



---

Tra il comando *print\_stats* è possibile osservare come il numero di letterali sia sceso da 132 a 60 per  $Y_2$  e da 204 a 96 per  $Y_1$ . Per quanto riguarda  $Y_0$ , invece, lo strumento SIS non è stato in grado di ridurre il numero di letterali, dunque la funzione è già in forma minima.

Per quanto concerne la procedura di minimizzazione manuale, si è utilizzato il metodo di **Mc-Cluskey** sulle tre uscite separate. Confrontando le due soluzioni, si è notato che i numeri di letterali ottenuti in entrambi i casi sono coerenti tra loro. Si è inoltre notato come l'uscita  $Y_0$  ancora una volta non sia stata alterata durante il processo di minimizzazione, a differenza delle altre due uscite che risultano notevolmente ridotte in entrambi i casi. Ciò è perfettamente compatibile con il risultato ottenibile attraverso Mc-Cluskey (fig.3), dove si nota chiaramente che per l'uscita  $Y_0$  non vengono generati consensi: questo è dovuto al fatto che non ci siano classi adiacenti aventi distanza di Hamming pari a 1, pertanto è impossibile che vengano generati consensi.

Per il mapping tecnologico, si è utilizzata la libreria *mcnc.genlib*, contenente le caratteristiche di ogni porta in termini di equazioni, area e ritardi. Come riportato in fig.4, sono stati effettuati diversi esperimenti variando la funzione di costo rispetto alla quale viene ottimizzata in base alla tecnologia scelta per il mapping. Ciò è stato fatto utilizzando l'opzione **-m** del comando **map**: in particolare, con -m 1 si è preferito ottimizzare il ritardo, con -m 0 l'area, mentre con -m 0.5 si è effettuata un mapping più bilanciato.

I risultati ottenuti sono perfettamente coerenti con quanto stabilito: nel primo caso, ottimizzando il ritardo, lo slack negativo totale è di -41.13, ma l'area totale risulta essere 196. Nel secondo caso invece, ottimizzando l'area, questa risulta essere scesa a 159, ma lo slack negativo totale raggiunge -53.00. Nel terzo caso infine, dove si è scelta una mediazione tra tempo e area, si è ottenuta una rete la cui area e slack negativo totale si assestano ad un valore "intermedio" rispetto ai due casi precedenti: in particolare, la rete avrà un'area di 169 e uno slack negativo totale pari a -46.50.

### 0.1.3 Esercizio 3

Si calcoli la forma minima della macchina M come rete multi-uscita utilizzando lo strumento SIS e si disegni il grafo corrispondente.

Per effettuare quest'operazione è stato possibile scegliere tra i diversi algoritmi visti a lezione in grado di minimizzare una funzione a due livelli multiuscita fornendoci una funzione a due livelli o multilivello. Si è deciso di utilizzare lo script *rugged.script*, in grado di operare sia su funzioni multilivello che a due livelli applicando una serie di trasformazioni prestabilite e fornendo, in uscita, una funzione multilivello che ben si presta alla rappresentazione grafica mediante grafo. In fig.5 è possibile osservare il risultato.

Si noti come minimizzando tutte le uscite contemporaneamente, e dunque grazie al riutilizzo di alcuni dei nodi della rete per la realizzazione di più uscite, il numero totale di letterali sia sceso a 59, mentre nel caso della minimizzazione delle uscite separate si erano ottenuti 60, 96 e 192 letterali rispettivamente per  $Y_2$ ,  $Y_1$  e  $Y_0$ .

Il grafo ottenuto da questo risultato è consultabile in fig.6.

### 0.1.4 Esercizio 4

Si implementi la macchina M, nella forma ottenuta al punto 3, in VHDL seguendo una modalità di descrizione di tipo "data-flow".

Di seguito è riportata l'implementazione in VHDL della macchina M. Si noti come, descrivendo la macchina in modalità data-flow, sono stati riportati i nodi forniti da *rugged.script* come segnali d'appoggio da utilizzare per la realizzazione di  $Y_2$ ,  $Y_1$  e  $Y_0$ . Sono stati inoltre utilizzati dei segnali temporanei d'uscita *y2\_temp*, *y1\_temp* e *y0\_temp* per permettere la definizione di  $Y_2$  in funzione di  $Y_0$  e di  $Y_1$  in funzione di  $Y_2$ . Il codice è disponibile qui: *M\_dataflow.vhd*.

