Progetto Reti Logiche

Giulio Montuori - 955248

3 ottobre 2023

Indice

1 Introduzione

Nel campo dell'elaborazione dei segnali e delle comunicazioni digitali, il presente lavoro si concentra sulla progettazione e realizzazione di un modulo in linguaggio VHDL per gestire l'elaborazione, il routing e la trasmissione dei messaggi all'interno di un sistema di comunicazione. Il sistema è chiamato a ricevere sequenze di dati in ingresso, interpretare le informazioni riguardanti il canale di uscita e l'indirizzo di memoria che contiene il messaggio e, successivamente, instradare il messaggio al canale designato.

Per fornire un quadro chiaro del funzionamento del sistema, si presenta un esempio ipotetico in cui il modulo viene impiegato nella gestione della trasmissione di messaggi tra utenti di una rete di comunicazione. I messaggi sono codificati come sequenze di dati composte da una coppia di bit d'intestazione, seguiti da N bit relativi all'indirizzo di memoria. L'interpretazione dell'intestazione consente al modulo di identificare il canale di uscita, mentre l'indirizzo di memoria fornisce le informazioni sulla posizione del messaggio da trasmettere. Il modulo deve quindi elaborare tali sequenze di dati, decifrare il canale di uscita corretto e instradare il messaggio di conseguenza.

Questa relazione si propone di approfondire l'architettura complessiva del sistema, la logica di implementazione e i risultati ottenuti attraverso la sintesi e le simulazioni sperimentali. L'analisi si concentrerà sull'indagine delle funzionalità del modulo VHDL, in particolare sulla gestione dell'elaborazione e la trasmissione dei messaggi, e sulla valutazione delle soluzioni proposte in relazione alle specifiche di progetto richieste.

2 Architettura

Il modulo VHDL proposto si basa su una macchina a stati finiti (FSM) di Mealy controllata dai segnali di `clock`, `start` e `reset`. La FSM gestisce l'elaborazione e il routing dei messaggi nel sistema di comunicazione. Di seguito, viene presentata l'architettura funzionale del sistema con un immagine della FSM (Figura 1), una tabella (Figura 3) che rappresenta l'Output Logic con i valori dei vettori al variare dello stato della FSM e una tabella (Figura 2) che rappresenta la next state logic e, infine, una descrizione dettagliata di ogni stato.

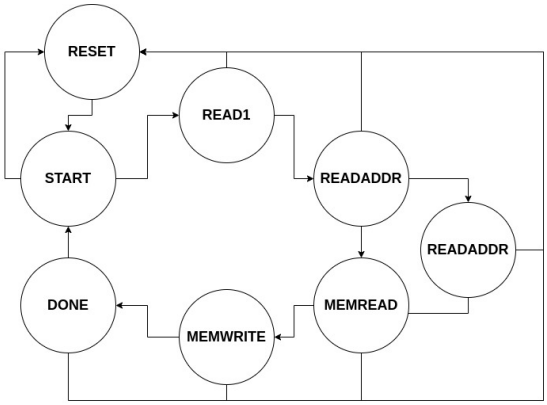

Figura 1: Rappresentazione della FSM

	start = 1	start = 0	rst = 1	rst = 0
RESET	START	START	RESET	-
START	READ1	START	RESET	-
READ1	READ0	-	RESET	-
READ0	READADDR	MEMWORK	RESET	-
READADDR	READADDR	MEMWORK	RESET	-
MEMREAD	MEMWRITE	MEMWRITE	RESET	-
MEMWRITE	DONE	DONE	RESET	-
DONE	START	START	RESET	-

Figura 2: Next State Logic

	RESET	START	READ1	READ0	READADDR	MEMREAD	MEMWRITE	DONE
o_z0[8]	0	0	-	-	-	-	-	uscite(0)
o_z1[8]	0	0	-	-	-	-	-	uscite(1)
o_z2[8]	0	0	-	-	-	-	-	uscite(2)
o_z3[8]	0	0	-	-	-	-	-	uscite(3)
o_done	0	0	-	-	-	-	-	1
o_mem_addr[16]	0	0	-	-	-	mem_addr	mem_addr	-
o_mem_we	0	0	-	-	-	0	-	-
o_mem_en	0	0	-	-	-	1	0	-
target(1)	0	0	i_w	-	-	-	-	-
target(0)	0	0	-	i_w	-	-	-	-
mem_addr[16]	0	0	-	-	---	-	-	-
i_mem_data	ingresso							
uscite(0)	0	-	-	-	-	-	-	una
uscite(1)	0	-	-	-	-	-	-	delle
uscite(2)	0	-	-	-	-	-	-	quattro
uscite(3)	0	-	-	-	-	-	-	cambia

