



**POLITECNICO
DI TORINO**

Laurea Magistrale in Ingegneria Elettronica

Anno Accademico 2018/2019

Sistemi Elettronici a basso consumo

prof. Maurizio Zamboni, prof. Mariagrazia Graziano



Relazione Laboratori

Raffaele Tuzzo 263722

Giulio Pecoraro 266391

Luigi Massari 265396

Contents

1 Laboratorio 1:

Power Estimation: probabilistic techniques	1
1.1 Probability and Activity Calculation: Simple Logic Gates	1
1.2 Probability and Activity Calculation: Half and Full adder	4
1.3 RCA synthesis and power analysis	7
1.4 A simple MUX: glitch generation and propagation	10
1.5 Probability and Activity Calculation: Syncronous Counter	12

2 Laboratorio 2:

FSM State Assignment and VHDL Synthesis	16
2.1 FSM State Assignment	16
2.2 VHDL synthesis	17

3 Laboratorio 3:

Clock gating, pipelining and parallelizing	24
3.1 A first approach to clock gating	24
3.2 Clock Gating for a complex circuit	27
3.2.1 Some more clock gating?	34
3.2.2 An automatic way to annotate activities	43
3.3 Pipelining and parallelizing	43
3.3.1 Are you sure it was correct?	50

4 Laboratorio 4:

Bus Encoding	51
4.1 Simulation	51

4.1.1	Non-encoded	51
4.1.2	Bus-invert technique, Transition based technique, Gray technique	53
4.1.3	T0 techinque	58
4.1.4	Confronto tra le tecniche	59
4.2	Synthesis	59
5	Laboratorio 5:	
	Leakage: using spice for characterizing cells and pen&paper for memory organization	61
5.1	Characterizing a library gate	61
5.2	Characterizing a gate for output load	64
5.3	Comparing different gate sizing	66
5.4	Comparing high speed and low leakage optimization	71
5.5	Temperature dependency	77
5.6	Analysis of a memory power components	79
6	Laboratorio 6:	
	Functional Verification	82
6.1	VHDL Testing	82
6.1.1	A given RCA	82
6.1.2	A more complex case	82
6.1.3	Finite State Machine	85
6.2	Scripting in Python	88
6.2.1	Automatic verification through a script	88
6.2.2	Further Generalization	92
7	Appendix	94
7.1	lab2	94
7.2	lab3	94
7.3	lab6	95
7.3.1	Testbench Incomp	95
7.3.2	Counter_tb.vhd	96
7.3.3	Counter_tb.do	97

7.3.4	Testing_script.py	99
7.3.5	Tb_generator.py	100

1. Laboratorio 1: Power Estimation: probabilistic techniques

1.1 Probability and Activity Calculation: Simple Logic Gates

Durante la prima parte dell'esercitazione è stato richiesto di calcolare la probabilità di avere '1' logico in uscita di alcuni gate elementari, con la relativa Switching Activity.



Figure 1.1: Gate elementari

La probabilità di '1' logico è stata stimata semplicemente andando a valutare il rapporto fra il numero di possibili combinazioni con '1' logico diviso il numero di combinazioni totali. Invece per il calcolo della Switching Activity è stata utilizzata la formula vista a lezione:

$$A = E_{SW} = P_1 P_0 + P_1 P_0 = 2P_1(1 - P_1)$$

Dove P_1 e P_0 sono le probabilità di avere '1' e '0' logici in uscita dalla mia porta.

Di seguito si riportano i procedimenti matematici per ottenere le probabilità

1.1. Probability and Activity Calculation: Simple Logic Gates

e le Switching Activity dei gate elementari visti in Figura 1.1.

- *Inverter*

$$P_1(Y) = P_0(A) = \frac{1}{2}$$

$$P_0(Y) = P_1(A) = \frac{1}{2}$$

$$E_{SW} = 2P_1(Y)(1 - P_1(Y)) = 2\frac{1}{2}\frac{1}{2} = \frac{1}{2}$$

- *AND*

$$P_1(Y) = P_1(A)P_1(B) = \frac{1}{2}\frac{1}{2} = \frac{1}{4}$$

$$E_{SW} = 2P_1(Y)(1 - P_1(Y)) = 2\frac{1}{4}\frac{3}{4} = \frac{3}{8}$$

- *OR*

$$P_1(Y) = P_1(A)P_1(B) + P_0(A)P_1(B) + P_1(A)P_0(B) = \frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{3}{4}$$

$$E_{SW} = 2P_1(Y)(1 - P_1(Y)) = 2\frac{3}{4}\frac{1}{4} = \frac{3}{8}$$

- *XOR*

$$P_1(Y) = P_1(A)P_0(B) + P_0(A)P_1(B) = \frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{1}{2}$$

$$E_{SW} = 2P_1(Y)(1 - P_1(Y)) = 2\frac{1}{2}\frac{1}{2} = \frac{1}{2}$$

Per maggiore chiarezza, i risultati sono raccolti nella Tabella 1.1.

	INVERTER	AND	OR	XOR
$P_1(Y)$	1/2	1/4	3/4	1/2
E_{SW}	1/2	3/8	3/8	1/2

Table 1.1: *Risultati ottenuti manualmente*

In seguito, tramite il programma *ModelSim* è stato analizzato il numero di toogle delle varie porte utilizzando un testbench sviluppato appositamente dai docenti. Si è andato a variare il numero di colpi di clock, come richiesto dalla traccia ed in seguito si sono comparati i valori ottenuti dalla simulazione con ciò che si era calcolato manualmente.

1.1. Probability and Activity Calculation: Simple Logic Gates

Tramite appositi comandi di Modelsim (**-power report**), sono stati stilati dei report relativi ad una stima delle commutazioni delle varie porte, della quale se ne riporta un esempio in Figura 1.2. Questi report consentono di stimare l'attività delle porte come verrà descritto in seguito.

Power Report		Node	Tc	Ti	Time At
1	Time At 0	Time At X			
<hr/>					
ps	5000000 ps	0 ps	/tbprob/clk	20000	0 5000000
ps	9999000 ps	0 ps	/tbprob/reset	1	0 1000
ps	9998000 ps	0 ps	/tbprob/ld	1	0 2000
ps	5071000 ps	0 ps	/tbprob/dout(15)	4924	0 4929000
ps	5071500 ps	0 ps	/tbprob/dout(14)	4923	0 4928500
ps	5072000 ps	0 ps	/tbprob/dout(13)	4922	0 4928000
ps	5072000 ps	0 ps	/tbprob/dout(12)	4922	0 4928000
ps	5072000 ps	0 ps			

Figure 1.2: *Probabilità e Switching Activity stimati dal simulatore*

Si riportano nella tabella 1.2, i risultati ottenuti dalle varie simulazioni.

Tc(CK)	Tc(INV)	Tc(AND)	Tc(OR)	Tc(XOR)
20	1	5	4	4
200	43	40	42	44
2000	533	418	352	470
20000	4916	3606	3784	4876

Table 1.2: *Risultati simulazione*

Dai seguenti valori è facile ricavare i valori di Switching Activity simulate, in quanto si possono stimare da:

$$A = E_{SW} = \frac{Tc(PORT)}{T_{CLK}}$$

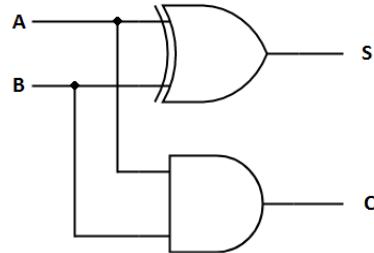
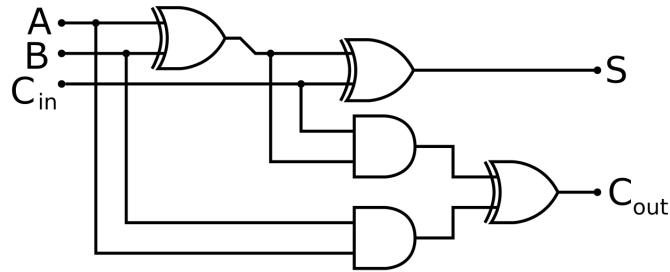
Come ci si aspettava, essendo la Switching Activity il numero di toggle avvenuti in un periodo, i valori delle simulazioni vengono molto simili ai valori calcolati analiticamente. Aumentando il tempo di simulazione, i valori di Switching Activity diventano sempre più precisi, arrivando ad avere un errore tra 0.01-0.5. I vari valori sono stati calcolati e raccolti nella Tabella 1.3

N_{CLOCK}	$E_{SW}(INV)$	$E_{SW}(AND)$	$E_{SW}(OR)$	$E_{SW}(XOR)$
10	0.1	0.5	0.4	0.4
100	0.43	0.40	0.42	0.44
1000	0.533	0.418	0.352	0.470
10000	0.492	0.36	0.378	0.487

 Table 1.3: *Switching activities calculate*

1.2 Probability and Activity Calculation: Half and Full adder

Nella seconda parte dell'esercitazione viene richiesto di analizzare le probabilità e la Switching Activity di un Half Adder e di un Full Adder, sia per quanto riguarda l'uscita S sia per quanto riguarda il carry out C_{OUT} , dei quali se ne riporta lo schema circuitale rispettivamente in Figura 1.3 e 1.4.


 Figure 1.3: *Schema circuitale Half Adder*

 Figure 1.4: *Schema circuitale Full Adder*

Il calcolo è stato effettuato in entrambi i casi, analizzando il blocco come un unico gate e andando a studiare la rispettiva Tavola di Verità. Anche in questa sezione, si considerano gli ingressi equiprobabili e scorrelati, con probabilità $P('input') = 0.5$. Il risultato è stato il seguente:

- *Half-Adder*

$$P_1(S) = P_1(A)P_0(B) + P_0(A)P_1(B) = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$E_{SW}(S) = 2P_1(S)(1 - P_1(S)) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$P_1(C_{OUT}) = P_1(A)P_1(B) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

$$E_{SW}(C_{OUT}) = 2P_1(C_{OUT})(1 - P_1(C_{OUT})) = 2 \cdot \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{8}$$

- *Full-Adder*

$$P_1(S) = P_0(A)P_0(B)P_1(C_{IN}) + P_0(A)P_1(B)P_0(C_{IN}) + P_1(A)P_0(B)P_0(C_{IN}) + \\ + P_1(A)P_1(B)P_1(C_{IN}) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$E_{SW}(S) = 2P_1(S)(1 - P_1(S)) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$P_1(C_{OUT}) = P_0(A)P_1(B)P_1(C_{IN}) + P_1(A)P_0(B)P_1(C_{IN}) + P_1(A)P_1(B)P_0(C_{IN}) + \\ + P_1(A)P_1(B)P_1(C_{IN}) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

$$E_{SW}(C_{OUT}) = 2P_1(C_{OUT})(1 - P_1(C_{OUT})) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

I risultati sono sintetizzati nella Tabella 1.4

	$P(S = 1)$	$P(C_{OUT} = 1)$	$E_{SW}(S = 1)$	$E_{SW}(C_{OUT} = 1)$
Half-Adder	1/2	1/4	1/2	3/8
Full-Adder	1/2	1/2	1/2	1/2

Table 1.4: *Risultati ottenuti manualmente*

In seguito si sono calcolate, sempre analiticamente, le probabilità di uscita con le rispettive switching activities del Ripple Carry Adder, riportato in Figura 1.5, utilizzando i dati calcolati in precedenza per il singolo Full-Adder. Anche per questo calcolo iniziale gli ingressi sono stati considerati scorrelati ed equiprobabili.

I risultati sono riportati nella Tabella 1.5.

1.2. Probability and Activity Calculation: Half and Full adder

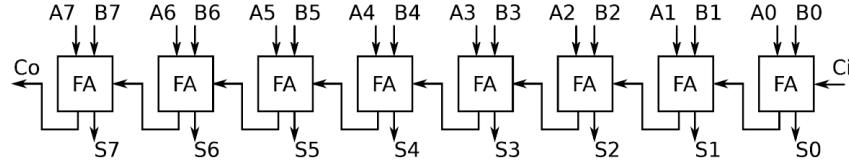


Figure 1.5: Ripple Carry Adder

	FA7	FA6	FA5	FA4	FA3	FA2	FA1	FA0
$P(S = 1)$	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2
E_{SW}	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2

Table 1.5: Probabilità e Switching Activity stimati manualmente con ingressi equiprobabili

Se invece si considerano gli ingressi non equiprobabili, ma comunque scorrelati, con probabilità rispettivamente $P(A = '1') = 0.4$ e $P(B = '1') = 0.6$, ci si aspetta un risultato differente, infatti le probabilità dei nodi di uscita trovate sono: $P(S = '1') = 0.52$ e $P(Co = '1') = 0.24$

In seguito, tramite il programma *ModelSim* è stato simulato il Ripple Carry Adder utilizzando un testbench sviluppato appositamente dai docenti e utilizzando le procedure viste nel paragrafo 1.1.

La simulazione viene fatta andando ad inserire dei ritardi sulle uscite S e C_{OUT} dei singoli Full-Adder. Inizialmente si setta esclusivamente un ritardo di 25 ps esclusivamente sul segnale S , mentre in secondo luogo si introduce un ritardo uguale anche sul segnale di C_{OUT} .

Il testbench è stato costruito appositamente per assegnare ritardi diversi al bit di somma, DRCAS, e al bit di carry, DRCCAC. Inoltre per garantire una simulazione generale si è utilizzato l'LFSR per generare ingressi randomici. Per una corretta visualizzazione dei risultati si è impostata una risoluzione di 1ps. Dopo aver visualizzato il power report, come fatto già in precedenza, si sono comparati i valori ottenuti dalla simulazione con ciò che si era calcolato manualmente, seguendo lo stesso ragionamento del punto precedente.

1.3. RCA synthesis and power analysis



Figure 1.6: *Power report*

(cosa noto?) e bisogna mettere le waveforms?

j- ATTENZIONE Si è poi simulato il caso in cui i due ritardi riguardanti il bit di somma e il bit di carry fossero uguali, e anche per questo è stato visualizzato il power report.



Figure 1.7: *Power report con ritardi uguali*

Da un analisi e confronto tra i risultati ottenuti manualmente e quelli ottenuti con le simulazioni, si nota come questi coincidano dato che avendo uguali ritardi, non riesco a simulare la presenza di eventuali glitch.

In seguito si è calcolata la switching activity totale dei due sommatori, utilizzando la seguente formula:

$$A = \sum_{i=1}^{N-1} A(S_i)$$

What is the overhead computation of the second adder? <- cioè?

Come ultima cosa è stato simulato il secondo testbench che ci è stato fornito, dove si simulava sempre un Ripple carry adder, ma questa volta in maniera puramente combinatoria. Analizzando i risultati, si può concludere come non avendo un segnale di temporizzazione, lavoro alla massima velocità ma ho la presenza di glitch.

1.3 RCA synthesis and power analysis

Nella seguente sezione dell'esercitazione è stata analizzata la potenza del sommatore RCA già analizzato in precedenza, tramite il software *Synopsys*. Dopo aver analizzato ed elaborato i file che descrivono la struttura dell'RCA,

1.3. RCA synthesis and power analysis

il tutto è stato sintetizzato e sono stati raccolti i vari report relativi alla potenza. Un primo report di potenza, riportato in Figura 1.8, descrive i contributi di potenza relativi alle 8 istanze dei Full-Adder che compongono il somamtore RCA.

Design	Wire Load Model	Library
RCA	5K_hvratio_1_1	NangateOpenCellLibrary
<hr/>		
Global Operating Voltage = 1.1		
Power-specific unit information :		
Voltage Units = 1V		
Capacitance Units = 1.000000ff		
Time Units = 1ns		
Dynamic Power Units = 1uW (derived from V,C,T units)		
Leakage Power Units = 1nW		
<hr/>		

Total	Switch	Int
Hierarchy	Power	Power
Power %		Leak
<hr/>		
RCA	9.741	16.743
27.437 100.0	953.483	
FAI_8 (FA_1)	0.848	2.154
3.121 11.4	119.291	
FAI_7 (FA_2)	1.293	2.150
3.562 13.0	118.962	
FAI_6 (FA_3)	1.332	2.191
3.642 13.3	119.340	
FAI_5 (FA_4)	1.328	2.171
3.618 13.2	119.110	
FAI_4 (FA_5)	1.278	2.101
3.498 12.7	119.294	
FAI_3 (FA_6)	1.263	2.090
3.471 12.7	118.979	
FAI_2 (FA_7)	1.229	2.002
3.351 12.2	119.508	
FAI_1 (FA_0)	1.171	1.884
3.175 11.6	118.999	
1		

Figure 1.8: *Power report*

Come ci si aspettava, i contributi dei vari Full Adder sono tutti simili tra di loro, ad eccezione dell'istanza *FAI_8*: il motivo consiste nel fatto che il Carry Out dell'ultimo Full Adder non è connesso a nessun'altra porta, dunque il carico da pilotare è decisamente minore.

Diventa ora interessante andare ad analizzare la singola istanza, per andare a valutare l'origine dei singoli contributi di potenza. Tramite il comando *current_instance FAI_1* si va ad analizzare l'istanza relativa al primo Full Adder. Viene riportato il report in Figura 1.9.

1.3. RCA synthesis and power analysis

Cell Leakage Cell Power	Attrs	Cell	Driven Net	Tot	Dynamic
		Internal Power	Switching Power	(% Cell/Tot)	
U1 36.0111		0.8205	0.0512	0.872	(94%)
U4 36.1637		0.5889	0.5488	1.138	(52%)
U3 32.5747		0.3374	0.1749	0.512	(66%)
U2 14.2499		0.1377	0.3964	0.534	(26%)
<hr/>		<hr/>		<hr/>	
Totals (4 cells)		1.884uW	1.171uW	3.056uW	(62%)
118.999nW					

Figure 1.9: *Power report -current_instance*

Come ci si aspettava i valori di potenza risultano assolutamente identici al report trovato in precedenza e riportato in Figura 1.8. Il passo successivo è comprendere come avvenga la stima della potenza dinamica (*switching power*) dei singoli nodi del FA, dato che la potenza interna e quella di leakage dipendono solo dal gate e non dal circuito. Si utilizza a questo scopo il comando *report_power -net -verbose* e si riporta il report risultante in Figura 1.10.

Attributes					
<pre>a - Switching activity information annotated on net d - Default switching activity information on net</pre>					
Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power	
Attrs					
n1	4.694	0.493	0.1932	0.5488	
Co	4.554	0.512	0.1439	0.3964	
n2	2.010	0.488	0.1439	0.1749	
S	0.310	0.507	0.2735	0.0512	
<hr/>		<hr/>		<hr/>	
Total (4 nets)				1.1714 uW	

Figure 1.10: *Power report -net -verbose*

Si può notare come la potenza dinamica consumata dal nodo di uscita S sia praticamente nulla, in quanto essa non deve pilotare alcun carico, a meno della potenza che il software considera per le varie capacità parassite.

1.4. A simple MUX: glitch generation and propagation

Con questa analisi, il software si preoccupa anche di fare un'analisi probabilistica delle varie attività dei nodi. Dal report si può notare come siano assolutamente concordi con i valori teorici trovati in precedenza.

Si ritorna ora al circuito complessivo, salendo di livello logico e si analizzano i nodi anche in questa situazione utilizzando lo stesso comando utilizzato in precedenza. Il report viene raffigurato in Figura 1.11.

Net Attrs	Net Switching Power	Driver Internal Power	Tot Dynamic Power (% Net/Tot)	Driver Leakage Power
<hr/>				
CTMP[6]	0.5020	0.1743	0.676 (74%)	14.3223
CTMP[5]	0.4978	0.1729	0.671 (74%)	14.3131
CTMP[7]	0.4846	0.1683	0.653 (74%)	14.2968
CTMP[4]	0.4730	0.1643	0.637 (74%)	14.4166
CTMP[3]	0.4601	0.1598	0.620 (74%)	14.3492
CTMP[2]	0.4392	0.1526	0.592 (74%)	14.3724
CTMP[1]	0.3964	0.1377	0.534 (74%)	14.2499
S[5]	0.0635	0.9916	1.055 (6%)	36.1498
S[6]	0.0628	0.9868	1.050 (6%)	35.8838
S[7]	0.0624	0.9733	1.036 (6%)	36.1631
S[4]	0.0621	0.9747	1.037 (6%)	36.1707
S[3]	0.0606	0.9492	1.010 (6%)	36.1063
S[2]	0.0605	0.9521	1.013 (6%)	35.9520
S[1]	0.0571	0.8921	0.949 (6%)	36.1233
S[0]	0.0512	0.8205	0.872 (6%)	36.0111
Co	0.0332	0.1774	0.211 (16%)	14.2174
<hr/>				
Totals (16 nets)	3.766uW	8.848uW	12.614uW (30%)	
403.098nW				

Figure 1.11: *Power report -net -verbose Full Adder*

1.4 A simple MUX: glitch generation and propagation

Durante questa sezione, viene chiesto di analizzare il comportamento di un Multiplexer, riportato in Figura 1.12, dove si assume che i ritardi delle varie porte elementari siano nulli, ad eccezione dell'Inverter che presenta un ritardo pari a 0,1 ns. Inizialmente si simula il circuito utilizzando *ModelSim* e si riportano le onde in Figura 1.13.

1.4. A simple MUX: glitch generation and propagation

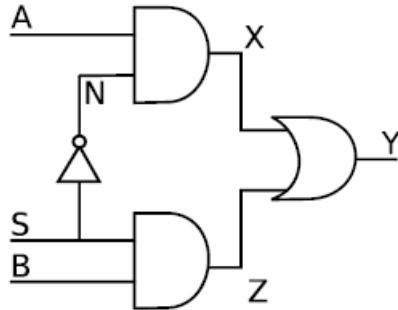


Figure 1.12: *Mux*

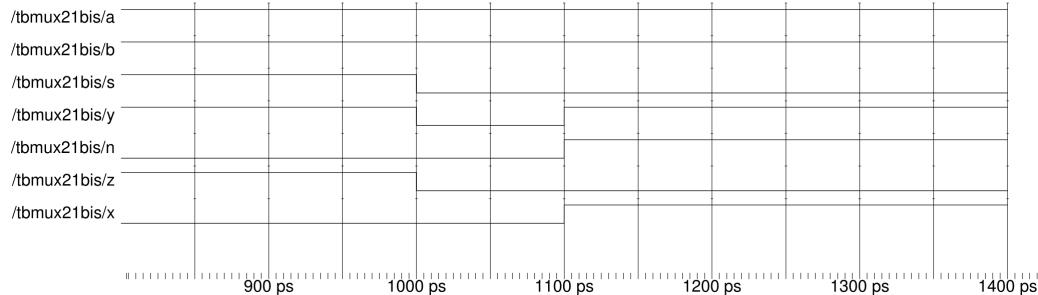


Figure 1.13: *Mux's waves*

Si può notare benissimo come l'uscita Y abbia un glitch tra 1000 ps e 1100 ps, in quanto effettua una transizione 0-1 non voluta. Il tutto è causato dal ritardo introdotto dall'Inverter, che si può notare dal segnale 'n', che commuta al suo valore corretto dopo un ritardo di 0,1 ns, rispetto all'istante in cui cambiano tutti i restanti gate del circuito, ossia 1000 ps. Si verifica dunque un intervallo di tempo, ossia quello compreso tra 1000 ps e 1100 ps in cui la porta OR ha entrambi gli ingressi bassi e dunque produce un'uscita Y errata.

Questo produce quindi un glitch in uscita: in generale i glitch sono problematici a livello di potenza, in quanto si tratta di commutazioni spurie all'interno del mio circuito, che causano uno spreco di potenza. L'energia consumata per ogni doppia commutazione indesiderata è

1.5. Probability and Activity Calculation: Synchronous Counter

$$E = C_L V_{DD}^2$$

considerando C_L come il carico che viene pilotato e V_{DD} la tensione di alimentazione alla quale si lavora.

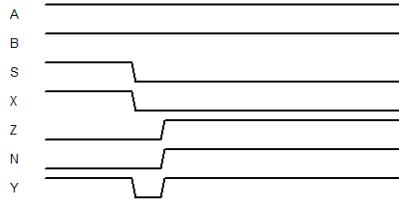


Figure 1.14: *Timing diagram*

1.5 Probability and Activity Calculation: Synchronous Counter

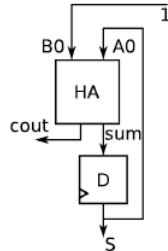


Figure 1.15: *Counter 1 bit*

Si analizza in questa sezione un *Synchronous Counter* ad 1 bit, realizzato con una Half Adder e un D-FlipFlop, riportato in Figura 1.15. Il primo ingresso A_0 dell'Half Adder è l'uscita del D-FlipFlop, mentre il secondo ingresso B_0 viene collegato fisso ad 1.

1.5. Probability and Activity Calculation: Syncronous Counter

Si riporta il timing diagram del circuito in Figura 1.16.

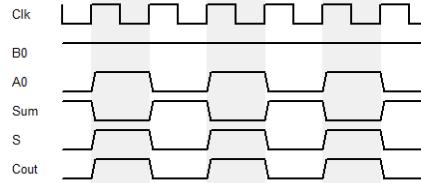


Figure 1.16: *Timing counter 1 bit*

Si può dedurre dal timing, che il segnale S ha una transizione per ogni colpo di clock, di conseguenza la sua Switching Activity sarà metà dell'attività del segnale di clock. Nel caso in cui il segnale B_0 vada a 0, il valore del nodo S dipenderà esclusivamente dall'ultimo valore presente all'uscita del Flip-Flop, in quanto l'Half Adder sommerà l'ultimo valore presente nel Flip Flop con uno '0'. Quindi non si verificheranno più transizioni sul nodo S e di conseguenza neanche sul nodo C_{OUT} .

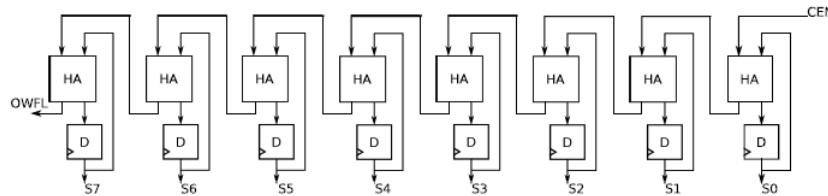


Figure 1.17: *Counter 8 bit*

In seguito viene chiesto di analizzare la struttura presente in Figura 1.17. Questo circuito permette di ottenere un contatore sincrono ad 8 bit, utilizzando in cascata 8 celle del circuito precedente. Anche in questo si riporta, in Figura 1.18, il timing diagram del funzionamento del circuito. In Tabella 1.6, vengono ora riportati il numero di commutazioni di tutti i segnali di uscita ricavati in modo analitico, considerando un intervallo di tempo compreso da quando il circuito parte dal valore '00000000', fino a quando non arriva al valore '11111111'.

1.5. Probability and Activity Calculation: Syncronous Counter

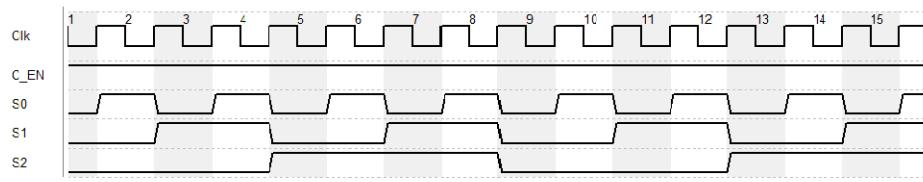


Figure 1.18: *Timing counter 8 bit*

Signal	Number of Transitions
Clock	511
S0	255
S1	127
S2	63
S3	31
S4	15
S5	7
S6	3
S7	1

Table 1.6: *Risultati simulazione*

Il segnale *CEN*, rappresenta l'enable del contatore, mentre il segnale *OWFL* va ad 1 in presenza di overflow, dunque quando si supera la massima dinamica del contatore. Il segnale CEN è l'ingresso B0 del primo Half-Adder, mentre il segnale OWFL è il carry out dell'ultimo Half-Adder.

Il Segnale OWFL farà solo una transizione per ogni ciclo di conta, dunque andrà ad '1', solo quando le uscite S7-S0 andranno dal valore '11111111' al valore '00000000'. Di conseguenza avrà che la Switching Activity sarà:

$$E_{SW} = \frac{1}{2^8 - 1} = 0.00392$$

Si vuole ora confrontare i risultati analitici ricavati in precedenza ed i risultati ottenuti sfruttando le simulazione di *Synopsis*. In Figura 1.19, viene riportata la simulazione ottenuta.

Viene chiesto di confrontare i valori analitici calcolati e riportati nella Tabella 1.6, con quelli derivanti da una simulazione *Synopsis*. Si simula il comportamento del circuito utilizzando un $T_{CLK} = 2\text{ns}$, per un periodo pari a 260 colpi di clock. Il report risultante è presente in Figura 1.19.

