Prova Finale di Reti Logiche

Politecnico di Milano

Lorenzo Gadolini,
Giuseppe Lischio

March 20, 2020

Contents

1 Introduzione

Il metodo di codifica a working-zone propone un'ottimizzazione orientata alla riduzione del consumo di energia introdotto dall'input/output di un microprocessore.

Sia dato un generico sistema composto da un processore e una memoria esterna al chip referenziabile tramite un bus indirizzi. Questo metodo suggerisce che un programma in esecuzione sul dato sistema sfrutti durante la sua esecuzione un insieme di indirizzi "preferiti", e che quindi sia più efficiente racchiudere tutti questi indirizzi dentro uno spazio di lavoro (detto appunto Working-Zones). Gli indirizzi di queste Working-Zones sono codificati tramite un sistema base e offset, così da ridurre ulteriormente la quantità di informazione trasmessa sul bus indirizzi, e di conseguenza ridurre anche l'energia dissipata.

1.1 Dati Progettuali e Specifica

Il componente da progettare ha come compito quello di stabilire se un indirizzo che riceve in input appartiene o meno ad una delle Working-Zones stabilite, e in caso affermativo di effettuare la traduzione dell'indirizzo da binario naturale a codifica WZE.

Il componente si interfaccia con una memoria indirizzabile al byte a partire dall'indirizzo 0, e in cui vengono inizializzati i dati necessari per effettuare la computazione. La memoria inoltre è indirizzabile tramite indirizzi di 16 bit.

Gli indirizzi di memoria da 0 a 7 contengono le basi delle 8 Working-Zones stabilite in fase di setup e scritte in binario naturale senza segno su 7 bit. La cella di memoria 8 contiene l'indirizzo su cui effettuare la verifica e l'eventuale codifica, anch'esso scritto in binario naturale senza segno su 7 bit. L'indirizzo di memoria 9 conterrà a fine computazione un numero di 8 bit senza segno, che come verrà descritto più avanti potrà essere una codifica o un indirizzo semplice.

I restanti indirizzi sono tutti inizializzati a 0 e non vengono sfruttati durante la computazione.

1.2 La fase di traduzione

Nella fase di codifica il componente deve essere in grado di stabilire se un indirizzo depositato nella cella numero 8 della memoria appartiene o meno ad una Working-Zone. Un indirizzo che non appartiene a nessuna Working-Zone fa entrare il componente in uno stato di "Non appartenenza", stato in cui la macchina secondo specifica deve depositare in uscita il dato così come le è stato fornito, preceduto da un bit posto a "0". Il dato in uscita è un intero binario naturale senza segno di 8 bit

```
"0" & "0101010"   --Codifica[1] esempio di indirizzo che non cade in alcuna WZ
```

Se il componente stabilisce che l'indirizzo appartiene ad una Working-Zone, scatta la fase di "Appartenenza", il dato necessita di encoding. Un indirizzo codificato è un numero di 8 bit senza segno composto da tre sottogruppi di bit.

- Il bit più significativo viene posto ad "1", che indica "Codifica avvenuta";
- I tre bit successivi codificano posizionalmente la Working-Zone a cui tale indirizzo appartiene. Il dato in uscita è un numero binario naturale senza segno da 0 a 7 che indica che l'indirizzo appartiene alla "n-esima" Working-Zone.
- I quattro bit finali codificano l'offset dell'indirizzo dalla base. Il numero codificato è un numero binario One Hot. L'offset indica la distanza posizionale in memoria dall'indirizzo base, ovvero l'indirizzo "si trova a" n indirizzi dalla base.

```
"1" & " 010" & " 0100"   --Codifica esempio di indirizzo appartenente ad una WZ
```

Sia che l'indirizzo appartenga o che non appartenga ad una Working-Zone, il componente prosegue sempre lungo la stessa fase di output. Il dato viene depositato in memoria, salvato, e la macchina entra in fase di attesa di reset e avvio per una successiva codifica.

[1] Il simbolo & indica concatenazione di bit

1.2.1 Entity del componente

Il componente hardware è stato progettato tramite linguaggio VHDL in base alle specifiche fornite dal tema di progetto.

La struttura input/output del componente in VHDL è stata fornita nelle specifiche ed è di seguito riportata.

```vhdl
--- ENTITY DEL PROGETTO ---
entity project_reti_logiche is
    Port ( i_clk : in STD_LOGIC;                          --Input clock proveniente dal testbench.
           i_start : in STD_LOGIC;                        --Segnale di avvio della computazione generato dal testbench.
           i_rst : in STD_LOGIC;                          --Segnale di reset generato dal testbench.
           i_data : in STD_LOGIC_VECTOR (7 downto 0);     --Input Byte proveniente dalla memoria esterna.
           o_address : out STD_LOGIC_VECTOR (15 downto 0); --Indirizzo di memoria per cui si richiede lettura/scrittura.
           o_done : out STD_LOGIC;                        --Segnale di terminazione della computazione generato dal componente.
           o_en : out STD_LOGIC;                          --Segnale di ENABLE per poter comunicare con la memoria.
           o_we : out STD_LOGIC;                          --Segnale di abilitazione alla scrittura in memoria.
           o_data : out STD_LOGIC_VECTOR (7 downto 0));   --Dato da scrivere in memoria.
end project_reti_logiche;
```

2 Architettura e Scelte Progettuali

2.1 L'algoritmo

Il progetto si basa su un algoritmo alquanto semplice.

