Tesina di Architettura dei Sistemi di Elaborazione
Gruppo 14

Gabriele Previtera - Mat. M63/000834 Mirko Pennone - Mat. M63/000876
Simone Penna - Mat. M63/000858

February 8, 2019

Contents

Chapter 1

Sintesi di macchine combinatorie

1.1 Traccia

Eseguire gli esercizi riportati nel documento fornito.

1.2 Soluzione

1.2.1 Esercizio 1

Si progetti una macchina M che, data una parola X di 6 bit in ingresso ($X_5X_4X_3X_2X_1X_0$), restituisca una parola Y di 3 bit ($Y_2Y_1Y_0$) che rappresenta la codifica binaria del **numero di bit alti in X.**

Utilizzando una rappresentazione 4-2-1 per l'uscita **Y**, si riportano gli ON-SET ottenuti per ogni uscita:

Y_2: ON-SET = {15, 23, 27, 29, 30, 31, 39, 43, 45, 46, 47, 51, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63};
Y_1: ON-SET = {3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 24, 25, 26, 28, 33, 34, 35, 36, 37, 38, 40, 41, 42, 44, 48, 52, 56, 63};
Y_0: ON-SET = {1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 31, 32, 35, 37, 38, 41, 42, 44, 47, 49, 50, 52, 55, 56, 59, 61, 62}.

1.2.2 Esercizio 2

Si derivi la forma minima (**SOP**) per ciascuna delle variabili in uscita dalla macchina M (considerate separatamente l'una dall'altra) utilizzando lo strumento **SIS**, e si confronti la soluzione trovata dal tool con quella ricavabile con una procedura esatta manuale (Karnaugh o Mc-Cluskey). Per una delle uscite si effettui anche il **mapping** su una delle librerie disponibili in SIS e si commentino i risultati ottenuti in diverse modalità di sintesi.

Per poter effettuare tale esercizio, rispettando i requisiti forniti, è stato necessario suddividere la descrizione delle tre uscite in di tre file blif separati. Tale operazione si è resa necessaria in quanto il comando *simplify* di SIS, nel procedere alla minimizzazione di una rete combinatoria,

utilizza anche trasformazioni globali come ad esempio *substitute*.

Si riportano i risultati ottenuti con lo strumento SIS: si è stampata la funzione non semplificata usando *write_eqn* (fig.1.1) e poi si è proceduto alla minimizzazione tramite comando *simplify* delle uscite separate (fig.1.2).

```
sis> read_blif Esercitazione1_2.blif
sis> print_stats
esercizio1_2    pi= 6   po= 3   nodes=  3       latches= 0
lits(sop)= 528
sis> write_eqn
INORDER = X5 X4 X3 X2 X1 X0;
OUTORDER = Y2 Y1 Y0;
Y2 = 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*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*X2*X1*X0 + X5*!X4*X3*X2*X1*X0 + !X5*X4*X3*X2*X1*X0 + X5*X4*
X3*X2*X1*X0;
Y1 = 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*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*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*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;
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*!
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*!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*!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;
sis>
```

Figure 1.1: Stampa della funzione non semplificata in SIS.

Tra il comando *print_stats* è possibile osservare come i 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.1.3), 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.1.4, sono stati effettuati diversi esperimenti variando la funzione di costo rispetto alla quale viene ottimizzata in base alla

```
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*!X2*!X0 + !X5*X4*!X3*X2*!X0 + X5*!X4*!X2*!X1*X0 + !X5*X4*!X2*!
X1*X0 + !X4*!X3*X2*!X1*X0 + !X5*!X3*X2*!X1*X0 + !X5*!X4*X3*!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*!
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*!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*!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;
sis>
```

Figure 1.2: 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	1	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	✓

Figure 1.3: Minimizzazione di Y_0 con Mc-Cluskey.