1.5. Probability and Activity Calculation: Syncronous Counter

Power Report Interval 520000						
Power Report	Node	Tc	Ti	Time At 1	Time At 0	Time At X
/testcount/ucounter1/a	1	0	514000	6000	0	
/testcount/ucounter1/ck	520	0	260000	260000	0	
/testcount/ucounter1/reset	1	0	2000	518000	0	
/testcount/ucounter1/s(7)	2	0	256000	264000	0	
/testcount/ucounter1/s(6)	4	0	256000	264000	0	
/testcount/ucounter1/s(5)	8	0	256000	264000	0	
/testcount/ucounter1/s(4)	16	0	256000	264000	0	
/testcount/ucounter1/s(3)	32	0	256000	264000	0	
/testcount/ucounter1/s(2)	64	0	256000	264000	0	
/testcount/ucounter1/s(1)	128	0	256000	264000	0	
/testcount/ucounter1/s(0)	257	0	257000	263000	0	
/testcount/ucounter1/co	16	0	2000	517800	200	
/testcount/ucounter1/stmp(7)	30	0	256000	263600	400	
/testcount/ucounter1/stmp(6)	52	0	256000	263600	400	
/testcount/ucounter1/stmp(5)	88	0	256000	263600	400	
/testcount/ucounter1/stmp(4)	144	0	256000	263600	400	
/testcount/ucounter1/stmp(3)	224	0	256000	263600	400	
/testcount/ucounter1/stmp(2)	320	0	256000	263600	400	
/testcount/ucounter1/stmp(1)	385	0	256600	263000	400	
/testcount/ucounter1/stmp(0)	258	0	257000	262800	200	
/testcount/ucounter1/ctmp(8)	16	0	2000	517800	200	
/testcount/ucounter1/ctmp(7)	28	0	4000	515800	200	
/testcount/ucounter1/ctmp(6)	48	0	8000	511800	200	
/testcount/ucounter1/ctmp(5)	80	0	16000	503800	200	
/testcount/ucounter1/ctmp(4)	128	0	32000	487800	200	
/testcount/ucounter1/ctmp(3)	192	0	64000	455800	200	
/testcount/ucounter1/ctmp(2)	256	0	128000	391800	200	
/testcount/ucounter1/ctmp(1)	257	0	256800	263000	200	
/testcount/ucounter1/ctmp(0)	1	0	514000	6000	0	
/testcount/ucounter1/stmpsync(7)	2	0	256000	264000	0	
/testcount/ucounter1/stmpsync(6)	4	0	256000	264000	0	
/testcount/ucounter1/stmpsync(5)	8	0	256000	264000	0	
/testcount/ucounter1/stmpsync(4)	16	0	256000	264000	0	
/testcount/ucounter1/stmpsync(3)	32	0	256000	264000	0	
/testcount/ucounter1/stmpsync(2)	64	0	256000	264000	0	
/testcount/ucounter1/stmpsync(1)	128	0	256000	264000	0	
/testcount/ucounter1/stmpsync(0)	257	0	257000	263000	0	

Figure 1.19: Power report

Andando a studiare i file di testbench riportati, si può notare come vengano introdotti dei ritardi di 0.2 ns nelle uscite dell'Half Adder: questo provoca dei glitch che si andranno a propagare lungo la rete ed avranno effetti sul segnale OWFL, che invece di andare ad '1' solo una volta a fine conteggio, andrà ad '1' per tre volte.

Tramite il power report si può notare come, in generale, il numero di commutazioni della simulazione sia superiore a quello ottenuto analiticamente: si può notare questo fenomeno nei segnali *stmp(n)*, ossia le uscite non sincrone dei Flip Flop, e nei segnali di Carry Out. Il tutto è dovuto ai glitch che si propagano interamente al contatore a causa del ritardo di 0.2 ns di ciascun Half Adder.

Fortunatamente, questi glitch vengono filtrati dai Flip-Flop, in quanto il T_{CLK} è superiore al ritardo introdotto dalla rete combinatoria.

2. Laboratorio 2: FSM State Assignment and VHDL Synthesis

2.1 FSM State Assignment

Durante la prima parte dell'esercitazione di laboratorio, viene richiesto di implementare un circuito per sommare 6 numeri

$$s = a + b + c + d + e + f$$

utilizzando un unico sommatore, due multiplexer e un registro. Viene richiesto di valutare e minimizzare il consumo di potenza, andando a modificare la connessione degli input A-H, considerando esclusivamente l'attività della FSM e i bit di selezione del MUX S0-S3.

Il circuito completo è riportato in Figura 2.1, mentre la FSM è presente in Figura ??.

Dopo varie ottimizzazioni, si è arrivati ad avere un'attività totale pari a 8 per il multiplexer e 6 per la State transition della macchina a stati, andando a considerare che la macchina a stati e il multiplexer ricomincino le operazioni una volta terminate. Nella tabella 2.1 viene riportata la configurazione degli stati e dei bit del multiplexer scelta:

2.2. VHDL synthesis

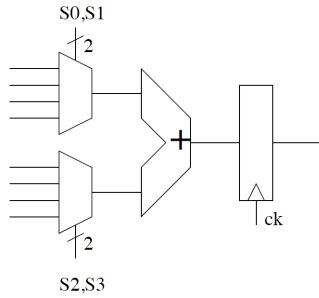


Figure 2.1: *Datapath*

STATI	$S_3S_2S_1S_0$
000	0000
001	0101
011	0111
010	1110
110	1010

Table 2.1: *Tabella degli stati*

2.2 VHDL synthesis

Il secondo punto del laboratorio prevede di sintetizzare l'FSM tramite synopsys e studiarne le caratteristiche in termini di area, potenza e timing in modo da ricercare possibili ottimizzazioni. Si è utilizzata la libreria a 45 nm, definito un segnale di clock di periodo corrispondente a 10 ns, si è verificato il corretto inserimento tramite il comando *report_clock* e si è sintetizzato il circuito.

clock	period	waveform
CLK	10.00 ns	{0 5} V

Table 2.2: *Report clock*

Di seguito è riportato lo schema generato da synopsys:

2.2. VHDL synthesis

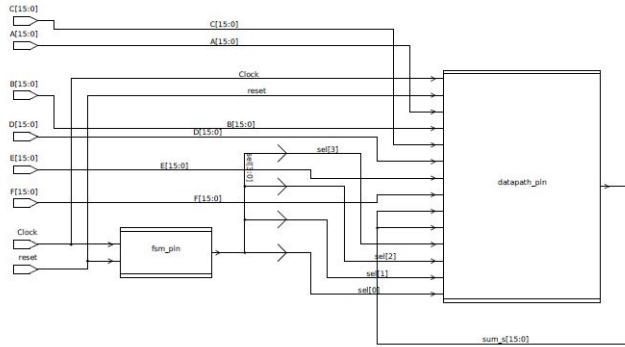


Figure 2.2: Schematico del circuito sintetizzato

Dal report sull'area si sono ottenute informazioni riguardanti la quantità di componenti, le connessioni, l'area relativa occupata dalla logica combinatoria, circa il doppio rispetto a quella non combinatoria e quindi dell'area totale.

type	number
ports	114
nets	118
cells	2
references	2

area type	value
combinational	195.244003
noncombinational	101.08003
total cell	296.324006

Successivamente, dopo aver verificato la corretta codifica degli stati della FSM si è analizzato il timing del circuito, dal quale si sono ottenute importanti informazioni riguardo ai ritardi delle varie porte e allo Slack time nel caso del percorso peggiore che è di 8.03 ns, parametro che consente di ottimizzare frequenza di funzionamento del circuito, visto che la condizione necessaria è che lo slack time sia positivo. Inoltre si è eseguito il timing per i peggiori 10 percorsi e si sono ricavati i valori di slack.

2.2. VHDL synthesis

Slack (MET)	value [ns]
1	8.04
2	8.04
3	8.04
4	8.04
5	8.04
6	8.05
7	8.05
8	8.05
9	8.05
10	8.05

Non si evidenziano rilevanti differenze tra i diversi slack, ciò è dovuto alla simmetria dei percorsi critici che presentano la stessa struttura. Ecco come sono distribuiti i peggiori slack:

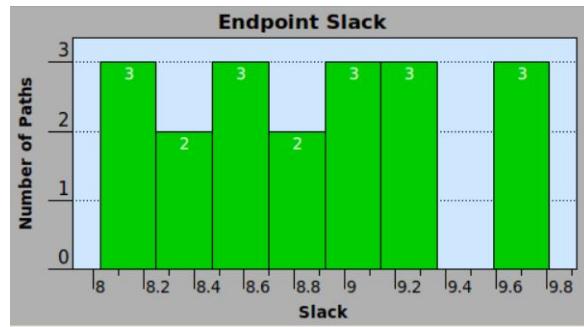


Figure 2.3: *Grafico distribuzione slacks*

Dunque si è analizzata la potenza dissipata sia dall'intera logica che da ogni singola cella. Il report sulla potenza distingue la potenza dissipata dinamicamente, staticamente e quella dovuta alle correnti di leakage, fornendo la percentuale rispetto alla potenza dissipata totale e analizzando il circuito a livello gerarchico. In questo modo è possibile capire quanto influiscono i diversi contributi di potenza dissipata e se necessario intervenire opportunamente per consumare meno.

2.2. VHDL synthesis

hierarchy	switch	int	leak	tot	%
m_adder	11.943	28.443	5.46e+3	45.844	100
datapath_adder	10.930	25.532	5.03e+3	41.495	90.5
add_78	1.765	4.921	1.19e+3	7.879	17.2
fsm	1.012	2.911	425.171	4.349	9.5

Inoltre si è analizzata l'attività delle singole celle in modo da studiare i consumi di ogni singola cella. Da quest'ultimo report si è ricavato che le celle corrispondenti ai registri hanno dei consumi più elevati di potenza statica e consumano più del doppio della corrente di leakage rispetto alle altre celle del circuito.

cell	cell internal	driven net switching	tot dynamic [% cell/tot]	cell leakage
REG[0]	1.0163	0.0584	1.075 (95%)	87.1072
REG[1]	0.8660	0.1134	0.979 (88%)	81.4649
REG[2]	0.7468	0.1083	0.855 (87%)	84.7325
U8	0.0465	0.0297	7.62e-2 (61%)	33.6813
U9	0.0386	0.1368	0.175 (22%)	31.6341
U6	0.0375	0.1356	0.173 (22%)	18.0848
U5	0.0314	0.0193	5.07e-2 (62%)	19.3118
U4	0.0304	0.0174	4.77e-2 (64%)	17.9767
U10	0.0285	0.0746	0.103 (28%)	12.9020
U7	0.0233	0.1282	0.151 (15%)	15.8344
U3	0.0190	0.1236	0.143 (13%)	17.0242
U11	9.993e-3	0.0392	4.92e-2 (20%)	14.3532
tot (12 cells)	2.894 uW	984.349 nW	3.878 uW (75%)	434.107 nW

E' utile inoltre valutare il numero di commutazioni delle singole uscite valutando sia la capacità del carico che il tasso di commutazioni. In accordo con i risultati precedenti si denota un'attività più intensa per i registri per hanno un carico capacitivo molto più alto delle altre uscite anche se il toggle rate è praticamente equivalente per tutte le uscite e la probabilità statistica è comunque minore.

2.2. VHDL synthesis

hierarchy	switch	int	leak	tot	%
m_adder	11.943	28.443	5.46e+3	45.844	100
datapath_adder	10.930	25.532	5.03e+3	41.495	90.5
add_78	1.765	4.921	1.19e+3	7.879	17.2
fsm	1.012	2.911	425.171	4.349	9.5

Inoltre si è analizzata l'attività delle singole celle in modo da studiare i consumi di ogni singola cella. Da quest'ultimo report si è ricavato che le celle corrispondenti ai registri hanno dei consumi più elevati di potenza statica e consumano più del doppio della corrente di leakage rispetto alle altre celle del circuito.

net	total net load	static prob.	toggle rate	switching power
S[2]	11.104	0.326	0.0244	0.1641
S[0]	9.304	0.228	0.0244	0.1375
S[1]	10.166	0.295	0.0221	0.1360
S[3]	9.541	0.186	0.0200	0.1153
n21	10.518	0.088	0.0181	0.1153
n5	6.169	0.814	0.0200	0.0746
n6	3.949	0.772	0.0244	0.0584
n8	4.078	0.706	0.0222	0.0546
n25	3.843	0.500	0.0221	0.0514
n28	6.482	0.772	0.0100	0.0392
n27	1.980	0.706	0.0244	0.0293
N8	1.438	0.098	0.0221	0.0193
n7	1.438	0.0392	0.0200	0.0174
tot (13 nets)				1.0125 uW

Si è adesso focalizzata l'attenzione sui consumi della Macchina a Stati. La FSM è caratterizzata da una leakage current di 418 nW a fronte dei 434 nW totali e anche per gli altri contributi di consumo di potenza dinamica i dati tendono ad evidenziare il ruolo preponderante del componente sul totale consumo del circuito.

2.2. VHDL synthesis

Cell	Cell Internal Power	Driven Net Switching Power	Tot Dynamic Power (% Cell/Tot)	Cell Leakage Power	Attrs
CURRENTSTATE_reg[0]	0.8986	0.0979	0.997 (90%)	89.1728	
CURRENTSTATE_reg[1]	0.5562	0.0461	0.602 (92%)	81.9647	
CURRENTSTATE_reg[2]	0.3711	2.719e-03	0.374 (99%)	79.5076	
U11	0.0999	0.3922	0.492 (20%)	14.3532	
U6	0.0989	0.0453	0.144 (69%)	30.0052	
U5	0.0644	0.0595	0.124 (52%)	17.0430	
U7	0.0384	0.0491	8.75e-02 (44%)	11.6295	
U4	0.0193	1.813e-03	2.12e-02 (91%)	34.6953	
U10	0.0109	8.310e-03	1.92e-02 (57%)	20.1295	
U9	2.905e-03	1.593e-03	4.50e-03 (65%)	16.3885	
U8	2.689e-03	6.834e-03	9.52e-03 (28%)	5.3388	
U3	1.734e-03	3.431e-04	2.08e-03 (83%)	18.5247	
Totals (12 cells)	2.165uW	711.655nW	2.877uW (75%)	418.753nW	

Figure 2.4: *Potenza dissipata dalla FSM*

Dopodiché si è provato a variare la frequenza di lavoro del circuito, provando a sintetizzare il circuito in modo da lavorare alla massima frequenza di funzionamento consentita dal percorso critico. Dall'analisi sul timing si è trovato lo slack peggiore di circa 8.02 ns lavorando a 10 ns di periodo di clock. Si può allora decrementare il periodo di clock fino a $10 - 8.02 = 1.98$ ns. Difatti si è scelto un periodo di clock di 2 ns.

cell internal power	141.4894 uW (71%)
net switching power	58.8064 uW (29%)
total dynamic power	200.2958 uW (100%)
cell leakage power	5.5273 uW

Si nota come sia aumentata la Total Dynamic Power, questo poiché aumentando la frequenza operativa aumentano anche il numero di commutazioni interne e quindi viene dissipata maggiore potenza dinamica. Resta invece invariata la corrente di leakage che infatti dipende solo dalla tecnologia usata.

Infine è stato posto al sintetizzatore un ulteriore vincolo sulla massima potenza dinamica dissipabile a 200 uW, considerando che nell'ultimo report

2.2. VHDL synthesis

la potenza totale dissipata ammonta a 200.2958 uW.

cell internal power	140.3547 uW (70%)
net switching power	59.0397 uW (30%)
total dynamic power	199.3944 uW (100%)
cell leakage power	5.5832 uW

Adesso la potenza dinamica totale dissipata è di 199.3944 uW e rispetta il vincolo. Inoltre, si nota che stavolta il parametro della leakage power è leggermente variato, questo poichè stavolta per rispettare il vincolo sul consumo di potenza è stata variata la topologia del circuito usando differenti porte logiche.

3. Laboratorio 3: Clock gating, pipelining and parallelizing

Durante questa esperienza di laboratorio verranno analizzate una serie di tecniche per ridurre i consumi mediante l'ottimizzazione dell'architettura dei circuiti.

3.1 A first approach to clock gating

Un'altra tecnica utilizzata per ridurre i consumi andando a lavorare sull'architettura è il *Clock Gating*. Questa tecnica permette di "staccare" il clock ad un determinato blocco del mio circuito, quando questo non deve lavorare. Un circuito di massima è riportato in Figura 3.1.

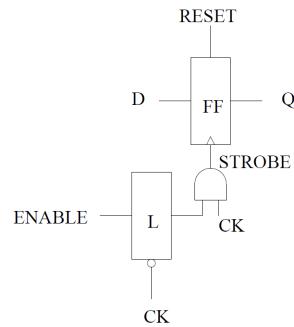


Figure 3.1: Schema implementativo della tecnica del Clock Gating

3.1. A first approach to clock gating

Nella prima parte dell'esperienza viene chiesto di analizzare il file *ckgbug.vhd* che contiene la descrizione VHDL di una struttura composta da due registri in cascata, denominati L1 ed L2. La tecnica del clock gating viene applicata al secondo registro, mediante una AND tra il clock e un segnale di ENABLE. Bisogna prestare attenzione al fatto che i segnali di ingresso dei due registri (D1 e D2) siano rispettivamente *std_logic_vector (7 downto 0)* e *std_logic_vector (0 to 7)*.

In una prima simulazione, si forza il segnale D1 al valore '01111111' e, attivando il segnale di ENABLE, ci si aspetterebbe che D2, al colpo di clock successivo, vada al valore '111111101' e che l'uscita di L2, denominata D3, vada al valore '01111111' al colpo di clock ancora successivo. In realtà la simulazione porta al risultato in Figura 3.2. Si può ben notare come l'uscita

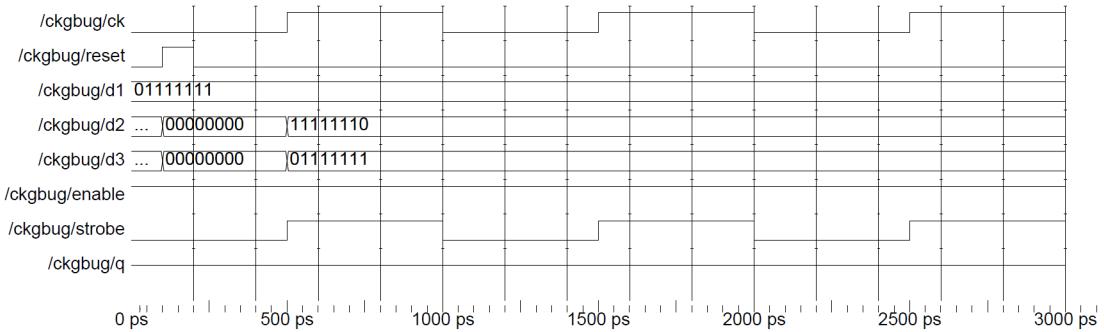


Figure 3.2: Schema implementativo della tecnica del Clock Gating

D3 dopo esattamente D1 dopo un solo colpo di clock. Questo è dovuto al fatto che il clock gated arriva a L2 un "passo di simulazione" dopo L1, perchè il simulatore programma il calcolo dell'uscita AND dopo l'assegnazione del clock. Quindi accade come se l'AND avesse un ritardo interno.

La traccia suggerisce che si può risolvere questo inconveniente andando ad aggiungere un ritardo *Clock-to-Output* pari a 0.1 ps all'uscita del generico Flip-Flop. Andando a risimulare il file, si ottiene il comportamento desiderato, che viene riportato in Figura 3.3.

3.1. A first approach to clock gating

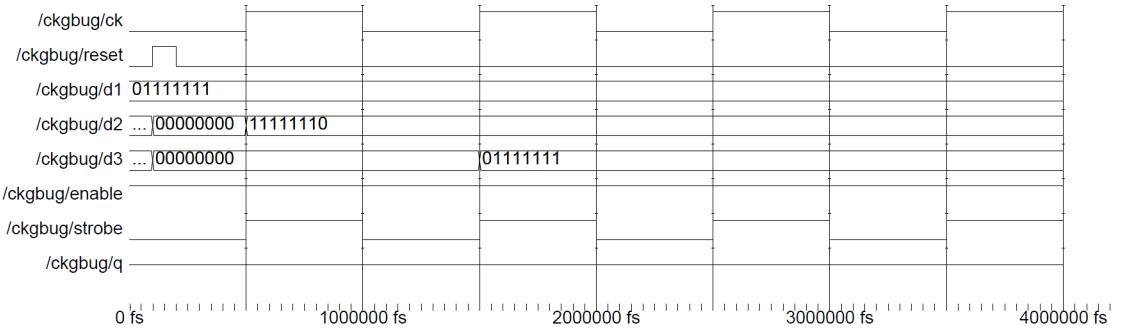


Figure 3.3: Schema implementativo della tecnica del Clock Gating

Infine si aggiunge un ulteriore ritardo alla porta AND pari a 0.2 ps, ossia un tempo superiore a quello *Clock-to-Output* inserito in precedenza. In questo modo si va a violare il t_{hold} e dunque il circuito ritorna nella situazione precedente con un funzionamento non corretto. Il risultato della simulazione è riportato in Figura 3.4.

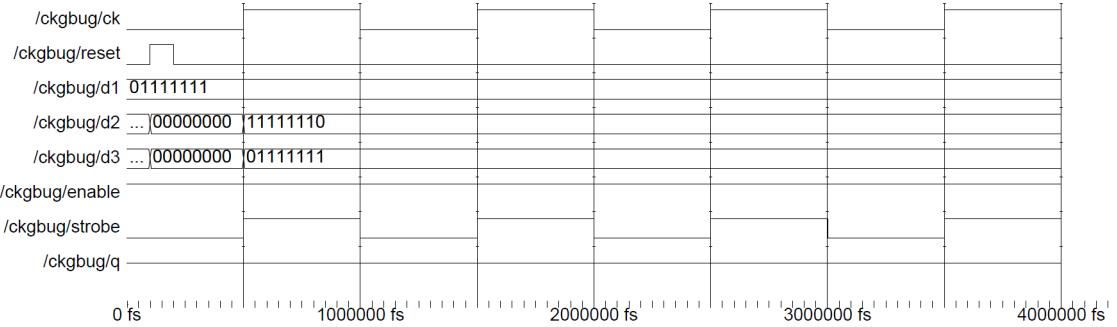


Figure 3.4: Schema implementativo della tecnica del Clock Gating

3.2. Clock Gating for a complex circuit

3.2 Clock Gating for a complex circuit

Nella seguente sezione viene chiesto di analizzare il funzionamento e in seguito il consumo di potenza, applicando il concetto del clock gating, nel circuito illustrato in Figura 3.5.

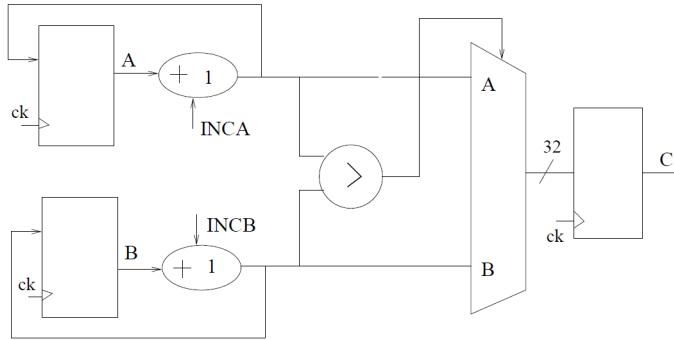


Figure 3.5: Schema implementativo della tecnica del Clock Gating

Si prova, esclusivamente a livello cartaceo ad applicare la tecnica del Clock Gating al circuito in figura, ottenendo il circuito in Figura 3.6.

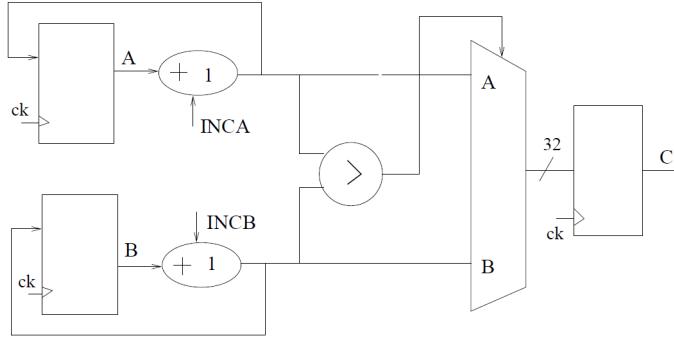


Figure 3.6: Schema implementativo della tecnica del Clock Gating

Si è proceduto dunque alla sintesi del circuito, sempre tramite *Synopsis*, creando un clock con periodo pari a 5 s. Tramite il comando *report_power*

3.2. Clock Gating for a complex circuit

-*include_input_nets* si è potuto ricavare un resoconto dei contributi di potenza dissipata, che vengono riportati in Figura 3.7.

Global Operating Voltage = 1.1					
Power-specific unit information :					
Voltage Units = 1V					
Capacitance Units = 1.000000ff					
Time Units = 1ns					
Dynamic Power Units = 1uW (derived from V,C,T units)					
Leakage Power Units = 1nW					
Cell Internal Power	= 38.7514 uW	(74%)			
Net Switching Power	= 13.6944 uW	(26%)			

Total Dynamic Power	= 52.4459 uW	(100%)			
Cell Leakage Power	= 3.7969 uW				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%) Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)
register	36.0041	0.7880	1.9565e+03	38.7486	(84.01%)
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)
combinational	2.7474	2.7899	1.8405e+03	7.3777	(15.99%)
Total	38.7514 uW	3.5780 uW	3.7969e+03 nW	46.1263 uW	
1					

Figure 3.7: Schema implementativo della tecnica del Clock Gating

Si è poi andato ad eseguire un’analisi più dettagliata dei contributi tramite il comando *report_power -net -include_input_nets*, che viene riportato in Figura 3.8.

Net	Total Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	39.765	0.500	0.4000	9.6231	d
INCA	19.382	0.500	0.0200	0.2345	d
INC _B	19.382	0.500	0.0200	0.2345	d
rst	2.010	0.500	0.0200	0.0243	d
Total (92 nets)				13.5445 uW	
1					

Figure 3.8: Schema implementativo della tecnica del Clock Gating

Come si può ben notare dalla quarta colonna, il toogle rate dei vari ingressi

3.2. Clock Gating for a complex circuit

presenta una *static probability* pari a 0.5, che non risulta essere un valore sempre ragionevole.

Questo valore viene assegnato di default dal software, che però, tramite degli appositi comandi, permette di settare valori diversi di Toogle Rate a determinati segnali. Si è allora modificato il toogle rate del clock e del reset portandoli rispettivamente a 2 e a 0. Queste modifiche portano ad un risparmio di potenza del 20% rispetto al caso precedente. Il report di potenza risultante viene raffigurato in Figura 3.9.

	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
Power Group						
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)	
register	23.5753	1.4790	2.1179e+03	27.1722	(74.37%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	4.0608	3.2429	2.0621e+03	9.3657	(25.63%)	
Total	27.6360 uW	4.7219 uW	4.1801e+03 nW	36.5380 uW		
1						

Figure 3.9: Schema implementativo della tecnica del Clock Gating

Si va ora a modificare anche la probabilità degli input, portando sia INCA che INCB ad un Toogle Rate pari a 0.12. Si ottiene allora un risparmio ulteriore di potenza, come testimoniato da Power Report riportati in Figura 3.10 e 3.11.

