Prova finale – Progetto di reti logiche

Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Ingegneria Informatica
Sezione Prof. Fornaciari
A.A. 2019/2020

Indice

1 Introduzione

Scopo della Prova Finale di Reti Logiche è implementare un encoder per Working Zones come descritto dalla specifica fornita. Per la realizzazione del progetto è stata utilizzata la fpga xc7a200tfbg484-1 nell'editor di Vivado Webpack 2019; la documentazione utilizzata per la realizzazione del progetto consiste nelle guide base di Vivado e di VHDL fornite dai docenti.

1.1 Scelte progettuali

Secondo la specifica definita, per la realizzazione del progetto sono stati utilizzati due registri per mantenere l'indirizzo da codificare e l'indirizzo della Working Zone analizzata al momento. Inoltre, si è deciso di mantenere un registro contatore per le operazioni effettuate, usato anche per comandare gli indirizzi cui accedere; in questo modo la gestione dell'accesso alla memoria è stato delegato al data path invece che alla macchina a stati finiti.

In base alle richieste sono anche state effettuate le seguenti scelte:

- Poiché non viene dichiarato che le WZ si trovino sempre in un ordine definito il circuito analizzerà tutti gli indirizzi prima di giungere alla conclusione di non appartenenza a nessuna WZ dell'indirizzo.

- Per quanto riguarda i valori non validi, il circuito dovrà considerare gli indirizzi di input come interi unsigned, quindi non sarà possibile inserire valori al di sotto di quelli considerati validi; se viene inserito un valore da codificare superiore a 127 il comportamento non dovrà differire da un'esecuzione normale, ma in caso di indirizzo non presente nelle WZ il MSB verrà sovrascritto dal WZ_BIT, mantenendo quindi un comportamento consistente anche in casi eccezionali.

1.2 Ottimizzazione con pipeline

Per sfruttare al meglio la capacità di parallelizzazione dell'hardware si è optato per una soluzione con una pipeline, cosicché i moduli di gestione indirizzi, input e analisi (illustrati in seguito) potessero lavorare in contemporanea, riducendo notevolmente il tempo di esecuzione. A causa di questa scelta è stato però necessario utilizzare un registro contatore per gestire l'offset tra i moduli.

Un esempio di esecuzione delle tre componenti in parallelo è mostrato in Figura 1: una volta iniziata l'esecuzione il circuito richiede alla memoria prima l'indirizzo da codificare, quindi l'indirizzo della prima Working Zone. Da

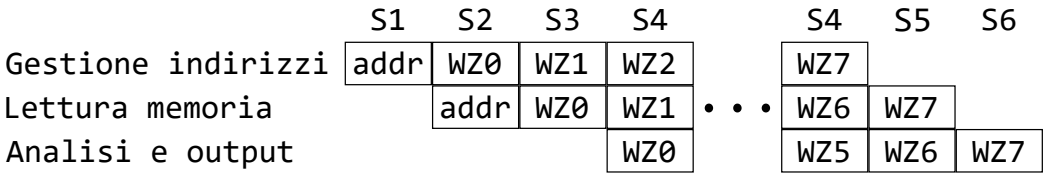

Figura 1: Esempio di funzionamento della pipeline per caricamento e analisi

questo punto inizia l'esecuzione parallela a pieno regime, per cui avvengono contemporaneamente:

- Analisi dell'appartenenza a una WZ;
- Caricamento della WZ successiva (se esiste);
- Richiesta di quella dopo ancora (se esiste).

Se l'analisi di appartenenza è positiva il sistema viene congelato e si procede all'output, altrimenti questo avviene una volta terminata l'analisi dell'ultima Working Zone.