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.

```
sis> read_blif Esercitazione1_2.blif
sis> read_library ~/sis-1.3/sis/sis_lib/mcnc.genlib
sis> map -W -m 1 -s                          sis> map -W -m 0 -s                          sis> map -W -m 0.5 -s
>>> before removing serial inverters <<<     >>> before removing serial inverters <<<     >>> before removing serial inverters <<<
# of outputs:          3                     # of outputs:          3                     # of outputs:          3
total gate area:       244.00                total gate area:       161.00                total gate area:       223.00
maximum arrival time: (16.41,16.41)          maximum arrival time: (21.50,21.50)          maximum arrival time: (19.10,19.10)
maximum po slack:     (-13.21,-13.21)        maximum po slack:     (-14.30,-14.30)        maximum po slack:     (-15.50,-15.50)
minimum po slack:     (-16.41,-16.41)        minimum po slack:     (-21.50,-21.50)        minimum po slack:     (-19.10,-19.10)
total neg slack:      (-43.76,-43.76)        total neg slack:      (-53.20,-53.20)        total neg slack:      (-50.10,-50.10)
# of failing outputs:  3                     # of failing outputs:  3                     # of failing outputs:  3
>>> before removing parallel inverters <<<  >>> before removing parallel inverters <<<  >>> before removing parallel inverters <<<
# of outputs:          3                     # of outputs:          3                     # of outputs:          3
total gate area:       196.00                total gate area:       161.00                total gate area:       169.00
maximum arrival time: (15.51,15.51)          maximum arrival time: (21.50,21.50)          maximum arrival time: (17.90,17.90)
maximum po slack:     (-12.31,-12.31)        maximum po slack:     (-14.30,-14.30)        maximum po slack:     (-14.30,-14.30)
minimum po slack:     (-15.51,-15.51)        minimum po slack:     (-21.50,-21.50)        minimum po slack:     (-17.90,-17.90)
total neg slack:      (-41.13,-41.13)        total neg slack:      (-53.20,-53.20)        total neg slack:      (-46.50,-46.50)
# of failing outputs:  3                     # of failing outputs:  3                     # of failing outputs:  3
# of outputs:          3                     # of outputs:          3                     # of outputs:          3
total gate area:       196.00                total gate area:       159.00                total gate area:       169.00
maximum arrival time: (15.51,15.51)          maximum arrival time: (21.30,21.30)          maximum arrival time: (17.90,17.90)
maximum po slack:     (-12.31,-12.31)        maximum po slack:     (-14.30,-14.30)        maximum po slack:     (-14.30,-14.30)
minimum po slack:     (-15.51,-15.51)        minimum po slack:     (-21.30,-21.30)        minimum po slack:     (-17.90,-17.90)
total neg slack:      (-41.13,-41.13)        total neg slack:      (-53.00,-53.00)        total neg slack:      (-46.50,-46.50)
# of failing outputs:  3                     # of failing outputs:  3                     # of failing outputs:  3
sis> print_map_stats                         sis> print_map_stats                         sis> print_map_stats
Total Area              = 196.00             Total Area              = 159.00             Total Area              = 169.00
Gate Count              = 72                 Gate Count              = 52                 Gate Count              = 58
Buffer Count            = 0                  Buffer Count            = 0                  Buffer Count            = 0
Inverter Count          = 6                  Inverter Count          = 9                  Inverter Count          = 10
Most Negative Slack     = -15.51             Most Negative Slack     = -21.30             Most Negative Slack     = -17.90
Sum of Negative Slacks  = -41.13             Sum of Negative Slacks  = -53.00             Sum of Negative Slacks  = -46.50
Number of Critical PO   = 3                  Number of Critical PO   = 3                  Number of Critical PO   = 3
```

Figure 1.4: Mapping tecnologico effettuato tramite libreria *mcnc.genlib* fornendo come parametri di bilanciamento, rispettivamente, 1, 0 e 0.5.

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.

1.2.3 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.1.5 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.1.6.

1.2.4 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.1.7.

1.2.5 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.1.8.