3.2. Clock Gating for a complex circuit

Cell Internal Power	=	21.5273 uW	(66%)	
Net Switching Power	=	10.8718 uW	(34%)	
<hr/>				
Total Dynamic Power	=	32.3990 uW	(100%)	
<hr/>				
Cell Leakage Power	=	4.0921 uW		
<hr/>				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power (%) Attrs
io_pad	0.0000	0.0000	0.0000	0.0000 (0.00%)
memory	0.0000	0.0000	0.0000	0.0000 (0.00%)
black_box	0.0000	0.0000	0.0000	0.0000 (0.00%)
clock_network	0.0000	0.0000	0.0000	0.0000 (0.00%)
register	20.5257	0.3486	2.1135e+03	22.9878 (85.93%)
sequential	0.0000	0.0000	0.0000	0.0000 (0.00%)
combinational	1.0016	0.7828	1.9786e+03	3.7629 (14.07%)
Total	21.5273 uW	1.1314 uW	4.0921e+03 nW	26.7508 uW
1				

Figure 3.10: Schema implementativo della tecnica del Clock Gating

Net	Total Load	Static Prob.	Toggle Rate	Switching Power	Attrs
<hr/>					
ck	39.765	0.500	0.4000	9.6231 a	
INCA	19.382	0.120	0.0050	0.0586 a	
INC B	19.382	0.120	0.0050	0.0586 a	
rst	2.010	0.000	0.0000	0.0000 a	
Total (92 nets)				10.7715 uW	
1					

Figure 3.11: Schema implementativo della tecnica del Clock Gating

Infine, tramite il comando *report cell*, si riesce ad ottenere il numero di celle impiegate con la relativa dimensione. Viene riportato in Figura 3.12. Si può ben notare come nel caso analizzato le celle siano 74 e l'area utilizzata sia $229.292007 \mu m^2$

3.2. Clock Gating for a complex circuit

Cell	Reference	Library	Area	Attributes				
C_reg[0]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U82	INV_X1	NangateOpenCellLibrary	0.532000
C_reg[1]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U83	INV_X1	NangateOpenCellLibrary	0.532000
C_reg[2]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U84	INV_X1	NangateOpenCellLibrary	0.532000
C_reg[3]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U85	INV_X1	NangateOpenCellLibrary	0.532000
C_reg[4]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U86	INV_X1	NangateOpenCellLibrary	0.532000
C_reg[5]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U87	MUX2_X1	NangateOpenCellLibrary	1.862000
C_reg[6]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U88	MUX2_X1	NangateOpenCellLibrary	1.862000
C_reg[7]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U89	MUX2_X1	NangateOpenCellLibrary	1.862000
U55	INV_X1	NangateOpenCellLibrary	5.320000	n	U90	MUX2_X1	NangateOpenCellLibrary	1.862000
U56	MUX2_X1	NangateOpenCellLibrary	5.320000	n	U91	MUX2_X1	NangateOpenCellLibrary	1.862000
U57	MUX2_X1	NangateOpenCellLibrary	1.862000		U92	MUX2_X1	NangateOpenCellLibrary	1.862000
U58	MUX2_X1	NangateOpenCellLibrary	1.862000		U93	MUX2_X1	NangateOpenCellLibrary	1.862000
U59	MUX2_X1	NangateOpenCellLibrary	1.862000		U94	MUX2_X1	NangateOpenCellLibrary	1.862000
U60	MUX2_X1	NangateOpenCellLibrary	1.862000		U95	MUX2_X1	NangateOpenCellLibrary	1.862000
U61	MUX2_X1	NangateOpenCellLibrary	1.862000		U96	MUX2_X1	NangateOpenCellLibrary	1.862000
U62	MUX2_X1	NangateOpenCellLibrary	1.862000		U97	MUX2_X1	NangateOpenCellLibrary	1.862000
U63	MUX2_X1	NangateOpenCellLibrary	1.862000		U98	MUX2_X1	NangateOpenCellLibrary	1.862000
U64	AOI21_X1	NangateOpenCellLibrary	1.064000		U99	MUX2_X1	NangateOpenCellLibrary	1.064000
U65	INV_X1	NangateOpenCellLibrary	0.532000		U100	MUX2_X1	NangateOpenCellLibrary	1.064000
U66	AOI221_X1	NangateOpenCellLibrary	1.596000		U101	MUX2_X1	NangateOpenCellLibrary	1.064000
U67	AOI21_X1	NangateOpenCellLibrary	1.596000		U102	MUX2_X1	NangateOpenCellLibrary	1.596000
U68	AOI21_X1	NangateOpenCellLibrary	1.064000	r67		incomp_DW01_inc_1	18.080000	BO, h
U69	AOI221_X1	NangateOpenCellLibrary	1.596000	r68		incomp_DW01_inc_0	18.080000	BO, h
U70	AOI221_X1	NangateOpenCellLibrary	1.596000			DFFR_X1	NangateOpenCellLibrary	0.532000
U71	AOI221_X1	NangateOpenCellLibrary	1.596000			syncha_reg[10]		
U72	AOI221_X1	NangateOpenCellLibrary	1.596000			syncha_reg[1]		
U73	AOI21_X1	NangateOpenCellLibrary	1.596000			syncha_reg[2]		
U74	INV_X1	NangateOpenCellLibrary	1.064000			syncha_reg[3]		
U75	NOR3_X1	NangateOpenCellLibrary	0.532000			syncha_reg[4]		
U76	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[5]		
U77	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[6]		
U78	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[7]		
U79	INV_X1	NangateOpenCellLibrary	0.532000			synchb_reg[0]		
U80	INV_X1	NangateOpenCellLibrary	0.532000			synchb_reg[1]		
U81	INV_X1	NangateOpenCellLibrary	0.532000			synchb_reg[2]		
U82	INV_X1	NangateOpenCellLibrary	0.532000			synchb_reg[3]		
						synchb_reg[4]		
						synchb_reg[5]		
						synchb_reg[6]		
						synchb_reg[7]		
Total 74 cells					1		229.292005	

Figure 3.12: Schema implementativo della tecnica del Clock Gating

Ora si ri-eseguono le medesime simulazioni, ma utilizzando il circuito dove è stato applicata la tecnica del Clock Gating. Inizialmente si sono lasciati tutti i parametri esattamente come la prima analisi, quindi tutte le probabilità pari a 0.5, e si è ottenuto comunque una potenza inferiore del 16% rispetto al caso senza la tecnica del Clock Gating. Il tutto viene riportato in Figura 3.13 e 3.14.

3.2. Clock Gating for a complex circuit

Cell Internal Power	=	32.9555 uW	(75%)			
Net Switching Power	=	10.8986 uW	(25%)			
<hr/>						
Total Dynamic Power	=	43.8542 uW	(100%)			
<hr/>						
Cell Leakage Power	=	3.9340 uW				
<hr/>						
Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	4.3808	2.8803	116.8092	7.3779	(17.01%)	
register	25.7196	0.7909	1.9461e+03	28.4567	(65.62%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	2.8551	2.8057	1.8711e+03	7.5318	(17.37%)	
Total	32.9555 uW	6.4769 uW	3.9340e+03 nW	43.3664 uW		
1						

Figure 3.13: Schema implementativo della tecnica del Clock Gating

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power	Attrs
<hr/>					
ck	16.095	0.500	0.4000	3.8951	d
INCA	20.758	0.500	0.0200	0.2512	d
INCB	20.758	0.500	0.0200	0.2512	d
rst	2.010	0.500	0.0200	0.0243	d
Total (97 nets)				10.7469 uW	
1					

Figure 3.14: Schema implementativo della tecnica del Clock Gating

Esattamente come prima è stata cambiata la probabilità di reset, clock e input per vedere come cambiano i valori di potenza, portandole rispettivamente a 0, 0.5, 0.12. Si è eseguito il comando `report_power -include_input_nets` e si sono analizzati i report di potenza che sono riportati in Figura 3.15 e 3.16.

3.2. Clock Gating for a complex circuit

```

Global Operating Voltage = 1.1
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000ff
  Time Units = 1ns
  Dynamic Power Units = 1uW      (derived from V,C,T units)
  Leakage Power Units = 1nW

```

```

Cell Internal Power = 13.3592 uW (70%)
Net Switching Power = 5.8499 uW (30%)
-----
Total Dynamic Power = 19.2091 uW (100%)

```

```
Cell Leakage Power = 4.2570 uW
```

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	2.8527	0.6731	120.9131	3.6467	(18.75%)	
register	9.4563	0.3485	2.1091e+03	11.9140	(61.27%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	1.0502	0.8075	2.0269e+03	3.8847	(19.98%)	
Total	13.3592 uW	1.8292 uW	4.2570e+03 nW	19.4454 uW		
1						

Figure 3.15: Schema implementativo della tecnica del Clock Gating

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	16.095	0.500	0.4000	3.8951 a	
INCA	20.758	0.120	0.0050	0.0628 a	
INC B	20.758	0.120	0.0050	0.0628 a	
rst	2.010	0.000	0.0000	0.0000 a	
Total (97 nets)				5.7496 uW	
1					

Figure 3.16: Schema implementativo della tecnica del Clock Gating

Si può notare come si è ottenuto un risparmio del 41% rispetto al caso senza Clock Gating.

Infine si ri-esegue l'analisi tramite il report cell, raffigurato in Figura 3.17.

Si può notare come ora il numero di celle sia aumentato e con esso anche l'area.

Rispetto al caso senza Clock Gating ho ora un'area pari a $238.0700005 \mu m^2$ e un numero di celle pari a 78. L'aumento dell'area è uno degli svantaggi della tecnica che Clock Gating, ma comunque, in questa situazione, si tratta di un'incremento pari solo al 3.47%, che risulta assolutamente accettabile visto che comporta un risparmio di potenza pari al 41%.

3.2. Clock Gating for a complex circuit

Cell	Reference	Library	Area	Attributes	U78	MUX2_X1	NangateOpenCellLibrary
C_req(0)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U79	INV_X1	NangateOpenCellLibrary
C_req(1)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U80	MUX2_X1	NangateOpenCellLibrary
C_req(2)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U81	INV_X1	NangateOpenCellLibrary
C_req(3)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U82	MUX2_X1	NangateOpenCellLibrary
C_req(4)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U83	INV_X1	NangateOpenCellLibrary
C_req(5)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U84	MUX2_X1	NangateOpenCellLibrary
C_req(6)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U85	INV_X1	NangateOpenCellLibrary
C_req(7)	DFFR_X1	NangateOpenCellLibrary	.5_320000	n	U86	MUX2_X1	NangateOpenCellLibrary
U49	INV_X1	NangateOpenCellLibrary	.5_320000		U87	INV_X1	NangateOpenCellLibrary
U50	MUX2_X1	NangateOpenCellLibrary	.862000		U88	MUX2_X1	NangateOpenCellLibrary
U51	MUX2_X1	NangateOpenCellLibrary	.862000		U89	MUX2_X1	NangateOpenCellLibrary
U52	MUX2_X1	NangateOpenCellLibrary	.862000		U90	INV_X1	NangateOpenCellLibrary
U53	MUX2_X1	NangateOpenCellLibrary	.862000		U91	MUX2_X1	NangateOpenCellLibrary
U54	MUX2_X1	NangateOpenCellLibrary	.862000		U92	INV_X1	NangateOpenCellLibrary
U55	MUX2_X1	NangateOpenCellLibrary	.862000		U93	MUX2_X1	NangateOpenCellLibrary
U56	MUX2_X1	NangateOpenCellLibrary	.862000		U94	MUX2_X1	NangateOpenCellLibrary
U57	MUX2_X1	NangateOpenCellLibrary	.862000		U95	INV_X1	NangateOpenCellLibrary
U58	AO121_X1	NangateOpenCellLibrary	.1_064000		U96	MUX2_X1	NangateOpenCellLibrary
U59	AO1221_X1	NangateOpenCellLibrary	.1_064000		U97	MUX2_X1	NangateOpenCellLibrary
U60	INV_X1	NangateOpenCellLibrary	.1_596000		U98	MUX2_X1	NangateOpenCellLibrary
U61	AO121_X1	NangateOpenCellLibrary	.1_064000			clk_gate_syncha_reg	SNPF_CLOCK_GATE_HIGH_Incomp_0 3.990000 b, cg, h
U62	AO121_X1	NangateOpenCellLibrary	.1_064000			clk_gate_synchb_reg	SNPF_CLOCK_GATE_HIGH_Incomp_1 3.990000 b, cg, h
U63	INV_X1	NangateOpenCellLibrary	.1_064000			r67	inccomp_DW01_inc_1 18.088000 B0, h
U64	AO1221_X1	NangateOpenCellLibrary	.1_064000			r68	inccomp_DW01_inc_0 18.088000 B0, h
U65	AO1221_X1	NangateOpenCellLibrary	.1_596000			syncha_reg[0]	DFFR_X1 NangateOpenCellLibrary
U66	AO1221_X1	NangateOpenCellLibrary	.1_596000				
U67	AO1221_X1	NangateOpenCellLibrary	.1_596000			syncha_reg[1]	DFFR_X1 NangateOpenCellLibrary
U68	AO121_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[2]	DFFR_X1 NangateOpenCellLibrary
U69	MUX2_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[3]	DFFR_X1 NangateOpenCellLibrary
U70	INV_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[4]	DFFR_X1 NangateOpenCellLibrary
U71	MUX2_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[5]	DFFR_X1 NangateOpenCellLibrary
U72	INV_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[6]	DFFR_X1 NangateOpenCellLibrary
U73	NOR2_X1	NangateOpenCellLibrary	.1_064000			syncha_reg[7]	DFFR_X1 NangateOpenCellLibrary
U74	INV_X1	NangateOpenCellLibrary	.1_064000				
U75	INV_X1	NangateOpenCellLibrary	.1_064000				
U76	MUX2_X1	NangateOpenCellLibrary	.1_064000				
U77	INV_X1	NangateOpenCellLibrary	.1_064000				
							Total 78 cells
							238.070005
					1		

Figure 3.17: Schema implementativo della tecnica del Clock Gating

3.2.1 Some more clock gating?

Il codice VHDL fornito è scritto in modo tale che il sintetizzatore non applichi la tecnica del Clock Gating sul registro di uscita: la motivazione risiede nell'utilizzo del costrutto *if-else*, in quanto nel codice iniziale non veniva dichiarati tutti i possibili casi tramite *elsif*. Modificando opportunamente il codice VHDL, riportato per interezza in Figura 3.18, e inserendo le linee di codice riportate in Figura 3.19, il sintetizzatore inserisce un blocco di clock gating sul registro di uscita perchè interpreta che potrebbero esserci

3.2. Clock Gating for a complex circuit

condizioni in cui non si andrà a scrivere sul registro di uscita.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;

entity inccomp is
Port(C: out std_logic_vector(7 downto 0);
      ck: in std_logic;
      rst: in std_logic;
      INCA: in std_logic;
      INCB: in std_logic);
end inccomp;

architecture behavioral of inccomp is
signal syncha, synchb : std_logic_vector(7 downto 0);
begin
begin
    p1: process(ck,rst)
variable tmpa, tmpb : std_logic_vector(7 downto 0);
begin
    if rst='1' then
        syncha <= (others => '0');
        synchb <= (others => '0');
        C <= (others => '0');
    elsif ck'event and ck='1' then
        tmpa:= syncha;
        tmpb:= synchb;
        if INCA='1' then
            syncha <= syncha+1;
            tmpa:= tmpa+1;
        end if;
        if INCB='1' then
            synchb <= synchb+1;
            tmpb:= tmpb+1;
        end if;
        if ((tmpa)>(tmpb)) then
            C <= tmpa;
        elsif ((tmpb)>(tmpa)) then
            C<=tmpb;
        else
            NULL;
        end if;
    end if;
    end process;
end behavioral;

```

Figure 3.18: *Schema implementativo della tecnica del Clock Gating*

```

elsif ((tmpb)>(tmpa)) then
C<=tmpb;

```

Figure 3.19: *Schema implementativo della tecnica del Clock Gating*

Le modifiche al VHDL hanno portato al risultato richiesto, come si può notare in Figura 3.20 che riporta il circuito sintetizzato e in Figura 3.21 che riporta il particolare del blocco di Clock Gating inserito dal sintetizzatore.

3.2. Clock Gating for a complex circuit

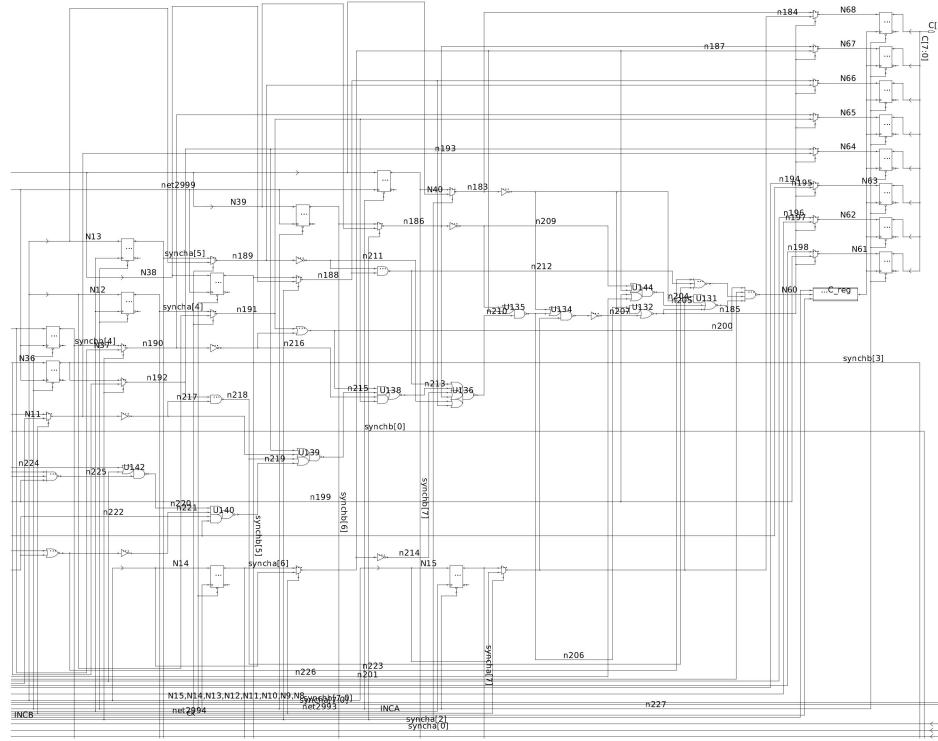


Figure 3.20: Schema implementativo della tecnica del Clock Gating

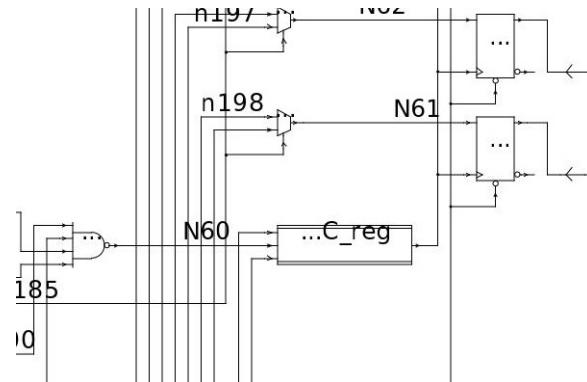


Figure 3.21: Schema implementativo della tecnica del Clock Gating

3.2. Clock Gating for a complex circuit

Si eseguono ora le analisi dei consumi, esattamente come nel punto precedente.

Le prima analisi viene svolta senza la tecnica del Clock Gating e vengono lasciati i valori di Toggle Rate di default, quindi a 0.5. I power report ricavati sono riportati in Figura 3.22 e 3.23. Si può ben notare come rispetto al caso

Global Operating Voltage = 1.1 Power-specific unit information :					
	Voltage Units = 1V	Capacitance Units = 1.000000ff	Time Units = 1ns	Dynamic Power Units = 1uW	(derived from V,C,T units)
	Leakage Power Units = 1nW				
Cell Internal Power	= 38.7833 uW	(72%)			
Net Switching Power	= 14.9396 uW	(28%)			
Total Dynamic Power	= 53.7229 uW	(100%)			
Cell Leakage Power	= 4.4703 uW				
<hr/>					
Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%) Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)
memory	0.0000	0.0000	0.0000	0.0000	(0.00%)
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)
register	35.8409	0.9237	1.9549e+03	38.7195	(80.54%)
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)
combinational	2.9424	3.8994	2.5154e+03	9.3572	(19.46%)
Total	38.7833 uW	4.8231 uW	4.4703e+03 nW	48.0767 uW	
1					

Figure 3.22: Schema implementativo della tecnica del Clock Gating

Net	Total Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	39.765	0.500	0.4000	9.6231	d
INCA	19.382	0.500	0.0200	0.2345	d
INC B	19.382	0.500	0.0200	0.2345	d
rst	2.010	0.500	0.0200	0.0243	d
Total (112 nets)				14.7896	uW
1					

Figure 3.23: Schema implementativo della tecnica del Clock Gating

precedente, la potenza sia leggermente aumentata in quanto, avendo aggiunto un blocco di Clock Gating prima del registro di uscita, sono aumentati i gate totali che quindi portano ad una maggiore dissipazione di potenza. Viene ora modificato il Toggle Rate del clock, del reset e degli ingressi, ottenendo i report in Figura 3.24 e 3.25. L'aggiunta del blocco in uscita porta

3.2. Clock Gating for a complex circuit

ad un primo risparmio di potenza pari a XX% rispetto al caso analizzato al punto precedente.

```

Global Operating Voltage = 1.1
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000ff
  Time Units = 1ns
  Dynamic Power Units = 1uW    (derived from V,C,T units)
  Leakage Power Units = 1nW

  Cell Internal Power = 38.7833 uW (72%)
  Net Switching Power = 14.9396 uW (28%)
  -----
  Total Dynamic Power = 53.7229 uW (100%)
  Cell Leakage Power = 4.4703 uW

  Power Group      Internal          Switching          Leakage          Total
                    Power            Power            Power          Power ( % ) Attrs
  -----
  io_pad           0.0000           0.0000           0.0000        0.0000 ( 0.00%)
  memory          0.0000           0.0000           0.0000        0.0000 ( 0.00%)
  black_box        0.0000           0.0000           0.0000        0.0000 ( 0.00%)
  clock_network   0.0000           0.0000           0.0000        0.0000 ( 0.00%)
  register         35.8409          0.9237          1.9549e+03    38.7195 ( 80.54%)
  sequential        0.0000           0.0000           0.0000        0.0000 ( 0.00%)
  combinational    2.9424          3.8994          2.5154e+03    9.3572 ( 19.46%)
  -----
  Total           38.7833 uW       4.8231 uW       4.4703e+03 nW     48.0767 uW
  1

```

Figure 3.24: Schema implementativo della tecnica del Clock Gating

Net	Total Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	39.765	0.500	0.4000	9.6231	d
INCA	19.382	0.500	0.0200	0.2345	d
INCB	19.382	0.500	0.0200	0.2345	d
rst	2.010	0.500	0.0200	0.0243	d
Total (112 nets)				14.7896	uW
1					

Figure 3.25: Schema implementativo della tecnica del Clock Gating

Si esegue infine il comando *report cell* per ottenere l'area e il numero di celle utilizzate per la sintesi. Il report viene raffigurato i Figura 3.26.

3.2. Clock Gating for a complex circuit

Cell	Reference	Library	Area	Attributes	Cell	Reference	Library	Area	Attributes
C_reg[0]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U101		NAND2_X1	NangateOpenCellLibrary	0.798000
C_reg[1]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U102		NOR3_X1	NangateOpenCellLibrary	1.064000
C_reg[2]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U103		INV_X1	NangateOpenCellLibrary	0.532000
C_reg[3]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U104		INV_X1	NangateOpenCellLibrary	0.532000
C_reg[4]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U105		INV_X1	NangateOpenCellLibrary	0.532000
C_reg[5]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U106		INV_X1	NangateOpenCellLibrary	0.532000
C_reg[6]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U107		INV_X1	NangateOpenCellLibrary	0.532000
C_reg[7]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U108		NOR2_X1	NangateOpenCellLibrary	0.798000
U67	INV_X1	NangateOpenCellLibrary	0.532000		U109		INV_X1	NangateOpenCellLibrary	0.532000
U68	OAI222_X1	NangateOpenCellLibrary	2.128000		U110		INV_X1	NangateOpenCellLibrary	0.532000
U69	OAI222_X1	NangateOpenCellLibrary	2.128000		U111		MUX2_X1	NangateOpenCellLibrary	1.862000
U70	INV_X1	NangateOpenCellLibrary	0.532000		U112		MUX2_X1	NangateOpenCellLibrary	1.862000
U71	OAI222_X1	NangateOpenCellLibrary	2.128000		U113		MUX2_X1	NangateOpenCellLibrary	1.862000
U72	INV_X1	NangateOpenCellLibrary	0.532000		U114		MUX2_X1	NangateOpenCellLibrary	1.862000
U73	OAI222_X1	NangateOpenCellLibrary	2.128000		U115		MUX2_X1	NangateOpenCellLibrary	1.862000
U74	INV_X1	NangateOpenCellLibrary	0.532000		U116		MUX2_X1	NangateOpenCellLibrary	1.862000
U75	OAI222_X1	NangateOpenCellLibrary	2.128000		U117		MUX2_X1	NangateOpenCellLibrary	1.862000
U76	INV_X1	NangateOpenCellLibrary	0.532000		U118		MUX2_X1	NangateOpenCellLibrary	1.862000
U77	OAI222_X1	NangateOpenCellLibrary	2.128000		U119		MUX2_X1	NangateOpenCellLibrary	1.862000
U78	INV_X1	NangateOpenCellLibrary	0.532000		U120		MUX2_X1	NangateOpenCellLibrary	1.862000
U79	OAI222_X1	NangateOpenCellLibrary	2.128000		U121		MUX2_X1	NangateOpenCellLibrary	1.862000
U80	INV_X1	NangateOpenCellLibrary	0.532000		U122		MUX2_X1	NangateOpenCellLibrary	1.862000
U81	OAI222_X1	NangateOpenCellLibrary	2.128000		U123		MUX2_X1	NangateOpenCellLibrary	1.862000
U82	NAND2_X1	NangateOpenCellLibrary	0.798000		U124		MUX2_X1	NangateOpenCellLibrary	1.862000
U83	INV_X1	NangateOpenCellLibrary	0.532000		U125		MUX2_X1	NangateOpenCellLibrary	1.862000
U84	NAND4_X1	NangateOpenCellLibrary	1.330000		U126		MUX2_X1	NangateOpenCellLibrary	1.862000
U85	AOI221_X1	NangateOpenCellLibrary	1.596000		r72	inccomp_DW01_inc_1		18.088000	BO, h
U86	OAI22_X1	NangateOpenCellLibrary	1.330000		r73	inccomp_DW01_inc_0		18.088000	BO, h
U87	INV_X1	NangateOpenCellLibrary	0.532000		syncha_reg[0]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U88	AND3_X1	NangateOpenCellLibrary	0.532000		syncha_reg[1]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U89	OAI21_X1	NangateOpenCellLibrary	1.064000		syncha_reg[2]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U90	OAI22_X1	NangateOpenCellLibrary	1.330000		syncha_reg[3]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U91	INV_X1	NangateOpenCellLibrary	0.532000		syncha_reg[4]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U92	AOI22_X1	NangateOpenCellLibrary	1.330000		syncha_reg[5]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U93	NAND2_X1	NangateOpenCellLibrary	0.798000		syncha_reg[6]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U94	OAI221_X1	NangateOpenCellLibrary	1.596000		syncha_reg[7]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U95	NAND3_X1	NangateOpenCellLibrary	1.064000		synchb_reg[0]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U96	NAND2_X1	NangateOpenCellLibrary	0.798000		synchb_reg[1]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U97	NAND2_X1	NangateOpenCellLibrary	0.798000		synchb_reg[2]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U98	OAI221_X1	NangateOpenCellLibrary	1.596000		synchb_reg[3]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U99	OAI211_X1	NangateOpenCellLibrary	1.330000		synchb_reg[4]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
U100	OAI21_X1	NangateOpenCellLibrary	1.064000		synchb_reg[5]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
					synchb_reg[6]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
					synchb_reg[7]	DFFR_X1	NangateOpenCellLibrary	5.320000	n
Total 86 cells					1				241.529005

Figure 3.26: Schema implementativo della tecnica del Clock Gating

Si può ben notare che ora il numero di celle è leggermente aumentato e di conseguenza anche l'area, in quanto si è aggiunta una logica prima del registro di uscita.

3.2. Clock Gating for a complex circuit

Si analizza allo stesso modo il circuito con l'aggiunta della tecnica del Clock Gating. Tutti i report, con lo stesso ordine dei punti precedenti, vengono riportati nelle Figure 3.27, 3.28, 3.29, 3.30.

