Progetto di Reti Logiche

Prof. Fabio Salice – Anno 2019/20

Francesco Leone (Codice persona 1057235)

Indice:

Introduzione

Lo scopo del progetto è sintetizzare in VHDL un componente hardware in grado di codificare degli indirizzi di memoria secondo il metodo delle ***Working Zone.***

In un dato spazio di indirizzamento a 7 bit vengono definiti otto gruppi di quattro indirizzi contigui di particolare interesse, detti appunto Working Zone (WZ da qui in avanti). Preso in input un indirizzo ADDR qualsiasi, il componente restituisce un output diverso a seconda dei casi:

1) ADDR non fa parte di alcuna WZ --> l'output è (0 & ADDR);
2) ADDR fa parte di una WZ --> l'output è (1 & WZ_NUM & WZ_OFFSET)

Dove & rappresenta l'operazione di concatenamento di bit, WZ_NUM l'etichetta (in codifica binaria da 0 a 7) della WZ e WZ_OFFSET l'offset di ADDR rispetto all'indirizzo base della WZ (in codifica one-hot).

Gli indirizzi base delle WZ e ADDR vengono prelevati dalle prime nove celle di memoria di una RAM mentre la codifica finale di ADDR verrà scritta nella decima cella di memoria della medesima RAM. La RAM è già data da specifica e ogni sua cella di memoria è formata da 8 bit, quindi l'indirizzo dato in input al componente, così come gli indirizzi base delle WZ, è già codificato come (0 & ADDR).

Di seguito viene riportato un esempio di come dovrebbe avvenire la codifica.

CASO 1 CON VALORE NON PRESENTE IN NESSUNA WORKING-ZONE

Indirizzo Memoria	**Valore**	**Commento**
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	42	// ADDR da codificare
9	42	// Valore codificato con in OUTPUT (0 - 0101010)

CASO 2 CON VALORE PRESENTE IN UNA WORKING-ZONE

Indirizzo Memoria	**Valore**	**Commento**
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	33	// ADDR da codificare
9	180	// Valore codificato con in OUTPUT (1 - 011 - 0100)

L'interfaccia del componente è la seguente:

```
entity project_reti_logiche is
    port (
        i_clk       : in std_logic;                      [segnale di clock]
        i_start     : in std_logic;                      [segnale di start – inizio codifica]
        i_rst       : in std_logic;                      [segnale di reset]
        i_data      : in std_logic_vector(7 downto 0);   [input dalla RAM – ADDR da codificare]
        o_address   : out std_logic_vector(15 downto 0); [indirizzo RAM in cui leggere/scrivere]
        o_done      : out std_logic;                     [segnale di done – fine codifica]
        o_en        : out std_logic;                     [abilita l'accesso alla RAM]
        o_we        : out std_logic;                     [abilita la scrittura nella RAM]
        o_data      : out std_logic_vector (7 downto 0)  [output per la RAM – ADDR codificato]
    );
end project_reti_logiche;
```

Algoritmo

L'algoritmo eseguito dal componente è il seguente:

1) **Inizializzazione**: al segnale di *reset*, il componente salva in dei registri interni gli 8 indirizzi base delle WZ, per poi entrare in uno stato di attesa del segnale di *start* (*wait*_start).

2) **Calcolo differenza**: ricevuto il segnale di *start*, il componente calcola la differenza tra ADDR ricevuto in input e gli indirizzi base delle WZ salvate, una alla volta.

3a) **Not_in_WZ**: nel caso in cui la differenza sia >3 (ADDR non fa parte della WZ) per ogni WZ, il componente procede a salvare sulla RAM ADDR così com'è.

3b) **In_WZ**: se per una delle WZ la differenza è <=3 (ADDR fa parte della WZ) il componente procede alla codifica secondo le specifiche e in seguito salva il risultato in RAM.

4) **Attesa**: il componente torna in *wait_start*, pronto per codificare un nuovo ADDR.

Scelte implementative

Per mettere in atto l'algoritmo il componente simula una macchina a stati.

Non essendo imposti limiti temporali, si è deciso di puntare su un componente molto sequenziale, che sfrutta poco il parallelismo, consentendo però di sapere sempre quando viene eseguito un assegnamento ad un determinato segnale: in ogni stato vengono aggiornati determinati segnali oppure viene eseguita una specifica operazione, per un totale di 16 stati.

Si è deciso di creare i registri interni per gli indirizzi base delle WZ perché in questo modo, al costo di 9 cicli di clock per assegnare i valori ai registri, si ottiene di non dover più accedere alla RAM per sapere quali siano gli indirizzi base delle WZ fino al prossimo segnale di *reset*. Questo perché è presumibile che, per un uso reale del componente, le variazioni delle WZ siano rare.

In particolare, sono stati usati tre *process*:

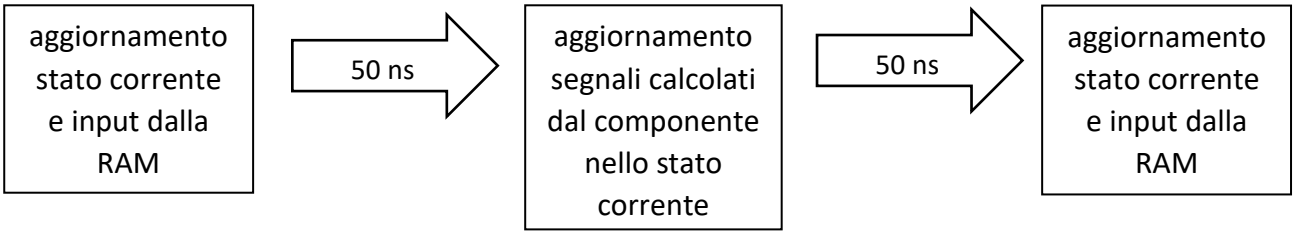
1°: implementa la parte combinatoria del componente;

2°: implementa la parte di memoria, ovvero la codifica dello stato corrente;

3°: codifica la differenza tra ADDR e gli indirizzi base delle WZ da binario a one-hot ("0000" nel caso in cui ADDR-WZ>3).