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
```

```
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>
```

Figure 1.5: Risultato della minimizzazione con *rugged.script*.

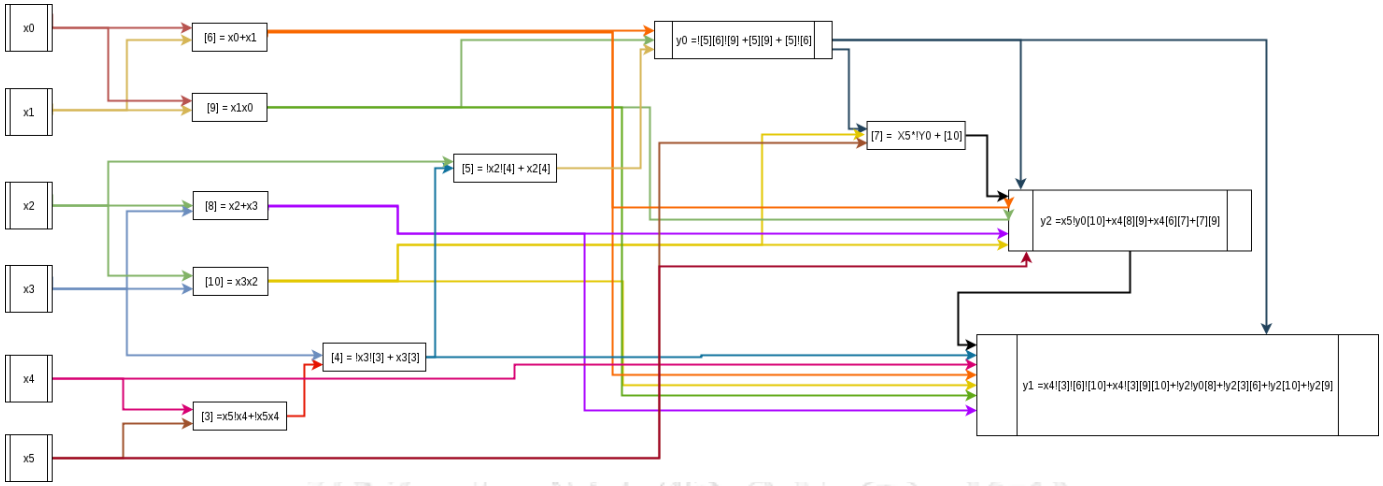

Figure 1.6: Grafo della funzione minimizzata tramite *rugged.script*.

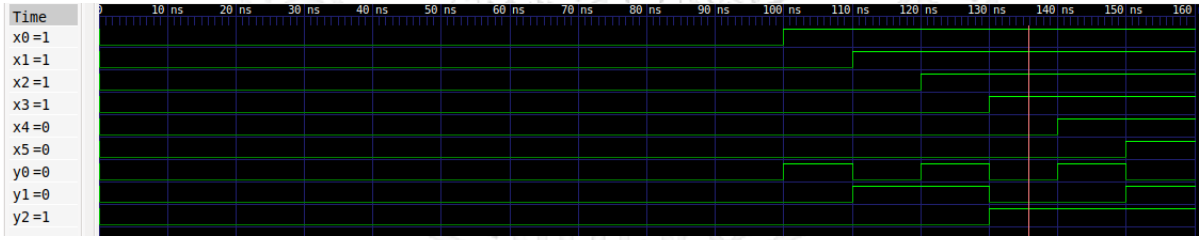

Figure 1.7: Simulazione della macchina M in *gtkwave*.

