Prova Finale di Reti Logiche

Truong Kien Tuong

May 1st, 2020

Matricola:	887907
Codice Persona:	10582491
Docente:	Gianluca Palermo

1 Requisiti del Progetto

Il progetto richiesto consiste nell'implementazione in VHDL del metodo di codifica basato sulle **"Working Zones"** (Abbr. WZ).
Dato un indirizzo di memoria su 7 bit (8 di cui il primo è settato a zero) e l'indirizzo di 8 blocchi di memoria da 4 celle, detti "Working Zones", viene richiesto al componente di:

1. Accedere ad una memoria RAM per recuperare i dati sopra descritti
2. Verificare se l'indirizzo di memoria appartiene ad una delle working zones
   (a) Se sì, codificare l'indirizzo settando il primo bit a 1, i successivi 3 bit al numero della WZ codificato in binario e gli ultimi 4 bit alla codifica one-hot della posizione dell'indirizzo nella WZ
   (b) Altrimenti, restituire l'indirizzo come è stato fornito, con il primo bit a 0.
3. Scrivere il risultato nella memoria RAM

Inoltre, l'implementazione deve essere in grado di gestire un segnale di Reset. Per l'implementazione si è scelto di supporre il Reset asincrono rispetto al segnale di clock.
L'implementazione deve essere poi sintetizzata con target FPGA xc7a200tfbg484-1.

1.1 Esempio

Si riporta in seguito un esempio di codifica con il metodo delle Working Zones. Nella figura è riportata una memoria con indirizzamento a 8 bit (Gli indirizzi sono rappresentati nella memoria tra parentesi quadre) con una working zone (WZ5) con indirizzo di partenza 00101000.

Gli indirizzi contenuti nella working zone riportano a lato il relativo offset rispetto all'indirizzo di partenza tra parentesi tonde.
Nella figura è illustrato un esempio in cui l'indirizzo è effettivamente contenuto in una WZ, in modo da esplicitare come deve essere effettuata la codifica.

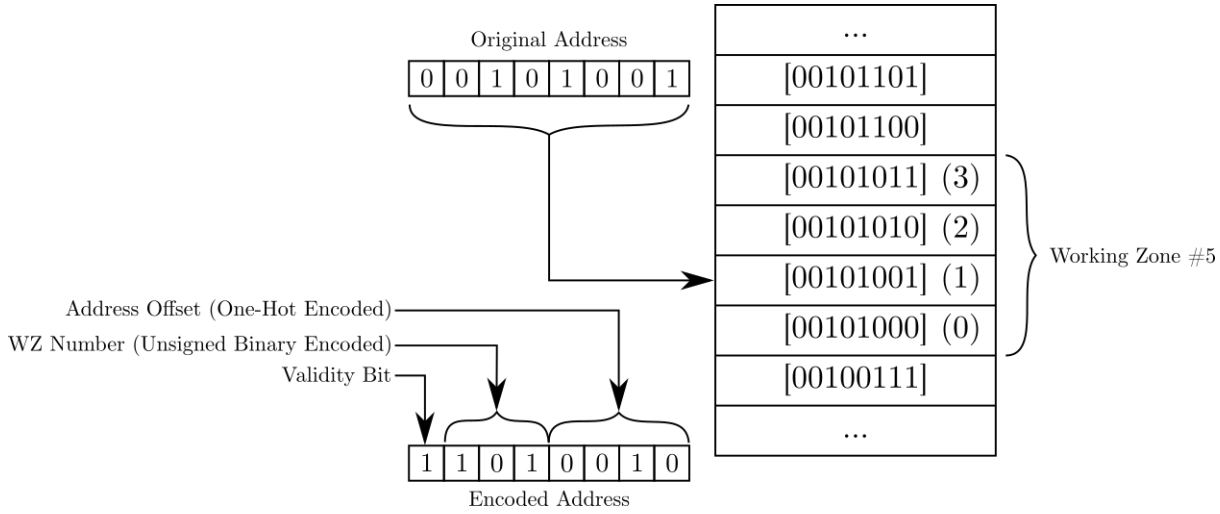

Figure 1: Esempio di codifica di indirizzo contenuto in una Working Zone

1.2 Ipotesi Progettuali

Si sono supposti veri i seguenti fatti:

1. Le WZ non possono mai sovrapporsi (i.e. Non possono esistere due WZ tali che l'indirizzo di una sia all'interno dell'altra WZ). Si è scelto di supporre ciò per evitare ambiguità nella codifica dell'indirizzo.