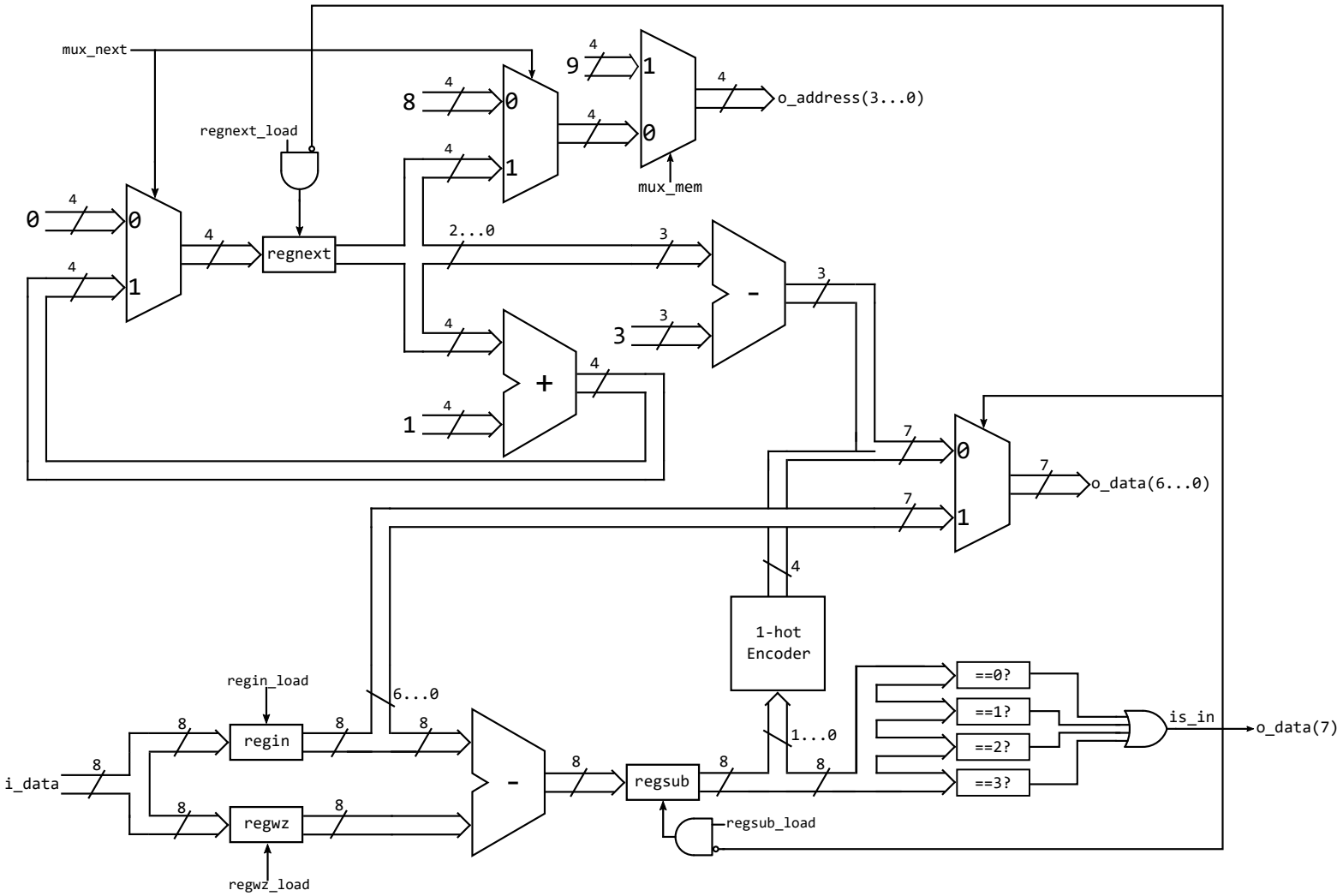

Figura 2: Data Path

2 Architettura

2.1 Data Path

In Figura 2 è mostrato il datapath; sono stati omessi i componenti utilizzati per il reset dei registri per favorire la leggibilità.

Al fine di sfruttare al meglio la pipeline è stato suddiviso in 4 parti:

- Gestione indirizzi, per calcolare gli indirizzi di memoria cui accedere e mantenere un contatore;
- Gestione input, per caricare nei registri i valori presenti in memoria;
- Analisi, per valutare l'appartenenza di un indirizzo a una WZ;
- Output, per gestire i segnali da inviare alla memoria.

2.1.1 Gestione degli indirizzi di memoria

La gestione degli indirizzi cui accedere è realizzata con il registro `regnext` a 4 bit, che a inizio esecuzione viene inizializzato a 0 mentre alla memoria è richiesto l'indirizzo da codificare. In seguito, il registro inizia un ciclo di conteggio per poter accedere a tutte le celle di memoria contenenti gli indirizzi delle Working Zones e mantenere un contatore utile a gestire gli offset dati dalla pipeline.

Questo modulo è realizzato in VHDL con un process per realizzare `regnext` (compreso il sommatore collegato) e due costrutti `with/select` per i multiplexer in serie. Il valore di `o_address` viene infine ricavato concatenando i dodici zeri meno significativi all'uscita dei multiplexer (per raggiungere i 16 bit degli indirizzi).

