POLITECNICO
MILANO 1863

Prova Finale di Reti Logiche

Gabriele Ghezzi

Codice persona: 10775052
Matricola: 984763
Docente: Salice Fabio

A.A. 2023/2024

INDICE

INTRODUZIONE 3
1.1 FORMALIZZAZIONE DEL PROBLEMA 3
ARCHITETTURA 4
2.1 COUNT WORDS 5
2.2 CHECK COUNT 6
2.3 REGISTER 6
2.4 COUNT VALIDITY 6
2.5 FSM 7
RISULTATI SPERIMENTALI 8
3.1 SINTESI 8
3.2 SIMULAZIONI 8
CONCLUSIONI 9
Appendice A 10

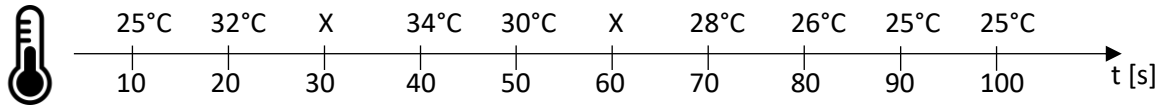
INTRODUZIONE

Il problema sottoposto è costituito da due parti:

I. Analisi e salvataggio di una sequenza di dati con relativa imputazione delle letture assenti.
II. Attribuzione di un indice di credibilità al dato salvato in memoria.

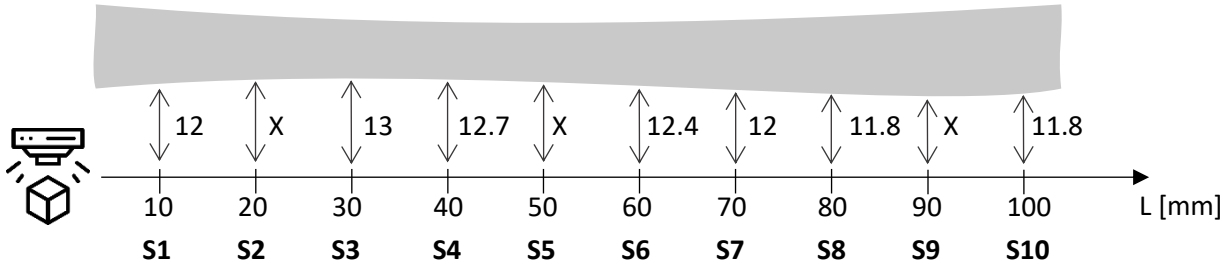
Possiamo pensare a questa sequenza di dati come a delle letture svolte da uno o più sensori, ad esempio:

Un termometro che rileva la temperatura con una frequenza fissata.



oppure

Diversi sensori di prossimità equidistanti che rilevano la distanza da un corpo contemporaneamente.



Casi nei quali, quindi, sotto l'ipotesi che i dati rilevati siano corretti, la scelta di imputare i valori assenti con l'ultima lettura valida effettuata ha senso anche per un'ipotetica analisi statistica.

Ed è per questo che la seconda parte del problema consiste nell'assegnare un indice di credibilità che si decrementa ogni volta che manca un dato, ripartendo dal suo valore massimo ad ogni lettura valida effettuata.

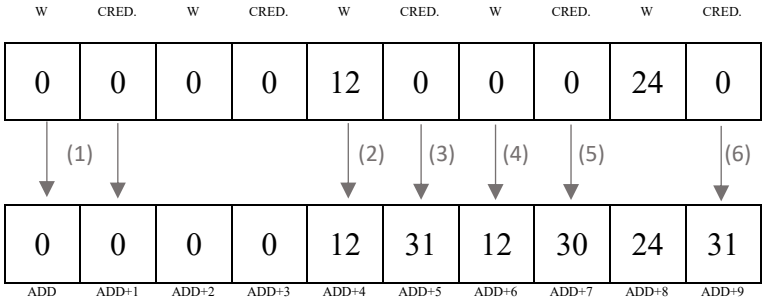
1.1 FORMALIZZAZIONE DEL PROBLEMA

Formalizziamo quindi il problema con una sequenza di K parole W (da un byte l'una), il cui valore può variare tra 0 e 255 con estremi inclusi.
Notare che i valori [1, 255] sono interpretabili come dati letti dal sensore, lo 0 invece è interpretato come assenza di valore e quindi come caso in cui applicare la sostituzione stabilita.
L'indice di credibilità $c \in [0, 31]$, partendo da 31 ogni volta che W è diversa da 0 e decrementando di 1 in caso contrario (rimanendo però sempre ≥ 0).

