POLITECNICO
MILANO 1863

Progetto di Reti Logiche 2022/2023

Professore Referente di progetto: Prof. Gianluca Palermo

Maria Rita Sartini

Matricola n. 961435

Codice Persona: 10766901

Martina Quaregna

Matricola n. 959256

Codice Persona:10739683

POLITECNICO
MILANO 1863

Indice

Introduzione

Di seguito presentiamo la nostra proposta per la consegna relativa alla Prova Finale per il corso di Reti Logiche dell'anno accademico 2022-2023. Il progetto è stato sviluppato coerentemente con le specifiche riportate nel relativo *paragrafo*.

Obiettivo

L'obiettivo del progetto è implementare un componente hardware codificato in VHDL in grado di interfacciarsi con una memoria. La richiesta del progetto è infatti, lo sviluppo di un elemento in grado di leggere dati presenti da una memoria senza andarne a modificare il contenuto.

Il funzionamento prevede che, dato uno stream di bit in ingresso, il componente sia in grado di identificare un'uscita alla quale indirizzerà poi il contenuto letto dalla memoria, e l'indirizzo di lettura, identificato da 16 bit. Le richieste vengono spiegate in maniera più estesa nel paragrafo *Specifiche.*

Interfaccia del componente

L'interfaccia del componente (in *figura 1*) rappresenta i segnali necessari per il funzionamento del componente in relazione all'input stream e alla memoria:

```
entity project_reti_logiche is
port (
    i_clk : in std_logic;
    i_rst : in std_logic;
    i_start : in std_logic;
    i_w : in std_logic;
    o_z0 : out std_logic_vector(7 downto 0);
    o_z1 : out std_logic_vector(7 downto 0);
    o_z2 : out std_logic_vector(7 downto 0);
    o_z3 : out std_logic_vector(7 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_we : out std_logic;
    o_mem_en : out std_logic
);
```

figura 1

- **i_clk** è il segnale di clock;
- **i_rst** è il segnale di reset, usato per inizializzare il componente. Quando i_rst = 1 il componente si pone nello stato iniziale, le cui caratteristiche vengono mostrate nel dettaglio in seguito;
- **i_start** è il segnale di start, usato per determinare i bit letti dall'input i_w;
- **i_w** è uno stream di bit dal quale vengono letti i bit relativi all'uscita alla quale scrivere e l'indirizzo di memoria dal quale leggere;
- **o_z0, o_z1, o_z2 e o_z3** sono le quattro uscite;
- **o_done** è il segnale usato per rendere disponibile sull'uscita corretta il messaggio richiesto;
- **o_mem_addr** è un segnale che identifica l'indirizzo di memoria dal quale leggere;
- **i_mem_data** è un segnale che riporta il messaggio contenuto nell'indirizzo di memoria richiesto;
- **o_mem_we** è un segnale necessario per disabilitare la scrittura in memoria;
- **o_mem_en** è un segnale necessario per abilitare la comunicazione con la memoria (lettura e scrittura).

Specifiche

I bit letti in ingresso variano da un minimo di 2 a un massimo di 20 e sono ripartiti come raffigurato in *figura 2.*

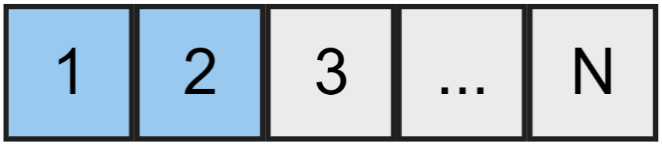

figura 2

I **primi due bit** (rappresentati in azzurro in *figura 2*) rappresentano una delle quattro uscite (Z0, Z1, Z2 o Z3), i restanti **N bit** (N rappresenta un intero che varia da 0 a 16) rappresentano l'indirizzo di memoria al quale si vuole accedere. Nel caso in cui N sia minore di 16 sarà necessario estendere l'indirizzo di memoria acquisito con degli **0** sui bit più significativi. In qualsiasi caso l'indirizzo di memoria deve avere una lunghezza di **16 bit**.

Il componente risulta attivo dopo aver ricevuto un segnale **i_rst**, a seguito del quale avviene l'inizializzazione dei segnali e il componente viene portato allo stato iniziale. Il segnale di **i_rst** è il primo ricevuto in seguito all'accensione del componente e può essere ricevuto in qualsiasi istante, sempre con il medesimo effetto. Dopo la ricezione del reset, il componente deve attendere il primo segnale alto di **i_start**, istante in cui inizia la fase di lettura. In questo intervallo i valori richiesti per i segnali di output appartenenti all'interfaccia sono tutti azzerati. Il valore di **i_w** ricevuto in questa fase viene ignorato.

