Documentazione Progetto Reti Logiche

Andrea Franchini
Cod. Persona: 10560276

March 2020

Sommario

Descrizione e sintesi in VHDL di un componente che implementa la codifica *Working Zone* di indirizzi letti da una RAM esterna, in cui vengono anche scritti i risultati. Realizzazione mediante una FSM sincrona di Mealy in 4 stati (*Idle*, richiesta alla RAM, lettura dalla RAM, codifica). Il componente viene testato in pre-sintesi e in post-sintesi mediante casi di test incentrati sui casi limite.

1 Introduzione

La specifica richiede che il componente legga dalla RAM gli indirizzi da codificare, è quindi necessario implementare una macchina sincrona a stati finiti. In particolare la lettura dalla RAM è da effettuarsi in due passi, mentre la scrittura è possibile in un un unico passo.

Nel caso trattato, la porzione di RAM utilizzata è composta dalle prime 10 celle: le prime 8 (dalla cella 0 alla cella 7) contengono gli indirizzi iniziale di una *working zone*, l'ottava contiene l'indirizzo da codificare e nella nona verrà scritto l'indirizzo codificato.

I valori contenuti nella RAM sono di lunghezza 8 bit, ma solo 7 sono effettivamente utilizzati.

Il componente impiega dunque un tempo variabile per finire il suo compito, poichè è necessario scorrere le celle 0-7 della RAM (almeno parzialmente) per verificare che l'indirizzo da codificare appartenga ad almeno una di esse.

Il modulo termina il proprio processo quando il segnale in uscita `o_done` viene portato alto, ma bisogna assicurarsi che venga riportato a 0 una volta che `i_start` torna basso, in modo da poter eventualmente iniziare nuovamente l'operazione di codifica. Sarà quindi necessario gestire opportunamente lo stato di reset.

Lo strumento di sintesi utilizzato è Xilinx Vivado WEBPack 2018.3, e la FPGA target il modello *xc7a200tfbg484-1*.

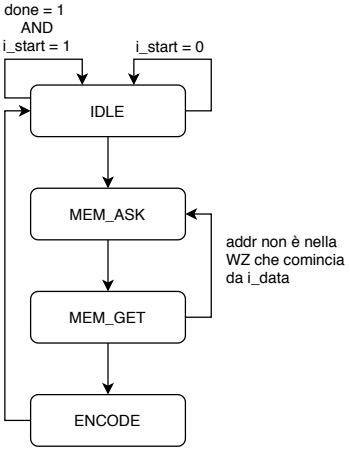

Figura 1: FSM con transizioni

2 Architettura

2.1 Moduli

L'unità si compone di un unico modulo che contiene i processi necessari per la macchina a stati finiti ed i registri.

Il primo processo (`state_sync`) gestisce l'aggiornamento dei flip-flop in maniera sincrona (sulla *rising edge*) mentre l'attuazione del *reset* è asincrona.

Il secondo processo contiene la parte combinatoria e implementa le i comportamenti della FSM.

Sono stati impiegati in fase di progettazione i seguenti registri, oltre a quello per lo stato:

- `done` tiene traccia di quando la codifica è terminata;
- `get_addr` tiene traccia se dalla RAM viene ricevuto l'indirizzo da codificare o di una *working zone*;
- `addr` (di dimensione 8 bit) salva l'indirizzo da codificare letto dalla RAM;
- `wz_bit` indica se l'indirizzo codificato appartiene a una *working zone* (1) o meno (0);
- `wz_num` (di dimensione 3 bit) indica l'eventuale *working zone* a cui appartiene address ma è anche un contatore utilizzato per leggere i valori dalla RAM;
- `wz_offset` (di dimensione 4 bit) indica l'offset di `addr` se esso appartiene a una *working zone*.

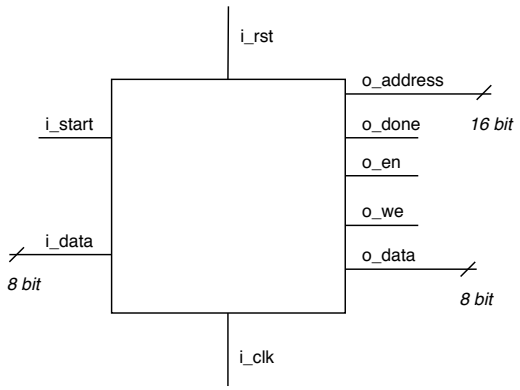

Figura 2: Modulo implementato

2.2 Macchina a stati finiti

La macchina a stati finiti può essere implementata con soli quattro stati. Prendendo come riferimento il codice VHDL allegato, i quattro stati sono: *IDLE*, *MEM_ ASK*, *MEM_ GET*, *ENCODE*.