La sequenza è memorizzata nella stessa memoria RAM in cui verrà sovrascritta: precisamente sarà registrata a partire da un indirizzo designato ADD a byte alterni, riservando lo spazio aggiuntivo per il coefficiente di credibilità.

Nel caso in cui i primi elementi della sequenza manchino, sia W che la credibilità sono impostati in memoria a 0, fino alla prima lettura di una W diversa da 0.

Es:



(1) Sequenza che parte con 0: riscrivo 0 sia per W, sia per l'indice di credibilità.
(2) Primo W letta diversa da 0, la salvo in memoria.
(3) Lettura precedente valida: l'indice di credibilità riparte dal suo valore massimo.
(4) W letta a zero: riscrivo l'ultima lettura valida.
(5) La lettura precedente era nulla: decremento l'indice di credibilità.
(6) Lettura precedente valida: l'indice di credibilità riparte dal suo valore massimo.

ARCHITETTURA

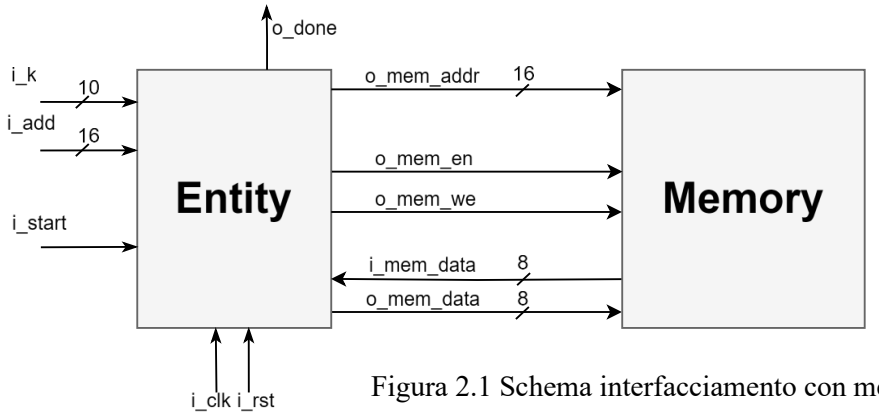


Figura 2.1 Schema interfacciamento con memoria.

Per risolvere il problema il compito è di progettare un componente HW che si interfacci a una memoria RAM come indicato in figura 2.1.

Segnale	Descrizione
i_clk	Clock di sincronizzazione.
i_rst	Segnale asincrono di reset.
i_k ……(16 bit)	Numero di parole da leggere da memoria.
i_add…….(10 bit)	Indirizzo di partenza della sequenza.
i_start	Segnale di inizio elaborazione.
i_mem_data …….(8 bit)	Valore letto da memoria all'indirizzo o_mem_addr.
o_mem_data…….(8 bit)	Valore inoltrato alla memoria.
o_mem_addr ……...(16 bit)	Segnale di indirizzamento della memoria.
o_mem_en	Segnale di abilitazione della memoria (necessario per leggere e scrivere).
o_mem_we	Segnale di abilitazione in scrittura della memoria (necessita di o_mem_en attivo).
o_done	Segnale di fine elaborazione.

Tabella 2.1 Segnali di interfacciamento.