Si è poi proceduto alla realizzazione di un *testbench* per simulare la macchina tramite il tool *GHDL*. In tale testbench, i sei ingressi vengono portati da 0 ad 1 a distanza di 10 ns da una transizione all'altra. Il risultato è visibile in fig.7.

```
UC Berkeley, SIS 1.3 (compiled 20-Mar-00 at 1:56 PM)
sis> read_blif Esercitazione1_2_Y2.blif
sis> simplify
sis> write eqn
INORDER = X5 X4 X3 X2 X1 X0;
OUTORDER = Y2;
Y2 = X3*X2*X1*X0 + X4*X2*X1*X0 + X5*X2*X1*X0 + X4*X3*X1*X0 + X5*X3*X1*X0 + X5*
X4*X1*X0 + X4*X3*X2*X0 + X5*X3*X2*X0 + X5*X4*X2*X0 + X5*X4*X3*X0 + X4*X3*X2*X1
+ X5*X3*X2*X1 + X5*X4*X2*X1 + X5*X4*X3*X1 + X5*X4*X3*X2;
sis> read_blif Esercitazione1_2_Y1.blif
sis> simplify
sis> write eqn
INORDER = X5 X4 X3 X2 X1 X0;
OUTORDER = Y1;
Y1 = X5*X4*X3*X2*X1*X0 + X5*X4!*X2!*X1!*X0 + X5!*X3*X2!*X1!*X0 + !X4*X3*X2!*X1!
X0 + !X5*X3*X2!*X1!*X0 + !X4!*X3*X2*X1*X0 + !X5*X4*X3*X1*X0 + X5!*X4*X3!*X2!
X0 + !X5*X4*X3*X1*X2*X0 + !X5*X4*X3*X2*X1*X0 + X5!*X4*X1*X2*X0 + !X5*X4*X1*X2!*
X1*X0 + !X4*X1*X3*X2*X1*X0 + !X5*X3*X2*X1*X0 + !X5*X4*X3*X1*X1*X0 + !X5!*X4!*
X2*X1*X0 + !X5!*X4!*X3*X1*X0 + X5!*X4!*X3!*X2*X1 + !X5*X4!*X3!*X2*X1;
sis> read_blif Esercitazione1_2_Y0.blif
sis> simplify
sis> write eqn
INORDER = X5 X4 X3 X2 X1 X0;
OUTORDER = Y0;
Y0 = X5!*X4!*X3!*X2!*X1!*X0 + !X5*X4!*X3!*X2!*X1!*X0 + !X5!*X4*X3!*X2!*X1!*X0
+ X5*X4*X3*X2!*X1!*X0 + !X5!*X4*X3*X2!*X1!*X0 + X5*X4*X3*X2*X1*X0 + X5!*
X4*X3*X2!*X1!*X0 + !X5*X4*X3*X2*X1*X0 + !X5!*X4!*X3*X2*X1*X0 + X5*X4*X3*X1
X2*X1*X0 + X5*X4*X3*X2*X1*X0 + !X5*X4*X3*X1*X2*X1*X0 + X5!*X4!*X3*X2*X1*X0
+ !X5*X4*X3*X2*X1*X0 + !X5!*X4*X3*X2*X1*X0 + X5*X4*X3*X2*X1*X0 + !X5!*X4!*
X3*X2!*X1*X0 + X5*X4*X3*X2!*X1*X0 + X5!*X4*X3*X2!*X1*X0 + !X5*X4*X3*X2!*X1*
X0 + X5!*X4*X1*X3*X2*X1*X0 + !X5*X4!*X3*X2!*X1*X0 + !X5!*X4*X3*X2*X1*X0 +
X5*X4*X3*X2*X1*X0 + X5!*X4!*X3*X2*X1*X0 + !X5*X4*X3*X2*X1*X0 + !X5!*X4*X3*X1
X2*X1*X0 + X5*X4*X3*X1*X2*X1*X0 + !X5!*X4!*X3*X2*X1*X0 + X5*X4!*X3*X2*X1*X0 + X5*
X4*X3*X2*X1*X0 + !X5*X4*X3*X2*X1*X0;
```

Figura 2: Stampa della uscite semplificate separatamente in SIS.

