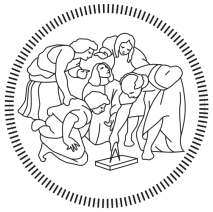
Annechini Alessandro

# Prova Finale Reti Logiche



POLITECNICO
MILANO 1863

a.a. 2022 - 2023

# Indice

**1 Introduzione** **2**
1.1 Funzionamento del componente . . . . . 2
1.2 Interfaccia del componente . . . . . 3

**2 Architettura del componente** **4**
2.1 Shift_register . . . . . 5
2.2 Output_register . . . . . 5
2.3 Fsm_register . . . . . 6

**3 Simulazione e sintesi** **8**
3.1 Esempio di funzionamento . . . . . 8
3.2 Testing . . . . . 9
3.3 Sintesi . . . . . 9

**4 Possibili ottimizzazioni** **10**
4.1 Riduzione degli stati . . . . . 10
4.2 Riduzione dei Flip Flop . . . . . 10

**5 Conclusioni** **12**

# Capitolo 1

# Introduzione

Lo scopo del progetto consiste nella realizzazione di un componente hardware per l'estrazione di dati da una memoria esterna a partire da un indirizzo serializzato. Il componente è dotato di quattro canali d'uscita, e l'input fornisce l'indirizzo da cui estrarre il dato dalla memoria esterna, oltre all'indirizzo della porta d'uscita sulla quale questo dato dovrà essere copiato. Il dispositivo permette, inoltre, di memorizzare l'ultimo dato proiettato su ognuno dei quattro canali, in modo da mantenere l'informazione fino ad un eventuale sovrascrittura.

## 1.1 Funzionamento del componente

Il dispositivo è dotato di due segnali di input: `i_start` e `i_w`. Quando il segnale `i_start` viene alzato, ha inizio una nuova richiesta. Ad ogni fronte di salita del clock viene letto il segnale `i_w`, il quale fornisce, in ordine, le seguenti informazioni:

- L'indirizzo della porta d'uscita sulla quale verrà fornito il dato finale (2 bit)
- L'indirizzo sul quale interrogare la memoria esterna (da cui verrà estratto il dato), fornito a partire dal bit più significativo (da 0 a 16 bit)

La lettura dell'input termina quando il segnale `i_start` viene abbassato.
L'interfaccia tra il componente e la memoria esterna è composta da quattro segnali:

- L'indirizzo da cui estrarre il dato `o_mem_addr` (16 bit)
- Un segnale che abilita la lettura dalla memoria `o_mem_en`
- Il dato fornito a seguito di una lettura `i_mem_data` (8 bit)
- Un segnale che abilita la scrittura in memoria `o_mem_we` (qui non utilizzato)

Una volta ricevuto il dato dalla memoria esterna, esso viene copiato sull'uscita indicata nella fase di lettura dell'input. I segnali in uscita dal componente sono:

- Quattro uscite: `o_z0`, `o_z1`, `o_z2`, `o_z3` (8 bit ciascuna)
- Un segnale `o_done` che comunica la fine dell'elaborazione e la validità dei segnali visibili in uscita

Il componente necessita, inoltre, di un segnale di input `i_clk` (clock) ed un segnale di reset asincrono `i_rst`, che resetta i valori memorizzati in uscita e annulla tutte le eventuali operazioni in corso, re inizializzando la macchina.

## 1.2 Interfaccia del componente

L'interfaccia del componente risulta quindi la seguente:

```
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;
```

- `i_clk` è il segnale di clock
- `i_rst` è il segnale di reset che inizializza la macchina
- `i_start` è il segnale che indica l'inizio di una nuova richiesta
- `i_w` è il segnale dove vengono indicati, un bit alla volta, i dati relativi alla richiesta
- `o_z0`, `o_z1`, `o_z2`, `o_z3` sono i quattro canali di uscita
- `o_done` è il segnale che comunica la fine dell'elaborazione
- `o_mem_addr` è il segnale di uscita che indica alla memoria esterna l'indirizzo su cui effettuare la lettura
- `i_mem_data` è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura
- `o_mem_en` è il segnale di enable da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura)
- `o_mem_we` è il segnale di write enable da dover mandare alla memoria per poter scriverci, mentre per leggere da memoria esso deve essere 0. Dato che le interazioni di questo componente con la memoria esterna sono di sola lettura, questo segnale è sempre basso