2.1.2 Gestione degli input

Per gestire i valori di input sono utilizzati due registri a 8 bit collegati a `i_data`; il registro `regin` contiene l'indirizzo da codificare, mentre `regwz` l'indirizzo base della Working Zone in esame in questa fase (il cui numero corrisponde a `regnext`-1 per l'offset di pipeline).

La realizzazione in VHDL si compone semplicemente di due registri (realizzati con dei process) collegati all'input e attivati mai contemporaneamente.

2.1.3 Analisi

La fase di analisi si avvale infine del registro `regsub` a 8 bit (inizializzato a un valore arbitrario non significativo), che contiene la differenza tra l'indirizzo da codificare e l'indirizzo base della WZ; se questo valore è compreso o uguale tra 0 e 3 allora l'indirizzo fa parte della WZ e viene attivato il flag `is_in` (che blocca il caricamento dei registri `regnext` e `regsub` per congelare il sistema).

Il codice di realizzazione si avvale di un process per calcolare e salvare il valore di `regsub`, e di una batteria di comparatori per determinare il valore di `is_in`, ottenuto in modo combinatorio con un'espressione logica.

2.1.4 Output

Da un punto di vista di pipeline la fase di output può essere considerata unita a quella di analisi, poiché composta solo da reti combinatorie (si trovano quindi sulla stessa Working Zone). Utilizzando il valore di `is_in` vengono selezionati i 7 bit meno significativi dell'output:

- Se `is_in` = 0, viene selezionato il valore di `regin`;
- Se `is_in` = 1, l'uscita risulta essere la concatenazione della WZ e della codifica one-hot del valore di `regsub`.

In quest'ultimo caso, il valore della Working Zone in uscita è determinato come i 3 bit meno significativi del valore (bloccato da `is_in`) contenuto in `regnext`, cui viene sottratto l'offset dovuto al ritardo nella pipeline (ovvero 2 + 1 dovuto alla commutazione di `regnext`); la WZ è quindi `regnext`−3. Inoltre, `is_in` stesso viene utilizzato come bit più significativo dell'output. La codifica one-hot a 2 ingressi è invece ottenuta attraverso una rete combinatoria (Illustrata in Figura 3) ricavata dalla seguente tavola di verità:

A	B	U3	U2	U1	U0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

La codifica VHDL del modulo viene realizzata con un costrutto `with/select` per il multiplexer (per cui uno degli ingressi risulta essere la concatenazione del valore della WZ attuale e dell'uscita dell'encoder one hot) e un'espressione logica per l'encoder one-hot.

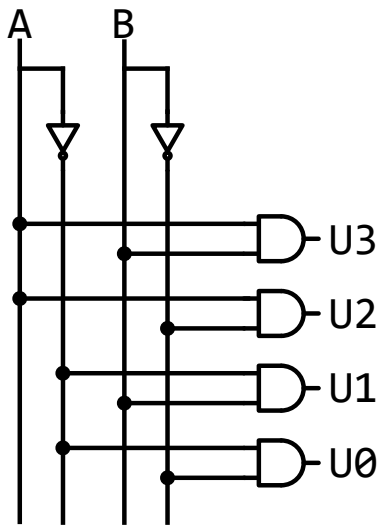

Figura 3: Encoder one-hot a due ingressi

2.2 Macchina a stati finiti

Per la gestione dei segnali di controllo è stata realizzata la macchina con 9 stati illustrata in Figura 4. Oltre a S0 si compone di 3 macrostati:

- S1 + S2 + S3: Stati di inizializzazione;
- S4 + S5 + S6: Stati di analisi;
- S7 + S8: Stati di output.

L'interazione degli stati con la pipeline è illustrata in Figura 1.

2.2.1 S0

S0 è lo stato iniziale e di reset, in cui il sistema rimane finché `i_start` = 0. Sono stati omessi dallo schema tutti i collegamenti da ogni stato che rappresentano la transizione per `i_rst` = 1.

2.2.2 Stati di inizializzazione: S1, S2, S3

In **S1** viene inizializzato `regnext` e viene richiesto alla memoria l'indirizzo da codificare (poiché `mux_nxt` = 0).

S2 si occupa invece di caricare in `regin` l'indirizzo da codificare e richiedere l'indirizzo della prima Working Zone (ponendo a 1 `mux_nxt`).