	$x_5$	$x_4$	$x_3$	$x_2$	$x_1$	$x_0$	
1:	0	0	0	0	0	1	✓
2:	0	0	0	0	1	0	✓
4:	0	0	0	1	0	0	✓
8:	0	0	1	0	0	0	✓
16:	0	1	0	0	0	0	✓
32:	1	0	0	0	0	0	✓
7:	0	0	0	1	1	1	✓
11:	0	0	1	0	1	1	✓
13:	0	0	1	1	0	1	✓
14:	0	0	1	1	1	0	✓
19:	0	1	0	0	1	1	✓
21:	0	1	0	1	0	1	✓
22:	0	1	0	1	1	0	✓
25:	0	1	1	0	0	1	✓
26:	0	1	1	0	1	0	✓
28:	0	1	1	1	0	0	✓
35:	1	0	0	0	1	1	✓
37:	1	0	0	1	0	1	✓
38:	1	0	0	1	1	0	✓
41:	1	0	1	0	0	1	✓
42:	1	0	1	0	1	0	✓
44:	1	0	1	1	0	0	✓
49:	1	1	0	0	0	1	✓
50:	1	1	0	0	1	0	✓
52:	1	1	0	1	0	0	✓
56:	1	1	1	0	0	0	✓
31:	0	0	1	1	1	1	✓
47:	1	0	1	1	1	1	✓
55:	1	1	0	1	1	1	✓
59:	1	1	1	0	1	1	✓
61:	1	1	1	1	0	1	✓
62:	1	1	1	1	1	0	✓

Figura 3: Minimizzazione di  $Y_0$  con Mc-Cluskey.

```

sis> read_blif Esercitazione1_2.blif
sis> read_library ~/sis-1.3/sis/sis_lib/mcnc.genlib
sis> map -W -m 1 -s
>>> before removing serial inverters <<<
# of outputs: 3
total gate area: 244.00
maximum arrival time: (16.41,16.41)
maximum po slack: (-13.21,-13.21)
minimum po slack: (-16.41,-16.41)
total neg slack: (-43.76,-43.76)
# of failing outputs: 3
>>> before removing parallel inverters <<<
# of outputs: 3
total gate area: 196.00
maximum arrival time: (15.51,15.51)
maximum po slack: (-12.31,-12.31)
minimum po slack: (-15.51,-15.51)
total neg slack: (-41.13,-41.13)
# of failing outputs: 3
# of outputs: 3
total gate area: 196.00
maximum arrival time: (15.51,15.51)
maximum po slack: (-12.31,-12.31)
minimum po slack: (-15.51,-15.51)
total neg slack: (-41.13,-41.13)
# of failing outputs: 3
sis> print_map_stats
Total Area = 196.00
Gate Count = 72
Buffer Count = 0
Inverter Count = 6
Most Negative Slack = -15.51
Sum of Negative Slacks = -41.13
Number of Critical PO = 3

sis> map -W -m 0 -s
>>> before removing serial inverters <<<
# of outputs: 3
total gate area: 161.00
maximum arrival time: (21.50,21.50)
maximum po slack: (-14.30,-14.30)
minimum po slack: (-21.50,-21.50)
total neg slack: (-53.20,-53.20)
# of failing outputs: 3
>>> before removing parallel inverters <<<
# of outputs: 3
total gate area: 161.00
maximum arrival time: (21.50,21.50)
maximum po slack: (-14.30,-14.30)
minimum po slack: (-21.50,-21.50)
total neg slack: (-53.20,-53.20)
# of failing outputs: 3
# of outputs: 3
total gate area: 159.00
maximum arrival time: (21.30,21.30)
maximum po slack: (-14.30,-14.30)
minimum po slack: (-21.30,-21.30)
total neg slack: (-53.00,-53.00)
# of failing outputs: 3
sis> print_map_stats
Total Area = 159.00
Gate Count = 52
Buffer Count = 0
Inverter Count = 9
Most Negative Slack = -21.30
Sum of Negative Slacks = -53.00
Number of Critical PO = 3

sis> map -W -m 0.5 -s
>>> before removing serial inverters <<<
# of outputs: 3
total gate area: 223.00
maximum arrival time: (19.10,19.10)
maximum po slack: (-15.50,-15.50)
minimum po slack: (-19.10,-19.10)
total neg slack: (-50.10,-50.10)
# of failing outputs: 3
>>> before removing parallel inverters <<<
# of outputs: 3
total gate area: 169.00
maximum arrival time: (17.90,17.90)
maximum po slack: (-14.30,-14.30)
minimum po slack: (-17.90,-17.90)
total neg slack: (-46.50,-46.50)
# of failing outputs: 3
# of outputs: 3
total gate area: 169.00
maximum arrival time: (17.90,17.90)
maximum po slack: (-14.30,-14.30)
minimum po slack: (-17.90,-17.90)
total neg slack: (-46.50,-46.50)
# of failing outputs: 3
sis> print_map_stats
Total Area = 169.00
Gate Count = 58
Buffer Count = 0
Inverter Count = 10
Most Negative Slack = -17.90
Sum of Negative Slacks = -46.50
Number of Critical PO = 3