# Capitolo 2

# Architettura del componente

Un ciclo di funzionamento del componente si divide in tre macro-fasi:

- Lettura delle informazioni da input
- Interrogazione memoria e lettura del dato
- Copia del dato sull'uscita e termine dell'elaborazione

Viste le diverse azioni che il dispositivo deve svolgere, la sua architettura è stata suddivisa in tre componenti:

- shift_register : un registro dotato di shift logico per la memorizzazione dell'indirizzo ricevuto in input
- output_register : un banco di registri che memorizza il dato restituito dalla memoria nel registro corretto e copia sulle uscite i dati coerentemente con il segnale o_done
- fsm_register : una macchina a stati che tiene traccia del ciclo di funzionamento del dispositivo e fornisce agli altri componenti i segnali necessari

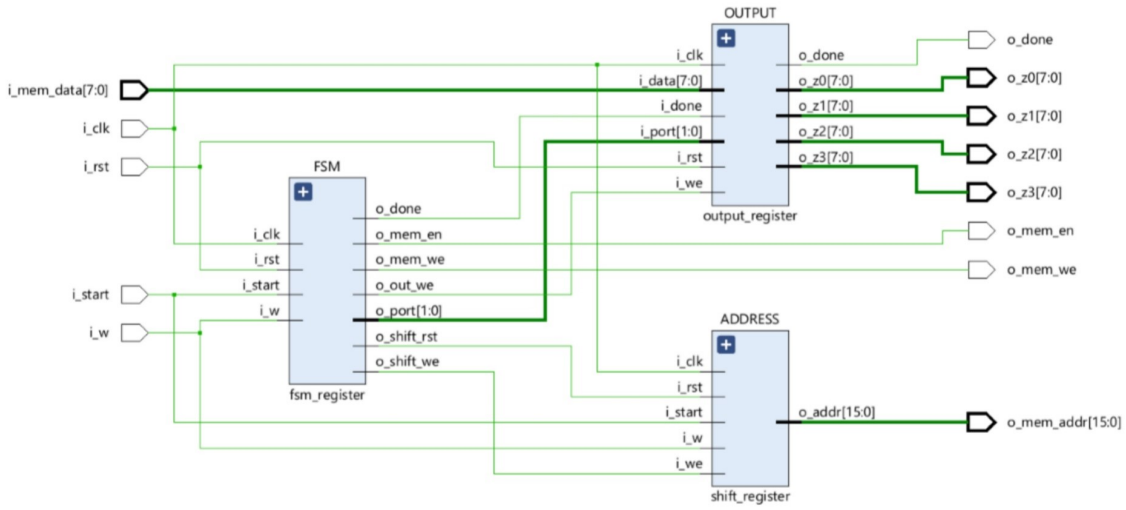



Figura 2.1: Schema generale del componente

## 2.1 Shift_register

Interfaccia di shift_register:

```
entity shift_register is
    port(
        i_clk : in std_logic;
        i_rst: in std_logic;
        i_start: in std_logic;
        i_w: in std_logic;
        i_we: in std_logic;
        o_addr: out std_logic_vector(15 downto 0)
        );
end shift_register;
```

Oltre ai segnali di clock e reset (segnale sincrono fornito dalla machina a stati all'inizio di ogni ciclo di elaborazione) il componente dispone di tre segnali di input così utilizzati: quando sul fronte di salita del clock sia `i_we` (fornito dalla macchina a stati) che `i_start` (fornito in input al dispositivo) sono alti, l'indirizzo da 16 bit memorizzato dal componente subisce uno shift sinistro: il bit più significativo viene scartato, mentre il bit meno significativo viene sostituito dal dato presente in `i_w`. In questo modo, quando `i_start` viene posto a 0, in uscita da `o_addr` sono visibili i 16 bit memorizzati, che compongono l'indirizzo corretto su cui interrogare la memoria.