```

Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW      (derived from V,C,T units)
    Leakage Power Units = 1nW

Cell Internal Power   =  31.8459 uW  (74%)
Net Switching Power  =  10.9320 uW  (26%)
-----
Total Dynamic Power   =  42.7779 uW  (100%)
Cell Leakage Power   =   4.3939 uW

          Internal           Switching           Leakage           Total
Power Group     Power            Power            Power            Power  ( % )  Attrs
-----
io_pad         0.0000          0.0000          0.0000          0.0000 ( 0.00%)
memory         0.0000          0.0000          0.0000          0.0000 ( 0.00%)
black_box       0.0000          0.0000          0.0000          0.0000 ( 0.00%)
clock_network  6.8165          4.7454        174.3911        11.7364 ( 26.05%)
register        22.0516         0.7807        1.9403e+03      24.7727 ( 54.99%)
sequential       0.0000          0.0000          0.0000          0.0000 ( 0.00%)
combinational   2.9778          3.2860        2.2792e+03      8.5430 ( 18.96%)
-----
Total           31.8459 uW      8.8122 uW      4.3939e+03 nW      45.0520 uW
1

```

Figure 3.27: Schema implementativo della tecnica del Clock Gating

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	6.583	0.500	0.4000	1.5931	d
INCA	20.758	0.500	0.0200	0.2512	d
INCb	20.758	0.500	0.0200	0.2512	d
rst	2.010	0.500	0.0200	0.0243	d
Total (119 nets)				10.7803	uW
1					

Figure 3.28: Schema implementativo della tecnica del Clock Gating

3.2. Clock Gating for a complex circuit

```

Global Operating Voltage = 1.1
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000ff
  Time Units = 1ns
  Dynamic Power Units = 1uW      (derived from V,C,T units)
  Leakage Power Units = 1nW

Cell Internal Power = 16.0938 uW (71%)
Net Switching Power = 6.5875 uW (29%)
-----
Total Dynamic Power = 22.6813 uW (100%)

Cell Leakage Power = 4.5746 uW

          Internal           Switching           Leakage           Total
Power Group    Power        Power       Power      Power   (   %   )   Attrs
-----
io_pad         0.0000     0.0000     0.0000     0.0000 (  0.00%)
memory         0.0000     0.0000     0.0000     0.0000 (  0.00%)
black_box      0.0000     0.0000     0.0000     0.0000 (  0.00%)
clock_network  5.5979    3.5161    176.6525    9.2907 ( 36.38%)
register       9.3388    0.3469    2.1096e+03  11.7953 ( 46.19%)
sequential     0.0000     0.0000     0.0000     0.0000 (  0.00%)
combinational  1.1572    1.0058    2.2883e+03  4.4513 ( 17.43%)
-----
Total          16.0938 uW    4.8688 uW    4.5746e+03 nW  25.5373 uW
1

```

Figure 3.29: Schema implementativo della tecnica del Clock Gating

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power	Attrs
ck	6.583	0.500	0.4000	1.5931	a
INCA	20.758	0.120	0.0050	0.0628	a
INC B	20.758	0.120	0.0050	0.0628	a
rst	2.010	0.000	0.0000	0.0000	a
Total (119 nets)				6.4872	uW
1					

Figure 3.30: Schema implementativo della tecnica del Clock Gating

Si può ben notare come nell'ultimo caso, quindi con i Toggle Rate modificati rispetto ai valori di default, la potenza sia aumentata del 18% rispetto al medesimo caso ma senza il blocco di Clock Gating prima del registro di uscita. Ovviamente se invece si confronta l'ultimo risultato con il risultato dell'analisi senza il Clock Gating, ma con il blocco in uscita, si ottiene un risparmio di potenza considerevole.

Come ultima analisi si esegue il *report cell* di quest'ultimo caso, riportato in Figura 3.31, che testimonia come l'applicazione di questa tecnica comporti un aumento dell'area occupata a causa dell'inserimento di una logica di controllo.

3.2. Clock Gating for a complex circuit

Cell	Reference	Library	Area	Attributes			
C_reg[0]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U90	INV_X1	NangateOpenCellLibrary
C_reg[1]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U91	MUX2_X1	NangateOpenCellLibrary
C_reg[2]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U92	INV_X1	NangateOpenCellLibrary
C_reg[3]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U93	NOR3_X1	NangateOpenCellLibrary
C_reg[4]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U94	MUX2_X1	NangateOpenCellLibrary
C_reg[5]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U95	MUX2_X1	NangateOpenCellLibrary
C_reg[6]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U96	INV_X1	NangateOpenCellLibrary
C_reg[7]	DFFR_X1	NangateOpenCellLibrary	5.320000	n	U97	MUX2_X1	NangateOpenCellLibrary
U52	INV_X1	NangateOpenCellLibrary	0.532000		U98	MUX2_X1	NangateOpenCellLibrary
U53	MUX2_X1	NangateOpenCellLibrary	1.862000		U99	MUX2_X1	NangateOpenCellLibrary
U54	MUX2_X1	NangateOpenCellLibrary	1.862000		U100	INV_X1	NangateOpenCellLibrary
U55	MUX2_X1	NangateOpenCellLibrary	1.862000		U101	INV_X1	NangateOpenCellLibrary
U56	INV_X1	NangateOpenCellLibrary	0.532000		U102	NAND2_X1	NangateOpenCellLibrary
U57	MUX2_X1	NangateOpenCellLibrary	1.862000		U103	MUX2_X1	NangateOpenCellLibrary
U58	INV_X1	NangateOpenCellLibrary	0.532000		U104	MUX2_X1	NangateOpenCellLibrary
U59	MUX2_X1	NangateOpenCellLibrary	1.862000		U105	INV_X1	NangateOpenCellLibrary
U60	INV_X1	NangateOpenCellLibrary	0.532000		U106	MUX2_X1	NangateOpenCellLibrary
U61	MUX2_X1	NangateOpenCellLibrary	1.862000		U107	MUX2_X1	NangateOpenCellLibrary
U62	INV_X1	NangateOpenCellLibrary	0.532000		U108	INV_X1	NangateOpenCellLibrary
U63	MUX2_X1	NangateOpenCellLibrary	1.862000		U109	MUX2_X1	NangateOpenCellLibrary
U64	MUX2_X1	NangateOpenCellLibrary	1.862000		U110	INV_X1	NangateOpenCellLibrary
U65	INV_X1	NangateOpenCellLibrary	0.532000		U111	INV_X1	NangateOpenCellLibrary
U66	NAND4_X1	NangateOpenCellLibrary	1.330000		U112	MUX2_X1	NangateOpenCellLibrary
U67	AND4_X1	NangateOpenCellLibrary	1.596000		U113	INV_X1	NangateOpenCellLibrary
U68	AOI22_X1	NangateOpenCellLibrary	1.330000				
U69	AOI22_X1	NangateOpenCellLibrary	1.330000			clk_gate_C_reg	SNPS_CLOCK_GATE_HIGH_inccomp_1 3.290000 b, cg, h
U70	AOI22_X1	NangateOpenCellLibrary	1.330000			clk_gate_syncha_reg	SNPS_CLOCK_GATE_HIGH_inccomp_2 3.290000 b, cg, h
						clk_gate_synchb_reg	SNPS_CLOCK_GATE_HIGH_inccomp_3 3.990000 b, cg, h
U71	INV_X1	NangateOpenCellLibrary	0.532000			r72	inccomp_DW01_inc_1 18.088000 BO, n
U72	INV_X1	NangateOpenCellLibrary	0.532000			r73	inccomp_DW01_inc_0 18.088000 BO, n
U73	AOI22_X1	NangateOpenCellLibrary	1.330000				
U74	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[0]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U75	MUX2_X1	NangateOpenCellLibrary	1.862000			syncha_reg[1]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U76	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[2]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U77	INV_X1	NangateOpenCellLibrary	0.532000			syncha_reg[3]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U78	MUX2_X1	NangateOpenCellLibrary	1.862000			syncha_reg[4]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U79	NAND2_X1	NangateOpenCellLibrary	0.798000			syncha_reg[5]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U80	AOI22_X1	NangateOpenCellLibrary	2.128000			syncha_reg[6]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U81	AOI22_X1	NangateOpenCellLibrary	1.330000			syncha_reg[7]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U82	OR2_X1	NangateOpenCellLibrary	1.064000				
U83	MUX2_X1	NangateOpenCellLibrary	1.862000			synchb_reg[0]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U84	INV_X1	NangateOpenCellLibrary	0.532000			synchb_reg[1]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U85	MUX2_X1	NangateOpenCellLibrary	1.862000			synchb_reg[2]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U86	AOI21_X1	NangateOpenCellLibrary	1.064000			synchb_reg[3]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U87	AOI21_X1	NangateOpenCellLibrary	1.330000			synchb_reg[4]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U88	AOI21_X1	NangateOpenCellLibrary	1.064000			synchb_reg[5]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
U89	NAND2_X1	NangateOpenCellLibrary	0.798000			synchb_reg[6]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
						synchb_reg[7]	DFFR_X1 NangateOpenCellLibrary 5.320000 n
					Total 91 cells		251.902005
					1		

Figure 3.31: Schema implementativo della tecnica del Clock Gating

Per sintetizzare si confronta la potenza dinamica totale prima e dopo l'introduzione del Clock Gating nel registro di uscita e si riportano i valori nella Tabella 3.1. Si può ben notare come la potenza nel secondo caso sia leggermente

3.3. Pipelining and parallelizing

aumentata in quanto si introduce una logica di controllo più complessa che impatta sulla switching activity della macchina.

	Potenza Dinamica Totale
Prima	$19.2091\mu W$
Dopo	$22.6813\mu W$

Table 3.1: *Risultati tempi NAND simulazione ELDO*

3.2.2 An automatic way to annotate activities

3.3 Pipelining and parallelizing

Nella terza sezione dell'esercitazione si vede come si riesce a ridurre la potenza utilizzando la parallelizzazione e/o il pipelining. Si considera nuovamente il datapath del punto precedente, in Figura 3.5, di cui le caratteristiche di ritardo, potenza e occupazione di area sono riportate nella Tabella 3.2 nel caso in cui $V_{DD} = 1V$ e $f_{CLK} = 5MHz$.

Cell Type	Delay (ns)	Power ($\mu W@1V, 5MHz$)	Area (μm^2)
REGISTER	2.0 (CK- iQ)	0.6	319
INCREMENT	40.0	2.55	256.0
COMPARATOR	84.0	2.16	161.0
MUX	14.0	1.67	117.0

Table 3.2: *Risultati tempi NAND simulazione ELDO*

Utilizzando questi dati è possibile ricavare il percorso critico del circuito e da questo la massima frequenza di funzionamento.

Il percorso critico risulta essere causato dalla catena composta da registro di ingresso, incrementatore, comparatore, mux, registro di uscita. Viene allora calcolato il valore esatto del ritardo e la sua conseguente frequenza massima di funzionamento e da ciò si può ricavare anche la potenza totale consumata, in quanto esiste una dipendenza lineare tra potenza e frequenza. Tutti i dati sono raccolti in Tabella 3.3.

3.3. Pipelining and parallelizing

Original Solution	
Delay (critical path)	142 ns
Allowed Clock Frequency	7.0422 MHz
Power	15.1125 μ W
Area	1747 μ m ²

Table 3.3: *Risultati datapath "originale"*

Si prova ora a parallelizzare il circuito per ridurre il consumo di potenza ma avendo lo stesso throughput. Si duplica allora il datapath, raddoppiando dunque le capacità che commutano, ma si riesce a lavorare ad una frequenza dimezzata. Il nuovo circuito è raffigurato in Figura 3.32

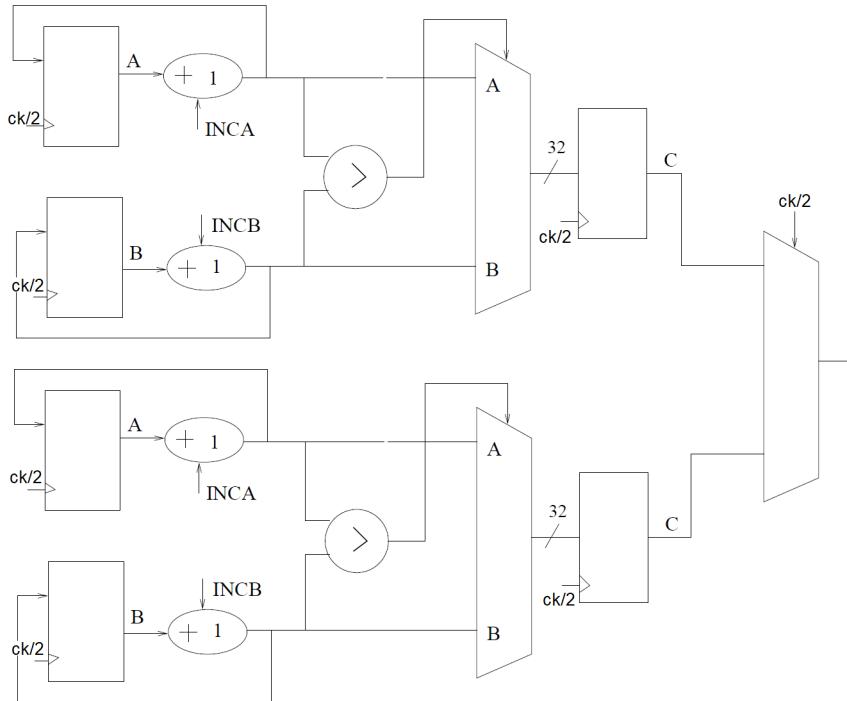


Figure 3.32: *Configurazione datapath con parallelizzazione a due livelli*

I due datapath lavorano dunque in parallelo, campionando uno sul fronte di salita del clock e l'altro sul fronte di discesa. Di conseguenza la frequenza operativa diventa ora esattamente la metà del caso precedente, dunque $f_{CK} = 3.5211MHz$ a cui corrisponde un periodo di clock pari a $T_{CK} = 284ns$.

3.3. Pipelining and parallelizing

Ovviamente, nonostante la parallelizzazione, il percorso critico del circuito rimane pari a 140 ns e dunque si può pensare di allungarlo fino al valore del periodo del T_{CK} in modo da poter scalare la tensione di alimentazione V_{DD} . Per il calcolo del nuovo valore di alimentazione, vengono fornite delle formule che legano il periodo di clock alla tensione di alimentazione normalizzata rispetto a quella nominale (contrassegnata dalla variabile u). Si riportano in seguito i calcoli sviluppati per ottenere la nuova tensione di alimentazione normalizzata:

$$\begin{aligned} T(u) &= T(V_{DD} = V_{DD,NORM}) \frac{0.75u}{u-0.25} = 284ns \implies \\ \implies u &= \frac{0.25T(u)}{T(u)-0.75T(V_{DD}=V_{DD,NORM})} \implies \\ \implies u &= \frac{0.25 \times 284}{284 - 0.75 \times 142} = 0.4 \end{aligned}$$

Ottenuto il valore $u = 0.4$, si può ricavare la potenza dissipata da ogni componente tramite l'equazione:

$$P(V_{DD,new}) = P(V_{DD,NOM}) \times u^2$$

Dove per ottenere ottenere i valori di $P(V_{DD,NOM})$ alla frequenza desiderata, ossia $f_{CK} = 3.5211MHz$, si sfrutta la linearità tra potenza e frequenza.

$$P(V_{DD,NOM}) = P(@V_{DD} = 1V, f_{CK} = 5MHz) \times \frac{3.5211MHz}{5MHz}$$

Si riportano nella Tabella 3.4 i nuovi valori di potenza per ogni singolo blocco.

Cell Type	$P(V_{DD,new})$
REGISTER	$0.4225\mu W \times 0.4^2 = 0.0676\mu W$
INCREMENT	$1.7958\mu W \times 0.4^2 = 0.2873\mu W$
COMPARATOR	$1.5211\mu W \times 0.4^2 = 0.2534\mu W$
MUX	$1.1760\mu W \times 0.4^2 = 0.1882\mu W$

Table 3.4: *Risultati datapath "originale"*

Nella Tabella 3.5, sono riportati ora i risultati riferiti alla parallelizzazione del circuito.

3.3. Pipelining and parallelizing

Parallel Solution	
Delay (critical path)	284 ns
Allowed Clock Frequency	3.5211 MHz
Power	2.6262 μW
Area	3611 μm^2

Table 3.5: *Risultati datapath "parallelizzato"*

Si può notare che tramite la tecnica della parallelizzazione si ottiene un risparmio dell'82% rispetto al caso originale, mentre l'area risulta all'incirca raddoppiata.

Si prova ora ad applicare allo stesso modo la tecnica del pipelining che consiste nello spezzare i percorsi combinatori inserendo alcuni registri: in questo modo si riduce drasticamente il percorso critico.

Il nuovo circuito è rappresentato in Figura 3.33.

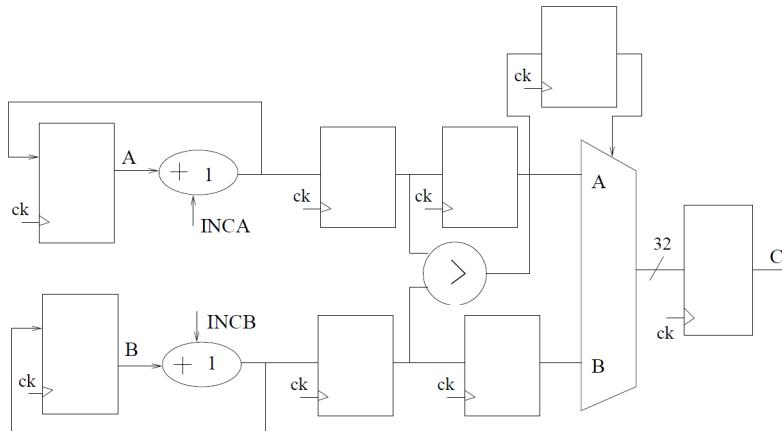


Figure 3.33: *Configurazione datapath pipelining*

Con questa nuova configurazione il percorso critico diventa quello tra il secondo registro e il comparatore.

$$T_{critico} = T_{CK \rightarrow Q} + T_{COMP} = 2ns + 84ns = 86ns$$

Con questo percorso critico si potrebbe arrivare ad una frequenza di clock pari a $f_{CK} = 11.6279MHz$, ma così facendo si otterrebbero miglioramenti

3.3. Pipelining and parallelizing

solo nell'ambito della velocità di funzionamento del circuito. L'obiettivo è sfruttare il pipelining per ridurre il consumo senza aumentare le prestazioni: di conseguenza si lascia la frequenza pari a $f_{CK} = 7.0422MHz$.

In questo modo il percorso critico risulterà comunque pari sempre a $142ns$ e si potrà scalare la tensione di alimentazione V_{DD} .

Si eseguono dunque gli stessi calcoli fatti per il caso della parallelizzazione.

$$\begin{aligned}
 T(u) &= T(V_{DD} = V_{DD,NORM}) \frac{0.75u}{u-0.25} = 142ns \implies \\
 \implies u &= \frac{0.25T(u)}{T(u)-0.75T(V_{DD}=V_{DD,NORM})} \implies \\
 \implies u &= \frac{0.25 \times 142}{142 - 0.75 \times 86} = 0.4581 \\
 P(V_{DD,new}) &= P(V_{DD,NOM}) \times u^2 \\
 P(V_{DD,NOM}) &= P(@V_{DD} = 1V, f_{CK} = 5MHz) \times \frac{7.0422MHz}{5MHz}
 \end{aligned}$$

Tutti i nuovi valori di potenza dissipata dei singoli blocchi vengono riportati nella Tabella 3.6

Cell Type	$P(V_{DD,new})$
REGISTER	$0.0448\mu W \times 0.4581^2 = 0.1773\mu W$
INCREMENT	$3.5910\mu W \times 0.4581^2 = 0.7536\mu W$
COMPARATOR	$3.0422\mu W \times 0.4581^2 = 0.6384\mu W$
MUX	$2.3521\mu W \times 0.4581^2 = 0.4936\mu W$

Table 3.6: *Risultati datapath pipelinato*

Nella Tabella 3.7, sono riportati ora i risultati riferiti alla tecnica del pipelining applicata al circuito.

Parallel Solution	
Delay (critical path)	142 ns
Allowed Clock Frequency	7.0411 MHz
Power	$4.0576\mu W$
Area	$3342\mu m^2$

Table 3.7: *Risultati datapath "parallelizzato"*

Si può ben notare come, grazie alla tecnica del pipelining si ottiene un

3.3. Pipelining and parallelizing

risparmio del 73% rispetto al caso "originale".

Confrontando le due tecniche si può notare come ci sia all'incirca un fattore 2 di differenza tra le potenze risultanti che rende più efficace il metodo della parallelizzazione rispetto al metodo del pipelining. Per quanto riguarda l'area occupata, le due tecniche risultano all'incirca equivalenti, in quanto portano ad un raddoppio dell'area rispetto al circuito originale.

Si può pensare ora di sfruttare entrambe le tecniche per scalare il percorso critico, facendolo passare da 86 ns a 284 ns in modo da poter ridurre ancora la tensione di alimentazione e di conseguenza la potenza consumata. Il nuovo circuito è rappresentato in Figura 3.34.

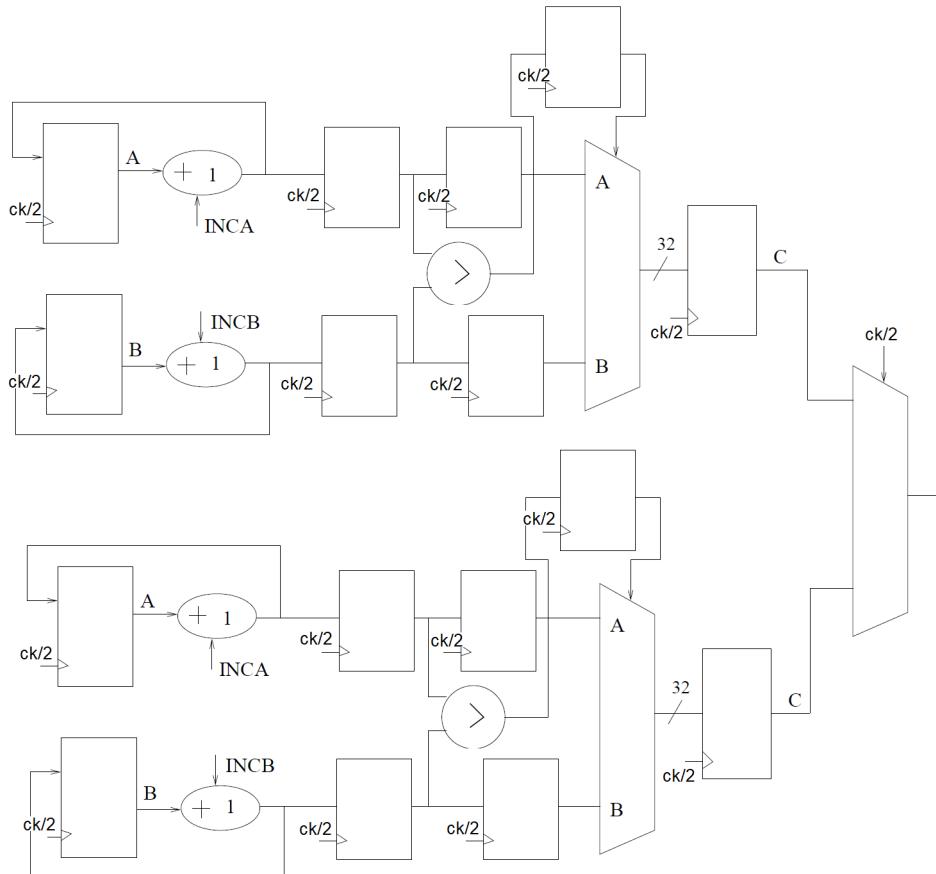


Figure 3.34: Configurazione datapath pipelining

3.3. Pipelining and parallelizing

I risultati di questo nuovo approccio, che vengono calcolati mediante i medesimi passaggi fatti in precedenza sono riportati nella Tabella 3.8.

Parallel and Pipelining Solution	
u	0.3525
Delay (critical path)	284 ns
Allowed Clock Frequency	3.5211 MHz
Power	$2.2\mu W$
Area	$7120\mu m^2$

Table 3.8: *Risultati datapath "parallelizzato e pipelinato"*

Tuttavia si può notare come questa soluzione non sia ottimale, poiché si ottiene un risparmio di potenza all'incirca pari al caso "parallelizzato", ma impiegando un'area due volte superiore.

Infine si è analizzato il caso in cui si parallelizzi il sistema a tre livelli, andando a lavorare su una frequenza pari a $f_{ck}/3$: si scala di conseguenza il percorso critico, portandolo a 426 ns e si può ridurre ulteriormente la tensione di alimentazione V_{DD} . Anche in questo caso si riportano esclusivamente i risultati dei calcoli in Tabella 3.9.

Parallel and Pipelining Solution	
u	0.3333
Delay (critical path)	426 ns
Allowed Clock Frequency	2.347 MHz
Power	$1.76596\mu W$
Area	$5358\mu m^2$

Table 3.9: *Risultati datapath "parallelizzazione a tre livelli"*

Si può concludere affermando che la soluzione che porta al minor consumo di potenza sia la parallelizzazione a tre livelli, che comporta una riduzione della potenza pari a circa 8 volte la potenza del caso "originale". L'unico svantaggio di questa soluzione è che comporta un'area tre volte superiore. Se si vuole cercare un trade-off tra area e potenza, sicuramente la soluzione migliore è la parallelizzazione a due livelli, che comporta un risparmio di potenza di circa 6 volte rispetto al caso originario, utilizzando "solo" un'area due volte superiore.

3.3. Pipelining and parallelizing

3.3.1 Are you sure it was correct?

Nelle Figure 3.35 e 3.36, sono riportati i timing diagram rispettivamente del circuito originale e del caso "parallelizzato".

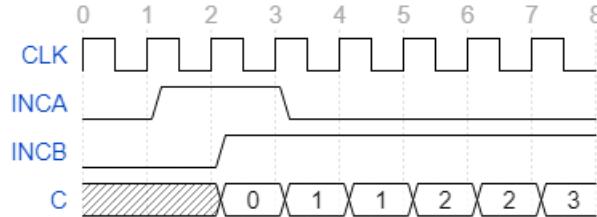


Figure 3.35: *Timing datapath "originale"*

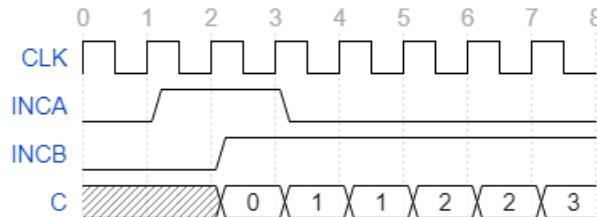


Figure 3.36: *Timing datapath "parallelizzato"*

Si può ben notare come l'uscita del datapath nel secondo caso non sia corretta. Questo errore è dovuto al fatto che nel circuito siano presenti dei loop: questo implica che l'elaborazione del dato $i + 1 - esima$ può dipendere dall'elaborazione $i - esima$ e dunque non può essere effettuata su un altro circuito, come nel caso della parallelizzazione.

4. Laboratorio 4: Bus Encoding

Durante questa esperienza di laboratorio, viene chiesto di analizzare e di valutare alcune tecniche di bus encoding. Nella prima parte dell'esperienza viene chiesto di valutare le performance delle varie tecniche in termini di transizioni e poi ——

4.1 Simulation

Durante la prima parte dell'esperienza viene chiesto di valutare, in termini di commutazioni, diverse tecniche di bus-encoding. Il tutto viene simulato grazie ad un testbench fornito e tramite i *power report* generati da *ModelSim*.

4.1.1 Non-encoded

Inizialmente si è simulato il caso in cui utilizzi un bus non codificato, andando a vedere l'evoluzione dei toogle in 10000 colpi di clock in modo tale da avere un punto di riferimento nel confronto con le altre varie tecniche. All'interno del testbench da simulare sono distinti due processi:

- processo per gli indirizzi
- processo per i dati

i quali prendono come input i dati contenuti nel file *rndin.txt*, che, come sottolineato nella traccia, contiene delle stringhe ad 8 bit con una probabilità bassa di 1 logico. Questo fa già prevedere che la tecnica *Transition-Based*

4.1. Simulation

risulterà la meno performante, in quanto questa tipologia di codifica porta ad riduzione delle commutazioni, solo nel caso in cui le probabilità di '1' logico e di '0' logico siano equiprobabili. risulterà meno efficiente. Nelle Figure 4.2 e 4.1, sono riportati i *power report* rispettivamente nel caso di dati e di indirizzi. Infine, nelle Tabelle 4.1 e 4.2 si riportano le switching activity rispettivamente nel caso di indirizzi e dati.

Power Report	Node	Tc	Ti	Time At 1	Time At 0	Time At X
/testbench/ABUSNORM(7)	4974	0	50030000	49970000	0	
/testbench/ABUSNORM(6)	5022	0	49460000	50540000	0	
/testbench/ABUSNORM(5)	4987	0	49384900	50615100	0	
/testbench/ABUSNORM(4)	4972	0	49970000	50030000	0	
/testbench/ABUSNORM(3)	4960	0	51250000	48750000	0	
/testbench/ABUSNORM(2)	4965	0	49894900	50105100	0	
/testbench/ABUSNORM(1)	5024	0	50000000	50000000	0	
/testbench/ABUSNORM(0)	5060	0	49940000	50060000	0	
/testbench/COUNTBUSNORM(7)	4974	0	50030000	49969900	100	
/testbench/COUNTBUSNORM(6)	5021	0	49455000	50544900	100	
/testbench/COUNTBUSNORM(5)	4987	0	49375000	50624900	100	
/testbench/COUNTBUSNORM(4)	4972	0	49970000	50029900	100	
/testbench/COUNTBUSNORM(3)	4959	0	51245000	48754900	100	
/testbench/COUNTBUSNORM(2)	4965	0	49885000	50114900	100	
/testbench/COUNTBUSNORM(1)	5023	0	49995000	50004900	100	
/testbench/COUNTBUSNORM(0)	5060	0	49940000	50059900	100	
/testbench/CBUSNORM(7)	4973	0	50025000	49974900	100	
/testbench/CBUSNORM(6)	5021	0	49445000	50554900	100	
/testbench/CBUSNORM(5)	4987	0	49365000	50634900	100	
/testbench/CBUSNORM(4)	4972	0	49970000	50029900	100	
/testbench/CBUSNORM(3)	4958	0	51240000	48759900	100	
/testbench/CBUSNORM(2)	4964	0	49880000	50119900	100	
/testbench/CBUSNORM(1)	5023	0	49985000	50014900	100	
/testbench/CBUSNORM(0)	5060	0	49940000	50059900	100	
/testbench/CK	20000	0	50000000	50000000	0	
/testbench/RST	2	0	200	99999800	0	
/testbench/A(7)	4974	0	50030000	49970000	0	
/testbench/A(6)	5022	0	49460000	50540000	0	
/testbench/A(5)	4987	0	49384900	50615100	0	
/testbench/A(4)	4972	0	49970000	50030000	0	
/testbench/A(3)	4960	0	51250000	48750000	0	
/testbench/A(2)	4965	0	49894900	50105100	0	
/testbench/A(1)	5024	0	50000000	50000000	0	
/testbench/A(0)	5060	0	49940000	50060000	0	

Figure 4.1: *Power Report*, tecnica no-encoding, caso 'data'

Nodo	E_{SW}
countbusnorm(7)	0.0097
countbusnorm(6)	0.0118
countbusnorm(5)	0.0312
countbusnorm(4)	0.0624
countbusnorm(3)	0.1249
countbusnorm(2)	0.2499
countbusnorm(1)	0.4999
countbusnorm(0)	0.9999

Table 4.1: *Switching Activity*, tecnica no-encoding, caso 'data'

4.1. Simulation

Power Report	Node	Tc	Ti	Time At 1	Time At 0	Time At X
	/testbench/ABUSNORM(7)	97	0	48164900	51835100	0
	/testbench/ABUSNORM(6)	118	0	37760000	62240000	0
	/testbench/ABUSNORM(5)	312	0	49927000	50073000	0
	/testbench/ABUSNORM(4)	625	0	49931900	50068100	0
	/testbench/ABUSNORM(3)	1250	0	50007000	49993000	0
	/testbench/ABUSNORM(2)	2500	0	50007000	49993000	0
	/testbench/ABUSNORM(1)	5000	0	50007000	49993000	0
	/testbench/ABUSNORM(0)	10000	0	50007000	49993000	0
	/testbench/COUNTBUSNORM(7)	97	0	48155000	51844900	100
	/testbench/COUNTBUSNORM(6)	118	0	37760000	62239900	100
	/testbench/COUNTBUSNORM(5)	312	0	49920000	50079900	100
	/testbench/COUNTBUSNORM(4)	624	0	49920000	50079900	100
	/testbench/COUNTBUSNORM(3)	1249	0	49995000	50004900	100
	/testbench/COUNTBUSNORM(2)	2499	0	49995000	50004900	100
	/testbench/COUNTBUSNORM(1)	4999	0	49995000	50004900	100
	/testbench/COUNTBUSNORM(0)	9999	0	49995000	50004900	100
	/testbench/CBUSNORM(7)	97	0	48145000	51854900	100
	/testbench/CBUSNORM(6)	118	0	37760000	62239900	100
	/testbench/CBUSNORM(5)	312	0	49920000	50079900	100
	/testbench/CBUSNORM(4)	624	0	49920000	50079900	100
	/testbench/CBUSNORM(3)	1249	0	49985000	50014900	100
	/testbench/CBUSNORM(2)	2499	0	49985000	50014900	100
	/testbench/CBUSNORM(1)	4999	0	49985000	50014900	100
	/testbench/CBUSNORM(0)	9998	0	49990000	50009900	100
	/testbench/CK	20000	0	50000000	50000000	0
	/testbench/RST	2	0	200	99999800	0
	/testbench/A(7)	97	0	48164900	51835100	0
	/testbench/A(6)	118	0	37760000	62240000	0
	/testbench/A(5)	312	0	49927000	50073000	0
	/testbench/A(4)	625	0	49931900	50068100	0
	/testbench/A(3)	1250	0	50007000	49993000	0
	/testbench/A(2)	2500	0	50007000	49993000	0
	/testbench/A(1)	5000	0	50007000	49993000	0
	/testbench/A(0)	10000	0	50007000	49993000	0

Figure 4.2: *Power Report, tecnica no-encoding, caso 'address'*

Nodo	E_{SW}
countbusnorm(7)	0.4974
countbusnorm(6)	0.5021
countbusnorm(5)	0.4987
countbusnorm(4)	0.4972
countbusnorm(3)	0.4959
countbusnorm(2)	0.4965
countbusnorm(1)	0.5023
countbusnorm(0)	0.506

Table 4.2: *Switching Activity, tecnica no-encoding, caso 'address'*

4.1.2 Bus-invert technique, Transition based technique, Gray technique

La codifica del Bus-invert consiste nel verificare che il dato successivo comporti un numero di commutazioni superiori di $\frac{N}{2}$, dove N è il numero di

4.1. Simulation

linee che compone il bus. In caso, conviene mandare il dato complementato e forzare ad '1' un bit denominato *inv*, per segnalare che il dato in arrivo è in realtà complementato. Nelle Tabelle 4.3 e 4.4, si riportano le Switching Activity rispettivamente nel caso di indirizzi e dati.

Nodo	E_{SW}
countbusinv(8)	0.0624
countbusinv(7)	0.0527
countbusinv(6)	0.0506
countbusinv(5)	0.0312
countbusinv(4)	0
countbusinv(3)	0.0625
countbusinv(2)	0.1875
countbusinv(1)	0.4375
countbusinv(0)	0.9375

Table 4.3: *Switching Activity, tecnica bus-invert, caso 'address'*

Nodo	E_{SW}
countbusinv(8)	0.365
countbusinv(7)	0.3576
countbusinv(6)	0.3611
countbusinv(5)	0.3617
countbusinv(4)	0.364
countbusinv(3)	0.3613
countbusinv(2)	0.3601
countbusinv(1)	0.3639
countbusinv(0)	0.3722

Table 4.4: *Switching Activity, tecnica bus-invert, caso 'data'*

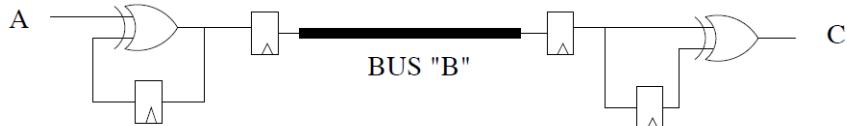


Figure 4.3: *Schema tecnica 'Transition Based'*

Si esegue la medesima analisi per la tecnica *Transition Based*, riportata in Figura 4.3. Questa tecnica consiste nell'inviare uno '0' logico ogni volta la linea corrisponde alla precedente, mentre un '1' logico viene inteso come

4.1. Simulation

complemento del valore precedente della linea. I risultati delle simulazioni sono riportati per il caso dati in Tabella 4.5 e per il caso indirizzi in Tabella 4.6.

Nodo	E_{SW}
countbustran(7)	0.5003
countbustran(6)	0.4946
countbustran(5)	0.4938
countbustran(4)	0.4997
countbustran(3)	0.5125
countbustran(2)	0.4989
countbustran(1)	0.500
countbustran(0)	0.4994

Table 4.5: *Switching Activity, tecnica Transition-Based, caso 'data'*

Nodo	E_{SW}
countbustran(7)	0.4816
countbustran(6)	0.3776
countbustran(5)	0.4992
countbustran(4)	0.4992
countbustran(3)	0.500
countbustran(2)	0.500
countbustran(1)	0.500
countbustran(0)	0.500

Table 4.6: *Switching Activity, tecnica Transition-Based, caso 'address'*

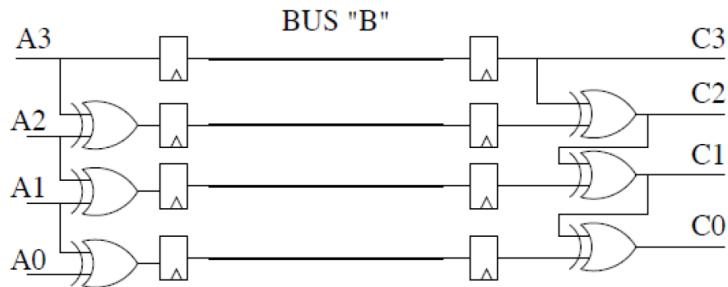


Figure 4.4: *Schemma tecnica 'Codifica di Gray'*

4.1. Simulation

La tecnica *Codifica di Gray*, riportata in Figura 4.4: questa tecnica viene sfruttata specialmente nelle trasmissioni sequenziali, come ad esempio nel caso di indirizzi, in modo tale cambio esclusivamente un solo bit alla volta. I risultati delle Switching Activity sono riportati per i dati nella Tabella 4.7 e nella Tabella 4.8 per gli indirizzi.

Nodo	E_{SW}
countbusgray(7)	0.4974
countbusgray(6)	0.5087
countbusgray(5)	0.4952
countbusgray(4)	0.4981
countbusgray(3)	0.4937
countbusgray(2)	0.5056
countbusgray(1)	0.4962
countbusgray(0)	0.5075

Table 4.7: *Switching Activity, tecnica Codifica di Gray, caso 'data'*

Nodo	E_{SW}
countbusgray(7)	0.097
countbusgray(6)	0.055
countbusgray(5)	0.194
countbusgray(4)	0.312
countbusgray(3)	0.625
countbusgray(2)	0.125
countbusgray(1)	0.250
countbusgray(0)	0.500

Table 4.8: *Switching Activity, tecnica Transition Based, caso 'address'*

4.1. Simulation

Si riportano per completezza i *Power Report* delle simulazioni svolte con *ModelSim*. In Figura 4.5 si riporta il Report nel caso di indirizzi, mentre in Figura 4.6 il caso di dati.

Power Report	Node	Tc	Ti	Time At 1	Time At 0	Time At X
/testbench/A(7)	97	0	48164900	51835100	0	
/testbench/A(6)	118	0	37760000	62240000	0	
/testbench/A(5)	312	0	49927000	50073000	0	
/testbench/A(4)	625	0	49931900	50068100	0	
/testbench/A(3)	1250	0	50007000	49993000	0	
/testbench/A(2)	2500	0	50007000	49993000	0	
/testbench/A(1)	5000	0	50007000	49993000	0	
/testbench/A(0)	10000	0	50007000	49993000	0	
/testbench/COUNTBUSNORM(7)	97	0	48155000	51844900	100	
/testbench/COUNTBUSNORM(6)	118	0	37760000	62239900	100	
/testbench/COUNTBUSNORM(5)	312	0	49920000	50079900	100	
/testbench/COUNTBUSNORM(4)	624	0	49920000	50079900	100	
/testbench/COUNTBUSNORM(3)	1249	0	49995000	50004900	100	
/testbench/COUNTBUSNORM(2)	2499	0	49995000	50004900	100	
/testbench/COUNTBUSNORM(1)	4999	0	49995000	50004900	100	
/testbench/COUNTBUSNORM(0)	9999	0	49995000	50004900	100	
/testbench/COUNTBUSTRAN(7)	4816	0	24080000	75919900	100	
/testbench/COUNTBUSTRAN(6)	3776	0	18880000	81119900	100	
/testbench/COUNTBUSTRAN(5)	4992	0	24960000	75039900	100	
/testbench/COUNTBUSTRAN(4)	4992	0	24960000	75039900	100	
/testbench/COUNTBUSTRAN(3)	5000	0	25000000	74999900	100	
/testbench/COUNTBUSTRAN(2)	5000	0	25000000	74999900	100	
/testbench/COUNTBUSTRAN(1)	5000	0	25000000	74999900	100	
/testbench/COUNTBUSTRAN(0)	5000	0	50000000	49999900	100	
/testbench/COUNTBUSGRAY(7)	97	0	48155000	51844900	100	
/testbench/COUNTBUSGRAY(6)	55	0	24475000	75524900	100	
/testbench/COUNTBUSGRAY(5)	194	0	49920000	50079900	100	
/testbench/COUNTBUSGRAY(4)	312	0	49920000	50079900	100	
/testbench/COUNTBUSGRAY(3)	625	0	49995000	50004900	100	
/testbench/COUNTBUSGRAY(2)	1250	0	50000000	49999900	100	
/testbench/COUNTBUSGRAY(1)	2500	0	50000000	49999900	100	
/testbench/COUNTBUSGRAY(0)	5000	0	50000000	49999900	100	
/testbench/COUNTBUSINV(8)	624	0	49920000	50079900	100	
/testbench/COUNTBUSINV(7)	527	0	50075000	49924900	100	
/testbench/COUNTBUSINV(6)	506	0	49920000	50079900	100	
/testbench/COUNTBUSINV(5)	312	0	49920000	50079900	100	
/testbench/COUNTBUSINV(3)	625	0	49995000	50004900	100	
/testbench/COUNTBUSINV(2)	1875	0	49995000	50004900	100	
/testbench/COUNTBUSINV(1)	4375	0	49995000	50004900	100	
/testbench/COUNTBUSINV(0)	9375	0	49995000	50004900	100	

Figure 4.5: *Power Report complessivo, caso 'address'*

4.1. Simulation

Power Report Interval 100000000						
Power Report	Node	Tc	Ti	Time At 1	Time At 0	Time At X
/testbench/A(7)	4974	0	50030000	49970000	0	
/testbench/A(6)	5022	0	49460000	50540000	0	
/testbench/A(5)	4987	0	49384900	50615100	0	
/testbench/A(4)	4972	0	49970000	50030000	0	
/testbench/A(3)	4960	0	51250000	48750000	0	
/testbench/A(2)	4965	0	49894900	50105100	0	
/testbench/A(1)	5024	0	50000000	50000000	0	
/testbench/A(0)	5060	0	49940000	50060000	0	
/testbench/COUNTBUSNORM(7)	4974	0	50030000	49969900	100	
/testbench/COUNTBUSNORM(6)	5021	0	49450000	50544900	100	
/testbench/COUNTBUSNORM(5)	4987	0	49375000	50624900	100	
/testbench/COUNTBUSNORM(4)	4972	0	49970000	50029900	100	
/testbench/COUNTBUSNORM(3)	4959	0	51245000	48754900	100	
/testbench/COUNTBUSNORM(2)	4965	0	49885000	50114900	100	
/testbench/COUNTBUSNORM(1)	5023	0	49995000	50004900	100	
/testbench/COUNTBUSNORM(0)	5060	0	49940000	50059900	100	
/testbench/COUNTBUSTRAN(7)	5003	0	49825000	50174900	100	
/testbench/COUNTBUSTRAN(6)	4946	0	49830000	50169900	100	
/testbench/COUNTBUSTRAN(5)	4938	0	50480000	49519900	100	
/testbench/COUNTBUSTRAN(4)	4997	0	49585000	50414900	100	
/testbench/COUNTBUSTRAN(3)	5125	0	49945000	50054900	100	
/testbench/COUNTBUSTRAN(2)	4989	0	50245000	49754900	100	
/testbench/COUNTBUSTRAN(1)	5000	0	49890000	50109900	100	
/testbench/COUNTBUSTRAN(0)	4994	0	49540000	50459900	100	
/testbench/COUNTBUSGRAY(7)	4974	0	50030000	49969900	100	
/testbench/COUNTBUSGRAY(6)	5087	0	50105000	49894900	100	
/testbench/COUNTBUSGRAY(5)	4952	0	50100000	49899900	100	
/testbench/COUNTBUSGRAY(4)	4981	0	50505000	49494900	100	
/testbench/COUNTBUSGRAY(3)	4937	0	49415000	50584900	100	
/testbench/COUNTBUSGRAY(2)	5056	0	50320000	49679900	100	
/testbench/COUNTBUSGRAY(1)	4962	0	50710000	49289900	100	
/testbench/COUNTBUSGRAY(0)	5075	0	49715000	50284900	100	
/testbench/COUNTBUSINV(8)	3650	0	49260000	50739900	100	
/testbench/COUNTBUSINV(7)	3576	0	48810000	51189900	100	
/testbench/COUNTBUSINV(6)	3611	0	49595000	50404900	100	
/testbench/COUNTBUSINV(5)	3617	0	52235000	47764900	100	
/testbench/COUNTBUSINV(4)	3640	0	50190000	49809900	100	
/testbench/COUNTBUSINV(3)	3613	0	49685000	50314900	100	
/testbench/COUNTBUSINV(2)	3601	0	50405000	49594900	100	
/testbench/COUNTBUSINV(1)	3639	0	50295000	49704900	100	
/testbench/COUNTBUSINV(0)	3722	0	51500000	48499900	100	

Figure 4.6: *Power Report complessivo, caso 'data'*

4.1.3 T0 technique

L’ultima tecnica richiesta è di implementare un codice per realizzare la tecnica *T0 Encoding*. Nella traccia viene riportata la formula richiesta da implementare, riportata anche in Figura 4.7.

$$\left(B^{(t)}, INC^{(t)} \right) = \begin{cases} \left(B^{(t-1),1} \right) & if \quad b^{(t)} = b^{(t-1)} + 1 \\ \left(b^{(t)}, 0 \right) & otherwise \end{cases}$$

Figure 4.7: *Formula implementata, tecnica 'T0'*

4.2. Synthesis

La codifica T0 consiste nell'inserire un bit in più, denominato *INC*. Ogni volta che si va in sequenza, si forza questo segnale a '0' in modo tale che il ricevitore capisca di dover andare semplicemente in sequenza e si preoccupi lui di incrementare il dato. In caso di brach, salito o subroutine, forzo il segnale ad '1' e trasmetto sul bus il nuovo dato. Si riporta in seguito il *Codice VHDL* dell'implementazione della codifica.

4.1.4 Confronto tra le tecniche

Confrontando i risultati tra le varie tecniche, si può notare come l'unica tecnica che porta ad una riduzione dei consumi nel caso dei dati, risulta essere la **Bus-Invert** con la quale si ottiene un risparmio sulle commutazioni pari a circa il 20%.

Per quanto concerne la trasmissione di indirizzi, le codifiche più performanti risultano la **Codifica di Gray**, che porta ad un risparmio sulle commutazioni del 50%, e la **Codifica T0**, che porta ad un risparmio del XX%.

Si può notare come la tecnica che risulta meno performante è la **Transition Based**, che porta ad un incremento del 93% nel caso di dati e ad un incremento circa del 0,1% nel caso di indirizzi. Ovviamente questo è concorde con quanto ci si aspettava, in quanto, come già detto prima, questa tipologia di decodifica funziona efficientemente solo nel caso in cui le probabilità di '1' e di '0' logico non risultino sbilanciate.

4.2 Synthesis

Una volta aver stimato il consumo di potenza per la tecnica di bus non-encoded si può adesso calcolare i consumi per le quattro diverse tecniche di bus encoding, in presenza di due ingressi con diversa probabilità statistica: Address e Data. Per rendere più comoda e veloce la procedura sono stati utilizzati degli script, organizzati in modo da: sintetizzare le architetture e salvarne il design (i file SDF e la netlist) tramite Synopsys; leggere i file SDF e simularli salvando i risultati in file VCD, poi convertiti in file SAIF per la backannotation, tramite il programma Modelsim. calcolare a partire

4.2. Synthesis

dai file SAIF la sintesi del design e il consumo di potenza, tramite Synopsys; Inoltre sono presenti dei file di configurazione contenenti variabili e opzioni per impostare lo script in base al tipo di simulazione richiesta. Per entrambi i bus sono stati calcolati i consumi in base a diversi carichi capacitivi alle linee di bus.

uW	1fF	10fF	50fF	100fF
normal	10.158	11.387	16.220	23.236
invert	26.500	28.121	32.539	38.893
transbased	27.684	30.211	39.619	53.373
gray	9.440	10.059	12.496	16.012
t0	0.138	0.154	0. 220	0.316

Table 4.9: *potenza dinamica dissipata nella trasmissione di indirizzi per le diverse codifiche in base al carico della linea.*

Nel caso degli ADDRESS si può notare come la tecnica del T0 permette di ridurre i consumi di potenza dinamica di circa 10 volte, mentre la transbased e la invert sono le peggiori.

uW	1fF	10fF	50fF	100fF
normal	14.561	17.029	26.736	40.870
invert	36.643	38.460	46.385	57.812
transbased	26.628	29.114	38.929	53.082
gray	21.370	23.840	33.564	47.719
t0	17.298	20.106	29.503	44.776

Table 4.10: *potenza dinamica dissipata nella trasmissione di dati per le diverse codifiche in base al carico della linea.*

Per quanto riguarda la trasmissione dei DATA invece, la codifica migliore dal punto di vista dei consumi si rivela essere la normal. Chiaramente per entrambi gli ingressi e per qualsiasi codifica, è sempre valido il fatto che all'aumentare della capacità effettiva del bus i consumi di potenza dinamica aumentano, come dimostrano i risultati ottenuti.

5. Laboratorio 5: Leakage: using spice for characterizing cells and pen&paper for memory organization

In questo laboratorio viene richiesto di analizzare i contributi di potenza, prestando particolare attenzione particolarmente alla potenza di leakage.

5.1 Characterizing a library gate

In questa prima parte dell'esercitazione viene richiesto di analizzare le performance di una *NAND* a due ingressi, ottimizzata per essere *High-Speed* per avere esattamente un punto di riferimento quando si analizzerà il caso con basso leakage. La porta NAND in questione si compone di due pMOS in parallelo, collegati alla Vdd e due nMOS in serie collegati al GND, come si può osservare in Figura 5.1.

La definizione del pMOS e dell'nMOS avviene tramite una piccola libreria '*CMOS2013*', all'interno della quale sono definiti i modelli che verranno analizzati (nel nostro caso si tratta del **EPHSGP_BS2JU** per il pMOS e **ENHSGP_BS2JU** per l'nMOS). All'interno di questa libreria ogni transistore viene identificato tramite alcuni parametri, quali la lunghezza e la larghezza del MOS e perimetri e aree di draine source. L'analisi avviene tramite uno script '*nandHS.sp*' che contiene una netlist dove vengono fissati i valori dei parametri di larghezza e lunghezza dei transitori.

5.1. Characterizing a library gate

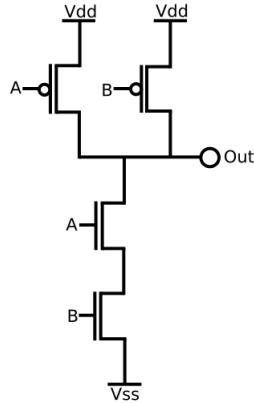


Figure 5.1: Schema circuitale porta NAND

Inoltre lo script contiene il alcuni comandi .measure tramite il quale vengono stimati i tempi di salita, di discesa e di propagazione della porta in questione. Viene richiesto di completare questi comandi andando a scrivere un comando per la misura tel tempo di propagazione basso-alto. Il comando aggiunto in questione è il seguente:

```
.measure tran nanddelayHL TRIG V(inB) VAL='alim*0.5' RISE=1 +
TARG V(out) VAL='alim*0.5' FALL=1
```

Settando l'ambiente di simulazione **ELDO**, si può procedere con la simulazione della NAND in questione andando a riportare i parametri richiesti dalla traccia. Tra tutti i parametri viene richiesto ai annatore la potenza totale dissipata, che risulta essere pari a 6.7908 nW. I risultati temporali invece sono riportati in Tabella 5.1.

Tempo	Misura
t_{rise}	89.303 ps
t_{fall}	74.319 ps
t_{pdHL}	48.993 ps
t_{pdLH}	56.490 ps

Table 5.1: Risultati tempi NAND simulazione ELDO

Si possono confrontare questi valori andando ad avviare una simulazione

5.1. Characterizing a library gate

grafica tramite **ezwave**. Le onde risultati sono riportate in Figura 5.2, dove sono riportati i valori della due tensioni di ingresso che, come visto dalla netlist, risultano essere:

- INA: costante al valore 1.2 V, assimilato come '1' logico
- INB: 0 V fino ad 1 ns, 1.2 V fino a 2 ns, 0 V fino al termine della simulazione

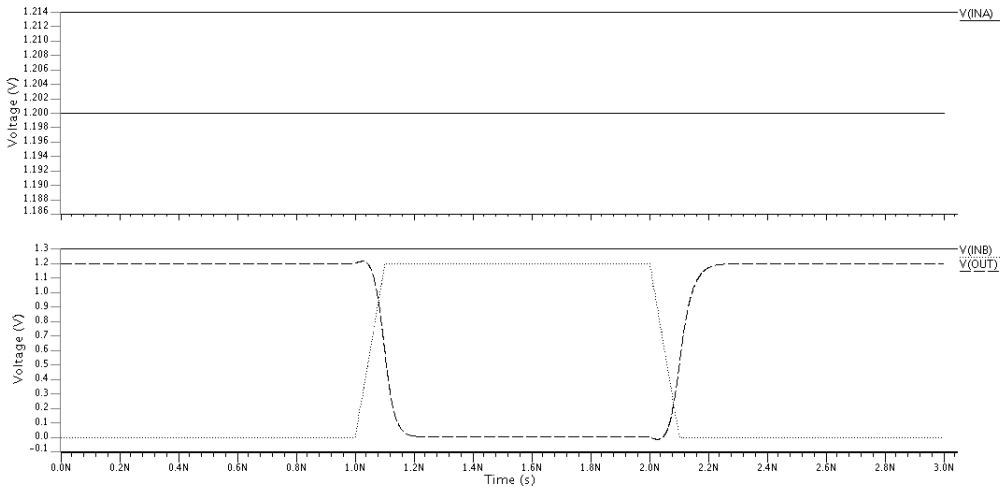


Figure 5.2: Schema circuitale porta NAND

Tramite gli appositi cursori del programma sono stati ricavati i tempi calcolati in precedenza con ELD0. Tutti i tempo sono riportati i Tabella 5.2.

Tempo	Misura
t_{rise}	84.64 ps
t_{fall}	74.01 ps
t_{pdHL}	47.24 ps
t_{pdLH}	56.71 ps

Table 5.2: Risultati tempi NAND simulazione ezwave

Si può ben notare come i risultati siano assolutamente confrontabili.

In seguito viene chiesto di fare un analisi i continua, andando a decommentare alcune linee dello script, per andare a valutare le tensioni di soglia dei

diversi MOS. I risultati sono riportati in Tabella 5.3. .

Transistore	Tensione di Soglia V_{TH}
VT(XNAND.XMN0.M1)	0.31371 V
VT(XNAND.XMN1.M1)	0.27241 V
VT(XNAND.XMP0.M1)	-0.24712 V
VT(XNAND.XMP1.M1)	-0.24712 V

Table 5.3: *Tensioni di soglia*

Ovviamente, come ci si aspettava, le tensioni di soglia dei pMOS sono negative, mentre quelle degli nMOS sono positive. Le due tensioni di soglia dei pMOS risultano identiche; non vale lo stesso nel caso degli nMOS, in quanto si ha che la tensione di soglia del MOS0 è superiore alla tensione di soglia del MOS1. Questo genera di conseguenza una corrente di leakage più alta nel caso del MOS1 rispetto al MOS0.

5.2 Characterizing a gate for output load

L’obiettivo di questa seconda sezione dell’esercitazione è di analizzare il funzionamento della porta NAND andando a variare il carico in uscita. Si analizzano i casi con carico pari a:

- 0.005 fF
- 0.05 fF
- 0.5 fF
- 5 fF
- 50 fF

A tale scopo si analizza, sempre tramite l’ambiente di simulazione *ELDO*, il file *nandHScharLoad.sp*. Rispetto al file precedente viene inserita una misura per valutare i picchi di corrente che scorrono nel GND e nella capacità di carico. I risultati della simulazione sono contenuti nella Tabella 5.4 per i tempi e nella Tabella 5.5 per i risultati legati alla corrente. . .

5.2. Characterizing a gate for output load

C_{Load}	0.005 fF	0.05 fF	0.5 fF	5 fF	50 fF
t_{rise}	67.226 ps	67.604 ps	71.228 ps	102.81 ps	363.91 ps
t_{fall}	68.551 ps	68.830 ps	72.112 ps	98.194 ps	296.52 ps
t_{pdHL}	20.524 ps	20.851 ps	23.980 ps	48.550 ps	180.16 ps
t_{pdLH}	37.948 ps	38.286 ps	41.521 ps	66.657 ps	209.00 ps

Table 5.4: *Tensioni di soglia*

C_{Load}	0.005 fF	0.05 fF	0.5 fF	5 fF	50 fF
$I_{GND,f}^{max}$	76.530 μ A	77.056 μ A	81.964 μ A	116.64 μ A	240.77 μ A
$I_{Vdd,r}^{max}$	-70.006 μ A	-70.423 μ A	-74.383 μ A	-102.77 μ A	-209.19 μ A
$I_{GND,r}^{max}$	45.879 μ A	45.692 μ A	44.055 μ A	34.693 μ A	12.571 μ A
$I_{Vdd,f}^{max}$	-47.912 μ A	-47.680 μ A	-45.637 μ A	-34.365 μ A	-11.039 μ A
$I_{Load,f}^{max}$	8.2877 nA	8.2877 nA	8.2877 nA	8.2893 nA	372.90 nA
$I_{Load,r}^{max}$	-5.6590 nA	-5.690 nA	-5.6590 nA	-5.6734 nA	-12.547 nA

Table 5.5: *Tensioni di soglia*

Analogamente al punto precedente, si è sfruttato *ezwave* per andare a plottere gli andamenti delle tensioni e delle correnti al variare del carico. I risultati sono riportati in Figura 5.3 per il caso delle correnti e in Figura 5.4 per il caso delle tensioni.

MANCANO I COMMENTI RELATIVI A QUESTI RISULTATI

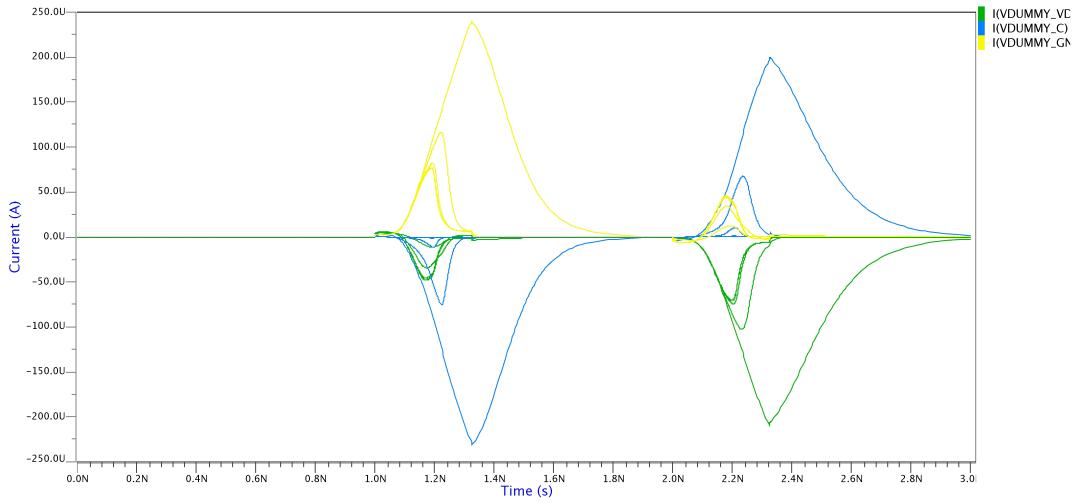


Figure 5.3: *Schema circuitale porta NAND*

5.3. Comparing different gate sizing

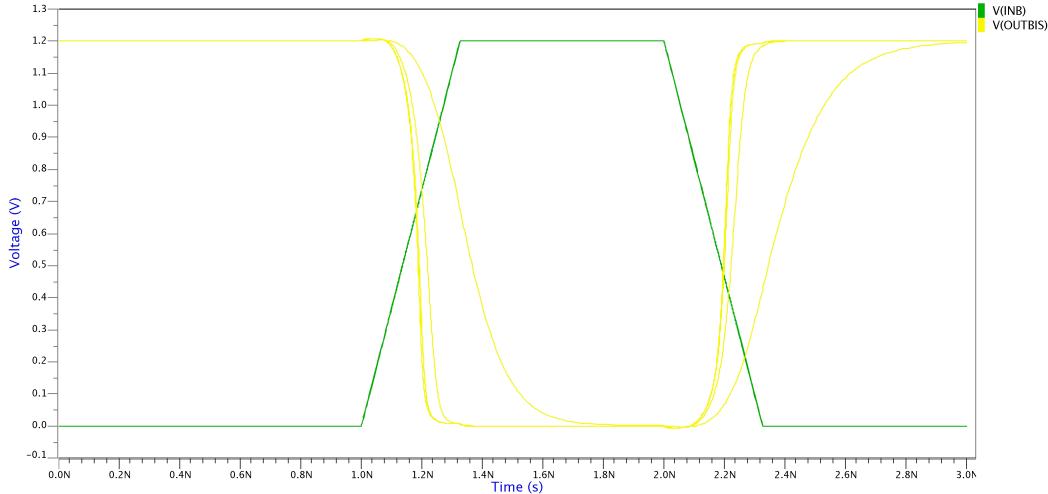


Figure 5.4: Schema circuitale porta NAND

In modo analogo al caso precedente, si sono andate a calcolare le varie Tensioni di Soglia dei MOS. I risultati sono riportati nella Tabella 5.6.

TRANSISTOR/C_{Load}	0.005 fF	0.05 fF	0.5 fF	5 fF	50 fF
XMN0.M1	0.31371	0.31371	0.31371	0.31371	0.31371
XMN1.M1	0.27241	0.27241	0.27241	0.27241	0.27241
XMP0.M1	-0.24712	-0.24712	-0.24712	-0.24712	-0.24712
XMP1.M1	-0.24712	-0.24712	-0.24712	-0.24712	-0.24712

Table 5.6: Tensioni di soglia

Esattamente come ci si aspettava, i valori delle tensioni di soglia sono identici rispetto al caso ottenuto nel paragrafo 5.1. Questo è dovuto al fatto che la V_{TH} dipende esclusivamente dai parametri tecnologici con la quale viene realizzato il transistore e non ha alcuna dipendenza dal carico in uscita.

5.3 Comparing different gate sizing

Le porte NAND analizzate fino ad ora erano ottimizzate per supportare carichi capacitivi che non superassero i 0.16 fF. Per avere dei gate ottimizzati per

5.3. Comparing different gate sizing

pilotare carichi superiori, bisogna utilizzare porte logiche differenti, comunque presenti nella libreria.

Viene chiesto di analizzare ora due porte NAND, di dimensioni differenti: una prima porta X1 ed una seconda porta, più grande della prima, denominata X8. Queste porte sono ottimizzate per pilotare carichi capacitivi fino ad un massimo di 1.28 fF.

Le due netlist contenenti le due diverse porte NAND sono: *nandHScharMax_Load.sp* e *nandHSX8charMax_Load.sp*.

Tutte le simulazioni vengono ora fatte utilizzando due carichi capacitivi diversi: 0.06 fF e 60.0 fF. I risultati nel caso della NAND X1 sono riportati nelle Tabelle 5.7 per i tempi e 5.8 per le correnti; mentre i risultati nel caso della NAND X8 sono riportati nelle Tabelle 5.9 per i tempi e 5.10 per le correnti.

NAND X1 C_{Load}	0.06 fF	60 fF
t_{rise}	67.692 ps	425.04 ps
t_{fall}	68.978 ps	341.77 ps
t_{pdHL}	20.923 ps	203.71 ps
t_{pdLH}	38.361 ps	236.61 ps

Table 5.7: *Tensioni di soglia*

NAND X8 C_{Load}	0.06 fF	60 fF
$I_{GND,f}^{max}$	77.172 μ A	245.47 μ A
$I_{Vdd,r}^{max}$	-70.514 μ A	-213.72 μ A
$I_{GND,r}^{max}$	45.653 μ A	10.965 μ A
$I_{Vdd,f}^{max}$	-47.631 μ A	-9.4773 μ A
$I_{Load,f}^{max}$	8.2877 nA	1201.6 nA
$I_{Load,r}^{max}$	-5.6590 nA	-19.994 nA

Table 5.8: *Tensioni di soglia*

5.3. Comparing different gate sizing

NAND X8 C_{Load}	0.06 fF	60 fF
t_{rise}	67.692 ps	425.04 ps
t_{fall}	68.978 ps	341.77 ps
t_{pdHL}	20.923 ps	203.71 ps
t_{pdLH}	38.361 ps	236.61 ps

Table 5.9: *Tensioni di soglia*

NAND X8 C_{Load}	0.06 fF	60 fF
$I_{GND,f}^{max}$	637.45 μ A	1069.0 μ A
$I_{Vdd,r}^{max}$	-552.88 μ A	-928.61 μ A
$I_{GND,r}^{max}$	378.92 μ A	264.86 μ A
$I_{Vdd,f}^{max}$	-410.35 μ A	-258.52 μ A
$I_{Load,f}^{max}$	79.078 nA	79.133 nA
$I_{Load,r}^{max}$	-41.224 nA	-41.503 nA

Table 5.10: *Tensioni di soglia*

Allo stesso modo del caso precedente vengono plottati i risultati delle onde con *ezwave*. Nei seguenti grafici vengono confrontate tensioni e correnti tra il caso X1 e il caso X8. Il grafico risultato delle due tensioni di uscita è riportato in Figura 5.5; mentre nelle Figure 5.6, 5.7 e 5.8 sono riportati i grafici risultati rispettivamente per le correnti sul carico, sul GND e sull'alimentazione.

Analizzando i risultati ottenuti, si può notare come i vari tempi siano superiori nel caso della porta X1 rispetto alla porta X8. La porta X8 quindi comporta una maggiore velocità, che però viene pagata con un'area superiore rispetto alla porta X1.

Andando poi ad analizzare il consumo complessivo di potenza risulta che:

$$TotalPowerDissipationX8 = 49.469nW$$

$$TotalPowerDissipationX1 = 6.7908nW$$

Come ci si aspettava, la NAND X1 consuma circa l'86% in meno della NAND X8. Questo è sicuramente dovuto alle correnti di leakage superiori, che si posso anche apprezzare nella Tabella precedente. (FRASE DA CONTROL-LARE).

5.3. Comparing different gate sizing

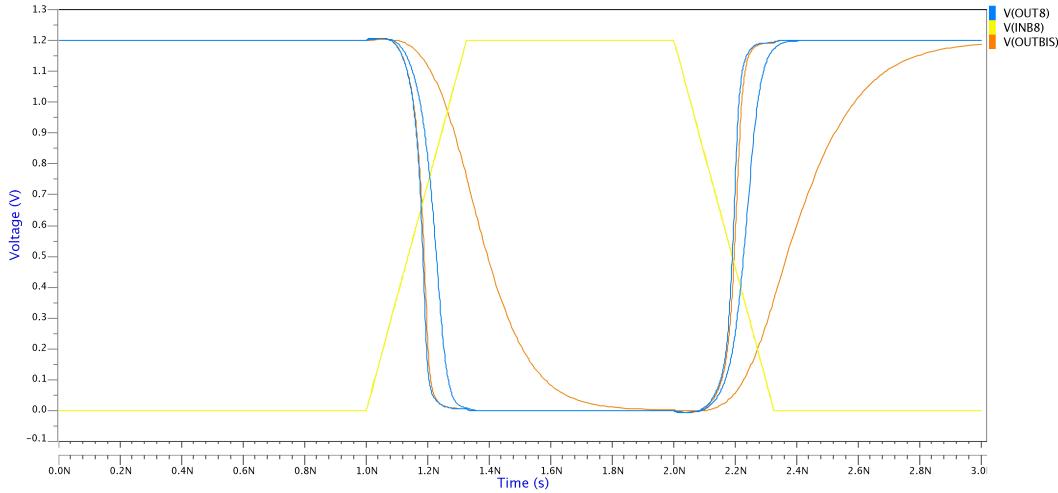


Figure 5.5: Schema circuitale porta NAND

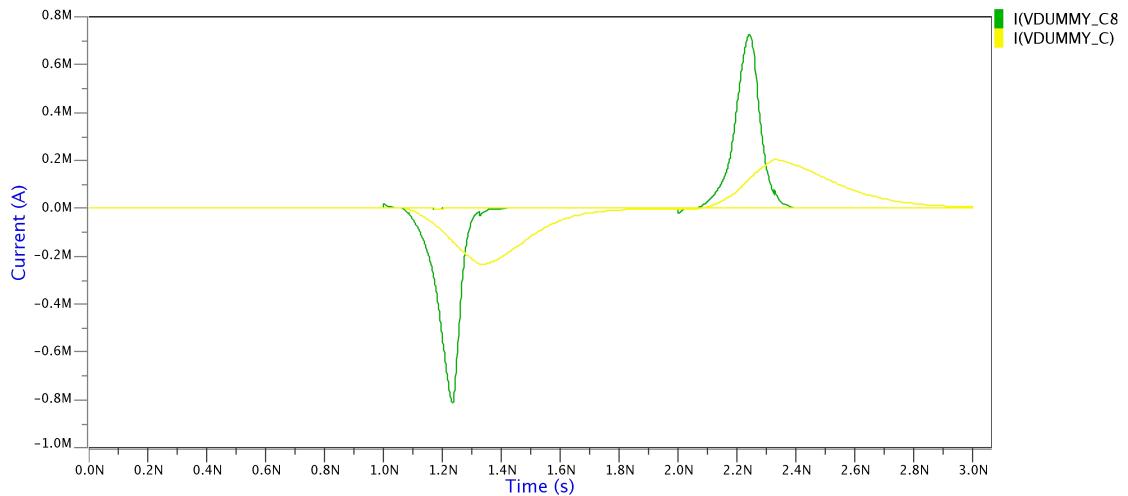


Figure 5.6: Schema circuitale porta NAND

5.3. Comparing different gate sizing

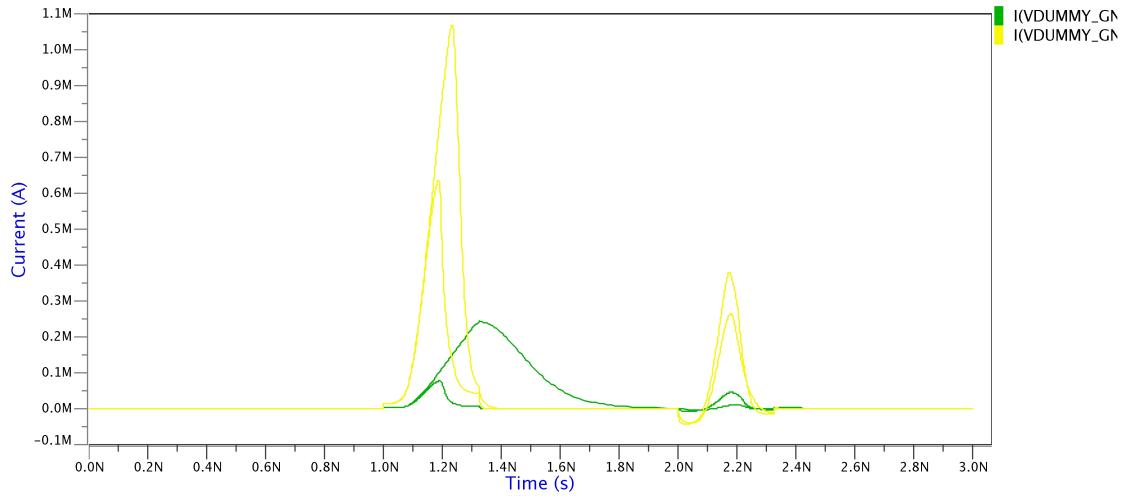


Figure 5.7: Schema circuitale porta NAND

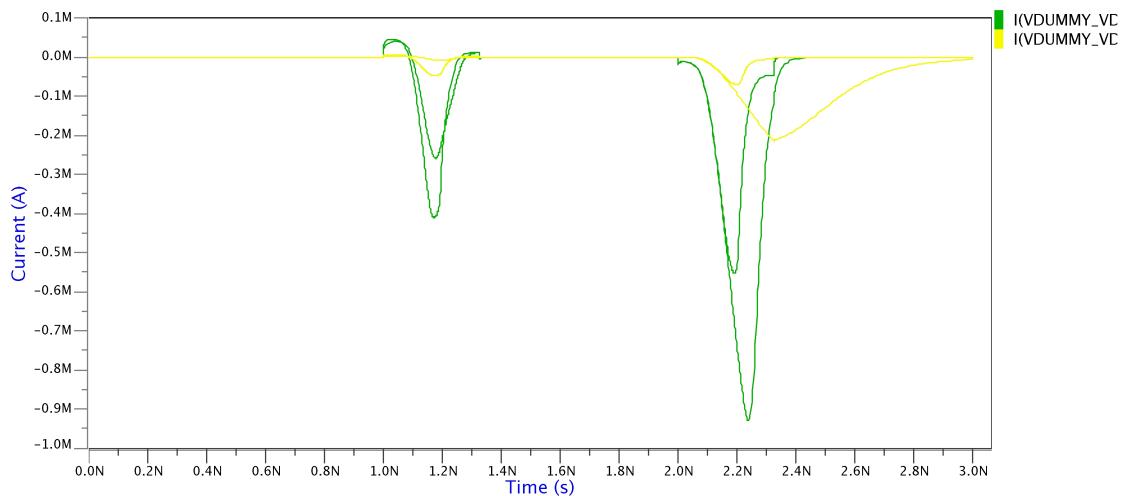


Figure 5.8: Schema circuitale porta NAND

5.4. Comparing high speed and low leakage optimization

Infine si riportano nelle Tabelle 5.11 e 5.12 i risultati delle tensioni di soglia, rispettivamente della NAND X1 e della NAND X8, ottenuti attraverso la simulazione dc.

NAND X1 Transistor/ C_{Load}	0.06 pF	60 pF
XMN0.M1	0.31371	0.31371
XMN1.M1	0.27241	0.27241
XMP0.M1	-0.24712	-0.24712
XMP1.M1	-0.24712	-0.247122

Table 5.11: *Tensioni di soglia*

NAND X8 Transistor/ C_{Load}	0.06 pF	60 pF
XMN0.M1	0.31893	0.31893
XMN1.M1	0.27763	0.27763
XMN2.M1	0.31893	0.31893
XMN3.M1	0.27763	0.27763
XMN4.M1	0.27763	0.27763
XMN5.M1	0.31893	0.31893
XMN6.M1	0.27763	0.27763
XMN7.M1	0.31893	0.31893
XMP0.M1	-0.24712	-0.24712
XMP1.M1	-0.24712	-0.24712
XMP6.M1	-0.24712	-0.24712
XMP7.M1	-0.24712	-0.24712

Table 5.12: *Tensioni di soglia*

5.4 Comparing high speed and low leakage optimization

Viene ora richiesto di fare le medesime analisi svolte nei punti precedenti, ma utilizzando delle tipologie di gate ottimizzati per una bassa corrente sottosoglia. I modelli utilizzati sono precisamente: **ND2LL** e **ND2LLX8**. I dati raccolti dalla simulazione ELD0 sono riportati:

- per quanto riguarda il transistore X1, nella Tabella 5.13 per i tempi, mentre nella Tabella 5.14 per le correnti.

5.4. Comparing high speed and low leakage optimization

- per quanto riguarda il transistore X8, nella Tabella 5.15 per i tempi, mentre nella Tabella 5.16 per le correnti.

NAND LL X1		
C_{Load}	0.06 fF	60 fF
t_{rise}	71.556 ps	592.23 ps
t_{fall}	63.464 ps	406.23 ps
t_{pdHL}	31.612 ps	257.67 ps
t_{pdLH}	52.657 ps	331.22 ps

Table 5.13: *Tensioni di soglia*

NAND LL X1		
C_{Load}	0.06 fF	60 fF
$I_{GND,f}^{max}$	$404.32\mu A$	$867.10\mu A$
$I_{Vdd,r}^{max}$	$-303.01\mu A$	$-683.65\mu A$
$I_{GND,r}^{max}$	$132.79\mu A$	$68.306\mu A$
$I_{Vdd,f}^{max}$	$-165.52\mu A$	$-70.310\mu A$
$I_{Load,f}^{max}$	3.2194nA	3.3727nA
$I_{Load,r}^{max}$	-2.4740nA	-3.2278nA

Table 5.14: *Tensioni di soglia*

NAND LL X8		
C_{Load}	0.06 fF	60 fF
t_{rise}	63.480 ps	137.67 ps
t_{fall}	57.371 ps	112.39 ps
t_{pdHL}	26.768 ps	78.324 ps
t_{pdLH}	44.524 ps	102.77 ps

Table 5.15: *Tensioni di soglia*

NAND LL X8		
C_{Load}	0.06 fF	60 fF
$I_{GND,f}^{max}$	$51.878\mu A$	$194.22\mu A$
$I_{Vdd,r}^{max}$	$-42.315\mu A$	$-151.76\mu A$
$I_{GND,r}^{max}$	$16.961\mu A$	$3.6258\mu A$
$I_{Vdd,f}^{max}$	$-19.758\mu A$	$-2.4801\mu A$
$I_{Load,f}^{max}$	0.39863nA	1923.7nA
$I_{Load,r}^{max}$	-0.37117nA	-36.882nA

Table 5.16: *Tensioni di soglia*

5.4. Comparing high speed and low leakage optimization

Si può ben notare come, tutti i tempi si siano dilatati per le porte *Low Leakage* rispetto a quelle *High Speed*. Si può analizzare l'incremento percentuale del caso LL rispetto al caso HS nella Tabella 5.17 per il gate X1 e nella Tabella 5.18 per il gate X8.

Confronto LL-HS NAND X1			
C_{Load}	0.06 fF	60 fF	
t_{rise}	5.71%	39.33%	
t_{fall}	%	18.86%	
t_{pdHL}	51.1%	26.49%	
t_{pdLH}	37.27%	39.98%	

Table 5.17: *Aumento percentuale dei tempi LL/HS NAND X1*

Confronto LL-HS NAND X8			
C_{Load}	0.06 fF	60 fF	
t_{rise}	16.43%	21.76%	
t_{fall}	%	5.16%	
t_{pdHL}	53.27%	36.36%	
t_{pdLH}	41.99%	42.23%	

Table 5.18: *Aumento percentuale dei tempi LL/HS NAND X8*

Al contrario, si può ben notare come le varie correnti si riducano dal caso LL rispetto al caso HS, arrivando anche a diminuire di un ordine di grandezza. In Tabella 5.19 e in Tabella 5.20 si può notare quanto valga la riduzione percentuale della corrente dal caso HS al caso LL, rispettivamente per il gate X1 e il gate X8.

