Politecnico di Torino

III Facoltà di Ingegneria

Gruppo 21

Sistemi Elettronici a Basso Consumo
01NOHOQ

Relazioni di Laboratorio

Autori:	*Matricole:*
Cuccu Riccardo	s253986
Mastronardi Cristian	s253400
Melibeo Alessio	s253297

10 luglio 2019

Indice

CAPITOLO 1

Power Estimation: Probabilistic Techniques

1.1 Probability and Activity Calculation: Simple Logic Gates

Il primo esercizio di questo LAB richiede di calcolare probabilità ed attività (*Switching Activity*) di porte logiche elementari (INV, AND, OR, XOR, mostrate in Figura 1.1) con approccio "carta e penna".

Figura 1.1: Porte Logiche Elementari

Innanzitutto è opportuno richiamare dei concetti base derivanti dalla teoria delle probabilità:

- **Probabilità**: dato un segnale *g(t)*, la probabilità che esso assuma valore '1' può essere calcolata come la media del segnale stesso su un intervallo di osservazione infinito, come mostrato nell'Equazione 1.1

$$P(g=1) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} g(t)dt \tag{1.1}$$

- **Switching Activity**: dato un segnale *g(t)*, la sua attività è data dal rapporto tra il numero totale di commutazioni (toggle count, T_C) ed il tempo di osservazione (idealmente infinito), come mostrato nell'Equazione 1.2

$$A(g) = \lim_{T\to\infty} \frac{T_C(g)}{T} \tag{1.2}$$

Tuttavia, nei casi pratici, si preferisce stimare tali parametri tramite un approccio basato sull'analisi delle tabelle di verità piuttosto che uno puramente matematico.

Con l'ipotesi che i segnali siano spazialmente e temporalmente incorrelati, e che gli ingressi siano equiprobabili ($P(A=1) = P(B=1) = 1/2$), possiamo calcolare che[1]:

[1]Per una maggiore leggibilità abbiamo sostituito $P(A=1)$ con $P(A_1)$.

INV	
A	Y
0	1
1	0

AND		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

- **INV**
  $P(Y_1) = P(A_0) = 1 - P(A_1) = 1/2$
  $A(Y) = P(Y_{0\to1}) + P(Y_{1\to0}) = P(Y_0)\cdot P(Y_1) + P(Y_1)\cdot P(Y_0) = 2\cdot P(Y_1)\cdot(1-P(Y_1)) = 1/2$
- **AND**
  $P(Y_1) = P(A_1)\cdot P(B_1) = 1/2\cdot 1/2 = 1/4$
  $A(Y) = 2\cdot P(Y_1)\cdot(1-P(Y_1)) = 2\cdot 1/4\cdot 3/4 = 3/8$
- **OR**
  $P(Y_1) = P(A_0)\cdot P(B_1) + P(A_1)\cdot P(B_0) + P(A_1)\cdot P(B_1) = 3\cdot(1/2\cdot 1/2) = 3/4$
  $A(Y) = 2\cdot P(Y_1)\cdot(1-P(Y_1)) = 2\cdot 3/4\cdot 1/4 = 3/8$
- **XOR**
  $P(Y_1) = P(A_0)\cdot P(B_1) + P(A_1)\cdot P(B_0) = 2\cdot(1/2\cdot 1/2) = 1/2$
  $A(Y) = 2\cdot P(Y_1)\cdot(1-P(Y_1)) = 2\cdot 1/2\cdot 1/2 = 1/2$

I risultati ottenuti sono raggruppati in Tabella 1.1.

	P(Y=1)	*A(Y)*
INV	1/2	1/2
AND	1/4	3/8
OR	3/4	3/8
XOR	1/2	1/2

Tabella 1.1: Risultati delle stime “Carta e Penna”

Simulation

Una volta terminata la fase teorica è stato possibile partire con la simulazione circuitale. Il tool utilizzato è *ModelSim* che consente, tra le altre funzionalità, di abilitare la raccolta dati in merito alle commutazioni dei singoli nodi. Nel TestBench utilizzato le UUT (Units Under Test) vengono fornite di ingressi pseudo-casuali prodotti da un *LFSR* (Linear Feedback Shift Register).

Il testo del laboratorio richiedeva 5 diverse simulazioni, differenti solo per la loro durata (e di conseguenza per il toggle-count del CLK), allo scopo di confrontare i risultati ottenuti con quelli stimati con approccio teorico. Analizzando il VHDL sorgente abbiamo notato che

il più piccolo delay presente negli elementi circuitali è di 0.1 ns, quindi una simulazione con risoluzione di 100 ps è sufficiente.

Per velocizzare ed automatizzare il processo, si è fatto uso dello script riportato in Appendice A.1, in cui il comando **power report** è utilizzato per ottenere i valori di Toggle Count, Time At 1, Time At 0 e Time At X utili per la stima delle probabilità. È possibile inoltre estendere il comando con l'utilizzo di **-file** per esportare i risultati in un file di testo.

Le statistiche sul numero di commutazioni delle uscita sono riportate in Tabella 1.2:

$T_C(CK)$	$T_C(INV)$	$T_C(AND)$	$T_C(OR)$	$T_C(XOR)$
20	1	0	4	4
200	43	40	42	44
2000	533	418	352	470
20000	4916	3606	3784	4876
200000	49967	37384	37541	49939

Tabella 1.2: Toggle Count

Dal toggle-count è possibile derivare la Switching Activity delle varie porte (risultati in Tabella 1.3), tenendo presente che:

- **Numero di CLK Cycles**: $N_{CK} = T_C(CK)/2$
- **Switching Activity**: $A_Y = T_C(Y)/N_{CK}$

N_{CK}	$A(INV)$	$A(AND)$	$A(OR)$	$A(XOR)$
10	0.1	0	0.4	0.4
100	0.43	0.40	0.42	0.44
1000	0.533	0.418	0.352	0.470
10000	0.4916	0.3606	0.3784	0.4876
100000	$0.49967 \approx 1/2$	$0.37384 \approx 3/8$	$0.37541 \approx 3/8$	$0.49939 \approx 3/8$

Tabella 1.3: Switching Activity Stimata

Dalla teoria introdotta nell'Equazione 1.2 è noto che per una stima esatta occorre osservare il circuito per un tempo infinito. Questo concetto è chiaramente confermato dai risultati sperimentali riportati in Tabella 1.3; si può infatti notare come aumentando il tempo di osservazione le stime si avvicinino asintoticamente al loro valore teorico.

Un'analisi molto simile può essere condotta sulla $P(Y = 1)$, la Tabella 1.4 riporta i dati ottenuti in merito al tempo totale per cui un dato segnale ha assunto valore '1'.

È allora possibile calcolare la probabilità:

- $P(Y_1) = TimeAt1(Y)/SimulationTime$

Anche in questo caso all'aumentare del tempo di simulazione i risultati sperimentali approcciano quelli teorici calcolati tramite l'Equazione 1.1.

$SimTime$	$T_1(INV)$	$T_1(AND)$	$T_1(OR)$	$T_1(XOR)$
10	7.4	0	7.9	7.9
100	52.4	26	71.9	45.9
1000	470.4	277	786.9	509.9
10000	5078	2448	7402.9	4954.9
100000	50043.4	24976	74945.5	49969.5

Tabella 1.4: Time at 1 (ns)

$SimTime$	$P_1(INV)$	$P_1(AND)$	$P_1(OR)$	$P_1(XOR)$
10	0.74	0	0.79	0.79
100	0.524	0.26	0.719	0.459
1000	0.4704	0.277	0.7869	0.5099
10000	0.5078	0.2448	0.74029	0.49549
100000	$0.500434 \approx 1/2$	$0.24976 \approx 1/4$	$0.749455 \approx 3/4$	$0.499695 \approx 1/2$

Tabella 1.5: Probabilità Stimata

1.2 Probability and Activity Calculation: Half and Full Adder

Il punto di partenza del secondo eserczio è ancora una volta un calcolo "carta e penna"[2] di probabilità ed attività di Half-Adder e Full-Adder partendo dalle loro truth-tables (Tabella 1.6).

Half-Adder

A	B	S	C_O
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full-Adder

A	B	C_i	S	C_O
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabella 1.6: Tabelle di verità di Half-Adder e Full-Adder

- ***Half-Adder***
  - **Probabilità**
    $P(S_1) = P(A_0 \cdot B_1) + P(A_1 \cdot B_0)$
    $P(Co_1) = P(A_1) \cdot P(B_1)$

[2]Per una maggiore leggibilità abbiamo sostituito $P(A_1) \cdot P(B_1)$ con $P(A_1 \cdot B_1)$.

  - **Switching Activity**
    $A(S) = A(Co) = 2 \cdot P_1 \cdot (1 - P_1)$

- ***Full-Adder***
  - **Probabilità**
    $P(S_1) = P(A_0 \cdot B_0 \cdot Ci_1) + P(A_0 \cdot B_1 \cdot Ci_0) + P(A_1 \cdot B_0 \cdot Ci_0) + P(A_1 \cdot B_1 \cdot Ci_1)$
    $P(Co_1) = P(A_0 \cdot B_1 \cdot Ci_1) + P(A_1 \cdot B_0 \cdot Ci_1) + P(A_1 \cdot B_1 \cdot Ci_0) + P(A_1 \cdot B_1 \cdot Ci_1)$
  - **Switching Activity**
    $A(S) = A(Co) = 2 \cdot P_1 \cdot (1 - P_1)$

Di seguito i risultati, assumendo **ingressi equiprobabili** ($P(A_1) = P(B_1) = P(Ci_1) = 1/2$) **ed incorrelati**:

- ***Half-Adder***
  - **Probabilità**
    $P(S_1) = 1/2$
    $P(Co_1) = 1/4$
  - **Switching Activity**
    $A(S) = 1/2$
    $A(Co) = 3/8$

- ***Full-Adder***
  - **Probabilità**
    $P(S_1) = 1/2$
    $P(Co_1) = 1/2$
  - **Switching Activity**
    $A(S) = 1/2$
    $A(Co) = 1/2$

A partire dai precedenti risultati è possibile analizzare un RCA (Ripple Carry Adder, Figura 1.2).

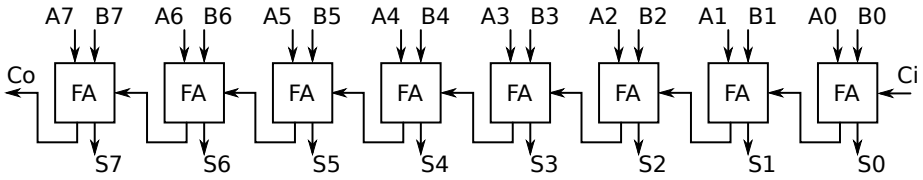

Figura 1.2: 8-bit Ripple Carry Adder

Essendo una configurazione "in cascata", è opportuno prestare attenzione al fatto che la probabilità associata agli ingressi del blocco *i-esimo* dipende dalla probabilità dell'uscita del blocco *(i-1)esimo*; in questo caso specifico il C_{out} di un nodo è connesso al C_{in} del successivo.

Tuttavia abbiamo già calcolato che $P(Co_1) = 1/2$, di conseguenza tutti i blocchi sono equivalenti da un punto di vista probabilistico, in quanto $P(A_1) = P(B_1) = P(Ci_1) = 1/2$ per tutti (Tabella 1.7).

$P(A_1) = P(B_1) = P(Ci_1) = 1/2$								
	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
$A(S)$	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2

Tabella 1.7: Stima della Probabilità di un RCA con **ingressi equiprobabili**

Variando la probabilità associata agli ingressi come suggerito dal testo del laboratorio, ovvero $P(A_1) = 0.4$, $P(B_1) = 0.6$, $P(Ci_1) = 0.5$, i risultati ottenuti utilizzando le medesime formule viste in precedenza sono stati i seguenti:

- ***Full-Adder***
  - **Probabilità**
    $P(S_1) = (0.6 \cdot 0.4 \cdot 0.5) + (0.6 \cdot 0.6 \cdot 0.5) + (0.4 \cdot 0.4 \cdot 0.5) + (0.6 \cdot 0.4 \cdot 0.5) = 1/2$
    $P(Co_1) = (0.6 \cdot 0.6 \cdot 0.5) + (0.4 \cdot 0.4 \cdot 0.5) + (0.4 \cdot 0.6 \cdot 0.5) + (0.4 \cdot 0.6 \cdot 0.5) = 1/2$
  - **Switching Activity**
    $A(S) = 1/2$
    $A(Co) = 1/2$

In conclusione, come mostrato in Tabella 1.8, non c'è alcuna differenza rispetto al caso precedente.

$P(A_1) = 0.4$; $P(B_1) = 0.6$; $P(Ci_1) = 0.5$								
	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
$A(S)$	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2

Tabella 1.8: Stima della Probabilità di un RCA con **ingressi non equiprobabili**

Simulation

Analizzando il codice VHDL abbiamo notato che il minimo delay presente nel circuito è di 0.025 ns, il che rende necessaria una simulazione con risoluzione di 10 ps in questo caso.

tb_rca

Nel file di TestBench *'tb_rca.vhd'* sono istanziati 2 RCA, differenti solo per il ritardo associato alla generazione del Carry-Out[3]:

- **UADDER1**:
  $DRCAS$ = 25ps, $DRCAC$ = 0ps;
- **UADDER2**:
  $DRCAS$ = 25ps, $DRCAC$ = 25ps.

Abbiamo simulato separatamente le due istanze e poi confrontato i risultati in merito al toggle-count.

Confrontando Tabella 1.10 con Tabella 1.9 possiamo notare che per quanto riguarda C_O non ci sono variazioni, mentre i risultati concernenti la generazione dei bit di somma sono molto diversi a causa appunto del differente ritardo associato al carry.

[3] $DRCAS$ è il delay associato alla somma, $DRCAC$ quello associato al carry.

$T_C(CK)$	$T_C(S_7)$	$T_C(S_6)$	$T_C(S_5)$	$T_C(S_4)$	$T_C(S_3)$	$T_C(S_2)$	$T_C(S_1)$	$T_C(S_0)$	C_O
20	3	4	3	1	3	3	2	4	0
200	40	51	52	43	47	52	51	44	52
2000	481	485	500	458	479	500	492	470	638
20000	4958	4948	5007	4794	4979	5022	5000	4876	6107
200000	49926	49804	50088	49131	50062	50007	49924	49939	61633

Tabella 1.9: UADDER1 Power Estimation

$T_C(CK)$	$T_C(S_7)$	$T_C(S_6)$	$T_C(S_5)$	$T_C(S_4)$	$T_C(S_3)$	$T_C(S_2)$	$T_C(S_1)$	$T_C(S_0)$	C_O
20	3	4	3	1	3	3	2	4	0
200	112	101	92	85	99	96	95	44	52
2000	1243	1179	1076	1070	1111	1060	922	470	638
20000	11884	11546	10537	10954	10825	10454	8576	4876	6107
200000	118828	115360	105496	110879	109322	106199	87310	49939	61633

Tabella 1.10: UADDER2 Power Estimation

Per stimare la switching activity utilizzeremo l'Equazione 1.3 riportata di seguito:

$$A(S) = \sum_{i=1}^{N-1} A(S_i) = \sum_{i=1}^{N-1} \frac{T_C(S_i)}{N_{CK}} = 2 \cdot \sum_{i=1}^{N-1} \frac{T_C(S_i)}{T_C(CK)} \tag{1.3}$$

A questo punto è possibile calcolare l'overhead in termini di switching acticity introdotto da UADDER2 rispetto a UADDER1, come mostrato in Tabella 1.11.

$T_C(CK)$	**UADDER1** $A(S)$	**UADDER2** $A(S)$	**Overhead**
20	2.300	2.300	+0.00%
200	3.800	7.240	+90.53%
2000	3.865	8.131	+110.38%
20000	3.958	7.965	+101.24%
200000	3.989	8.033	+101.38%

Tabella 1.11: Switching Activity Overhead di UADDER2 rispetto a UADDER1

In definitiva un $DRCAC$ diverso da zero provoca quasi un **raddoppio della switching activity** relativa alla generazione dei bit di somma.

La ragione principale di questo consistente sbilanciamento risiede nel fenomeno di generazione dei **glitch**. Come sappiamo dalla teoria, se gli ingressi di una porta logica sono temporalmente sbilanciati (ovvero non arrivano simultaneamente) possono verificarsi commutazioni indesiderate. Consideriamo l'esempio in Figura 1.3, dove i ritardi sono stati modellati secondo il modello *Unit Delay*.

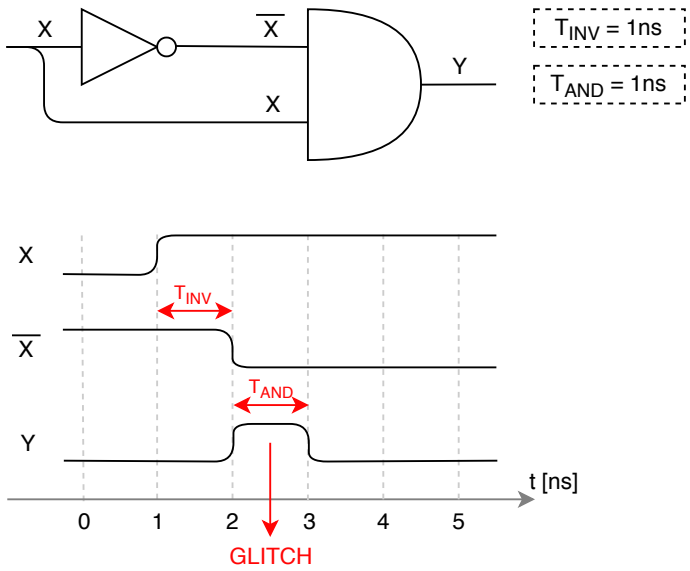

Figura 1.3: Esempio di Glitch con una porta NOT e una porta AND

Dal punto di vista booleano la funzione $Y = X\overline{X}$ dovrebbe sempre essere uguale a '0'. Tuttavia, a causa del delay non-nullo introdotto dall'INVERTER, gli ingressi della porta AND sono temporalmente sbilanciati, ovvero c'è un intervallo tra [1 ns, 2 ns] in cui assumono entrambi valore '1' e questo causa una commutazione '0' $\rightarrow$ '1' sull'uscita della porta AND.

Questa condizione non è particolarmente critica dal punto di vista logico in circuiti sequenziali, in quanto con ogni probabiltà in occorrenza del colpo di clock successivo le commutazioni indesiderate si saranno estinte. D'altra parte dal punto di vista energetico è una condizione che va assolutamente limitata.

Tornando al Ripple Carry Adder abbiamo analizzato i 2 possibili scenari causati dai differenti ritardi dei carry (Figura 1.4).

- **UADDER1** ($DRCAC = 0ps$):
  non ci sono glitch in quanto tutti gli ingressi sono simultanei;
- **UADDER2** ($DRCAC = 25ps$):
  l'ingresso C_{IN} di ogni Full-Adder, eccetto il primo (FA_0), è ritardato rispetto agli altri ingressi A e B, di conseguenza il fenomeno dei glitch diventa considerevole.

Una conferma visiva dei concetti teorici fin qui enunciati si può ottenere confrontando le forme d'onda prodotte dalla contemporanea simulazione per 200 ns di entrambe le istanze del RCA.

La Figura 1.5 mostra un esempio di 5 commutazioni indesiderate sul UADDER2 prima di stabilizzarsi sul risultato corretto.

Un dettaglio sulle singole uscite dei Full-Adder è riportato in Figura 1.6.

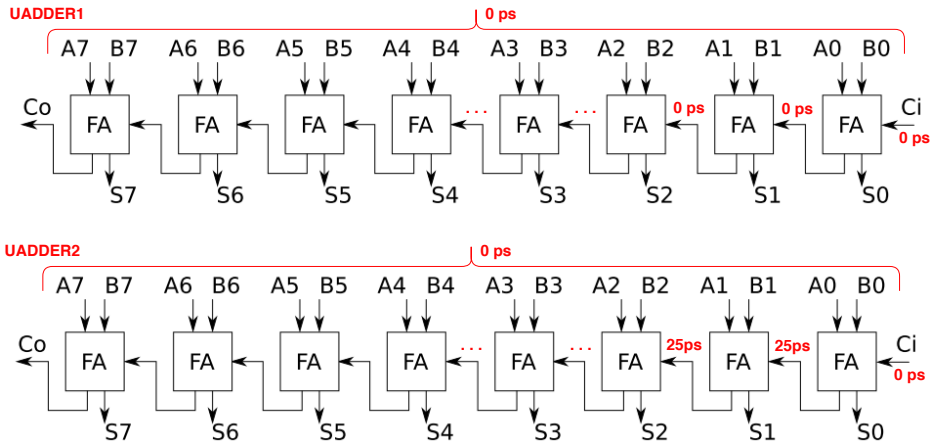

Figura 1.4: Comparazione dei ritardi nel RCA

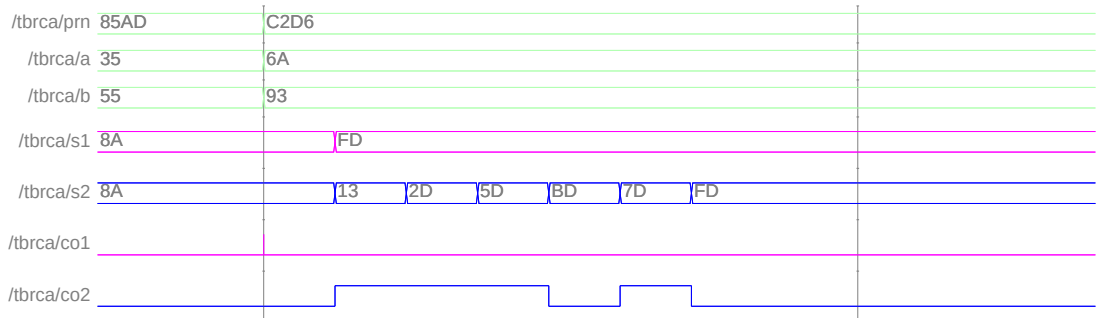

Figura 1.5: RCA-UADDER2 Glitches

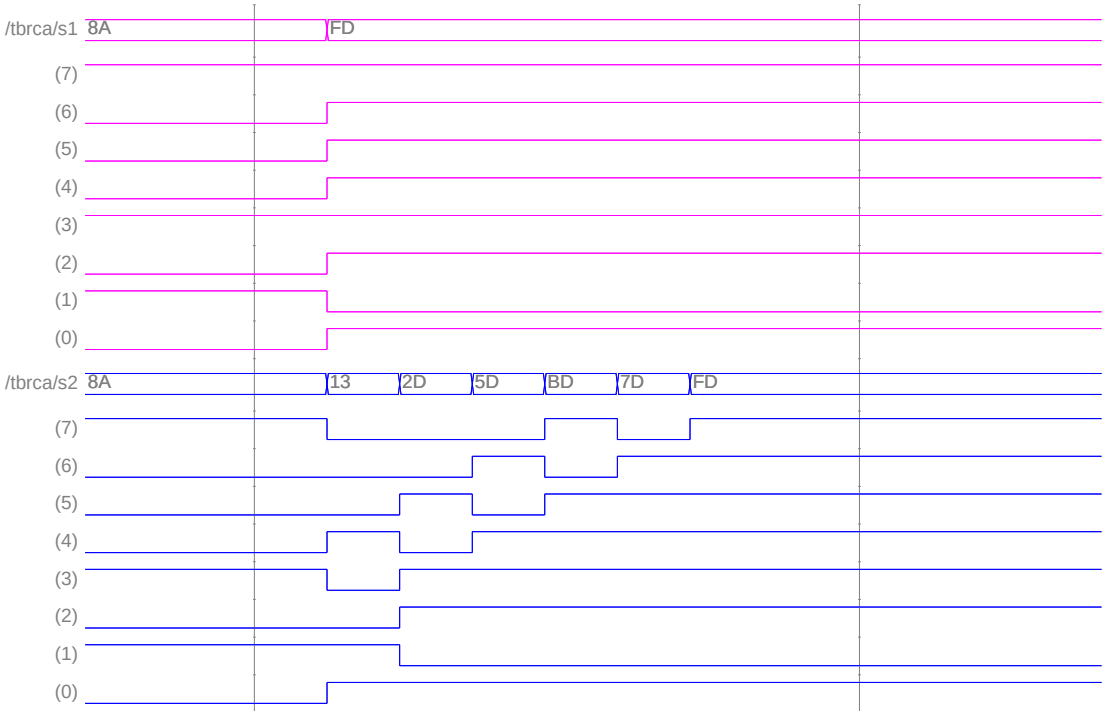

Figura 1.6: RCA-UADDER2 Glitch Detail

Procedendo come in precedenza, possiamo stimare lo switching activity overhead introdotto da UADDER2 rispetto a UADDER1 (Tabella 1.13).

	$T_C(S_7)$	$T_C(S_6)$	$T_C(S_5)$	$T_C(S_4)$	$T_C(S_3)$	$T_C(S_2)$	$T_C(S_1)$	$T_C(S_0)$	C_O
UADDER1	84	102	105	85	98	101	102	92	123
UADDER2	250	242	213	199	210	201	178	92	123

Tabella 1.12: Toggle Count degli output del RCA

UADDER1 $A(S)$	3.8450
UADDER2 $A(S)$	7.9250
Overhead	+106.11%

Tabella 1.13: Switching Activity Overhead di UADDER2 rispetto a UADDER1

tb_rca2

L'ultimo step riguardante il RCA richiedeva di analizzare l'attività di caso peggiore, ovvero la combinazione degli ingressi che causa il maggior numero di commutazioni. Dato che quest'ultimo è molto maggiore se il fenomeno dei glitch diventa rilevante, analizzeremo soltanto il comportamento di UADDER2.

Nel TestBench proposto venivano forzate le seguenti commutazioni degli ingressi[4]:

- C_{IN}: fisso a '0';
- A: $A8 \rightarrow 04$;
- B: $A8 \rightarrow FC$.

I risultati della simulazione di questo caso particolare sono riportati in Tabella 1.14 e Figura 1.7.

	$T_C(S_7)$	$T_C(S_6)$	$T_C(S_5)$	$T_C(S_4)$	$T_C(S_3)$	$T_C(S_2)$	$T_C(S_1)$	$T_C(S_0)$	C_O
UADDER2	6	5	4	3	2	0	0	0	6

Tabella 1.14: UADDER2 Worst Case Activity

[4]Notazione esadecimale

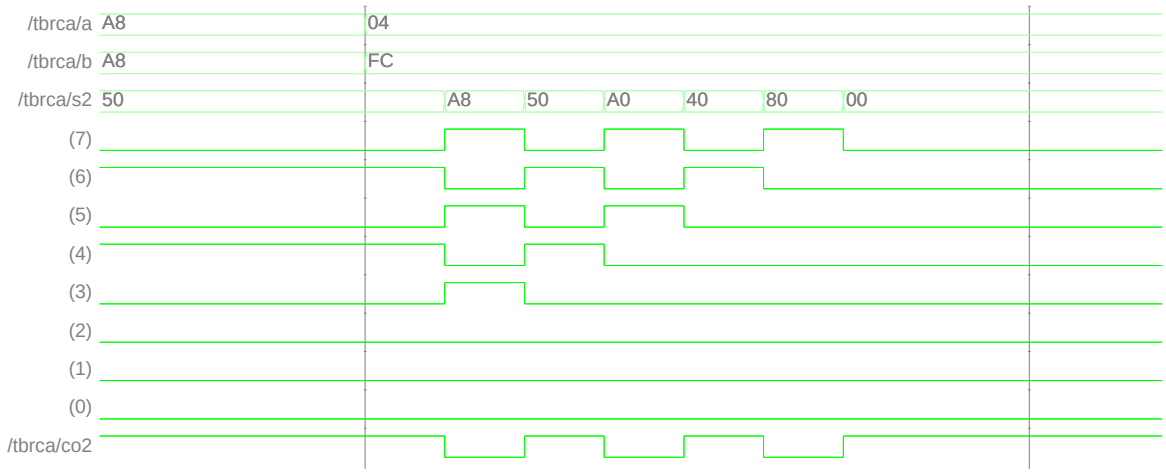

Figura 1.7: UADDER2 Worst Case Activity

Tuttavia, testando configurazioni alternative siamo giunti alla conclusione che esiste una situazione persino peggiore di quella proposta:

- C_{IN}: fisso a '0';
- A: $AA \rightarrow 01$;
- B: $AA \rightarrow FF$.

I risultati della simulazione sono riportati in Tabella 1.15 e Figura 1.8.

	$T_C(S_7)$	$T_C(S_6)$	$T_C(S_5)$	$T_C(S_4)$	$T_C(S_3)$	$T_C(S_2)$	$T_C(S_1)$	$T_C(S_0)$	C_O
UADDER2	8	7	6	5	4	3	2	0	8

Tabella 1.15: UADDER2 Worst Case Activity - New

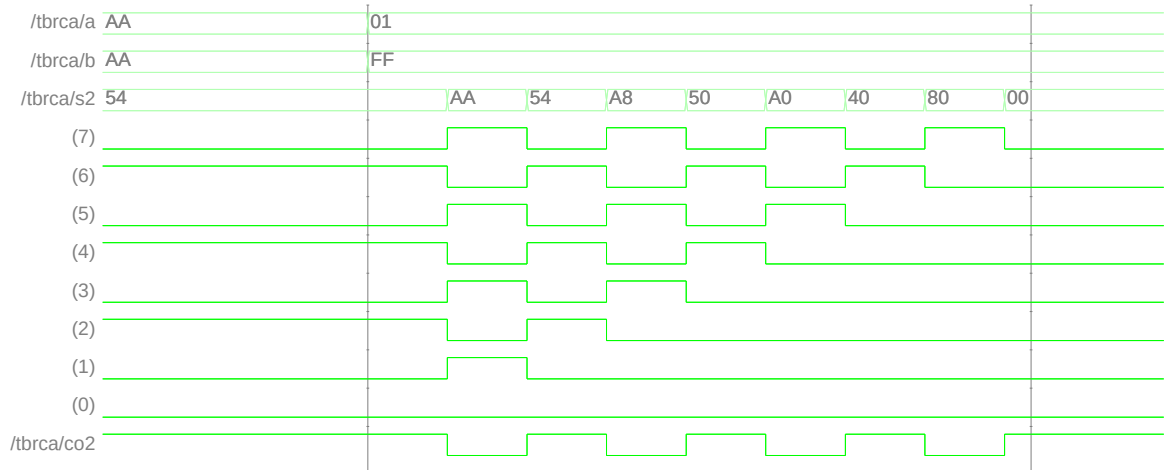

Figura 1.8: UADDER2 Worst Case Activity - New

1.3 RCA Synthesis And Power Analysis

Lo scopo di questa sezione è quello di analizzare la potenza del Ripple Carry Adder dopo la fase di sintesi tramite *Power Compiler* di *Synopsys*.