S3 infine carica l'indirizzo della prima WZ in `regwz` e prepara l'esecuzione parallela con la pipeline.

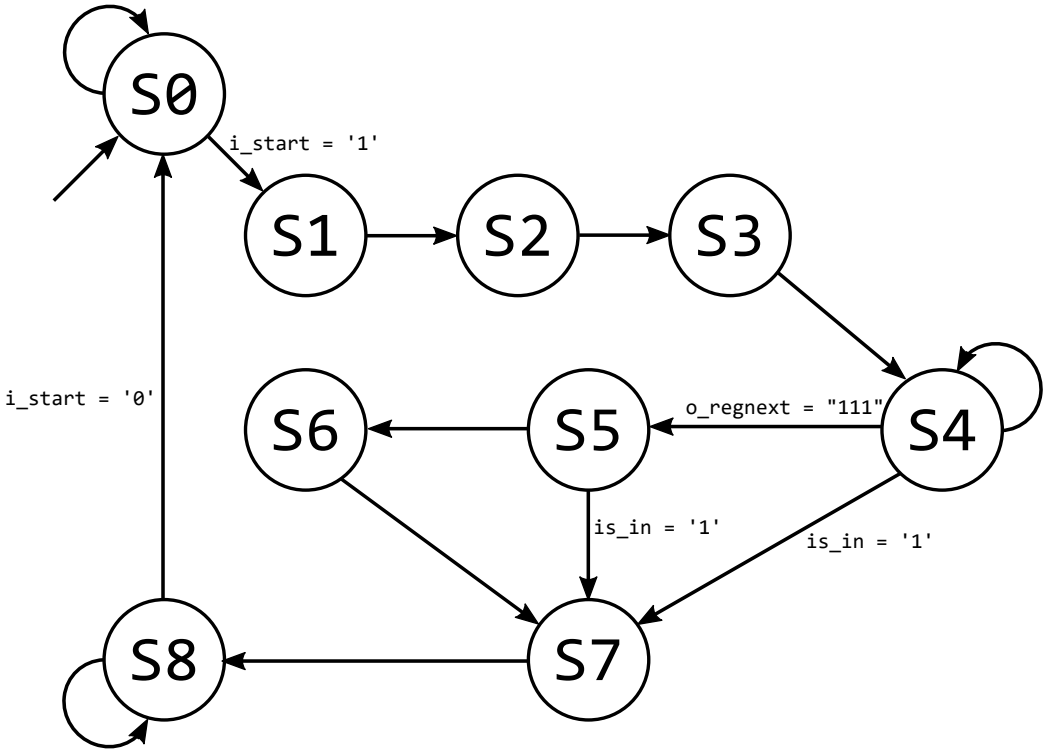

Figura 4: Macchina a stati finiti

2.2.3 Stati di analisi: S4, S5, S6

S4 è il primo stato di elaborazione, con pipeline a regime; analizza l'indirizzo in `regin` e la WZ in `regwz`, finché non trova la WZ di appartenenza (per cui il sistema passa in stato di output) o il registro `regnext` raggiunge la fine delle celle di memoria contenenti le Working Zones (e avviene la transizione verso S5).

S5 è lo stato di elaborazione in cui non è più necessario leggere in memoria, quindi viene bloccato l'accesso agli indirizzi successivi al 7; le altre due linee di pipeline possono continuare se non è stata trovata la WZ corretta (altrimenti si passa in output).

S6 è l'ultimo stato di elaborazione e consiste nel bloccare anche il caricamento in memoria mentre viene effettuata l'analisi dell'ultima WZ. Sia che la WZ analizzata contenga l'indirizzo sia che non lo riguardi, S6 evolve nello stato di output.

2.2.4 Stati di Output: S7, S8

S7 permette di scrivere in memoria, in posizione 9, il risultato (controllato interamente da `is_in`).

S8 è l'ultimo stato di un'elaborazione; attiva i flag `o_done` e `end_reset`, quest'ultimo con funzione analoga a `i_reset` per i registri. Infine, riporta il sistema allo stato iniziale quando `i_start` torna a 0.

