Prova Finale (Progetto di Reti Logiche)

Anno 2019/2020

Prof. Fabio Salice

Luca Morandini

Codice Persona: 10622606
Matricola: 890478

Politecnico di Milano
Consegna 1° settembre 2020

Indice

1. Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è implementare il metodo di codifica a bassa dissipazione di potenza denominato Working Zone.
Questo metodo di codifica specifica il formato dei bit che compongono un indirizzo da trasmettere sul bus indirizzi. La riduzione del consumo di energia viene realizzata attraverso una speciale trasformazione degli indirizzi che appartengono a specifici intervalli chiamati Working Zone (WZ).
La colonna portante di questa codifica è il principio di località spaziale e temporale, tipico della maggior parte dei software, il quale indica che gli indirizzi vicini ad un dato indirizzo saranno generati con buona probabilità nel breve termine.
Una Working Zone è quindi un insieme di indirizzi adiacenti che vengono spesso utilizzati in un intervallo di tempo.

1.2 Specifiche di progetto

Gli indirizzi trattati hanno dimensione 7 bit ed il risultato della codifica viene rappresentato su 1 byte.
La specifica richiede che un indirizzo non appartenente a nessuna WZ deve essere riportato identico in uscita sui 7 bit meno significativi e `WZ_BIT=0` (bit più significativo) per indicare che il valore non è stato modificato, altrimenti il risultato dell'operazione *(fig. 1)* è formato dalla concatenazione di `WZ_BIT=1`, per indicare che l'indirizzo è stato codificato, 3 bit che rappresentano il numero della WZ (`WZ_NUM`) codificato in binario e 4 bit che rappresentano l'offset dell'indirizzo elaborato all'interno della WZ (`WZ_OFFSET`) codificato in one-hot.

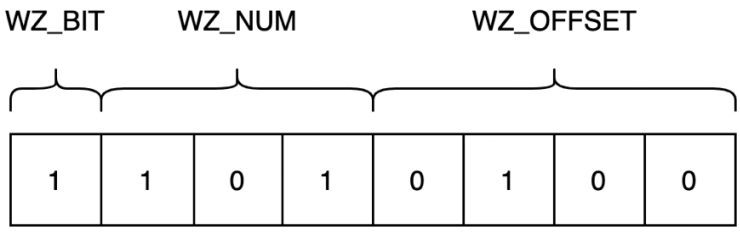

Fig. 1: esempio di indirizzo codificato

Il modulo *(fig. 2)* riceve dall'esterno il segnale `i_start` che determina l'inizio della codifica e comunica la terminazione del calcolo attraverso il segnale `o_done`.
Il componente realizzato riceve anche un segnale `i_rst` che viene utilizzato per interrompere l'esecuzione e riportare lo stato interno ad una situazione iniziale dove il modulo è pronto per ricevere un livello alto su `i_start` per cominciare una nuova codifica.

Il modulo si interfaccia con una RAM in cui sono memorizzate 8 WZ e 1 indirizzo da codificare. Nella stessa memoria verrà salvato il risultato dell'elaborazione, ovvero l'indirizzo codificato.

Il modulo interagisce con la memoria attraverso due linee dati di 8 bit per inviare (`o_data`) e ricevere (`i_data`) i valori da elaborare e specifica l'indice della cella di memoria richiesta attraverso il bus indirizzi `o_address` di 16 bit. I segnali di controllo `o_en` e `o_we` servono rispettivamente per abilitare la RAM e per attivare la modalità di scrittura.