```

Figura 4: Mapping tecnologico effettuato tramite libreria *mcnc.genlib* fornendo come parametri di bilanciamento, rispettivamente, 1, 0 e 0.5.

```

sis> write_eqn
INORDER = X5 X4 X3 X2 X1 X0;
OUTORDER = Y2 Y1 Y0;
Y2 = X5*!Y0*[10] + X4*[8]*[9] + X4*[6]*[7] + [7]*[9];
Y1 = X4*! [3]*! [6]*! [10] + X4*! [3]*[9]*[10] + !Y2*!Y0*[8] + !Y2*[3]*[6] + !Y2*[10] + !Y2*[9];
Y0 = ! [5]*[6]*! [9] + [5]*[9] + [5]*! [6];
[3] = X5*!X4 + !X5*X4;
[4] = !X3*! [3] + X3*[3];
[5] = !X2*! [4] + X2*[4];
[6] = X0 + X1;
[7] = X5*!Y0 + [10];
[8] = X2 + X3;
[9] = X1*X0;
[10] = X3*X2;
esercizio1_2 pi= 6 po= 3 nodes= 11 latches= 0
lits(sop)= 59
sis>

```

Figura 5: Risultato della minimizzazione con *rugged.script*.

### 0.1.5 Esercizio 5

Si progetti la macchina M per composizione di macchine a partire da blocchi full-adder, e si implementi la soluzione trovata in VHDL.

Ricordando che un full-adder è in grado di sommare 3 bit riportando in uscita il bit meno significativo  $s_i$  e quello più significativo  $r_i$ , possiamo procedere come segue: scomponendo la somma di 6 bit in due somme di 3 bit, effettuabili tramite 2 full-adder, otterremo due somme parziali  $s_0$  e  $s_1$  che andranno a loro volta sommate tra loro per ottenere il bit meno significativo dell'uscita  $y_0$ . Per quanto riguarda i riporti  $r_0$  e  $r_1$ , aventi entrambi peso 1, questi andranno sommati tra loro tenendo anche conto del riporto ottenuto calcolando  $y_0$  (ossia  $r_2$ , di peso 1). Il risultato di questa ultima operazione di somma sarà la cifra di peso 1 ( $y_1$ ) della nostra soluzione, mentre il riporto sarà la cifra di peso 2 ( $y_2$ ). Usando dunque 4 full-adder, lo schema ottenuto è consultabile in fig.8.