2.2.5 Segnali di controllo e VHDL

La codifica della macchina a stati è stata realizzata con due process, rispettivamente per gestire le transizioni (sia in casi normali che in caso di reset) e per attivare i segnali di controllo opportuni in base allo stato corrente (con un `case`); nelle due tabelle sottostanti sono riassunti i segnali attivi per ogni stato.

	o_en	o_we	o_done	mux_nxt	mux_mem	end_reset
S0	0	0	0	0	0	0
S1	1	0	0	0	0	0
S2	1	0	0	1	0	0
S3	1	0	0	1	0	0
S4	1	0	0	1	0	0
S5	1	0	0	1	1	0
S6	1	0	0	1	1	0
S7	1	1	0	0	1	0
S8	0	0	1	0	0	1

	regnext_load	regin_load	regwz_load	regsub_load
S0	0	0	0	0
S1	1	0	0	0
S2	1	1	0	0
S3	1	0	1	0
S4	1	0	1	1
S5	1	0	1	1
S6	1	0	0	1
S7	0	0	0	0
S8	0	0	0	0

3 Sintesi

La sintesi del componente descritto in VHDL attraverso Vivado ha generato i risultati riportati di seguito.

3.1 Report di timing

Il report di timing illustra come sia rispettato il constraint definito (ovvero che il sistema deve funzionare con un tempo di clock di al più 100ns). È riportato il riassunto del report:

```
------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------

 WNS(ns)      TNS(ns)   TNS Failing Endpoints   TNS Total Endpoints
 -------      -------   ---------------------   -------------------
  96.767        0.000                       0                    77

 WHS(ns)      THS(ns)   THS Failing Endpoints   THS Total Endpoints
 -------      -------   ---------------------   -------------------
   0.144        0.000                       0                    77

WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
--------     --------  ----------------------  --------------------
   4.500        0.000                       0                    33

All user specified timing constraints are met.
```

In particolare il WNS è calcolato come $max(required\_time - arrival\_time)$ per la propagazione dei segnali; questo implica che, relativamente a questo parametro, è possibile diminuire il tempo di clock fino a circa 4ns senza che il tempo sia insufficiente per la propagazione.

3.2 Report di utilizzo

Nel report di utilizzo è stato possibile valutare il numero di componenti utilizzati; in particolare per i registri non sono stati riportati latch:

Site Type	Used	Fixed	Available	Util%
Slice Registers	37	0	269200	0.01
Register as Flip Flop	37	0	269200	0.01
Register as Latch	0	0	269200	0.00

Nella lista di primitives si può invece notare che vengono utilizzati esclusivamente flip-flop di tipo D (FDCE e FDPE).

Ref Name	Used	Functional Category
FDCE	35	Flop & Latch
OBUF	27	IO
LUT2	12	LUT
IBUF	11	IO
LUT4	8	LUT
LUT5	7	LUT
LUT3	7	LUT
LUT6	4	LUT
FDPE	2	Flop & Latch
CARRY4	2	CarryLogic
BUFG	1	Clock

Le connessioni dei componenti sono mostrate in Figura 5.