## 2.2 Output_register

Interfaccia di output_register:

```
entity output_register is
    port(
        i_clk : in std_logic;
        i_rst: in std_logic;
        i_we: in std_logic;
        i_port: in std_logic_vector(1 downto 0);
        i_data: in std_logic_vector(7 downto 0);
        i_done: 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
        );

end output_register;
```

All'interno del componente sono presenti quattro registri di memoria, identificati dal relativo indirizzo a due bit. I segnali `o_z0`, `o_z1`, `o_z2`, `o_z3` e `o_done` sono direttamente collegati agli omonimi segnali in uscita al dispositivo, mentre `i_data` è il segnale in ingresso al dispositivo connesso alla memoria esterna.
Il funzionamento del componente è il seguente: quanto `i_we` viene posto a 1, il

registro corrispondente all'uscita indicata da `i_port` viene sovrascritto da `i_data`. Tra i registri ed i segnali d'uscita, i dati memorizzati sono posti in AND con il segnale `i_done`, in modo che quando `i_done` è alto il loro contenuto sia visibile, mentre quando è basso le uscite mostrino solo zeri.

## 2.3 Fsm_register

Interfaccia di fsm_register:

```
entity fsm_register is
    port(
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_start : in std_logic;
        i_w : in std_logic;
        o_port: out std_logic_vector(1 downto 0);
        o_done : out std_logic;
        o_mem_we : out std_logic;
        o_mem_en : out std_logic;
        o_out_we : out std_logic;
        o_shift_rst : out std_logic;
        o_shift_we : out std_logic
        );
end fsm_register;
```

Questo componente ha lo scopo di regolare i segnali da inviare a shift_register e output_register. È modellabile da una macchina a stati con 7 stati, le commutazioni tra uno stato ed il successivo avvengono sul fronte di salita del clock. Se non diversamente specificato, la macchina passa allo stato successivo sul fronte di salita seguente.
I sette stati sono:

1. `WAIT_START`
   Il componente rimane in attesa del segnale di inizio. Quando `i_start` viene posto a 1, viene attivato il reset di shift_register (segnale `o_shift_rst`), viene posto `o_port(1) = i_w` e l'automa passa allo stato successivo.

2. `READ_PORT`
   Viene letto il secondo bit della porta d'uscita, ovvero `o_port(0) = i_w`.

3. `READ_ADDRESS`
   Finchè `i_start` è alto, l'automa segnala a shift_register di leggere l'indirizzo in ingresso (segnale `o_shift_we`). Quando `i_start` scende, si passa allo stato successivo.

4. `ENABLE_MEMORY`
   Viene alzato il segnale `o_mem_en`, per effettuare la lettura del dato all'indirizzo memorizzato in shift_register.

5. `WAIT_MEMORY`
   Dopo 2 ns sull'ingresso `i_mem_data` è presente il dato da memorizzare.

6. `SAVE_DATA`
   L'automa segnala a output_register (segnale `o_out_en`) che il dato è pronto per essere memorizzato nel registro corrispondente alla porta indicata da `o_port`.

7. `PRINT_OUTPUT`
   Viene posto `o_done` a 1, in uscita saranno visibili i valori di `o_z0` … `o_z3` correttamente aggiornati.

Il segnale di reset azzera tutti i segnali e pone l'automa nello stato `WAIT_START`.
La durata dell'elaborazione tra la discesa del segnale `i_start` e la salita del segnale `o_done` è di 3 cicli di clock: il vincolo di tempo di 20 cicli di clock imposto dalla specifica è quindi rispettato.

# Capitolo 3

# Simulazione e sintesi

## 3.1 Esempio di funzionamento

Qui di seguito è riportato un esempio di funzionamento: in ingresso viene richiesto il dato all'indirizzo $3_{10}$ ($11_2$) della memoria esterna (che vale in questo caso $175_{10}$) sulla porta identificata dal codice $10_2$ (`o_z2`).

