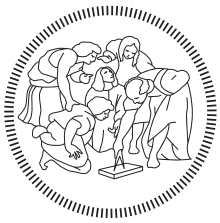


# Progetto finale Reti Logiche

Luca De Martini

## 1 Introduzione

Il componente realizzato implementa la codifica "Working Zone" per indirizzi di 7 bit con 8 working zones di dimensione 4.
La codifica working zone consiste nell'esprimere un indirizzo in funzione della sua posizione relativa all'inizio di una working zone, se ne fa parte, altrimenti lasciarlo invariato.
Se l'indirizzo da codificare non fa parte di nessuna working zone il primo bit viene messo a '0' e i restanti 7 bit sono quelli originali.
Altrimenti il risultato è composto da tre parti: il primo bit messo a '1' a indicare che l'indirizzo fa parte di una working zone, i successivi 3 bit contengono il numero della working zone di cui fa parte, espresso in codifica binaria (da 000 a 111), mentre gli ultimi 4 bit contengono l'offset dell'indirizzo rispetto all'inizio (o base) della working zone espresso in codifica one-hot (da 0001 a 1000)
In tabella 1 sono riportati degli esempi di codifica, dove l'indirizzo codificato è separato a indicare le diverse parti del risultato.

| **Address** | **Working zones** | **Encoded address** |
|---|---|---|
| ... | | ... |
| 0000110 | | 0 0000110 |
| 0000111 | wz2 | 1 010 0001 |
| 0001000 | wz2 | 1 010 0010 |
| 0001001 | wz2 | 1 010 0100 |
| 0001010 | wz2 | 1 010 1000 |
| 0001011 | | 0 0001011 |
| 0001100 | | 0 0001100 |
| 0001101 | | 0 0001101 |
| 0001110 | wz3 | 1 011 0001 |
| ... | wz3 | ... |

wz2 = 0000111, wz3 = 0001110

Table 1: Esempio di codifica

# 2 Architettura

Al livello più alto il componente presenta un modulo `project_reti_logiche` che contiene tutti i moduli interni, un'automa a stati finiti e un registro di cache dell'indirizzo in output.