2. Non possono esistere WZ malformate (i.e Non può esistere una WZ che, nel range di indirizzi da 0 a 127, abbia meno di 4 celle).

2 Implementazione

L'architettura è stata progettata in maniera modulare, in modo da specializzare i singoli componenti creati e separare le funzionalità di calcolo della codifica dell'indirizzo dalla gestione della macchina a stati finiti.

2.1 Descrizione ad Alto Livello

Da un'ottica di alto livello, l'implementazione esegue i seguenti passi:

1. Carica l'indirizzo da codificare dalla RAM in una memoria interna
2. Se le Working Zones non sono ancora state tutte caricate:
   (a) Carica l'indirizzo della WZ dalla RAM
   (b) Verifica se il nuovo indirizzo della WZ contiene l'indirizzo da codificare
   (c) Se sì, setta il risultato all'indirizzo codificato e salta a 4
   (d) Altrimenti torna a 2
3. Controlla se dal caricamento dell'ultima WZ è stato ottenuto un risultato
   (a) Se sì, setta il risultato a tale indirizzo codificato
   (b) Altrimenti, setta il risultato all'indirizzo originale
4. Scrive il risultato nella RAM
5. Se è richiesto un reset, torna a 1, altrimenti mantiene tutte le WZ e richiede solamente l'indirizzo
6. Carica il nuovo indirizzo da codificare
7. Carica il risultato nella RAM e torna al punto 5

Per gestire questo algoritmo si è scelta un'implementazione costituita da 3 tipi di entità: una macchina a stati finiti, che rappresenta il top-level component e che gestisce il salvataggio nei registri degli stati e l'esecuzione delle transizioni, un Address Offset Units Controller e 8 Address Offset Units. Segue una descrizione dettagliata di tutte le entità riportate.

2.2 Address Offset Unit

Il componente denominato **Address Offset Unit** (AOU) è un circuito combinatorio che, ricevuto un indirizzo di WZ `i_wzaddress` e l'indirizzo da codificare `i_address` verifica se l'indirizzo da codificare sia contenuto nella WZ.
Nel caso in cui la risposta sia positiva, calcola anche l'indirizzo codificato `o_address`, altrimenti restituisce un indirizzo interamente settato a 0.

La struttura modulare di tali componenti garantisce una facile estensione dell'implementazione, se ciò risultasse necessario. Inoltre, poichè verranno istanziate 8 AOU uguali (eccetto per il numero di WZ), si ottiene un'architettura simile ad un array sistolico, in grado di effettuare la computazione su tutte le 8 WZ contemporaneamente e sfruttando così il parallelismo hardware fornito dall'FPGA.

Infine, un'architettura di questo genere ben si adatta al problema, che richiede una computazione di tipo SIMD (Single Instruction Multiple Data), con un

layer di reduce dato dal Controller.

L'architettura dell'AOU è stata implementata con una specifica behavioural

2.3 Address Offset Units Controller

L'**Address Offset Units Controller** (AOU Controller) è un circuito combinatorio che interfaccia la macchina a stati finiti con le 8 AOU, istanziando quest'ultime mediante un'istruzione generate in VHDL. A ciascuna AOU è assegnato un numero di WZ mediante generic mapping. Il process del Controller verifica per ciascun AOU se:

1. È stato ottenuto un risultato valido dall'AOU (Ovvero l'indirizzo da codificare appartiene alla WZ assegnata)
2. La WZ assegnata è valida (Ovvero è aggiornata allo stato attuale dei dati contenuti nella RAM)

La seconda condizione è necessaria per consentire l'esistenza di una WZ che inizia all'indirizzo 0. Se non si tenesse conto della validità della WZ, subito dopo il reset si avrebbe che tutte le AOU possiedono un risultato valido (Dato dall'indirizzo da codificare `0x00` che è banalmente contenuto in una WZ che inizia in `0x00`).
Non appena il Controller ha trovato un risultato valido, questo viene asserito come output. Se non viene trovato alcun risultato valido dalle AOU, viene asserito l'indirizzo in ingresso come output.
L'architettura dell'AOU Controller è stata implementata con una specifica in parte strutturale, data dai collegamenti con le AOU, e in parte comportamentale, nella gestione dei risultati delle AOU.

