POLITECNICO
MILANO 1863

Prova Finale - Progetto di Reti Logiche

Prof. Gianluca Palermo - Anno 2022/2023
Simone Ponginibbio - 10699608 / 933724

Indice

1 Introduzione

1.1 Specifica del progetto

La specifica del progetto richiede la descrizione in linguaggio VHDL di un componente hardware che, ricevendo informazioni sulla locazione di memoria, invi al corretto canale di uscita i bit della parola contenuta in memoria. I possibili canali di uscita sono quattro e vengono selezionati in base ai primi due bit della sequenza letta quando il segnale START è alto. Il componente dovrà inoltre funzionare con un periodo di clock di almeno 100 ns.

1.2 Funzionamento della specifica

Il modulo implementato è dotato di due ingressi primari da 1 bit ciascuno (W e START) e 5 uscite primarie. Le uscite comprendono quattro canali da 8 bit ciascuno (Z0, Z1, Z2, Z3) e un'uscita da 1 bit (DONE). Il modulo riceve anche un segnale di clock CLK, che è condiviso da tutto il sistema, e un segnale di reset RESET, anch'esso unico.

Il sistema descritto è un modulo di gestione di dati sequenziali basato su segnali di controllo e consente la lettura e l'indirizzamento di dati da una memoria verso quattro canali di uscita (Z0, Z1, Z2, Z3).

Nella fase iniziale, quando il segnale di RESET è alto, il modulo è riportato allo stato iniziale. Le uscite Z0, Z1, Z2 e Z3 sono tutte a 0 e rimangono tali fino a quando il segnale DONE è basso.

Quando il segnale START diventa alto, il modulo inizia a leggere i dati in ingresso sul pin W sul fronte di salita del clock. I primi 2 bit rappresentano l'intestazione del canale di uscita (Z0, Z1, Z2 o Z3), mentre i successivi N bit rappresentano l'indirizzo di memoria.

Gli N bit di indirizzo possono variare da 0 a un massimo di 16 bit. Se N è inferiore a 16, gli indirizzi vengono estesi con zeri sui bit più significativi. Ad esempio, se N=9, l'indirizzo 101011101 viene esteso a 0000000101011101. Il segnale START rimarrà alto per almeno 2 cicli di clock e non più di 18 cicli di clock.

Una volta letti i dati dalla memoria e indirizzati al canale appropriato, il segnale DONE diventa alto per un ciclo di clock. Durante questo periodo, il valore del canale associato al messaggio cambierà, mentre gli altri canali manterranno il loro ultimo valore, inoltre è garantito che il segnale START rimarrà a 0.

Quando il segnale DONE è basso, invece, tutti i canali Z0, Z1, Z2 e Z3 devono essere a 0.

Prima della prima operazione con START=1, il modulo verrà sempre riportato allo stato iniziale quando il segnale di RESET è alto. Dopo la prima operazione di RESET, successive operazioni START=1 non richiederanno un nuovo reset del modulo.

Il modulo è stato progettato in modo tale da garantire che il tempo massimo per produrre il risultato, ovvero il tempo trascorso tra START=0 e DONE=1, sia inferiore a 20 cicli di clock.

1.3 Interfaccia del componente

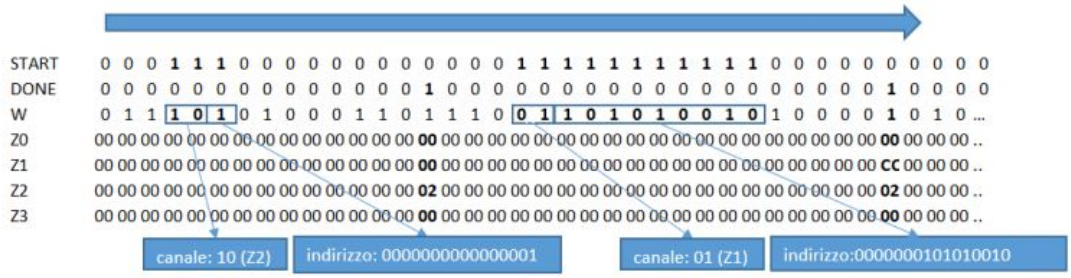
Il componente descritto ha la seguente interfaccia:

```
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;
);
end project_reti_logiche;
```