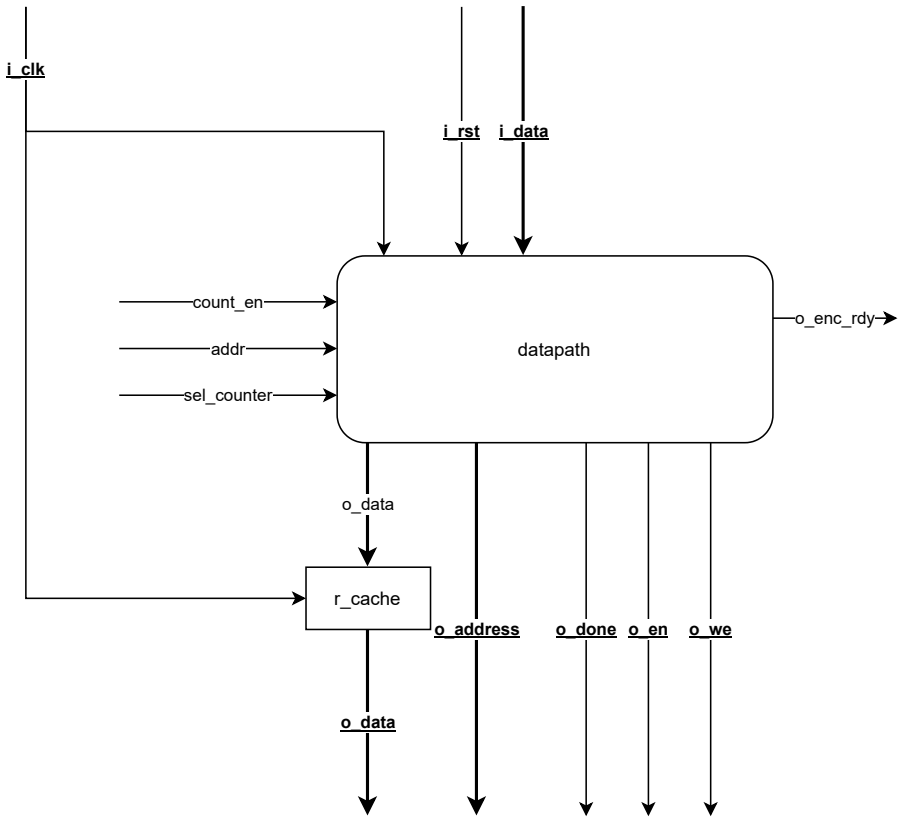



Figure 1: Schema modulo principale

L'automa a stati finiti è implementato con una collezione di process:
Due process descrivono il comportamento del registro di cache e quello del registro di stato, entrambi sono realizzati con flip-flop sincroni attivati sul fronte di discesa, la scelta di usare il falling edge è stata fatta per ridurre il numero di cicli di clock necessari per la codifica.
Un altro process aggiorna il valore del segnale `next_state` in base allo stato attuale, al segnale in input di `i_start` e al segnale `enc_rdy`, output del modulo interno `datapath` che verrà descritto nel dettaglio successivamente.
Infine un ultimo process aggiorna i valori dei segnali in input ai moduli interni e in output dal componente necessari al funzionamento basandosi sullo stato attuale.

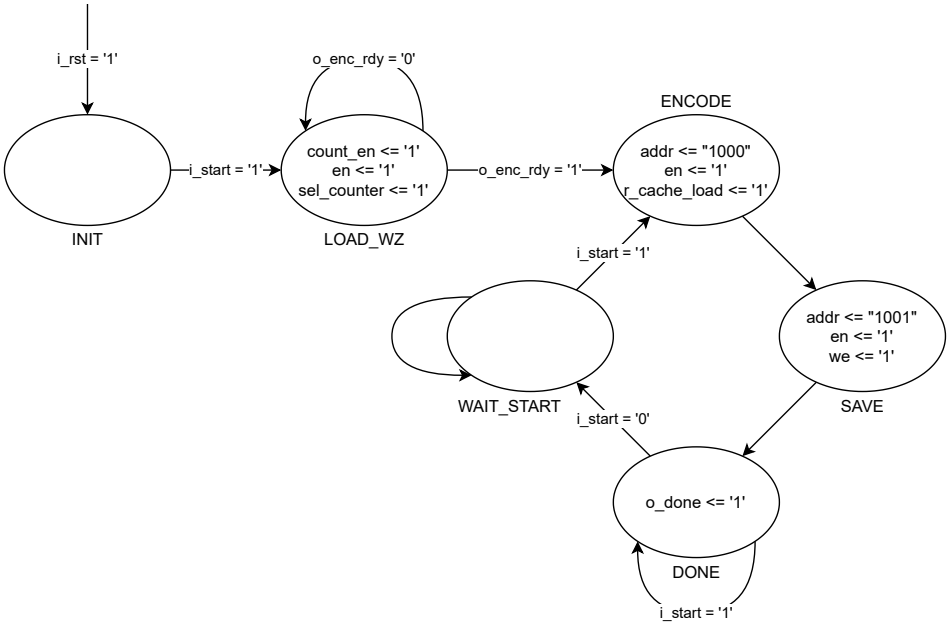



Figure 2: Automa a stati finiti

L'FSA ha 6 stati ed è caratterizzato da due zone di funzionamento, la prima zona è quella iniziale di setup costituita dagli stati 'INIT' e 'LOAD_WZ': una volta ricevuto il segnale di `i_start` il componente si prepara alla codifica rimanendo nello stato 'LOAD_WZ' mentre sta caricando gli indirizzi delle working zones nei suoi registri interni.
La fine di questa fase è segnalata dal segnale `o_enc_rdy` proveniente dai componenti interni che viene alzato a '1' quando è pronto a codificare, avanzando così l'automa al prossimo stato 'ENCODE' e iniziando la fase di codifica.
La fase di codifica iniza nello stato 'ENCODE' dove il componente legge l'indirizzo da codificare, lo codifica e lo carica nel registro di cache in un unico ciclo di clock. Nello stato seguente, 'SAVE', viene scritto in RAM il risultato e in quello dopo, 'DONE', viene alzato il segnale di `o_done` segnando la fine della codifica.
Dopo di che il componente attenderà l'abbassarsi del segnale di `i_start` per abbassare `o_done` e poi mettersi in attesa di un nuovo segnale di start nello stato 'WAIT_START', pronto per una nuova codifica.