2.4 Macchina a Stati Finiti

La **Macchina a Stati Finiti** (FSM) è il circuito che si occupa di gestire lo stato della computazione e che si interfaccia con la RAM per il trasferimento dei dati. La FSM è stata realizzata con specifica behavioural mediante due processi: `STATE_OUTPUT` e `DELTA_LAMBDA`.
Il processo `STATE_OUTPUT` è il processo sequenziale che ha il compito di asserire le uscite e di cambiare lo stato interno sul fronte di salita del clock. Il processo `DELTA_LAMBDA` è invece il processo combinatorio che calcola le uscite e lo stato validi per il prossimo fronte di salita.
Segue una descrizione degli stati formali della FSM:

- `IDLING`: Stato di idle in cui si posiziona la FSM al reset della computazione. La macchina aspetta che venga asserito il segnale `i_start`.
- `SET_READ_ADDR`: Stato in cui vengono preparate le uscite per leggere l'indirizzo da codificare dalla RAM

- `WAIT_READ_ADDR`: Stato in cui si aspetta un ciclo di clock per consentire alla RAM di asserire le sue uscite
- `FETCH_DATA_ADDR`: Stato in cui si legge il dato dalla RAM e viene salvato l'indirizzo da codificare
- `SET_READ_WZ`: Stato in cui vengono preparate le uscite per leggere l'indirizzo della i-esima WZ
- `WAIT_READ_WZ`: Stato in cui si aspetta un ciclo di clock per consentire alla RAM di asserire le sue uscite
- `FETCH_DATA_WZ`: Stato in cui si legge il dato dalla RAM e viene salvato l'indirizzo della i-esima WZ
- `WAIT_RESULT`: Stato in cui, dopo il caricamento di tutte le WZ, si aspetta un ciclo di clock per consentire la propagazione dei dati nell'AOU Controller
- `SET_WRITE`: Stato in cui vengono preparate le uscite per scrivere il risultato nella RAM e si asserisce `o_done` a 1
- `END_WRITE`: Stato in cui si aspetta che `i_start` venga portato giù, settando `o_done` a 0 e riportandosi allo stato di `IDLING`.

Più precisamente l'insieme degli stati della FSM dovrebbe comprendere almeno anche il counter `CURR_WZ`, che assume come valore il numero della WZ che deve essere caricata. Così facendo gli stati `SET_READ_WZ`, `WAIT_READ_WZ` e `FETCH_DATA_WZ` sarebbero, dunque, da replicare per tutte le 8 WZ.

Nella figura a seguito è riportato il disegno dell'automa. Sono state usate le seguenti abbreviazioni degli stati per chiarezza:

`IDLING`:	`IDLE`
`SET_READ_ADDR`:	`SR_ADDR`
`WAIT_READ_ADDR`:	`WR_ADDR`
`FETCH_DATA_ADDR`:	`FD_ADDR`
`SET_READ_WZ` (With `WZ_COUNT=i`):	`SR_WZ_i`
`WAIT_READ_WZ` (With `WZ_COUNT=i`):	`WR_WZ_i`
`FETCH_DATA_WZ` (With `WZ_COUNT=i`):	`FD_WZ_i`
`WAIT_RESULT`:	`WAIT_RES`
`SET_WRITE`:	`SET_WR`
`END_WRITE`:	`END_WR`