L'architettura progettata è la seguente:

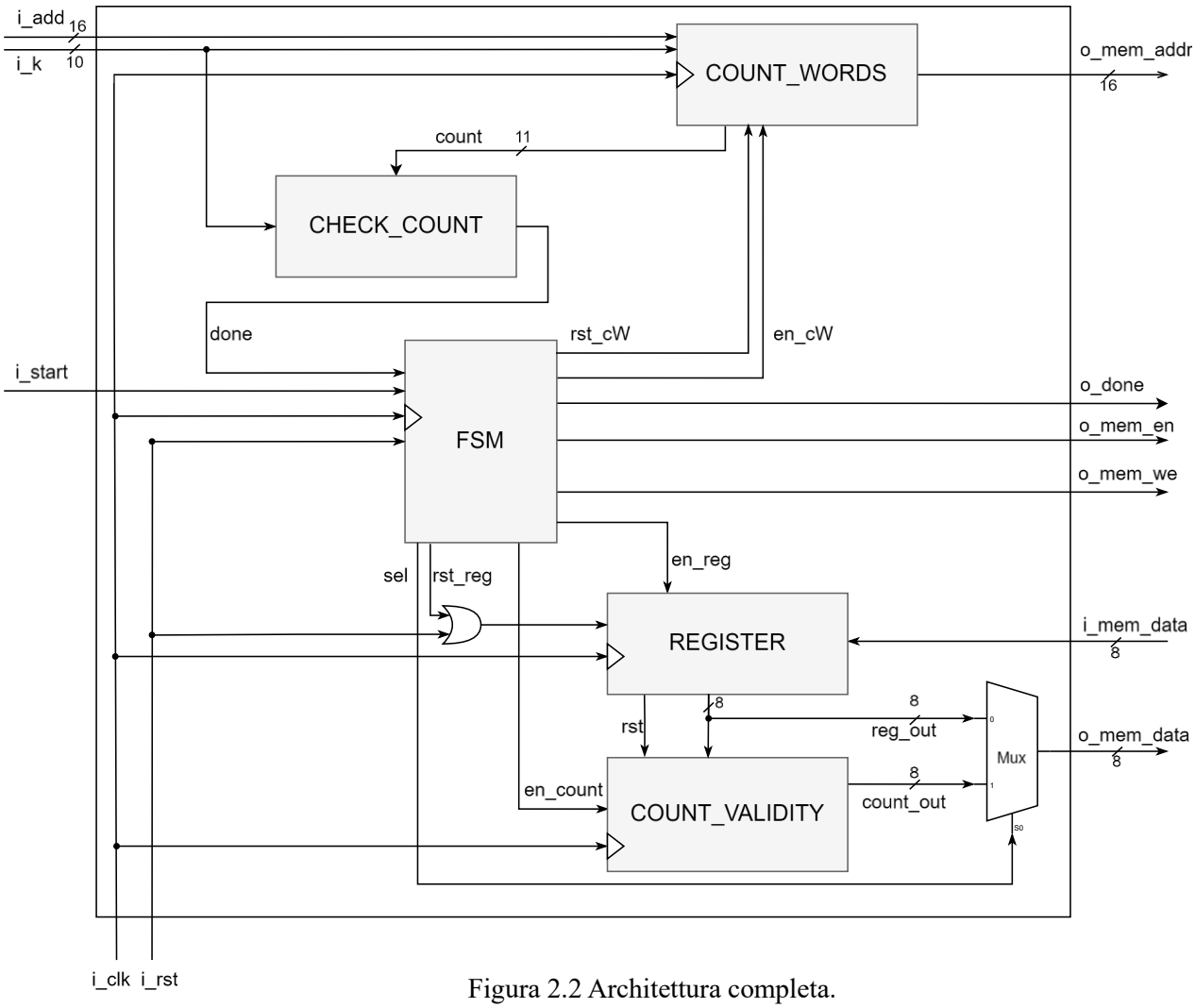


Figura 2.2 Architettura completa.

2.1 COUNT WORDS

Un contatore sincrono, descritto tramite dataflow e un process, che riceve in ingresso un indirizzo (**i_add**), un numero a 10 (**i_k**) e, se abilitato (**en** = 1), conta fino a 2 * **i_k**. Raggiunto il valore massimo smette di contare in attesa di un reset. L'uscita **count** è il conteggio effettuato, mentre **o_mem_addr** è il risultato della somma tra **i_add** e count.

In questa architettura, **en** e **i_rst** derivano da 2.5, **count** è collegato a 2.2.

