Prova Finale
Progetto di Reti Logiche
A.A. 2019/2020 - prof. William Fornaciari

Domenico Cacace (Codice Persona REDACTED - Matricola REDACTED)

Introduzione

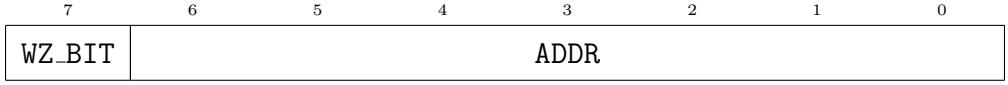
Si consideri un insieme di indirizzi di 0 a 127 e siano dati 8 indirizzi all'interno di questo range che identificano ognuno un intervallo di 4 indirizzi: scopo del progetto è descrivere un componente hardware in linguaggio VHDL che, dati in ingresso gli indirizzi base di questi intervalli e un *indirizzo da trasmettere*, produca una codifica del suddetto indirizzo usando una versione semplificata del metodo Working Zone.[1]

1 Descrizione Generale

1.1 Codifica Working Zone

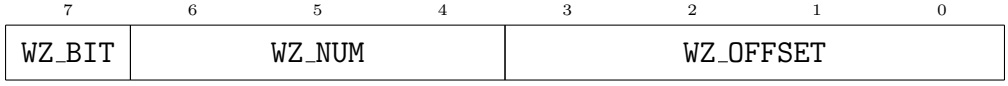
Nello schema di codifica Working Zone, si identificano un insieme di intervalli (Nwz) di indirizzi: ognuno di questi intervalli (Working Zone, WZ) contiene un numero costante di indirizzi consecutivi (Dwz), ed è identificato dal primo indirizzo in ordine crescente dell'insieme; le WZ sono numerate da 0 a Nwz-1, ma generalmente il numero di una WZ non ne riflette la posizione nell'insieme degli indirizzi.
Ai fini del progetto consideriamo Nwz=8, Dwz=4 e dimensione dell'indirizzo da trasmettere di 7 bit. La codifica dell'indirizzo (ADDR) a 7 bit produce un indirizzo ad 8 bit, in cui il MSB (WZ_BIT) indica come interpretare i restanti 7:

- WZ_BIT=0: ADDR non è contenuto all'interno di nessuna WZ, i restanti 7 bit sono l'indirizzo di partenza; l'indirizzo codificato è (WZ_BIT & ADDR).



- WZ_BIT=1: ADDR è contenuto in una WZ, i restanti 7 bit indicano il numero della WZ e l'offset dall'indirizzo base:
  - WZ_NUM (3 bit): il numero della WZ a cui appartiene ADDR, codificato in binario;
  - WZ_OFFSET (4 bit): l'offset tra ADDR e l'indirizzo di base della WZ, in codifica One-Hot

  l'indirizzo codificato è quindi (WZ_BIT & WZ_NUM & WZ_OFFSET).



[1] E. Musoll, T. Lang and J. Cortadella, "Working-zone encoding for reducing the energy in microprocessor address buses," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 6, no. 4, pp. 568-572, Dec. 1998, doi: 10.1109/92.736129.