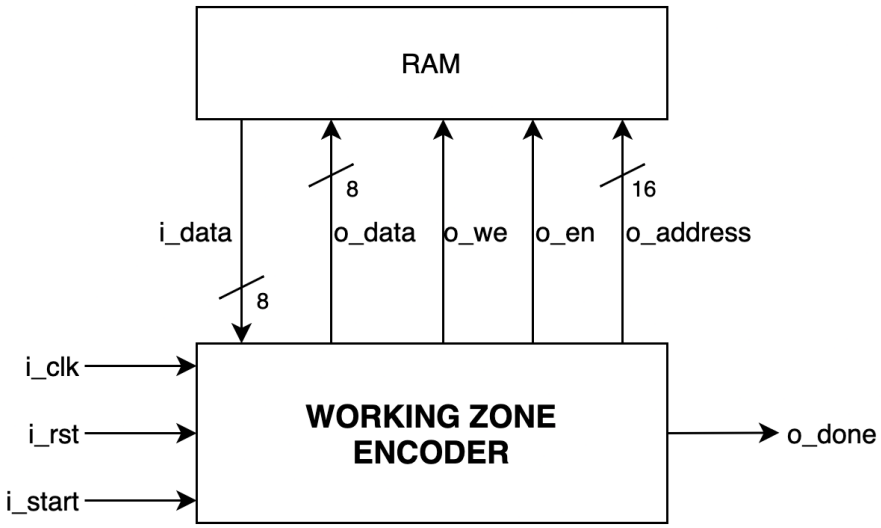

Fig. 2: schema del modulo implementato

2. Architettura

2.1 Scelte progettuali

Il componente è stato realizzato attraverso un unico modulo che implementa una macchina a stati finiti il cui scopo è quello di interagire con l'esterno, comunicare con la memoria RAM ed implementare l'algoritmo di codifica.
La FSM è stata implementata utilizzando un singolo `process` che definisce gli stati della macchina e gestisce i registri interni dove vengono salvati lo stato corrente, l'indirizzo da codificare ed i risultati intermedi del calcolo.

Alcuni segnali di output, ovvero `o_en` e `o_we`, non sono gestiti direttamente dalla FSM in modo sequenziale ma sono assegnati attraverso istruzioni puramente combinatorie concorrenti al `process`.
Il motivo di questa scelta sono la diminuzione dell'area utilizzata e la riduzione del percorso critico del componente (maggiori dettagli nella sezione *3.2 Ottimizzazioni*).

L'algoritmo che verifica l'appartenenza ad una Working Zone *(fig. 3)* calcola la differenza con l'indirizzo base di ogni WZ, controlla che questo valore sia minore della dimensione delle WZ ed in caso positivo codifica l'indirizzo secondo la specifica (sezione *1.2 Specifiche di progetto*).

```
diff <= address - unsigned(i_data);

if (diff < 4) then
      -- encode address
end if;
```

Fig. 3: calcolo e controllo dell'appartenenza ad una WZ

2.2 Macchina a stati

La macchina carica sequenzialmente le WZ dalla RAM, ogni volta che bisogna processare un indirizzo, senza salvarle in registri interni per risparmiare FF e quindi ridurre l'area utilizzata. Questa scelta è stata guidata dal fatto che potenzialmente gli indirizzi base delle Working Zone potrebbero cambiare frequentemente ed in questo caso il tempo necessario per aggiornare i registri sarebbe maggiore rispetto al tempo utilizzato per la fase di codifica.

Per ridurre il tempo di esecuzione della codifica, si è cercato di parallelizzare il più possibile le operazioni, evitando di sprecare cicli di clock in attesa del caricamento dei dati dalla RAM.

Nella versione finale della FSM *(fig. 4)* sono necessari solo due stati di attesa per caricare l'indirizzo da codificare e la prima WZ, successivamente ad ogni ciclo di clock viene processata una WZ finché ne viene trovata una che contiene l'indirizzo da codificare o fino a quando le WZ disponibili sono finite. Al termine del calcolo è necessario un ciclo di clock per salvare il risultato in memoria.

Grazie a questa architettura, mentre viene verificato se l'indirizzo da codificare appartiene ad una WZ, viene calcolata la differenza tra l'indirizzo da codificare e l'indirizzo base della WZ successiva (letto dalla RAM nel ciclo di clock precedente) ed allo stesso tempo viene caricato sul bus indirizzi della RAM, l'indice in memoria della WZ seguente che sarà disponibile nel ciclo di clock successivo.

