PROVA FINALE
PROGETTO DI RETI LOGICHE

PIERANTONIO MAURO -
MOUADH LTIFI - 955164 - 10723181

INDICE:

1. Introduzione

I. Scopo del Progetto

Il progetto consiste nello sviluppo di un modulo hardware descritto in VHDL che si interfacci con una memoria.
Il sistema riceve indicazioni circa una locazione di memoria e un canale di uscita e deve trasferire il contenuto della memoria nell'opportuna uscita.

II. Specifiche generali

Le indicazioni circa il canale da utilizzare e l'indirizzo di memoria a cui accedere vengono forniti al modulo mediante un ingresso seriale da un bit
I canali di uscita forniscono tutti i bit della parola di memoria in parallelo.

III. Interfaccia del modulo

L'interfaccia del modulo, attorno a cui ruota lo sviluppo del progetto, è come segue:

```
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;
```

dove in particolare:

- **i_clk** è il segnale di clock in ingresso al modulo;
- **i_rst** è il segnale di reset usato per inizializzare la macchina;
- **i_start** è un input che indica valido il contenuto dell'ingresso i_w;
- **i_w** è l'ingresso seriale su cui legge il modulo;
- **o_z0 … o_z3** sono i canali di uscita paralleli;
- **o_done** è il segnale di uscita che notifica la conclusione di un'operazione;

2. Architettura

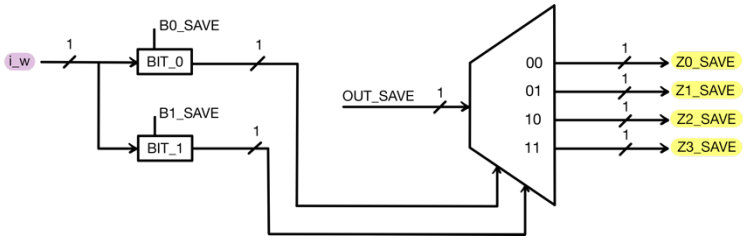
I. Scomposizione del problema

Ai fini della progettazione, il problema è stato scomposto in sotto-problemi:
- Selezione del canale di uscita
- Lettura dell'indirizzo di memoria
- Caricamento dati in uscita

II. Design del Datapath

a. Selezione del canale di uscita

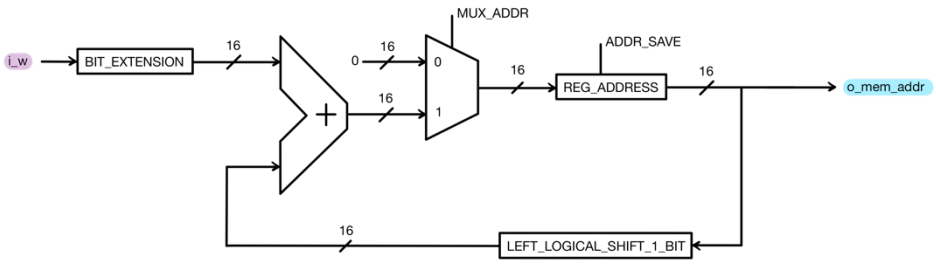
Questo blocco si occupa di leggere dall'ingresso seriale i due bit che rappresentano il canale di uscita voluto.
Ogni bit viene salvato in un registro dedicato e questi pilotano un demultiplexer il quale abilita uno dei quattro segnali usati successivamente per controllare i canali di uscita.