## 2.1 Moduli interni

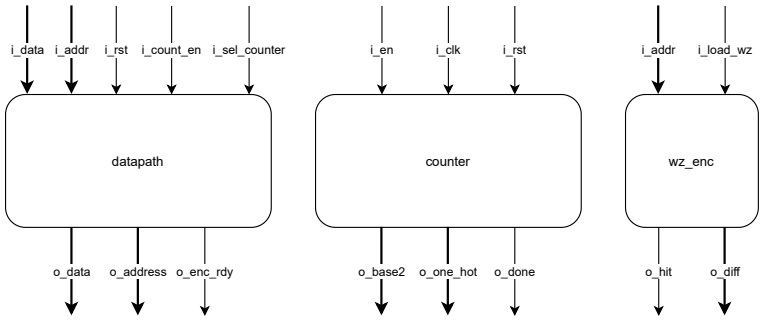



Figure 4: Moduli interni

Il componente contiene un modulo `datapath` che ha al suo interno un modulo `counter` e 8 moduli `wz_enc` identici (uno per ogni indirizzo della working zone).

### 2.1.1 wz_enc

I moduli `wz_enc` si occupano di stabilire se un indirizzo sia all'interno di una determinata working zone, segnalandolo con `o_hit`, e di calcolarne l'offset in caso positivo. L'offest rispetto alla working zone è `o_diff` che è da considerarsi valido solo se `o_hit` è a '1', altrimenti avrà tutti i bit a '0'

### 2.1.2 counter

Il modulo `counter` è un contatore da 0 a 7 che scatta sul falling edge. Esso inizia a contare quando riceve un segnale di enable, dando l'output sia in codifica binaria, sia in codifica one hot, dopo aver sorpassato il 7 tutti i bit degli output vengono messi a '0'