Nonostante il RCA sia un modulo che non presenta degli elementi di memoria necessita comunque di un clock per essere sintetizzato, questo, se non assegnato, assumerà il valore di default.

Per ottenere il critical path è stato utilizzato il comando **report_timing**, tramite cui, congruentemente a quanto previsto nel testo del laboratorio, il valore ottenuto è stato di 0.78 ns. Per semplicità si è dunque scelto di utilizzare un periodo di clock pari a 1 ns, anche se un valore, ad esempio, di 0.80 ns sarebbe comunque stato sufficiente.

La prima analisi tramite il comando **report_power** ha portato ai contributi di potenza riportati in Tabella 1.16.

Cell Internal Power	16.7429 µW
Net Switching Power	9.7406 µW
Total Dynamic Power	26.4835 µW
Cell Leakage Power	953.4835 nW

Tabella 1.16: Contributi di potenza nel RCA

Successivamente si sono analizzati i contributi dei singoli Full-Adder utilizzando l'opzione **-hier**, la quale permette di analizzare i valori di potenza dei blocchi gerarchicamente inferiori, tali valori sono riportati in Tabella 1.17.

Block	Switching Power	Internal Power	Leakage Power	Total Power	%
FAI_8 (FA_1)	**0.848** uW	2.154 uW	119.291 nW	3.121 uW	11.4
FAI_7 (FA_2)	1.293 uW	2.150 uW	118.962 nW	3.562 uW	13.0
FAI_6 (FA_3)	1.332 uW	2.191 uW	119.340 nW	3.642 uW	13.3
FAI_5 (FA_4)	1.328 uW	2.171 uW	119.110 nW	3.618 uW	13.2
FAI_4 (FA_5)	1.278 uW	2.101 uW	119.294 nW	3.498 uW	12.7
FAI_3 (FA_6)	1.263 uW	2.090 uW	118.979 nW	3.471 uW	12.7
FAI_2 (FA_7)	1.229 uW	2.002 uW	119.508 nW	3.351 uW	12.2
FAI_1 (FA_0)	1.171 uW	1.884 uW	118.999 nW	3.175 uW	11.6

Tabella 1.17: Contributi di potenza dei singoli Full-Adder nel RCA

Si noti che la Switching Power dell'ultimo Full-Adder (FAI_8) è minore rispetto agli altri di circa 1/3, questo è dovuto al fatto che l'uscita C_{out} nel design RTL non è collegata ad alcuna porta logica a differenza dei carry dei FA precedenti, per cui ipotizziamo che Design Compiler utilizzi un carico di default da posizionare sulle uscite lasciate flottanti. Tale supposizione è confermata selezionando come target d'analisi il singolo FA tramite il comando **current_instance FAI_8** e analizzando l'origine dei contributi di potenza delle singole celle che lo compongono tramite **report_power -cell**. In Tabella 1.18 sono messi a confronto i

contributi di potenza di FAI_1 e FAI_8, i quali differiscono principalmente per la Switching Power della cella U2 di circa un ordine di grandezza.

Cell	Cell Internal Power	Driven Net Switching Power	Tot Dynamic Power (% Cell/Tot)	Cell Leakage Power
U1(FAI_1)	0.8205	0.0512	0.872 (94%)	36.0111
U1(FAI_8)	0.9733	0.0624	1.036 (94%)	36.1631
U2(FAI_1)	0.1377	**0.3964**	0.534 (26%)	14.2499
U2(FAI_8)	0.1774	**0.0332**	0.211 (84%)	14.2174
U3(FAI_1)	0.3374	0.1749	0.512 (66%)	32.5747
U3(FAI_8)	0.4278	0.2155	0.643 (67%)	32.7466
U4(FAI_1)	0.5889	0.5488	1.138 (52%)	36.1637
U4(FAI_8)	0.5757	0.5365	1.112 (52%)	36.1637
Total(FAI_1)	1.884 uW	1.171 uW	3.056 uW (62%)	118.999 nW
Total(FAI_8)	2.154 uW	0.848 uW	3.002 uW (72%)	119.291 nW

Tabella 1.18: Origine dei contributi di potenza in FAI_1 e FAI_8

Per entrare maggiormente nel dettaglio, il comando **report_power -net -verbose** permette di analizzare i parametri associati ai singoli nodi per capire come venga stimata la Switching Power; tali parametri sono la capacità associata, la probabilità di commutazione e il numero di commutazioni in un periodo di clock. Come sopra, in Tabella 1.19 è riportato il confronto tra i nodi di FAI_1 e FAI_8, dove vengono evidenziate in grassetto le capacità sui nodi di uscita Co dei due FA e le Switching Power già viste nella tabella precedente.

Net	Total Net Load	Static Probability	Toggle Rate	Switching Power
n1(FAI_1)	4.694	0.493	0.1932	0.5488
n1(FAI_8)	4.694	0.499	0.1889	0.5365
n2(FAI_1)	2.010	0.488	0.1439	0.1749
n2(FAI_8)	2.010	0.484	0.1772	0.2155
S(FAI_1)	0.310	0.507	0.2735	0.0512
S(FAI_8)	0.310	0.497	0.3328	0.0624
Co(FAI_1)	**4.554**	0.512	0.1439	**0.3964**
Co(FAI_8)	**0.310**	0.516	0.1772	**0.0332**

Tabella 1.19: Capacità, probabilità e toggle-rate nei nodi di FAI_1 e FAI_8

Si noti che i valori delle capacità sui nodi S e sul nodo Co del RCA non sono nulli (nonstante siano nodi flottanti nella descrizione RTL) ma valgono 0.310 fF, a causa della capacità di default che il sintetizzatore associa a tali nodi per effettuare l'analisi. Questo spiega il comportamento apparentemente anomalo rinvenuto in Tabella 1.17.

Tornando all'analisi del RCA nel suo insieme lo stesso comando, **report_power -net -verbose**, può essere eseguito per ottenere la medesima analisi riportata in Tabella 1.20.

Net	Net Load	Static Prob.	Toggle Rate	Switching Power	Internal Power	Dynamic Power (% Net/Tot)	Leakage Power
CTMP[7]	4.554	0.507	0.1759	0.4846	0.1683	0.653 (74%)	14.2968
CTMP[6]	4.554	0.504	0.1822	0.5020	0.1743	0.676 (74%)	14.3223
CTMP[5]	4.554	0.505	0.1807	0.4978	0.1729	0.671 (74%)	14.3131
CTMP[4]	4.554	0.493	0.1717	0.4730	0.1643	0.637 (74%)	14.4166
CTMP[3]	4.554	0.500	0.1670	0.4601	0.1598	0.620 (74%)	14.3492
CTMP[2]	4.554	0.498	0.1594	0.4392	0.1526	0.592 (74%)	14.3724
CTMP[1]	4.554	0.512	0.1439	0.3964	0.1377	0.534 (74%)	14.2499
S[7]	0.310	0.497	0.3328	0.0624	0.9733	1.036 (6%)	36.1631
S[5]	0.310	0.500	0.3387	0.0635	0.9916	1.055 (6%)	36.1498
S[6]	0.310	0.496	0.3353	0.0628	0.9868	1.050 (6%)	35.8838
S[4]	0.310	0.493	0.3314	0.0621	0.9747	1.037 (6%)	36.1707
S[3]	0.310	0.506	0.3232	0.0606	0.9492	1.010 (6%)	36.1063
S[2]	0.310	0.494	0.3226	0.0605	0.9521	1.013 (6%)	35.9520
S[1]	0.310	0.504	0.3045	0.0571	0.8921	0.949 (6%)	36.1233
S[0]	0.310	0.507	0.2735	0.0512	0.8205	0.872 (6%)	36.0111
Co	0.310	0.516	0.1772	0.0332	0.1774	0.211 (16%)	14.2174
Total	-	-	-	3.7664 uW	8.848 uW	12.614 uW (30%)	403.098 nW

Tabella 1.20: Capacità, probabilità e toggle-rate nei nodi del RCA

In Appendice A.2 lo script utilizzato per la sintesi e l'analisi del RCA.

1.4 A simple MUX: glitch generation and propagation

Il file **tb_mux21_glitch.vhd** scaricato dal repository contiene al suo interno l'architettura di un multiplexer a due ingressi a singolo bit, rappresentato in Figura 1.9, ed il relativo testbench.

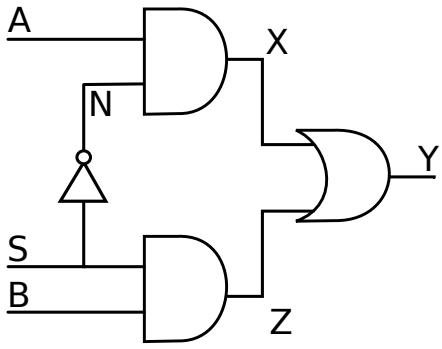

Figura 1.9: Multiplexer a due ingressi a singolo bit

Le porte logiche sono rappresentate da dei processi, tra questi solo il processo PSN, il

quale rappresenta la porta NOT, simula un ritardo, per l'esattezza un ritardo dal valore di 0.1 ns. Tale configurazione va a creare uno sbilanciamento tra i percorsi interni del multiplexer, difatti il ritardo sulla forma negata del selettore S, rappresentata da N, può essere causa di commutazioni spurie (*glitch*) sul nodo X e di conseguenza sull'uscita Y.

Il testbench in oggetto prevede esclusivamente la variazione da '1' a '0' del selettore S dopo $1ns$ dall'avvio della simulazione, metre gli ingressi A e B sono fissati al valore logico '1'. Come anticipato, a causa del ritardo inserito sulla porta NOT, tra 1.0 ns e 1.1 ns sia S che N assumono il valore '0', ciò fa sì che anche Z ed X valgano '0' e di conseguenza la porta OR vedendo in ingresso due valori negativi pilota la sua uscita Y anch'essa al valore logico '0' per 0.1 ns. Tale comportamento è evidenziato in Figura 1.10 in cui l'uscita Y, in magenta, presenta un glitch ('1' → '0' → '1').

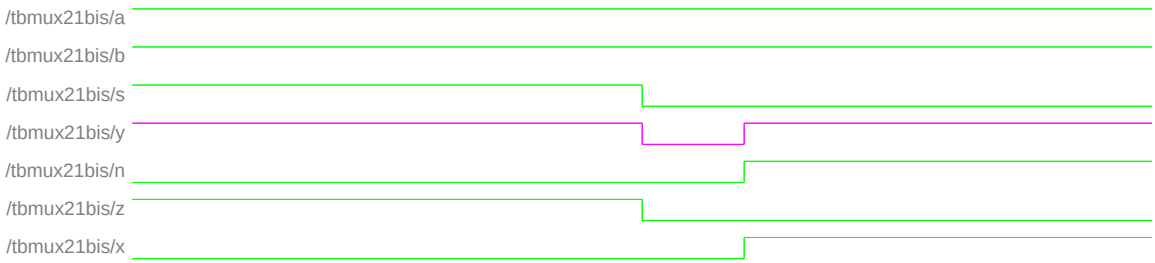

Figura 1.10: Glitch sull'uscita del multiplexer

Analizzando altri scenari abbiamo individuato un'altra possibile situazione in cui si verifica la generazione di un glitch in uscita. L'ingresso B è fissato a '0', mentre sia A che S effettuano una commutazione '0' → '1' al tempo $t = 1$ ns. A causa del ritardo della porta NOT, nell'intervallo tra 1 ns e 1.1 ns sia S che N sono al valore logico '1' e questo fa sì che entrambe le porte AND siano trasparenti ad A e B, causando sull'uscita Y una commutazione ('0' → '1' → '0'). Tale commutazione è indesiderata, in quanto idealmente quando S = '1' dovrebbe verificarsi Y = B, che però è fissato a '0'. Il comportamento descritto è evidenziato dalla simulazione in Figura 1.11

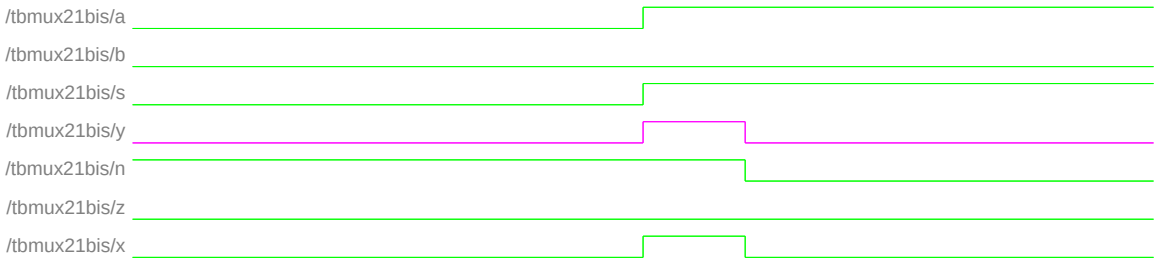

Figura 1.11: Glitch sull'uscita del multiplexer

Ricordiamo, infine, che è importante limitare la switching activity indesiderata (ovvero dovuta ai glitch) dei segnali, in quanto in ogni transizione vengono caricate/scaricate delle capacità. Questo provoca un consumo di potenza indesiderato, linearmente proporzionale alla capacità di carico C e quadraticamente alla tensione di alimentazione V_{dd} .

Nel caso del primo testbench, con la variazione esclusiva del selettore S, l'energia dissipata a causa del glitch è riportata in Equazione 1.4. Tale E_{glitch} è data dalla somma dell'energia dissipata sulla capacità di carico dall'NMOS di pull-down durante la prima transizione '1' → '0' e dell'energia per caricarla nuovamente attraverso il PMOS di pull-up durante la seconda transizione '0' → '1'.

$$E_{glitch} = 2 \cdot \left(\frac{1}{2} \cdot C \cdot V_{dd}^2 \right) \tag{1.4}$$

L'energia dissipata nel secondo caso considerato è la medesima ma per il procedimento inverso, vi è prima la carica della capacità di carico immediatamente seguita dalla sua scarica.

Il file VHDL utilizzato è stato lievemente modificato, e per completezza è riportato in Appendice A.3.

1.5 Probability and Activity Calculation: Syncronous Counter

L'architettura in Figura 1.12 rappresenta un counter a singolo bit, utilizzabile anche come divisore di frequenza, formato da un Half-Adder ed un Flip-Flop D.

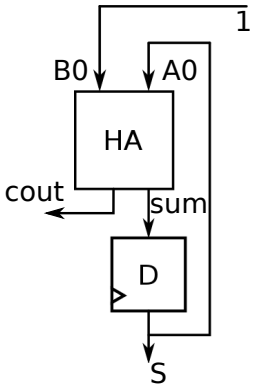

Figura 1.12: Counter

In Figura 1.13 è riportato l'andamento del nodo *sum* e dell'uscita S, la quale attraverso un feedback è riutilizzata come ingresso A_0 dell'Half-Adder, in un primo momento con l'ingesso B_0 (*counten* in figura) fissato al valore '1' e successivamente al valore '0'.

Collegando in cascata diverse unità come quelle appena analizzate è possibile ottenere il contatore sincrono riportato in Figura 1.14, questo, nel nostro caso, è formato da 8 counter a singolo bit e permette di contare fino a $2^8 - 1 = 255$.

L'ingresso B_0 sarà collegato ad un segnale di enable esterno (CEN), gli ingressi B_{i+1} saranno invece collegati alle uscite $Cout_i$, ovvero i carry out del blocco precedente. $Cout_8$ corrisponderà infine al segnale di overflow $OWFL$ il quale può essere interpretato come segnale di notifica di fine conteggio. Quando tutti i bit S_i valgono '1' (conteggio = 255), infatti, un ulteriore incremento porterebbe ad un conteggio di 256, non rappresentabile su

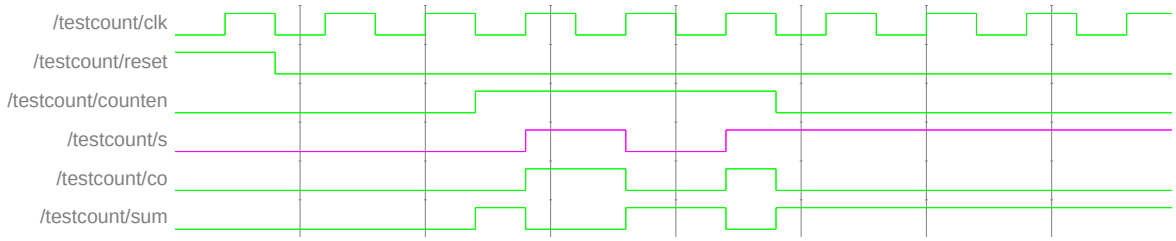

Figura 1.13: Forme d'onda del counter a singolo bit

8-bit; in questa situazione l'ultimo *Cout* passa ad '1' mentre gli 8-bit del counter tornano tutti a '0' (*wrap-around*) ed il conteggio ricomincia.

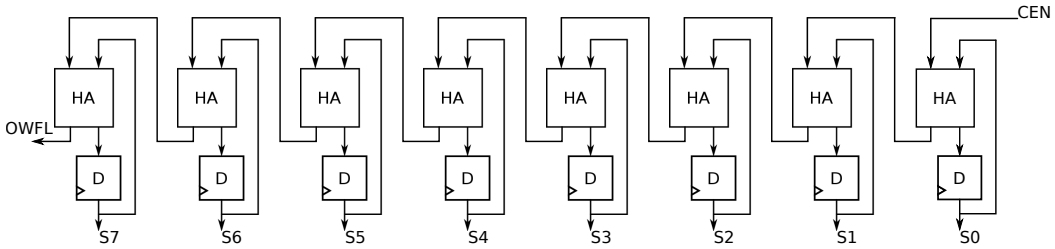

Figura 1.14: Synchronous Counter

Dopo aver descritto il funzionamento del counter possiamo provare ad immaginare quale possa essere l'attività associata ai singoli nodi. I risultati calcolati con approccio manuale, considerando 256 colpi di clock (incluso il wrap-around da '11111111' a '00000000'), sono riportati in Tabella 1.21.

Node	Activity
CLK	512
S0	256
S1	128
S2	64
S3	32
S4	16
S5	8
S6	4
S7	2
OWFL	2

Tabella 1.21: Counter Switching Activity - Pen and Paper

Simulando il circuito con *ModelSim* abbiamo cercato di verificare i risultati teorici appena ottenuti. I risultati della simulazione sui nodi di uscita hanno mostrato un'attività differente da quella aspettata soltanto per quanto riguarda il nodo di $OWFL$. Esso è infatti su un percorso puramente combinatorio, pertanto non è immune al fenomeno dei glitch che inevi-

tabilmente si può verificare a causa dello sbilanciamento dei ritardi dei percorsi interni del counter. Un dettaglio di questa situazione è mostrato in Figura 1.15.

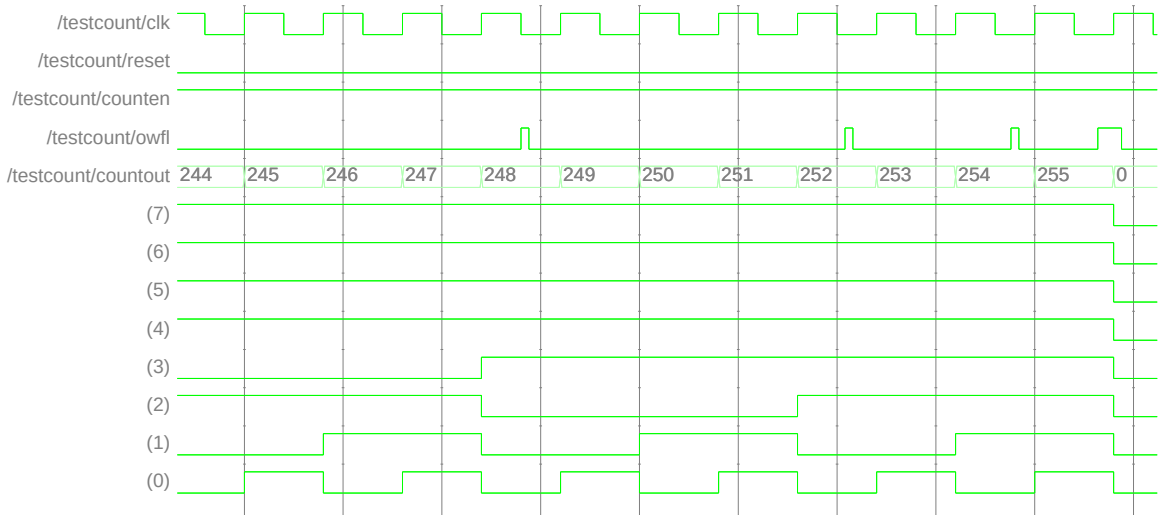

Figura 1.15: Glitch sul segnale OWFL

Nello specifico il nodo $OWFL$ è stato interessato da 16 commutazioni, corrispondenti ad 8 transizioni '1' → '0' → '1'. Oltre che per la potenza inutile dissipata questo può essere un problema da un punto di vista di correttezza logica del counter stesso. Infatti un eventuale modulo a valle che riceva in ingresso il segnale $OWFL$ potrebbe erroneamente considerare finito il conteggio.

Analizzando le forme d'onda abbiamo notato, però, che i glitch si estinguono sempre prima del successivo fronte di salita del CLK; di conseguenza per eliminare le commutazioni indesiderate abbiamo pensato alla semplice introduzione di un D-FF in uscita (Figura 1.16).

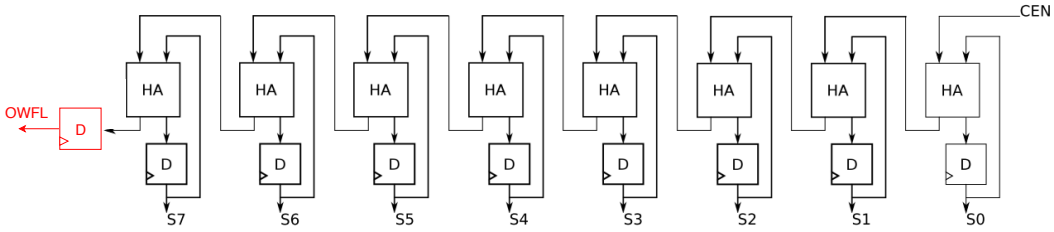

Figura 1.16: Soluzione Glitch OWFL

Come mostrato in Figura 1.17, il segnale $OWFL$ commuta correttamente da '0' ad '1' soltanto nel momento in cui il conteggio passa da 255 a 0.

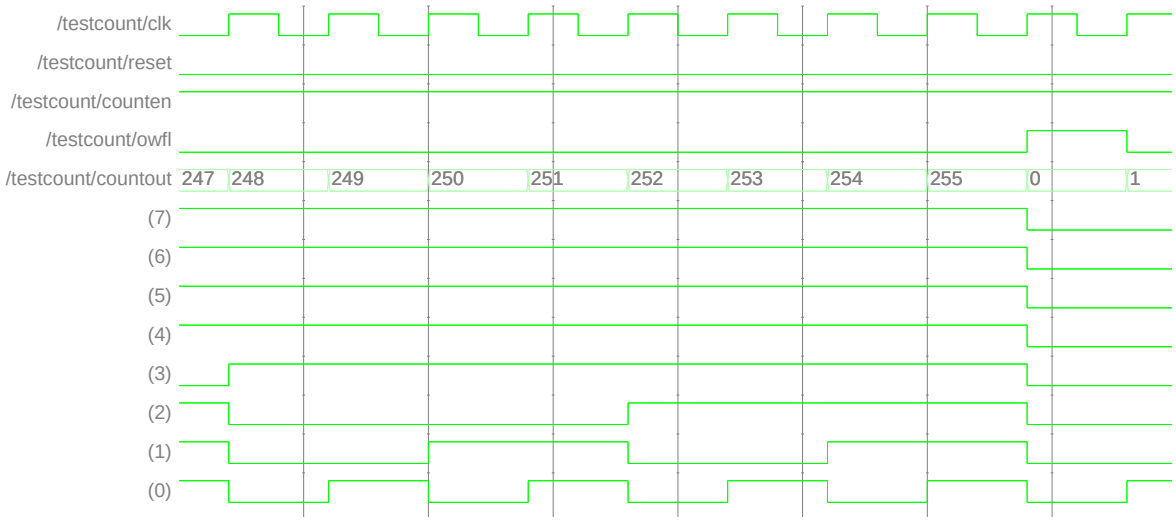

Figura 1.17: Soluzione Glitch OWFL

La soluzione proposta garantisce la correttezza logica, ma non incide particolarmente sulla switching activity del circuito in quanto i glitch relativi al segnale $OWFL$ sono di molto inferiori a quelli relativi alle uscite dei singoli Half-Adder, come si può notare dalla Tabella 1.22. Gli HA infatti sono moduli puramente combinatori collegati in una configurazione a cascata, pertanto ogni glitch può propagarsi nella catena e generarne degli altri.

Net	Toggle Count	Net	Toggle Count	Net	Toggle Count
CLK	520	S[7]	2	sum[7]	30
C[7]	16	S[6]	4	sum[6]	52
C[6]	28	S[5]	8	sum[5]	88
C[5]	48	S[4]	16	sum[4]	144
C[4]	80	S[3]	32	sum[3]	224
C[3]	128	S[2]	64	sum[2]	320
C[2]	192	S[1]	128	sum[1]	385
C[1]	256	S[0]	257	sum[0]	258
C[0]	257	OWFL	2		

Tabella 1.22: Switching Activity dopo la modifica sul nodo OWFL

Un possibile sviluppo futuro per limitare anche la propagazione dei glitch sulle uscite degli HA potrebbe essere quello di adottare una soluzione pipelinata.

In ultima analisi, abbiamo simulato il circuito ad una frequenza maggiore sostituendo il CLK da 2 ns con uno da 0.8 ns. In questa situazione il counter non funziona più correttamente (Figura 1.18), in quanto i Flip-Flop campionano troppo velocemente i bit di somma prodotti dagli HA, non dando sufficiente tempo ai glitch di estinguersi.

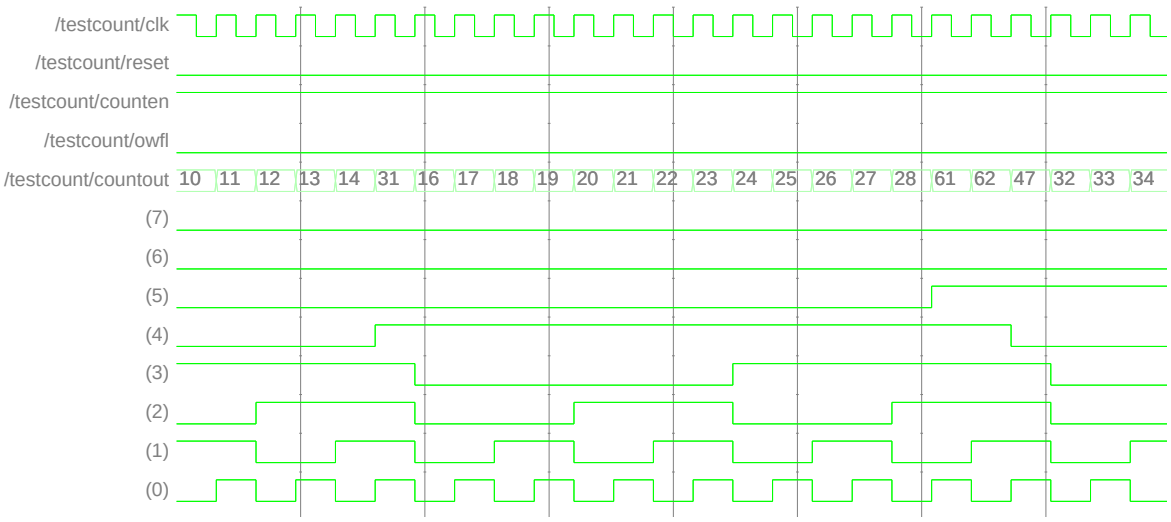

Figura 1.18: Counter a frequenza maggiore

Lo script utilizzato è riportato in Appendice A.4, i risultati dell'analisi di potenza per il circuito originale e quello modificato con l'utilizzo del Flip-Flop D sono riportati nella loro interezza rispettivamente in Appendice A.5 e Appendice A.6.

CAPITOLO 2

FSM State Assignment and VHDL Synthesis

2.1 FSM State Assignment

L'obiettivo di questa prova di laboratorio è stato quello di minimizzare il consumo di potenza di una FSM, lavorando soprattutto sulla logica di assegnazione degli stati della stessa. I risultati ottenuti durante la prima parte dell'esperienza sono stati poi successivamente utilizzati per la sintesi del circuito tramite *Synopsys*.

Un fattore fondamentale da tenere in considerazione è la probabilità di transizione e minimizzare la distanza di Hamming tra gli stati non è sempre la scelta migliore. I vari stati della FSM sono descritti da un grafo di transizione, ottimizzabile attraverso la minimizzazione di una specifica funzione costo riportata in Equazione 2.1 e legata alla struttura stessa del grafo.

$$\gamma = \sum_{over\ all\ edges} p_{ij}\, H(S_i, S_j) \tag{2.1}$$

Il segmento indicato dalla coppia (i, j) rappresenta una transizione dallo stato i allo stato j, mentre, p_{ij} rappresenta la probabilità della stessa. In letteratura sono stati proposti sofisticati algoritmi, gestiti da strumenti CAD, ma, come primo approccio, risulta sempre conveniente utilizzare metodi basati sul know-how ed il buon senso del designer.

In Figura 2.1 è riportato il circuito implementato connettendo gli input ai multiplexer di ingresso ed inserendo dei feedback affinchè il sistema sia in grado di eseguire con il minor consumo di potenza, la somma di 6 numeri in ingresso.

Osservando tale configurazione degli ingressi si può notare come sia stata applicato il codice Gray ai selettori del multiplexer.

In Figura 2.2 è riportato il diagramma degli stati (STG) corrispondente alla FSM progettata, in cui è stato sfruttato nuovamente il codice Gray per l'assegnazione dei vari stati.