In particolare:

- `i_clk`: è il segnale di clock in ingresso;
- `i_rst`: è il segnale che inizializza e fa il reset della macchina;
- `i_start`: è il segnale che avvia le operazioni di lettura;
- `i_w`: è il segnale che contiene le informazioni su canale di uscita e locazione di memoria del dato;
- `o_z0`: è il segnale di uscita rappresentato da "00" nei primi due bit del segnale `i_w`;
- `o_z1`: è il segnale di uscita rappresentato da "01" nei primi due bit del segnale `i_w`;
- `o_z2`: è il segnale di uscita rappresentato da "10" nei primi due bit del segnale `i_w`;
- `o_z3`: è il segnale di uscita rappresentato da "11" nei primi due bit del segnale `i_w`;
- `o_done`: è il segnale di uscita che comunica la fine delle operazioni di scrittura dei dati elaborati sui canali di uscita;
- `o_mem_addr`: è il segnale di uscita che invia l'indirizzo alla memoria;
- `i_mem_data`: è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_mem_we`: è il segnale da dover inviare alla memoria per permettere le operazioni di lettura (=0) e di scrittura (=1);
- `o_mem_en`: è il segnale da dover inviare alla memoria per poter avviare la comunicazione, sia in lettura che in scrittura.

1.4 Esempio di esecuzione



L'esecuzione inizia con uno stato di reset obbligatorio (non mostrato in figura) per inizializzare la macchina e renderla pronta a ricevere il primo segnale di START.

Nel 4° ciclo di clock il segnale di START è alto quindi si inizia a leggere il segnale `i_w` per capire su quale canale di uscita bsogna operare. Durante il succesivo ciclo di clock si continua a leggere il segnale `i_w` e si costruisce la coppia di bit che caratterizza un canale di uscita. In questo caso, essendo i bit 1 e 0, il canale di uscita selezionato sarà `o_z2`.

Nel 6° ciclo di clock si legge nel segnale `i_w` il bit che rappresenta l'indirizzo di memoria in cui è contenuto il messaggio che verrà scritto nel canale di uscita `o_z2`. Visto che il seganle di START è basso la lettura del segnale `i_w` si ferma e viene recuperato il messaggio presente nell'indirizzo 0000000000000001, ricavato precedentemente, che in questo caso è 02.

Nel 15° ciclo di clock viene alzato il segnale DONE e scritto il valore del messaggio appena trovato sul rispettivo canale di uscita che verrà memorizzato anche nelle seccessive esecuzioni fino a quando non verrà sovrascritto o avverà un reset. In questo caso il messaggio 02 verrà scritto sul canale di uscita `o_z2`, mentre gli altri canali resteranno a 0.

Nel successivo esempio il segnale START si alza e vengono letti dal segnale `i_w` i primi due bit che sono in questo caso 0 e 1 che rappresentano il canale di uscita `o_z1`.

Nei 9 cicli di clock successivi vengono letti i bit che rappresentano l'indirizzo di memoria interessato ovvero 101010010, trasformato a 0000000101010010 per farlo diventare di 16 bit.

Il seganle START viene portato a 0 e si cerca il messaggio in memoria all'indirizzo appena trovato che in questo caso è CC.