La lettura dell'input è vincolata al segnale **i_start** che determina l'istante dal quale il componente inizia a leggere dall'ingresso **i_w** (lo stream di bit in ingresso). Dal valore alto di **i_start = 1** inizia la lettura dallo stream di bit in input, la lettura termina quando **i_start** passa al valore basso.

Durante l'accesso alla memoria, dopo l'acquisizione dell'indirizzo in input (con relativa regolarizzazione), i valori di **o_mem_we** e **o_mem_en** devono essere rispettivamente **0** e **1**. Questi segnali garantiscono che l'indirizzo, situato sul segnale di output **o_mem_addr**, sia ricevuto dalla memoria, la quale restituisce il dato presente a tale indirizzo sul segnale di input **i_mem_data**.

Il contenuto della memoria ricevuto è quindi esposto sull'uscita determinata dai primi due bit ottenuti dallo stream, contestualmente a un segnale alto di **o_done**. Il segnale **o_done** mantiene valore alto per un ciclo di clock, successivamente torna ad un valore basso nell'attesa di una nuova lettura. Infine, il componente si riporta ad uno stato iniziale.

Architettura

Di seguito riportiamo i segnali che sono stati utilizzati in supporto al processo con la relativa spiegazione.

FIRST_BIT

Il segnale ha il compito di regolare il salvataggio del primo bit in ingresso dall'input stream. Come gli altri segnali di supporto, questo viene inizializzato a 0 nello stato 'START'. Il valore di first_bit viene modificato nel medesimo stato come segue:

durante lo stato di START avviene una verifica relativa al valore del segnale di i_start. La verifica si propone di identificare il valore alto di i_start, avviando il processo di lettura. Se la verifica ha esito positivo, innescando l'inizio dell'acquisizione dell'input, il segnale first_bit viene assegnato a i_w prima di passare allo stato di 'EXIT_WRITING'.

POSITION

Questo segnale svolge un ruolo fondamentale sia nell'acquisizione che nella regolarizzazione dell'indirizzo ricevuto in input. Il suo valore è inizializzato a 0 nello stato di 'START' e successivamente impostato a 15 nello stato di 'EXIT_WRITING'. L'evoluzione di questo segnale segue una logica simile a quella del segnale precedente:

nello stato 'EXIT_WRITING' viene effettuata una verifica sul valore di i_start. Un valore alto comporterà l'operazione di assegnamento del primo bit del vettore saved_address (illustrato nel dettaglio in seguito) al valore di i_w e la conseguente riassegnazione di position al valore 14, in seguito si passerà allo stato di 'ADD_WRITING'.

EXIT_PORT

Il segnale è definito come un vettore di valori std_logic composto da due bit. Questo segnale rappresenta il valore (letto in input) della porta di uscita selezionata:

exit_port=00 identifica la selezione della porta di uscita z0,

exit_port=01 identifica la selezione della porta z1, e similmente per le uscite z2 (10) e z3(11).

La gestione del segnale exit_port avviene, similmente ai precedenti in più fasi: inizialmente il segnale è inizializzato nello stato di 'START' al valore '00', per essere poi modificato nello stato di 'EXIT_WRITING'. La modifica avviene in un'unica fase e segue la seguente logica: il primo bit del vettore exit_port è assegnato al valore di first_bit, il secondo è assegnato al valore di i_w.

In seguito, il valore non viene più modificato, ma viene solo letto nello stato di 'EXIT SESTUP'.

SAVED_ADDRESS

Analogamente al segnale precedente, anche questo è definito come un vettore di valori std_logic, ma di dimensione 16 bit. Il ruolo svolto da questo segnale è quello di registrare l'indirizzo acquisito e veicolarne una eventuale necessaria regolarizzazione. Come per gli altri segnali, esso è inizializzato a 0 nello stato di 'START', per essere poi modificato in prima fase nello stato di 'EXIT_WRITING' (vedi descrizione del segnale *position*) e successivamente nello stato di 'ADD_WRITING'. In questa seconda fase la modifica si svolge seguendo un assegnamento di saved_address a i_w nella posizione indicata da *position* (che viene successivamente decrementato). Lo stato di 'ADD_WRITING' si ripete fintanto che il valore di i_start è pari a 1.