Inoltre si è fatto in modo che la somma parziale fosse collegata nei primi tre stati al primo multiplexer e per i restanti due al secondo multiplexer; in questo modo nel caso vi fosse la somma di uno zero, la somma parziale causerebbe delle comutazioni sull'ingresso dell'adder

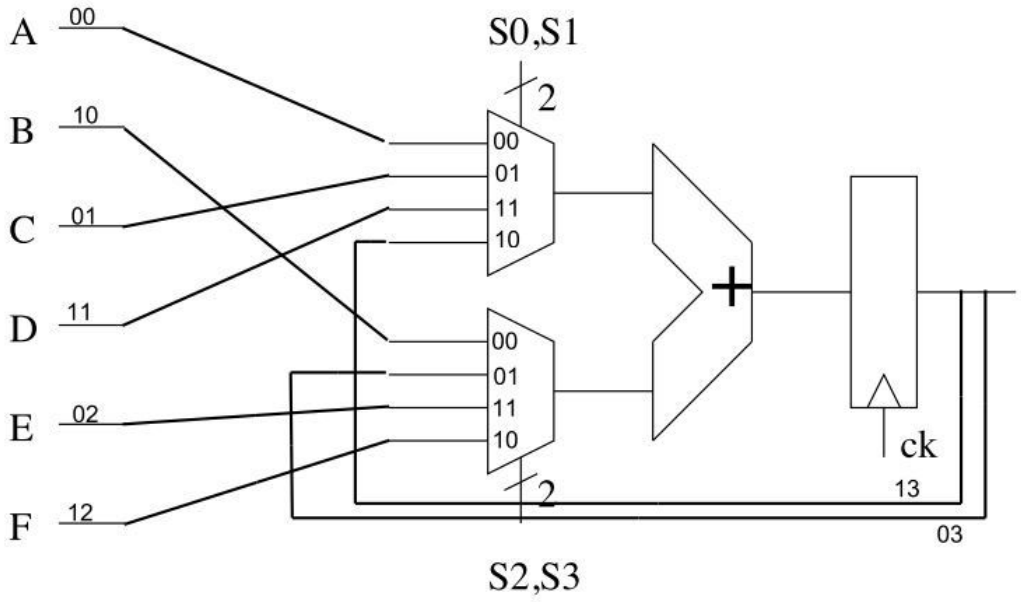

Figura 2.1: Connessione degli ingressi del circuito sommatore per la minimizzazione dello switching nei selettori dei **multiplexer** e negli ingressi dell'**adder**

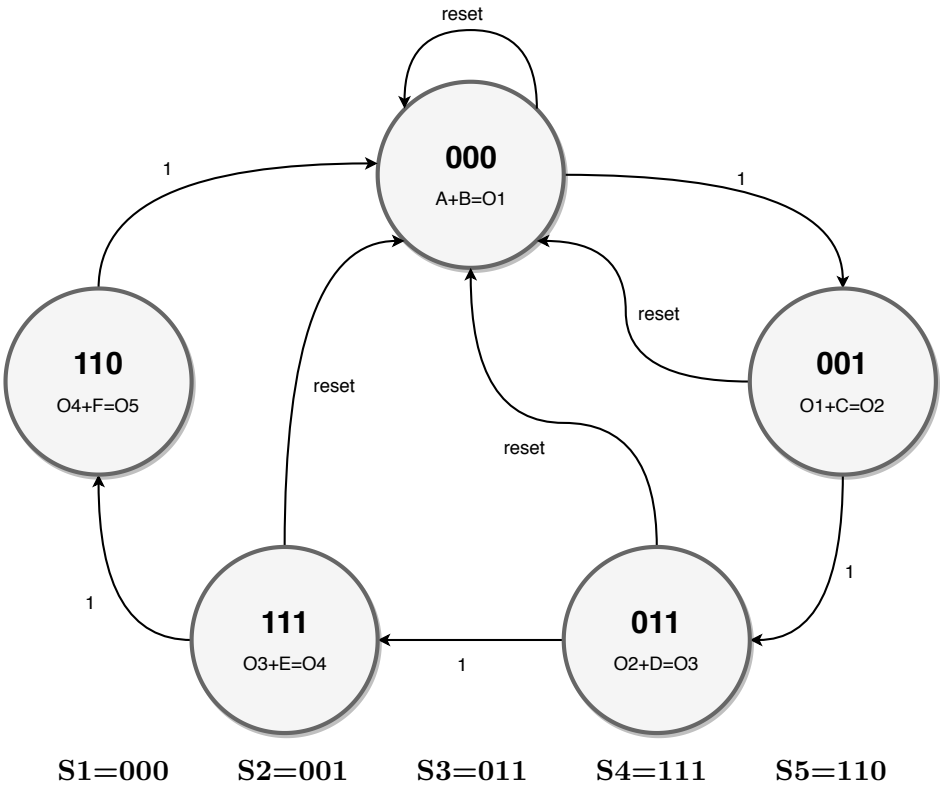

Figura 2.2: STG della FSM progettata utilizzando il codice Gray

a cui è collegata solo nei passaggi $S3 \rightarrow S4$ e $S5 \rightarrow S1$ ma non nei rimanenti casi.

Dopo aver editato il codice VHDL relativo alla FSM, riportato in Appendice B.1, ed aver inserito in un'unica entity sia quest'ultima che il circuito sommatore fornito, si è passati alla simulazione del codice in ambiente *Modelsim*. Attraverso l'utilizzo degli script riportati in Appendice B.2 e Appendice B.3, in grado di richiamare specifiche funzioni del tool, si è passati dapprima all'analisi di tipo comportamentale per poi generare dei report relativi alla potenza dissipata da parte dei vari moduli. Seppur prodotti dall'analisi di un circuito non sintetizzato, questi risultati sono stati utili per poter comprendere in modo qualitativo determinate caratteristiche del circuito progettato.

In Figura 2.3 sono riportate le forme d'onda ottenute per il DUT (Device Under Test), ove è possibile osservare che sia l'operazione di somma tra i sei numeri in ingresso che l'assegnazione degli stati della FSM vengono eseguite correttamente.

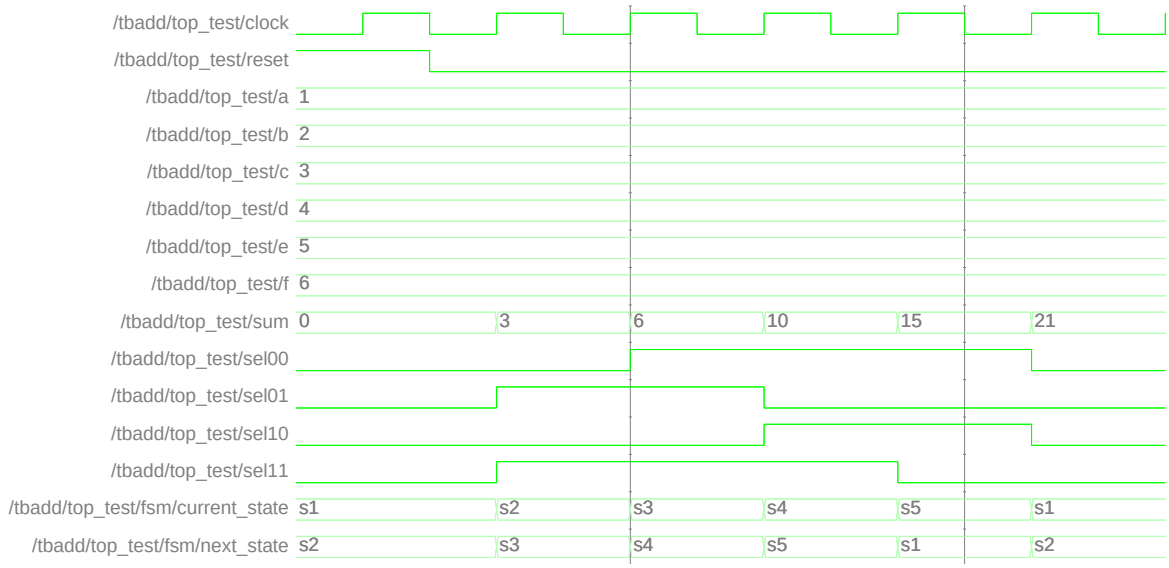

Figura 2.3: Estratto di un ciclo completo della simulazione della FSM

L'analisi di potenza, ha prodotto per la top entity, i risultati mostrati in Tabella 2.1.

Node	Tc	Ti	Time at 1	Time at 0
tbadd/clk	13	0	3000 ps	3500 ps
tbadd/reset	1	0	1000 ps	5500 ps
tbadd/sum(4)	2	0	1000 ps	5500 ps
tbadd/sum(3)	2	0	2000 ps	4500 ps
tbadd/sum(2)	4	0	3000 ps	3500 ps
tbadd/sum(1)	3	0	4000 ps	2500 ps
tbadd/sum(0)	3	0	3000 ps	3500 ps

Tabella 2.1: Risultati del power report per la top entity

Dall'osservazione dei risultati, si può notare come il segnale di clock sia quello con un Tc più alto, a differenza del segnale di reset, attivo esclusivamente allo start-up del sistema, e dei segnali di somma, dove, si può osservare un più elevato Tc sui bits meno significativi.

In Tabella 2.2 sono riportati inoltre i risultati relativi al datapath della FSM.

Node	Tc	Ti	Time at 1	Time at 0
tbadd/top_test/datapath/mux03(4)	2	0	1000 ps	5500 ps
tbadd/top_test/datapath/mux03(3)	2	0	2000 ps	4500 ps
tbadd/top_test/datapath/mux03(2)	4	0	3000 ps	3500 ps
tbadd/top_test/datapath/mux03(1)	3	0	4000 ps	2500 ps
tbadd/top_test/datapath/mux03(0)	3	0	3000 ps	3500 ps
tbadd/top_test/datapath/mux11(4)	2	0	1000 ps	5500 ps
tbadd/top_test/datapath/mux11(3)	2	0	2000 ps	4500 ps
tbadd/top_test/datapath/mux11(2)	4	0	3000 ps	3500 ps
tbadd/top_test/datapath/mux11(1)	3	0	4000 ps	2500 ps
tbadd/top_test/datapath/mux11(0)	3	0	3000 ps	3500 ps
tbadd/top_test/datapath/clock	13	0	3000 ps	3500 ps
tbadd/top_test/datapath/reset	1	0	1000 ps	5500 ps
tbadd/top_test/datapath/sel00	2	0	3000 ps	3500 ps
tbadd/top_test/datapath/sel01	3	0	2000 ps	4500 ps
tbadd/top_test/datapath/sel10	2	0	2000 ps	4500 ps
tbadd/top_test/datapath/sel11	3	0	3000 ps	3500 ps
tbadd/top_test/datapath/sum(4)	2	0	1000 ps	5500 ps
tbadd/top_test/datapath/sum(3)	2	0	2000 ps	4500 ps
tbadd/top_test/datapath/sum(2)	4	0	3000 ps	3500 ps
tbadd/top_test/datapath/sum(1)	3	0	4000 ps	2500 ps
tbadd/top_test/datapath/sum(0)	3	0	3000 ps	3500 ps
tbadd/top_test/datapath/operanda(3)	2	0	2000 ps	4500 ps
tbadd/top_test/datapath/operanda(2)	4	0	2000 ps	4500 ps
tbadd/top_test/datapath/operanda(1)	5	0	3000 ps	3500 ps
tbadd/top_test/datapath/operanda(0)	2	0	4500 ps	2000 ps
tbadd/top_test/datapath/operandb(2)	2	0	3000 ps	3500 ps
tbadd/top_test/datapath/operandb(1)	2	0	5500 ps	1000 ps
tbadd/top_test/datapath/operandb(0)	5	0	2000 ps	4500 ps

Tabella 2.2: Risultati del power report per il data path

Dall'osservazione di questo report appare chiaro che, anche in questo caso, il segnale di clock, è quello con un numero maggiore di toggle (essendo caratterizzato da una E_{SW} pari a 2). Inoltre, il numero di Tc risulta essere abbastanza omogeneo a livello di assegnazione degli stati della FSM e all'interno del datapath.

2.2 VHDL Synthesis

Dopo aver lanciato lo script contenente i comandi per analizzare ed elaborare i file VHDL relativi alla top entity, il tool *Synopsys* ha restituito una prima versione, riportata in Figura 2.4 del circuito finale, basata sul mapping a livello di porte logiche standard, senza alcun riferimento alla libreria che dovrà essere utilizzata (0.045 µm).

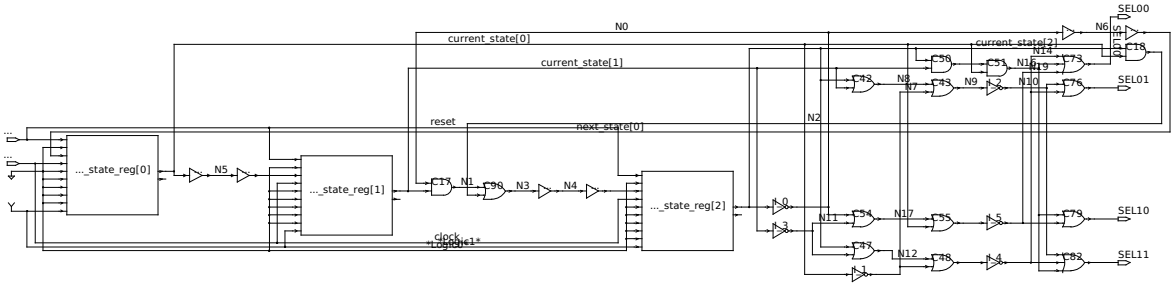

Figura 2.4: FSM pre-sintesi

Successivamente, è stato generato un segnale di clock da sostituire a quello di default, mediante il comando **create clock -name "CLK" -period 10 clock** e ne è stata testata la correttezza lanciando il comando **report_clock** che ha restituito un breve report composto da una sola riga, dove si è potuto verificare il periodo del clock settato, pari a 10 ns.

2.2.1 Synthesis of your structural fsm-adder

Mediante il comando **compile_design**, inserito successivamente all'interno dello stesso script (riportato in Appendice B.4), è stato possibile ottenere il circuito sintetizzato riportato in Figura 2.5.

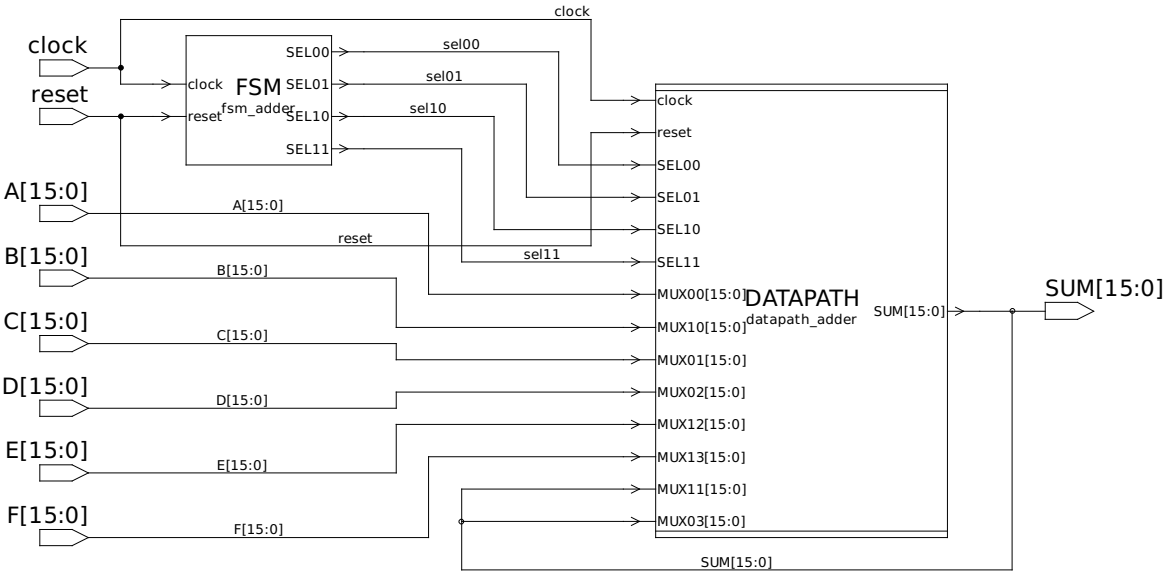

Figura 2.5: Schematico post-sintesi relativo alla top entity

Accedendo al livello inferiore del modulo relativo alla FSM, è stato inoltre possibile visualizzare lo schematico riportato in Figura 2.6, realizzato utilizzando e mappando i logic gate relativi alla libreria di riferimento ed inoltre è stato fatto generare a *Synopsys* il codice VHDL della struttura sintetizzata, utile per le backannotation analysis e riportato in Appendice B.5.

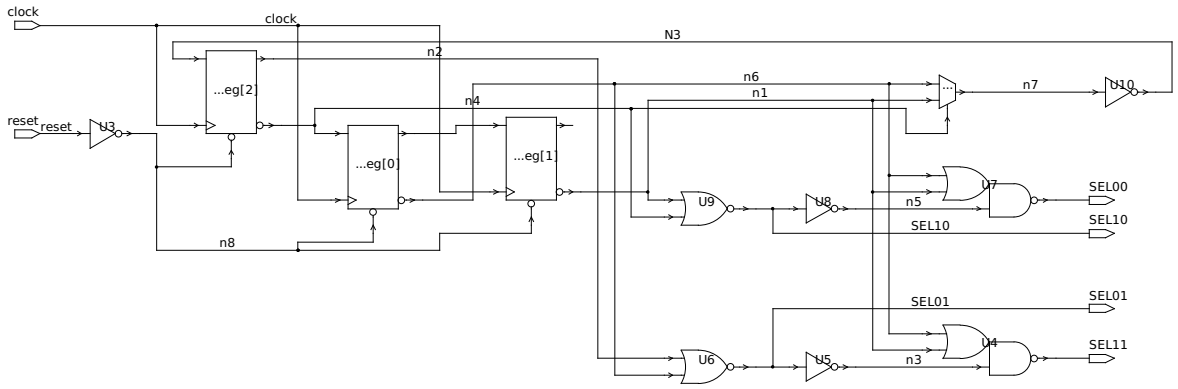

Figura 2.6: FSM post-sintesi

Dopo aver sintetizzato il circuito, ci si è focalizzati sull'analisi dell'area occupata, del timing e della potenza dissipata.

Come operazione preliminare, è stato utilizzato il comando **current_instance FSM**, in modo da poter verificare la correttezza della codifica degli stati all'interno della FSM, basandosi sulla analisi del relativo report, ottenuto mediante la direttiva **report_FSM**, funzionante solo nel caso in cui vengano utilizzate le "virtual STATE definitions", ovvero il vettore di stato. In Tabella 2.3 viene mostrata l'effettiva assegnazione degli stati.

States order	State encodings
S1	000
S2	001
S3	011
S4	111
S5	110

Tabella 2.3: Codifica degli stati mediante codice di Gray

Sfruttando il comando **report_area**, è stato possibile ricavare il numero di celle, porte logiche e connessioni generate e/o utilizzate dal sintetizzatore in questo design, i cui valori sono riportati in Tabella 2.4

Successivamente son stati ottenuti i report relativi al timing del **worst critical path** e dei 10 peggiori rami, mediante la direttiva **report_timing -nworst 10**. In Tabella 2.5 vengono presentati i risultati ottenuti per il worst critical path, mentre, per i 10 peggiori, non si riportano i risultati ottenuti poichè, sostanzialmente, il timing dei vari branch era pressochè lo stesso, con la sola eccezione di 3 rami dove il tempo si è visto essere inferiore di circa l'1% rispetto al WCP, con uno slack medio di circa 8.03 ns.

Reference	Units
Number of ports	114
Number of nets	118
Number of cells	2
Number of combinational cells	0
Number of sequential cells	0
Number of macros	0
Number of buf/inv	0
Number of references	2
Combinational area:	196.04
Noncombinational area:	101.08
Total cell area:	297.12

Tabella 2.4: Area report per la top entity

Point	Incr	Path
current_state_reg[0]/CK (DFFR_X1)	0.00	0.00 r
current_state_reg[0]/QN (DFFR_X1)	0.08	0.08 f
U6/ZN (NOR2_X1)	0.05	0.13 r
U5/ZN (INV_X1)	0.02	0.15 f
U4/ZN (OAI21_X1)	0.03	0.18 r
SEL11 (out)	0.00	0.18 r
data arrival time		0.18

Tabella 2.5: Analisi del cammino critico

Un'utile funzione del tool *Synopsys*, richiamata tramite il comando **endpoint_slack**, ha permesso di ottenere una rappresentazione grafica, riportata in Figura 2.7, della distribuzione dei cammini di timing con differenti slack, partendo dai percorsi più critici (con slack minore) e proseguendo fino ai percorsi meno impegnativi in termini di tempo di percorrenza del segnale.

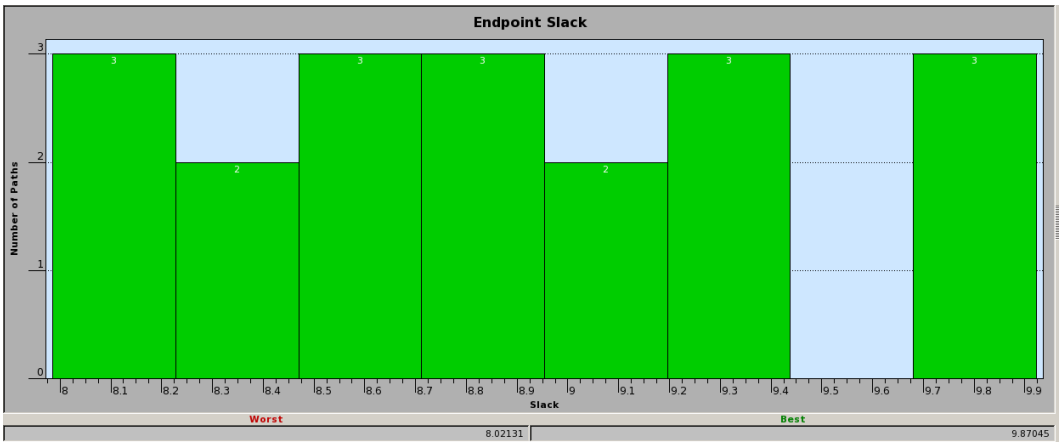

Figura 2.7: Distribuzione del timing

Terminate le analisi di area occupata e timing, un altro parametro fondamentale da misurare ed analizzare è stato il consumo di potenza del circuito sintetizzato.

Lanciando il comando **report_power** sono stati ottenuti dati relativi al consumo di potenza globale, tali dati sono riportati in Tabella 2.6.

Power Group	Switching Power	Internal Power	Leakage Power	Total Power	%
io_pad	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
memory	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
black_box	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
clock_network	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
register	18.0919 µW	1.7048 µW	1.6266×10^3 nW	21.4232 µW	43.19
sequential	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
combinational	12.1454 µW	12.1143 µW	3.9245×10^3 nW	125.2228 µW	56.81
total	30.2372 µW	13.8191 µW	5.5511×10^3 nW	49.6074 µW	100

Tabella 2.6: Contributi di potenza relativi all'intero sistema

Dall'osservazione della tabella, è possibile notare la presenza di 3 contributi di potenza, i quali concorrono a determinare il consumo globale del circuito.

Il primo contributo è dato dalla **Internal Power**, dovuta alle correnti di corto circuito che scorrono per brevi istanti di tempo sul percorso elettrico che connette l'alimentazione a ground. Questo è dovuto al fatto che, durante lo switching, ci sono brevi istanti nei quali sia gli nMOS che i pMOS sono in conduzione.

Il secondo contributo è dovuto alla **Switching Power**, ed è legato alla carica e scarica delle capacità pilotate dai vari gate, quando questi stanno lavorando. Questi due contributi,

se sommati, concorrono a definire la **Total Dynamic Power** che, per il nostro circuito, risultata dunque essere pari a 44.0563 µW.

Infine, l'ultimo contributo è dato dalla **Leakage Power**, potenza dissipata a causa del fenomeno della conduzione sotto soglia e delle correnti inverse di saturazione, circolanti attraverso le giunzioni del circuito integrato, ed impossibili da bloccare del tutto essendo le porte logiche alimentate anche quando non sono chiamate ad operare.

Successivamente, mediante l'uso di comandi quali **report_power -net** e **report_power-cell** è stato possibile ottenere dati specifici riguardo le varie reti di collegamento tra le porte logiche e riguardo la potenza dissipata da queste ultime. Inoltre è stato possibile selezionare il design da cui ottenere i vari dati e, di conseguenza, si è ritenuto opportuno analizzare in primis la top entity e successivamente il blocco relativo alla FSM.

In Tabella 2.7 vengono presentati i dati relativi alle net della top entity.

Net	Total net load	Static Prob.	Toggle Rate	Switching Power
sel10	12.346	0.153	0.0313	0.2338 µW
sel01	9.301	0.355	0.0382	0.2151 µW
sel00	10.169	0.321	0.0345	0.2120 µW
sel11	7.112	0.355	0.0382	0.1644 µW
SUM[12]	4.314	0.253	0.0289	0.0755 µW
SUM[4]	4.314	0.251	0.0289	0.0753 µW
SUM[7]	4.314	0.254	0.0288	0.0752 µW
SUM[1]	4.314	0.250	0.0287	0.0750 µW
SUM[8]	4.314	0.251	0.0287	0.0748 µW
SUM[15]	4.314	0.255	0.0286	0.0747 µW
SUM[2]	4.314	0.255	0.0286	0.0746 µW
SUM[6]	4.314	0.257	0.0285	0.0744 µW
SUM[9]	4.314	0.254	0.0285	0.0744 µW
SUM[0]	4.314	0.250	0.0285	0.0743 µW
SUM[14]	4.314	0.251	0.0284	0.0742 µW
SUM[3]	4.314	0.254	0.0284	0.0741 µW
SUM[10]	4.314	0.254	0.0284	0.0741 µW
SUM[5]	4.314	0.258	0.0283	0.0739 µW
SUM[11]	4.314	0.253	0.0283	0.0739 µW
SUM[13]	4.314	0.254	0.0283	0.0738 µW
Total (20 nets)				2.0177 µW

Tabella 2.7: Contributi di potenza associati alle net dell'intero sistema

Dall'osservazione di questo report appare chiaro che il carico è maggiormente distribuito sulle linee di selezione dei due multiplexers, ovvero, sulle uscite della FSM. La probabilità statistica è distribuita abbastanza uniformemente sulle varie connessioni, come anche il toggle rate. La switching power, dipendendo anche dalla capacità di carico, risulta essere maggiore per le net di output della FSM che, come già riportato, risultano maggiormente caricate.

In Tabella 2.8 sono riportati i dati relativi alle celle della top entity.

Hierarchy	Switch Power	Internal Power	Leakage	Total Power	%
top	13.819 µW	30.237 µW	5.55×10^3 nW	49.607 µW	100.0
FSM	1.532 µW	3.303 µW	458.223 nW	5.294 µW	10.7
DATAPATH	12.287 µW	26.934 µW	5.09×10^3 nW	44.313 µW	89.3

Tabella 2.8: Contributi di potenza associati alle celle dell'intero sistema

Dall'osservazione di questo report invece, si può notare fin da subito che gran parte della potenza associata alle celle viene dissipata dalla porzione combinatoria del nostro circuito, ovvero dal datapath. Questo è dovuto al fatto che le operazioni di somma tra i vari addendi portano i gate ad essere molto attivi e, di conseguenza, questi ultimi dissipano molta più potenza della FSM. A proposito di quest'ultima, è stato ritenuto opportuno ottenere dei report specifici, per nets e celle al suo interno, i cui risultati sono mostrati rispettivamente in Tabella 2.9 e Tabella 2.10.

Net	Total net load	Static Prob.	Toggle Rate	Switching Power
SEL10	12.346	0.153	0.0313	0.2338 µW
SEL01	9.301	0.355	0.0382	0.2151 µW
SEL00	10.169	0.321	0.0345	0.2120 µW
n6	7.558	0.645	0.0382	0.1748 µW
SEL11	7.112	0.355	0.0382	0.1644 µW
n1	7.439	0.679	0.0345	0.1552 µW
n4	5.846	0.847	0.0313	0.1108 µW
n3	1.980	0.645	0.0382	0.0458 µW
n7	2.010	0.831	0.0345	0.0419 µW
n8	6.482	0.500	0.0100	0.0392 µW
n2	2.024	0.153	0.0314	0.0384 µW
n5	1.980	0.847	0.0313	0.0375 µW
current_state[0]	1.438	0.354	0.0382	0.0332 µW
N3	1.438	0.169	0.0345	0.0300 µW
Total (14 nets)				1.5323 µW

Tabella 2.9: Contributi di potenza associati alle nets della FSM

In Tabella 2.9 si può chiaramente osservare una differenza netta rispetto alla top entity, difatti il carico non risulta più essere uniformemente distribuito tra le varie net ma, oltre al carico sugli output della FSM, anche le altre reti risultano essere abbastanza caricate e la switching activity è distribuita in maniera eterogenea portando ad una conseguente distribuzione disomogenea della potenza di switching totale dissipata.

Anche il report relativo alle celle, riportato in Tabella 2.10 risulta essere differente rispetto al caso della top entity, infatti in questo caso sono presenti più voci e si può subito notare che la maggior parte della potenza dinamica è dissipata dalle linee di output della FSM (situazione auspicabile dal report power relativo alle net).

Cell	Internal Power	Switching Power	Dynamic Power (% Net/Tot)	Leakage Power
current_state_reg[0]	1.0025 µW	0.2080 µW	1.211 µW (83%)	87.7227 nW
current_state_reg[2]	0.9620 µW	0.1492 µW	1.111 µW (87%)	82.7103 nW
current_state_reg[1]	0.8569 µW	0.1552 µW	1.012 µW (85%)	85.0164 nW
U11	0.1083 µW	0.0419 µW	0.150 µW (72%)	38.8760 nW
U4	0.0783 µW	0.1644 µW	0.243 µW (32%)	28.7098 nW
U7	0.0693 µW	0.2120 µW	0.281 µW (25%)	35.8295 nW
U6	0.0570 µW	0.2151 µW	0.272 µW (21%)	18.9302 nW
U9	0.0454 µW	0.2338 µW	0.279 µW (16%)	24.3839 nW
U5	0.0425 µW	0.0458 µW	8.83×10^{-2} µW (48%)	13.1191 nW
U8	0.0390 µW	0.0375 µW	7.65×10^{-2} µW (51%)	11.4022 nW
U10	0.0321 µW	0.0300 µW	6.21×10^{-2} µW (52%)	17.1692 nW
U3	9.993×10^{-3} µW	0.0392 µW	4.92×10^{-2} µW (20%)	14.3532 nW

Tabella 2.10: Contributi di potenza associati alle celle della FSM

Inoltre, si può notare chiaramente una maggiore potenza dinamica dissipata dai registri di stato, i quali, essendo elementi sequenziali, devono sempre lavorare in relazione al segnale di clock.

Sempre tramite il comando **create clock -name "CLK" -period 2 clock**, è stato creato un nuovo segnale di clock, questa volta con periodo 5 volte minore, in modo da poter testare il comportamento del circuito e la distribuzione di potenza quando questo viene fatto lavorare ad una velocità maggiore.

I risultati ottenuti sono riportati in Tabella 2.11.