Per quanto concerne l'implementazione in VHDL, si è dapprima proceduto all'implementazione di un full-adder seguendo una modalità di descrizione di tipo "behavioural".

```
1  entity full_adder is
2  Port ( X : in  STD_LOGIC;
3        Y : in  STD_LOGIC;
4        CIN : in  STD_LOGIC;
5        S : out STD_LOGIC;
6        C : out STD_LOGIC
7  );
8  end full_adder;
9  architecture dataflow of full_adder is
10 begin
11     S <= (X xor Y xor CIN);
12     C <= ((X and Y) or ((X xor Y) and CIN));
13 end dataflow;
```

Codice Componente 1: Implementazione in VHDL di un full-adder.

Dopodiché, utilizzando questi componenti, si è proceduto a costruire la macchina M seguendo una modalità di descrizione di tipo "structural". Il codice è visualizzabile qui: M.vhd. Il risultato della simulazione è analogo a quello dell'esercizio 4.

### 0.1.6 Esercizio 6

Si progetti una macchina S che, date 6 stringhe di 3 bit ciascuna in ingresso (A, B, C, D, E, F), rappresentanti la codifica binaria di numeri interi positivi, ne calcoli la somma W espressa su 6 bit. La macchina S deve essere progettata per composizione di macchine utilizzando la macchina M progettata al punto 5) e componenti full-adder, opportunamente collegati.

Come descritto nell'esercizio 5, la macchina M è in grado di determinare, dati 6 bit in ingresso, il numero di bit alti. Dal momento che si può considerare tale macchina come sommatore in grado di sommare 6 bit, si è deciso di utilizzarla per sommare tra loro le cifre dello stesso peso delle 6 stringhe fornite in ingresso alla macchina S. Essendo tali stringhe composte da 3 bit ciascuna (di peso 2, 1 e 0), si è scelto di usare 3 macchine M per sommare le cifre di stesso peso tra loro. Una volta ottenute tali somme (ciascuna, rispettivamente, espressa su 3 bit in codifica binaria), si è proceduto con tali osservazioni: il bit di peso 0 della somma dei 6 bit di peso 0 non è altro che la cifra di peso 0 del risultato della macchina S, ossia della somma delle 6 stringhe. Il bit di peso 1 della stessa somma, invece, rappresenta invece un bit di peso 1 della somma totale delle stringhe, e lo stesso ragionamento è valido per il bit di peso 2. Passando alla somma dei 6 bit di peso 1 delle stringhe di partenza, si noti come la cifra di peso 0 di tale somma non è altro che un bit di peso 1 della somma totale delle stringhe, mentre la cifra di peso 1 è un bit di peso 2 per la somma totale, e così via.

Seguendo questo ragionamento, è stato possibile combinare le cifre delle somme di peso analogo utilizzando dei full-adder, ottenendo lo schema consultabile in fig.9.

Dopodiché si è proceduto alla sua realizzazione in VHDL utilizzando una modalità di descrizione "structural". Il codice è visualizzabile qui: S.vhd.

Il risultato della simulazione è riportato in fig.10.