b. Lettura dell'indirizzo di memoria

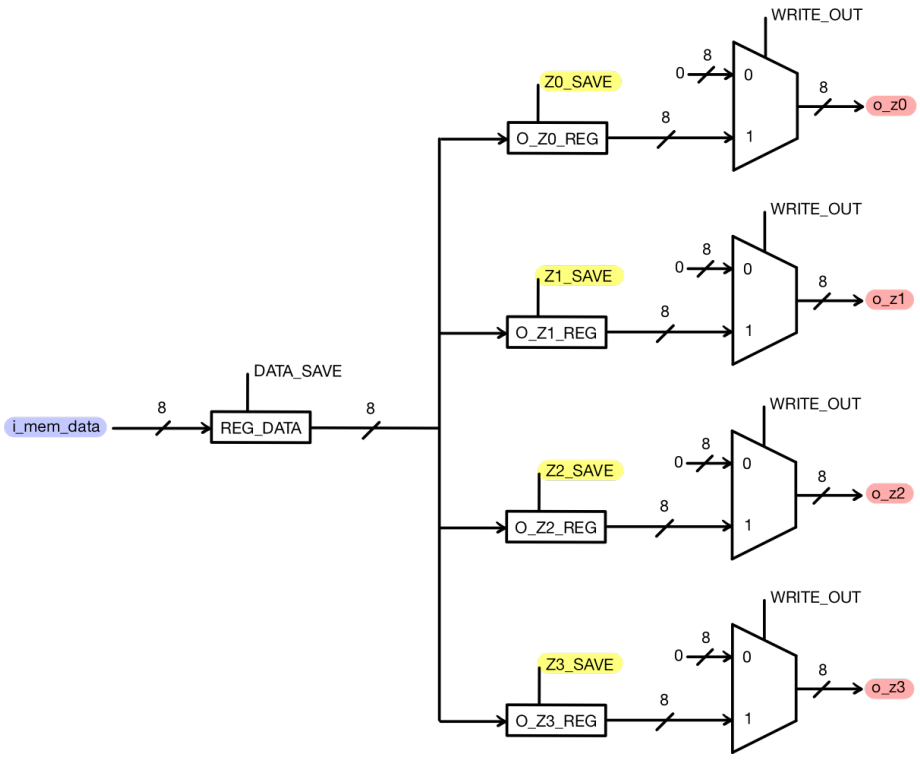
Qui l'obiettivo è quello di ottenere dall'ingresso seriale l'indirizzo su cui è salvato il dato.
Questa fase è critica per l'efficienza complessiva del progetto per due motivi:
- Richiede che l'indirizzo venga letto dal bit più significativo al bit meno significativo
- In caso l'indirizzo passato in ingresso fosse più corto dei 16 bit richiesti, questo va esteso con zeri nelle posizioni più significative

Una cattiva gestione di questa fase del progetto può portare a uno spreco di risorse in caso di indirizzi bassi.
Alcune implementazioni possibili, infatti, potrebbero usare un numero di cicli di esecuzione fisso a 16 per la lettura dell'indirizzo di memoria anche nei casi in cui questo non fosse necessario (per esempio il caso limite dell'indirizzo 0000 0001 che sarebbe gestibile con un solo ciclo).
La soluzione proposta è la seguente:

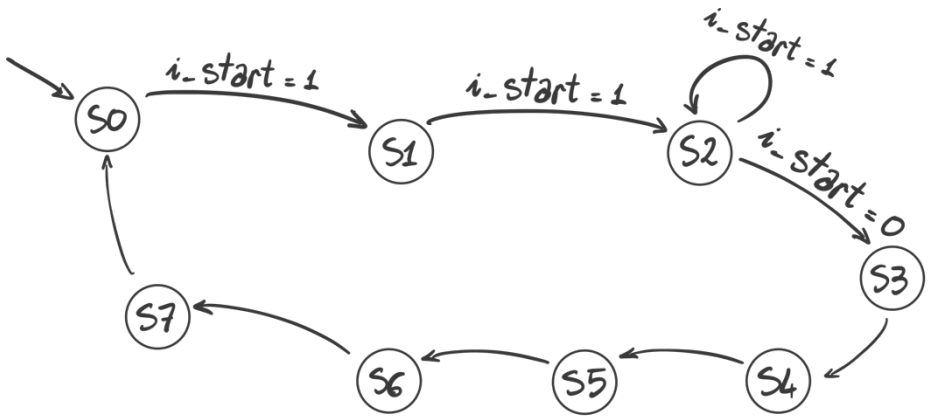


Il modulo prende un bit in ingresso e lo incorpora nella posizione meno significativa di un vettore di 16 bit zero.
Il vettore viene salvato nel registro REG_ADDRESS e per fare spazio al nuovo bit in arrivo al successivo ciclo di clock viene applicato uno shift logico a sinistra di una posizione su questo vettore.
Si usa quindi il vettore shiftato per eseguire un loop e sommarlo al bit, opportunamente esteso, in arrivo al successivo ciclo di clock.
Finita la lettura il contenuto di REG_ADDRESS può essere passato all'uscita o_mem_addr per proseguire con l'esecuzione.
La particolarità di questa soluzione risiede nel fatto che indipendente da quanto lungo è l'indirizzo letto in ingresso, il contenuto di REG_ADDRESS a ogni ciclo di clock è pronto per essere usato come o_mem_data senza ulteriori aggiustamenti nel caso in cui i_start diventasse zero al ciclo successivo.
Questo vale anche per i casi limite in cui l'indirizzo fosse formato interamente da bit zero o in cui l'indirizzo avesse un solo bit.

c. Caricamento dati in uscita
Questo blocco sfrutta le scelte mostrate sopra e, una volta caricati i dati in REG_DATA, passa all'uscita, selezionata al punto a., il contenuto del registro.