Power Group	Switching Power	Internal Power	Leakage Power	Total Power	%
io_pad	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
memory	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
black_box	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
clock_network	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
register	90.4594 µW	8.5239 µW	1.6266×10^3 nW	100.6098 µW	44.55
sequential	0.0000 µW	0.0000 µW	0.0000 nW	0.0000 µW	0.00
combinational	60.7268 µW	60.5715 µW	3.9245×10^3 nW	125.2228 µW	55.45
total	151.1862 µW	69.0953 µW	5.5511×10^3 nW	225.8326 µW	100

Tabella 2.11: Contributi di potenza relativi all'intero sistema con frequenza di clock 5 volte maggiore

Dai valori numerici riportati, appare chiaro che il consumo di potenza è aumentato di un fattore più o meno pari a quello del clock (a meno di lievissime differenze). Questo era auspicabile, poichè, in questo caso, le probabilità di switching non vengono intaccate, mentre le switching activities ed i toggle rates aumentano. Infine, è stata testata la funzione di

auto-limitazione della potenza dissipata, impostando come valore limite per la TDP 35 µW, valore inferiore ai circa 44 µW dissipati dalla versione standard. Il sintetizzatore ha restituito un nuovo schematico, riportato in Figura 2.8 contenente meno gate della versione standard, e di conseguenza, in grado di dissipare meno potenza.

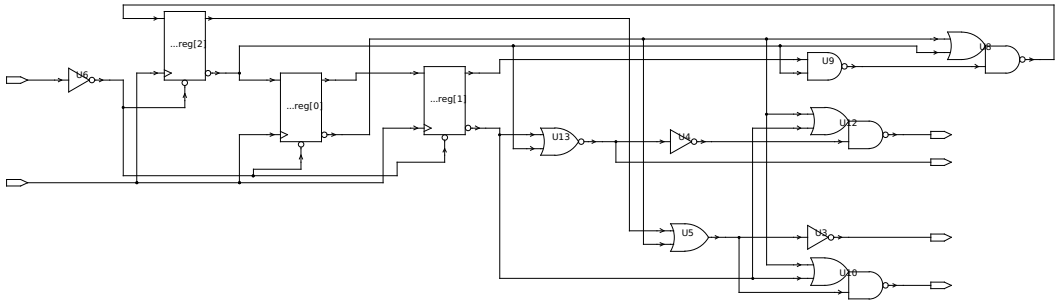

Figura 2.8: Schematico della FSM con limitazione sulla dissipazione di potenza dinamica

A causa di errori interni legati alla elaborazione del report, pur effettuando numerosi tentativi con parametri differenti, il relativo report ha restituito dei valori privi di senso e di conseguenza, non attendibili. Ad esempio, in uno dei test svolti, impostando come valore massimo di potenza 40 µW (per il circuito pilotato con clock "lento"), *Synopsys* ha restituito un power report in cui la potenza dissipata dal circuito è stata pari a 48 µW, valore che viola il limite imposto e quindi, inconsistente.

Gli script utilizzati per le diverse analisi sono riportati per completezza in Appendice B.6 e Appendice B.7.

CAPITOLO 3

Clock gating, pipelining and parallelizing

3.1 A first approach to clock gating

Lo scopo di questo terzo laboratorio è quello di prendere confidenza con la tecnica del **clock gating**, una delle più utilizzate per il risparmio energetico. In Figura 3.1 è riportata l'implementazione di tale tecnica su di un singolo Flip-Flop tramite l'utilizzo di una porta AND ed un latch. A livello puramente logico sarebbe sufficiente una porta AND tra i segnali ENABLE e CK per disabilitare la trasmissione del clock al Flip-Flop, tuttavia sarebbe una soluzione combinatoria, non immune al fenomeno dei glitch e di conseguenza potenzialmente sorgente di errori. Serve dunque un elemento di memoria che fornisca alla porta AND una versione stabile del segnale di ENABLE. Il motivo per cui viene scelto un Latch e non un FF attivo sul fronte negativo del clock è correlato al timing del circuito; in questo modo infatti il segnale di ENABLE ha circa un intero colpo di clock per stabilizzarsi (anzichè metà, se fosse stato scelto un FF) e quindi il critical path non viene sostanzialmente modificato.

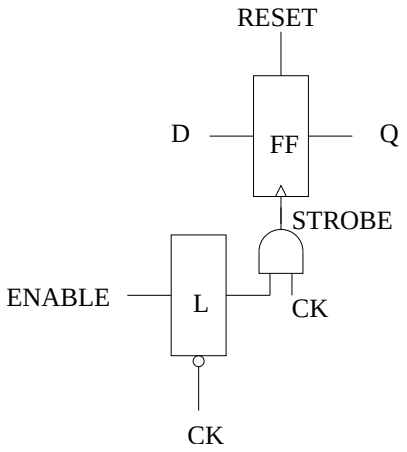

Figura 3.1: Clock gating di un FF

Il file `ckgbug.vhd` descrive l'architettura mostrata in Figura 3.2, dove *D1, D2, D3* sono byte, *L1* e *L2* sono registri da 8-bit. In particolare l'ordinamento dei bit per D1 e D3 è **(7 downto 0)** mentre per D2 è **(0 to 7)**, motivo per cui i bit risulteranno "specchiati".

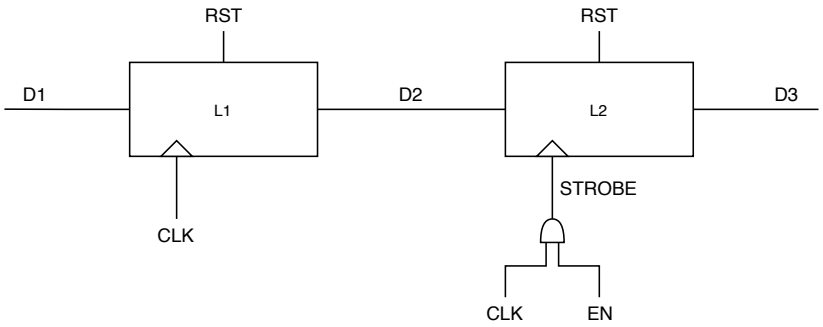

Figura 3.2: Architettura descritta in `ckgbug.vhd`

Il testbench proposto fissa D1 al valore *01111111* ed applica un segnale di RST ai 2 registri. Il comportamento previsto con approccio "pen-and-paper" è riportato in Figura 3.3. Al primo colpo di clock L2 campiona il segnale D1 che viene mandato su D2; **contemporaneamente** L2 campiona D2 (che vale ancora *00000000* in questo momento) e lo manda su D3. Al successivo colpo di clock, infine, il valore iniziale di D1 viene propagato su D3.

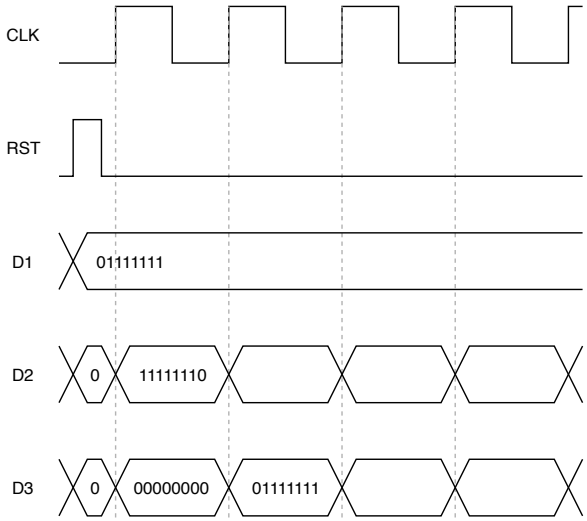

Figura 3.3: Simulazione pen-and-paper dell'architettura **ckgbug.vhd** (expected behavior)

Tuttavia il concetto di **contemporaneità** è difficile da realizzare in fase di simulazione, in quanto il simulatore deve schedulare le operazioni in sequenza (*causalità*). Per questo motivo il risultato della simulazione diverge da quello teorico previsto, come mostrato in Figura 3.4.

La presenza della AND relativa al clock gating fa sì che la sequenza di operazioni simulate sia probabilmente la seguente:

- D1 <= *01111111*
- Reset: D2 e D3 <= *00000000*
- CK <= *1*, D2 <= D1 (*11111110*)
- STROBE <= CK and EN (<= *1*)
- D3 <= D2, che però ha già commutato a *11111110*

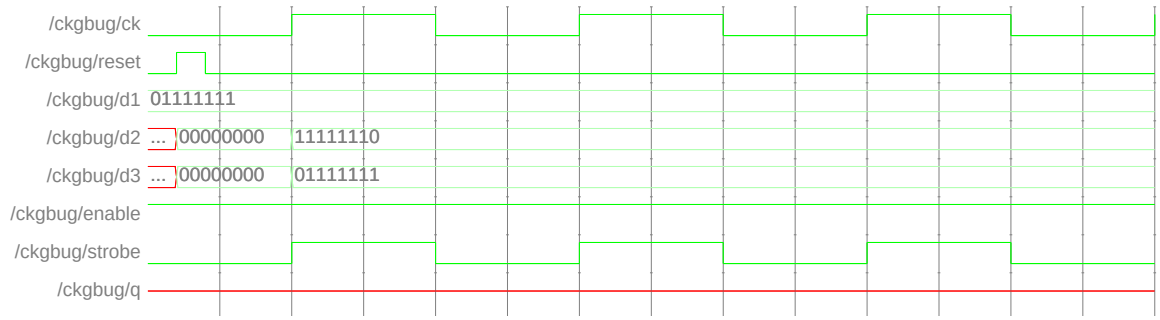

Figura 3.4: Risultato della simulazione che evidenzia il concetto di **causalità** nei simulatori

Per risolvere il problema in fase di simulazione si può inserire un ritardo $CK \rightarrow Q$ nei Flip-Flop (0.1 ps nell'esempio), in modo che quando il registro L2 campiona D2 quest'ultimo assuma ancora il valore iniziale di *00000000*; il concetto appena esposto è riportato in Figura 3.5.

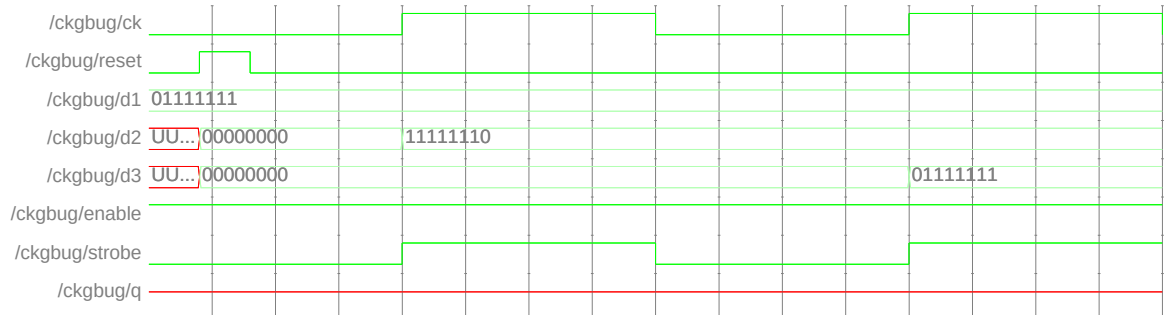

Figura 3.5: Simulazione fixata impostando un ritardo FF di 0.1 ps e ritardo AND di 0 ps

In ultima analisi possiamo considerare il caso in cui la porta AND abbia un ritardo maggiore a quello $CK \rightarrow Q$ dei Flip-Flop, ad esempio 0.2 ps e 0.1 ps rispettivamente (Figura 3.6); anche in questo caso il segnale di STROBE attiverà il registro L2 quando la commutazione di D2 è già avvenuta.

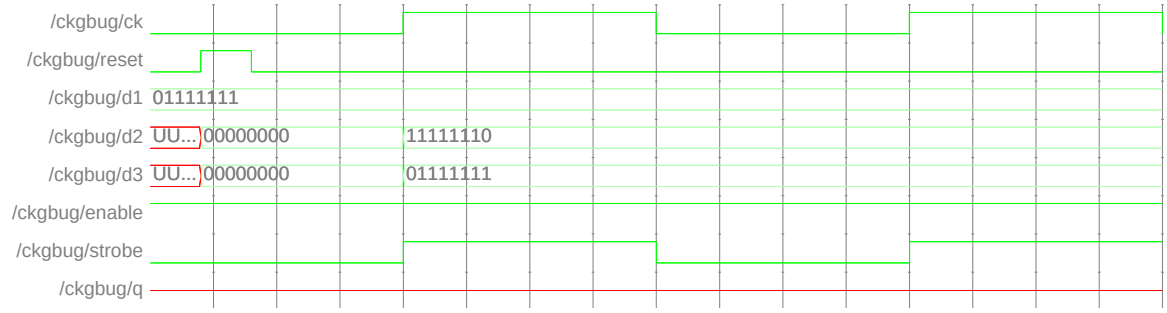

Figura 3.6: Simulazione con ritardo FF di 0.1 ps e ritardo AND di 0.2 ps

3.2 Clock gating for a complex circuit

Il circuito contenunto nel file `inccomp.vhd` è mostrato in Figura 3.7 e rappresenta un comparatore. La sua funzionalità è quella di incrementare il contenuto di 2 registri a seconda che il rispettivo segnale INC sia attivo o meno, comparare i valori ottenuti e memorizzare il maggiore in un registro a valle.

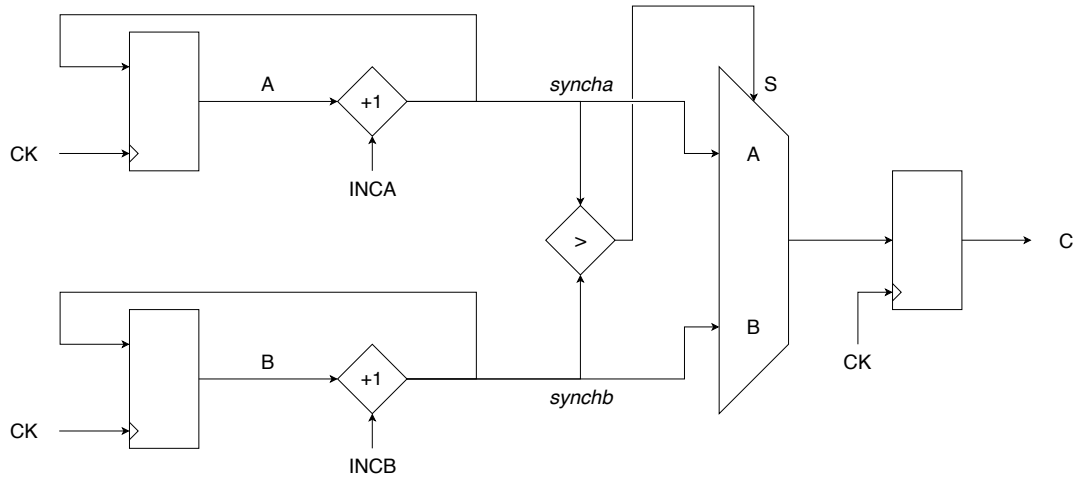

Figura 3.7: Circuito `inccomp.vhd` costituito da 2 incrementer ed un comparatore

Tale comportamento è evidenziato dalla simulazione logica riportata in Figura 3.8, dove l'output C assume sempre il valore del maggiore tra *syncha* e *synchb*.

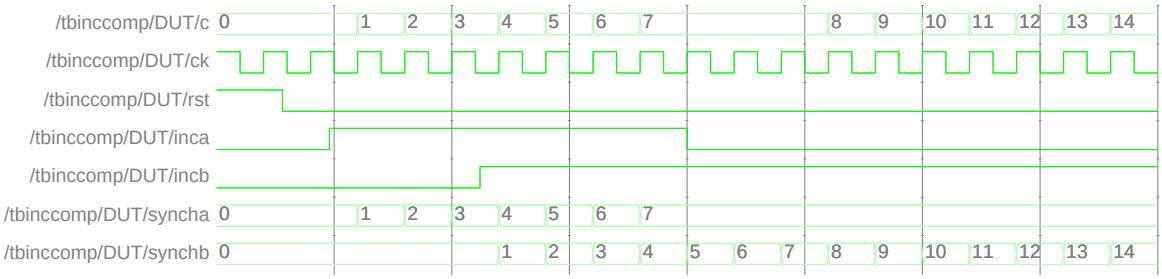

Figura 3.8: Simulazione dell'architettura descritta in `inccomp.vhd`

Come già anticipato l'obiettivo del laboratorio è quello di ridurre i consumi sfruttando la tecnica del **clock gating**. Prima di utilizzare tools automatici abbiamo provato ad utilizzare un approccio *pen-and-paper*, ottenendo il circuito in Figura 3.9.

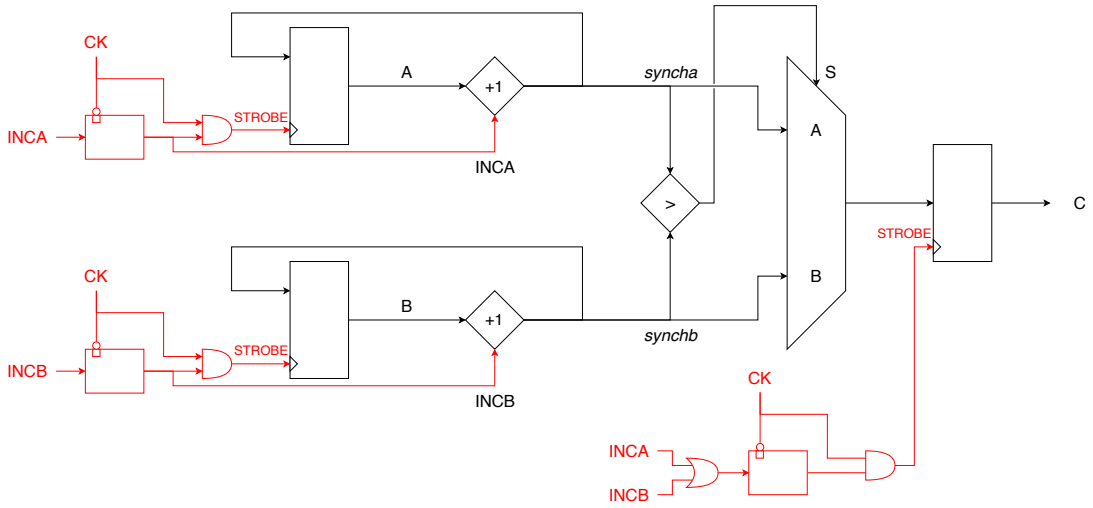

Figura 3.9: Applicazione manuale del clock gating sul circuito `inccomp.vhd`

L'idea è quella di fornire il clock ai registri solo quando necessario, per farlo è possibile sfruttare i segnali di incremento *INCA* ed *INCB*. Nello specifico, i registri a monte commuteranno solo quando effettivamente ci sia stato un incremento, quello a valle commuterà solo quando almeno uno dei due rami in ingresso al MUX abbia cambiato valore.

In seguito all'approccio pen-and-paper si è passati ad analizzare con *Design Compiler* gli effetti benefici del clock gating sul consumo di potenza del circuito. Lo script scritto a questo scopo è riportato in Appendice C.1.[1]

Occorre dapprima sintetizzare il design originale ed analizzarne i consumi, includendo nell'analisi anche i nodi di input *CK*, *RST*, *INCA* ed *INCB* con il comando **-include_input_nets**, i risultati sono riportati in Tabella 3.1.

Cell Internal Power	38.7514 µW	(74%)
Net Switching Power	13.6944 µW	(26%)
Total Dynamic Power	**52.4459 µW**	(100%)
Cell Leakage Power	**3.7969 µW**	

Tabella 3.1: Analisi di potenza del circuito **inccomp** *originale*

Un dettaglio sull'attività di ogni singolo nodo può essere ottenuto con il comando **-nets -include_input_nets**, i cui dati relativi ai nodi più significativi sono riportati in Tabella 3.2.

[1] Tutte le versioni del design sono state sintetizzate con un CK di 5 ns, in modo da poter effettuare paragoni più significativi in merito ai consumi

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power
ck	39.765	0.500	0.4000	9.6231
rst	2.010	0.500	0.0200	0.0243
INCA	19.382	0.500	0.0200	0.2345
INCB	19.382	0.500	0.0200	0.2345
syncha[0]	7.241	0.195	0.0550	0.2408
syncha[1]	4.831	0.184	0.0272	0.0794
[...]	[...]	[...]	[...]	[...]
Total (92 nets)				**13.5445 µW**

Tabella 3.2: Dettaglio sulla switching activity dei nodi del circuito **inccomp** *originale*

Di default il tool assegna probabilità 0.5 ai nodi di input ed il calcolo del ToggleRate viene effettuato mediante l'Equazione 3.1.

$$ToggleRate = \frac{2 \cdot P_1 \cdot (1 - P_1)}{T_{CK}} \tag{3.1}$$

Sostanzialmente il valore restituito da *Design Compiler* è dato dal numero di commutazioni per colpo di clock, diviso la durata di quest'ultimo. Ad esempio il segnale CK, che commuta 2 volte per ogni periodo di clock (5 ns), avrà $ToggleRate = 2/5 = 0.4$.

Per ottenere delle stime di potenza più precise, è possibile assegnare manualmente i parametri probabilistici tramite il comando **set_switching_activity**. Ad esempio, come mostrato in Tabella 3.3, è possibile impostare[2] il ToggleCount del **CK** a 2, la probabilità statica del **RST** a 0, ottenendo i risultati riportati in Tabella 3.4. Quest'ultima dimostra come la potenza dinamica si sia effettivamente abbassata rimuovendo tutte le commutazioni relative al segnale di *RST*.

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power
ck	39.765	**0.500**	**0.4000**	**9.6231**
rst	2.010	**0.000**	**0.0000**	**0.0000**
INCA	19.382	0.500	0.0200	0.2345
INCB	19.382	0.500	0.0200	0.2345
syncha[0]	7.241	0.503	0.0989	0.4332
syncha[1]	4.831	0.506	0.0494	0.1443
[...]	[...]	[...]	[...]	[...]
Total (92 nets)				**14.3766 µW**

Tabella 3.3: Analisi dei *nodi* del circuito **inccomp** originale dopo la modifica manuale delle attività dei nodi **CK** e **RST**

[2]Riferirsi ad Appendice C.1 per lo script completo

Cell Internal Power	27.6360 µW	(65%)
Net Switching Power	14.8140 µW	(35%)
Total Dynamic Power	**42.4501 µW**	(100%)
Cell Leakage Power	**4.1801 µW**	

Tabella 3.4: Analisi di *potenza* del circuito **inccomp** originale, dopo modifica manuale delle attività dei nodi **CK** e **RST**

Come terzo step si sono assegnate delle probabilità più realistiche ai due segnali **INCA** e **INCB** (Tabella 3.5); dal power report complessivo riportato in Tabella 3.6 si può notare come in questo caso ci sia stata un'attività ancora minore e di conseguenza un **minor consumo di potenza** rispetto al caso precedente.

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power
ck	39.765	0.500	0.4000	9.6231
rst	2.010	0.000	0.0000	0.0000
INCA	19.382	**0.120**	**0.0050**	**0.0586**
INCB	19.382	**0.120**	**0.0050**	**0.0586**
syncha[0]	7.241	0.443	0.0237	0.1037
syncha[1]	4.831	0.501	0.0118	0.0345
[...]	[...]	[...]	[...]	[...]
Total (92 nets)				**10.7715 µW**

Tabella 3.5: Analisi dei *nodi* del circuito **inccomp** originale dopo la modifica manuale delle attività dei nodi **INCA** e **INCB**

Cell Internal Power	21.5273 µW	(66%)
Net Switching Power	10.8718 µW	(34%)
Total Dynamic Power	**32.3990 µW**	(100%)
Cell Leakage Power	**4.0921 µW**	

Tabella 3.6: Analisi di *potenza* del circuito **inccomp** originale, dopo la modifica dei parametri relativi ad **INCA** ed **INCB**

In ultima analisi è possibile ricavare il numero di celle utilizzate dal sintetizzatore in questo design tramite il comando **report_cell**, da cui si evince che il circuito originale è stato sintetizzato utilizzando **74 celle** con un'area totale di **229.292005 unità**.

La Tabella 3.7 riassume il confronto sui consumi di potenza nei 3 casi analizzati. Come si può notare l'analisi effettuata da *Design Compiler* è strettamente dipendente dalle attività impostate sui singoli nodi. Si vedrà più avanti nel report (sezione 3.2.2) come si possano impostare dei parametri veritieri in modo del tutto automatico tramite la *back-annotation*.

	Parametri Default	Mod CK-RST	Mod CK-RST INCA-INCB
Cell Internal Power	38.7514 µW	27.6360 µW	21.5273 µW
Net Switching Power	13.6944 µW	14.8140 µW	10.8718 µW
Total Dynamic Power	**52.4459 µW**	**42.4501 µW**	**32.3990 µW**
Cell Leakage Power	**3.7969 µW**	**4.1801 µW**	**4.0921 µW**

Tabella 3.7: Confronto fra le 3 analisi del circuito **inccomp** originale, con e senza settaggio dei parametri probabilistici

Clock Gating · A questo punto l'obiettivo è quello di applicare la tecnica del clock gating (**compile -exact_map -gate_clock**), ripetendo gli stessi 3 step del paragrafo precedente allo scopo di verificare se e quanto questa tecnica consenta di risparmiare energia. Il confronto tra design prima e dopo il clock gating nei 3 casi è riportato in Tabella 3.8, Tabella 3.9 e Tabella 3.10. In particolare quest'ultima evidenzia come il clock gating abbia permesso di **risparmiare il 40.70% in potenza dinamica**, a patto di accettare un **aumento del 4.03% del leakage** dovuto agli elementi aggiuntivi introdotti.

Parametri Default	CK-Gating	Originale	Variazione
Cell Internal Power	32.9555 µW	38.7514 µW	
Net Switching Power	10.8986 µW	13.6944 µW	
Total Dynamic Power	**43.8542 µW**	**52.4459 µW**	**-16.38%**
Cell Leakage Power	**3.9340 µW**	**3.7969 µW**	**+3.61%**

Tabella 3.8: Circuito **inccomp** originale con e senza clock gating

Mod CK-RST	CK-Gating	Originale	Variazione
Cell Internal Power	26.5070 µW	27.6360 µW	
Net Switching Power	11.9602 µW	14.8140 µW	
Total Dynamic Power	**38.4672 µW**	**42.4501 µW**	**-9.38%**
Cell Leakage Power	**4.3465 µW**	**4.1801 µW**	**+3.98%**

Tabella 3.9: Circuito **inccomp** originale con e senza clock gating, dopo modifica manuale delle attività dei nodi **CK** e **RST**

Mod CK-RST-INCA-INCB	CK-Gating	Originale	Variazione
Cell Internal Power	13.3592 µW	21.5273 µW	
Net Switching Power	5.8499 µW	10.8718 µW	
Total Dynamic Power	**19.2091 µW**	**32.3990 µW**	**-40.71%**
Cell Leakage Power	**4.2570 µW**	**4.0921 µW**	**+4.03%**

Tabella 3.10: Circuito **inccomp** originale con e senza clock gating, dopo la modifica dei parametri relativi ad **INCA** ed **INCB**

Per avere una spiegazione del ridotto consumo energetico si è analizzato il report relativo all'attività delle singole celle nell'ultimo caso (Tabella 3.11).

Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power
ck	**16.095**	0.500	0.4000	**3.8951**
net2925	**11.835**	0.254	0.2033	**1.4553**
net2919	**11.835**	0.249	0.1990	**1.4250**
rst	2.010	0.000	0.0000	0.0000
INCA	20.758	0.120	0.0050	0.0628
INCB	20.758	0.120	0.0050	0.0628
syncha[0]	7.241	0.515	0.0237	0.1040
syncha[1]	4.831	0.468	0.0118	0.0345
[...]	[...]	[...]	[...]	[...]
Total (92 nets)				**5.7496 µW**

Tabella 3.11: Analisi dei *nodi* del circuito **inccomp** dopo applicazione del **clock-gating** e la modifica manuale delle attività dei nodi **INCA** e **INCB**

In precedenza la rete di *CK* doveva pilotare 39.765 fF, diversamente dopo il clock gating tale carico complessivo è stato ridistribuito sui nodi *CK*, *net2919* e *net2925*. Dal momento che questi ultimi due nodi hanno attività decisamente inferiori a quella del *CK*, la quantità di energia dissipata durante la carica/scarica della capacità complessiva iniziale è notevolmente ridotta.

Per avere un'idea degli elementi circuitali introdotti è sufficiente navigare nello schematico generato da *Design Compiler* e riportato in Figura 3.10. Come avevamo previsto durante l'analisi manuale in Figura 3.9 (Pag. 37), gli elementi di clock gating utilizzano i segnali *INCA* ed *INCB* come enable.

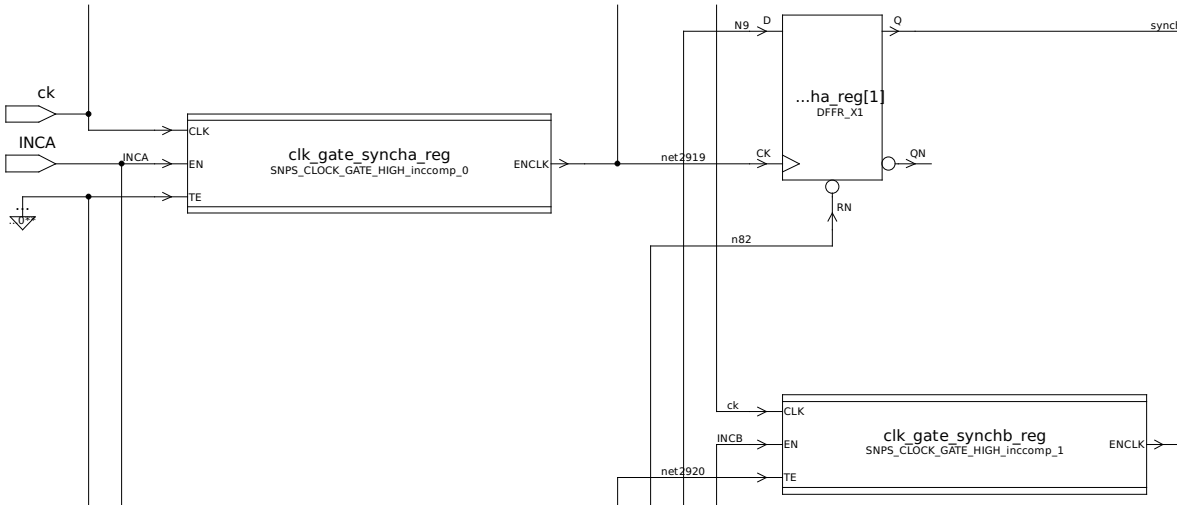

Figura 3.10: Elementi di clock gating inseriti automaticamente da *Design Compiler*

Dallo schematico sembrerebbe che il tool abbia aggiunto solo 2 blocchi al circuito. Tuttavia tramite il comando **report_cell** si evidenzia come ora ci siano **78 celle** (4 in più di prima) ed un'area di **238.070005 unità**.