1.2 Specifiche progettuali

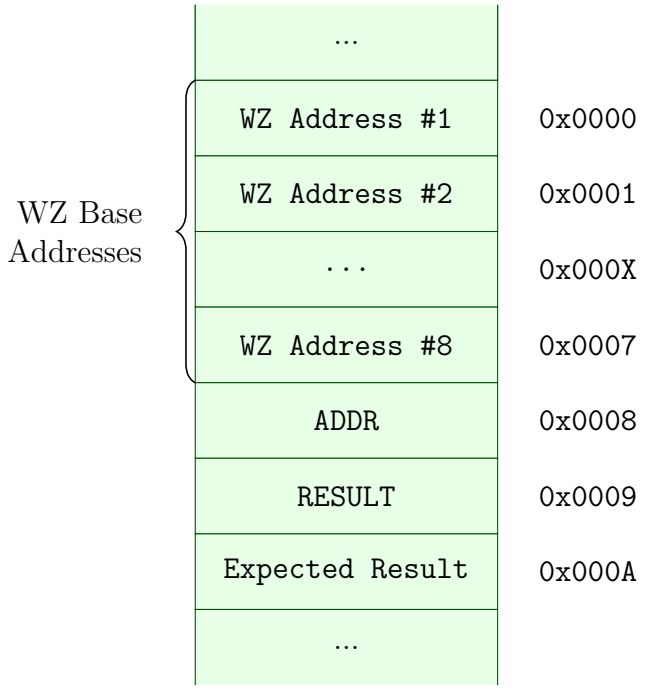
Da specifica, è richiesto che il componente abbia la seguente interfaccia per poter comunicare con un testbench:

```
entity project_reti_logiche is
  port (
    i_clk      : in   std_logic;
    i_start    : in   std_logic;
    i_rst      : in   std_logic;
    i_data     : in   std_logic_vector(7 downto 0);
    o_address  : out  std_logic_vector(15 downto 0);
    o_done     : out  std_logic;
    o_en       : out  std_logic;
    o_data     : out  std_logic;
  );
end project_reti_logiche;
```

In particolare:

- `i_clk`: segnale di `CLOCK` (a 10MHz) generato dal testbench;
- `i_start`: segnale di `START` generato dal testbench: viene tenuto ad `1` per tutta la durata dell'esecuzione, e dev'essere riportato a `0` dopo che è stato alzato il segnale di `o_done`;
- `i_rst`: segnale di `RESET` generato dal testbench: quando viene portato ad `1`, anche in modo asincrono rispetto ad `i_clk`, la macchina deve tornare allo stato di attesa;
- `i_data`: contiene il valore che arriva dalla memoria a seguito di una richiesta di lettura;
- `o_address`: indirizzo di memoria a 16 bit da cui leggere/scrivere;
- `o_done`: segnale di uscita, va alzato ad `1` per un ciclo di clock per segnalare la fine dell'esecuzione;
- `o_en`: segnale di `ENABLE`, da alzare ad `1` per poter comunicare con la RAM;
- `o_we`: segnale di `WRITE_ENABLE`, dev'essere posto ad `1` per poter scrivere su RAM, `0` per leggere; va usato insieme al segnale di `ENABLE`;
- `o_data`: valore da scrivere in memoria a seguito di una richiesta di scrittura.

Nel testbench è presente una memoria RAM, composta da 65536 celle di 8 bit ciascuna; di queste, sono utilizzate solo le prime 10(+1):



Le prime 8 celle contengono gli indirizzi base delle rispettive `WZ`, la nona cella contiene l'indirizzo `ADDR` da codificare, mentre nella decima cella dev'essere scritto il risultato dell'elaborazione. Durante la fase di simulazione, è stata utilizzata anche l'undicesima cella, per poter confrontare l'output atteso e quello effettivo e automatizzare l'analisi dei risultati prodotti. Le restanti celle non sono utilizzate.
La RAM utilizzata è di tipo sincrono, per cui bisogna attendere un ciclo di clock dopo aver settato il segnale di `ENABLE` prima di poter leggere i dati.