Figura 3: Output Logic

2.1 Stati della FSM

La FSM comprende i seguenti stati:

- `RESET`: Durante questo stato, sia le uscite (`o_z0`, `o_z1`, `o_z2` e `o_z3`) sia il vettore `uscite` che svolge il ruolo di buffer vengono azzerati, il segnale `o_done` (DONE) viene impostato a `'0'` e i comandi di memoria vengono resettati. La macchina a stati si prepara a ricevere una nuova sequenza di ingresso.

- `START`: La FSM entra in questo stato quando il segnale `i_start` diventa alto (`='1'`). Anche in questa fase, le uscite e i comandi di memoria vengono resettati, la FSM si prepara a leggere la sequenza di ingresso che rappresenta l'intestazione del canale di uscita e l'indirizzo di memoria.

- `READ1`: In questo stato, la FSM legge il primo bit dell'intestazione del canale di uscita. Questo bit viene memorizzato nel bit più significativo del registro temporaneo `target`. Questo registro temporaneo viene utilizzato per identificare il canale di uscita a cui il messaggio deve essere instradato. Durante questo stato, tutte le altre uscite e comandi di memoria vengono mantenuti nei loro stati precedenti

- `READ0`: La FSM legge il secondo bit di intestazione del canale di uscita e lo memorizza nel bit meno significativo nel registro temporaneo `target` usato già in precedenza dallo stato `READ1`. A questo punto, la macchina a stati ha identificato il canale di uscita.

- `READADDR`: La FSM legge singolarmente e serialmente `N` bit dell'indirizzo di memoria e li memorizza in un registro temporaneo `mem_addr` preoccupandosi anche di fare uno shift register per permettere di memorizzare il bit in entrata sempre nella posizione meno significativa. Questo stato si ripete per `N` cicli di clock, corrispondenti al numero di bit dell'indirizzo di memoria, decisi dal segnale `i_start = '1'`

- `MEMREAD`: La FSM imposta l'indirizzo di memoria `o_mem_addr` uguale a `mem_addr`, abilita la memoria `o_mem_en = '1'` e disabilita la scrittura in memoria `o_mem_we = '0'`.
- `MEMWRITE`: La FSM scrive il messaggio letto dalla memoria `i_mem_data` nel vettore `uscite` in posizione `target` e disabilita la memoria `o_mem_en = '0'`.
- `DONE`: In questa fase, il segnale `o_done` viene impostato a `'1'`, il messaggio viene assegnato all'uscita corrispondente e vengono mostrati in uscita i messaggi precedentemente assegnati alle altre uscite. La FSM è pronta a tornare allo stato START per ricevere una nuova sequenza di ingresso, a condizione che il segnale `i_start` sia alto (`='1'`).

2.2 Processi

Il codice VHDL presenta tre processi principali:

- Processo `OL` (Output Logic): Questo processo è responsabile della gestione delle uscite (`o_z0`, `o_z1`, `o_z2`, `o_z3`) e dei comandi di memoria (`o_mem_addr`, `o_mem_en`, `o_mem_we`) in base allo stato corrente della FSM. Il processo OL utilizza uno switch-case per impostare le uscite e i comandi di memoria a seconda dello stato attuale. Ad esempio, durante lo stato `DONE`, il messaggio letto dalla memoria viene assegnato all'uscita corrispondente e il segnale `o_done` viene impostato a `= '1'`.
- Processo `TL` (Transition Logic): Questo processo determina il prossimo stato della FSM in base allo stato corrente e ai segnali di controllo. Anche in questo caso, viene utilizzato uno switch-case per decidere il prossimo stato a seconda dello stato attuale. Ad esempio, se la FSM si trova nello stato START e il segnale `i_start` diventa alto (`=1`), il prossimo stato sarà `READ1`.
- Processo `SL` (Synchronous Logic): Questo processo è responsabile dell'aggiornamento dello stato della FSM e delle operazioni di lettura e scrittura dei registri temporanei. Il suo effetto avviene ad ogni fronte di salita del segnale `i_clk` e finisce impostando a `RESET` lo stato corrente se il segnale `i_rst` è alto (`= '1'`). In seguito, un switch-case viene utilizzato per eseguire azioni specifiche a seconda del prossimo stato. Ad esempio, durante lo stato `READ1`, il primo bit di intestazione del canale di uscita viene letto e memorizzato nel registro temporaneo `target(1)`. Infine, lo stato corrente viene aggiornato con il valore di `next_state`.