Per giustificare questo risultato si è esplorato più a fondo lo schematico arrivando ai blocchi `CLKGATETST_X1` (Figura 3.11), i quali rappresentano l'ultimo livello visualizzabile che riteniamo contengano al loro interno un Latch ed una porta AND.

Figura 3.11: Dettaglio del latch di clock gating inserito da *Design Compiler*

3.2.1 Some more clock gating?

Rispetto all'applicazione manuale del clock gating riportata in Figura 3.9, *Design Compiler* non ha applicato la tecnica sul registro di uscita; affinchè si possa ottenere il risultato previsto è necessario modificare il codice VHDL inserendo una porta OR che vede in ingresso i due segnali INCA e INCB, come riportato in Appendice C.2. In questo modo il registro in uscita memorizza il dato solo se almeno uno dei due rami ha effettuato la commutazione.

Successivamente si sono ripetuti gli stessi passaggi effettuati per il circuito originale allo scopo di confrontare i risultati.

- Tabella 3.12, circuito con parametri probabilistici di default;

Parametri Default	MOD	Originale
Cell Internal Power	39.1897 µW	38.7514 µW
Net Switching Power	14.8742 µW	13.6944 µW
Total Dynamic Power	**54.0639 µW**	**52.4459 µW**
Cell Leakage Power	**4.6358 µW**	**3.7969 µW**

Tabella 3.12: Circuito **inccomp** originale vs circuito con forzatura per clock gating, con parametri probabilistici di default

- Tabella 3.13, circuito con parametri di **CK** e **RST** modificati;

Mod CK-RST	MOD	Originale
Cell Internal Power	28.3881 µW	27.6360 µW
Net Switching Power	16.4611 µW	14.8140 µW
Total Dynamic Power	**44.8492 µW**	**42.4501 µW**
Cell Leakage Power	**4.6611 µW**	**4.1801 µW**

Tabella 3.13: Circuito **inccomp** originale vs circuito con forzatura per clock gating, dopo modifica manuale delle attività dei nodi **CK** e **RST**

- Tabella 3.14, circuito con parametri di **INCA** e **INCB** modificati;

Mod CK-RST-INCA-INCB	MOD	Originale
Cell Internal Power	21.3760 µW	21.5273 µW
Net Switching Power	11.4117 µW	10.8718 µW
Total Dynamic Power	**32.7876 µW**	**32.3990 µW**
Cell Leakage Power	**4.5424 µW**	**4.0921 µW**

Tabella 3.14: Circuito **inccomp** originale vs circuito con forzatura per clock gating, dopo la modifica dei parametri relativi ad **INCA** ed **INCB**

Clock gating · Apparentemente, quindi, la modifica al circuito iniziale introducendo la porta OR ha apportato esclusivamente effetti peggiorativi. Tuttavia tale modifica è stata pensata per applicare anche sul registro d'uscita la tecnica del clock gating e sfruttare al massimo le sue potenzialità.

Di seguito i risultati ottenuti dopo aver forzato *Design Compiler* a sintetizzare utilizzando tale tecnica.

- Tabella 3.15, circuito con parametri probabilistici di default;

Parametri Default	MOD	Originale
Cell Internal Power	33.1194 µW	32.955 µW
Net Switching Power	10.8040 µW	10.898 µW
Total Dynamic Power	**43.9234 µW**	**43.854 µW**
Cell Leakage Power	**4.0346 µW**	**3.934 µW**

Tabella 3.15: Circuito con e senza forzatura per clock gating, con parametri probabilistici di default

- Tabella 3.16, circuito con parametri di **CK** e **RST** modificati;

Mod CK-RST	MOD	Originale
Cell Internal Power	27.8883 µW	26.5070 µW
Net Switching Power	11.9228 µW	11.9602 µW
Total Dynamic Power	**39.8112 µW**	**38.4672 µW**
Cell Leakage Power	**4.4199 µW**	**4.3465 µW**

Tabella 3.16: Circuito con e senza forzatura per clock gating, dopo modifica manuale delle attività dei nodi **CK** e **RST**

- Tabella 3.17, circuito con parametri di **INCA** e **INCB** modificati;

Mod CK-RST-INCA-INCB	MOD	Originale
Cell Internal Power	10.0773 µW	13.3592 µW
Net Switching Power	4.1839 µW	5.8499 µW
Total Dynamic Power	**14.2612 µW**	**19.2091 µW**
Cell Leakage Power	**4.3468 µW**	**4.2570 µW**

Tabella 3.17: Circuito con e senza forzatura per clock gating, dopo la modifica dei parametri relativi ad **INCA** ed **INCB**

La Tabella 3.17 mostra che, dopo il clock gating e l'impostazione di parametri probabilistici veritieri, il circuito modificato consumi meno di quello originale, nonostante presenti più celle (**81** rispetto a **78**).

In Figura 3.12 è riportato il blocco aggiuntivo introdotto da *Design Compiler* per il clock gating sul registro d'uscita.

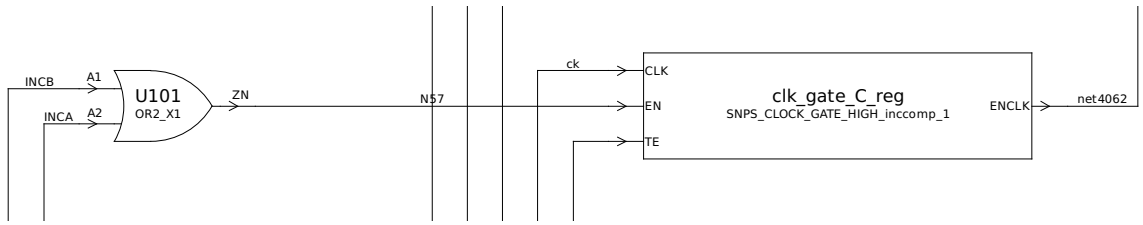

Figura 3.12: Elementi di clock gating inseriti automaticamente da *Design Compiler* sul registro d'uscita

3.2.2 An automatic way to annotate activities

Nella sezione precedente si è visto come si possano ottenere dei report di potenza più precisi semplicemente impostando dei parametri di attività e probabilità più realistici sui singoli nodi. Questo è un processo che è conveniente effettuare in modo automatico, soprattutto in caso di circuiti molto complessi. Lo schema generale di tale procedura è riportato in Figura 3.13.

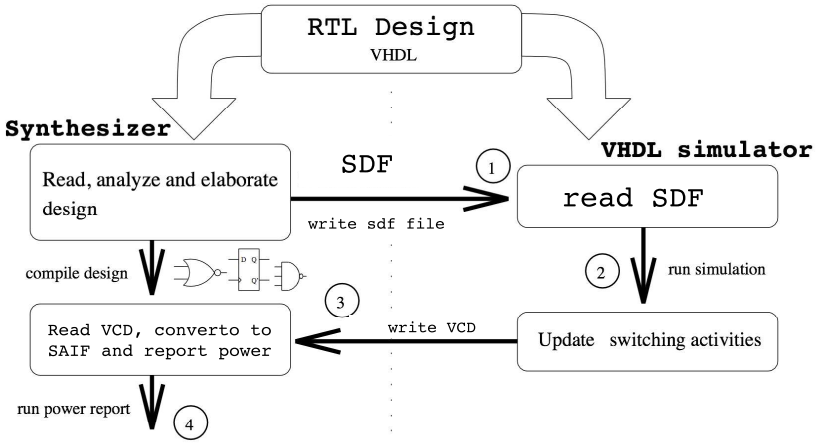

Figura 3.13: Procedura per l'annotazione automatica delle attività e probabilità di switching dei nodi di una netlist

Sostanzialmente i passaggi che occorre eseguire sono i seguenti:

- Design Compiler
  - Sintesi del circuito
  - Generazione del file SDF, contenente le informazioni sui delay effettivi delle celle utilizzate
  - Generazione della netlist Verilog
- ModelSim
  - Simulazione ed annotazione delle commutazioni
  - Generazione del file VCD, contenente le informazioni sull'attività dei singoli nodi (*strettamente dipendente dal testbench utilizzato*)
  - Conversione del file da VCD a SAIF (leggibile da *Design Compiler*)

- Design Compiler
  - Report di potenza, tenendo in conto i dati contenuti nel file SAIF

I comandi utilizzati sono tutti raccolti in Appendice C.3. Volendo analizzare il file *SAIF* generato, si considera come esempio il nodo di *RST*:

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
[...]
(INSTANCE tbinccomp
  (INSTANCE DUT
    (NET
      [...]
      (rst
        (T0 1993000) (T1 7000) (TX 0)
        (TC 1) (IG 0)
      )
[...]
```

Come si può notare esso commuta solo una volta nel testbench, quindi ha $T_c = 1$. Dato che il periodo di simulazione è stato di 2000 ns, applicando l'Equazione 3.1 ci si aspetta di trovare nei report generati da *Design Compiler* un ToggleRate di $1/2000 = 0.0005$ per il nodo *RST*. La conferma di quanto previsto si ha in Tabella 3.18, dove sono stati evidenziati i parametri impostati diversamente da quelli di default e calcolati grazie al lavoro sincronizzato di *Design Compiler* e *ModelSim*.

	Net	Total Net Load	Static Prob.	Toggle Rate	Switching Power
Default	ck	39.765	0.500	0.4000	9.6231
	rst	2.010	**0.500**	**0.0200**	**0.0243**
	INCA	19.382	**0.500**	**0.0200**	**0.2345**
	INCB	19.382	**0.500**	**0.0200**	**0.2345**
	syncha[0]	7.241	**0.195**	**0.0550**	**0.2408**
	syncha[1]	4.831	**0.184**	**0.0272**	**0.0794**
	[...]	[...]	[...]	[...]	[...]
	Total (92 nets)				**13.5445 µW**
Backannotation	ck	39.765	0.500	0.4000	9.6231
	rst	2.010	**0.004**	**0.0005**	**0.0006**
	INCA	19.382	**0.019**	**0.0010**	**0.0117**
	INCB	19.382	**0.986**	**0.0005**	**0.0059**
	syncha[0]	7.241	**0.010**	**0.0040**	**0.0175**
	syncha[1]	4.831	**0.010**	**0.0020**	**0.0058**
	[...]	[...]	[...]	[...]	[...]
	Total (92 nets)				**14.6775 µW**

Tabella 3.18: Confronto tra i parametri probabilistici di default e quelli ottenuti mediante processo di **back-annotation**

3.3 Pipelining and parallelizing

Il punto di partenza per l'analisi effettuata in questa sezione è la Tabella 3.19, contenente la caratterizzazione degli elementi circuitali @1 V, 5 MHz.

Cell Type	Delay (ns)	Power (µW @1 V, 5 MHz)	Area (µm^2)
Register	2.0 ($CK \rightarrow Q$)	0.6	319.0
Incrementer	40.0	2.55	256.0
Comparator	84.0	2.16	161.0
MUX	14.0	1.67	117.0

Tabella 3.19: Caratterizzazione degli elementi circuitali

Inoltre ci sono state fornite l'Equazione 3.2 relativa al ritardo e l'Equazione 3.3 relativa alla potenza, in cui $u = V_{DD}/V_{DD,NOM}$ è il potenziale normalizzato.

$$T(u) = T(V_{DD} = V_{DD,NOM}) \cdot \frac{0.75u}{u - 0.25} \tag{3.2}$$

$$P(u) = P_{NOM} \cdot u^2 \tag{3.3}$$

Come prima cosa ci è stato richiesto di analizzare il circuito originale (Figura 3.14), ricalcolandone i parametri alla massima frequenza operativa possibile[3] (Tabella 3.20).

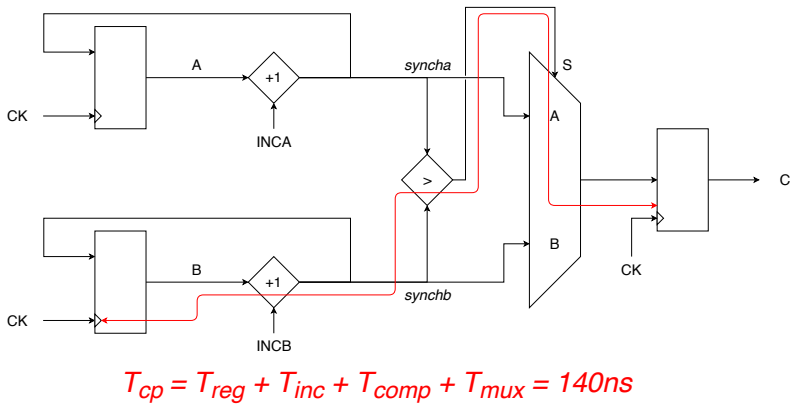

Figura 3.14: Critical path dell'architettura originale

Circuito Originale (high-speed)		
Critical Path	$T_{REG} + T_{INC} + T_{COMP} + T_{MUX}$	140 ns
Max Frequency	$1/T_{CP}$	7.14 MHz
Area	$3A_{REG} + 2A_{INC} + A_{COMP} + A_{MUX}$	1747 µm^2
Power (@f_{MAX})	$(3P_{REG} + 2P_{INC} + P_{COMP} + P_{MUX}) \cdot f_{MAX}/5\,\text{MHz}$	15.32 µW

Tabella 3.20: Circuito **originale** operante alla sua **massima frequenza**

[3]Si suppone una relazione lineare tra la P_{NOM}@5 MHz e P_{MAX}@f_{MAX}

A questo punto per ridurre i consumi ci sono due possibilità architetturali che si possono adottare (singolarmente o in combinazione): la **parallelizzazione** e la **pipeline**, che verranno analizzate separatamente nel seguito.

Parallelizzazione · Parallelizzare significa replicare un numero L di volte il datapath del circuito in modo da poter gestire parallelamente L dati in ingresso. Dato che in questo contesto non si è intenzionati ad aumentare le performance del circuito, è possibile sfruttare il maggior throughput per far lavorare le varie repliche del datapath ad una frequenza L volte inferiore all'originale. I dati in ingresso saranno processati in sequenza dai diversi datapath ed un MUX in uscita selezionerà opportunamente il dato da mandare in output; il rallentamento dei singoli datapath consente di ridurre la V_{DD}e e di conseguenza i consumi con legge quadratica,al costo di un aumento (circa di un fattore L) dell'area e della potenza di leakage. L'architettura parallelizzata con ordine *L=2* è mostrata in Figura 3.15.

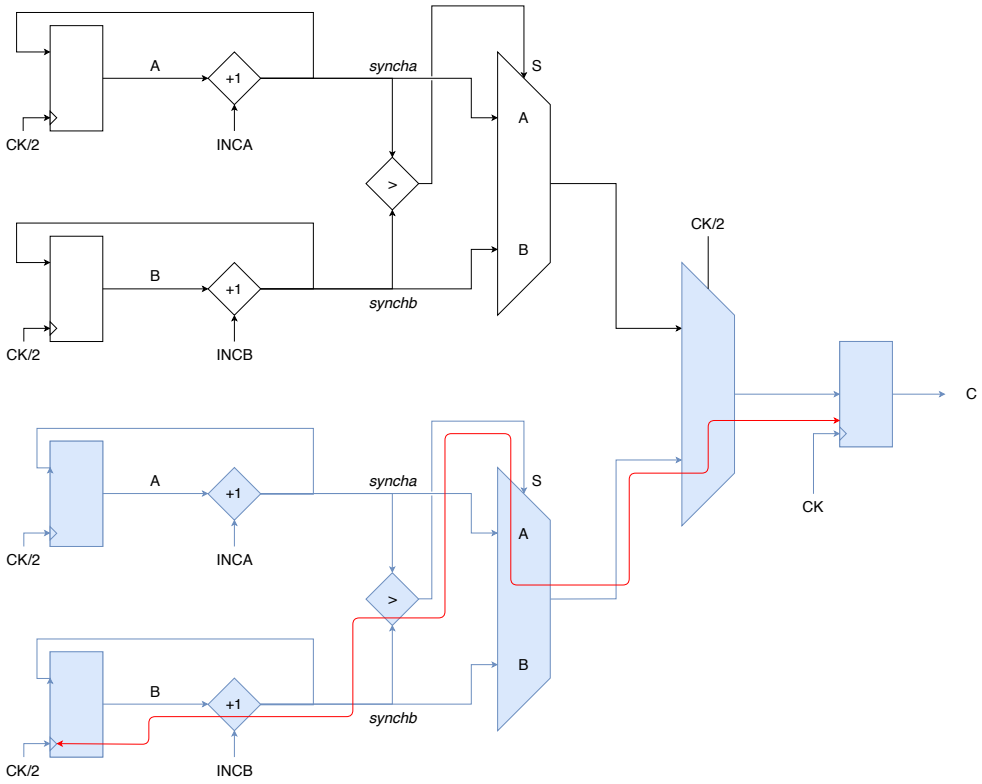

Figura 3.15: Architettura parallelizzata con ordine L=2

Come prima cosa si sono calcolati i parametri del circuito in condizioni di **massime performance** (Tabella 3.21).

Circuito Parallelizzato (high-speed)		
Critical Path	$T_{REG} + T_{INC} + T_{COMP} + 2T_{MUX}$	154 ns
Max Frequency	$\frac{1}{T_{CP}}$	6.49 MHz
Area	$5A_{REG} + 4A_{INC} + 2A_{COMP} + 3A_{MUX}$	3292 µm^2
Power (@f_{MAX})	$(5P_{REG} + 4P_{INC} + 2P_{COMP} + 3P_{MUX}) \cdot \frac{f_{MAX}}{5\,\text{MHz}}$	29.24 µW

Tabella 3.21: Circuito **parallelizzato** operante alla sua **massima frequenza**

Dopodichè, è stato possibile dimezzare la frequenza in modo da abbassare la tensione di alimentazione. Utilizzando l'Equazione 3.2 e l'Equazione 3.3, si sono ricavati i seguenti risultati:

$$T_{LOW,PAR} = 2 \cdot T_{MAX,ORIG} = 2 \cdot 140 = \mathbf{280\,ns}$$

$$T_{LOW,PAR} = T_{MAX,PAR} \cdot \frac{0.75u}{u - 0.25} \Rightarrow 280 = 154 \cdot \frac{0.75u}{u - 0.25} \Rightarrow \mathbf{u = 0.426}$$

$$P_{LOW,PAR} = P_{MAX,PAR} \cdot u^2 \cdot \frac{f_{MAX,ORIG}/2}{f_{MAX,PAR}} = 29.24 \cdot 0.426^2 \cdot \frac{7.14}{2 \cdot 6.49} = \mathbf{2.919\,\mu W}$$

La configurazione Low-Power del circuito parallelizzato è dunque riassunta in Tabella 3.22, che evidenzia un risparmio di circa **81%**.

Circuito Parallelizzato (low-power)	
Working Frequency	3.57 MHz
V_{DD}	0.426 V
Power (@f_{LOW})	2.919 µW
Power Saving	**80.95%**

Tabella 3.22: Circuito **parallelizzato** in condizioni di **risparmio energetico**

Pipeline · Pipelinare significa spezzare il percorso critico mediante l'inserimento di registri, andando ad aumentare la frequenza massima disponibile.

Tuttavia in questo contesto non si è intenzionati ad aumentare le performance, quindi è possibile lavorare alla stessa frequenza del circuito originale, riducendo la tensione di alimentazione e risparmiando energia. Inoltre l'inserimento di registri migliora le caratteristiche del circuito in termini di propagazione dei glitch. Anche in questo caso il costo da pagare è in termini di area e potenza di leakage.

L'architettura pipelinata proposta è mostrata in Figura 3.16; il massimo risultato possibile si ottiene isolando il comparatore, in quanto è l'elemento con il maggior ritardo.

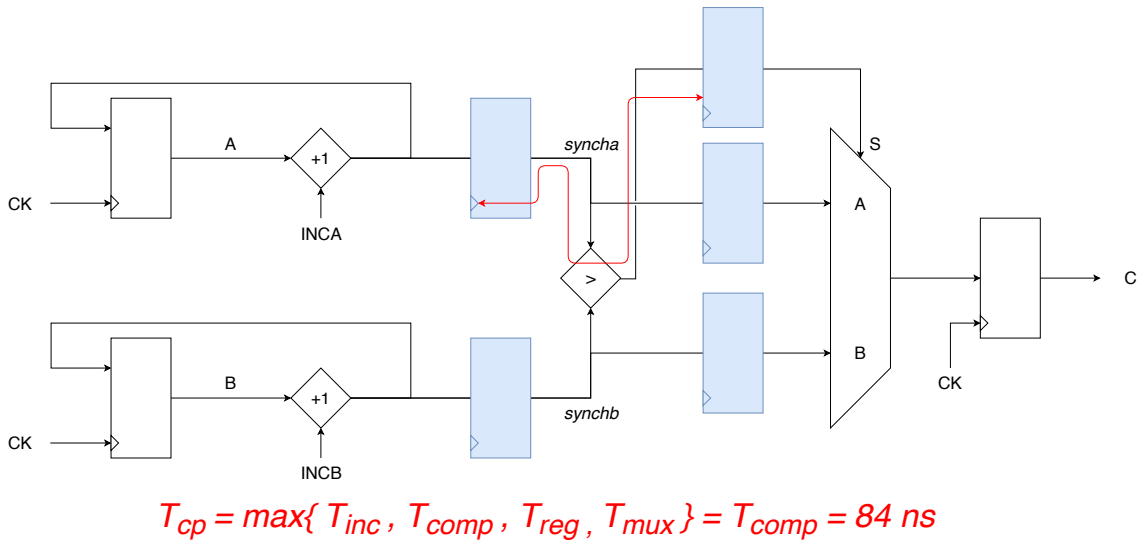

Figura 3.16: Architettura pipelinata

Come nel caso precedente, innanzitutto si calcolano i parametri del circuito in condizioni di **massime performance**. Possiamo notare che tutti i registri inseriti nel circuito sono da 8-bit, fatta eccezione per quello all'uscita del comparatore che è solo da 1-bit, pertanto, per avere delle stime più veritiere, ne abbiamo considerato area e consumi 8 volte più piccoli di quelle degli altri registri (rispettivamente $39.87\,\mu m^2 \cong \mathbf{40\,\mu m^2}$ e $0.075\,\mu W \Rightarrow$ **trascurabile**). I risultati dell'analisi sono riportati in Tabella 3.23.

Circuito Pipelinato (high-speed)		
Critical Path	$max[T_{REG} + T_{INC}, T_{REG} + T_{COMP}, T_{REG} + T_{MUX}]$	86 ns
Max Frequency	$\frac{1}{T_{CP}}$	11.63 MHz
Area	$7A_{REG} + A_{REG,1bit} + 2A_{INC} + A_{COMP} + A_{MUX}$	$3063\,\mu m^2$
Power ($@f_{MAX}$)	$(7P_{REG} + 2P_{INC} + P_{COMP} + P_{MUX}) \cdot \frac{f_{MAX}}{5\,MHz}$	$30.54\,\mu W$

Tabella 3.23: Circuito **pipelinato** operante alla sua **massima frequenza**

Successivamente si è proceduto ai calcoli per abbassare la tensione di alimentazione per l'ottimizzazione dei consumi. Utilizzando l'Equazione 3.2 e l'Equazione 3.3, si sono ricavati i seguenti risultati:

$$T_{LOW,PIPE} = T_{MAX,ORIG} = \mathbf{140\,ns}$$

$$T_{LOW,PIPE} = T_{MAX,PIPE} \cdot \frac{0.75u}{u - 0.25} \Rightarrow 140 = 86 \cdot \frac{0.75u}{u - 0.25} \Rightarrow \mathbf{u = 0.464}$$

$$P_{LOW,PIPE} = P_{MAX,PIPE} \cdot u^2 \cdot \frac{f_{MAX,ORIG}}{2 \cdot f_{MAX,PIPE}} = 30.54 \cdot 0.464^2 \cdot \frac{7.14}{11.63} = \mathbf{4.04\,\mu W}$$

La configurazione Low-Power del circuito pipelinato è riportata in Tabella 3.24, che evidenzia un risparmio di oltre il **73%**.

Circuito Pipelinato (low-power)	
Working Frequency	7.14 MHz
V_{DD}	0.464 V
Power ($@f_{LOW}$)	4.04 µW
Power Saving	**73.63%**

Tabella 3.24: Circuito **pipelinato** in condizioni di **risparmio energetico**

Parallelizzazione e Pipeline · Un'altra possibilità per ridurre ulteriormente la tensione di alimentazione è quella di combinare le due tecniche, a patto di accettare un considerevole aumento dell'area e della potenza di leakage. Nel seguito è riportato come esempio una parallelizzazione di ordine 3 abbinata a 2 stadi di pipeline (Figura 3.17).

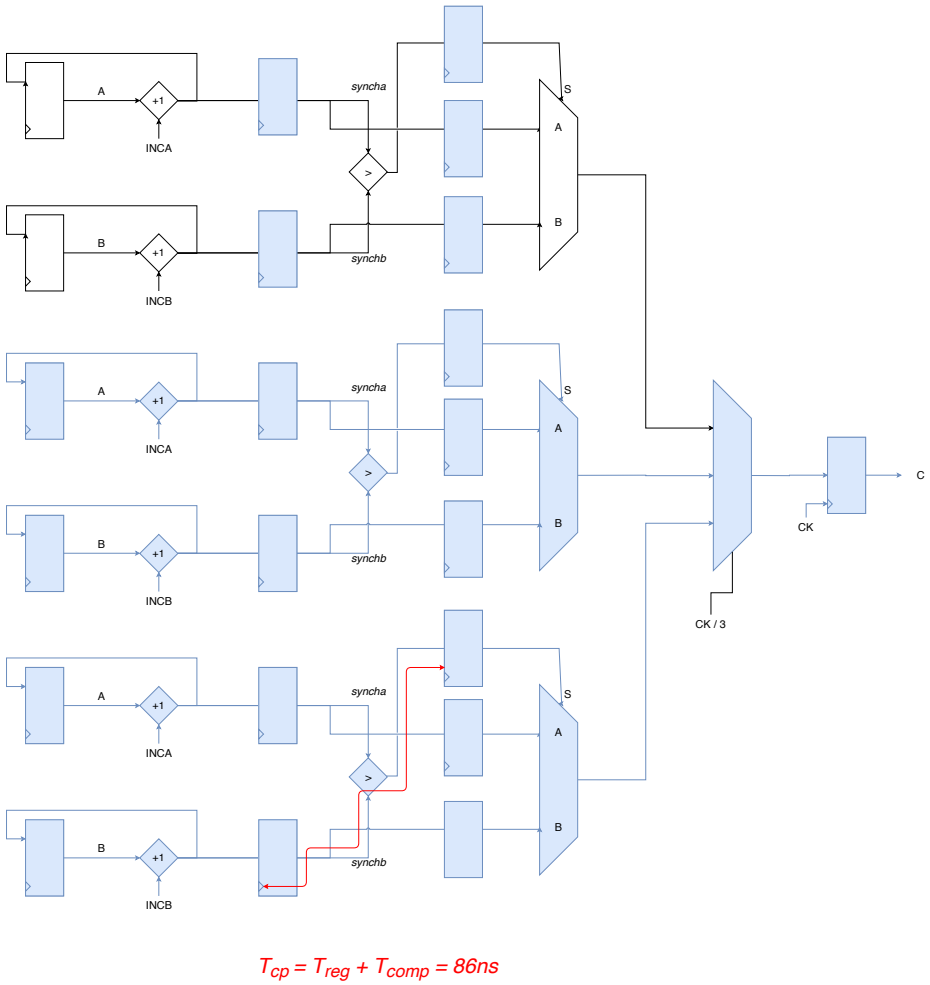

Figura 3.17: Architettura parallelizzata (L=3) e pipelinata

I parametri del circuito in condizioni di **massime performance** sono riportati in Tabella 3.25.

Circuito Parallelizzato e Pipelinato (high-speed)		
Critical Path	$max[T_{REG}+T_{INC}, T_{REG}+T_{COMP}, T_{REG}+T_{MUX}]$	86 ns
Max Frequency	$\frac{1}{T_{CP}}$	11.63 MHz
Area	$19A_{REG}+3A_{REG,1bit}+6A_{INC}+3A_{COMP}+4A_{MUX}$	8668 µm^2
Power (@f_{MAX})	$(19P_{REG}+6P_{INC}+3P_{COMP}+4P_{MUX})\cdot\frac{f_{MAX}}{5\,\text{MHz}}$	92.71 µW

Tabella 3.25: Circuito **parallelizzato e pipelinato** operante alla sua **massima frequenza**

Successivamente si sono svolti i seguenti calcoli per abbassare la tensione di alimentazione. In questo caso il circuito può funzionare 3 volte più lentamente dell'originale.

$$T_{LOW,PARPIPE} = 3 \cdot T_{MAX,ORIG} = 3 \cdot 140 = \mathbf{420\,ns}$$

$$T_{LOW,PARPIPE} = T_{MAX,PARPIPE} \cdot \frac{0.75u}{u-0.25} \Rightarrow 3 \cdot 140 = 86 \cdot \frac{0.75u}{u-0.25} \Rightarrow \mathbf{u = 0.295}$$

$$P_{LOW,PARPIPE} = P_{MAX,PARPIPE} \cdot u^2 \cdot \frac{f_{MAX,ORIG}/3}{f_{MAX,PARPIPE}} = 92.71 \cdot 0.295^2 \cdot \frac{7.14}{3 \cdot 11.63} = \mathbf{1.65\,\mu W}$$

La configurazione Low-Power del circuito parallelizzato e pipelinato è riportata in Tabella 3.26, che evidenzia un risparmio del **89%**.

Circuito Parallelizzato e Pipelinato (low-power)	
Working Frequency	2.38 MHz
V_{DD}	0.295 V
Power (@f_{LOW})	1.65 µW
Power Saving	**89.23%**

Tabella 3.26: Circuito **parallelizzato e pipelinato** in condizioni di **risparmio energetico**

Comparazione · Come conclusione si riporta in Tabella 3.27 un confronto tra le varie soluzioni low-power. Il maggior risparmio di **potenza dinamica** si ottiene con una combo di parallelizzazione e pipeline, che però comporta un aumento di area e del numero di dispositivi, il quale potrebbe aumentare di molto il leakage. In definitiva la scelta di una soluzione dipende dall'area disponibile e dall'analisi sulla potenza statica oltre che dinamica.

Implementazioni Low-Power				
	Original	Parallel-2	Pipeline	Par-3 + Pipe
Working Frequency	7.14 MHz	3.57 MHz	7.14 MHz	2.38 MHz
V_{DD}	1 V	0.426 V	0.464 V	0.295 V
Power (@f_{LOW})	15.32 µW	2.919 µW	4.04 µW	1.65 µW
Area	1747 µm^2	3292 µm^2	3063 µm^2	8668 µm^2
Power Saving		80.95%	73.63%	89.23%

Tabella 3.27: Confronto tra le diverse implementazioni a **risparmio energetico**

3.3.1 Are you sure it was correct?

Tuttavia è abbastanza intuitivo notare che applicando la parallelizzazione in modo standard, come fatto finora, in realtà si ottengono soluzioni logicamente errate. Per averne conferma si sono simulate con approccio pen-and-paper le forme d'onda del circuito parallelizzato (Figura 3.18) per poi confrontarle con quelle del circuito originale (Figura 3.19).

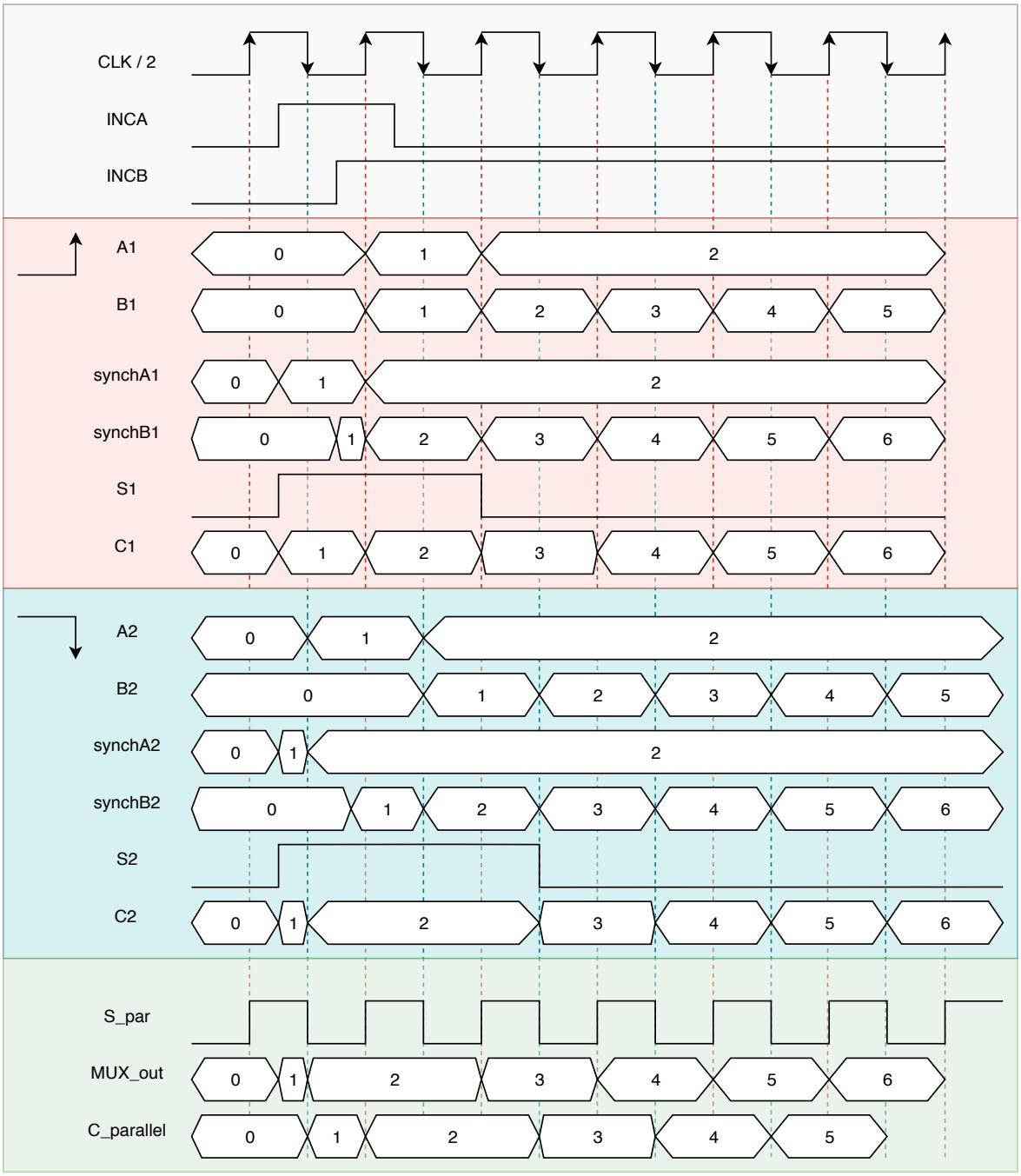

Figura 3.18: Timing Diagram dell'architettura parallelizzata con ordine L = 2 di Figura 3.15

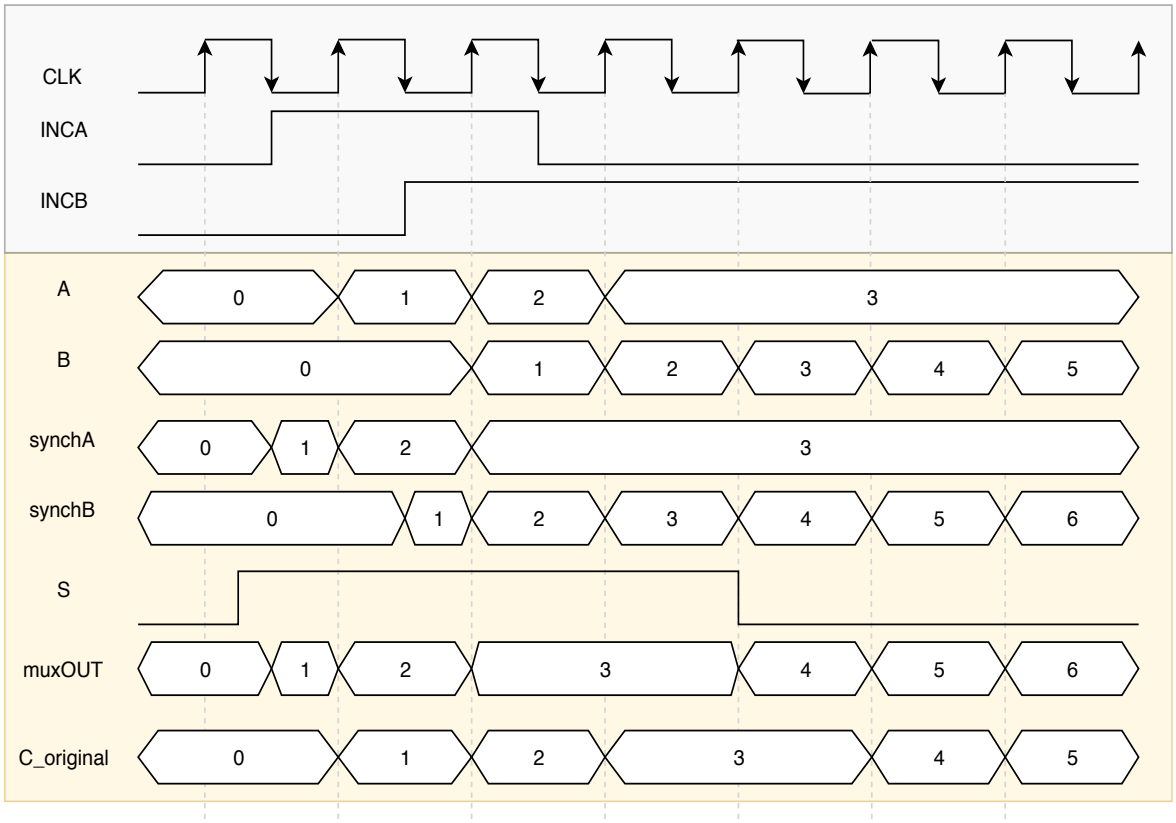

Figura 3.19: Timing Diagram dell'architettura originale mostrata in Figura 3.7

Effettivamente i timing diagram relativi all'uscita C non combaciano. Quando si applica la parallelizzazione, infatti, è fondamentale che le L repliche del circuito processino alternatamente campioni appartenenti allo stesso stream di dati, senza farne perdere il sincronismo e la coerenza reciproca.

Nel caso del circuito in esame, la perdita di coerenza è motivata dal fatto che il circuito originale presenta una sorta di *retroazione*, per cui una replica del datapath ha necessariamente bisogno del dato prodotto dall'incrementer della replica precedente per poter mantenere la correttezza logica del flusso di informazioni.

La soluzione proposta consiste nel memorizzare l'output di ogni incrementer direttamente nel registro d'ingresso relativo al datapath successivo (Figura 3.20).

Occorre però fare alcune considerazioni. Le due repliche del circuito lavorano ad una frequenza $f/2$, ma su fronti di clock contrapposti. Affinchè logicamente il circuito sia funzionante in modo corretto, è necessario che **anche con la nuova alimentazione** gli incrementer del *Blocco 1* producano in tempo (ovvero $@f$) il dato da memorizzare nei registri del *Blocco 2*.

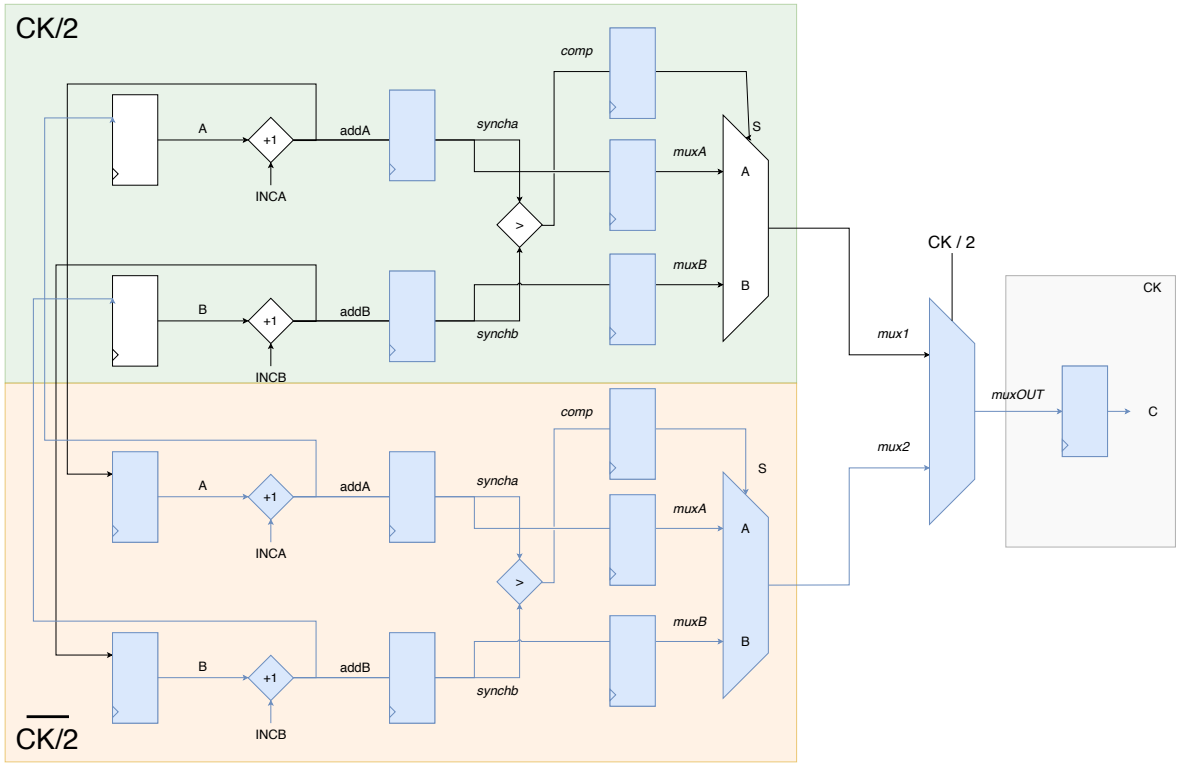

Figura 3.20: Architettura parallelizzata (L=2) e pipelinata - Fixata

Innanzitutto si riportano in Tabella 3.28 i nuovi parametri del circuito operante alla massima frequenza, come già fatto in precedenza.

Circuito Parallelizzato e Pipelinato (high-speed)		
Critical Path	$max[T_{REG}+T_{INC}, T_{REG}+T_{COMP}, T_{REG}+T_{MUX}]$	86 ns
Max Frequency	$1/T_{CP}$	11.63 MHz
Area	$15A_{REG}+2A_{REG,1bit}+4A_{INC}+2A_{COMP}+3A_{MUX}$	6562 µm^2
Power ($@f_{MAX}$)	$(15P_{REG}+4P_{INC}+3P_{COMP}+3P_{MUX})\cdot f_{MAX}/5\,\text{MHz}$	66.36 µW

Tabella 3.28: Circuito **parallelizzato e pipelinato** operante alla sua **massima frequenza**

A questo punto è possibile rallentare i due blocchi di un fattore 2:

$$T_{LOW,FIX} = 2 \cdot T_{MAX,ORIG} = 2 \cdot 140 = \mathbf{280\ ns}$$

$$T_{LOW,FIX} = T_{MAX,FIX} \cdot \frac{0.75u}{u-0.25} \Rightarrow 2 \cdot 140 = 86 \cdot \frac{0.75u}{u-0.25} \Rightarrow \mathbf{u = 0.325}$$

Ciò che è necessario verificare è che, con un'**alimentazione di 0.325 V**, il path costituito da registro ed incrementer riesca a completare il calcolo in meno di 140 ns, ovvero prima che i registri del *Blocco 2* effettuino il campionamento (continuando a considerare trascurabili tempi di *setup* e *hold* dei Flip-Flop).

$$T_{(INC+REG)}(@0.325\,\text{V}) = (40+2) \cdot \frac{0.75 \cdot 0.325}{0.325-0.25} = \mathbf{136.5\ ns} < 140\,\text{ns}$$

Pertanto, **con un parallelismo di ordine $L = 2$ non si hanno problemi**. Se invece si aumentasse il numero di repliche ad esempio ad $L = 3$ si otterrebbe un valore più basso della tensione di alimentazione:

$$T_{LOW,FIX} = 3 \cdot T_{MAX,ORIG} = 3 \cdot 140 = \mathbf{420\,ns}$$
$$T_{LOW,FIX} = T_{MAX,FIX} \cdot \frac{0.75u}{u - 0.25} \Rightarrow 3 \cdot 140 = 86 \cdot \frac{0.75u}{u - 0.25} \Rightarrow \mathbf{u = 0.295}$$

In questo caso il path costituito da registro ed incrementer non riesce a produrre il dato in tempo:

$$T_{(REG)}(@0.295\,\mathrm{V}) = (2) \cdot \frac{0.75 \cdot 0.295}{0.295 - 0.25} = \mathbf{9.83\,ns}$$
$$T_{(INC)}(@0.295\,\mathrm{V}) = (40) \cdot \frac{0.75 \cdot 0.295}{0.295 - 0.25} = \mathbf{196.67\,ns}$$
$$Total\_Time = \mathbf{206.5\,ns} > 140\,\mathrm{ns}$$

Affinchè tutto funzioni a dovere, l'incrementer deve riuscire a completare il calcolo in $140 - 9.83 = 130.17\,\mathrm{ns}$. Una possibile soluzione potrebbe essere quella di usare un approccio **Multi-Voltage Supply**, ovvero adoperare una tensione di alimentazione differente per gli elementi circuitali relativi agli incrementer; è possibile calcolare tale tensione nel modo seguente:

$$T_{INC} = 40 \cdot \frac{0.75 \cdot u}{u - 0.25} < 130.17\,\mathrm{ns} \Rightarrow u > 0.3248 \Rightarrow V_{INC} > \mathbf{0.3248\,V}$$

In sintesi, con una parallelizzazione di livello $L = 3$ è possibile lavorare con un'alimentazione di **0.295 V**, a patto di alimentare gli incrementer con un'alimentazione di **almeno 0.325 V** affinchè non ci siano errori logici. Infatti:

$$T_{(REG)}(@0.295\,\mathrm{V}) = (2) \cdot \frac{0.75 \cdot 0.295}{0.295 - 0.25} = \mathbf{9.83\,ns}$$
$$T_{(INC)}(@0.325\,\mathrm{V}) = (40) \cdot \frac{0.75 \cdot 0.325}{0.325 - 0.25} = \mathbf{130\,ns}$$
$$Total\_Time = \mathbf{139.83\,ns} < 140\,\mathrm{ns}$$

È possibile effettuare lo stesso ragionamento all'aumentare dell'ordine di parallelismo, dove sarà necessaria una tensione di alimentazione riservata agli incrementer via via maggiore per evitare errori logici, riuscendo a computare il dato entro 140 ns.

Naturalmente l'utilizzo di una $V_{DD,INC}$ crescente potrebbe limitare il risparmio di potenza dinamica; inoltre l'aumento dell'ordine di parallelismo farebbe inevitabilmente crescere il numero di dispositivi e di conseguenza il contributo dovuto al leakage, pertanto occorre valutare caso per caso se il risparmio complessivo che si riesce ad ottenere sia effettivamente quello sperato.

Per una maggiore completezza si è simulata con approccio pen-and-paper l'architettura fixata proposta in Figura 3.20; il timing diagram mostrato in Figura 3.21 conferma che effettivamente il nuovo circuito ha lo stesso funzionamento logico di quello originale, a patto di accettare una latenza iniziale dovuta all'introduzione di registri di pipeline.

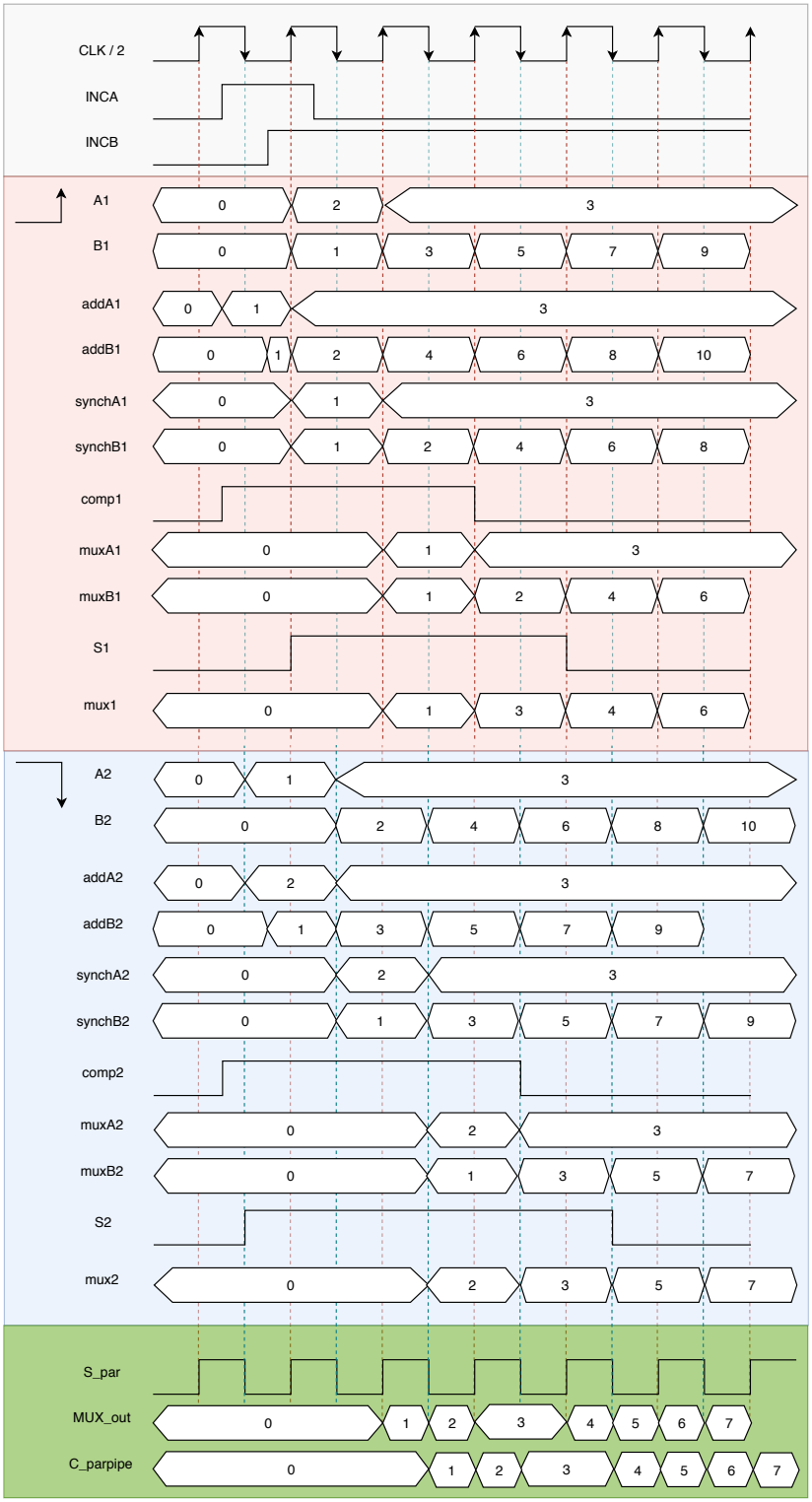

Figura 3.21: Timing Diagram dell'architettura parallelizzata (L=2) e pipelinata - Fixata, mostrata in Figura 3.20

CAPITOLO 4

Bus encoding

Durante questo quarto laboratorio il lavoro si è incentrato sull'analisi dei consumi dovuti ai bus, che, com'è noto, rappresentano una larga fetta della spesa energetica sui sistemi digitali.

Si è preso come riferimento il bus **non-encoded** ed è stato messo a confronto con quattro delle codifiche studiate durante il corso ideate per ridurne la switching activity, ovvero il bus **invert** e il **transition based** nel caso di trasmissione dati, e il **gray encoding** e il **T0** per la trasmissione di indirizzi. In un primo momento si sono analizzati i consumi legati esclusivamente alle transizioni sul bus, in seguito si è preso in considerazione anche il dispendio energetico per la codifica e la decodifica legato relativamente all'encoder e al decoder tramite il metodo SAIF visto nel laboratorio precedente.

4.1 Simulation

Per la presente simulazione sono stati forniti i file `.vhd` di tutte le codifiche ad esclusione della **T0** che abbiamo implementato e discuteremo successivamente.

Tramite il file **tb_encdec.v**, anch'esso fornitoci assieme ai file `.vhd`, abbiamo simulato i consumi sul bus prima nel caso di invio di indirizzi (**ADDRESS**) partendo dal valore d'ingresso 0 per poi aumentare progressivamente con alcuni salti, e successivamente di dati (**DATA**) sfruttando questa volta il file *rndin.txt* fornitoci per simularne degli ingressi casuali, il tutto sempre su 8 bit.

All'interno di tale file si può notare che, al contrario di quanto ci si aspetterebbe in un sistema complesso, non vi è una bassa percentuale di bit ad '1' da trasmettere, questo farà sì che una codifica come quella della **transition based** che analizzeremo più avanti, pensata per la trasmissione dati, sia molto meno efficiente.

4.1.1 Non-encoded

Come anticipato il bus **non-encoded** sarà utilizzato da qui in avanti come riferimento per le successive codifiche.

Come ci si aspettava il Toggle Count dell'encoder, del bus e del decoder sono identici in quanto non vi è una codifica che modifica il segnale trasmesso, l'unica differenza risiede nella latenza di un colpo di clock tra i diversi stadi come mostarto in Figura 4.1.

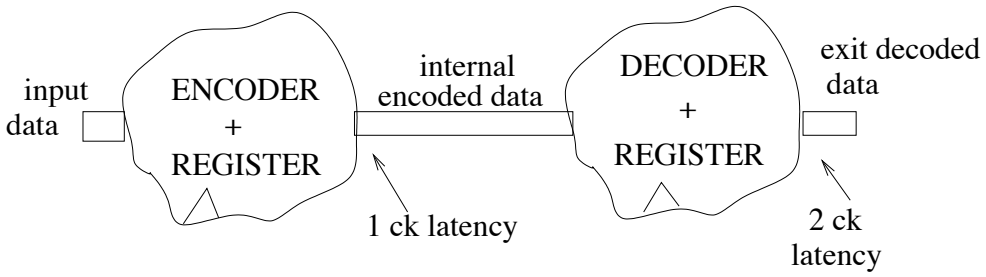

Figura 4.1: Struttura del bus e relativi ritardi

4.1.2 Bus-Invert

La tecnica **bus-invert** fà parte delle codifiche ridondanti e necessita di un bit aggiuntivo sul bus per sfruttare la distanza di Hamming e far commutare ad ogni colpo di clock massimo la metà delle linee del bus.