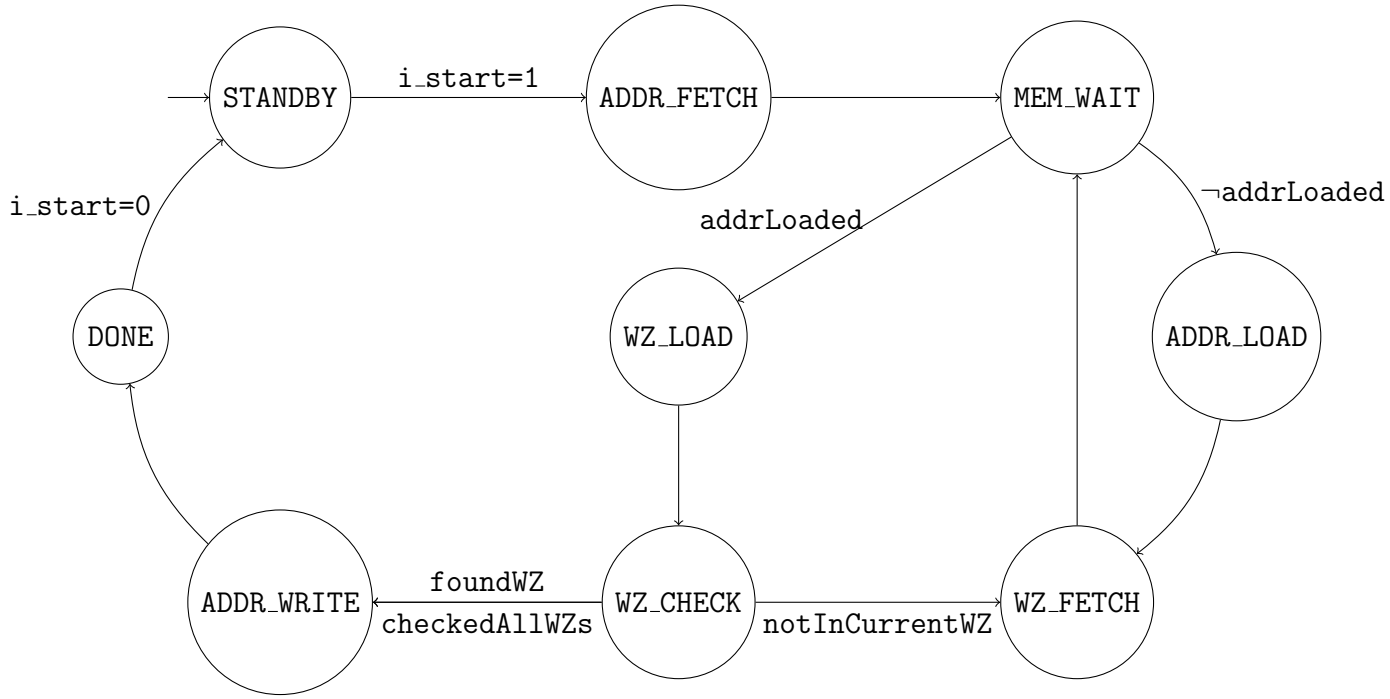
2 Design della Macchina a Stati Finiti

2.1 Stati della FSM

Seguendo le specifiche sopra riportate, si può procedere alla definizione degli stati dell'automa.

- `STANDBY`: la macchina attende il set del segnale di `START`; una transizione verso questo stato può avvenire da qualsiasi altro stato quando viene settato il segnale di `RESET`, oppure a fine esecuzione in modo "automatico" dallo stato di `DONE`. Quando il segnale di `START` viene abilitato, la macchina passa allo stato `ADDR_FETCH`;
- `ADDR_FETCH`: la macchina richiede la lettura dalla memoria di `ADDR` dalla cella di memoria `0x0008`, quindi passa allo stato `MEM_WAIT`;
- `MEM_WAIT`: la macchina attende *l'arrivo* dei dati dalla memoria; passa quindi allo stato `ADDR_LOAD` se `ADDR` non è stato ancora letto per la prima volta, altrimenti passa in `WZ_LOAD`;
- `WZ_FETCH`: la macchina richiede la lettura della `WZ` da analizzare dalla memoria, quindi passa allo stato `MEM_WAIT`; si è ritenuta una buona scelta dividere gli stati di `FETCH` e `LOAD`, poiché così facendo la gestione dei dati è risultata più semplice;
- `ADDR_LOAD`: la macchina memorizza il valore dell'indirizzo da codificare e segnala, tramite un `signal`, l'avvenuta lettura di `ADDR`; passa quindi a leggere il valore della prima `WZ` passando allo stato di `WZ_FETCH`
- `WZ_LOAD`: la macchina memorizza il valore della `WZ` da analizzare, e passa quindi allo stato `WZ_CHECK`;
- `WZ_CHECK`: la macchina verifica se `ADDR` è contenuto nell'ultima `WZ` caricata o meno: calcola la differenza tra `ADDR` e la `WZ` corrente e, se compresa tra 0 e 3 (inclusi), *assembla* il risultato e passa allo stato `ADDR_WRITE`; questo stato viene raggiunto anche nel caso non ci sia stato riscontro positivo per tutte le `WZ`; in caso di riscontro negativo e presenza di ulteriori `WZ` da analizzare, la macchina passa di nuovo allo stato di `WZ_FETCH`;
- `ADDR_WRITE`: la macchina richiede la scrittura in memoria del risultato calcolato allo stato `WZ_CHECK`;
- `DONE`: la macchina segnala che ha terminato il calcolo, alzando il segnale di `DONE` per un ciclo di clock, per poi passare allo stato di `STANDBY` in cui resterà finché il segnale `START` non sarà portato di nuovo alto;