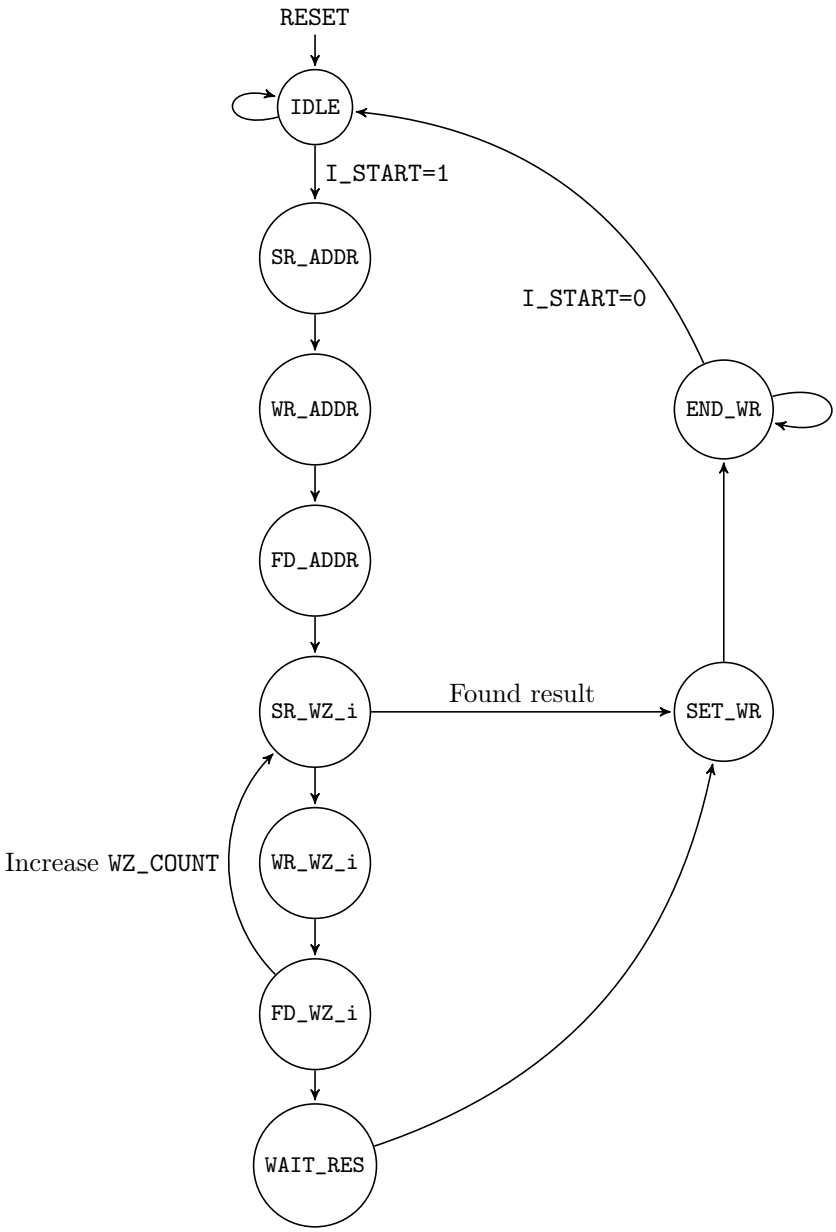

Figure 2: Diagramma degli stati della Macchina a Stati Finiti

Si noti come il fatto di poter dare il risultato anche durante la fase di setup nasconde parte del tempo impiegato per caricare le WZ, riuscendo a fornire il risultato non appena si hanno i dati per poterlo calcolare.