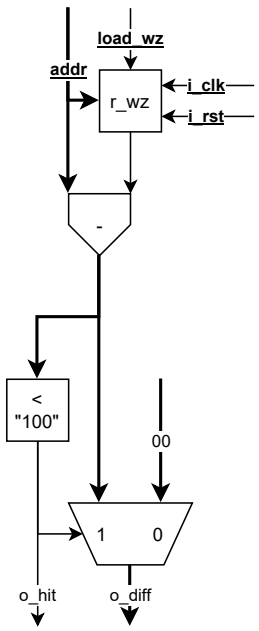



Figure 3: Schema di `wz_enc`

### 2.1.3 datapath

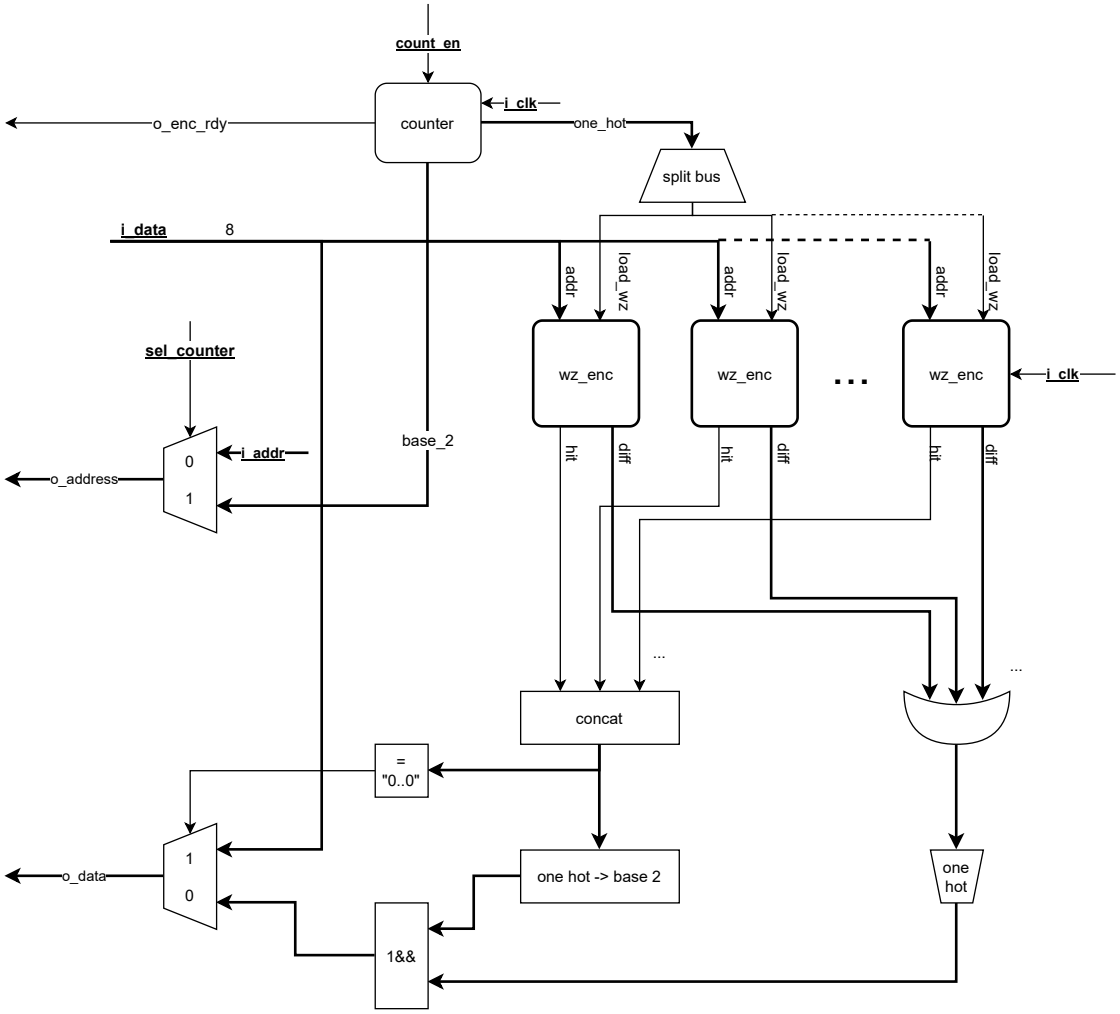



Figure 5: Schema di `datapath`

Il `datapath` divide il bus contenente la codifica one hot in uscita dal `counter` collegando ogni bit del segnale al segnale di load di un `wz_enc`, in questo modo, usando il segnale in codifica binaria per l'indirizzo in uscita (`o_address`) permette di caricare l'indirizzo delle working zone nei rispettivi registri.
Utilizzando l'output degli `o_hit` dei `wz_enc` il `datapath` può rilevare se un indirizzo faccia parte di una working zone o meno, poi combinando gli output degli `o_diff` può calcolare la codifica completa dell'indirizzo.

# 3 Risultati sperimentali

Dal report di sintesi si può osservare che il design del componente utilizza in minima parte le risorse disponibili sulla superficie dell'FPGA, non sono presenti black-box e non sono state sintetizzate strutture con un numero di componenti indesiderato.

| Site Type | Used | Util% |
|---|---|---|
| Slice LUTs | 98 | 0.07 |
| LUT as Logic | 98 | 0.07 |
| LUT as Memory | 0 | 0.00 |
| Slice Registers | 71 | 0.03 |
| Register as Flip Flop | 71 | 0.03 |
| Register as Latch | 0 | 0.00 |

Table 2: Slice Logic

Il numero di nodi utilizzato scala linearmente con il numero di working zone e visto l'ampio margine disponibile sarebbe possibile creare un componente che supporti più working zones con questo stesso design senza incorrere in problemi di eccessiva utilizzazione della superficie dell'FPGA