```
10 architecture dataflow of full_adder is
11   begin
12     S <=  (X xor Y xor CIN);
13     C <=  ((X and Y) or ((X xor Y) and CIN));
14   end dataflow;
```

Codice Componente 1.1: 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.

1.2.6 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, mentree 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.1.9.

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.1.10.

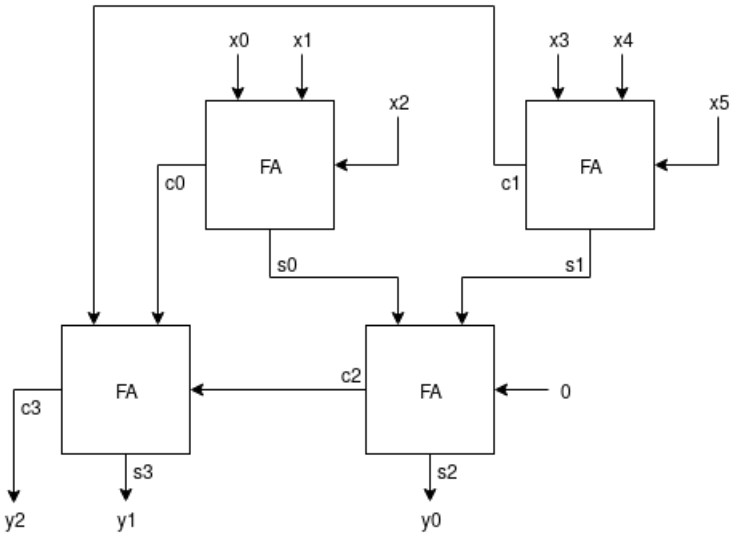

Figure 1.8: Schema della macchina M a partire da blocchi full-adder.

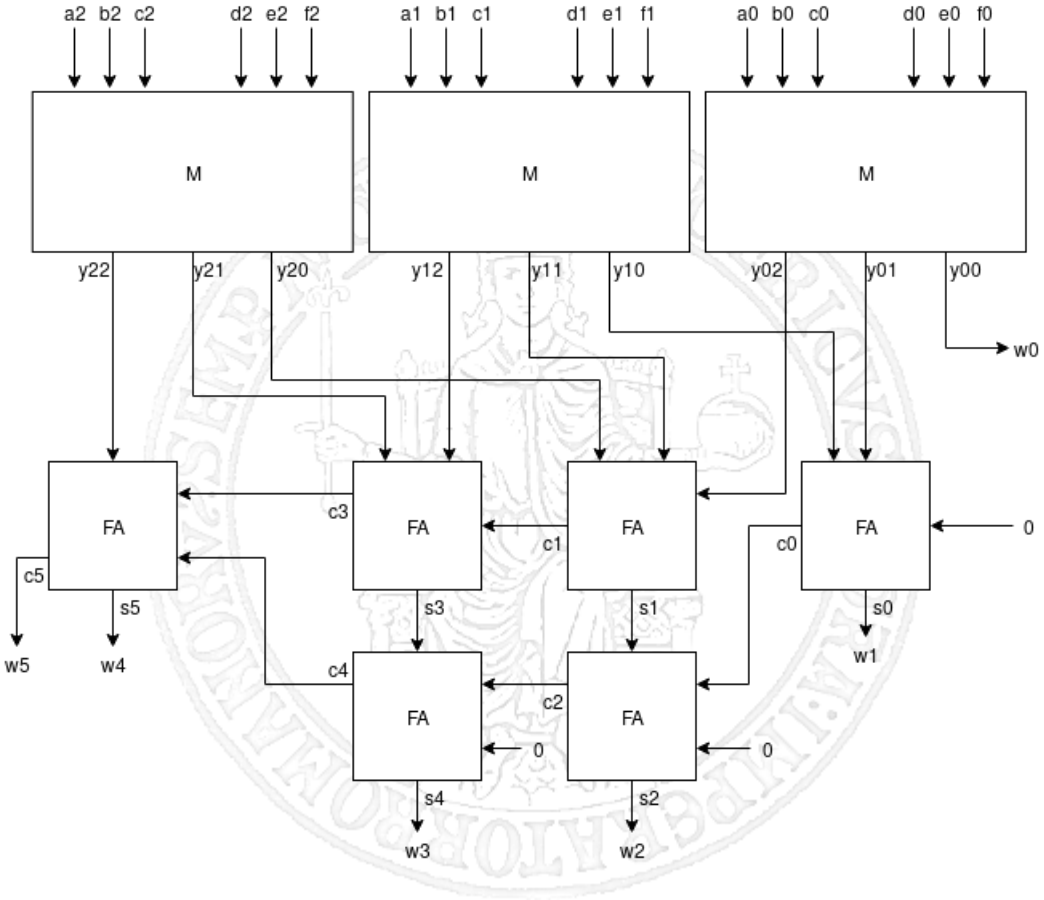

Figure 1.9: Schema della macchina S.

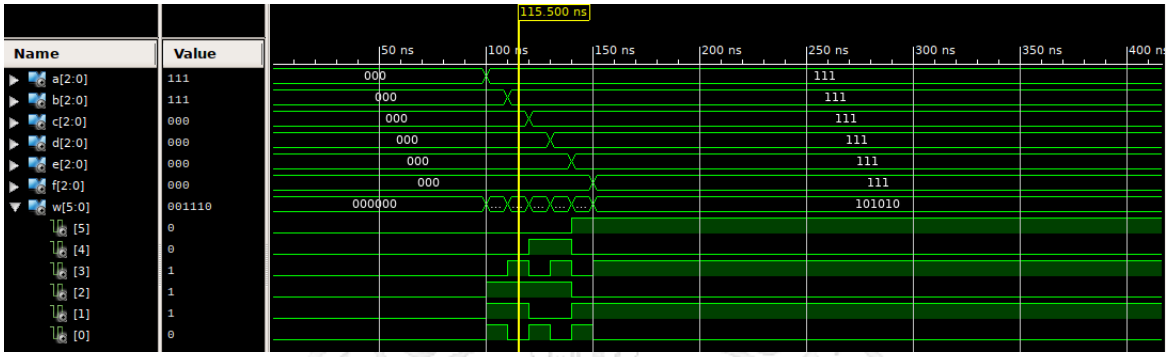

Figure 1.10: Risultato della simulazione della macchina S.

Chapter 2

Latch e Flip-Flop

Sviluppare i circuiti illustrati nel documento sui flip-flop. Eseguire per ciascun esercizio una simulazione comportamentale e post-sintesi, illustrando i passaggi salienti.

2.1 Latch RS

2.1.1 Traccia

Implementare e simulare un latch RS 1-attivo abilitato.

2.1.2 Soluzione

2.1.2.1 Implementazione

Il latch RS abilitato è stato realizzato tramite l'utilizzo di porte NOR e AND per l'abilitazione, come riportato in fig.2.1.

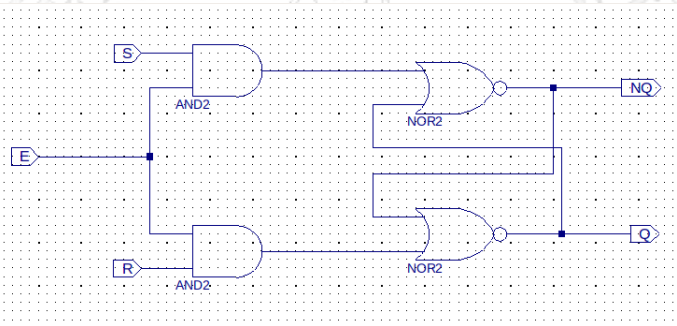

Figure 2.1: Schematico del latch RS abilitato.

Tale soluzione porterà il nostro componente latch RS a funzionare secondo una logica 1-attiva: in particolare, quando S=1 e R=0 il valore di Q si porta a 1, mentre per portare Q a 0 bisognerà portare R a 1 e S a 0. Si noti come, in tale componente, la combinazione S=1 e R=1 non è ammessa, in quanto porterebbe a cambiamenti di stato non definiti.

2.1.2.2 Simulazione