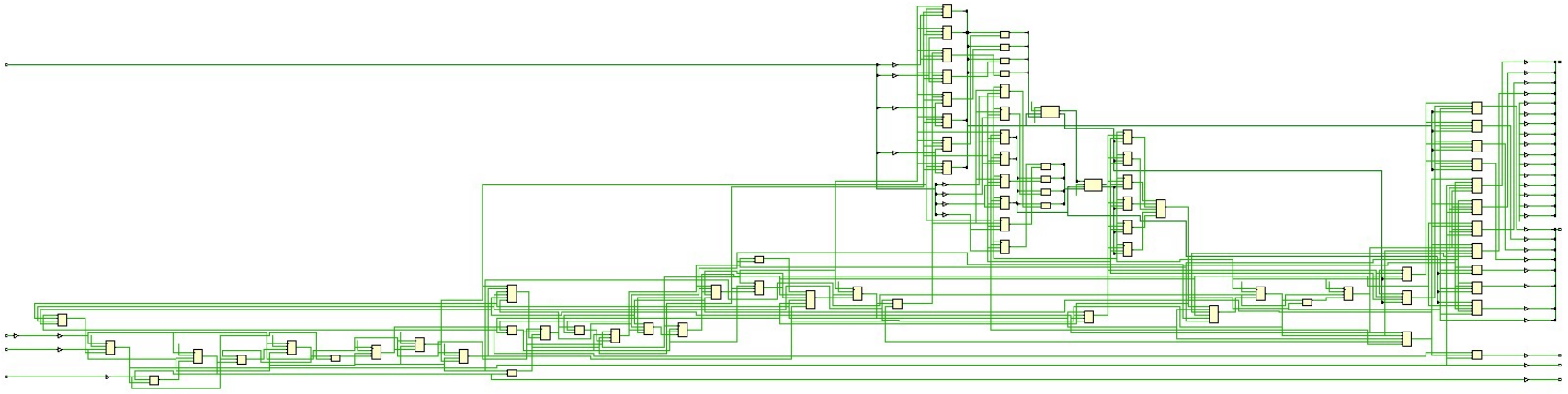

Figura 5: Schema sintetizzato

4 Testing

Per valutare la correttezza del componente sviluppato rispetto alle specifiche si è fatto uso di alcune serie di test di cui sono riportati alcuni esempi nella tabella a fine sezione. Il tempo è considerato da quando viene rilevata l'attivazione del segnale `i_start` a quando viene attivato `o_done` (In pratica il tempo impiegato per tornare a S0, valutato come numero di cicli di clock necessari).

4.1 Casi normali

I tipi di test per questo tipo di esecuzione hanno seguito le modalità dei testbench forniti (Esempi BT1 e BT2), valutando quindi sia casi di appartenenza a WZ sia casi negativi. I test effettuati hanno dato riscontri corretti sia in presintesi sia in postsintesi.

4.1.1 Casi limite

Per valutare il corretto funzionamento anche in casi limite sono stati realizzati testbench (Esempi BT3 e BT4) per cui l'indirizzo:

- Appartenesse a WZ0 (Caso iniziale, con pipeline a regime);
- Appartenesse a WZ7 (Caso finale, nell'ultimo stato di analisi);

Per quanto riguarda invece le WZ, negli stessi esempi si sono testati casi di indirizzi base fuori ordine e al limite dei valori validi. Tutti questi tipi di test sono stati eseguiti correttamente anche in postsintesi.

4.2 Casi eccezionali

Sono stati effettuati anche test in casi di esecuzione non normale per valutare la consistenza di comportamento del sistema.

4.2.1 Esecuzioni con restart

Sono stati effettuati test aggiungendo dei reset asincroni all'esecuzione dei benchtest già illustrati; prima dell'istruzione di `wait` per `tb_done` è stato quindi inserito il seguente codice (con tempo di attesa variabile):

```
wait for 500 ns;
wait for c_CLOCK_PERIOD;
tb_rst <= '1';
```

```
wait for c_CLOCK_PERIOD;
tb_rst <= '0';
wait for c_CLOCK_PERIOD;
tb_start <= '1';
wait for c_CLOCK_PERIOD;
```

È stata testata anche la possibilità di esecuzioni multiple sequenziali senza reset ma con cambio di valori tra la terminazione del primo e lo start del secondo.

Dopo il restart i test sono stati completati correttamente, sia in presintesi sia in postsintesi.

4.2.2 Valori non validi

Sono anche stati effettuatia alcuni test con valori non validi, al fine di controllare che le ipotesi effettuate in sezione 1.1 fossero corrette. Un esempio è mostrato dai test BT5 e BT6.

4.3 Modifica del tempo di clock

Come ultimi test sono stati diminuiti i tempi di clock delle simulazioni secondo quanto riportato nel report di sintesi. La simulazione postsintesi functional si comporta correttamente per i tempi di clock ipotizzati nel report di timing.

Test	0	1	2	3	4	5	6	7	8	Risultato	Nclock
BT1	4	13	22	31	37	45	77	91	33	180 (1 011 0100)	9
BT2	4	13	22	31	37	45	77	91	42	42 (0 00101010)	12
BT3	126	13	22	31	37	45	77	0	127	130 (1 000 0010)	6
BT4	1	13	22	31	37	45	77	0	0	241 (1 111 0001)	12
BT5	4	13	22	31	37	45	77	91	255	127 (0 1111111)	12
BT6	4	13	22	31	37	45	77	254	255	226 (1 111 0010)	12

5 Conclusioni

Secondo la specifica fornita è stato possibile realizzare correttamente tramite descrizione in VHDL e successiva sintesi tramite Vivado un encoder di Working Zones. Il sistema è ottimizzato rispetto al tempo di esecuzione grazie all'uso della pipeline ed impiega tra 6 e 12 cicli di clock per fornire un output; il vincolo relativo al tempo di clock è rispettato ed è possibile un funzionamento in postsintesi con tempi fino a circa 4ns (functional).