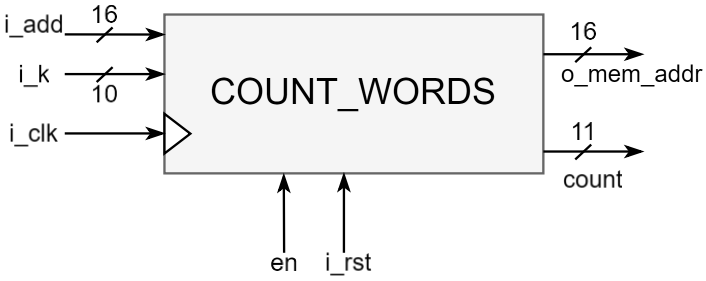


Figura 2.3

2.2 CHECK COUNT

Figura 2.4

Un componente asincrono, interamente descritto in dataflow, che riceve in ingresso un numero a 10 bit (**i_k**), un segnale **count** a 11 bit (in questo schema derivante da 2.1) pone l'uscita **done** a 1 se **count** >= 2 * **i_k**, 0 altrimento. Il suo scopo è quello di mandare un segnale di done a 1 asincronamente nel caso in cui **i_k** fosse nullo. Negli altri casi viene pilotato da **count** che in questo caso è l'output di un componente sincrono, quindi l'uscita **done** viene sincronizzata.

In questa architettura **count** deriva da 2.1; **done** e collegato a 2.5.

2.3 REGISTER

Un registro sincrono, descritto tramite un process, che riceve dati a 8 bit (**i_mem_data**) e, se abilitato, pilota le uscite in base al dato letto:

- lettura valida: **o_mem_data** assume il dato letto; **rst** viene posto a 1;
- lettura nulla: **o_mem_data** assume l'ultimo valore valido; **rst** viene posto a zero;

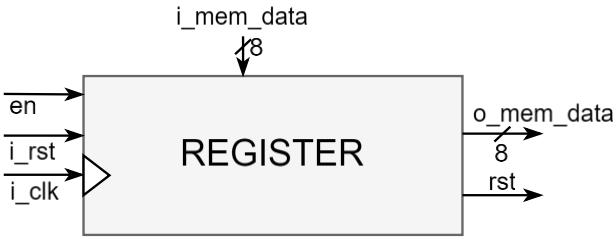


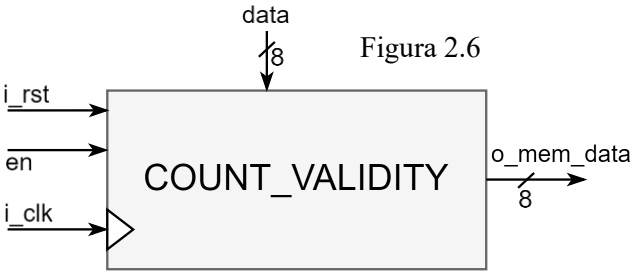
Figura 2.5

In questa architettura **en** e **i_rst** derivano da 2.5; **rst** è collegato a 2.4; **o_mem_data** è collegato al multiplexer e a 2.4.

2.4 COUNT VALIDITY

Contatore sincrono, descritto tramite un process. Se il segnale **i_rst** = 1 pone l'uscita **o_mem_data** a 31 (valore massimo di credibilità stabilito).

Figura 2.6



Se **en** = 1:

- **data** = 0: **o_mem_data** = 0;
- **data** diverso da 0: **o_mem_data** viene decrementato di uno (sempre se il risultato di questa sottrazione è ≥ 0);

In questa architettura **en** deriva da 2.5, **i_rst** e **data** derivano da 2.3; **o_mem_data** è collegato al multiplexer.

2.5 FSM

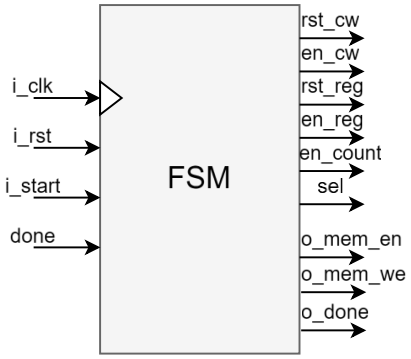


Figura 2.7

Macchina a stati finiti descritta tramite una collezione di process. La scelta implementativa consiste in una macchina di Mealy a 4 stati, frutto della semplificazione di una macchina di Moore a 6 stati. (per la semplificazione si riporta all'appendice A)

La macchina la un segnale **i_rst** asincrono che imposta come stato corrente **S_0**, gli archi che indicano questi passaggi sono stati omessi per comodità di lettura.