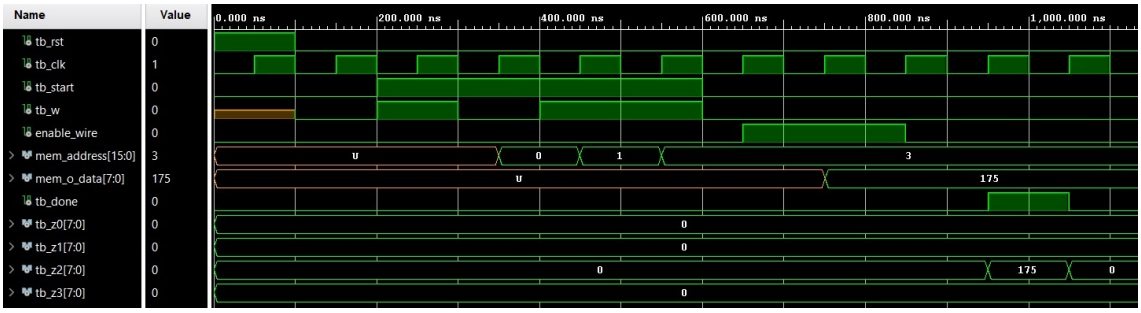



Figura 3.1: Diagramma temporale dei segnali in entrata ed in uscita

| Timestamp | Azione |
|---:|---|
| 0 ns | Il dispositivo viene resettato |
| 250 ns | `i_start` è alto, viene letto il primo bit del canale d'uscita |
| 350 ns | Secondo bit del canale d'uscita: l'uscita selezionata è $10_2$, ovvero `o_z2` |
| 450 ns | Inizia la lettura dell'indirizzo di memoria. Indirizzo: $1_2$ |
| 550 ns | Indirizzo: $11_2$ |
| 650 ns | `i_start` è basso: viene attivato `o_mem_en` per la lettura da memoria |
| 752 ns | Il dato (175) compare in ingresso al componente |
| 850 ns | Il dato viene memorizzato nel registro corrispondente a `o_z2` |
| 950 ns | `o_done` viene alzato: in uscita è possibile vedere il contenuto dei registri. Su `o_z2` è visibile 175, sulle altre uscite 0 |
| 1050 ns | `o_done` viene abbassato, il dispositivo torna nello stato di attesa |

## 3.2 Testing

Per verificare la correttezza dei componenti, sono stati realizzati diversi testbench. È stato testato, in particolare, il comportamento del dispositivo in diversi casi limite:

- Valori dell'input:
  - Indirizzo in ingresso con zeri nei bit più significativi
  - Multiple sovrascritture dei registri d'uscita
- Lunghezza dell'indirizzo in input:
  - Indirizzo in ingresso da 0 bit
  - Indirizzo in ingresso da 16 bit
- Tempistiche dell'input
  - Segnali `i_start` e `i_w` modificati durante un reset
  - Segnale `i_start` alto alla discesa di `o_done`
- Reset durante l'elaborazione:
  - Reset asincrono durante la fase di lettura dell'indirizzo
  - Reset asincrono durante la fase di lettura da memoria

Il dispositivo supera tutti i test in fase di pre-sintesi.

## 3.3 Sintesi

Il dispositivo è sintetizzabile, non presenta latch, rispetta il vincolo di funzionamento con un periodo di clock di 100 ns e in post-sintesi supera gli stessi testbench a cui è stato sottoposto in pre-sintesi.

**Design Timing Summary**

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 96,809 ns | Worst Hold Slack (WHS): | 0,168 ns | Worst Pulse Width Slack (WPWS): | 49,500 ns |
| Total Negative Slack (TNS): | 0,000 ns | Total Hold Slack (THS): | 0,000 ns | Total Pulse Width Negative Slack (TPWS): | 0,000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 132 | Total Number of Endpoints: | 132 | Total Number of Endpoints: | 73 |

**All user specified timing constraints are met.**

Figura 3.2: Timing report di Vivado

# Capitolo 4

# Possibili ottimizzazioni

Il dispositivo descritto in questa relazione rispetta tutti i vincoli della specifica.
È possibile attuare alcune piccole ottimizzazioni utili a ridurre le risorse utilizzate ed il tempo di elaborazione (ottimizzazioni non incluse nella versione finale del progetto in quanto non strettamente necessarie).

## 4.1 Riduzione degli stati

È possibile ridurre il numero degli stati dell'automa da sette a cinque, collassando gli stati `ENABLE_MEMORY`, `WAIT_MEMORY` e `SAVE_DATA` in un unico stato `FETCH_DATA`.
Il numero di bit necessari per memorizzare lo stato rimane invariato (3 bit), ma il tempo di elaborazione viene ridotto.
Per attuare questa ottimizzazione vanno effettuate le seguenti modifiche:

- Il segnale `o_mem_en` per la lettura da memoria viene alzato in modo asincrono sul fronte di discesa di `i_start`, in modo da avere il dato disponibile 2 ns dopo il fronte di salita successivo del clock
- La scrittura di `i_data` sul registro d'uscita selezionato avviene sul fronte di discesa del clock, a 48 ns dalla scrittura del dato su `i_data`
- Il segnale `o_done` viene alzato dopo un solo ciclo di clock dalla discesa di `i_start`

Il tempo di elaborazione successivo alla lettura dell'input passa così da 3 cicli di clock (300 ns) a 1 ciclo di clock (100 ns).