Il vettore saved_address viene quindi utilizzato per normalizzare l'indirizzo di memoria nello stato di 'ADD_CONVERTING' secondo la seguente logica:

il caso di necessità di normalizzazione dell'indirizzo viene identificato grazie ad una valutazione iniziale di *position*, se quest'ultimo assume valore -1 o valore 15 non si procede alla normalizzazione. Il valore 15 indica che nessun bit è stato salvato all'interno di saved_address, caso che risulta in un indirizzo di memoria interamente composto da zeri. Il caso di position =-1 indica invece un indirizzo di memoria che è stato interamente acquisito da input e che pertanto non necessita di normalizzazione.

Il processo di normalizzazione si sviluppa quindi come di seguito: valutando il valore di position si assegna ai bit meno significativi del segnale o_mem_addr la porzione di saved_address ottenuta da input stream e si impostano i restanti bit a zero. Questa fase è gestita con uno 'switch-case'.

DATA

Il segnale data è definito come un vettore di lunghezza 8 (bit) composto da valori std_logic. Il suo ruolo è quello di registrare il valore restituito dalla memoria a seguito della lettura dell'indirizzo fornito. La sua gestione prevede una fase di inizializzazione a 0 nello stato di 'START' e una seguente fase di assegnamento nello stato di 'MEM_READING' al valore i_mem_data.

PREVIOUS_Z0, PREVIOUS_Z1, PREVIOUS_Z2, PREVIOUS_Z3

Questi segnali rappresentano la soluzione implementativa ad una richiesta illustrata nella specifica di progetto: contestualmente all'esposizione sull'uscita indicata in input del dato presente all'indirizzo di memoria acquisito, le altre uscite devono riportare l'ultimo valore che hanno mostrato. Per garantire questa funzionalità i segnali sono inizialmente assegnati a 0 nello stato di 'START', per essere poi modificati in fase di 'EXIT_SETUP'. L'idea di base della logica implementativa è quella di assegnare il dato ottenuto al segnale *previous* corrispondente all'uscita preselezionata, e non direttamente al segnale di uscita stesso, in modo da poterne mantenere memoria. Durante lo stato di 'EXIT_DISPLAY' le uscite vengono infine assegnate ai valori riportati nei rispettivi *previous*.

CURRENT_STATE, P_STATE

Questi due segnali sono definiti come di tipo state_type. La definizione è congeniale all'utilizzo che se ne fa, infatti essi sono segnali di supporto nel passaggio fra uno stato e l'altro coordinato dalla FSM.

Processo principale

Di seguito è illustrata l'implementazione della FSM presente nel progetto (figura 3).