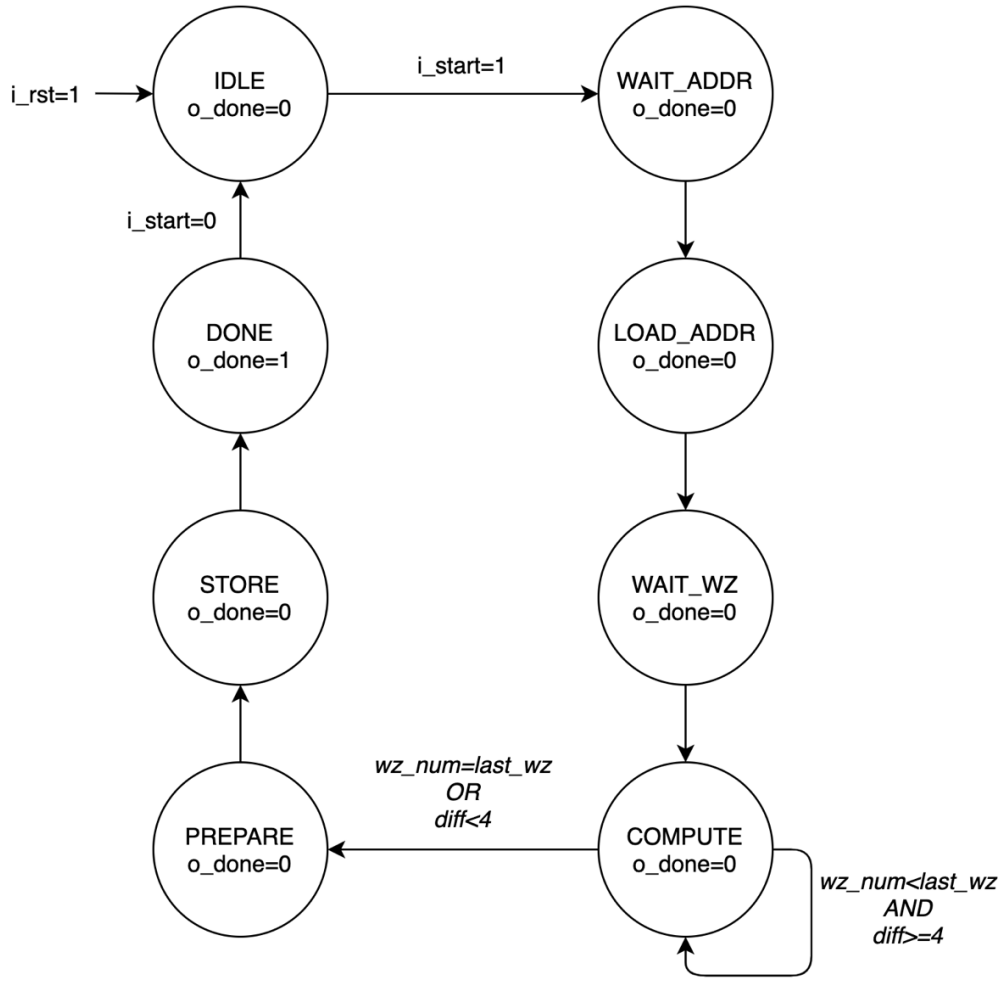

Fig. 4: diagramma della macchina a stati

Di seguito viene riportata una breve descrizione degli stati che compongono la FSM realizzata:

2.2.1 IDLE

Tutti i segnali ed i registri interni vengono inizializzati, sul bus indirizzi della RAM viene preparato l'indirizzo di memoria del valore da codificare e si attende un livello alto sul segnale di start per iniziare la computazione;

2.2.2 WAIT_ADDR

In questo ciclo di clock viene abilitata la RAM in lettura e si attende il caricamento dalla memoria dell'indirizzo da codificare.
Durante questo periodo, viene precaricato sul bus indirizzi della RAM l'indirizzo della prima WZ che verrà presentata sul bus dati in uscita della memoria nello stato successivo;

2.2.3 LOAD_ADDR

L'indirizzo da codificare, letto dal bus dati della RAM, viene salvato in un registro interno alla macchina.
Mentre si attende il caricamento della prima WZ, indirizzata nello stato precedente, si precarica sul bus indirizzi della RAM l'indirizzo della seconda WZ che sarà disponibile nel seguente ciclo di clock;

2.2.4 WAIT_WZ

In questa fase sul bus dati della RAM è presente l'indirizzo base della prima WZ che viene utilizzato per calcolare la differenza con l'indirizzo da codificare.
Il risultato viene memorizzato in un registro il cui valore sarà disponibile nel ciclo di clock successivo. Nello stesso stato viene richiesta alla RAM la WZ successiva;