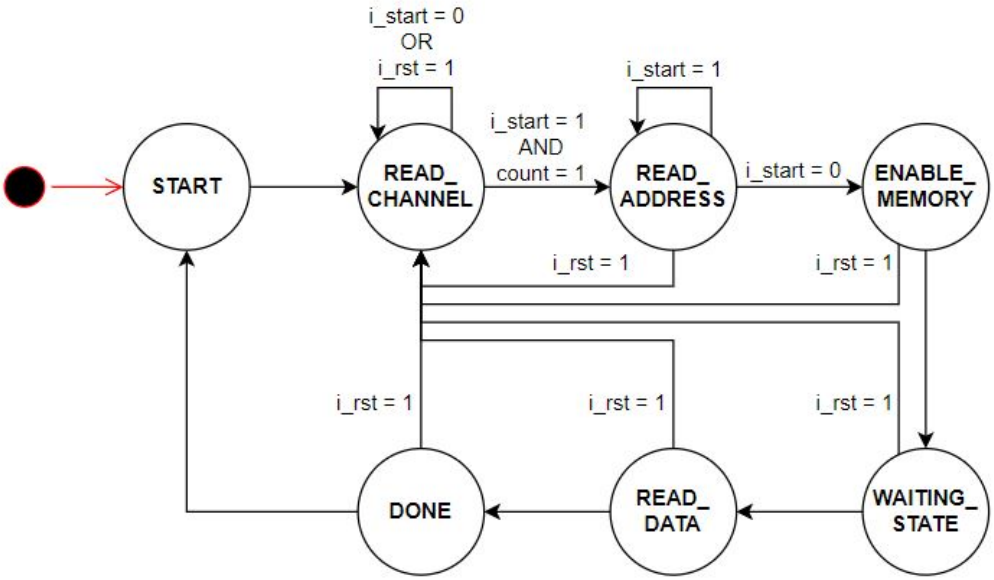
Poi viene alzato il seganle DONE e scritto il messaggio CC sul canale di uscita `o_z1` oltre a mantenere il messaggio 02 sul canale di uscita `o_z2`, mentre gli altri rimarranno a 0.

I valori presenti sui 4 canali sono all'inizio 0 e rimangono tali mentre il segnale DONE è basso. Quando il sagnale diventa alto vengono mostrati gli ultimi valori memorizzati per ogni canale.

2 Architettura

Il componente è organizzato tramite una macchina a stati che scandisce gli accessi alla memoria e le fasi di elaborazione. All'avvio e dopo ogni reset, la macchina rimane in attesa che il segnale `i_start` venga alzato per poter iniziare la lettura della sequenza di bit presenti nel segnale `i_w`. Si prendono i primi 2 bit per scegliere il corretto canale di uscita tra i 4 possibili e i successivi bit per determinare l'indirizzo di memoria su cui operare per poi scrivere sul apposito canale il messaggio in memoria.

2.1 Macchina a stati



2.2 Stati della macchina

La macchina a stati sintetizzata è composta da i seguenti 7 stati:

- **START**: È lo stato da cui la macchina parte per effettuare l'elaborazione. Vengono resettati i registri count, channel e address, oltre alle uscite `o_z0`, `o_z1`, `o_z2`, `o_z3`, `o_done`, `o_mem_we` e `o_mem_en`. Dopo aver settato i valori lo stato della macchina diventa `READ_CHANNEL`;
- **READ_CHANNEL**: È lo stato in cui si aspetta che il segnale `i_start` sia alto per procedere con la lettura dei 2 bit nel seganle `i_w` che rappresentano il canale di uscita. In questo stato viene fatto uso del registro count per tenere traccia del numero di bit letti fino ad ora e del conseguente passaggio di stato a `READ_ADDRESS` qualora ve ne siano letti 2;
- **READ_ADDRESS**: È lo stato in cui si controlla ad ogni ciclo di clock che il segnale `i_start` sia alto per poter costruire l'indirizzo di memoria che contiene il messaggio da srivere sul canale di uscita apposito. Il registro address viene riempito da un bit del canale `i_w` ogni ciclo di clock. Se in qualsiasi momento il segnale `i_start` dovesse diventare basso il nuovo stato della macchina diventa `ENABLE_MEMORY` e l'indirizzo esteso a 16 bit se necessario;
- **ENABLE_MEMORY**: È lo stato in cui l'indirizzo calcolato nello stato precedente viene scritto nell'uscita `o_mem_addr` e inoltre viene alzato il segnale `o_mem_en` per permettere la comunicazione con la memoria;
- **WAITING_STATE**: È lo stato di attesa della macchina che permette di far leggere la memoria al prossimo stato della macchina;
- **READ_DATA**: È lo stato in cui viene letto il contenuto nell'indirizzo di memoria calcolato precedentemente tramite il segnale di ingresso `i_mem_data` e in base al valore del registro channel viene modificato un registro tra z0, z1, z2 e z3;
- **DONE**: È lo stato in cui la macchina si trova quando ha terminato l'elaborazione. Vengono scritti i valori dei 4 registri z0, z1, z2 e z3 nei corrispettivi canali di uscita e viene alzato il segnale `o_done`. Alla fine di queste operazioni lo stato della macchina diventa `START`.