In questa architettura **done** deriva da 2.2; **rst_cW**, **en_cW** sono collegati a 2.1; **rst_reg**, **en_reg** sono collegati a 2.3; **en_count** è collegato a 2.4; **sel** è collegato al multiplexer.

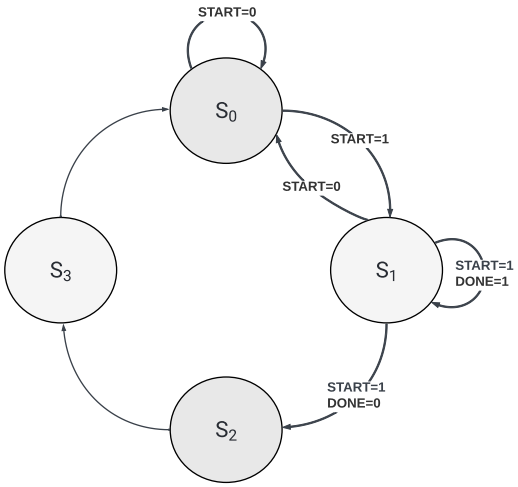


Figura 2.8 FSM.

Nello specifico:

Stato	Descrizione
S_0	Incorpora l'originale stato di reset nel quale attendo finché i_start è diverso da 1 e quello di scrittura della credibilità in memoria.
S_1	Incorpora lo stato di abilitazione della lettura e quello di attesa della discesa del segnale i_start a zero dopo aver abilitato o_done.
S_2	Stato arrivato al quale il valore W è stato letto.
S_3	Stato arrivato al quale il nuovo valore W è stato scritto.

Tabella 2.2 Tabella descrizione stati.

I segnali:

	START=0	START=1	START=0	START=1 DONE=0	START=1 DONE=1		
	S_0		S_1			S_2	S_3
o_mem_we	0		0	1	0	1	1
o_mem_en	0	1	0	1	0	1	1
o_done	0		0	0	1	0	0
sel	0		0	0	0	0	1
en_reg	0		0	1	0	0	0
en_c32	0		0	0	0	1	0
en_cW	0		0	0	0	1	1
rst_cW	1	0	1	0	0	0	0
rst_reg	0		0	0	1	0	0

Figura 2.3 Tabella delle transizioni.

RISULTATI SPERIMENTALI

La rete è stata sviluppata usando una FPGA xc7a200tfbg484-1 (famiglia Artix-7).

3.1 SINTESI

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	57	0	0	134600	0.04
LUT as Logic	57	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	32	0	0	269200	0.01
Register as Flip Flop	32	0	0	269200	0.01
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figura 3.1 Report utilization.

Dal report di utilizzazione possiamo notare che, come da progettazione, non sono presenti latch e sono utilizzati 32 FF e 57 Look Up Table solo di natura logica. La quota di componenti usata della FPGA scelta è irrisoria: 0.04% delle LUT disponibili e 0.01% dei FF.

Il report di timing invece mostra come lo slack (calcolato come la differenza tra il tempo necessario al segnale per attraversare la rete e il tempo disponibile per farlo) soddisfi i vincoli imposti dal clock:

- Requisiti: T = 20ns e duty cycle del 50%.
- Slack time:

required time	21.834
arrival time	-6.795
slack	15.040

Slack soddisfatto vuol dire che il segnale non arriva né troppo presto (setup violation) né troppo tardi (hold violation). Questo è necessario perché nella simulazione post-sintesi viene simulato anche il ritardo effettivo del componente utilizzato.

3.2 SIMULAZIONI

Dopo aver progettato e realizzato la macchina tramite VHDL ho simulato il comportamento prima su un testbench contenente un solo scenario per verificarne il funzionamento e successivamente su file contenenti numerosi scenari realizzati con l'ausilio di un generatore che ho scritto in python.