NAND LL X1			
C_{Load}	0.06 fF	60 fF	
$I_{GND,f}^{max}$	32.78%	20.88%	
$I_{Vdd,r}^{max}$	39.99%	29.32%	
$I_{GND,r}^{max}$	62.85%	66.93%	
$I_{Vdd,f}^{max}$	58.52%	73.83%	
$I_{Load,f}^{max}$	95.19%	?????	
$I_{Load,r}^{max}$	93.44%	????	

Table 5.19: *Riduzione percentuale delle correnti NAND X1*

5.4. Comparing high speed and low leakage optimization

NAND LL X8		
C_{Load}	0.06 fF	60 fF
$I_{GND,f}^{max}$	32.78%	20.88%
$I_{Vdd,r}^{max}$	39.99%	29.32%
$I_{GND,r}^{max}$	62.85%	66.93%
$I_{Vdd,f}^{max}$	58.52%	73.83%
$I_{Load,f}^{max}$	95.19%	?????
$I_{Load,r}^{max}$	93.44%	?????

Table 5.20: Riduzione percentuale delle correnti NAND X8

In seguito, nelle Tabelle 5.21 e 5.22, sono riportati rispettivamente i valori delle Tensioni di Soglia dei gate *Low Leakage* rispettivamente nel caso del gate X1 ed il gate X8. Si può notare come i valori siano superiori rispetto a quelli del caso *High Speed*: questo fenomeno è assolutamente concorde con quanto appreso in teoria, in quanto un transistor a basso leakage deve avere delle tensioni di soglia più elevate.

NAND X1		
Transistor/C_{Load}	0.06 pF	60 pF
XMN0.M1	0.41333	0.41333
XMN1.M1	0.38062	0.38062
XMP0.M1	-0.36712	-0.36712
XMP1.M1	-0.36712	-0.36712

Table 5.21: Tensioni di soglia

NAND X8		
Transistor/C_{Load}	0.06 pF	60 pF
XMN0.M1	0.41333	0.41333
XMN1.M1	0.38062	0.38062
XMN2.M1	0.41333	0.41333
XMN3.M1	0.38062	0.38062
XMN4.M1	0.38062	0.38062
XMN5.M1	0.41333	0.41333
XMN6.M1	0.38062	0.38062
XMN7.M1	0.41333	0.41333
XMP0.M1	?????	?????
XMP1.M1	?????	?????
XMP6.M1	-0.36764	-0.36764
XMP7.M1	-0.36764	-0.36764

Table 5.22: Tensioni di soglia

5.4. Comparing high speed and low leakage optimization

Vengono confrontate, tramite *ezwave*, le tensioni e le correnti nei casi *High Speed* e *Low Leakage*. Tutti i grafici risultanti sono riportati nelle Figure 5.9,5.10,5.11,5.12 ,5.13.

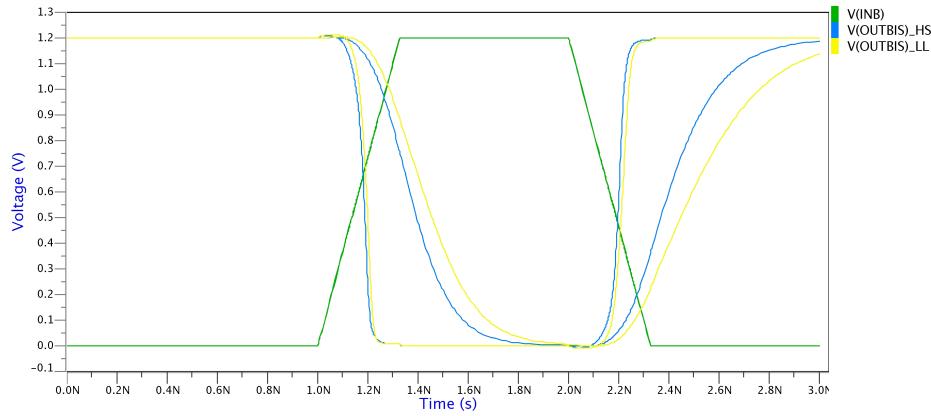


Figure 5.9: Schema circuitale porta NAND

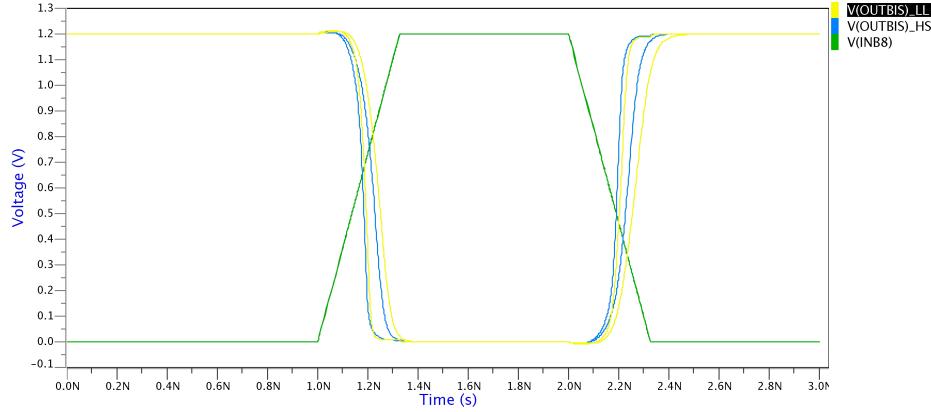


Figure 5.10: Schema circuitale porta NAND

5.4. Comparing high speed and low leakage optimization

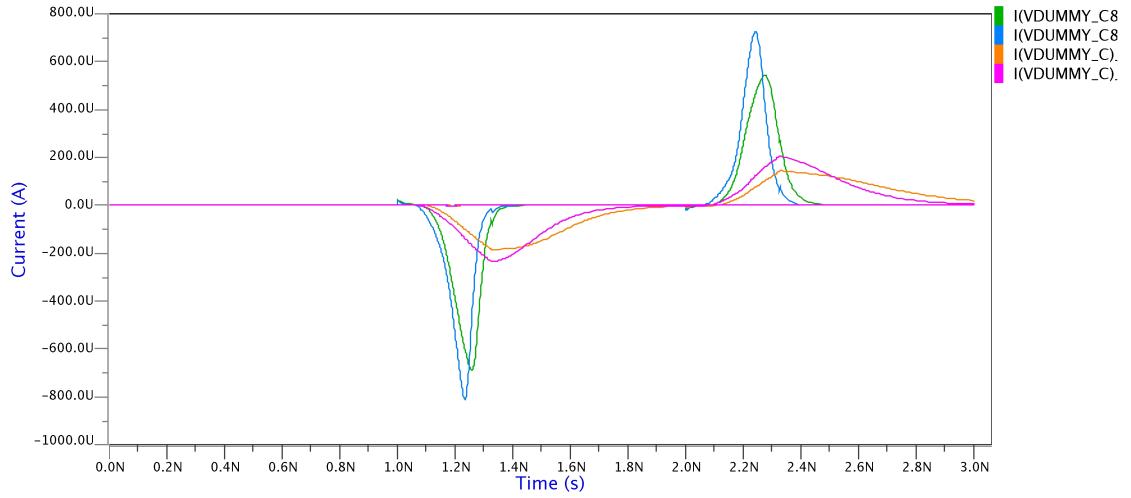


Figure 5.11: *Schema circuitale porta NAND*

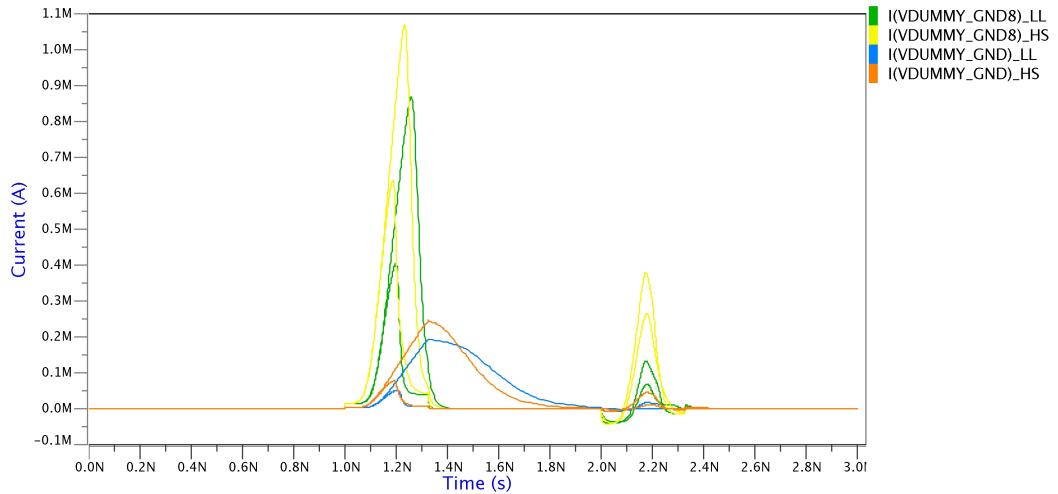


Figure 5.12: *Schema circuitale porta NAND*

5.5. Temperature dependency

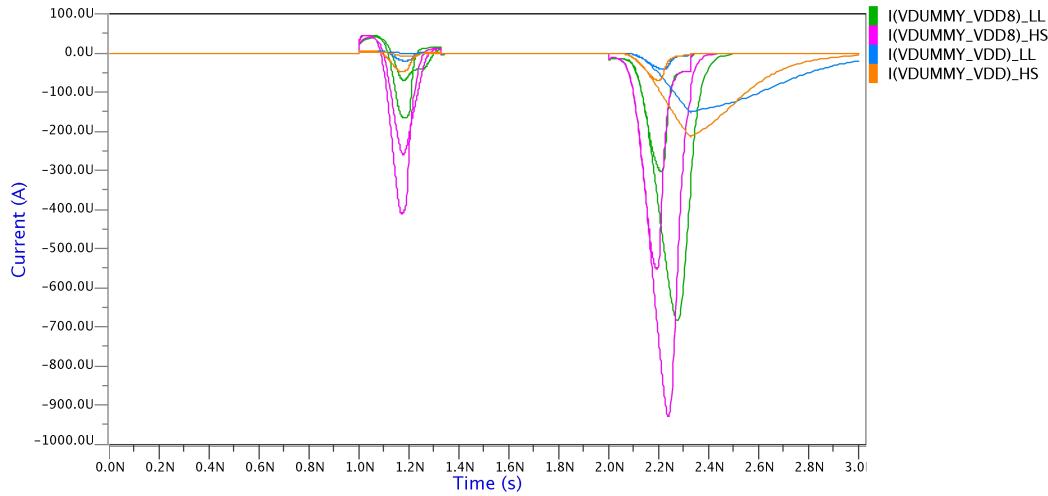


Figure 5.13: Schema circuitale porta NAND

Infine, sempre tramite la simulazione ELDO, si è andata a stimare la potenza totale dissipata che è risultata essere pari a:

$$TotalPowerDissipationX8 = 2.9686nW$$

$$TotalPowerDissipationX1 = 445.09pW$$

Andando a confrontare questi valori con quelli ottenuti nel punto precedente, si può notare come la potenza si sia ridotta drasticamente. Precisamente si è ottenuta una riduzione del 93.99% nel caso del gate X8 e del 93.44% nel caso del gate X1.

5.5 Temperature dependency

In questa sezione si è andato a studiare l'andamento della potenza e della tensione di soglia, al variare della temperatura. Andando a modificare leggermente lo script, commentando il valore di capacità di 0.06 pF e inserendo un particolare comando `.temp -40 0 40 80 120 150 180`, si ottiene una simulazione ELDO dove posso andare a vedere come varia la potenza in funzione della temperatura. I dati sono stati raccolti nella Tabella 5.23.

5.5. Temperature dependency

Temperatura	Potenza
-40°C	0.011197 nW
0°C	0.12121 nW
40°C	0.76913 nW
80°C	3.2020 nW
120°C	9.8880 nW
150°C	19.907 nW
180°C	36.351 nW

Table 5.23: *Tensioni di soglia*

Infine, tramite il programma **gnuplot** ed uno script appositamente realizzato, si ottiene un preciso plot che mostra l'andamento della potenza in funzione della temperatura. Il grafico è riportato in Figura 5.14.

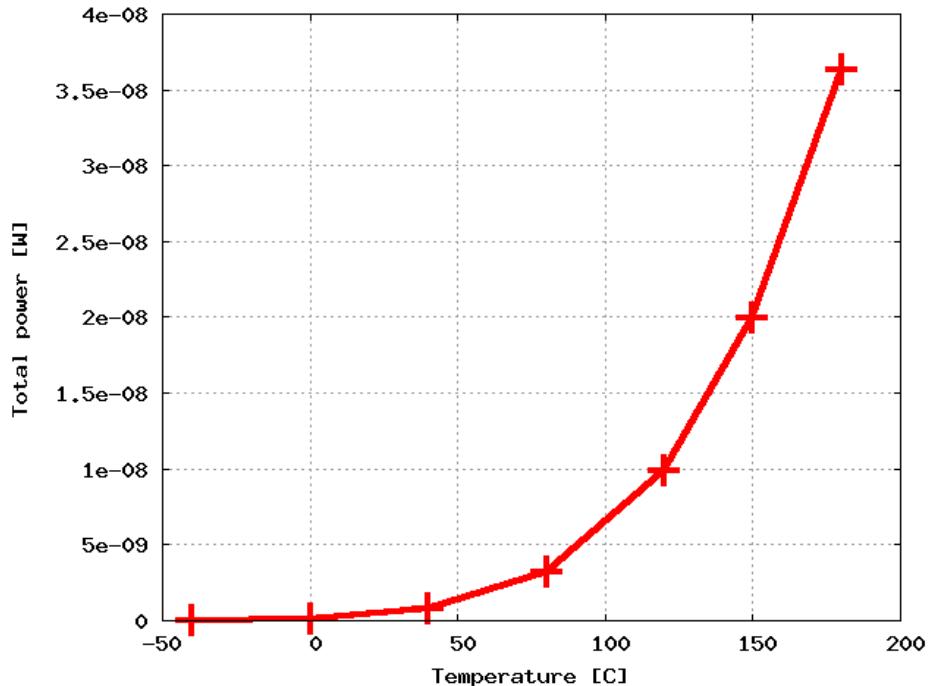


Figure 5.14: *Schema circuitale porta NAND*

Allo stesso modo si è andato a simulare il comportamento della Tensione di Soglia al variare della Temperatura. I dati sono raccolti nella Tabella 5.24 e il grafico è presente in Figura 5.15

5.6. Analysis of a memory power components

Temperatura	Tensione di Soglia V_{TH}			
	XMN0.M1	XMN1.M1	XMP0.M1	XMP1.M1
-40°C	0.45753	0.42482	-0.41556	-0.41556
0°C	0.43114	0.39843	-0.38664	-0.38664
40°C	0.40476	0.37204	-0.35772	-0.35772
80°C	0.37837	0.34566	-0.32880	-0.32880
120°C	0.35199	0.31927	-0.29988	-0.29988
150°C	0.33220	0.29949	-0.27819	-0.27819
180°C	0.31241	0.27970	-0.25650	-0.25650

Table 5.24: *Tensioni di soglia*

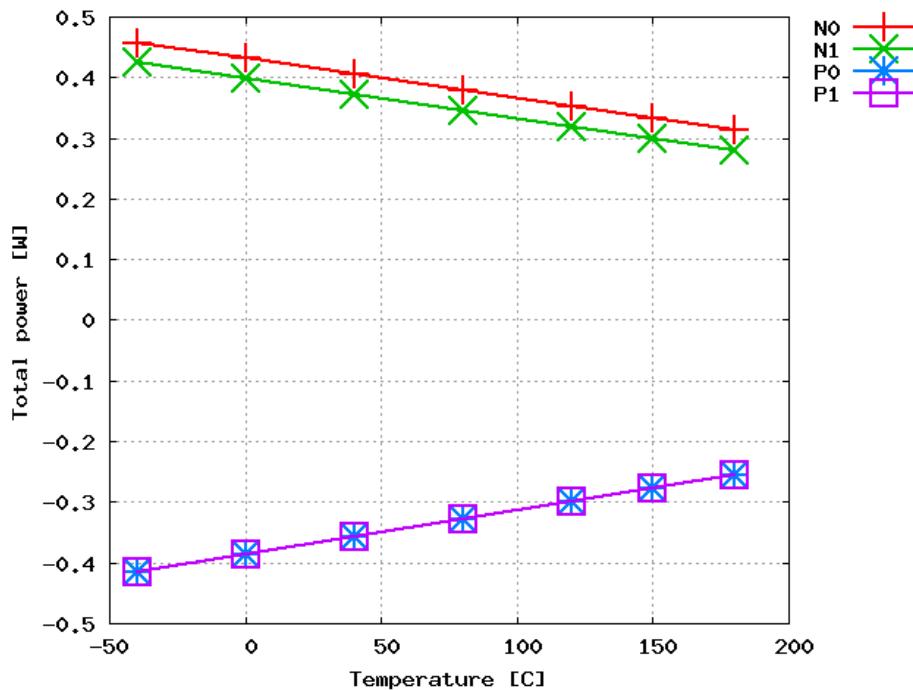


Figure 5.15: *Schema circuitale porta NAND*

SERVONO DEI COMMENTI?????? PORCA TROIA I COMMENTI

5.6 Analysis of a memory power components

Lo scopo di quest'ultimo punto dell'esercitazione è analizzare i datasheet delle varie versioni di memoria fornite e scegliere la migliore struttura di

5.6. Analysis of a memory power components

memoria che ottimizzi i consumi di potenza. Infatti dal punto di vista dei consumi è molto più conveniente dividere la struttura della memoria in banche in modo da ridurre il numero di celle coinvolte nell'operazione di accesso alla memoria. La memoria da progettare deve avere la stessa capienza di una SRAM_8192_16_16, con varie possibilità di scelta del parallelismo e delle dimensioni. Dopo aver scelto un parallelismo a 16 bit, si è andato ad analizzare il consumo delle varie memorie tramite i vari contributi di corrente con la formula fornita nella user guide delle SRAM a 90 nm:

$$I_{tot} = I_{avg} + I_{des} * N_{des}$$

$$I_{avg} = I_{sw} + I_{sby} + \frac{1}{2} * (C_{out} * V_{al} * f * N_b * N_p)$$

dove:

- I_{DES} è la corrente che scorre nella cella non selezionata
- N_{DES} è il numero di celle non selezionate ($16 - 1 = 15$)
- I_{sw} è il valore medio della corrente durante le fasi di scrittura e lettura
- I_{sby} è la corrente che scorre nella cella quando i segnali di ingresso e di uscita sono stabili
- C_{out} è il valore della capacità di carico in uscita, scelta uguale per tutte le memorie visto che il valore non è noto a priori ($1pF$)
- V_{al} è la tensione di alimentazione per il caso tipico, $1V$
- f è la frequenza operativa ($1MHz$)
- N_{bits} è il numero di bit in uscita (16)
- N_{ports} è il numero di porte della memoria scelta (2, memoria dual-port)

Sono quindi state analizzate tutte le memorie con parallelismo a 16 bit così da confrontarne i consumi in termini di corrente totale, trascurando i consumi dei multiplexer di uscita.

5.6. Analysis of a memory power components

Memory	$I_{sw} \mu A$	$I_{sby} \mu A$	$I_{des} \mu A$	$N_{bits} \mu A$	$N_{blocks} \mu A$	$I_{tot} \mu A$
SRAM 256_16_4	7.27	82	2.7	4	32	189
SRAM 512_16_4	7.70	118	2.8	4	16	184
SRAM 512_16_16	10.8	156	5.21	16	16	261
SRAM 1024_16_4	8.34	190	2.79	4	8	234
SRAM 1024_16_16	11.2	181	5.51	16	8	247
SRAM 2048_16_16	11.8	232	5.71	16	4	277
SRAM 4096_16_16	12.4	333	5.68	16	2	367

Table 5.25: *Caratteristiche delle SRAM analizzate.*

Dalla tabella si evince che la formula più conveniente con questo parallelismo è una memoria formata da 16 blocchi di SRAM 512_16_4, formata da 512 words di 16 bit con profondità dei multiplexer uguale a 4, che consuma una corrente totale di circa 184 uA.

6. Laboratorio 6: Functional Verification

6.1 VHDL Testing

6.1.1 A given RCA

Nella prima parte dell'esperienza di laboratorio viene fornito il *Testbench* di un *Ripple Carry Adder*, contenente un errore.

Il Test Bench era diviso sostanzialmente in tre parti:

- Random Input Values
- Reference Architecture
- Assert Instructions

6.1.2 A more complex case

Nella seconda parte di questo primo esercizio viene richiesto di verificare la correttezza di quattro versioni di un circuito *incrementer&comparator* presente in Figura 6.1. La struttura presenta 4 blocchi diversi, replicati due volte, e un comparatore:

- Due multiplexer a due vie che selezionano l'ingresso *A* e *B* oppure l'uscita del rispettivo sommatore con poi un Registro in uscita
- un registro per campionare l'ingresso *INCA* e *INCB* che decide se incrementare o meno l'ingresso *A* e *B*.

6.1. VHDL Testing

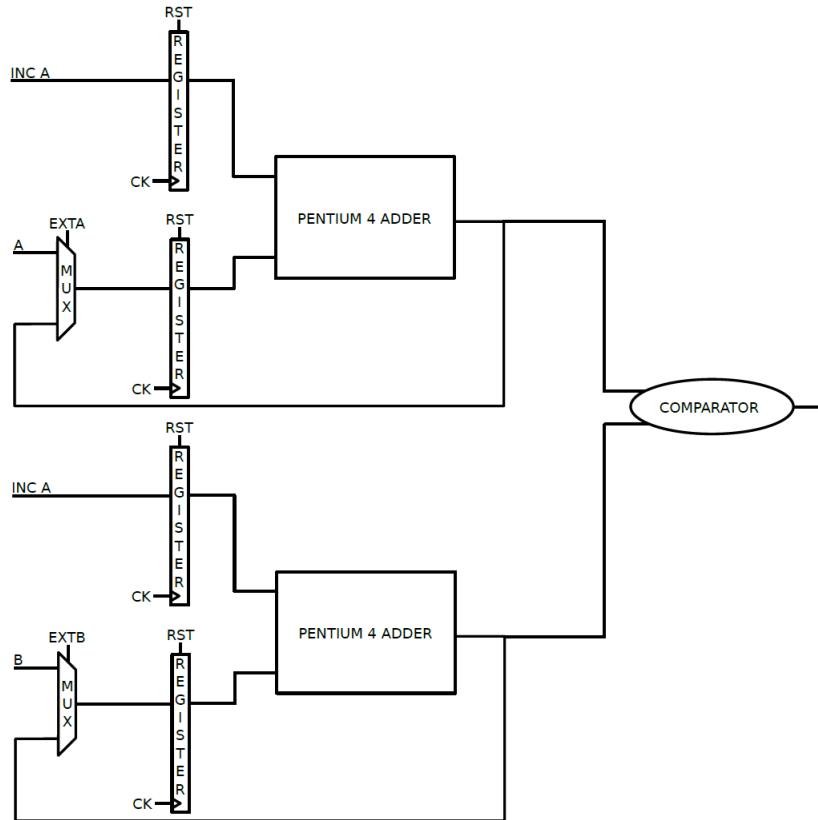


Figure 6.1: *Datapath*

- due sommatori, realizzati tramite la struttura *Pentium 4 Adder*
- un comparatore che manda in uscita il valore maggiore tra A e B

Dopo aver letto il funzionamento del Pentium 4 tramite l'appendice si è iniziata la realizzazione del TestBench per scoprire quali fossero i bug presenti. Veniva fornita una versione di riferimento del circuito, priva di bug in modo da avere un'architettura di riferimento con la quale confrontare le 4 versioni del circuito.

Il VHDL implementato per questo scopo è presente in appendice, fig. ?? e 7.1.

6.1. VHDL Testing

La struttura del Testbench è simile a quella fornita nell'esercizio precedente. Si ha una prima parte di generazione di numeri casuali, nello specifico si generano in modo random gli ingressi A e B, i selettori dei multiplexer EXTA e EXTB e gli ingressi di incremento INCA e INCB.

In realtà, simulando il circuito ci si è resi conto di come la soluzione con numeri casuali non sia sempre ottimale per la ricerca di un bug, in quanto, specialmente nel caso di un ingresso con un numero di bit molto elevato, sono necessarie numerose iterazioni per provare tutte le possibili combinazioni. Questo sta alla base della mancata ritrovo del bug nella quarta versione, in quanto tramite un Testbench con numeri random non siamo riusciti a "sollecitare" l'errore: probabilmente si sarebbe dovuto studiare degli ingressi opportuni per sollecitare il sommatore in modo più specifico.

La seconda parte del Testbench prende gli ingressi sviluppati nella parte prima e genera le uscite sia dell'architettura di riferimento che della versione da analizzare.

Infine viene effettuato il confronto tra le due uscite e viene definita una funzione di *assert* che fornisce sul *Transcript* di Modelsim un messaggio di errore nel caso in cui venga rilevato un bug. Nella prima versione del circuito risulta un errore all'interno di un multiplexer presente nel selettore di riporto del sommatore, in quanto l'ingresso riferito all' '1' e allo '0' risultavano invertiti. Questa prima versione ha un errore molto "visibile" e facile da trovare, sono state necessarie quindi pochissime iterazioni.

Nella seconda versione del circuito invece, risulta un errore nel selettore del multiplexer del sommatore, in quanto invece di assegnare

$$C_IN(N_BIT/CARRY - 1)$$

assegna

$$C_IN(N_BIT/CARRY - 2)$$

Per trovare quest'errore sono state necessarie 500000000 iterazioni, un numero già ritenuto eccessivo. Si riporta in allegato in Figura 6.2 la simulazione Modelsim contenente l'errore.

6.1. VHDL Testing

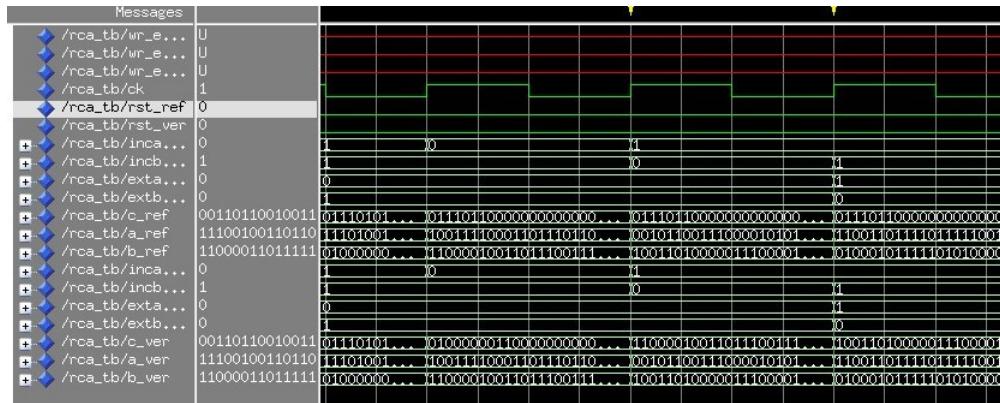


Figure 6.2: Waves, Modelsim simulation

6.1.3 Finite State Machine

Questa parte prevede di testare il funzionamento di una FSM (Macchina a stati finiti), disponibile in 4 versioni differenti, contenenti dei bug. La FSM da testare è un contatore up/down a 3 bit, la FSM è riportata alla fig. 6.3:

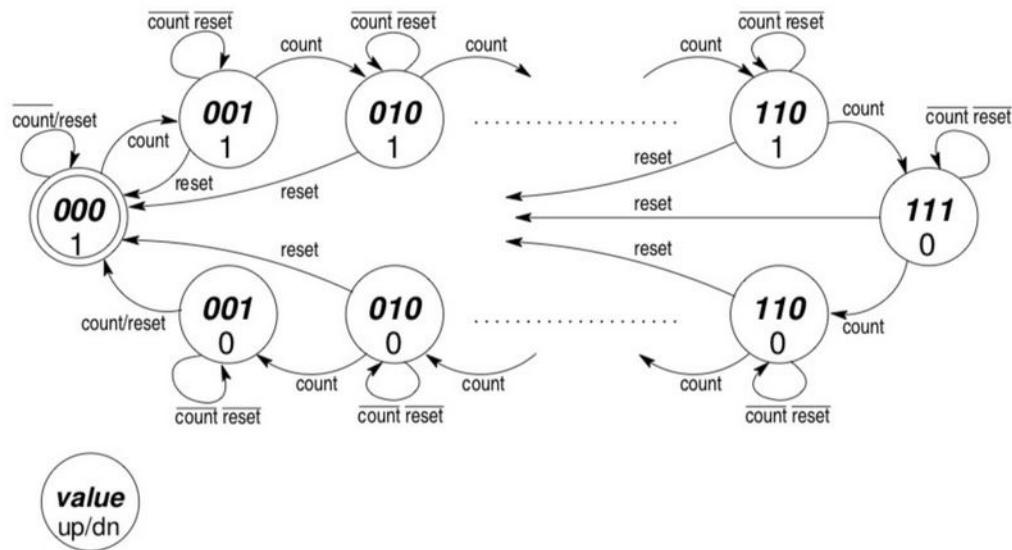


Figure 6.3: Macchina a stati da testare

6.1. VHDL Testing

Prendendo spunto dal punto precedente il test è stato realizzato mediante i seguenti file:

- *counter_ref.vhd*, è il file del contatore dal corretto funzionamento, utilizzato per calcolare le uscite corrette da confrontare successivamente.
- *counter.vhd*, è il dispositivo da testare, disponibile in 4 versioni, site in diverse cartelle, poi estratte da modelsim volta per volta durante la simulazione
- *counter_tb.do*, è lo script che si occupa di dirigere la compilazione e simulazione dei 4 files da testare su Modelsim, in cui è impostata anche la durata della simulazione e la visualizzazione delle forme d'onda dei segnali rilevanti ai fini della verifica, per consentire una comoda visione dell'andamento temporale dei segnali. Di seguito un estratto del file disponibile in appendice fig. ?? 7.2.

.....

.....