Dato uno stato iniziale di memoria, per prima cosa viene letto il dato di cui si necessita la codifica dall'indirizzo di memoria numero 8, successivamente il componente entra in un ciclo di calcolo. In questo ciclo di calcolo vengono letti uno alla volta i byte di memoria 0-7 contenenti gli indirizzi base delle Working zones. Il componente sottrae l'indirizzo da codificare all'indirizzo base letto, e verifica che il risultato cada fra 0 e 3. Questo risultato ha il significato di offset, e come da specifica, un offset compreso fra 0 e 3 (inclusi) indica un'appartenenza alla working zone. Qualsiasi altro risultato viene interpretato come "Non appartenenza".

2.2 Non Appartenenza

Un risultato che non appartiene a nessuna working zone deve venire propagato in uscita così come è stato letto da memoria in entrata. Il componente ha terminato gli otto cicli di verifica e ha determinato che l'indirizzo in input non appartiene a nessuna working zone. A questo punto la macchina carica i 7 bit dell'indirizzo nel registro di output, il quale era stato inizializzato con 8 bit tutti posti a zero, così da rispettare la specifica che prevede di emettere in uscita uno 0 seguito dai 7 bit dell'indirizzo scritti in binario naturale senza segno.

2.3 Appartenenza

Se il componente durante la sua esecuzione verifica che l'indirizzo da codificare cade all'interno di una delle basi, allora si avvia la procedura di encoding. Per prima cosa viene codificato l'offset. Il risultato della sottrazione appartiene ad un insieme finito e ristretto di valori, per questo si è scelto di creare una lookup table che contenesse le codifiche in onehot da assegnare al valore di uscita a seconda del valore di offset. Una volta scritto in memoria il primo quartetto di bit, il componente passa alla traduzione della tripletta di bit contenente l'indirizzo della working zone. Questa fase scrive i tre bit nel registro di out che corrispondono al valore binario naturale del byte di memoria che contiene l'indirizzo base.

Infine il componente pone ad 1 il MSB del valore di uscita, cosi che chi riceve il valore tradotto a valle riconosce che è stato effettuato l'encoding.

2.4 Il design: La macchina a stati

Il progetto del componente parte dalla descrizione di una macchina a stati che esegua l'algoritmo poco fa descritto

La macchina a stati completa che esegue la codifica è disegnata qui:

2.4.1 Descrizione degli stati

- RESET: Lo stato in cui la macchina viene avviata e a cui giunge una volta che riceve il segnale di reset. La macchina cicla su questo stato attendendo un segnale di start. Successivamente inizializza la memoria e salta al primo vero stato di esecuzione.
- RQST_ ADDR: La macchina richiede alla memoria l'indirizzo da codificare.
- WAIT_ ADDR: La macchina attende per un ciclo di clock la propagazione dei segnali alla memoria.
- READ_ ADDR: La macchina ottiene il dato dalla memoria.
- RQST_ WZ: La macchina richiede alla memoria il valore della n_ esima base, dove n è compreso fra 0 e 7.
- WAIT_ WZ: La macchina attende un ciclo di clock la propagazione dei segnali alla memoria.
- READ_ WZ: La macchina ottiene la base n-esima dalla memoria.
- CMP_ WZ_ ADDR: La macchina esegue la sottrazione che determina l'appartenenza o meno e l'offset del dato da codificare.
- ONEHOT_ ENCODE: La macchina codifica l'offset di un indirizzo che ha dato esito positivo in una comparazione.
- WZ_ FOUND: La macchina prepara il byte contenente il dato codificato e notifica alla memoria che è pronto per essere scritto
- WZ_ NOT_ FOUND: La macchina notifica alla memoria che il dato non appartiene a nessuna Working Zone, e che è pronta a propagarlo senza codifica.
- MEM_ WRITE _ WAIT: La macchina attende un clock che la memoria riceva la sua richiesta di scrittura
- MEM_ WRITE_ DONE: La macchina completa la fase di scrittura.
- DONE: La macchina notifica il raggiungimento dello stato di terminazione.
- FSM_ CLOSE: La macchina entra in attesa di una richiesta di reset.

2.5 Il Codice VHDL

Fino a questo momento non è stata data nessuna descrizione fisica del componente al di fuori delle specifiche che descrivono il problema. In questo piccolo paragrafo viene fatta un'analisi di come è stata descritta la macchina a stati in linguaggio VHDL, analizzandone i segnali e i registri necessari al funzionamento.

Il componente è stato descritto tramite codice VHDL Behavioral che implementa tutti gli stati descritti poco prima.

Oltre all'architettura fornita dalla specifica, il componente possiede altri segnali e registri che si sono resi necessari in fase di scrittura di codice.

- Prima variabile

3 Risultati Sperimentali

3.1 Risultati della sintesi

4 Simulazioni

In questo paragrafo verranno discussi i metodi utilizzati per verificare il corretto funzionamento del componente progettato. In particolare è stato usato il software Xilinx "Vivado" per scrivere i test bench e per effettuare vari tipi di simulazione:

- Simulazione **Behavioral**: test del funzionamento logico del componente, basato sul codice VHDL scritto, senza passare per la sintesi virtuale del componente.
- Simulazione **Post-Synthesis Functional**: test che permette di verificare il comportamento strutturale del componente descritto. Viene eseguito dopo aver terminato il processo di sintesi virtuale
- Simulazione **Post-Synthesis Timing**: test analogo al *Post-Synthesis Functional*, che tiene però conto di tutti i ritardi introdotti dalle porte logiche usate nella simulazione del componente virtuale.

5 Conclusioni