Dopo aver quindi testato il componente su tantissime sequenze casuali, sono passato al controllo dei casi limite:

I. Sequenza di lunghezza nulla: consente di verificare che il componente 2.2 ricevendo **count** = 0 dal 2.1 funzioni correttamente e mandi **o_done** = 1 alla FSM così da farla terminare subito.
II. Sequenza di lunghezza 1023 (9 bit di k tutti a 1): consente di verificare che il conteggio raggiunga il valore corretto di 2***i_k** nonostante **i_k** sia su 9 bit e il risultato su 10.

III. Sequenza di tutti zeri: consente di verificare che il registro 2.3 dopo un reset se legge uno zero scriva correttamente 0 in memoria e piloti il 2.4 in modo da scrivere 0 anche come indice di credibilità.
IV. Sequenza contenente più di 30 zero consecutivi: consente di azzerare l'indice di credibilità così da verificare che il conteggio non diventi negativo.
V. Sequenza che inizia all'indirizzo 0/termina all'indirizzo 65535: due casi in cui, avendo il vincolo della fine della memoria, controllo che il conteggio parta e si concluda dove previsto senza letture/scritture indesiderate in zone non controllate della memoria.

Una volta verificato il corretto funzionamento di questi casi sono passato invece all'utilizzo del messaggio asincrono di reset: quando **i_start** è nullo, appena viene posto a 1, **o_done** viene posto a 1 e in numerosi momenti casuali dell'esecuzione.

Il componente si è sempre comportato come previsto sia in simulazione comportamentale, sia in quella di post-sintesi. L'unico problema riscontrato è il caso in cui dato **i_k** e **i_add**, **i_add**+2***i_k** sfori la dimensione della memoria, ma consisterebbe in un uso scorretto del componente.

CONCLUSIONI

L'architettura progettata rispetta le specifiche date: ingressi, uscite e segnale di clock. La sintesi non ha richiesto l'utilizzo di latch (che avrebbero potuto causare comportamenti indesiderati dovuti a ritardi di propagazione e mancanza di sincronizzazione) e ha superato tutti i casi di test presentati nel capitolo 3.2 sia in pre che post-sintesi, lasciando quindi ipotizzare che il dispositivo funzioni correttamente.

Una possibile ottimizzazione sarebbe quella di decidere di scrivere in memoria solamente quando viene fatta una lettura nulla invece di sovrascriverlo ogni volta; questa scelta però richiederebbe un aumento nel numero di stati della macchina e quindi un'ulteriore analisi dei benefici.

Appendice A

Macchina originale: $M = (Q, \sum, \delta, \lambda, q0)$

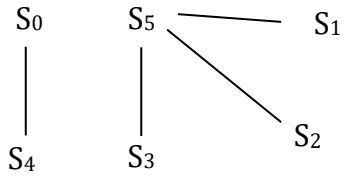
	START-DONE				OUTPUT								
	00	01	10	11	o_mem_we	o_mem_en	o_done	sel	en_reg	en_c32	en_cW	rst_cW	rst_reg
S_0	S_0	S_0	S_1	S_1	0	0	0	0	0	0	0	1	0
S_1	X	X	S_2	S_5	0	1	0	0	0	0	0	0	0
S_2	X	X	S_3	X	1	1	0	0	1	0	0	0	0
S_3	X	X	S_4	X	1	1	0	0	0	1	1	0	0
S_4	X	X	S_1	S_1	1	1	0	1	0	0	1	0	0
S_5	S_0	S_0	S_5	S_5	0	0	1	0	0	0	0	0	1

Tabella A.1 Macchina di Moore

Con: S_0 come stato di reset; S_1 stato raggiunto alla salita del segnale di start; S_2 stato in cui il valore di W è stato letto; S_3 stato in cui sovrascrivo W; S_4 scrive la credibilità; S_5 stato di attesa discesa del segnale di start

Macchina di Mealy equivalente: $M' = (Q, \sum, \delta, \lambda', q0)$ con $\lambda'(q, a) = \lambda(\delta(q, a))$ (Risultato Tabella A.1)