Per la simulazione behavioural di tale componente, dapprima si è testato il comportamento a fronte delle due tipologie di ingresso ammesse. In particolare si è posto prima S=1 e R=0 e, come atteso, il valore di Q risulta 1. Dopodiché, ponendo R=1 e S=0, il valore di Q risulta 0. Anche la situazione neutra S=0 e R=0, come previsto, fa in modo che Q non cambi il suo valore. Per quanto concerne la combinazione non ammessa S=1 e R=1, si noti come tale coppia di ingressi porti il latch ad assumere un comportamento non deterministico, poiché la condizione Q XOR NQ = 1 è violata (Q = NQ). Un caso di particolare interesse da osservare è la transizione da R=1, S=1 a R=0, S=0. Tale transizione porta idealmente a dei fenomeni oscillatori per le uscite, dovute all'assenza di ritardi di propagazione. Infatti nella simulazione behavioral si verifica che c'è l'oscillazione ma non è visibile, in quanto per tale simulazione si assumono ritardi di propagazione nulli, infatti il simulatore raggiunge il limite massimo di delta cycle a causa di queste oscillazione che si accumulano nello stesso istante di commutazione.Tale simulazione è visibile in fig.2.1 .

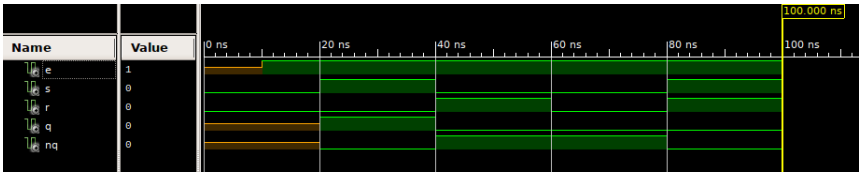

Figure 2.2: Simulazione behavioural del latch RS.

Tali oscillazioni sono visibili chiaramente nella simulazione post-map riportata in fig.2.1, dove ad ogni componente logico è associato il relativo ritardo di porta, secondo la libreria di ISE, ma non sono considerati i ritardi di propagazione.

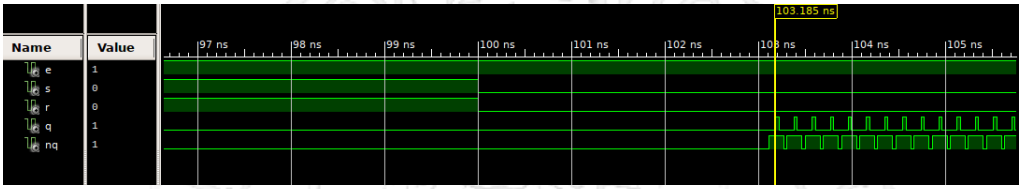

Figure 2.3: Simulazione post-map del latch RS.

Per quanto riguarda la simulazione post-route, le connessioni tra i componenti non sono più considerate ideali, senza ritardi, ma vengono considerati i ritardi di propagazione sulle linee di connessione. In questo caso, il simulatore, tenendo conto del posizionamento dei componenti, si ha che i ritardi di propagazione sono diversi per le due retroazione, i percorsi sono asimmetrici, ciò