2.5 Schema dell'Implementazione

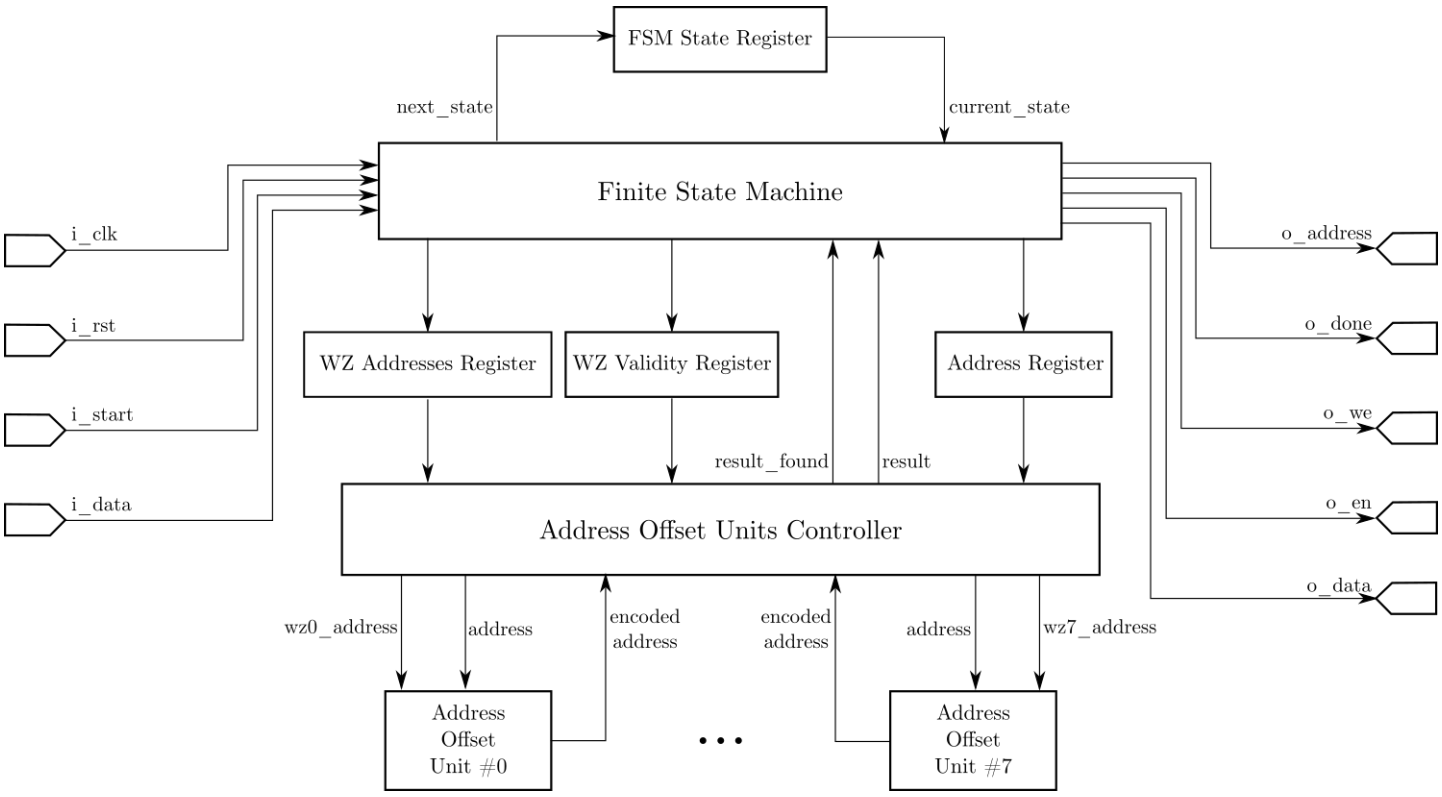

Figure 3: Schema ad alto livello dell'implementazione

3 Test Benches

I test effettuati hanno cercato di effettuare transizioni critiche oppure di verificare possibili configurazioni di memoria estreme. In particolare sono stati creati dei testbench comprendenti le seguenti situazioni:

- Segnale di `RESET` dopo che la macchina ha asserito il segnale di `o_done` ma prima del segnale di `i_start`

- Ultima WZ all'indirizzo `0x00` e indirizzo da codificare `0x00`, test particolarmente importante per le ragioni fornite prima sul funzionamento dell'AOU Controller.
- Ultima WZ all'indirizzo `0x7C` (Ovvero `0b01111100`, l'ultimo indirizzo possibile per una WZ) e indirizzo da codificare `0x7F` (Ovvero `0b01111111`)
- Altri test bench vari generati casualmente, mantenendo la validità della specifica
- 10000 indirizzi da codificare con sempre le stesse WZ e senza mai effettuare reset (`KEEP WZs`)
- Gli stessi 10000 indirizzi da codificare e con sempre le stesse WZ ma con un reset dopo ogni indirizzo (`RESET WZs`)

Questi test hanno evidenziato alcune criticità nel codice iniziale e hanno guidato lo sviluppo della soluzione fino alla sua versione finale.

Per tutti i test riportati e i test successivi è stata effettuata la simulazione behavioural e successivamente la simulazione functional e timing post-synthesis, tutte con successo.

I risultati rilevanti sono riportati nella sezione 4.

4 Risultati Sperimentali

4.1 Report di Sintesi

Dal punto di vista dell'area la sintesi riporta il seguente utilizzo dei componenti:

- LUT: 197 (0.15% del totale)
- FF: 108 (0.04% del totale)

Si è fatta particolare cautela nella scrittura del codice per evitare utilizzo di Latch.

4.2 Risultato dei Test Bench

Lavorando sul testing si è anche provato a variare il periodo di clock, confermando che, non solo l'implementazione rispetta le specifiche, funzionando a 100 ns, ma mantiene il suo funzionamento anche a 1 ns nella simulazione Behavioural e 7 ns nella simulazione Functional Post-Synthesis.

Si riportano i risultati di tempo legati agli ultimi 2 test riportati nella sezione 3

Simulazione KEEP WZs (Functional Post-Synthesis):	2 020 868 600 ps
Simulazione RESET WZs (Functional Post-Synthesis):	2 754 167 600 ps

Il risultato esprime una variazione del 36% dal caso senza reset al caso con reset, evidenziando come l'architettura, per scelta progettuale, è molto più efficiente quando non deve fare frequentemente la fase di setup, ossia la fase di caricamento delle WZ.

5 Conclusioni

Si ritiene che l'architettura progettata rispetti anzitutto le specifiche, fatto che è stato verificato mediante estensivo testing sia casuale, che con test benches manualmente scritti. Oltre a ciò l'architettura è stata pensata cercando di sfruttare al meglio il contesto in cui viene effettuata la computazione, ovvero quella di parallelizzare il più possibile il lavoro effettuato dal componente.

Dal punto di vista del design, scegliere un'architettura modulare e facilmente espandibile è un vantaggio, anche nell'ottica di un'implementazione System on Chip, in quanto consente di riutilizzare più volte lo stesso identico componente.