica il corretto funzionamento del reset dei segnali sincroni e la capacità d'interruzione dell'esecuzione durante il suo svolgimento.

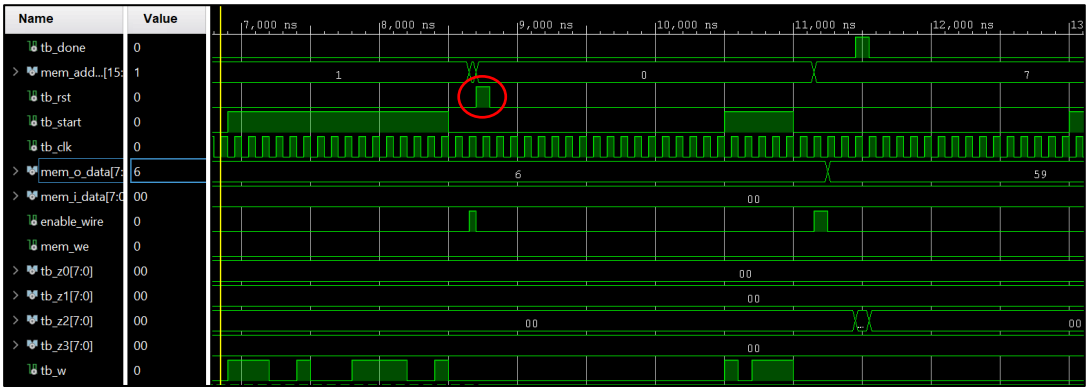

Immagine 3.8: Test 4, cerchiato in rosso i_rst = 1

2) In questa simulazione viene posto *i_rst* = 1 prima che il segnale *o_done* venga posto a '1'. Si testa che i segnali sincroni vengano resettati con successo anche in questa fase di lavoro. L'esecuzione viene interrotta e ripresa dall'inizio. I registri coinvolti sono azzerati e pronti all'uso.

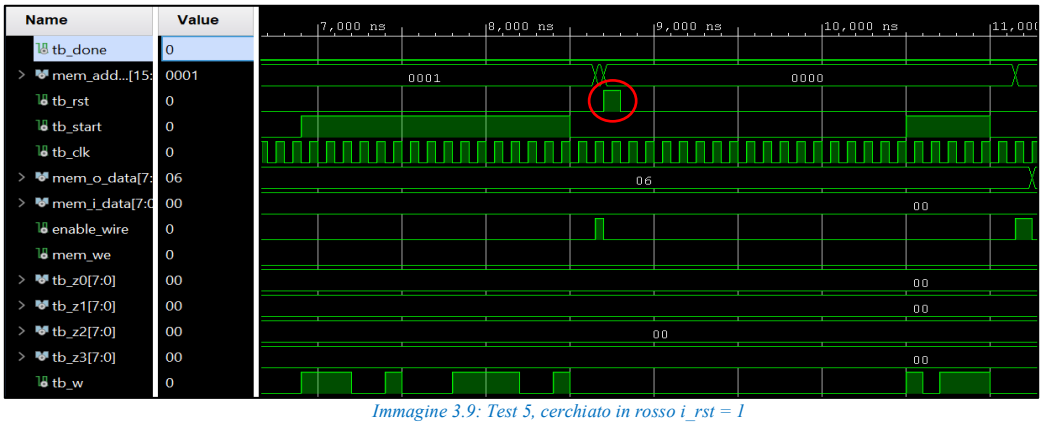

Immagine 3.9: Test 5, cerchiato in rosso i_rst = 1

3) Questa volta *i_rst* viene posto a '1' mentre *i_start* = 1. Si verifica la macchina sia capace di interrompere la trascrizione dei dati da *i_w* nei registri *reg_uscita* e *reg_address*. Gli shift register interrompono la trascrizione e vengono resettati. L'esecuzione della macchina è quindi capace di ricominciare senza problemi.

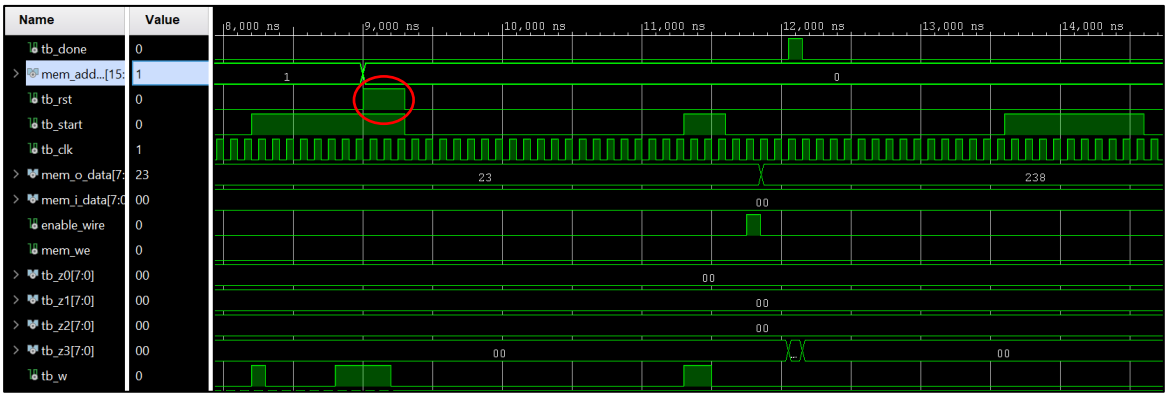

Immagine 3.10: Test 6, cerchiato in rosso i_rst = 1

Conclusioni

Durante le fasi iniziali del progetto abbiamo riscontrato difficoltà all'approccio di Vivado, per noi una novità. In seguito ad un attento studio dell'ambiente di sviluppo, abbiamo imparato a ragionare secondo la logica del linguaggio. Difatti, il nostro approccio alla programmazione, solitamente improntato all'implementazione prettamente software, ha dovuto vertere nello sviluppo di un hardware. Nonostante le problematicità, però, siamo riusciti a creare con successo la macchina descritta dalla specifica del progetto.

Quest'ultima è stata risolta mediante l'implementazione di una macchina a 9 stati. Essi sono pensati per suddividere le varie fasi del lavoro dell'hardware, rendendo l'esecuzione complessiva ben ordinata. Ad ogni stato in cui si trova la macchina vengono coinvolti vari registri per svolgere le differenti funzioni. Il risultato ottenuto è molto efficiente, come dimostrato dai parametri del Timing e dell'Utilization report. Il parametro di Slack-Timing dimostra la buona flessibilità del nostro progetto, con un valore di 97,311ns. L'assenza di registri Latch, indicata dall'Utilization report, denota l'integrità e solidità del circuito. Nessun valore, infatti, risulta indefinito durante il funzionamento del circuito HW. Tutti i test bench proposti dalla consegna del progetto sono stati superati. Inoltre, dopo la creazione e simulazione di test aggiuntivi abbiamo dimostrato come la macchina riesca a adempire al suo funzionamento anche in casi limite. La sintesi della macchina, basata sul nostro codice VHDL, viene svolta con successo. Lo schema del circuito sintetizzato, prodotto da Vivado, risulta essere molto simile allo schema del circuito sviluppato in fase di pre-implementazione. Questo, oltre che a renderci soddisfatti, denota che le scelte iniziali si sono rivelate corrette e funzionali allo sviluppo del progetto.