2.2.5 COMPUTE

In questo stato viene controllata la differenza calcolata nel ciclo di clock precedente e, se il valore da codificare appartiene alla WZ, viene codificato l'indirizzo in uscita e si passa allo stato successivo per memorizzare il risultato.
In caso negativo, nel ciclo di clock successivo verrà controllata la prossima WZ.
In questo stato viene inoltre calcolata la differenza con la WZ caricata nel ciclo di clock precedente, che verrà valutata nel prossimo ciclo, e viene richiesta alla RAM la prossima WZ che verrà processata nel ciclo di clock consecutivo;

2.2.6 PREPARE

Viene preparata la fase di scrittura in memoria caricando sul bus indirizzi della RAM l'indirizzo in cui salvare il risultato della codifica;

2.2.7 STORE

In questo stato viene abilitata la RAM in scrittura e si attende che la memoria salvi il risultato caricato sul bus dati nella fase `COMPUTE`;

2.2.8 DONE

Il segnale di done viene portato al livello alto per indicare la fine della computazione e si attende che il segnale di start venga riportato al valore logico basso per poter ritornare nello stato `IDLE` in attesa del prossimo indirizzo da codificare.

3. Risultati sperimentali

3.1 Risultati della sintesi

Il componente viene correttamente sintetizzato da Vivado e supera entrambe le simulazioni Post-Synthesis Functional e Timing con il periodo di clock richiesto dalle specifiche (100 ns).

Di seguito viene riportato un estratto delle parti più significative del report di sintesi *(fig. 5)* riguardanti l'utilizzo dei componenti logici presenti sulla scheda FPGA.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	39	0	134600	0.03
LUT as Logic	**39**	0	134600	**0.03**
LUT as Memory	0	0	46200	0.00
Slice Registers	39	0	269200	0.01
Register as Flip Flop	**39**	0	269200	**0.01**
Register as Latch	0	0	269200	0.00

Fig. 5: estratto del report di sintesi di Vivado

Il modulo utilizza 39 LUT (0,03% di quelle disponibili sulla FPGA assegnata) per implementare la logica del codificatore e necessita di 39 registri (0,01% dei registri presenti sulla scheda) utilizzati come Flip-Flop per memorizzare lo stato della macchina e gli altri dati necessari durante la codifica.

3.2 Ottimizzazioni

Considerando che il modulo sviluppato è un codificatore di indirizzi, le ottimizzazioni sono state concentrate principalmente sulla riduzione del tempo di esecuzione. Ovviamente è sempre stata analizzata anche l'area occupata per evitare di utilizzare un numero eccessivo di componenti necessari ad implementare l'algoritmo di codifica.

Come anticipato nella sezione *2.1 Scelte progettuali*, per ridurre il tempo di esecuzione della macchina si è cercato di ottimizzare principalmente due parametri: diminuire il periodo di clock minimo e ridurre il numero di stati richiesti per eseguire le operazioni.

La riduzione del periodo di clock è stata ottenuta riducendo le istruzioni sequenziali all'interno del `process` e sostituendole con assegnamenti concorrenti che vengono eseguiti in parallelo alla logica della FSM, con l'obiettivo di diminuire il percorso critico del circuito.
A seguito di un'analisi delle `waveform` *(fig. 6)*, è stato notato che i segnali `o_we` e `o_en` dipendono esclusivamente dallo stato corrente della macchina, sono quindi degli ottimi candidati per essere gestiti con una logica puramente combinatoria.

L'ottimizzazione della logica della macchina ha ridotto il numero di stati ed inoltre, grazie al parallelismo delle operazioni (illustrato nella sezione *2.2 Macchina a stati*), il tempo di esecuzione nel caso peggiore (indirizzo appartenente all'ultima WZ) è stato ridotto da 2850 ns a 1850 ns in simulazione Behavioral Functional e da 2953 ns a 1953 ns in simulazione Post-Synthesis Timing.