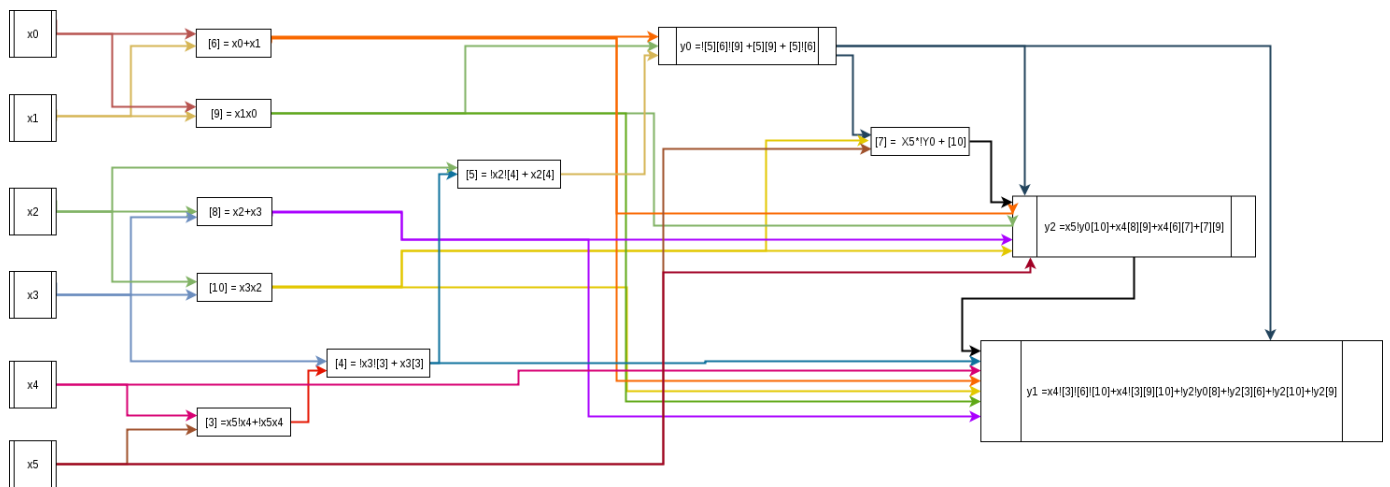


Figura 6: Grafo della funzione minimizzata tramite *rugged.script*.

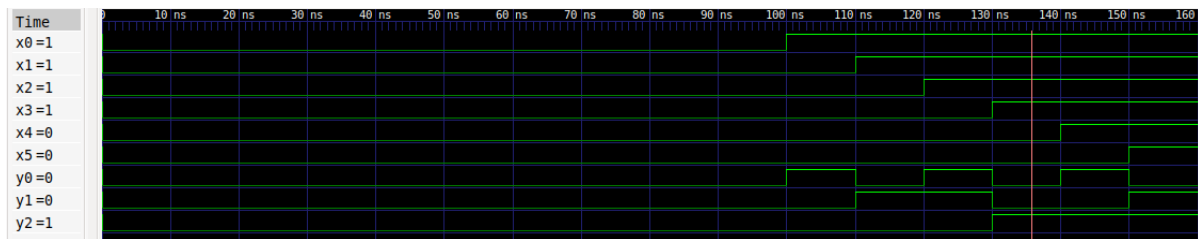


Figura 7: Simulazione della macchina M in *gtkwave*.

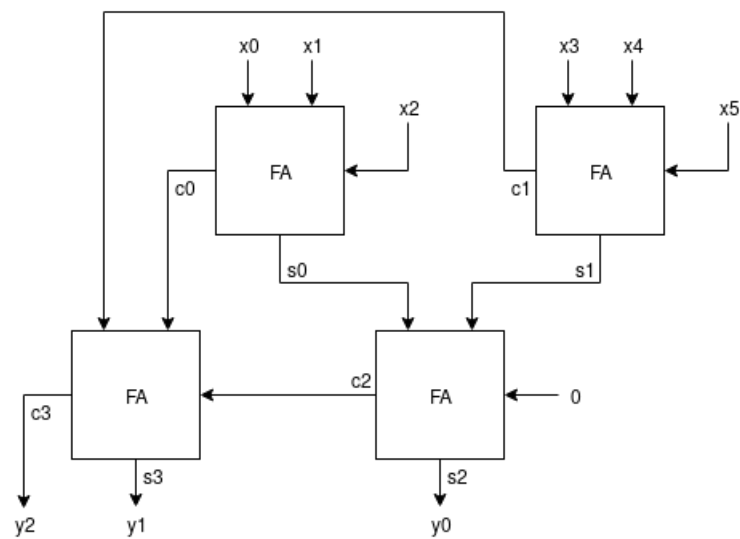


Figura 8: Schema della macchina M a partire da blocchi full-adder.