2.2.1 Diagramma

Si veda la figura 6 alla fine.

IDLE Lo stato *IDLE* coincide con lo stato di reset: la macchina parte da questo stato e vi ritorna una volta finita la fase di codifica, quindi è opportuno che riporti i registri al loro valore di default, in modo da poter cominciare una nuova codifica quando il segnale `i_start` viene portato alto. Una volta finita la codifica, si rimane in questo stato finchè il segnale di start non viene riportato a basso (a quel punto si riporta a basso anche `o_done`).

MEM_ ASK Imposta le uscite in modo da poter leggere dalla memoria (`o_en` alto, `o_address` l'indirizzo da leggere). Concatenando i valori di un flip-flop (`get_addr`) e un contatore da 3 bit (`wz_num`) è possibile scorrere la memoria scrivendo sugli ultimi 4 bit di `o_address`. La sequenza completa degli indirizzi che possono venire letti è quindi:

$$8 \to 0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7$$

In particolare il quarto bit meno significativo, cioè `get_addr`, quando è a '1' fa sì che il primo indirizzo richiesto alla RAM sia 0000 0...0 1000, cioè la cella in cui è contenuto l'indirizzo da codificare, oltre a tener traccia se è la prima volta che si arriva in *MEM_ ASK* durante una fase di codifica. In VHDL:

```
o_address(3) <= q_get_addr;
o_address(2 downto 0) <= q_wz_num;
```

MEM_ GET In questa fase si riceve e processa il valore ricevuto dalla RAM su `o_address`. Controllando il registro `get_addr` si stabilisce se è stato letto l'indirizzo da codificare (`get_addr = '1'`) o di una *working zone* (`get_addr = '0'`).

Nel primo caso si salva in un registro (`addr`) l'indirizzo da codificare e si impone `get_addr` = '0' per poter chiedere e ricevere dalla RAM il valore della prima *working zone* all'iterazione successiva. Il contatore `wz_bit` non viene incrementato e resta nella configurazione iniziale (`000`).

Nel secondo caso, si procede nel seguente modo:

1. Si riceve su `i_data` l'indirizzo della *working zone* da classificare e dal flip flop `addr` l'indirizzo da codificare;
2. Si prende l'offset $d =$ `addr` $-$ `i_data` come differenza di interi di tipo `signed`;
3. Si stabilisce se $0 \leq d \leq 3$:
   - Se l'espressione è vera allora `addr` appartiene a una *working zone*, si imposta il flip flop `wz_bit = '1'` e si esprime in codifica *one-hot* l'offset d;
   - Se l'espressione è falsa `wz_bit` rimane a zero, e se il contatore `wz_num` non è ancora arrivato al suo valore massimo (8, in bit `111`) si incrementa il contatore di un bit e si torna allo stato *MEM_ ASK*, altrimenti si può affermare con certezza che `addr` non appartenga ad alcuna delle otto *working zones* possibili.

ENCODE In quest'ultimo stato si procede a scrivere in memoria RAM l'indirizzo codificato come da specifica. Controllando `wz_bit` si stabilisce se l'indirizzo è da codificare come appartente a una *working zone* o meno.

Nel caso appartiene a una *working zone*

```
o_data <= wz_bit & wz_num & wz_offset
```

`wz_bit` è 1 poichè `addr` appartiene alla *working zone* `wz_num` (codificato in binario naturale, 3 bit) con offset `wz_offset` (codificato in one-hot).

Nel caso non appartenga a una *working zone*

```
o_data <= wz_bit & addr(6 downto 0)
```

`wz_bit` è 0 poichè `addr` non appartiene alla *working zone*, inoltre scegliamo di ignorare il bit più significativo di `addr`, che è irrilevante per specifica.

`o_data` viene scritto poi nella nona cella della RAM e il segnale `o_done` viene portato ad alto.

3 Risultati sperimentali

La rete è stata sintetizzata con un periodo di clock minimo di 100ns, come da specifica.

Post-sintesi sono stati utilizzati 20 flip-flops (FF), 31 look-up tables (LUT) e 38 pin tra input e output. Poichè non è stato esplicitato in VHDL, il registro dello stato utilizza 2 FF e la codifica naturale.

I timings sono accettabili: il WNS equivale a 95.437ns (Worst Negative Slack), è positivo e quindi accettabile, così come il WHS (Worst Hold Slack).