dovrebbe potare a non avere più delle oscillazioni. Nel caso in cui la simulazione venga effettuata su board Nexys 4 DDR (fig.2.1), le oscillazioni restano poiché i ritardi di propagazione sono identici sulle due linee non sono tali da forzare il nostro latch ad assumere un valore stabile dopo un certo intervallo di tempo. Infatti come si nota in figura, da aggiungere la foto di gab, si dovrebbe notare che i due percorsi delle retroazioni sono simmetrici. Ciò porterebbe all'identificazione di una probabile PUF (Physical Unclonable Function) ma quando tale componente viene sintetizzato e provato sulla Board, si ha che l'uscita va a 0 molto velocemente.

Nel caso di simulazione su board Basys 2 (fig.2.1), invece, i ritardi di propagazione sono diversi e le oscillazioni non sono più presenti, tale risultato ci mostra quello che accade nel caso reale.

2.2 Latch T

2.2.1 Traccia

Implementare e simulare un latch T 1-attivo abilitato.

2.2.2 Soluzione

2.2.2.1 Implementazione

2.2.2.2 Simulazione

2.3 Latch JK

2.3.1 Traccia

Implementare e simulare un latch JK 1-attivo abilitato.

2.3.2 Soluzione

2.3.2.1 Implementazione

2.3.2.2 Simulazione

2.4 Flip-flop D edge-triggered

2.4.1 Traccia

Implementare e simulare un flip-flop D edge-triggered che commuta sul fronte di salita con reset asincrono.

2.4.2 Soluzione

2.4.2.1 Implementazione

Il flip-flop D edge-triggered è stato realizzato tramite implementazione behavioural. Si noti come il process, sensibile solo al cambiamento di CLK e reset, porta il valore di D in Q solamente sul fronte di salita di CLK, descrivendo proprio il comportamento atteso del flip-flop. Nel caso in cui invece il reset venga portato al valore reset_level, il valore di Q viene resettato a prescindere dal comportamento del clock (reset asincrono).