Riassumendo, il codice VHDL utilizza tre processi principali (`OL`, `TL` e `SL`) per gestire le uscite, i comandi di memoria e gli aggiornamenti di stato. La descrizione dettagliata di ogni stato e processo fornisce una comprensione approfondita del funzionamento del sistema e delle scelte implementative adottate.

2.3 Esempio di Funzionamento

Per illustrare il funzionamento del modulo VHDL, consideriamo un esempio specifico. Supponiamo che il modulo riceva la seguente sequenza di dati di ingresso:

101101. In questa sequenza, 10 rappresenta l'intestazione che identifica il canale di uscita, mentre 1101 è l'indirizzo di memoria che contiene il messaggio da trasmettere.

Quando la FSM riceve questa sequenza di dati, entra nello stato START. Durante questo stato, tutte le uscite e i comandi di memoria vengono resettati, preparando la FSM a leggere la sequenza di ingresso. Successivamente, la FSM entra nello stato READ1. In questo stato, legge il primo bit dell'intestazione del canale di uscita ('1') e lo memorizza nel bit più significativo del registro temporaneo target. La FSM procede poi allo stato READ0. Qui, legge il secondo bit dell'intestazione del canale di uscita ('0') e lo memorizza nel bit meno significativo del registro temporaneo target. La FSM ha identificato il canale di uscita come 10 che identifica o_z3.

Successivamente, la FSM entra nello stato READADDR. In questa fase, legge serialmente gli N bit dell'indirizzo di memoria (1101) e li memorizza nel registro temporaneo mem_addr dove alla fine della il suo valore sarà '0000000000101101'. Una volta letto l'indirizzo di memoria, la FSM entra nello stato MEMREAD. Durante questo stato, imposta l'indirizzo di memoria (o_mem_addr) uguale a mem_addr, abilita la memoria (o_mem_en = '1') e disabilita la scrittura in memoria (o_mem_we = '0'). Infine, la FSM entra nello stato MEMWRITE. In questa fase, scrive il messaggio letto dalla memoria (i_mem_data) nel vettore uscite nella posizione identificata dall'intestazione e disabilita la memoria (o_mem_en = '0'). Quando il messaggio è stato scritto con successo, la FSM entra nello stato DONE. Durante questo stato, il segnale o_done viene impostato a '1' e vengono mostrati i segnali nel vettore uscite nelle uscite corrispondenti. La FSM è ora pronta a tornare allo stato START per ricevere una nuova sequenza di ingresso.

3 Risultati sperimentali

La fase sperimentale del progetto ha comportato la sintesi e la simulazione del modulo VHDL. Questa fase ha permesso di verificare la correttezza funzionale del modulo e di osservare il suo comportamento in vari scenari. In particolare, sono stati condotti due esperimenti chiave poiché testano 2 casi limite.

Il primo esperimento ha riguardato un caso in cui l'indirizzo di memoria non era specificato. Questo scenario ha permesso di verificare come il modulo gestisce situazioni in cui le informazioni necessarie per la trasmissione del messaggio non sono esplicitamente disponibili. Il secondo esperimento ha riguardato un caso in cui è stato utilizzato un RESET asincrono. Questo scenario ha permesso di osservare come il modulo risponde a un RESET improvviso e come ripristina le sue uscite e i suoi comandi di memoria. Nei paragrafi successivi, verranno presentati in dettaglio i risultati di questi due esperimenti.

3.1 Indirizzo di memoria non specificato

Questo test monitora il comportamento della FSM quando l'indirizzo di memoria non è esplicitamente dato. Come si può vedere dalla Figura 4, in assenza di input (successivo alla selezione del canale) il programma gestisce questo caso memorizzando nel registro temporaneo mem_addr una parola composta da tutti 0. Questo mostra un aspetto fondamentale dell'implementazione della specifica

(Figura 5), infatti già negli stati di `RESET` e `START` `mem_addr` viene impostato ad una parola composta da tutti `0` e alla lettura dell'ingresso modifica i bit necessari fornendo automaticamente il padding.