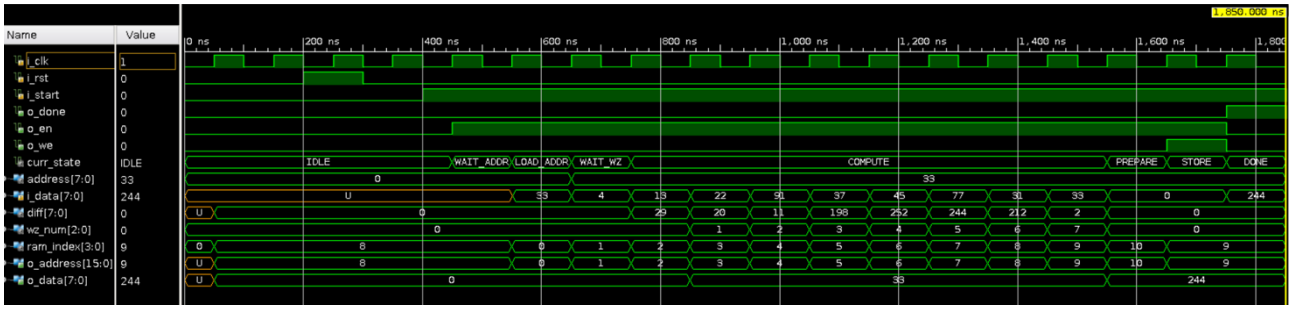

Fig. 6: waveform dopo le ottimizzazioni

Per quanto riguarda l'area utilizzata, il numero di LUT è stato ridotto da 69, nella prima versione, a 39 nella revisione finale, mentre i registri utilizzati come Flip-Flop sono stati diminuiti da 61 a 39.
Grazie a queste riduzioni di componenti, anche il percorso critico è stato leggermente diminuito e, come diretta conseguenza, la frequenza di clock massima può essere aumentata per ottenere tempi di esecuzione inferiori.

Un'ulteriore piccola ottimizzazione ha coinvolto il registro interno in cui viene memorizzato il successivo indirizzo da caricare dalla RAM.
Nella prima versione l'intero indirizzo di 16 bit veniva memorizzato, ma siccome gli indirizzi di memoria utili al componente sono solamente i primi 9, è stato deciso di salvare solo i 4 bit meno significativi del bus indirizzi per diminuire il numero di LUT e FF utilizzate per implementare il registro.
Certamente questa soluzione non è scalabile se le WZ da analizzare diventassero più di 15 (con 4 bit è possibile indirizzare i primi 16 indirizzi dei quali l'ultimo è utilizzato per l'output) ma, viste le specifiche del progetto, andrebbe comunque aggiornato l'intero modulo poiché non sarebbe più sufficiente 1 byte per salvare l'indirizzo codificato.
In sostanza questa ottimizzazione è accettabile perché per costruzione il modulo non è scalabile. In una situazione differente, dove il numero di WZ potrebbe essere più elevato, sarebbe stato salvato l'intero indirizzo.

4. Simulazioni

Il modulo è stato sottoposto ad una serie di test generati casualmente per verificare il corretto comportamento in una condizione di utilizzo realistica nel quale vengono codificati indirizzi consecutivi con lo stesso insieme di WZ (più segnali `i_start` successivi) con periodici aggiornamenti delle WZ in memoria (viene attivato il segnale `i_rst` per resettare il modulo).

Oltre ai test generati casualmente tramite uno script, sono stati realizzati dei testbench progettati per verificare alcuni casi limite ed alcune situazioni particolari non coperte dagli altri testbench.

4.1 Working Zone di indice 0

In questo testbench sono stati creati alcuni test nei quali l'indirizzo da codificare appartiene sempre al primo slot in memoria (`WZ_NUM=0`). Sono stati verificati i casi con la prima WZ (0), una WZ intermedia (53) e l'ultima WZ possibile (124).
Per ognuno di questi tre indirizzi base è stato fatto codificare alla macchina il primo e l'ultimo indirizzo appartenente ad ogni WZ.

Ad esempio per la WZ con indirizzo base 124, sono stati testati questi due casi:

- `WZ_NUM=0, ind_base=124, ind_da_codificare=124`
- `WZ_NUM=0, ind_base=124, ind_da_codificare=127`

Lo scopo di questi test è validare l'uscita anticipata dal ciclo di controllo delle Working Zone per verificare che non vengano processate le WZ successive alla prima.

4.2 Working Zone di indice 4

Questo testbench è simile a quello spiegato in precedenza poiché utilizza le stesse coppie di WZ ed indirizzi da codificare (prima/intermedia/ultima WZ e primo/ultimo indirizzo delle WZ). L'unica differenza è che in questo testbench tutte le WZ sono collocate nello slot 4 (`WZ_NUM=4`).