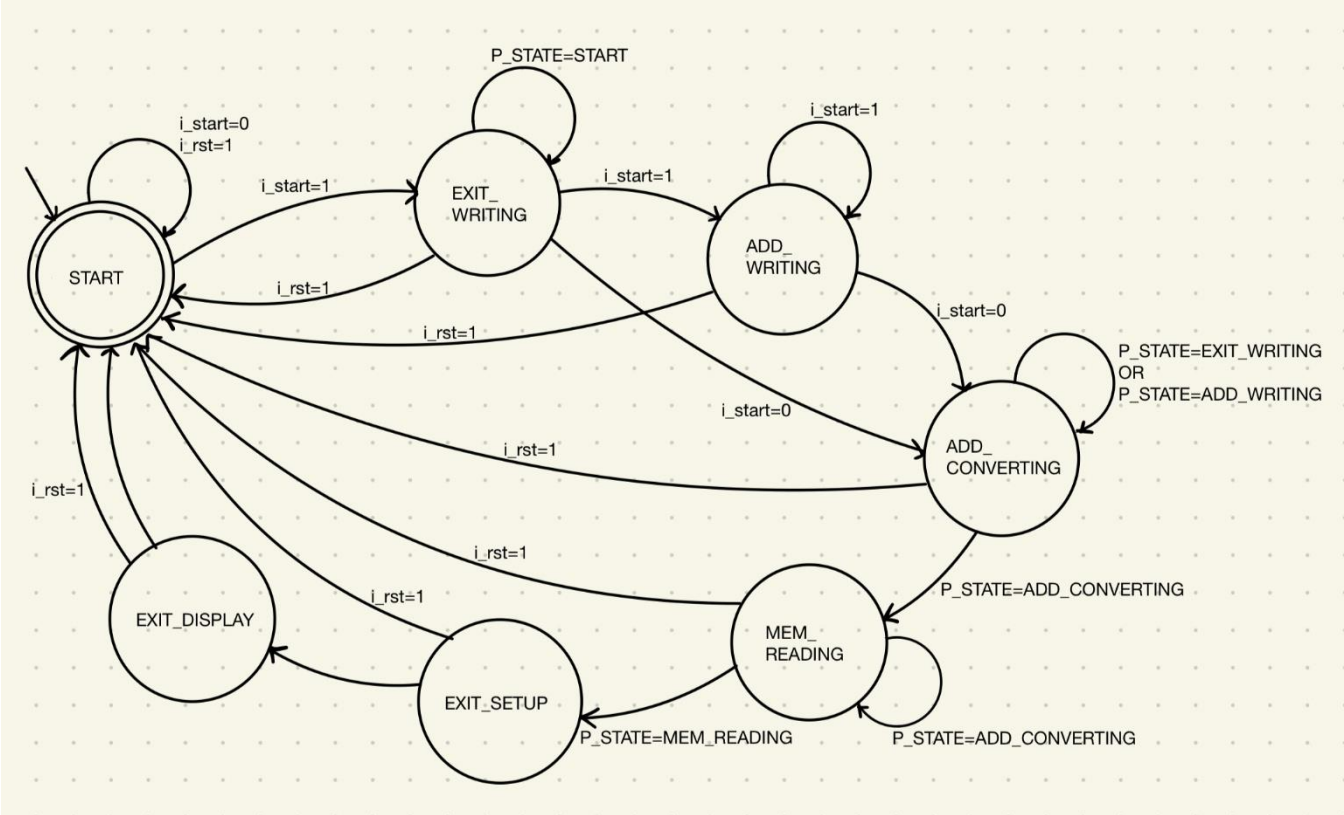

figura 3

Il funzionamento del componente è regolato attraverso un unico processo, che riporta nella sensitivity list i segnali '**CURRENT_STATE**', '**i_start**', '**i_clk**' e '**i_rst**'.

Nella prima parte del processo viene definito il comportamento del componente in presenza del segnale **i_rst=1**. Come già illustrato, questo valore del segnale costituisce un input per l'inizializzazione dei segnali dell'interfaccia e di quelli di supporto al processo. Il segnale '**CURRENT_STATE**' è inizializzato in questa fase a '**START**'. Tutti i segnali che costituiscono un output del componente sono settati a 0 (**previous_z0**, **previous_z1**, **previous_z2**, **previous_z3**, i quattro vettori presenti nell'interfaccia: **o_z0**, **o_z1**, **o_z2**, **o_z3**). Vengono inoltre settati a 0 i segnali usati per gestire la memoria (**o_mem_we**, **o_mem_en**), il vettore relativo all'indirizzo di memoria (**o_mem_addr**), il segnale relativo al primo bit letto (first_bit), il vettore relativo alla porta verso la quale indirizzare il messaggio (**exit_port**), il vettore relativo all' indirizzo di memoria salvato (**saved_address**), il segnale data e il segnale position.

Successivamente viene definito il comportamento richiesto in base allo stato attuale:

CURRENT_STATE è ***START***:

Si procede alle medesime inizializzazioni effettuate in caso di reset, ad eccezione dell' assegnamento a 0 dei segnali previous_z0, z1, z2 e z3, questo poiché, a meno che non venga chiamato il reset, è necessario mantenere traccia dell'ultimo messaggio indirizzato a ciascuna porta.

Infine, si procede ad un controllo sul segnale i_start: se si verifica la condizione i_start = '0' CURRENT_STATE verrà riassegnato a START, altrimenti si procederà alla modifica di first_bit descritta precedentemente e si assegnerà CURRENT_STATE ad 'EXIT_WRITIING' e P_STATE a 'START'.

CURRENT_STATE è ***EXIT_WRITING***:

Viene innanzitutto controllata una condizione su P_STATE: se P_STATE corrisponde a 'START' si procede alle modifiche di exit_port e di position descritte in precedenza per poi impostare i valori aggiornati di CURRENT_STATE e di P_STATE ('EXIT_WRITING' per entrambi). Se invece il valore di P_STATE corrisponde ad 'EXIT_WRITING') effettuo un ulteriore controllo su i_start. Un valore pari a 0 di i_start corrisponderà all'impostazione di CURRENT_STATE come 'ADD_CONVERTING'. Contrariamente, il riscontro di un valore alto di i_start corrisponderà alle modifiche di saved_address e position precedentemente illustrate e si imposterà il CURRENT_STATE ad 'ADD_WRITING'.