```
echo "testing v1..."  
vcom -reportprogress 300 -work work  
/home/lp19.12/repository/lab6/ese6/3/counter_tb.vhd  
/home/lp19.12/repository/lab6/ese6/3/vREF/counter_ref.vhd  
/home/lp19.12/repository/lab6/ese6/3/v1/counter.vhd
```

```
vsim -t ns -novopt work.counter_tb
```

```
set NumericStdNoWarnings 1
```

```
run 0 ps
```

```
set NumericStdNoWarnings 0
```

```
add wave -radix binary -color BLUE
```

```
sim:/counter_tb/DATA_OUT_tb
```

```
add wave -radix binary -color GREEN
```

```
sim:/counter_tb/DATA_OUT_ref
```

```
add wave -radix binary -color BLUE
```

6.1. VHDL Testing

```
sim:/counter_tb/UP_DN_tb
add wave -radix binary -color GREEN
sim:/counter_tb/UP_DN_ref
run 30000000 ns
```

```
echo "end v1"
```

```
.....  
.....
```

- *counter_tb.vhd*, il file di testbench che si occupa di:

- generare gli ingressi e definire i componenti: il REF, dispositivo di riferimento utilizzato per calcolare le uscite corrette a partire dagli ingressi generati; il DUT (device under test), il contatore da verificare che sarà rappresentato di volta in volta nella simulazione da uno dei quattro circuiti da testare.
- generare il segnale di clock e i gli ingressi generati in modo pseudo-casuale.
- confrontare le uscite dei due dispositivi per rilevare possibili errori e segnalarli con un apposito messaggio.

Di seguito sono riportate le parti più significative del file di testbench, disponibile integralmente in appendice fig. ?? ?? ??.

```
.....  
.....  
for I in 1 to 50000 loop  
  - Random number generation  
  UNIFORM(seed1, seed2, rand);  
  
  std_rand:=std_logic_vector(to_unsigned(integer(rand), 2));
```

```
COUNT_TB<= std_rand(0);
RST_TB<= std_rand(1);

- Assert
wait for 2 ns;
assert (DATA_OUT_TB==DATA_OUT_REF) report "There is a bug in result." severity warning;
assert (UP_DN_TB==UP_DN_REF) report "There is a bug in UP_DN signal" severity warning;
wait for 10 ns;
end loop;
wait;
.....
.....
```

E' evidente come testare una macchina a stati sia più complicato e dispendioso dal punto di vista del testing rispetto all'analisi di un circuito combinatorio, visto che l'uscita dipende anche dagli stati precedenti

6.2 Scripting in Python

6.2.1 Automatic verification through a script

Un modo per automatizzare il processo di verificare è realizzare uno script python che consente una verifica più veloce ed efficace. Lo scopo della prima parte è quella di completare il template del file testing_script.py, per testare il circuito del RCA utilizzato nella prima parte del laboratorio.

il file testing_script.py si compone di 5 parti:

- generazione degli input file in formato .txt da mandare poi al DUT
- lancio della simulazione Modelsim tramite comandi bash

6.2. Scripting in Python

- calcolo dei risultati tramite algoritmo di python che servirà a fornire i valori di riferimento per il confronto con i valori di uscita del dut
- prelievo dei risultati della simulazione modelsim salvati su file *output_data.txt*
- comparazione dei risultati ottenuti da modelsim con quelli di riferimento calcolati da python

Per la generazione pseudo-casuale degli input si è utilizzata la struttura, già fornita nel file *generate_inputs.py*:

```
.....  
.....  
# Loop for all steps  
for i in range(0,NumberOfSteps):  
    # Loop for all signals  
    for i in range (0, SignalsNumber):  
        # Generate integer number  
        RandomInputInteger=randint(1,2**GenericsAssignedValue[i]-1)  
        # Convert number to binary  
        RandomInputBinary = ('0:0{l}b'.format((RandomInputInteger),l=GenericsAssignedValue/  
        # Write file  
        Filepointer.write(RandomInputBinary + " ")  
        # Write line termination  
        Filepointer.write(" n")  
.....  
.....
```

La funzione *randint()* permette di generare un intero pseudocasuale entro il range fornito. Il metodo *format()* permette di convertire l'intero in binario in base al parallelismo degli ingressi definito nella lista *GenericsAssignedValue*.

Per quanto riguarda il lancio della simulazione Modelsim, il codice è già

6.2. Scripting in Python

stato fornito e si compone di istruzioni bash richiamate tramite il modulo *os* contenente la funzione *system()*.

Lo script si occupa di eliminare eventuali working directories ed input file, impostare le environment variables, e far partire la compilazione e simulazione di Modelsim. Il testo è disponibile in appendice fig. ?? 7.3 alla sezione “run modelsim simulation” del file *testing_script.py*.

Per il calcolo delle uscite con python si è utilizzato il seguente algoritmo:

```
.....
.....
#calculate the output values from the inputs generated
#repeat for each set of in's
for i in range(0, NumberOfSteps):
    #sum in's
    Sum=Inputs[i][0]+Inputs[i][1]+Inputs[i][2]
    #conversion of the sum in binary
    SumBin=('0:b'.format(Sum))
    #conversion in string
    SumStr=str(SumBin)
    #overflow bit preset to '0'
    Py_results[i][1]=0
    if len(SumStr)!=32:
        #management of the overflow
        if len(SumStr)>32:
            SumStr=SumStr[1:]
            #overflow bit to '1'
            Py_results[i][1]=1
        else:
            #adding zeros to get the value on 32 bit
            while len(SumStr)<32:
                SumStr='0'+SumStr
    #re-conversion to successive comparison of the values
    Py_results[i][0]=int(SumStr, 2)
.....
```

.....

Al momento della generazione, gli ingressi sono stati salvati nella lista *Inputs*[], formata da tutti gli ingressi generati, raggruppati a gruppi di 3, ogni gruppo corrisponde a uno step di ingressi da calcolare. Si sono sommati gli ingressi e si è poi convertita la somma in binario grazie al metodo *format()*. Per gestire gli overflow e mantenere il parallelismo della somma a 32 bit, si è preferito convertire la somma binaria in stringa in modo da gestire più comodamente il dato nel caso in cui il parallelismo sia variato. I risultati corretti sono stati salvati nella lista *Py_results*[], contenente i gruppi di risultati in intero a coppie di 2 (somma e carry-out), per ogni step di ingressi. I risultati inoltre sono stati salvati nel file *python_results.txt*

Per il prelievo dei risultati della simulazione i dati sono stati prelevati dal file *output_data.txt* nel quale modelsim ha salvato i valori di uscita. Si è utilizzato lo script già fornito leggendo ogni riga contenente le uscite di ogni step di calcolo tramite un ciclo for.

.....

.....