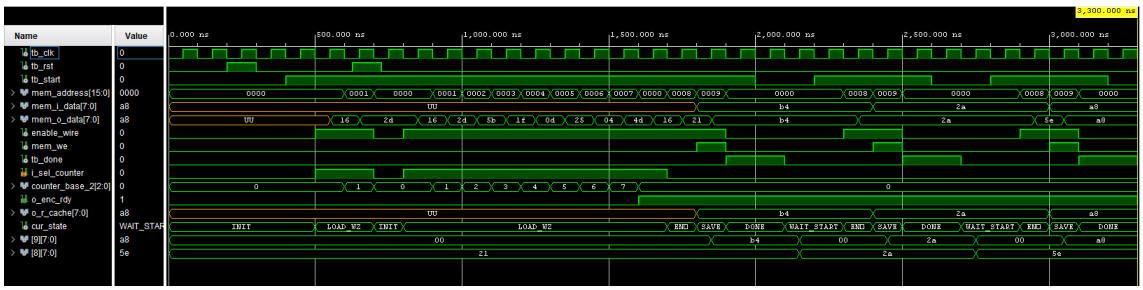
| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 71 | Flop & Latch |
| LUT2 | 58 | LUT |
| OBUF | 27 | IO |
| LUT4 | 16 | LUT |
| CARRY4 | 16 | CarryLogic |
| LUT5 | 15 | LUT |
| LUT6 | 13 | LUT |
| IBUF | 11 | IO |
| LUT3 | 7 | LUT |
| LUT1 | 1 | LUT |
| BUFG | 1 | Clock |

Table 3: Primitive utilizzate

Il timing report utilizzando un clock con periodo di 100ns segnala Worst Negative Slack di 94.1ns, Worst Hold Slack di 0.154ns e Worst Pulse Width Slack di 49.5ns.
Quindi anche per quanto riguarda il timing il componente rispetta con ampi margini la specifica del progetto e rispetterebbe i constraints di timing anche con un clock 10 volte più veloce.
Infine il power report stima un Total On-Chip Power di 0.131W

# 4 Simulazioni

Il componente è stato testato con simulazioni comportamentali pre sintesi, simulazioni functional e timing post sintesi e functional post implementazione:
Nelle test bench è stata testata la rubustezza del design con reset asincroni, codifiche in sequenza con indirizzi diversi e working zones cambiate dopo un reset
A seguire in figura 6 è riportata la vista dei segnali durante una simulazione significativa dove, dopo aver ricevuto un reset asincrono in fase di lettura delle working zones, vengono codificati correttamente tre indirizzi in sequenza di cui due fanno parte di una working zone e uno no.




Working zones: `[16,2D,5B,1F,0D,25,04,4D]`, Indirizzi: `[21,2A,5E]`

Figure 6: Wave view durante una simulazione

# 5 Conclusioni

Il design scelto utilizza un registro per working zone, per questo motivo la prima codifica dopo un reset sarà la più lenta, vista la necessità di caricare nei registri gli indirizzi base delle working zones, tuttavia le codifiche successive saranno più veloci usando solo due cicli di clock per la codifica e due per segnalare la fine e attendere un nuovo start.
Un approccio alternativo sarebbe quello di salvare in un registro l'indirizzo da codificare e scorrere tra le working zones confrontando passo passo l'indirizzo con la base delle working zones fermando la codifica se si trova un offset minore di 4 con la base di una working zone, o una volta passate tutte le working zones.
Il secondo approccio richiede un numero minore di registri, che non aumenterebbe se le working zones fossero più numerose e può essere più veloce sulla prima codifica dopo un reset in quanto non richiede una fase di setup iniziale e se l'indirizzo appartiene a una delle prime working zones la codifica termina con pochi cicli di clock.
Tuttavia per le specifiche progettuali il design scelto rientra con ampio margine nei limiti di superficie e, oltre a richiedere meno cicli di clock per codifiche successive, ha il vantaggio di richiedere un numero di cicli di clock costante per la codifica, indipendente dal contenuto della RAM, che potrebbe portare dei vantaggi nella semplicità di utilizzo del componente.