Questa condizione su i_start consente di gestire separatamente i casi di acquisizione dei soli bit di uscita e dell'acquisizione di un indirizzo di memoria completo o parziale.

CURRENT_STATE è ***ADD_WRITING***:

Avviene una verifica su i_start (come parzialmente mostrato nella spiegazione del segnale 'saved_address'). Se il segnale i_start ha valore pari a '1' si procede al relativo assegnamento del segnale saved_address e al decremento di position; si prosegue poi con l'assegnamento di P_STATE e di CURRENT_STATE ad ADD_WRITING.

Nel caso in cui il valore di i_start sia uguale a 0 si assegna CURRENT_STATE 'ADD_CONVERTING'.

CURRENT_STATE è ***ADD_CONVERTING***:

In questo processo si procede alla conversione dell'indirizzo come spiegato nella nota relativa a saved_address. Successivamente all'assegnamento viene valutata una condizione relativa a P_STATE: se P_STATE corrisponde a 'EXIT_WRITING' oppure ad 'ADD_WRITING', si imposta 'ADD_CONVERTING' sia come CURRENT_STATE che come P_STATE; se invece P_STATE corrisponde a 'ADD_CONVERTING, viene settato CURRENT_STATE a 'MEM_READING' e si imposta o_mem_en=1.

Quest'ultima verifica ha lo scopo di consentire la corretta computazione dell'indirizzo modificato ed è stata aggiunta in seguito a verifiche sperimentali della sua necessità.

CURRENT_STATE è ***MEM_READING***:

Similmente al caso precedente si effettua una verifica sul segnale P_STATE. In caso il valore del suddetto segnale corrisponda ad 'ADD_CONVERTING', allora si imposta 'MEM_READING' come CURRENT_STATE e 'MEM_READING' viene impostato come P_STATE. Contrariamente se P_STATE corrisponde a MEM_READING si prosegue all'assegnamento di 'data' e si imposta CURRENT_STATE a 'EXIT_SETUP'.

Analogamente al caso precedente, la verifica su P_STATE ha lo scopo di attendere un ciclo di clock, consentendo la corretta lettura del dato proveniente dalla memoria.

CURRENT_STATE è ***EXIT_SETUP***:

Si procede all'impostazione dei segnali previous_z0, z1, z2, z3 come precedentemente mostrato per poi impostare (senza ulteriori verifiche) CURRENT_STATE a 'EXIT_DISPLAY'.

CURRENT_STATE è ***EXIT_DISPLAY***:

I segnali di output che costituiscono le uscite del componente vengono assegnate ai valori previous, il segnale di o_done viene impostato a 1 e CURRENT_STATE viene impostato a 'START'.

In ogni altra condizione ci si riconduce al caso di default in cui CURRENT_STATE è impostato a 'START'.

Risultati sperimentali

Di seguito vengono riportati i risultati della sintesi e osservazioni formulate sulla base di test effettuati sul componente.

Sintesi

Informazioni dettagliate componente RTL:

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
      2 Input     32 Bit       Adders := 1
+---Registers :
                  32 Bit    Registers := 1
                  16 Bit    Registers := 2
                   8 Bit    Registers := 9
                   3 Bit    Registers := 1
                   2 Bit    Registers := 1
                   1 Bit    Registers := 4
+---Muxes :
      7 Input     32 Bit        Muxes := 1
      2 Input     16 Bit        Muxes := 3
      7 Input     16 Bit        Muxes := 3
      7 Input      8 Bit        Muxes := 5
      2 Input      5 Bit        Muxes := 1
      7 Input      3 Bit        Muxes := 1
     13 Input      3 Bit        Muxes := 1
      7 Input      2 Bit        Muxes := 1
      2 Input      1 Bit        Muxes := 16
      4 Input      1 Bit        Muxes := 2
      7 Input      1 Bit        Muxes := 15
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

figura 4

Simulazione

Sono stati testati i seguenti aspetti (documentati dalla relativa immagine facente riferimento alla forma d'onda laddove possibile):