La potenza stimata con parametri di default è di 0.131 W.

Sono presenti warning relativi a fasi successive alla sintesi.

4 Simulazioni

Dai test effettuati il modulo funziona correttamente sia in pre-sintesi che in post-sintesi.

Dopo il superamento del test-bench fornito e di diversi test generati in modo casuale con uno script Python[1] sono stati analizzati i seguenti casi limite, utilizzando come base di partenza il test-bench fornito. Il periodo di clock utilizzato è quello minimo, cioè 100 ns.

4.1 Indirizzo non appartiene a WZ

In questo caso di test (che è praticamente identico a quello fornito) si verifica che l'indirizzo da codificare (quello in `RAM(8)`) venga trasmesso così com'è, salvo per il primo bit che deve essere 0. Inoltre, questa codifica è il caso pessimo, perchè richiede di scorrere tutte le celle della RAM contenenti le WZ, impiegando così 2150 ns da quando il segnale di start è portato alto.

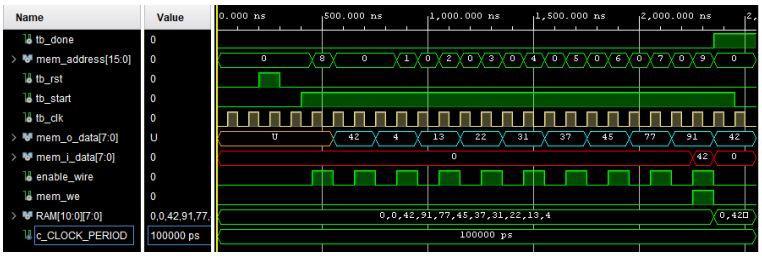

Figura 3: In rosso, la memoria scritta dal modulo, in azzurro, la memoria letta. 42 è il valore dell'indirizzo da codificare.

[1] https://github.com/Mark-Zampedroni/Multi-TB-progetto-Reti

4.2 Indirizzo appartiene a WZ

In questo caso di test si verifica che l'indirizzo da codificare venga codificato come working zone. Conviene testare ai limiti della RAM, ovvero un sotto-caso in cui l'indirizzo appartiene alla prima WZ (la numero 0) e uno in cui appariente all'ultima (la numero 7), testando con offset differenti.

Inoltre, possiamo osservare il caso ottimo, che richiede di scorrere solo la prima cella della RAM, impiegando così 748 ns da quando il segnale di start è portato alto.

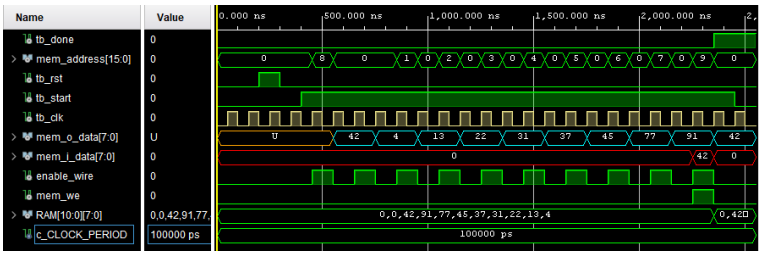

Figura 4: In rosso, la memoria scritta dal modulo, in azzurro, la memoria letta. 111 è il valore dell'indirizzo da codificare.

4.3 Reset durante codifica

Conviene testare che l'unità venga resettata correttamente se durante la codifica viene alzato il segnare di reset. Nella figura, il segnale di start viene tenuto alto, conseguentemente la codifica ricomincia non appena il segnale di reset torna basso.

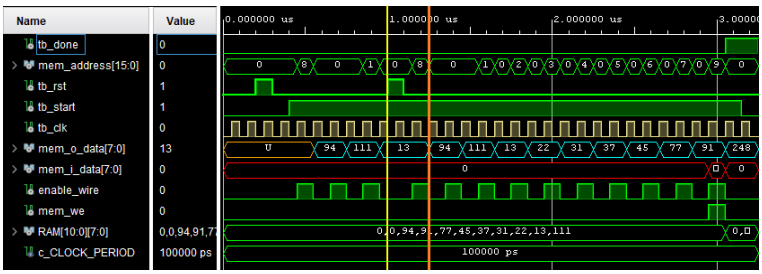

Figura 5: In rosso, la memoria scritta dal modulo, in azzurro, la memoria letta, in giallo, l'istante in cui il segnale di reset è portato alto, in arancione, l'istante in cui viene letta nuovamente RAM(8). 94 è il valore dell'indirizzo da codificare, e appartiene alla WZ 7.