```
# Open file and acquire data
with open(Filename, "r") as f:
    OutputsBinary = [x.strip().split() for x in f]
```

.....

.....

Per l'ultima parte, quella che si occupa della comparazione dei risultati, sono stati confrontati in modo ordinato i valori *OutputsDecimal*[] provenienti dalla simulazione con quelli di riferimento salvati in *Py_results*[] con un ciclo for. Per ogni step, in caso di valori diversi viene stampato un messaggio con l'avviso di errore e lo step i-esimo dell'errore. Inoltre alla fine del confronto viene anche riportato il numero totale di errori riscontrati. Di seguito un estratto del codice:

```

.....
.....
tot_err=0
#for each results' step
for i in range(0, NumberOfSteps):
    #comparison the outputs
    if str(OutputsDecimal[i])!=str(Py_results[i]):
        #error message for step "i"
        print("found an error in step "+str(i)+""
n")
        print(str(OutputsDecimal[i]), str(Py_results[i]))

tot_err=tot_err+1
#message for total number of errors
print("The number of wrong steps is: "+str(tot_err))
.....
.....

```

6.2.2 Further Generalization

L’ultima parte prevede di creare uno script che sia in grado di generare un file di testbench a partire dal file .vhd del device under test. Il file contente il codice di generazione del testbench è *tb_generation.py* file già in parte fornito, che è possibile trovare in appendice fig. ?? , che si occupa di:

- leggere il file del dut e salvare tutti i valori utili al testbench, ovvero i generics e gli ingressi e uscite del device. Di seguito un estratto del codice:

```

.....
.....
with open(fileName, "r") as fp:
    #save rows
```

6.2. Scripting in Python

```
for line in fp:  
    #divide the line in words  
    words=line.strip().split()  
    #save words in a list  
    for w in words:  
        word.append(w)  
    #scanning words to take datas for testbench  
    while word[i]!="architecture":  
        if word[i]=="entity":  
            EntityName=word[i+1]  
            i=i+3  
        if word[i]=="generic":  
            GenericFlag=1  
            i=i+2  
            while word[i]==";":  
                GenericsList.append(word[i])  
                i=i+2  
            GenericsType.append(word[i])  
            i=i+2  
        .....  
        .....
```

- creare il file di testbench, eliminando eventuali file con lo stesso nome
- definire il file di testbench in formato .vhd, utilizzando i dati estratti

Le parti 2 e 3 sono già state fornite, il file completo è comunque disponibile in appendice. La generalizzazione della creazione del testbench consente di automatizzare ulteriormente il processo di testing del DUT, rendendo lo script più flessibile e adatto a simulazioni future, che grazie a questo file saranno rese più veloci ed efficienti.

7. Appendix

7.1 lab2

7.2 lab3

7.3. lab6

7.3 lab6

7.3.1 Testbench Incomp

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.numeric_std.all ;
use ieee.math_real.all ; -- for UNIFORM, TRUNC

entity Testbench_incomp is
end entity ;

architecture arch of Testbench_incomp is
-- componente di riferimento
component incomp_ref is
generic
(
    N_BIT : integer
);
port(
    CK: in std_logic;
    RST: in std_logic;
    INCA: in std_logic;
    INCB: in std_logic;
    EXTA: in std_logic;
    EXTB: in std_logic;
    A: in signed (N_BIT-1 downto 0);
    B: in signed (N_BIT-1 downto 0);
    C: out signed (N_BIT-1 downto 0)
);
end component incomp_ref;
-- componente varie versioni
component incomp is
generic
(
    N_BIT : integer
);
port(
    CK: in std_logic;
    RST: in std_logic;
    INCA: in std_logic;
    INCB: in std_logic;
    EXTA: in std_logic;
    EXTB: in std_logic;
    A: in signed (N_BIT-1 downto 0);
    B: in signed (N_BIT-1 downto 0);
    C: out signed (N_BIT-1 downto 0)
);
end component incomp;

signal CK,RST_REF,RST_VER : std_logic;
-- riferimenti
signal INCA_REF: std_logic;
signal INCB_REF: std_logic;
signal EXTA_REF: std_logic;
signal EXTB_REF: std_logic;
signal C_REF:signed(31 downto 0);
signal A_REF:signed(31 downto 0);
signal B_REF:signed(31 downto 0);
-- versione test
signal INCA_VER: std_logic;
signal INCB_VER: std_logic;
signal EXTA_VER: std_logic;
signal EXTB_VER: std_logic;
signal C_VER:signed(31 downto 0);
signal A_VER:signed(31 downto 0);
signal B_VER:signed(31 downto 0);

begin
begin
Versione_riferimento : incomp_ref
generic map (
    N_BIT => 32
)
port map(CK=> CK,RST=> RST_REF,INCA=> INCA_REF,INCB=> INCB_REF,EXTA=> EXTA_REF,EXTB=>
EXTB_REF,A=> A_REF,B=> B_REF,C=> C_REF);

Versione_test: incomp
generic map (N_BIT => 32)
port map(CK=>CK, RST=>RST_VER,INCA=> INCA_VER,INCB=> INCB_VER,EXTA=> EXTA_VER,EXTB=>
EXTB_VER,A=> A_VER,B=> B_VER,C=> C_VER);

```

7.3. lab6

```

--CLOCK
PROCESS
BEGIN
FOR Z IN 1 TO 500
LOOP
    CK <= '1' ;
    WAIT FOR 20 ns ;
    CK<= '0' ;
    WAIT FOR 20 ns ;
END LOOP;
WAIT;
END PROCESS;

random: process
variable seed1, seed2: positive;
variable rand_1: real;
variable A_rand: real;
variable B_rand: real;
variable INCA_RAND: real;
variable INCB_RAND: real;
variable EXTA_RAND:real;
variable EXTB_RAND:real;
variable C_REF: integer;
variable int_rand_1: integer;
variable int_rand_2: integer;
variable int_rand_3: integer;
variable int_rand_4: integer;
variable int_rand_5: integer;
variable int_rand_6: integer;
begin
RST_REF<='1';
RST_VER<='1';
wait for 40 ns;
for i in 1 to 500 loop
RST_REF<'0';
RST_VER<'0';

-- Random number generation
UNIFORM(seed1, seed2, A_rand);
UNIFORM(seed1, seed2, B_rand);
UNIFORM(seed1, seed2, INCA_RAND);
UNIFORM(seed1, seed2, INCB_RAND);
UNIFORM(seed1, seed2, EXTA_RAND);
UNIFORM(seed1, seed2, EXTB_RAND);

int_rand_1 := INTEGER(TRUNC(A_rand*(4294967296.0) - (2147483649.0)));
int_rand_2 := INTEGER(TRUNC(B_rand*(4294967296.0) - (2147483649.0)));
int_rand_3 := INTEGER(TRUNC(INCA_rand*(2.0) - (2.0)));
int_rand_4 := INTEGER(TRUNC(INCB_rand*(2.0) - (2.0)));
int_rand_5 := INTEGER(TRUNC(EXTA_RAND*(2.0) - (2.0)));
int_rand_6 := INTEGER(TRUNC(EXTB_RAND*(2.0) - (2.0)));

-- riferimento
A_REF <= to_signed(int_rand_1, A_REF'LENGTH);
B_REF <= to_signed(int_rand_2, B_REF'LENGTH);
INCA_REF <= std_logic_vector(to_signed(int_rand_3,1));
INCB_REF <= std_logic_vector(to_signed(int_rand_4,1));
EXTA_REF <= std_logic_vector(to_signed(int_rand_5, 1));
EXTB_REF <= std_logic_vector(to_signed(int_rand_6, 1));

-- Versioni test
A_VER <= to_signed(int_rand_1, A_REF'LENGTH);
B_VER <= to_signed(int_rand_2, B_REF'LENGTH);
INCA_VER <= std_logic_vector(to_signed(int_rand_3,1));
INCB_VER <= std_logic_vector(to_signed(int_rand_4,1));
EXTA_VER <= std_logic_vector(to_signed(int_rand_5, 1));
EXTB_VER <= std_logic_vector(to_signed(int_rand_6, 1));

--Assert
assert (C_VER = C_REF) report "There is a bug." severity warning;
wait for 40 ns;
end loop;
wait;
end process;
end architecture ;

```

Figure 7.1: *Testbench Incomp*

7.3.2 Counter_tb.vhd

7.3. lab6

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all ; -- for UNIFORM, TRUNC

entity counter_tb is
end entity; --TESTBENCH

architecture arch of counter_tb is

component counter is
port
(
    CLK          : in std_logic;
    RST          : in std_logic;
    COUNT        : in std_logic;
    DATA_OUT     : out unsigned(8-1 downto 0);
    UP_DN        : out std_logic
);
end component counter;

component counter_ref is
port
(
    CLK          : in std_logic;
    RST          : in std_logic;
    COUNT        : in std_logic;
    DATA_OUT     : out unsigned(8-1 downto 0);
    UP_DN        : out std_logic
);
end component counter_ref;

--signals
signal CLK_TB           : std_logic:='0';
signal RST_TB           : std_logic:='0';
signal COUNT_TB         : std_logic:='0';
signal DATA_OUT_TB      : unsigned(8-1 downto 0);
signal UP_DN_TB          : std_logic;
signal DATA_OUT_REF      : unsigned(8-1 downto 0);
signal UP_DN_REF          : std_logic;

begin

--clock_gen
CLK_TB<=not(CLK_TB) after 10 ns;

DUT : counter
port map (
    CLK    => CLK_TB,
    RST    => RST_TB,
    COUNT  => COUNT_TB,
    DATA_OUT => DATA_OUT_TB,
    UP_DN   => UP_DN_TB
);

REF : counter_ref
```

7.3.3 Counter_tb.do

7.3. lab6

```
port map      (
    CLK    => CLK_TB,
    RST    => RST_TB,
    COUNT => COUNT_TB,
    DATA_OUT => DATA_OUT_REF,
    UP_DN  => UP_DN_REF
);

random: process
variable seed1: positive;
variable seed2: positive;
variable rand: real;
--variable rand_2: real;

variable std_rand: std_logic_vector(1 downto 0);
--variable int_rand_2: integer;
--variable sum: signed(63 downto 0);

begin
--CLK_TB    <= '0';
--COUNT_TB <= '0';
--RST_TB   <= '0';
--wait for 5 ns;

for I in 1 to 50000 loop
    -- Random number generation
    UNIFORM(seed1, seed2, rand);

    -- int rand_1 := INTEGER(TRUNC(rand_1*(18446744073709551616.0) -
    --(9223372036854775809.0)));
    -- int rand_2 := INTEGER(TRUNC(rand_2*(18446744073709551616.0) -
    --(9223372036854775809.0)));
    std_rand:=std_logic_vector(to_unsigned(integer(rand), 2));

    COUNT_TB    <= std_rand(0);
    RST_TB     <= std_rand(1);

    -- Reference architecture

    -- Assert
    wait for 2 ns;
    assert (DATA_OUT_TB=DATA_OUT_REF) report "There is a bug in
result." severity warning;
    assert (UP_DN_TB=UP_DN_REF) report "There is a bug in UP_DN signal"
severity warning;
    wait for 10 ns;
end loop;

wait;

end process;

end architecture ; -- arch
```

7.3. lab6

```
echo "testing v1..."  
vcom -reportprogress 300 -work work  
/home/lpi9.12/repository/lab6/ese6/3/counter_tb.vhd  
/home/lpi9.12/repository/lab6/ese6/3/vREF/counter_ref.vhd  
/home/lpi9.12/repository/lab6/ese6/3/v1/counter.vhd  
  
vsim -t ns -novopt work.counter_tb  
  
set NumericStdNoWarnings 1  
run 0 ps  
set NumericStdNoWarnings 0  
  
add wave -radix binary -color BLUE      sim:/counter_tb/DATA_OUT_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/DATA_OUT_ref  
add wave -radix binary -color BLUE      sim:/counter_tb/UP_DN_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/UP_DN_ref  
run 30000000 ns  
echo "end v1"  
  
echo "testing v2..."  
vcom -reportprogress 300 -work work  
/home/lpi9.12/repository/lab6/ese6/3/counter_tb.vhd  
/home/lpi9.12/repository/lab6/ese6/3/vREF/counter_ref.vhd  
/home/lpi9.12/repository/lab6/ese6/3/v2/counter.vhd  
  
vsim -t ns -novopt work.counter_tb  
  
set NumericStdNoWarnings 1  
run 0 ps  
set NumericStdNoWarnings 0  
  
add wave -radix binary -color BLUE      sim:/counter_tb/DATA_OUT_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/DATA_OUT_ref  
add wave -radix binary -color BLUE      sim:/counter_tb/UP_DN_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/UP_DN_ref  
run 30000000 ns  
echo "end test v2"  
  
echo "testing v3..."  
vcom -reportprogress 300 -work work  
/home/lpi9.12/repository/lab6/ese6/3/counter_tb.vhd  
/home/lpi9.12/repository/lab6/ese6/3/vREF/counter_ref.vhd  
/home/lpi9.12/repository/lab6/ese6/3/v3/counter.vhd  
  
vsim -t ns -novopt work.counter_tb  
  
set NumericStdNoWarnings 1  
run 0 ps  
set NumericStdNoWarnings 0  
  
add wave -radix binary -color BLUE      sim:/counter_tb/DATA_OUT_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/DATA_OUT_ref  
add wave -radix binary -color BLUE      sim:/counter_tb/UP_DN_tb  
add wave -radix binary -color GREEN     sim:/counter_tb/UP_DN_ref  
run 30000000 ns  
echo "end test v3"  
  
echo "testing v4..."  
vcom -reportprogress 300 -work work  
/home/lpi9.12/repository/lab6/ese6/3/counter_tb.vhd
```

7.3.4 Testing_script.py

7.3. lab6

```
/home/lp19.12/repository/lab6/ese6/3/vREF/counter_ref.vhd  
/home/lp19.12/repository/lab6/ese6/3/v4/counter.vhd  
  
vsim -t ns -novopt work.counter_tb  
  
set NumericStdNoWarnings 1  
run 0 ps  
set NumericStdNoWarnings 0  
  
add wave -radix binary -color BLUE sim:/counter_tb/DATA_OUT_tb  
add wave -radix binary -color GREEN sim:/counter_tb/DATA_OUT_ref  
add wave -radix binary -color BLUE sim:/counter_tb/UP_DN_tb  
add wave -radix binary -color GREEN sim:/counter_tb/UP_DN_ref  
run 30000000 ns  
echo "end test v4"
```

Figure 7.2: *Script di comando Modelsim*

7.3.5 Tb_generator.py

7.3. lab6

```
#!/usr/bin/env python
import os
import subprocess
from random import *

#####
# Create input data file
#####

# Number of signals
SignalsNumber = 3

# List containing the parallelism of each signal (the first one is
# referred to "IN1", the second one is referred to "IN2", ...)
GenericsAssignedValue = [32, 32, 1]

# Number of steps
NumberOfSteps = 100

# Name of the file
Filename = "file_INPUTS.txt";
#####

#####
# File creation
#####

# Clear previous version of the file
# Linux
CommandToEliminateTest = "rm ./" + Filename
os.system(CommandToEliminateTest)
# Windows
#CommandToEliminateTest = "del " + Filename
#os.system(CommandToEliminateTest) # Command used to execute external
commands on the terminal

# Create and open file
Filepointer = open(Filename, "w")

#####

#####
# Generate values
#####
RandomInputIntegerList=range(0,NumberOfSteps)
# Loop for all steps
Inputs=[[0,0,0] for x in range(0, NumberOfSteps)]
for i in range(0,NumberOfSteps):
    # Loop for all signals
    for j in range (0, SignalsNumber):
        # Generate integer number
        RandomInputInteger = randint(1,2**GenericsAssignedValue[j]-1)
        RandomInputIntegerList[j]=RandomInputInteger
        # Convert number to binary
        RandomInputBinary =
        ('{0:0{1}b}'.format((RandomInputInteger),l=GenericsAssignedValue[j]))
        Inputs[i][j]=RandomInputInteger
    # Write file
```

7.3. lab6

```
        Filepointer.write(RandomInputBinary + " ")
        # Write line termination
        Filepointer.write("\n")
        # Closing testbench file
        Filepointer.close()
#####
#####
#####

#####
# Run modelsim simulation
#####

# os.system is a command that allow you to execute external commands on a
shell

# Clean folder. Deleting working directories and outputs
os.system("rm -rf work")
os.system("rm transcript")
#os.system("rm vsim.wlf")

# Setting up environment variables
os.environ["PATH"] += os.pathsep +
"/software/mentor/modelsim_6.5c/modeltech/linux_x86_64/"
os.environ["LM_LICENSE_FILE"] = "1717@led-x3850-1.polito.it"

# Print out environment variables
os.system("echo $PATH")
os.system("echo $LM_LICENSE_FILE")

# Launch Modelsim simulation
print ("Starting simulation...")
process = subprocess.call(["vsim", "-c", "-do", "compile.do"])
print ("Simulation completed")
#####

#####
# Calculate output values through python
#####

#create and open the reference outputs file
Filename="python_results.txt"
Filepointer=open(Filename, "w")

#initialization list of python results
Py_results=[[0,0] for x in range(0, len(RandomInputIntegerList))]

gr
    #save results in file
    Filepointer.write(SumStr+' ')
    Filepointer.write('{0:b}'.format(Py_results[i][1]))
    Filepointer.write("\n")

Filepointer.close()
#####
```

Figure 7.3: *Script file di testing completo*

7.3. lab6

```
#!/usr/bin/env python
import os
from random import *

#name of the DUT's file
fileName="rcs.vhd"

#####
#           initialization data structure
#####
GenericsList=[]
GenericsType=[]
GenericsDefaultValue=[]
GenericsAssignedValue=[]
GenericFlag=0
SignalsList=[]
SignalsMode=[]
SignalsNumber=[]
SignalsSize=[]
SignalsSizeAssigned=[]
SignalsType=[]
word=[]
flag_port=1
i=0
#####

#####
#           Taking datas
#####
#open file to read datas
with open(fileName, "r") as fp:
    #save rows
    for line in fp:
        #divide the line in words
        words=line.strip().split()
        #save words in a list
        for w in words:
            word.append(w)
    #scanning words to take datas for testbench
    while word[i]!="architecture":
        if word[i]=="entity":
            EntityName=word[i+1]
            print EntityName
            i=i+3
        if word[i]=="generic":
            GenericFlag=1
            print GenericFlag
            i=i+2
            while word[i]!=";":
                GenericsList.append(word[i])
                print GenericsList
                i=i+2
            GenericsType.append(word[i])
            i=i+2
            GenericsDefaultValue.append(word[i])
            GenericsAssignedValue.append(word[i])
            i=i+1
```

7.3. lab6

```
if word[i]=="port" and flag_port:
    flag_port=0
    i+=2
    while word[i]!=");":
        SignalsList.append(word[i])
        i=i+2
        SignalsMode.append(word[i])
        i=i+1
        print word[i]
        SignalsType.append(word[i])
        if word[i+2]=="downto":
            i=i+1
            SignalsSize.append(word[i])
            SignalsSizeAssigned.append(word[i])
            i=i+3
        else:
            SignalsSize.append("1")
            SignalsSizeAssigned.append("1")
            i=i+1
    i+=1
print SignalsList
fp.close()

# Number of simulation steps
NumberOfSteps = 3000
# Delay between steps
DelayBetweenSteps = "3 ns"
#####
# File creation
#####
# Creation of file name
TestbenchFileName ="tb_"+fileName

# Clear previous version of the file
# Linux
CommandToEliminateTest = "rm ./" + TestbenchFileName
os.system(CommandToEliminateTest)
# Windows
#CommandToEliminateTest = "del " + TestbenchName
#os.system(CommandToEliminateTest) # Command used to execute external
commands on the terminal

# Create and open testbench file
TestbenchFile = open(TestbenchFileName, "w")
#####
# Libraries definition
#####
# Write libraries definition
TestbenchFile.write("library ieee;\n")    # Command used to write on a
file
TestbenchFile.write("use ieee.std_logic_1164.all;\n")
TestbenchFile.write("use ieee.numeric_std.all;\n")
TestbenchFile.write("use ieee.std_logic_textio.all;\n")
```

7.3. lab6

```
TestbenchFile.write("use STD.textio.all;\n\n")

#####
# Entity creation
#####

# Write testbench entity
TestbenchFile.write("entity " + FileName + " is\n")
TestbenchFile.write("end " + FileName + ";" + "\n")

#####

#####
# Architecture definition
#####

# Architecture definition
TestbenchFile.write("architecture test of " + FileName + " is\n\n")

#####

#####
# Component declaration
#####

# Component declaration
TestbenchFile.write("component " + EntityName + "\n")
# Check if there are generics in the code
if GenericFlag == 1:
    # Write generics declaration
    TestbenchFile.write("generic (\n")
    # Check if there is only one generic
    if len(GenericsList) == 1:
        TestbenchFile.write("      " + GenericsList[0] + ": " +
GenericsType[0] + " := " + str(GenericsDefaultValue[0]) + ");\n")
        # Check if there are more than one generics
    elif len(GenericsList) > 1:
        # Loop on all generics
        for i in range(0,len(GenericsList)):
            # Check if we have reached the last element of the list
            if i != (len(GenericsList)-1):
                TestbenchFile.write("      " + GenericsList[i] +
": " + GenericsType[i] + " := " + str(GenericsDefaultValue[i]) + ";" + "\n")
            # if we have reached the last element
            else:
                TestbenchFile.write("      " +
GenericsList[len(GenericsList)-1] + ": " +
GenericsType[len(GenericsList)-1] + " := " +
str(GenericsDefaultValue[len(GenericsList)-1]) + ");\n")
    # Write ports declaration
    TestbenchFile.write("port (\n")
    for i in range(0,len(SignalsList)):
        # Check if we have reached the last element of the list
        if i != (len(SignalsList)-1):
```

7.3. lab6

```

# Check if single bit
if SignalsType[i] == "std_logic":
    TestbenchFile.write("      " + SignalsList[i] + ": " +
SignalsMode[i] + " " + SignalsType[i] + ";" + "\n")
    # if vector
    else:
        TestbenchFile.write("      " + SignalsList[i] + ": " +
SignalsMode[i] + " " + SignalsType[i] + "(" + SignalsSize[i] + " downto
0); \n")
        # if we have reached the last element
        else:
            # Check if single bit
            if SignalsType[len(SignalsList)-1] == "std_logic":
                TestbenchFile.write("      " +
SignalsList[len(SignalsList)-1] + ": " + SignalsMode[len(SignalsList)-1]
+ " " + SignalsType[len(SignalsList)-1] + "); \n")
                # if vector
                else:
                    TestbenchFile.write("      " +
SignalsList[len(SignalsList)-1] + ": " + SignalsMode[len(SignalsList)-1]
+ " " + SignalsType[len(SignalsList)-1] + "(" +
SignalsSize[len(SignalsList)-1] + " downto 0); \n")
            # Closign component
TestbenchFile.write("end component; \n\n")
#####
##### Signals definition
#####

# Loop for all signals
for i in range(0, len(SignalsList)):
    # Check if single bit
    if SignalsType[i] == "std_logic":
        TestbenchFile.write("signal " + SignalsList[i] + "_i" + ": " +
SignalsType[i] + "= '0'; \n")
        # if vector
        else:
            TestbenchFile.write("signal " + SignalsList[i] + "_i" + ": " +
SignalsType[i] + "(" + str(SignalsSizeAssigned[i]) + " downto 0) := "
(others => '0'); \n")
        TestbenchFile.write("\n");

#Definition of input/output file
TestbenchFile.write("file file_INPUTS: text; \n")
TestbenchFile.write("file file_OUTPUTS: text; \n\n")
#####

#####
#           Begin architecture
#####

# Architecture definition
TestbenchFile.write("begin \n\n")

#####

```

Figure 7.4: *Script file di testing completo*

7.3. lab6

```
#####
# Component instantiation
#####
# Component instantiation
TestbenchFile.write("DUT: " + EntityName + "\n")
# Check if there are generics in the code
if GenericFlag == 1:
    # Write generics declaration
    TestbenchFile.write("generic map (\n")
    # Check if there is only one generic
    if len(GenericsList) == 1:
        TestbenchFile.write("      " + GenericsList[0] + "=> "
+ str(GenericsAssignedValue[0]) + ")\n")
        # Check if there are more then one generics
    elif len(GenericsList) > 1:
        # Loop on all generics
        for i in range(0,len(GenericsList)):
            # Check if we have reached the last element of the list
            if i != (len(GenericsList)-1):
                TestbenchFile.write("      " +
GenericsList[i] + " => " + str(GenericsAssignedValue[i]) + ",\n")
                # if we have reached the last element
            else:
                TestbenchFile.write("      " +
GenericsList[len(GenericsList)-1] + " => " +
str(GenericsAssignedValue[len(GenericsList)-1]) + ")\n")
    # Write ports declaration
TestbenchFile.write("port map (\n")
for i in range(0,len(SignalsList)):
    # Check if we have reached the last element of the list
    if i != (len(SignalsList)-1):
        TestbenchFile.write("      " + SignalsList[i] + " => "
SignalsList[i] + "_i,\n")
        # if we have reached the last element
    else:
        TestbenchFile.write("      " +
SignalsList[len(SignalsList)-1] + " => " + SignalsList[len(SignalsList)-
1] + "_i);\n\n")
#####
# Generation of input signals
#####

# definition variables for input reading
TestbenchFile.write("TEST: process \n\n")
TestbenchFile.write("variable var_InputLINE: line;\n")
TestbenchFile.write("variable var_OutputLINE: line;\n")
TestbenchFile.write("variable var_TEMP1: std_logic_vector(32-1 DOWNTO
0);\n")
TestbenchFile.write("variable var_TEMP2: std_logic_vector(32-1 DOWNTO
0);\n")
TestbenchFile.write("variable var_TEMP3: std_logic;\n")
```

7.3. lab6

```
TestbenchFile.write("variable var_TEMP4: std_logic_vector(32-1 DOWNTO
0);\n")
TestbenchFile.write("variable var_TEMP5: std_logic;\n")
TestbenchFile.write("variable var_SEP: character;\n\n")

TestbenchFile.write("begin\n\n")

#opening input files in read/write mode
TestbenchFile.write("-- Opening input and output files in read/write
modes\n")
TestbenchFile.write("file_open(file_INPUTS, \"file_INPUTS.txt\",
read_mode);\n")
TestbenchFile.write("file_open(file_OUTPUTS, \"output_data.txt\",
write_mode);\n\n")

TestbenchFile.write("while not endfile(file_INPUTS) loop\n")
#-- read a line from the file\n
TestbenchFile.write("readline(file_INPUTS, var_InputLINE);\n")
#-- get first value from the line\n
TestbenchFile.write("read(var_InputLINE, var_TEMP1);\n")
#-- read the separation character\n
TestbenchFile.write("read(var_InputLINE, var_SEP);\n")
#-- get the second value from the line\n
TestbenchFile.write("read(var_InputLINE, var_TEMP2);\n")
#-- read the separation character\n
TestbenchFile.write("read(var_InputLINE, var_SEP);\n")
#-- get the second value from the line\n
TestbenchFile.write("read(var_InputLINE, var_TEMP3);\n")
#-- repeat this last two instructions for every other data in the
file\n\n"

# Loop for all simulation steps
#for i in range(0,NumberOffSteps):
    # Loop for all signals
for i in range(0,len(SignalsList)):
    # Check if input signal
    if SignalsMode[i] == "IN":
        # Check if single bit
        if SignalsType[i] == "std_logic":
            TestbenchFile.write("    " + SignalsList[i] + "_i <=
var_TEMP3;\n")
            TestbenchFile.write("    " + "wait for " +
DelayBetweenSteps + ";\n\n")
            # Check if not single bit
            elif i==0:
                TestbenchFile.write("    " + SignalsList[i] + "_i <=
signed(var_TEMP1);\n")
            else:
                TestbenchFile.write("    " + SignalsList[i] + "_i <=
signed(var_TEMP2);\n")
        if SignalsMode[i] == "OUT":
            if SignalsType[i] == "signed":
                TestbenchFile.write("var_TEMP4 := "
std_logic_vector("+SignalsList[i]+_i);\n")
                TestbenchFile.write("var_TEMP5 := "
"+SignalsList[i+1]+_i;\n")
TestbenchFile.write("write(var_OutputLINE, var_TEMP4, right, 33);\n")
```

7.3. lab6

```
TestbenchFile.write("write(var_OutputLINE, var_TEMP5, right, 3);\n")
TestbenchFile.write("writeln(file_OUTPUTS, var_OutputLINE);\n")
TestbenchFile.write("end loop; \n\n")
TestbenchFile.write("wait; \n\n")
TestbenchFile.write("end process; \n\n")

#####
#
# Closing architecture
#
#####

TestbenchFile.write("file_close(file_INPUTS);\n")
TestbenchFile.write("file_close(file_OUTPUTS);\n\n")
TestbenchFile.write("end test;")

# Closing testbench file
TestbenchFile.close()

#####
```

Figure 7.5: *Python file di creazione testbench*