III. Macchina a stati
Il Datapath viene pilotato da una macchina a stati strutturata come segue:



dove:

- S0 - reset :
  La macchina inizializza il contenuto dei propri registri e delle uscite.
  Rimane in attesa dell'arrivo di un input valido.
  All'arrivo di un input valido, segnalato dalla permutazione di i_start da 0 a 1, la macchina salva il primo bit in arrivo che corrisponde al bit_0 di selezione dell'output e passa allo stato successivo.

- S1: reading_int_bit :
  La macchina legge in input e salva il bit_1 di selezione del canale di uscita.
  Passa poi allo stato successivo.

- S2: reading_address_bits :
  La macchina riceve in ingresso i bit che rappresentano l'indirizzo di memoria da cui sono contenuti i dati.
  Vengono ricevuti a partire dal bit più significativo fino ad arrivare al bit meno significativo.
  Fintanto che i_start è 1, la macchina rimane in questo stato continuando a leggere i bit in ingresso.
  Quando i_start diventa 0 si passa allo stato successivo.

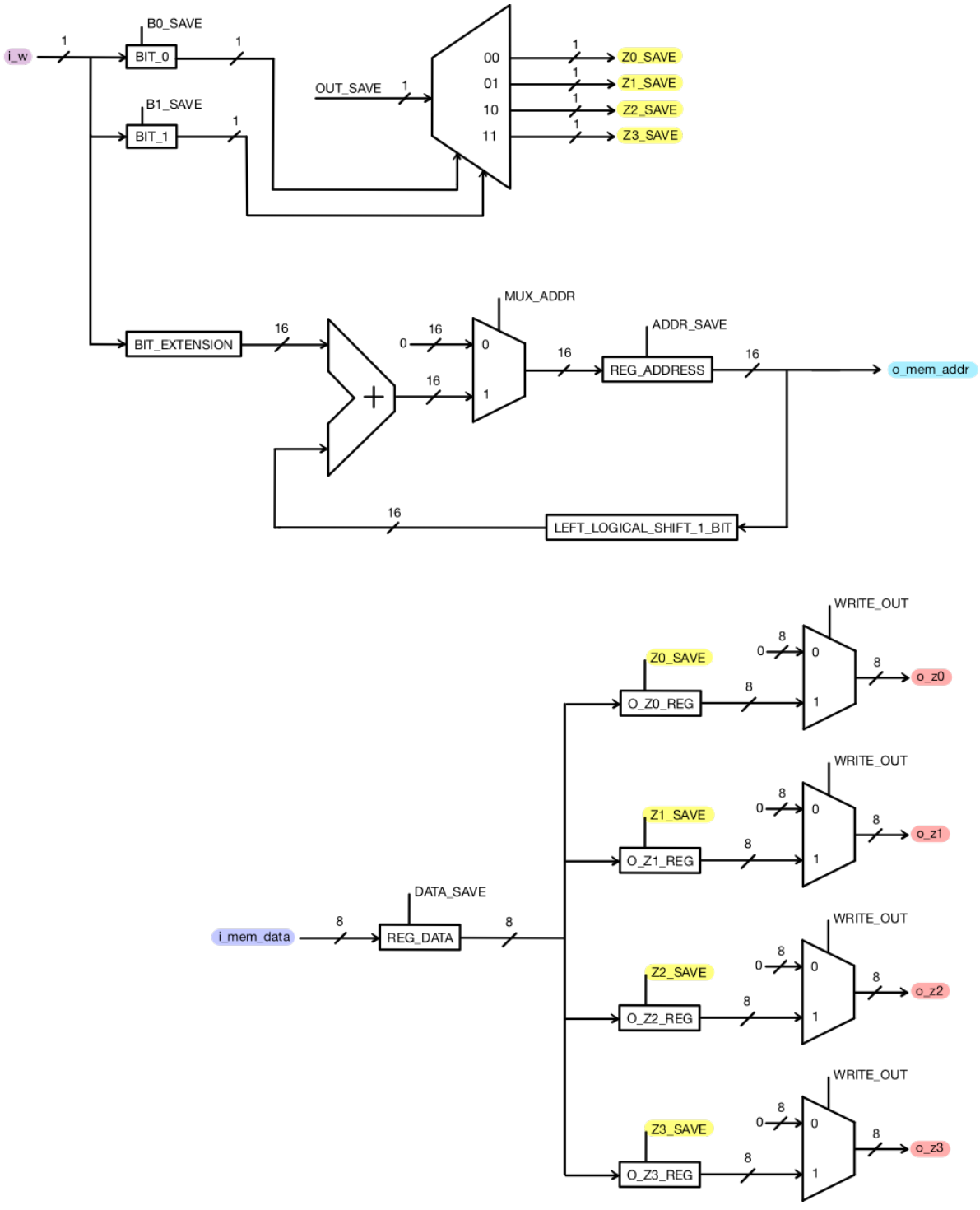
- S3: waiting_mem :
  Si usa l'indirizzo di memoria costruito nello stato precedente per comunicare l'indirizzo di memoria corretto.