4.4 Codifiche successive

Conviene testare anche che l'unità possa codificare nuovamente dopo essere tornata allo stato di reset dopo una computazione, sempre che vengano rispettate le condizioni sul segnale di `start` e di `done`. Non viene forzato il reset.

5 Conclusioni

Il modulo sembra funzionare correttamente, e credo che sia molto ottimizzato. Dopo diverse iterazioni sono riuscito a ridurre il numero di stati a quattro, mentre iterazioni precedenti funzionavano con 5 o più stati.

Il tempo massimo di codifica nel caso pessimo mi sembra accettabile, dato il periodo minimo di clock di 100 ns.

Poichè la codifica in working zones è nata per una questione di mantere una potenza dissipata ridotta, mi è sembrato anche opportuno ridurre il più possibile il numero di registri.

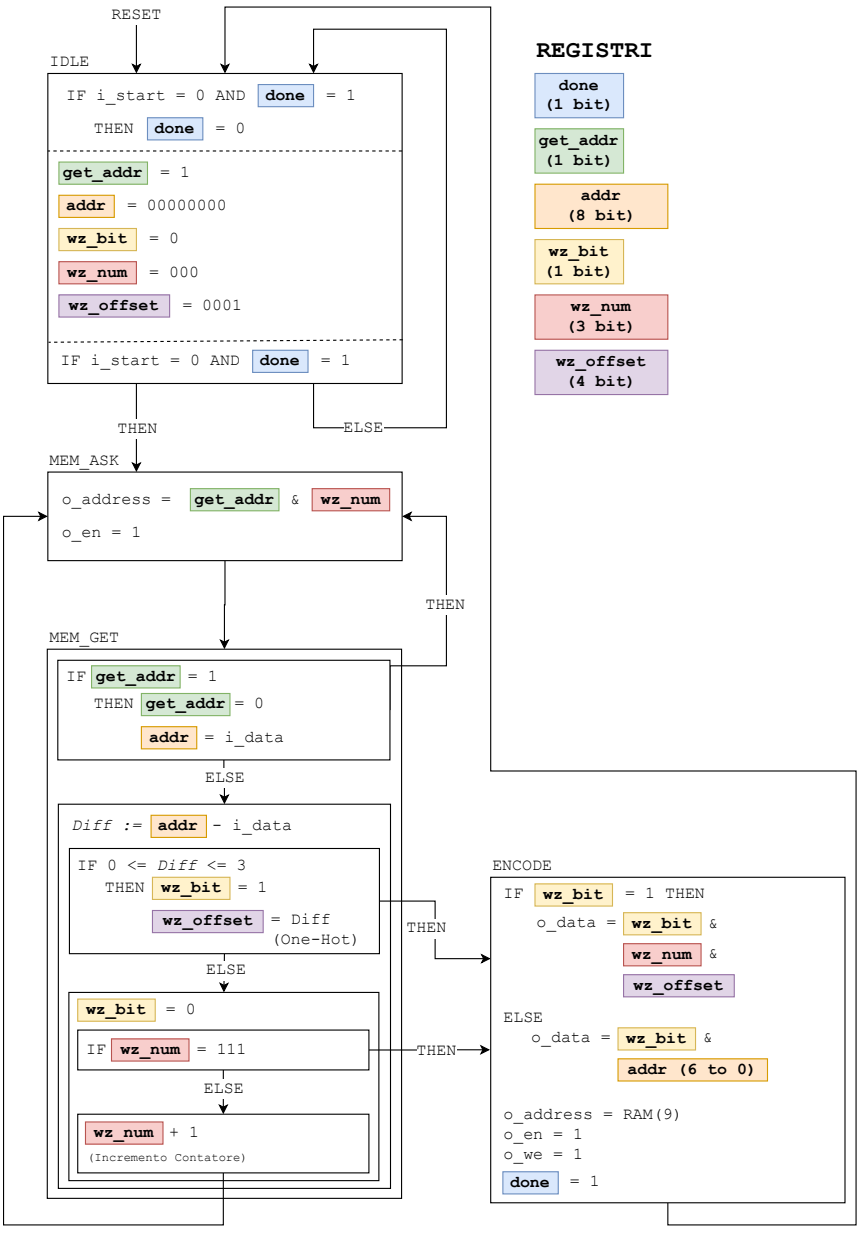

Figura 6: Diagramma degli stati con flip-flop (colorati).