Ad esempio per la WZ con indirizzo base 0, sono stati effettuati questi test:

- `WZ_NUM=4, ind_base=0, ind_da_codificare=0`
- `WZ_NUM=4, ind_base=0, ind_da_codificare=3`

L'obiettivo è assicurarsi che le WZ vengano processate in sequenza fino a trovare quella contenente l'indirizzo da codificare e controllare il corretto passaggio alla fase di salvataggio senza continuare a caricare le WZ successive.

4.3 Working Zone di indice 7

Questo testbench è identico ai due illustrati in precedenza con la sola differenza che la WZ ricercata è sempre collocata nell'ultimo slot di memoria (`WZ_NUM=7`).

Lo scopo è verificare che tutte le WZ vengano elaborate correttamente e che l'uscita dal ciclo di controllo non crei problemi nel caso limite in cui la WZ ricercata è l'ultima disponibile.

4.4 Working Zone adiacenti

Sono state predisposte due WZ adiacenti e sono state effettuate due codifiche successive dove gli indirizzi da codificare sono l'ultimo della prima WZ ed il primo della seconda WZ.

L'obiettivo è assicurarsi che il `WZ_NUM` e il `WZ_OFFSET` vengano aggiornati correttamente a seguito del cambio di WZ.

4.5 Reset sincrono ed asincrono

Oltre ai casi in cui il modulo esegue una codifica completa, sono stati creati dei testbench per verificare il corretto comportamento in caso di reset sincrono ed asincrono *(fig. 7)* e quindi controllare che la macchina ritornasse correttamente allo stato `IDLE` resettando tutti i segnali ed i registri.

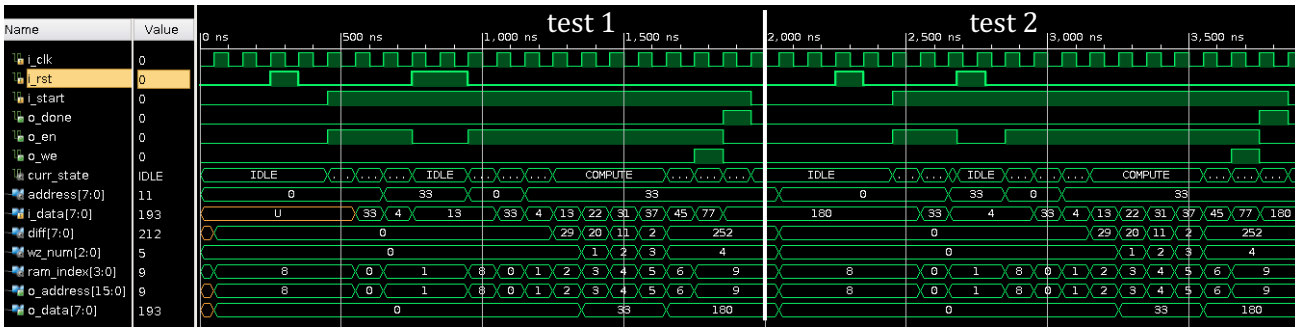

Fig.7: testbench con reset sincrono (test 1) ed asincrono (test 2)

5. Conclusioni

Oltre a verificare il corretto comportamento nelle simulazioni Behavioral, Post-Synthesis Functional e Post-Synthesis Timing, è stato stabilito che il modulo si comporta correttamente anche in entrambe le simulazioni Post-Implementation Functional e Timing.

Inoltre, è stato controllato il percorso critico del componente per quantificare il margine di tempo rispetto al periodo di clock.
Il report sul Timing generato da Vivado rispetto al constraint sul periodo di clock, impostato a 100 ns da specifica, indica che il margine minimo (quantificato dal parametro Worst Negative Slack) tra il tempo di setup del percorso critico ed il periodo di clock risulta essere di 97 ns.
Questo valore conferma che le ottimizzazioni sul tempo di esecuzione hanno dato i risultati ipotizzati poiché il percorso critico risulta essere di 3 ns.

Come prova finale, il modulo è stato simulato in Post-Synthesis Timing e Post-Implementation Timing utilizzando periodo di clock di 5 ns superando correttamente entrambi i test.