Quando il segnale `i_rst` è alto, ovunque si trovi la macchina, il nuovo stato diventa `READ_CHANNEL` e tutti i valori dei canali di uscita vengono portati a 0.

2.3 Registri della macchina

La macchina possiede 8 registri che mantengono informazioni necessarie per il corretto svolgimento dell'elaborazione:

In particolare:

- **state**: indica in quale stato si trova la macchina tra i seguenti: `START`, `READ_CHANNEL`, `READ_ADDRESS`, `ENABLE_MEMORY`, `WAITING_STATE`, `READ_DATA` e `DONE`;
- **count**: tiene tracca del numero di bit letti nel segnale `i_w`. Quando vengono letti 2 bit viene cambiato lo stato a `READ_ADDRESS` per permettere di continuare a leggere ulteriori bit utili per la creazione dell'indirizzo di memoria che conterrà il messaggio;
- **channel**: contiene i 2 bit che rappresentano uno dei 4 canali di uscita sul quale verrà scritto il messaggio;
- **address**: contiene i bit dell'indirizzo di memoria ricavati durante l'esecuzione estesi a 16 bit se necessario;
- **z0**: contiene il messaggio da scrivere sul canale di uscita `o_z0` nel momento in cui il segnale `o_done` viene alzato. Viene sovrascritto ogni volta che si elabora un nuovo messaggio da inoltrare sul canale `o_z0`;
- **z1**: contiene il messaggio da scrivere sul canale di uscita `o_z1` nel momento in cui il segnale `o_done` viene alzato. Viene sovrascritto ogni volta che si elabora un nuovo messaggio da inoltrare sul canale `o_z1`;
- **z2**: contiene il messaggio da scrivere sul canale di uscita `o_z2` nel momento in cui il segnale `o_done` viene alzato. Viene sovrascritto ogni volta che si elabora un nuovo messaggio da inoltrare sul canale `o_z2`;
- **z3**: contiene il messaggio da scrivere sul canale di uscita `o_z3` nel momento in cui il segnale `o_done` viene alzato. Viene sovrascritto ogni volta che si elabora un nuovo messaggio da inoltrare sul canale `o_z3`.

All'avvio e dopo ogni reset tutti i registri hanno valore 0.

3 Risultati sperimentali

3.1 Sintesi

Il componente progettato è perfettamente sintetizzabile e implementabile come mostrato in figura:

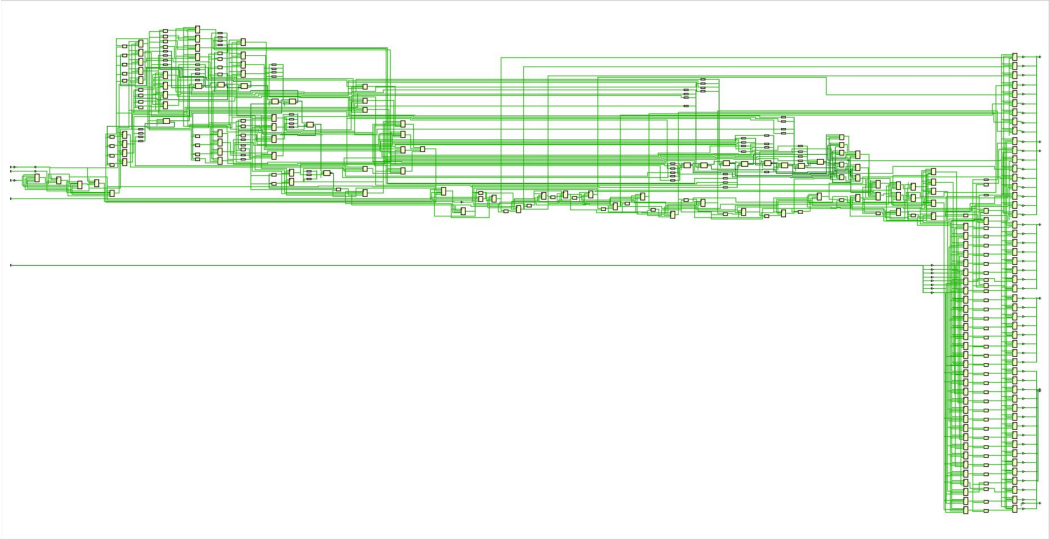
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIe %
synth_1	constrs_1	synth_design Complete!								178	135	0	0	0.000
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	4.548	0	147	135	0	0	0.000

Ed occupa le seguenti risorse:

Resource	Estimation	Available	Utilization %
LUT	178	134600	0.13
FF	135	269200	0.05
IO	63	285	22.11
BUFG	1	32	3.13

3.2 Rappresentazione a blocchi del componente

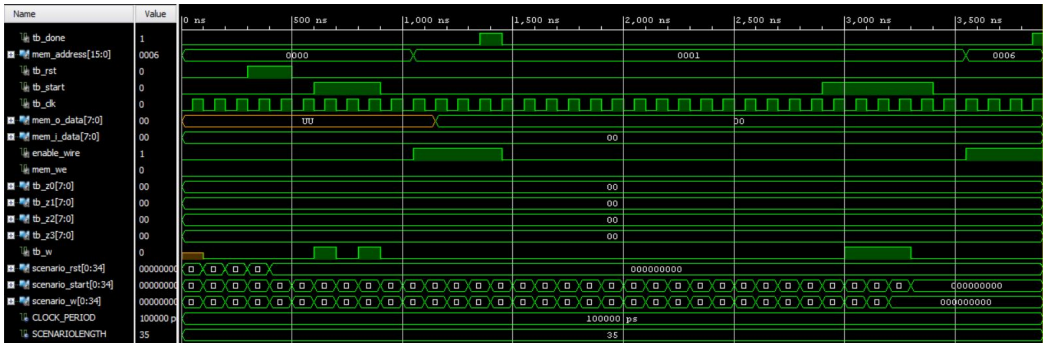
Il modulo sintetizzato può essere anche descritto dal seguente diagramma a blocchi:



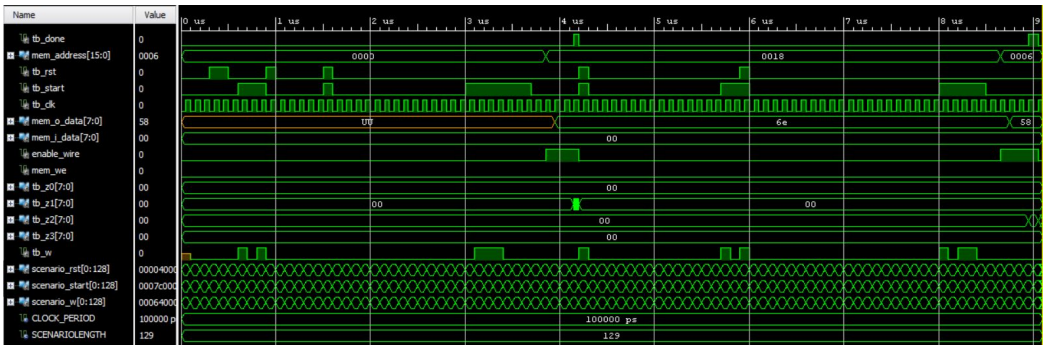
3.3 Simulazioni

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il testbench di esempio, sono stati definiti altri casi di test in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione. Di seguito è fornita una breve descrizione di alcuni dei test più significativi utilizzati:

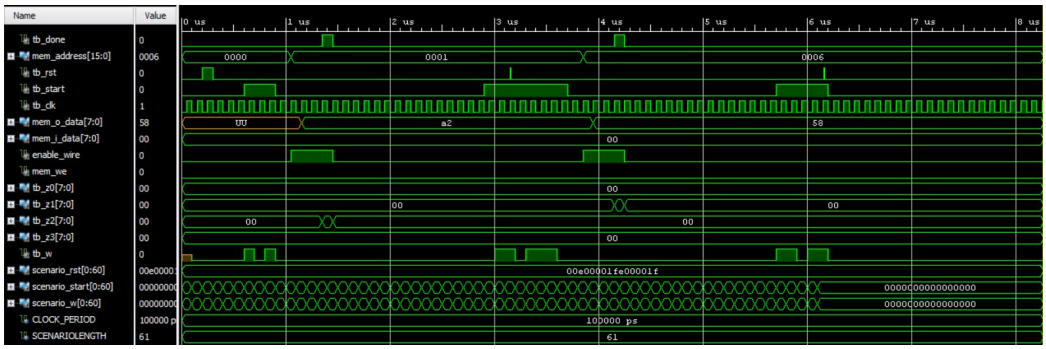
- **Sequenze di 8 zeri in input**: il test verifica il corretto funzionamento del componente al ricevimento di 2 sequenze di 8 zeri in input. Ci si aspetta che i 4 registri e i 4 canali di uscita rimangano a zero per tutta la durata dell'esecuzione:



- **Reset multipli**: il test verifica il corretto funzionamento del componenete in caso di ricevimento di molteplici segnali di reset anche quando il segnale `i_start` è alto:



- **Reset asincroni**: il test verifica il corretto funzionamento del componente in caso di ricevimento di segnali di reset con tempo di salita e discesa inferiore ad un ciclo di clock mentre il segnale `i_start` è alto:



- **Simulazione sotto sforzo**: il test verifica il corretto funzionamento del componente in caso di numerose operazioni di lettura di memoria e scrittura sui canali di uscita in sequenza:

4 Conclusioni

Lo sviluppo del progetto ha seguito delle fasi ben definite. Inizialmente, mi sono immerso nello studio del linguaggio VHDL attraverso lezioni fornite dai docenti, arricchite da risorse online come video e articoli. Successivamente, ho applicato le conoscenze acquisite su Vivado, affrontando la sintesi di esempi pratici presentati durante le lezioni e consolidando la comprensione del software, inclusa la simulazione attraverso test bench e l'analisi delle forme d'onda.

Una volta acquisita familiarità con gli strumenti, mi sono concentrato sulla specifica del progetto, approfondendo il funzionamento del selettore di canali di uscita. La chiarezza della documentazione, supportata da grafici autoesplicativi, ha agevolato la comprensione di questa componente. Successivamente, ho approfondito lo studio dell'accesso alla memoria, trovando risposte alle mie domande nella descrizione in VHDL.

Grazie a queste informazioni dettagliate, ho progettato la macchina a stati, identificando opportunità per ottimizzare i cicli di clock necessari per elaborare il flusso in uscita. Grazie a questa ottimizzazione, sono riuscito a conseguire una significativa riduzione del tempo complessivo di esecuzione del progetto.