- S4: loading_data :
  Si attende che il dato venga caricato dalla memoria in un registro dedicato.

- S5: selecting_channel:
  si abilita il Multiplexer associato all'uscita selezionata in modo da passare il contenuto del registro REG_DATA solo all'uscita voluta

- S6: saving_data
  //TODO

- S7: output_data
  L'uscita selezionata contiene il dato caricato.
  Le altre uscite contengono dei vettori 0000 0000.
  Si ritorna in attesa di un nuovo segnale in oppure, se segnalato, si passa allo stato di reset.


i_w
B0_SAVE
BIT_0
B1_SAVE
BIT_1
OUT_SAVE
00
01
10
11
Z0_SAVE
Z1_SAVE
Z2_SAVE
Z3_SAVE
BIT_EXTENSION
MUX_ADDR
ADDR_SAVE
REG_ADDRESS
o_mem_addr
LEFT_LOGICAL_SHIFT_1_BIT
WRITE_OUT
Z0_SAVE
O_Z0_REG
o_z0
Z1_SAVE
O_Z1_REG
o_z1
Z2_SAVE
O_Z2_REG
o_z2
Z3_SAVE
O_Z3_REG
o_z3
DATA_SAVE
i_mem_data
REG_DATA

3. Risultati Sperimentali

I. Sintesi

Il modulo risulta completamente e correttamente sintetizzabile sia con simulazione Post-Synthesis Functional che Timing tramite Vivado con una FPGA Artix-7 xc7a200tfbg484-1.
La sintesi non presenta inferred latch.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	36	0	0	134600	0.03
LUT as Logic	36	0	0	134600	0.03
LUT as Memory	0	0	0	46200	0.00
Slice Registers	86	0	0	269200	0.03
Register as Flip Flop	86	0	0	269200	0.03
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	97,500 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	4,500 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:	134	Total Number of Endpoints:	134	Total Number of Endpoints:	87

II. Simulazione

Nella realizzazione del modulo hardware, in aggiunta a quelli forniti, sono stati realizzati e utilizzati dei test bench che simulassero dei casi limite.
Il modulo supera con successo tutti i test proposti in:

- behavioural simulation
- post-synthesis functional simulation
- post-synthesis timing simulation

Riportiamo di seguito alcuni dei test che effettuato e alcune delle waveform relative alla loro esecuzione.

//WAVEFORM ESECUZIONE

III. Corner case considerati e gestiti

//TODO descrizione corner case

4. Conclusioni

I. Riflessioni finali

In conclusione, il progetto di sviluppo del modulo hardware descritto in VHDL che, interfacciandosi con una memoria, seleziona i dati voluti e li carica sul canale di uscita desiderato ha raggiunto i suoi obiettivi principali. La suddivisione del problema in sotto-problemi, come la selezione del canale di uscita, la lettura dell'indirizzo di memoria e il caricamento dei dati in uscita, ha permesso di affrontare in modo sistematico e organizzato le diverse fasi del processo. La soluzione proposta per la gestione dell'indirizzo di memoria si è dimostrata efficace nell'ottimizzare l'uso delle risorse, consentendo una gestione flessibile degli indirizzi di diversa lunghezza senza compromettere le prestazioni.

Il design del Datapath, con la macchina a stati, ha dimostrato di essere robusto e efficiente nella gestione delle diverse fasi del processo di trasferimento dati. La simulazione e i test eseguiti hanno confermato la correttezza e la funzionalità del modulo in scenari vari, inclusi casi limite e situazioni critiche. Le waveform delle simulazioni mostrano chiaramente il corretto comportamento del modulo in risposta a diversi input e situazioni.

II. Sfide affrontate

Come ogni progetto, anche questo ha affrontato alcune sfide e ha delle limitazioni da considerare. Durante lo sviluppo, è emersa la necessità di trovare un equilibrio tra efficienza e complessità del design, specialmente per quanto riguarda la gestione dell'indirizzo di memoria. Il vincolo di una soluzione flessibile per l'indirizzo ha richiesto un'attenta considerazione nella fase di disegno del datapath per ridurre al minimo lo spreco delle risorse computazionali.

In conclusione, il progetto ha offerto un'esperienza preziosa nella progettazione di circuiti digitali complessi, nella gestione delle sfide di ottimizzazione e nella validazione attraverso simulazioni approfondite.