Di seguito è riportato uno schema della macchina appena descritta.



3 Implementazione e Ottimizzazioni

L'idea di partenza dell'implementazione del componente non si discosta molto dal prodotto finale: il componente è composto da due `process`, in cui il primo si occupa della gestione dei segnali di `START` e `RESET`, mentre l'altro gestisce le transizioni dell'automa.
È stata valutata l'ipotesi di unire gli stati `ADDR_LOAD` e `WZ_LOAD`, in quanto svolgono pressoché la stessa azione, ma la loro separazione è risultata più semplice da implementare; inoltre, questo dettaglio non dovrebbe influire sul numero di cicli di clock necessari alla terminazione di un task.

4 Testing

Dopo aver implementato la FSM, questa è stata sottoposta a diversi test. Dopo aver analizzato le diverse possibilità, sono stati individuati tre casi d'interesse:

- Ricezione del primo segnale di `START`
- Ricezione del secondo segnale di `START`, senza `RESET` della macchina
- `RESET` asincrono dopo un numero casuale di cicli di clock

Questi tre casi vanno valutati sia per indirizzi contenuti all'interno di una `WZ` che per indirizzi non contenuti. Dopo aver testato alcuni casi limite *manualmente*, è stato realizzato in Python un generatore automatico di casi di test: questo generatore estrae casualmente le `WZ` e l'indirizzo `ADDR` da codificare, facendo attenzione a coprire i 6 casi sopra descritti.
Il testbench scritto carica quindi in memoria quanto generato, incluso il risultato atteso; alla fine di ogni computazione della macchina, il risultato viene confrontato e riportato su un altro file, in seguito analizzato tramite un altro script. Eseguendo una simulazione *post-sintesi funzionale* con 9 milioni di casi di test, sono stati riscontrati i seguenti risultati (espressi in cicli di `CLOCK`):

Tipo test	Max	Min	Avg	StDev
`ADDR` in `WZ`	37	9	23	9
`ADDR` non in `WZ`	37	37	37	0
`RESET`	10	4	7	2

I dati qui riportati si riferiscono al tempo intercorso tra l'innalzamento del segnale di `START` dal testbench e l'innalzamento del segnale di `DONE` della macchina.

Per i test che prevedono `RESET` asincroni, il segnale `i_rst` è stato generato all'interno del testbench; le statistiche sui tempi di reset sopra riportate sono espresse in numero di cicli di clock, e non sono considerate nelle statistiche dei test.
Si riporta di seguito un esempio di diagramma d'onda: nel caso in questione, `ADDR` è contenuto nella `WZ2`, ma la macchina riceve un segnale di `RESET` durante il fetch della seconda `WZ`; viene quindi riportata allo stato di `STANDBY`, dal quale ricomincia e porta a termine il calcolo.