## 4.2 Riduzione dei Flip Flop

È possibile rimuovere i flip flop (generati in fase di sintesi) corrispondenti ai segnali in uscita da fsm_register, codificando esplicitamente gli stati dell'automa e creando una rete combinatoria che produca i singoli segnali in uscita dal componente. Per rimuovere i flip flop relativi al segnale `o_port`, è necessario includerne l'informazione all'interno del registro di stato, in quanto il codice della porta d'uscita selezionata deve essere memorizzato fino al termine dell'elaborazione (nello specifico, fino allo stato `FETCH_DATA`).

Il numero minimo di stati risulta quindi:

| Stato | # $P_1$ | # $P_0$ | # Combinazioni |
|---|---|---|---|
| `WAIT_START` | – | – | 1 |
| `READ_PORT` | 2 | – | 2 |
| `READ_ADDRESS` | 2 | 2 | 4 |
| `FETCH_DATA` | 2 | 2 | 4 |
| `PRINT_OUTPUT` | – | – | 1 |
| **Totale** | | | **12** |

Dove $P_1$ e $P_0$ indicano i bit del segnale `o_port`.
La nuova macchina minima ha un totale di 12 stati: il numero di bit necessari per rimuovere tutti i flip flop è $\lfloor \log_2(12) \rfloor + 1 = 4$ bit (per memorizzare lo stato corrente e `o_port` separatamente erano necessari 5 bit).
Una possibile codifica degli stati è la seguente:

| Stato | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| `WAIT_START` | 0 | 0 | 0 | 0 |
| `READ_PORT` | 0 | 0 | 1 | $P_1$ |
| `READ_ADDRESS` | 0 | 1 | $P_0$ | $P_1$ |
| `FETCH_DATA` | 1 | 1 | $P_0$ | $P_1$ |
| `PRINT_OUTPUT` | 1 | 0 | – | – |

Le reti combinatorie dei segnali in uscita da fsm_register risultano quindi:

`o_port(0)` $= Q_1$
`o_port(1)` $= Q_0$
`o_mem_we` $= 0$
`o_mem_en` $= Q_2\ (Q_3 + \overline{\texttt{i\_start}})$
`o_out_we` $= Q_3\ Q_2$
`o_shift_we` $= \overline{Q_3}\ Q_2$
`o_shift_rst` $= \overline{Q_3}\ \overline{Q_2}\ \overline{Q_1}\ \texttt{i\_start}$
`o_done` $= Q_3\ \overline{Q_2}\ \overline{\texttt{i\_rst}}$

Uno dei problemi che si possono riscontrare in post-sintesi è la non contemporaneità della commutazione dei bit di stato, che potrebbe generare dei segnali parassiti. Per questo motivo, la codifica è stata scelta minimizzando la distanza di Hamming tra stati adiacenti. Inoltre, nel caso avvenga un reset nello stato `FETCH_DATA` (dove la discesa di $Q_2$ prima di $Q_3$ potrebbe generare problemi), il segnale `o_done` è tenuto basso da $\overline{\texttt{i\_rst}}$, che durante un reset vale 0.
Con queste ottimizzazioni, il numero di flip flop utilizzati (in fase di sintesi, in fase di implementazione questo numero potrebbe aumentare) risulta: 16 (shift_register) + 32 (output_register) + 4 (fsm_register) = 52 flip flop, ovvero il minimo teorico.

# Capitolo 5

# Conclusioni

Il componente esposto in questo elaborato supera con successo tutti i testbench, sia in pre-sintesi che in post-sintesi. Così come nel progetto VHDL, la sua descrizione è stata suddivisa in tre componenti, in modo da aumentare la leggibilità del codice e rendere più agevoli future modifiche o l'implementazione di eventuali ottimizzazioni.