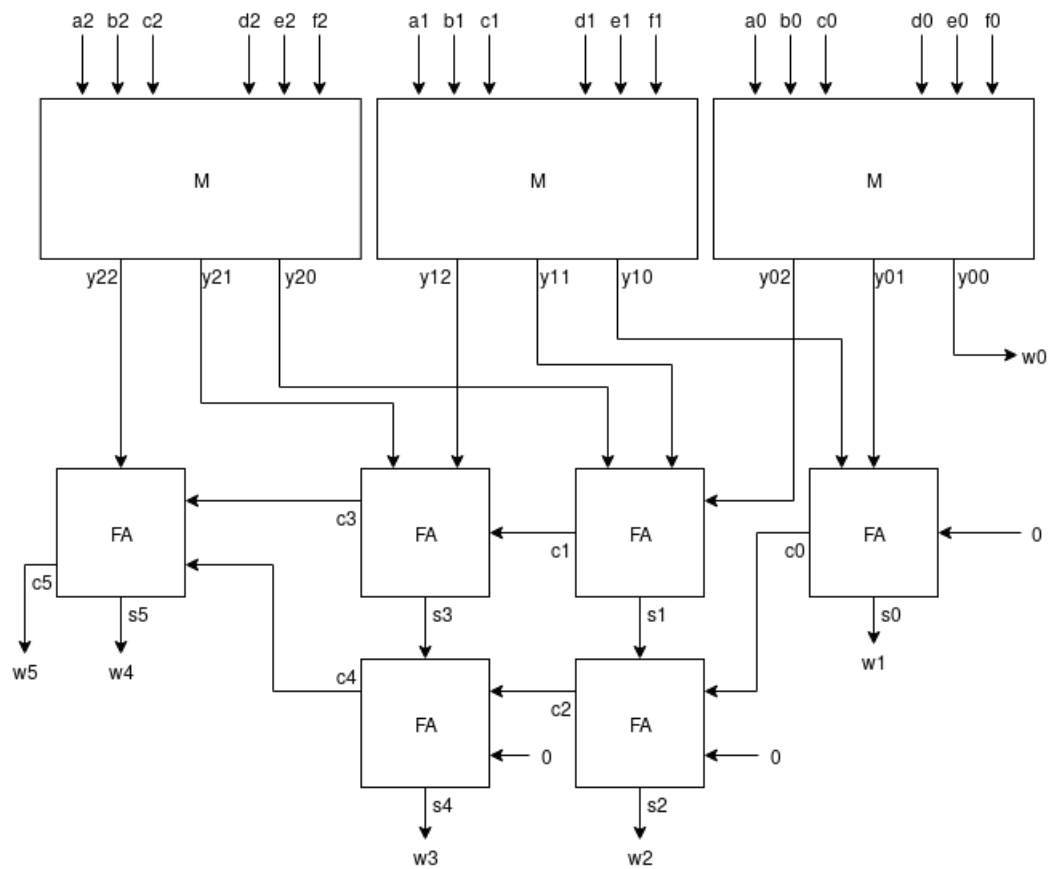


Figura 9: Schema della macchina S.

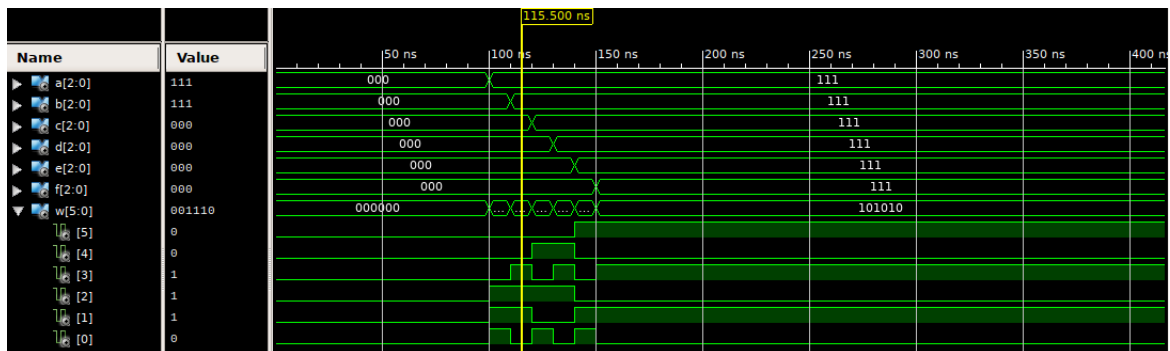


Figura 10: Risultato della simulazione della macchina S.