- Effettiva inizializzazione successiva al segnale di **reset** ricevuto (*figura 5*). In questo caso ci si aspetta che tutti i segnali vengano inizializzati, ma soprattutto che alle quattro uscite non compaia più il valore precedentemente mostrato, ma un vettore di otto zeri. Dall'immagine è possibile verificare il corretto reset del segnale previous_z2: successivamente al segnale i_rst, i dati precedentemente presenti in memoria vengono azzerati correttamente;

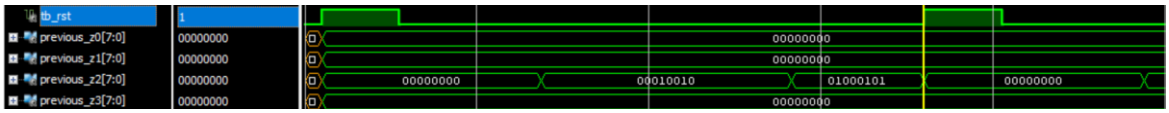

figura 5: segnale di reset ricevuto durante l'esecuzione

- Corretta lettura del messaggio in memoria con display dei messaggi presenti alle quattro uscite: in questo caso è irrilevante riportare le forme d'onda, infatti, il semplice fatto che vengano passati tutti i test sottoposti certifica il corretto funzionamento;
- Sovrascrittura del messaggio mostrato ad una uscita che presentava un messaggio precedente diverso da '0', anche in questo caso così come nel precedente non risulta particolarmente utile riportare la forma d'onda;
- Valutazione di alcuni casi limite, con forma d'onda riportata per maggiore chiarezza:
    - Caso limite con indirizzo di memoria rappresentato da 16 bit '1' (*figura 6*), in questo caso viene completato tutto l'indirizzo di memoria senza necessità di estendere l'indirizzo con '0'; è inoltre visibile il corretto funzionamento di *position* e del suo ruolo per il salvataggio di *saved_address* in *mem_address*;

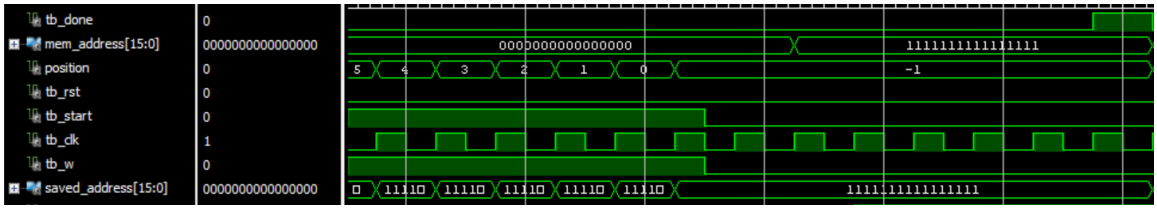

figura 6: caso limite con indirizzo di memoria '1111111111111111'

- Caso limite nel quale l'ingresso seriale *tb_w* resta sempre a zero, input che corrisponde all' uscita *tb_z0* e ad indirizzo di memoria composto da 16 bit a '0'. Come si può osservare dalla forma d'onda il messaggio viene correttamente letto dall'indirizzo di memoria richiesto e viene mostrato all'uscita corretta.

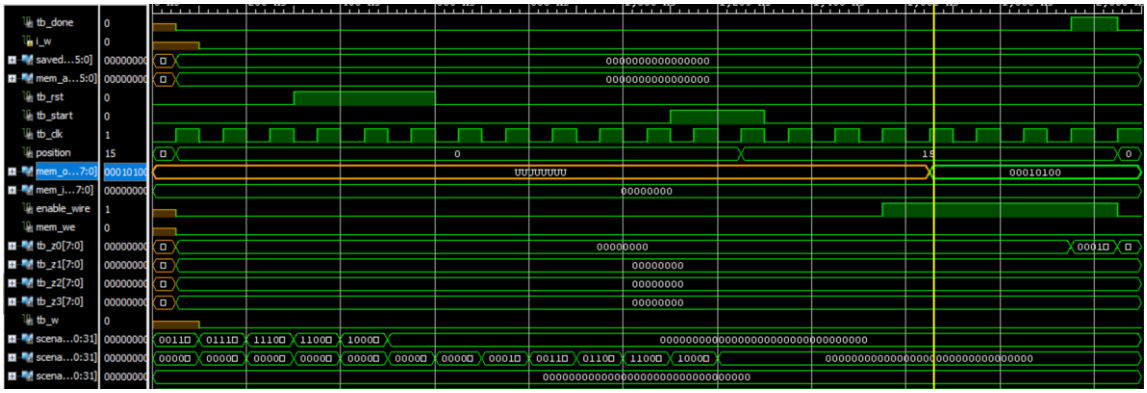

figura 7: segnale seriale in ingresso sempre a '0'