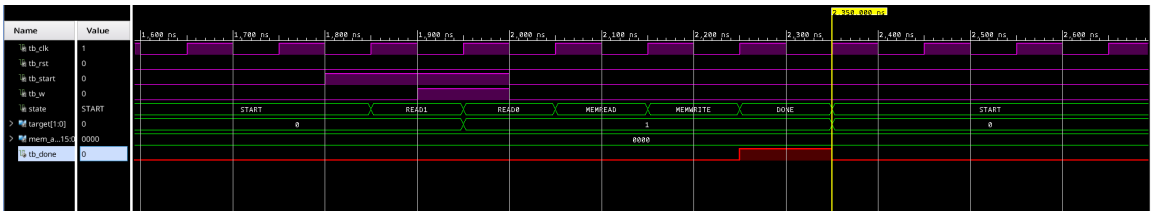

Figura 4: WaveForm della simulazione con indirizzo non specificato

```
122  SL : process(i_clk, i_rst)
...     ...
126     when RESET      =>
...       ...
129       mem_addr  <= (Others => '0');
130
131     when START      =>
...     ...
133     mem_addr    <= (Others => '0');
```

Figura 5: Codice che gestisce il cambiamento di stato

3.2 RESET asincrono

Questo test monitora il comportamento della FSM quando il `i_rst` viene impostato a `1` in maniera asincrona. Come si può vedere dalla Figura 6 i segnali vengono resettati sul fronte di salita del clock successivo al `i_rst = '1'` poiché lo stato corrente della FSM diventa, appunto, quello di `RESET` come evidenziato dall'estratto di codice mostrato in Figura 7.

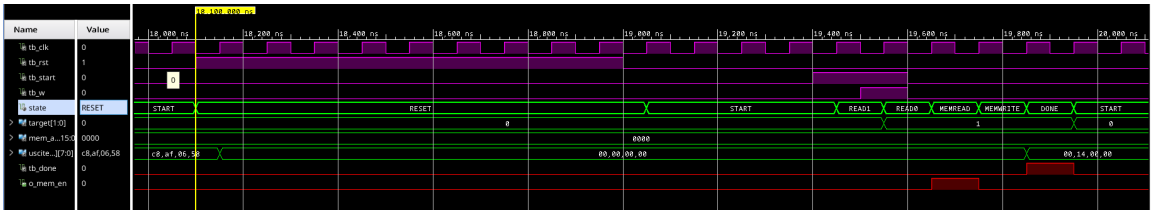

Figura 6: WaveForm della simulazione con un reset asincrono

```
122  SL : process(i_clk, i_rst)
123  begin
124    if rising_edge(i_clk) then
...    ...
153    end if;
154    if i_rst = '1' then
155      state <= RESET;
156    end if;
157  end process SL;
```

Figura 7: Codice dell'inizializzazione del vettore

4 Conclusioni

L'implementazione del modulo ha dimostrato di essere efficace nel gestire la trasmissione di messaggi nei canali di memoria disponibile. L'analisi delle funzionalità del modulo ha confermato che le soluzioni proposte rispettano le specifiche del progetto.

Inoltre, il progetto ha offerto l'opportunità di imparare a descrivere hardware in VHDL e di acquisire una maggiore familiarità con le tecniche di progettazione hardware e del software Vivado. La sfida di progettare un modulo che rispettasse le specifiche del progetto ha permesso di sviluppare competenze pratiche nel campo della progettazione di reti logiche.

Un esempio concreto di questo consiste in un errore nel codice durante lo sviluppo. Il modulo VHDL funzionava e superava i test in Behavioral ma non in Post-Synthesis, questo a causa del funzionamento specifico dei segnali all'interno di un `process` VHDL: i segnali vengono effettivamente aggiornati solo alla fine dell'esecuzione del process. Nella prima versione del `SL process` il controllo sul segnale `i_rst = '1'` era in cima al process ed assegnava al segnale `state <= RESET`; tuttavia se `rising_edge(i_clk) = '1'` l'operatore `switch case` controllava il segnale non ancora aggiornato ed infine sovrascriveva `state <= next_state`, vanificando l'effetto del reset. La seconda versione del `SL process` effettua il controllo sul segnale `i_rst` alla fine del process come mostrato in Figura 7.