Il componente è implementato secondo una logica simile a quella *master-slave:*

Il process di gestione dello stato corrente (*master*) è sincronizzato con il process che gestisce l'aggiornamento dei valori di input e output della RAM (che si ripete essere esterno al componente), mentre quello di logica combinatoria (*slave*) lavora nella metà di ciclo di clock successiva. Questo fa sì che, con il periodo di clock assegnato di 100ns, i valori dei segnali usati in ogni stato dalla parte di logica combinatoria siano sempre aggiornati, in quanto il calcolo dei nuovi valori dei segnali e quello del nuovo stato corrente è sfasato di mezzo ciclo di clock, secondo lo schema:



Simulazioni

Sono stati effettuati i seguenti test per verificare il corretto funzionamento del componente, sia in *Behavioral Simulation* che in *Post-Synthesis Functional Simulation*

Codifica ADDR:

1) Test casuali con ADDR appartenente a una WZ: verificano che il componente codifichi correttamente ADDR in uscita, ovvero calcoli correttamente offset e codifica binaria della WZ

2) Test casuali con ADDR non appartenente a una WZ: verificano che in questo caso il componente scriva in RAM ADDR così come l'ha ricevuto in ingresso

3) Segnali di start multipli con ADDR diverso ogni volta: verificano che il componente risponda in maniera corretta a richieste multiple di codifica di ADDR differenti, senza variazioni nelle WZ

Reset:

1) Reset durante il calcolo di un ADDR: verifica che il componente torni correttamente allo stato di inizializzazione prima di concludere la codifica di un ADDR precedentemente avviata

2) Reset immediatamente dopo una codifica: verifica che il componente torni allo stato di inizializzazione dopo aver correttamente soddisfatto una richiesta di codifica, con particolare attenzione al caso in cui *reset<='1'* immediatamente dopo *start<='0'*, ovvero prima che il componente abbia il tempo di tornare allo stato di *wait_start*

3) Reset durante l'inizializzazione: verifica che il componente sia in grado di ricominciare da capo un'inizializzazione già avviata

Timing (Post-Synthesis Functional Simulation):

1) Componente da inizializzare e ADDR non in WZ (caso pessimo): 3200 ns (32 cicli di clock)

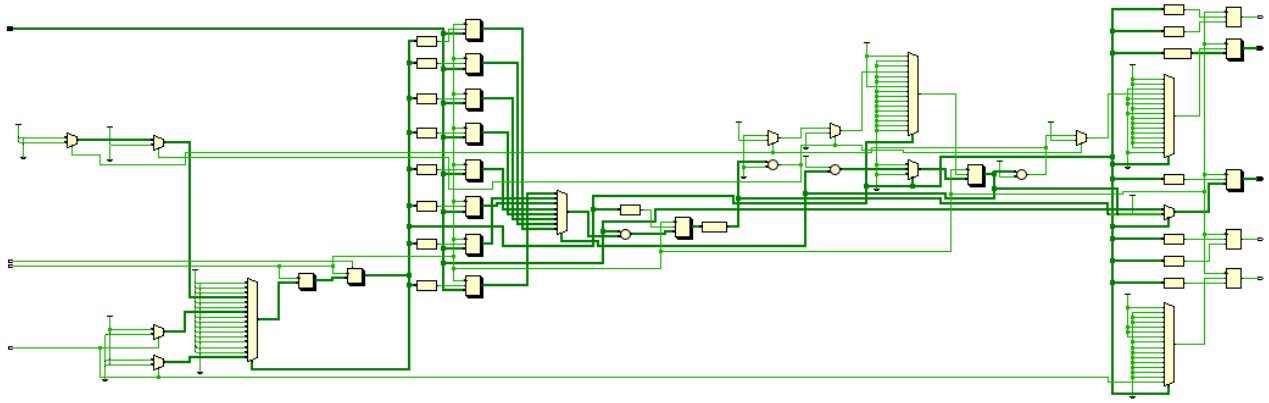
2) Componente già inizializzato e ADDR nella prima WZ (caso ottimo): 700 ns (7 cicli di clock)

3) Inizializzazione (da *reset<='1'* a *wait_start*): 900ns (9 cicli di clock)

Per ogni WZ in più controllata vengono impiegati 2 cicli di clock

Risultati sintesi

Dal report di sintesi si evince che, usando la FPGA xc7a200tfbg484-1, il componente è stato effettivamente sintetizzato come una FSM a 16 stati, codificati in one-hot, utilizzando 50 LUTs e 122 Flip Flops. Si riporta di seguito il diagramma *schematic* elaborato da Vivado:



Conclusioni

Il componente ha superato tutti i test casuali e specifici, sia in *Behavioral Simulation* che in *Post-Synthesis Functional Simulation*, come richiesto da specifica. Inoltre, ha dimostrato di essere molto robusto a variazioni importanti del periodo di clock, con un limite inferiore calcolato di 2ns (1/50 del periodo di clock imposto da specifiche) e nessun limite superiore.