Conclusioni:

Il componente presentato ha dimostrato di essere in grado di superare efficientemente le simulation Behavioural, Post-Synthesis Functional e Post-Synthesis Timing. La sintesi del componente ha prodotto il seguente schematico (*figura 8*):

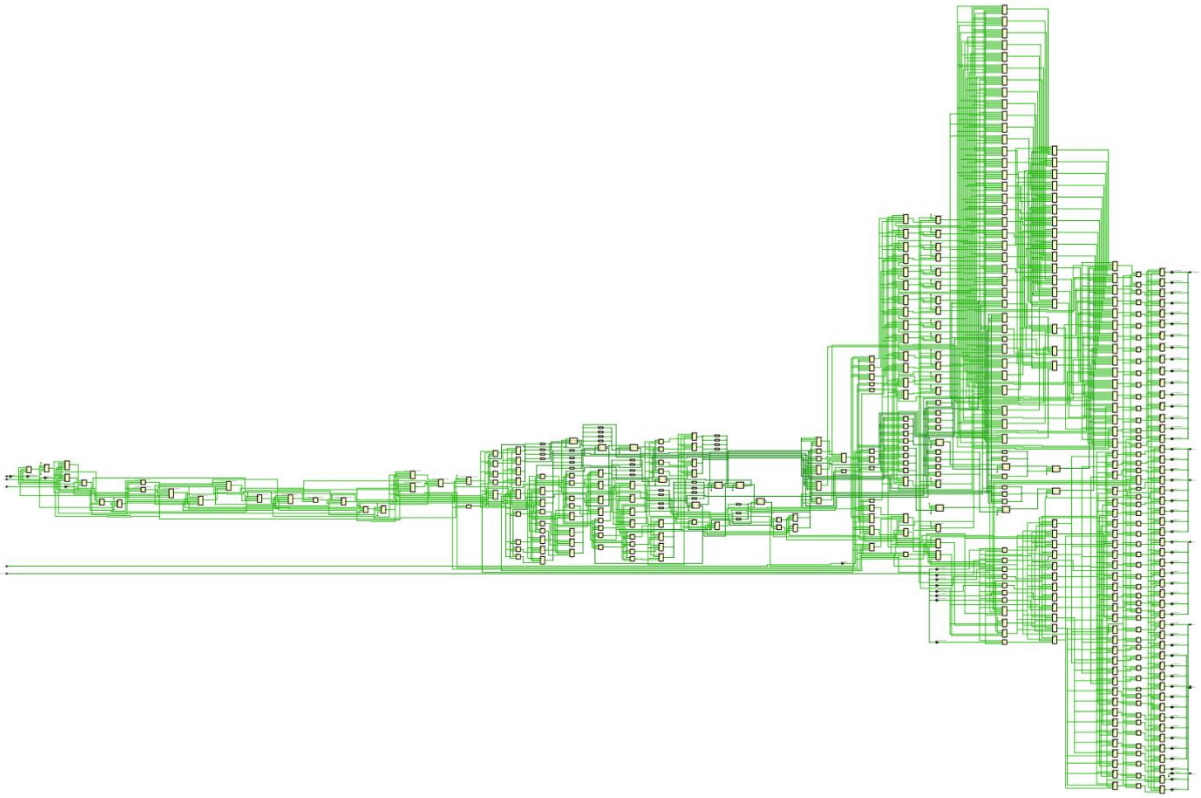

figura 8: Schematico del componente

Ottimizzazioni

Il progetto presentato è la risultante di una serie di ottimizzazioni. Una precedente soluzione prevedeva un processo per ogni stato, contestualmente ad un'analoga suddivisione fra stati e operazioni, e un processo interamente dedicato alla regolazione dello stato attuale della FSM. Questa soluzione è stata abbandonata in favore di questa in quanto troppo pesante, poiché frutto di una scarsa conoscenza dello strumento di programmazione e del linguaggio utilizzato. Grazie ad un progressivo miglioramento delle competenze sopra indicate è stato possibile sviluppare la soluzione qui presentata, che è risultata molto più performante.