```
1 architecture behavioural of flipflop_d_risingEdge_asyncReset is
2 signal q_temp   :   STD_LOGIC   :=init_value;
3 begin
4     q   <=  q_temp;
5       ff : process(clk, reset)
6         begin
7       if ( reset = reset_level ) then
8               q_temp <= init_value;
9           elsif ( rising_edge(clk) and (enable = enable_level) ) then
```

```
10          q_temp <= d;
11            end if;
12    end process ff;
13  end behavioural;
```

Codice Componente 2.1: Implementazione behavioural di un flip-flop D edge-triggered.

2.4.2.2 Simulazione

2.5 Flip-flop RS master-slave

2.5.1 Traccia

Implementare e simulare un flip-flop RS master-slave che commuta sul fronte di discesa.

2.5.2 Soluzione

2.5.2.1 Implementazione

Il flip-flop RS master-slave è stato realizzato tramite l'utilizzo di due latch RS collegati in cascata, come riportato in (fig.2.6). In particolare, il primo viene abilitato quando CLK = 1, mentre il secondo quando CLK = 0. Le uscite dunque saranno calcolate sul fronte di salita di clock, ma i risultati saranno visibili sul fronte di discesa.

2.5.2.2 Simulazione

Per la simulazione behavioural di tale componente (fig.2.7), si è proceduto analogamente al caso latch RS. In particolare, si è prima posto R=0 e S=1, in modo tale che, al fronte di discesa del CLK, Q si porta a 1. Anche il caso opposto, con R=1 e S=0, porta come previsto Q a 0, mentre per R=0 e S=0 la Q mantiene il proprio valore ad ogni fronte di discesa. Per quanto concerne il caso particolare R=1 e S=1, si noti come al fronte di discesa del clock si verifichino eventi oscillatori su Q dovuti all'utilizzo dei latch RS, che il simulatore behavioural non è in grado di mostrare a causa dell'elevatissimo numero di delta cycle.

Tali oscillazioni sono osservabili invece in simulazione post-map (fig.2.8).

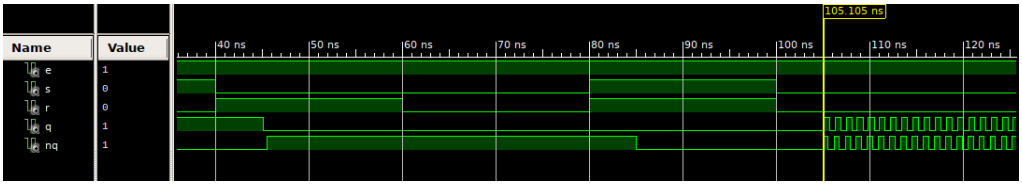

Figure 2.4: Simulazione post-route del latch RS su board Nexys 4 DDR.

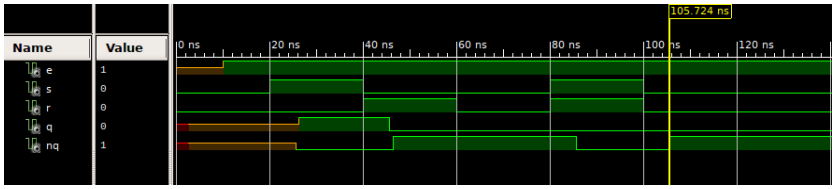

Figure 2.5: Simulazione post-route del latch RS su board Basys 2.

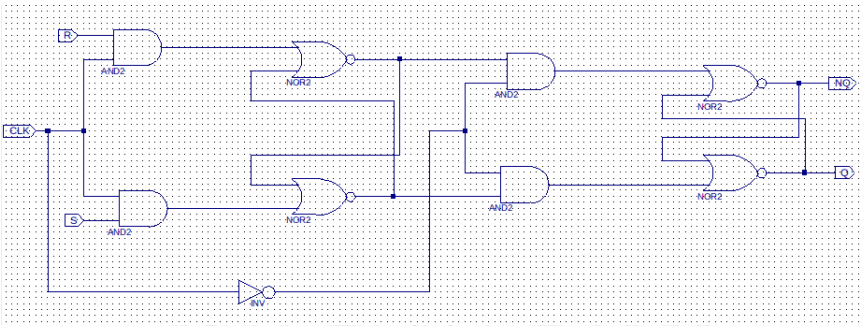

Figure 2.6: Schematico del flip-flop RS master-slave.

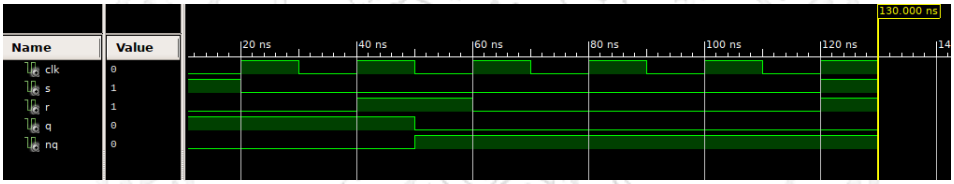

Figure 2.7: Simulazione behavioural del flip-flop RS master-slave.

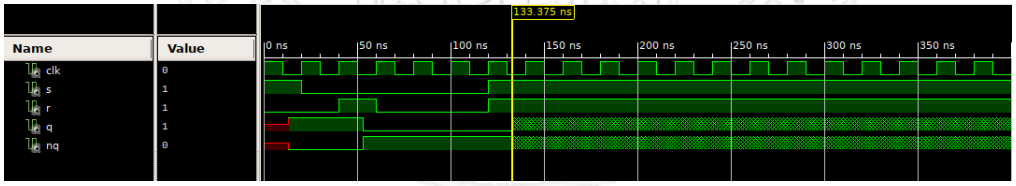

Figure 2.8: Simulazione post-map del flip-flop RS master-slave.