S_1	X				
S_2	X	X			
S_3	X	X	X		
S_4	~	X	X	X	
S_5	X	~	~	~	X
	S_0	S_1	S_2	S_3	S_4



α: S_0; S_4
β: S_1; S_5
γ: S_2
δ: S_3

Tabella A.2 Tabella di compatibilità

Risultato della semplificazione Tabella A.2

START	DONE			S_0	S_1	S_2	S_3	S_4	S_5
0	0	NEXT		S_0	X	X	X	X	S_0
		OUT	o_mem_we	0	X	X	X	X	0
			o_mem_en	0	X	X	X	X	0
			o_done	0	X	X	X	X	0
			sel	0	X	X	X	X	0
			en_reg	0	X	X	X	X	0
			en_c32	0	X	X	X	X	0
			en_cW	0	X	X	X	X	0
			rst_cW	1	X	X	X	X	1
			rst_reg	0	X	X	X	X	0
0	1	NEXT		S_0	X	X	X	X	S_0
		OUT	o_mem_we	0	X	X	X	X	0
			o_mem_en	0	X	X	X	X	0
			o_done	0	X	X	X	X	0
			sel	0	X	X	X	X	0
			en_reg	0	X	X	X	X	0
			en_c32	0	X	X	X	X	0
			en_cW	0	X	X	X	X	0
			rst_cW	1	X	X	X	X	1
			rst_reg	0	X	X	X	X	0
1	0	NEXT		S_1	S_2	S_3	S_4	S_1	X
		OUT	o_mem_we	0	1	1	1	0	X
			o_mem_en	1	1	1	1	1	X
			o_done	0	0	0	0	0	X
			sel	0	0	0	1	0	X
			en_reg	0	1	0	0	0	X
			en_c32	0	0	1	0	0	X
			en_cW	0	0	1	1	0	X
			rst_cW	0	0	0	0	0	X
			rst_reg	0	0	0	0	0	X
1	1	NEXT		S_1	S_5	X	X	S_1	S_5
		OUT	o_mem_we	0	0	X	X	0	0
			o_mem_en	1	0	X	X	1	0
			o_done	0	1	X	X	0	1
			sel	0	0	X	X	0	0
			en_reg	0	0	X	X	0	0
			en_c32	0	0	X	X	0	0
			en_cW	0	0	X	X	0	0
			rst_cW	0	0	X	X	0	0
			rst_reg	0	1	X	X	0	1

Tabella A.3 Macchina di Mealy equivalente.

START	DONE			α	β	γ	δ
0	0	NEXT		α	α	X	X
		OUT	o_mem_we	0	0	X	X
			o_mem_en	0	0	X	X
			o_done	0	0	X	X
			sel	0	0	X	X
			en_reg	0	0	X	X
			en_c32	0	0	X	X
			en_cW	0	0	X	X
			rst_cW	1	1	X	X
			rst_reg	0	0	X	X
0	1	NEXT		α	α	X	X
		OUT	o_mem_we	0	0	X	X
			o_mem_en	0	0	X	X
			o_done	0	0	X	X
			sel	0	0	X	X
			en_reg	0	0	X	X
			en_c32	0	0	X	X
			en_cW	0	0	X	X
			rst_cW	1	1	X	X
			rst_reg	0	0	X	X
1	0	NEXT		β	γ	δ	α
		OUT	o_mem_we	0	1	1	1
			o_mem_en	1	1	1	1
			o_done	0	0	0	0
			sel	0	0	0	1
			en_reg	0	1	0	0
			en_c32	0	0	1	0
			en_cW	0	0	1	1
			rst_cW	0	0	0	0
			rst_reg	0	0	0	0
1	1	NEXT		β	β	X	X
		OUT	o_mem_we	0	0	X	X
			o_mem_en	1	0	X	X
			o_done	0	1	X	X
			sel	0	0	X	X
			en_reg	0	0	X	X
			en_c32	0	0	X	X
			en_cW	0	0	X	X
			rst_cW	0	0	X	X
			rst_reg	0	1	X	X

Tabella A.4 Macchina di Mealy semplificata