Tale codifica prevede, dati N bit sul bus di trasmissione originario, di analizzare il dato da trasmettere con quello precedentemente inviato, nel caso i bit da commutare siano meno di $N/2$ questi vengono semplicemente invertiti e il dato inviato regolarmente, in caso contrario il bit di inversione viene attivato e vengono commutati solo i bit per creare la versione complementare del dato da trasmettere.

Per implementare la commutazione è necessario l'utilizzo dell'operatore XOR come prevede l'Equazione 4.1 per il calcolo della distanza di Hamming.

$$d_{Hamming} = \sum_{i=0}^{N-1} D^t(i) \oplus D^{t-1}(i) \tag{4.1}$$

In cui $D^t(i)$ rappresenta il bit i-esimo del dato trasmesso all'istante t e $D^{t-1}(i)$ il medesimo bit del dato trasmesso all'istante $t-1$.

Ma come si è precedentemente ribadito si effettua solo se la condizione in Equazione 4.2 è rispettata.

$$d_{Hamming} > N/2 \tag{4.2}$$

In fase di codifica sarà necessario dell'hardware aggiuntivo per effettuare una somma e successivamente una comparazione.

All'interno del file **businvbeh.vhd** la distanza di Hamming è salvata all'interno della variabile `hamdist` e nel caso questa raggiunga un valore maggiore a 4 il bit di inversione viene posto ad '1' e il dato negato viene inviato sul bus.

4.1.3 Transition Based

La tecnica **transition based** fà parte invece delle codifiche non rindondanti e, come anticipato, ha la sua massima efficacia durante una trasmissione dati con una bassa percentuale di bit ad '1' da inviare. Questa codifica prevede di far commutare esclusivamente le linee del bus che assumono il valore logico '1' e lasciare "congelate" quelle che invece devono trasmettere uno '0', ciò è facilmente implementabile, nuovamente, grazie all'operatore logico XOR secondo la Tabella 4.1.

b(t)	B(t-1)	B(t)
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 4.1: Codifica Transition Based

In cui $b(t)$ rappresenta il singolo bit del dato da trasmettere, $B(t-1)$ il dato precedentemente inviato sul bus e $B(t)$ il nuovo dato codificato.

Considerando che statisticamente i bit a '0' rappresentano il 90% dei dati trasmessi ciò significa ridurre enormemente l'energia spesa sul bus grazie all'aggiunta di un flip-flop e una porta XOR per ogni linea nell'encoder e nel decoder, come mostrato in Figura 4.2, per effettuare le comparazioni con il bit corrispondente del dato precedentemente inviato.

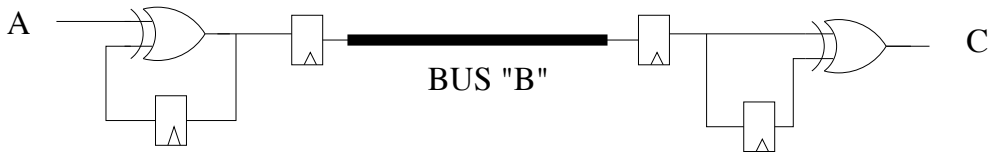

Figura 4.2: Implementazione circuitale della codifica Transition Based

All'interno del file **transbased.vhd** il processo che si occupa dell'encoding memorizza sul fronte di clock positivo l'operazione `buss xor A` direttamente sulla variabile `buss`, nel decoding invece l'operazione di XOR tra `CENC` e `CENCOLD` viene memorizzata sulla variabile temporanea `CDECTMP`, collegata al segnale di uscita `C`.

4.1.4 Gray

La tecnica **gray** fà parte delle codifiche non rindondanti ed è quindi facilmente compatibile con gli standard; tale tecnica prevede, nel caso di invio di una sequenza di dati progressiva, di ridurre la distanza di Hamming al valore 1 tra il nuovo dato e il precedente in modo da ridurre al minimo le commutazioni. È evidente come questa tecnica si presti particolarmente bene all'invio di indirizzi i quali sono spesso inviati in successione per svariati cicli di clock riducendo sensibilmente la switching activity.

Anche in questo terzo caso l'operatore XOR permette una semplice implementazione dell'encoder e del decoder, infatti collegando le corrispettive porte logiche seguendo lo schema riportato in Figura 4.3 è possibile notare come nel caso di dati progressivi solo una linea del bus commuti. Si noti che una tale implementazione non comporta particolari problemi durante la codifica, ma in fase di decodifica ogni porta XOR deve attendere il risultato della porta precedente creando una catena di ritardi che devono essere nel complesso minori del singolo ciclo di clock per non propagare errori a valle del bus.

Un esempio è riportato in Tabella 4.2 in cui le colonne $b(t) \cdots b(t+4)$ rappresentano i dati in ingresso che vengono incrementati progressivamente, le colonne $B(t) \cdots B(t+4)$ rap-

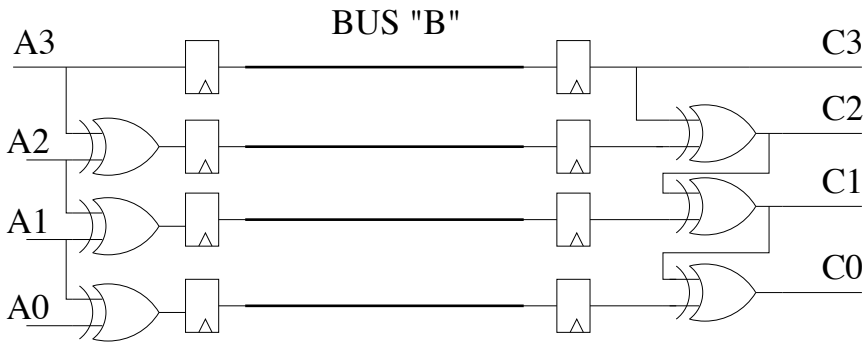

Figura 4.3: Implementazione circuitale della codifica Gray a 4 bit

presentano invece i dati codificati e in grassetto sono evidenziati i singoli bit che commutano istante per istante.

b(t)	b(t+1)	b(t+2)	b(t+3)	b(t+4)	B(t)	B(t+1)	B(t+2)	B(t+3)	B(t+4)
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	**1**
0	0	1	1	0	0	0	**1**	1	1
0	1	0	1	0	0	**1**	1	**0**	0

Tabella 4.2: Esempio di codifica Gray su 4 bit

All'interno del file **grayencoder.vhd** la codifica viene facilmente gestita da un ciclo `for` che effettua l'operazione XOR tra il bit `i` e il bit `i+1` del dato in ingresso, per la decodifica invece viene istanziata la variabile `C_tmp` che viene ricorsivamente aggiornata sfruttando il precedente valore di `C_tmp` e il dato ricevuto dal bus BTMP, il tutto ovviamente iterando sui singoli bit.

4.1.5 T0

La tecnica **T0** appartiene alla famiglia delle codifiche ridondanti e si presta particolarmente bene all'invio di indirizzi successivi in quanto, sfruttanto il bit di incremento, "congela" il bus arrivando quasi ad azzerare le commutazioni, nel caso ottimale, grazie alla seguente funzione:

$$(B^{(t)}, INC^{(t)}) = \begin{cases} (B^{(t-1)}, 1) & \text{se } b^{(t)} = b^{(t-1)} + 1 \\ (b^{(t)}, 0) & \text{altrimenti} \end{cases} \tag{4.3}$$

in cui $b^{(t)}$ rappresenta il dato in ingresso, $B^{(t)}$ il dato codificato sul bus e $INC^{(t)}$ il bit di incremento, tutti all'istante t.

L'implementazione del codificatore in questo caso prevede un registro di memoria per la memorizzazione del dato trasmesso non codificato, il quale deve essere confrontato tramite un comparatore ad ogni colpo di clock con il nuovo dato in ingresso a cui, tramite un incrementer, viene sommato un '1':

- nel caso siano uguali il bit di incremento viene attivato dal comparatore, il dato sul bus non viene modificato, sfruttando come enable del registro del dato codificato il risultato negato dello stesso comparatore e infine viene aggiornato il registro del dato trasmesso non codificato;

- nel caso opposto invece è necessario inviare un nuovo dato sul bus, il comparatore genera un risultato negativo che disattiva il bit di incremento e attiva l'enable del registro del dato codificato, il quale in questo caso memorizza e successivamente trasmette il dato in ingresso, infine, anche qui, viene aggiornato il registro del dato trasmesso non codificato.

All'interno del codificatore invece sarà presente un multiplexer comandato dal bit di incremento per selezionare l'uscita del bus o il risultato derivato da un circuito formato da un registro ed un incrementer utilizzato per decodificare il dato ricevuto. Tale registro si trova a valle di un secondo multiplexer il quale, sempre pilotato dal bit di incremento, salverà al suo interno il nuovo dato proveniente dal bus ($INC = 0$) o il risultato in uscita dall'incrementer ($INC = 1$).

Il file **t0encdec.vhd** è riportato nella sua interezza in Appendice D.2, in Figura 4.4 invece è riportato uno spezzone della simulazione esplicativa del comportamento della codifica T0 all'inizio del suo funzionamento e in Figura 4.5 nel caso di salto tra gli indirizzi trasmessi.

Figura 4.4: Simulazione della codifica T0 durante la trasmissione di indirizzi all'inizio del suo funzionamento

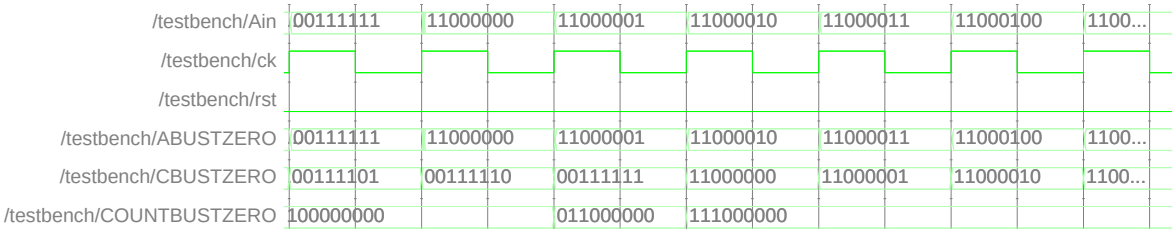

Figura 4.5: Simulazione della codifica T0 durante la trasmissione di indirizzi in presenza del primo *jump*

Comparison

Per effettuare la comparazione tra i consumi di potenza delle differenti codifiche è stato aggiornato il file **tb_encdec.v** aggiungendo i `component` e gli `assign` mancanti. Le differenti codifiche sono state simulate ognuna per 100 000 ns, prima nel caso **ADDRESS** e successivamente in quello **DATA** con i risultati riportati rispettivamente in Tabella 4.3 e in Tabella 4.4.

ADDRESS · Com'è possibile notare dalla Tabella 4.3 la miglior codifica in questo caso specifico risulta essere la **T0**, la quale riduce di quasi il 99% il numero di commutazioni, ciò era prevedibile data la quasi totale linearità dei dati in ingresso.

La codifica **gray** è anch'essa vantaggiosa dal punto di vista energetico considerando che arriva quasi a dimezzare le commutazioni rispetto al caso di riferimento.

Nonostante la codifica **bus-invert** sia solitamente utilizzata per la trasmissione dati, essa riduce anche in questo caso, seppur leggermente, il numero totale di commutazioni. È però da tenere presente che in questo esempio non sono tenuti in considerazione i consumi dell'encoder e del decoder, i quali se considerati supererebbero di certo quelli del caso **non-encoded**.

Infine la codifica **transition based**, non avendo una prevalenza di zeri in trasmissione, non porta alcun miglioramento al toggle count, anzi ne causa un considerevole peggioramento.

Node	UBUSNORM	UBUSINV	UBUSTRAN	UBUSGRAY	UBUSTZERO
b(8)	-	624	-	-	119
b(7)	97	527	4816	97	29
b(6)	118	506	3776	55	24
b(5)	312	312	4992	194	0
b(4)	624	0	4992	312	0
b(3)	1249	625	5000	625	0
b(2)	2499	1875	5000	1250	0
b(1)	4999	4375	5000	2500	0
b(0)	9999	9375	5000	5000	0
Total	19897	18219	38576	10033	252

Tabella 4.3: Toggle Count delle diverse codifiche nel caso **ADDRESS**

DATA · La Tabella 4.4 mostra come la trasmissione dati sia tendenzialmente molto più dispendiosa in termini energetici rispetto al caso precedente, si parla infatti nel caso di riferimento in oggetto di un raddoppio delle commutazioni. Si noti inoltre come in questo caso la codifica migliore sia la **bus-invert** nonostante questa porti una riduzione delle commutazioni inferiore al 20%.

In un caso reale, nella quale gli zeri rappresentano la gran parte del dato trasmesso, la codifica **transition based** avrebbe prodotto dei risparmi mediamente intorno al 90%, ma in questo caso, in cui la ripartizione dei bit ad '1' e a '0' all'interno dei singoli dati è abbastanza omogenea questa non porta alcun vantaggio rispetto al caso di riferimento.

Allo stesso modo la codifica **gray** e la **T0**, ideali per la trasmissione di dati dal valore crescente, risultano inutili nell'invio di dati casuali e non portano alcun beneficio in confronto al caso **non-encoded**, anzi considerando il consumo dell'hardware aggiuntivo porterebbero un sensibile aumento dell'energia spesa.

Node	UBUSNORM	UBUSINV	UBUSTRAN	UBUSGRAY	UBUSTZERO
b(8)	-	3650	-	-	72
b(7)	4974	3576	5003	4974	4974
b(6)	5021	3611	4946	5087	5021
b(5)	4987	3617	4938	4952	4987
b(4)	4972	3640	4997	4981	4972
b(3)	4959	3613	5125	4937	4957
b(2)	4965	3601	4989	5056	4963
b(1)	5023	3639	5000	4962	5001
b(0)	5060	3722	4994	5075	5028
Total	39961	32669	39992	40024	39975

Tabella 4.4: Toggle Count delle diverse codifiche nel caso **DATA**

4.2 Synthesis

Come precedentemente accennato le simulazioni viste finora con *Modelsim* tengono conto esclusivamente del numero di commutazioni sul bus indirizzi/dati senza considerare il dispendio energetico causato dai blocchi logici per la codifica e la decodifica.

Come fatto per il laboratorio precedente si sfrutteranno i SAIF file generati da *Synopsys* per avere una stima più accurata dei consumi, per fare ciò sono stati utilizzati gli script forniti assieme ai file `.vhd`.

Dopo aver opportunamente modificato ed eseguito lo script `create_sdf.scr`, riportato in Appendice D.1, tramite *Synopsys* per la generazione dei file `.ddc`, `.v` e `.sdf` è stato avviato *Modelsim* da cui, in seguito alla compilazione del testbench e dei file `.v` appena generati, è stato eseguito il file `fill_forward.scr` per ottenere il file `.vcd` che tramite il comando apposito è stato convertito infine in un file `.saif` contenente le informazioni associate ad ogni segnale.

Successivamente è stato eseguito il file `backward_all.scr` all'interno di *Synopsys* che a sua volta richiama il file `backward.scr` per ognuna delle codifiche implementate che sfrutta il file `.saif` e compila il design con un carico di 1 fF più i tre diversi carichi dichiarati nel file `definitions.scr`.

Tale procedimento è stato effettuato sia per il caso **ADDRESS** che per il caso **DATA**, ottenendo così un totale di 40 file di report, i cui risultati sono sintetizzati rispettivamente in Tabella 4.5 e Tabella 4.6, tali risultati saranno discussi successivamente.

Codifica	1 fF	10 fF	50 fF	100 fF
UBUSNORM	10.158 µW	11.387 µW	16.222 µW	23.240 µW
UBUSINV	19.089 µW	22.656 µW	27.074 µW	33.441 µW
UBUSTRAN	27.639 µW	30.165 µW	39.570 µW	53.325 µW
UBUSGRAY	9.436 µW	10.056 µW	12.493 µW	16.009 µW
UBUSTZERO	20.235 µW	21.900 µW	21.937 µW	21.984 µW

Tabella 4.5: Consumi di potenza dinamica nel caso **ADDRESS** per le diverse codifiche al variare del carico

Codifica	1 fF	10 fF	50 fF	100 fF
UBUSNORM	14.562 µW	17.030 µW	26.737 µW	40.872 µW
UBUSINV	25.915 µW	29.247 µW	37.173 µW	48.620 µW
UBUSTRAN	26.629 µW	29.114 µW	38.929 µW	53.083 µW
UBUSGRAY	18.136 µW	20.605 µW	30.332 µW	44.487 µW
UBUSTZERO	33.919 µW	38.783 µW	48.193 µW	62.264 µW

Tabella 4.6: Consumi di potenza dinamica nel caso **DATA** per le diverse codifiche al variare del carico

Per ottenere tali tabelle è stato necessario modificare il file `save_power_report.scr` aggiungendo al comando **report_power** l'opzione **-flat** per poter ottenere le informazioni sulla switching activity di tutti gli oggetti del design, diversamente nel caso del T0 non si sarebbe ottenuto un risultato valido (N/A) per tutti i nodi del design.

Prima di ottenere tali risultati sono state effettuate diverse sintesi in quanto inizialmente i consumi energetici della codifica **T0** non rispecchiavano quelli che ci si sarebbe potuti aspettare, difatti nel caso ADDRESS si è ottenuto un consumo di potenza dinamica pari a 39.641 µW con un carico da 1 fF e risultati tra i 26.381 µW e i 26.735 µW nei restanti tre casi.

Essendo la **T0** una codifica che riduce al minimo le commutazioni in caso di trasmissione di indirizzi ci si aspettava un consumo pressocchè costante, dato principalmente dal consumo dovuto al codificatore e al decodificatore, questo ci ha portato a modificare la risoluzione per la simulazione, all'interno del file `fill_forward.scr`, portandola dopo vari tentativi ad 1 ns. Tale modifica ha prodotto i risultati già analizzati, i quali risultano essere estremamente più congruenti con quanto ci si aspettava.

Le restanti codifiche mostrano una leggera variazione rispetto a quanto detto precedentemente durante l'analisi della switching activity.

Nel caso **ADDRESS**, graficato in Figura 4.6, i consumi del bus di riferimento crescono linearmente con il carico e la codifica **gray**, che presentava una switching activity dimezzata è sì più prestante in termini energetici ma, specialmente per carichi limitati, non raggiunge il dimezzamento del consumo di potenza. Le codifiche **bus-invert** e **transion based** sono delle soluzioni per nulla ottimali in questo caso, la prima a causa dell'aggiunta di hardware per

il calcolo della Distanza di Hamming, la seconda soprattutto a causa dell'elevata switching activity.

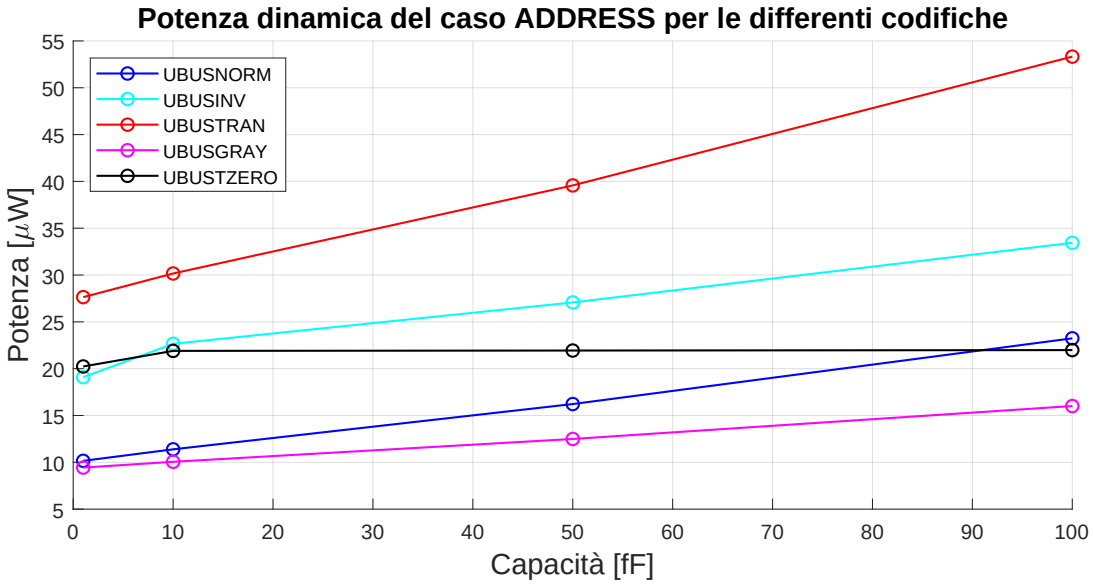

Figura 4.6: Trasposizione grafica dei consumi di potenza dinamica nel caso **ADDRESS** per le diverse codifiche al variare del carico

Nel caso **DATA**, graficato in Figura 4.7, si può notare come i consumi di tutte le codifiche aumentino all'aumentare del carico del bus. Come precedentemente analizzato nessuna delle codifiche porta una sostanziale diminuzione della switching activity a parte la codifica **bus-invert**, nonostante ciò il consumo associato al suo hardware la rende una soluzione peggiore sia della codifica **gray** che del caso di riferimento, che rappresenta la miglior soluzione in termini di potenza. Infine le codifiche **T0** e **transion based** risultano essere le più dispendiose essendo la prima ideata per la trasmissione di indirizzi e non di dati, la seconda utilizzata in un caso non ottimale, la loro switching activity infatti è pressochè identica al caso di riferimento, ma l'hardware aggiuntivo, specialmente nella T0, le rendono delle soluzioni da non prendere in considerazione per questo caso specifico.

Diversamente se la distribuzione dei bit di dato fosse stata sbilanciata verso lo '0' la codifica **transion based** sarebbe potuta essere una soluzione ottimale e la sua curva risulterebbe probabilmente ad di sotto di quella del caso di riferimento

È interessante analizzare anche il consumo causato dalle *correnti di leakage*, strettamente legato al quantitativo di hardware sintetizzato. In Tabella 4.7 sono riportate esclusivamente le potenze statiche delle differenti codifiche, senza distinzione tra il caso **ADDRESS** e il caso **DATA**, questo poichè non essendo tali potenze legate per definizione alla switching activity assumono un valore pressocchè identico.

Anche tali valori rispecchiano ciò che ci si aspettava, mentre il caso **non-encoded** riporta i consumi minori, la codifica **T0** risulta essere la più dispendiosa in termini di potenza, seguita subito dopo dalla **bus-invert** proprio a causa, come già detto, dell'hardware necessario per la loro implementazione. Infine si noti la differenza tra la codifica **gray** e la **transition**

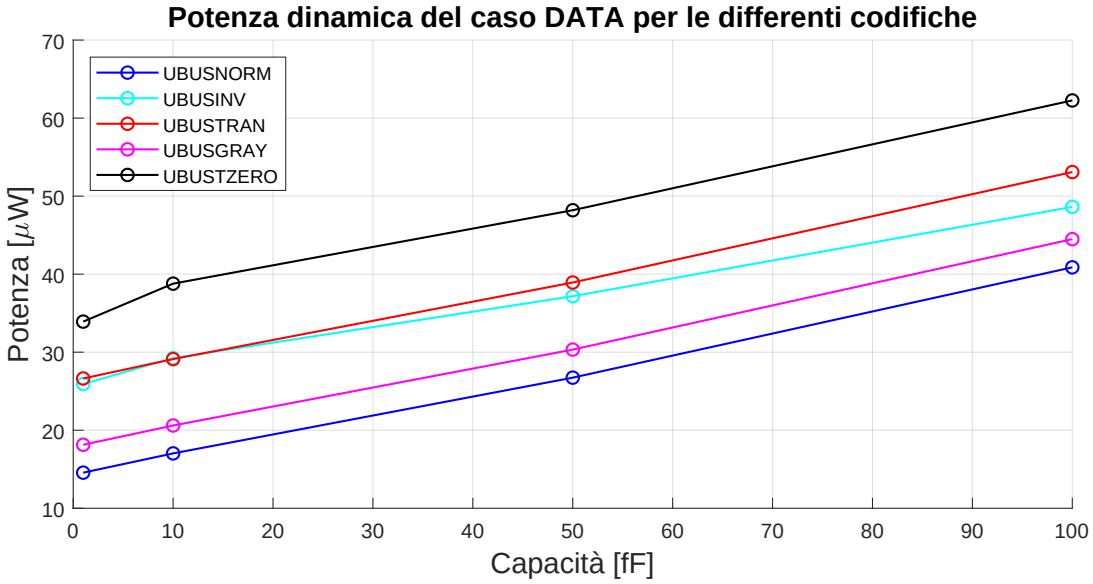

Figura 4.7: Trasposizione grafica dei consumi di potenza dinamica nel caso **DATA** per le diverse codifiche al variare del carico

Codifica	1 fF	10 fF	50 fF	100 fF
UBUSNORM	1.422 µW	1.422 µW	1.422 µW	1.758 µW
UBUSINV	3.203 µW	3.203 µW	3.221 µW	3.581 µW
UBUSTRAN	2.690 µW	2.744 µW	2.782 µW	3.019 µW
UBUSGRAY	1.916 µW	1.922 µW	1.922 µW	2.252 µW
UBUSTZERO	3.860 µW	3.931 µW	4.112 µW	4.482 µW

Tabella 4.7: Consumi di potenza statica nei casi **ADDRESS** e **DATA** per le diverse codifiche al variare del carico

based, le quali, come illustrato precedentemente, differiscono per la quantità di registri, la transition based infatti necessita di due flip-flop aggiuntivi per ogni linea del bus rispetto alla codifica gray, questo comporta la differenza di potenza riportata in tabella ed evidenziata in Figura 4.8.

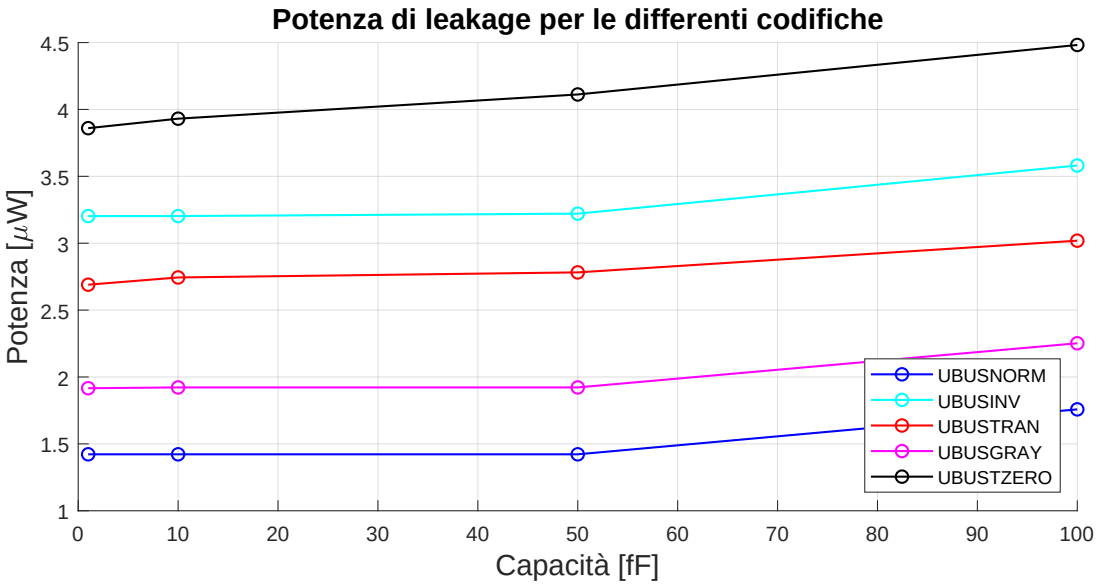

Figura 4.8: Trasposizione grafica dei consumi di potenza statica nei casi **ADDRESS** e **DATA** per le diverse codifiche al variare del carico

Data la moltitudine di simulazioni e sintesi lanciate è stato creato il file `master_script.sh` riportato in Appendice D.3 per automatizzare totalmente il processo.

CAPITOLO 5

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

5.1 Characterizing a library gate

Il punto di partenza di questa prova di laboratorio era la caratterizzazione SPICE di una cella **ND2HS**, ovvero una porta logica *NAND* a 2 ingressi ottimizzata dal punto di vista delle prestazioni. Sostanzialmente SPICE è una piattaforma che richiede una descrizione dettagliata del circuito a livello del transistor, in modo da poter derivare delle equazioni da risolvere per simulare a pieno il comportamento della netlist.

Come è noto, per una NAND in logica CMOS statica sono necessari 2 nMOS e 2 pMOS; in questo laboratorio verranno utilizzati i modelli *ENHSGP_BS3JU* e *EPHSGP_BS3JU* rispettivamente. La loro definizione è contenuta nel file `mos_bsim3_HS.lib`, nel quale sono specificati tutti i parametri caratteristici di un MOS transistor quali aree e perimetri di Drain e Source, tensione di soglia V_{TH} in diverse condizioni, ecc. Tali parametri sono poi utilizzati dal simulatore per derivare i valori delle *capacità intrinseche* utili nelle equazioni finali.

Tornando alla porta NAND, la sua descrizione è contenuta nella libreria `CMOS013.lib`, di cui di seguito è riportato un estratto:

```
.subckt ND2HS A B Z gnd vdd
    XMN0 net15 A gnd gnd ENHSGP_BS3JU W=0.640U L=0.130U
        +  AD=0.061P AS=0.305P PD=0.190U PS=2.870U
    XMN1 Z B net15 gnd ENHSGP_BS3JU W=0.640U L=0.130U
        +  AD=0.218P AS=0.061P PD=1.320U PS=0.190U
    XMP0 Z A vdd vdd EPHSGP_BS3JU W=0.770U L=0.130U
        +  AD=0.192P AS=0.471P PD=1.130U PS=4.130U
    XMP1 Z B vdd vdd EPHSGP_BS3JU W=0.770U L=0.130U
        +  AS=0.471P AD=0.192P PS=4.130U PD=1.130U
    C1 vdd gnd 0.142ff
    [...]
.ends ND2HS
```

In pratica viene creata una sorta di *black-box* chiamata ND2HS con 5 nodi (A, B, Z, gnd, vdd), nella quale sono istanziati e collegati opportunamente i 2 nMOS e i 2 pMOS necessari, compresa la definizione di alcuni parametri come W ed L. Volendo fare un'analogia con il VHDL, è come se l'architettura della cella ND2HS fosse stata definita in modo *structural*, utilizzando i MOS come *component*.

La traduzione grafica della netlist sopra riportata è mostrata in Figura 5.1.

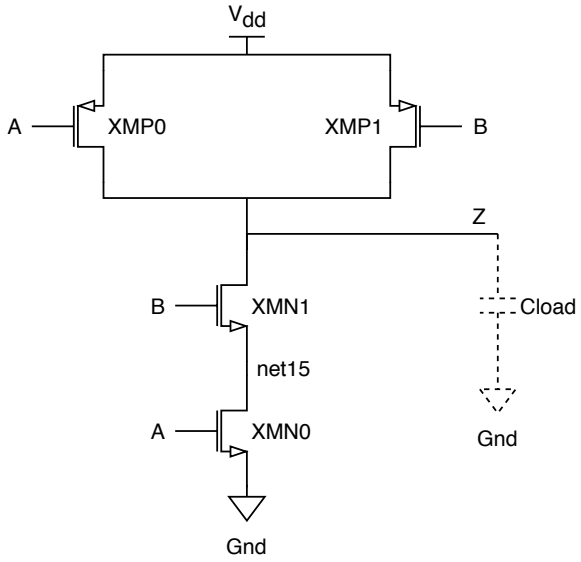

Figura 5.1: Rappresentazione grafica della cella **ND2HS** descritta nella libreria `CMOS013.lib`

Una volta definita la struttura del circuito si può passare alla definizione di alcune caratteristiche della simulazione, tra cui:

- **Variazione degli input**
  - A → costante a 1.2 V
  - B → distribuzione PieceWise Linear (PWL) riportata in Figura 5.2

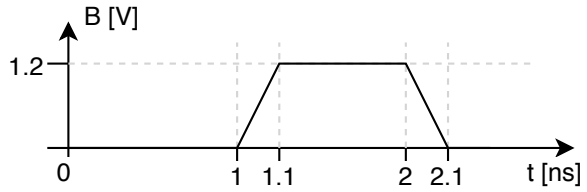

Figura 5.2: Distribuzione PWL dell'input B

- **Capacità di carico**, C → 10 fF
- **Tensione di alimentazione**, V_{DD} → 1.2 V

- **Misurazioni da effettuare**

  Nello script fornito erano già presenti i comandi per ottenere le seguenti misurazioni:

  – Tempo di salita
  – Tempo di discesa
  – Ritardo di propagazione in transizione HIGH → LOW in output

  La misurazione mancante era il ritardo di propagazione in transizione LOW → HIGH in output.

  Il comando **.measure** in SPICE richiede un evento *TRIGGER* e l'evento *TARGET*; nel caso specifico, essendo A fisso a V_{DD}, la transizione richiesta si può ottenere quando B passa da 1 → 0.

  La misurazione deve partire quando B raggiunge il 50% durante una transizione HL, e terminare quando l'uscita raggiunge il 50% in una transizione LH. Il comando finale che è stato aggiunto è il seguente:

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

- **Risoluzione e durata della simulazione**, 1 ps e 3 ns nella fattispecie.

A questo punto la simulazione può essere lanciata tramite **Eldo**, che fornisce i risultati riportati in Tabella 5.1.

Misurazioni sulla cella **ND2HS**	
Power Dissipation	6.7908 nW
Rising Time	89.303 ps
Falling Time	74.319 ps
Delay H → L	48.993 ps
Delay L → H	56.490 ps

Tabella 5.1: Risultati delle misurazioni sulla cella **ND2HS** ottenuti tramite comando *.measure*

In Figura 5.3 sono riportate le forme d'onda relative agli input e la corrispondente variazione dell'output.

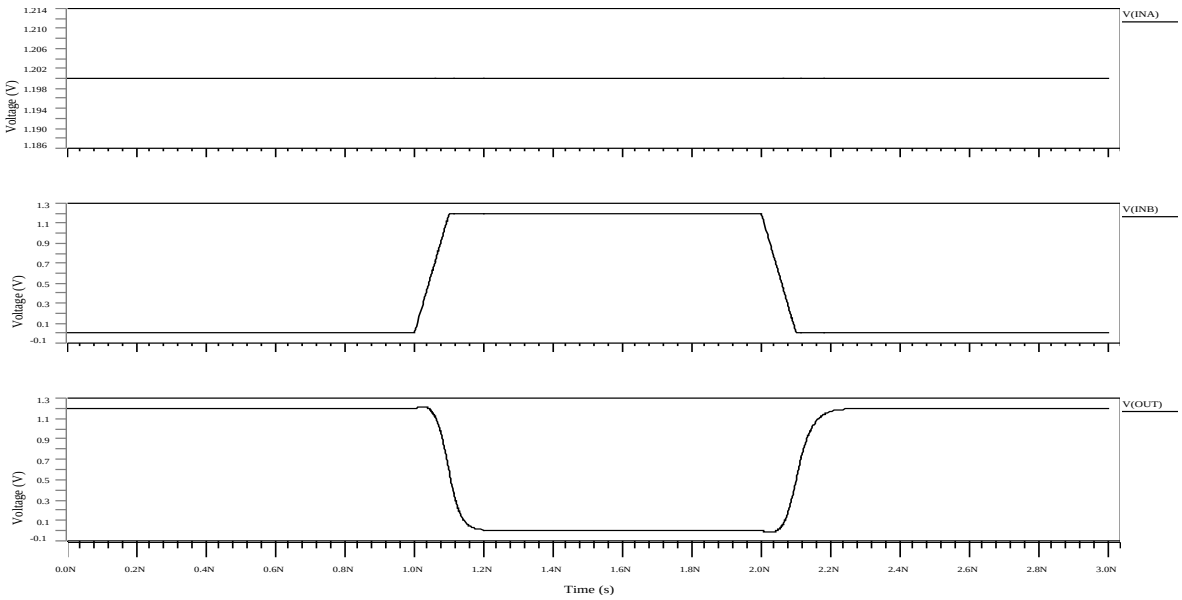

Figura 5.3: Forme d'onda relative alla variazione degli input A e B della cella **ND2HS** e corrispondente variazione dell'output

Inoltre, per avere una conferma visuale di quando ottenuto tramite comando *.measure*, è possibile effettuare misurazioni direttamente sulle forme d'onda mediante l'utilizzo di cursori. Effettivamente tutte le misurazioni riportate in Tabella 5.1 vengono confermate da quelle manuali riportate in Figura 5.4, Figura 5.5, Figura 5.6 e Figura 5.7.

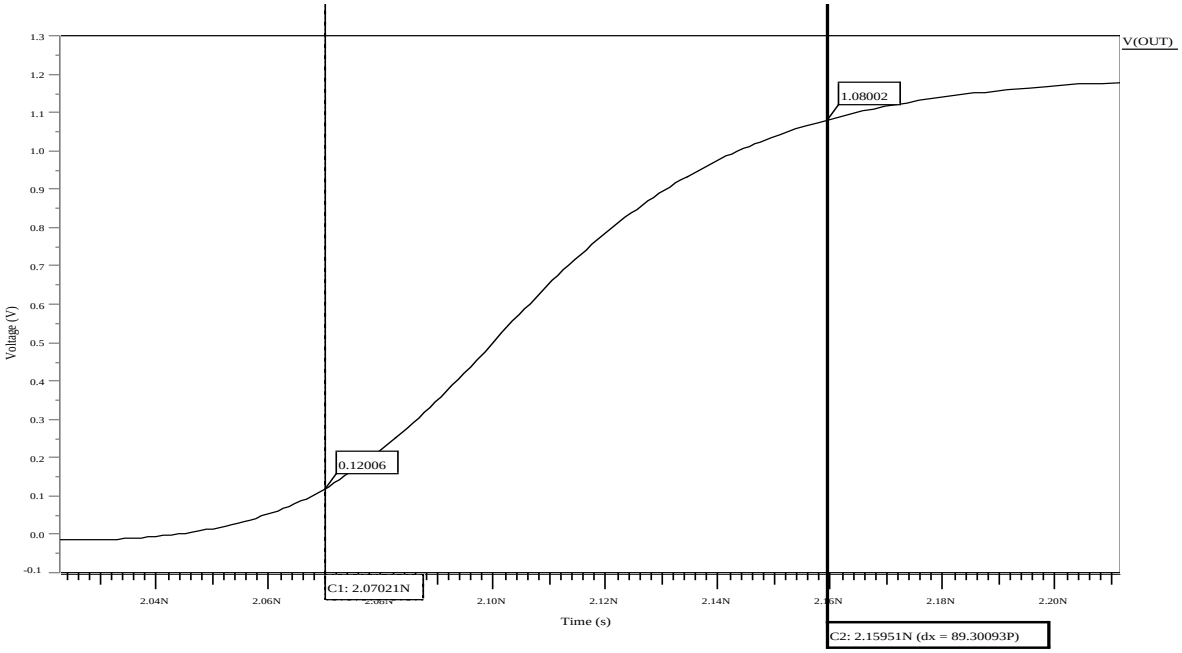

Figura 5.4: Misurazione del **tempo di salita** fatta manualmente con **ezwave**

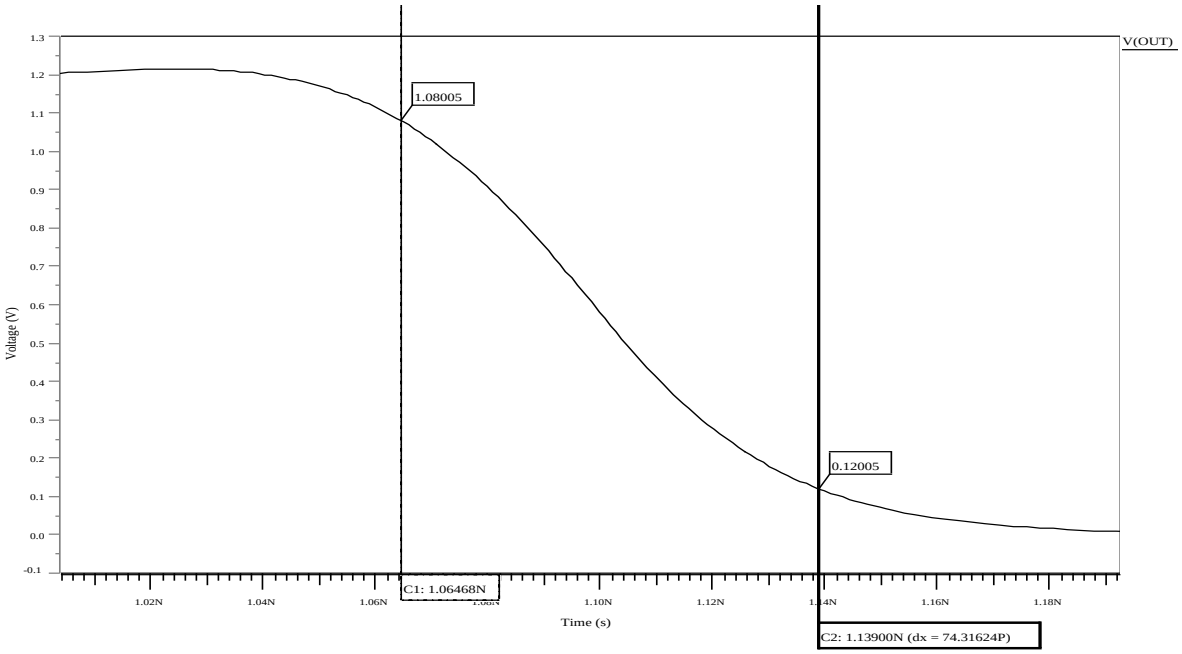

Figura 5.5: Misurazione del **tempo di discesa** fatta manualmente con **ezwave**

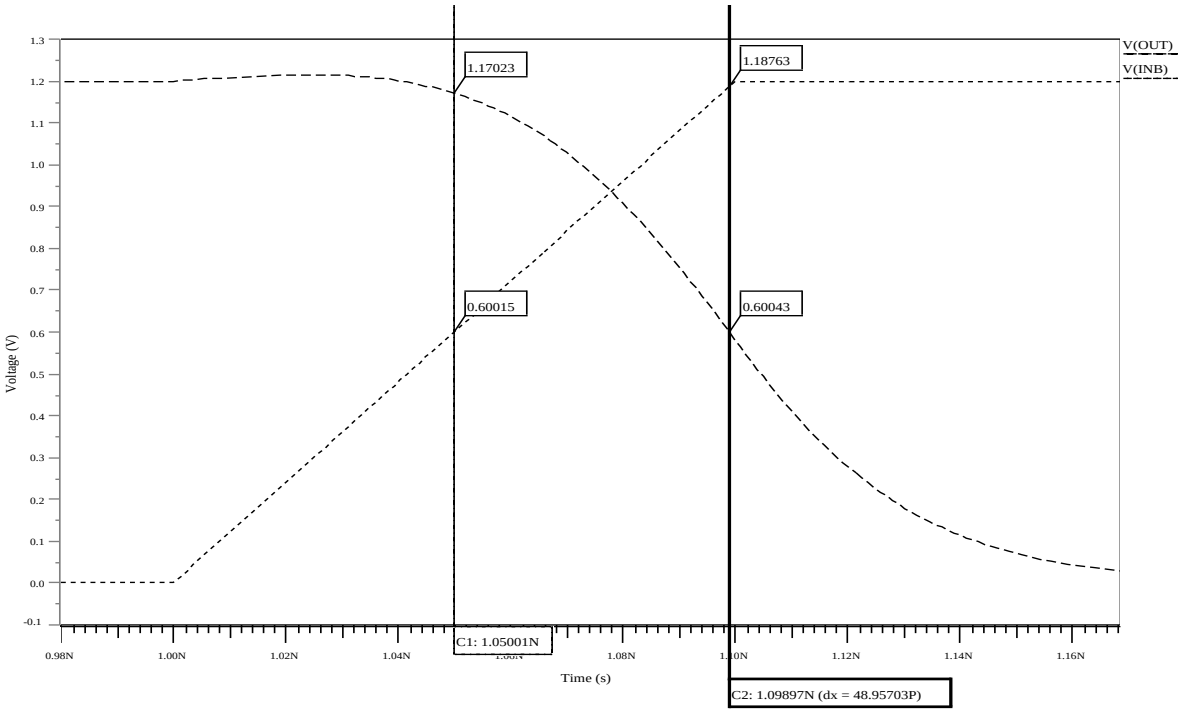

Figura 5.6: Misurazione del **tempo di propagazione in transizione HIGH → LOW** in output fatta manualmente con **ezwave**

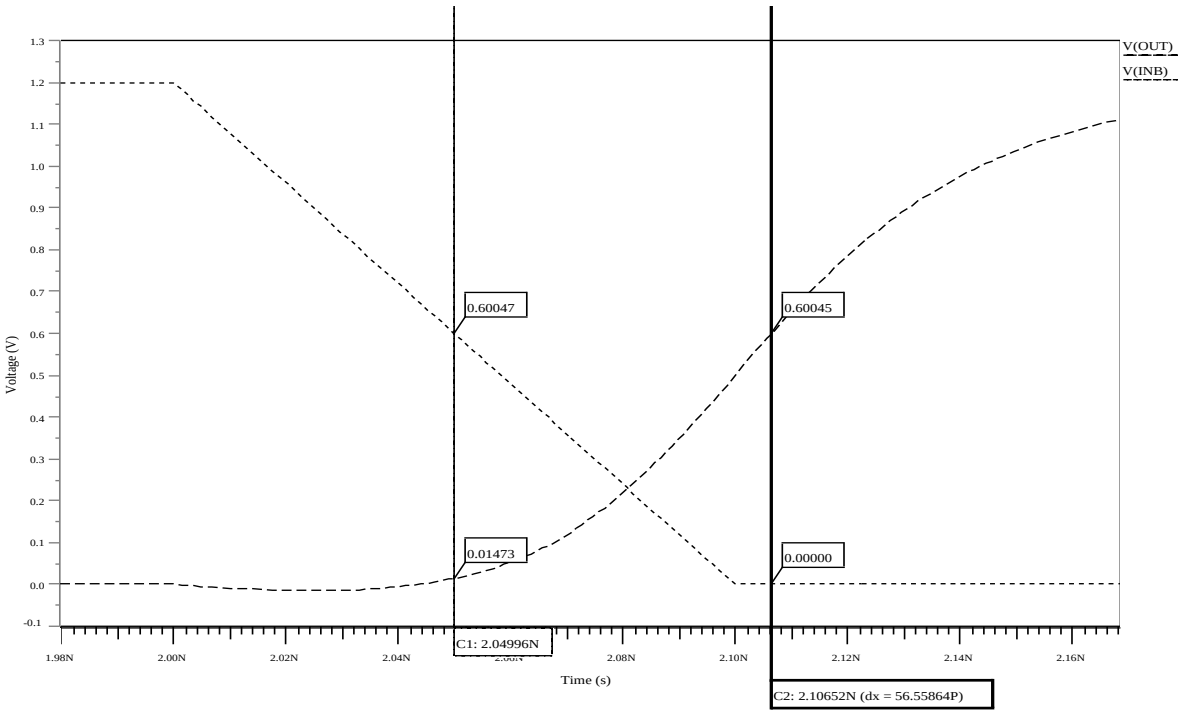

Figura 5.7: Misurazione del **tempo di propagazione in transizione LOW → HIGH** in output fatta manualmente con **ezwave**

5.1.1 Measuring the threshold voltage

Come richiesto, sono state fatte anche le misurazioni in DC sulle tensioni di soglia V_{TH} dei 4 transistor presenti nella cella; per una maggiore completezza d'analisi sono state simulate tutte le 4 combinazioni di input A-B. I risultati sono riportati in Tabella 5.2.

	A = 0 *B = 0*	*A = 0* *B = 1*	*A = 1* *B = 0*	*A = 1* *B = 1*
$V_G(N0)$	0.0	0.0	1.2	1.2
$V_G(N1)$	0.0	1.2	0	1.2
$V(net15)$	4.4758E-02	1.061	4.0840E-06	5.9826E-06
$V(LOAD)$	1.2	1.2	1.2	1.1965E-05
$V_{TH}(N0)$	3.1217E-01	2.7719E-01	3.1371E-01	3.1371E-01
$V_{TH}(N1)$	2.7935E-01	4.1784E-01	2.7241E-01	3.1371E-01
$V_{TH}(P0)$	2.4712E-01	2.4712E-01	2.4712E-01	2.2499E-01
$V_{TH}(P1)$	2.4712E-01	2.4712E-01	2.4712E-01	2.2499E-01

Tabella 5.2: Analisi di 4 differenti punti operativi in DC, e relative tensioni di soglia dei transistor della cella **ND2HS**

Innanzitutto è possibile accorgersi che le V_{TH} dei pMOS sono inferiori a quelle degli nMOS, in quanto tendenzialmente le lacune hanno mobilità inferiore agli elettroni e quindi

i pMOS hanno bisogno di una soglia più bassa per compensare la loro differenza in velocità. Inoltre, per quanto riguarda i 2 pMOS le tensioni di soglia sono identiche tra loro, in quanto la loro configurazione topologica nella cella è esattamente la stessa.

Per gli nMOS invece il discorso è differente e dipende dalla combinazione degli input, la quale definisce la V_G di ogni transistor e quindi il suo punto operativo in DC.

- **A = 0, B = 0**
  L'uscita vale 1, C_L è caricato a 1.2 V ed entrambi gli nMOS sono teoricamente spenti, ma in realtà la corrente di perdita su XMN1 fa sì che *net15*, il nodo tra i due nMOS, sia comunque a circa 45 mV. A questo punto si può notare che XMN1 fa passare corrente più difficilmente di XMN0, in quanto:

  $$V_{GS}(N1) = -44.76\,\text{mV}$$
  $$V_{GS}(N0) = 0\,\text{mV}$$

  Dato che i due transistor sono in serie, affinchè abbiano la stessa corrente è quindi necessario che N1 abbia una soglia più bassa di N0:

  $$V_{TH}(N1) = 0.279\,35\,\text{V} < V_{TH}(N0) = 0.312\,17\,\text{V}$$

- **A = 0, B = 1**
  L'uscita vale 1, C_L è caricato a 1.2 V, XMN0 è spento, XMN1 è acceso. Il nodo *net15* viene caricato ad una tensione molto prossima a V_{DD} (circa 1.1 V). A questo punto si nota che XMN1 lascia passare corrente più facilmente di XMN0, in quanto:

  $$V_{GS}(N1) = 13.9\,\text{mV}$$
  $$V_{GS}(N0) = 0\,\text{mV}$$

  Dato che i due transistor sono in serie, affinchè abbiano la stessa corrente è quindi necessario che N0 abbia una soglia più bassa di N1:

  $$V_{TH}(N0) = 0.277\,19\,\text{V} < V_{TH}(N1) = 0.417\,84\,\text{V}$$

- **A = 1, B = 0**
  L'uscita vale 1, C_L è caricato a 1.2 V, XMN1 è spento, XMN0 è acceso. Il nodo *net15* viene caricato ad una tensione molto bassa a causa della corrente di perdita su N1 (circa 4 µV). A questo punto si può notare che XMN1 lascia passare corrente più difficilmente di XMN0, in quanto:

  $$V_{GS}(N1) = -4.084\,\mu\text{V}$$
  $$V_{GS}(N0) = 1.2\,\text{V}$$

  Dato che i due transistor sono in serie, affinchè abbiano la stessa corrente è quindi necessario che N1 abbia una soglia più bassa di N0:

  $$V_{TH}(N1) = 0.272\,41\,\text{V} < V_{TH}(N0) = 0.313\,71\,\text{V}$$

- **A = 1, B = 1**
  L'uscita vale 0, XMN0 e XMN1 sono accesi, C_L è caricato ad una tensione molto bassa (circa 12 µV) a causa del leakage sui pMOS. Il ramo nMOS è un partitore di tensione ed il nodo *net15* vale praticamente metà della tensione sul carico (circa 6 µV). I due nMOS sono entrambi accesi, con la stessa $V_G = 1.2\,\text{V}$ e con la stessa $V_{DS} = 0.5 \cdot V_{LOAD}$, quindi hanno anche la stessa V_{TH}:

$$V_{TH}(N0) = V_{TH}(N1) = 0.313\,71\,\text{V}$$

5.2 Characterizing a gate for output load

Nel precedente paragrafo si è analizzato il comportamento della cella con un carico costante di 10 fF; adesso si vuole caratterizzare la cella per diversi valori di C_L, che viene definita parametricamente tramite il vettore *load* = [0.005 fF, 0.05 fF, 0.5 fF, 5.0 fF, 50.0 fF].

Diventa interessante in questo caso analizzare i contributi di corrente nelle varie situazioni, tramite l'istanziamento nello script di generatori *dummy* di tensione (sui nodi V_{DD}, GND e Z) e richiamando le misurazioni ancora una volta con il comando *.measure*.
Dopo aver lanciato la simulazione, sono stati ottenuti i risultati riportati in Tabella 5.3.

C_{LOAD}	0.005 fF	0.05 fF	0.5 fF	5.0 fF	50.0 fF
RNAND	6.7226E-11	6.7604E-11	7.1228E-11	1.0281E-10	3.6391E-10
FNAND	6.8511E-11	6.8830E-11	7.2112E-11	9.8194E-11	2.9652E-10
NANDDELAYHL	2.0524E-11	2.0851E-11	2.3980E-11	4.8550E-11	1.8016E-10
NANDDELAYLH	3.7948E-11	3.8286E-11	4.1521E-11	6.6657E-11	2.0900E-10
MAXIGNDF	7.6530E-05	7.7056E-05	8.1964E-05	1.1664E-04	2.4077E-04
MAXIVDDR	-7.0006E-05	-7.0423E-05	-7.4383E-05	-1.0277E-04	-2.0919E-04
MAXIGNDR	4.5879E-05	4.5692E-05	4.4055E-05	3.4693E-05	1.2571E-05
MAXIVDDF	-4.7912E-05	-4.7680E-05	-4.5637E-05	-3.4365E-05	-1.1039E-05
MAXILOADF	-1.2603E-07	-1.2504E-06	-1.1637E-05	-7.5275E-05	-2.3042E-04
MAXILOADR	1.1510E-07	1.1429E-06	1.0641E-05	6.7574E-05	1.9972E-04

Tabella 5.3: Confronto tra i parametri ottenuti con comando *.measure* nelle varie condizioni di carico C_L

Per quello che riguarda i tempi, i risultati sono in linea con ciò che era logico aspettarsi, ovvero all'aumentare della capacità di carico aumentano anche i tempi di salita, discesa e propagazione.

Volendo andare più nel dettaglio, sappiamo che la cella **ND2HS** ha **driving capabilities** ottimizzate per carichi fino a 0.16 fF; questo si nota dal fatto che per C_L uguale a 0.005 fF e 0.05 fF si hanno ritardi sostanzialmente identici, mentre a partire da 0.5 fF essi iniziano a divergere fino ad aumentare anche di un ordine di grandezza nel caso di 50 fF.

Si può avere una conferma visuale dei risultati osservando il grafico in Figura 5.8.

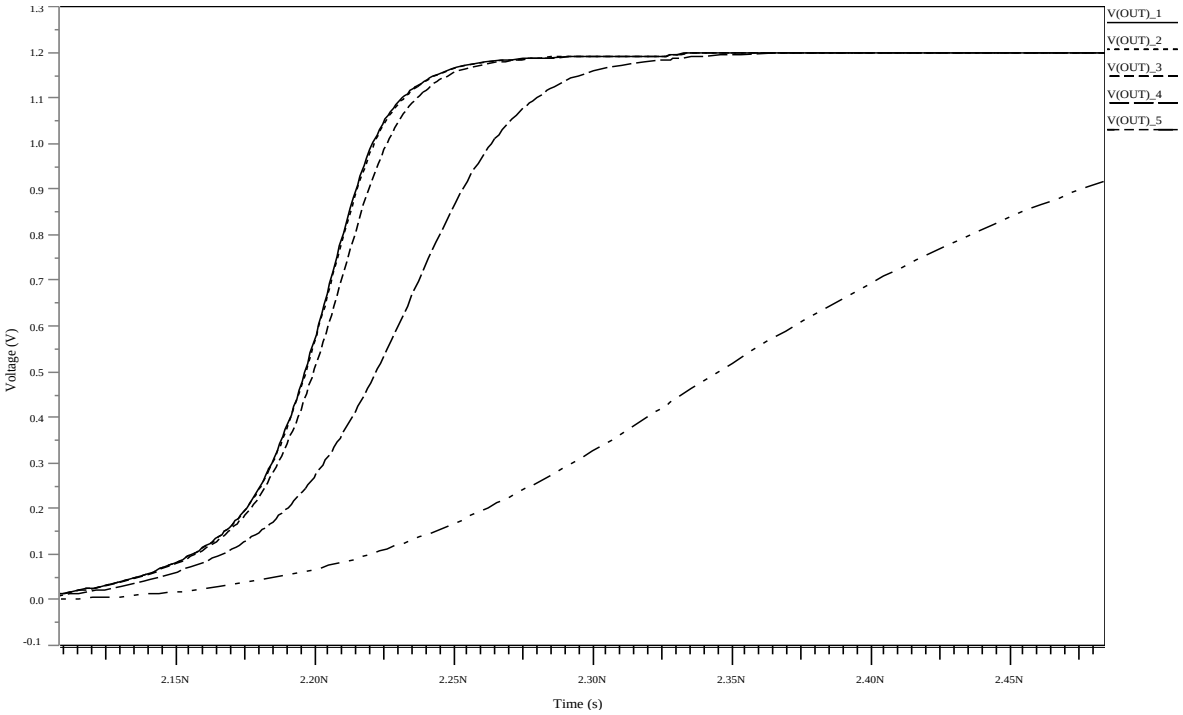

Figura 5.8: Dettaglio sul ritardo di propagazione LH della cella al variare del carico

Per quanto riguarda le correnti, esse vengono riportate in Figura 5.9, Figura 5.10 e Figura 5.11 al variare del carico C_L.

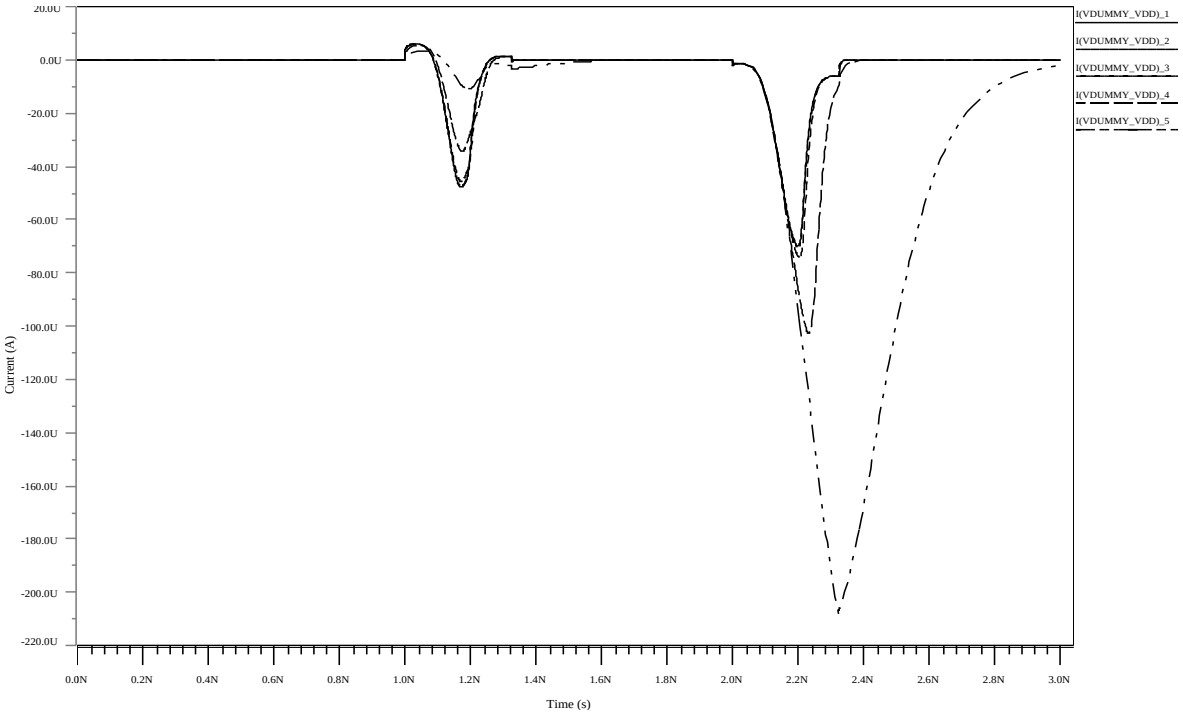

Figura 5.9: Dettaglio sulle correnti di V_{DD} al variare del carico C_L

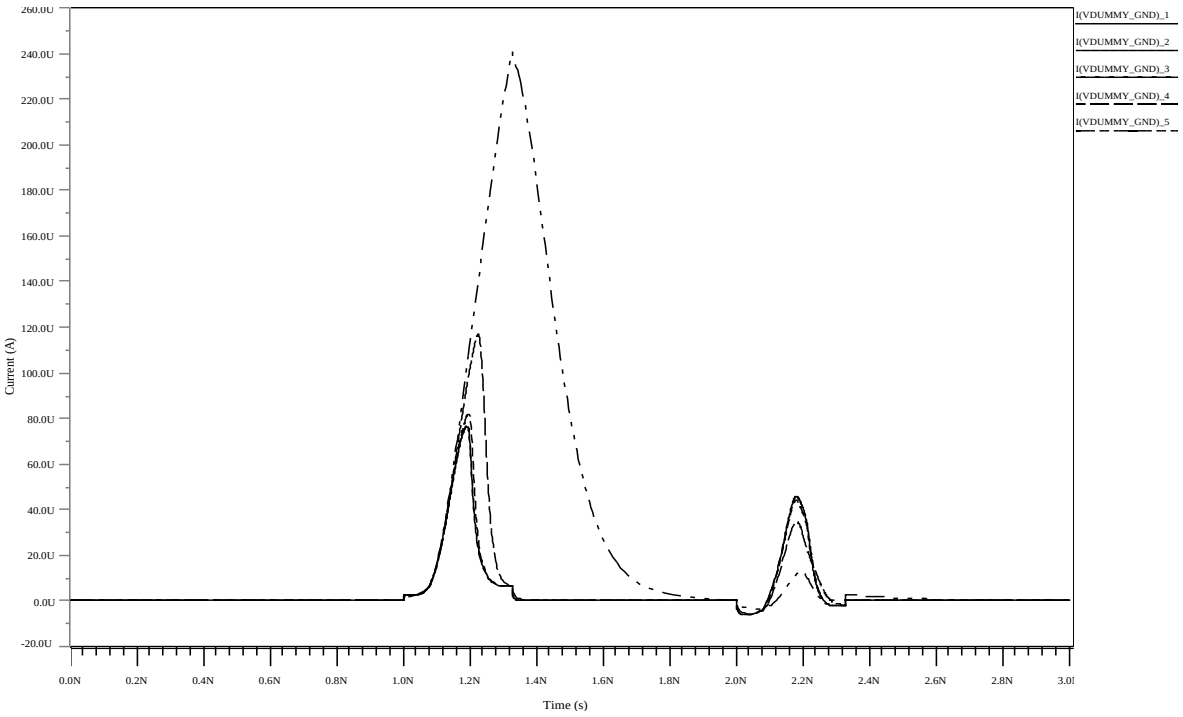

Figura 5.10: Dettaglio sulle correnti di GND al variare del carico C_L

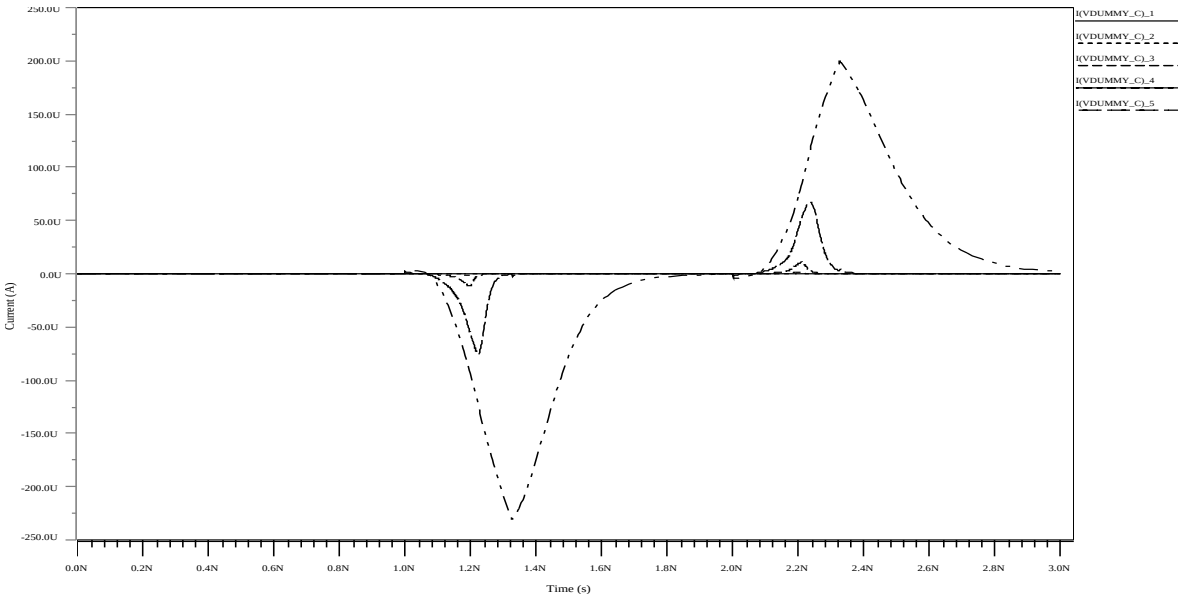

Figura 5.11: Dettaglio sulle correnti di C_L al variare della sua grandezza

Di seguito verranno analizzati separatamente i casi in cui l'output effettua una transizione HL e LH.

- **HIGH → LOW @ 1 ns**
  La rete di nMOS si attiva e deve scaricare C_L. La maggior parte della corrente fluirà verso il nodo GND, con entità crescente all'aumentare del valore del carico. Dal punto di vista di C_L, questa genererà una corrente anch'essa proporzionale alla sua capacità.
- **LOW → HIGH @ 2 ns**
  La rete di pMOS si attiva e deve caricare C_L. La maggior parte della corrente uscirà dal nodo V_{DD}, con entità crescente all'aumentare del valore del carico. Per quanto riguarda C_L, questa riceverà una corrente proporzionale alla sua capacità.

Infine prendiamo nota della potenza statica dissipata dal circuito, che in questo caso risulta essere di **6.7908 nW**. Da notare che si tratta di una potenza stimata in DC, ovvero **indipendente dal valore della capacità di carico** (tant'è che è uguale a quella ottenuta nel paragrafo precedente con carico a 10 fF).

5.2.1 Threshold voltages

Dalla simulazione effettuata in DC per avere informazioni sulle tensioni di soglia sono emersi i valori riportati in Tabella 5.4.

Cell	V_{TH}
XMN0	3.1371E-01
XMN1	2.7241E-01
XMP0	2.4712E-01
XMP1	2.4712E-01

Tabella 5.4: Tensioni di soglia in DC dei MOS

Come si può notare, le tensioni di soglia rimangono inalterate rispetto al caso analizzato nel paragrafo precedente. Il motivo è semplicemente dovuto al fatto che in regime DC le capacità vengono considerate come circuiti aperti e di conseguenza non influenzano in alcun modo l'analisi.

5.3 Comparing different gate sizing

Ogni cella, a seconda della sua struttura e del dimensionamento dei transistor che la compongono ha determinate *driving capabilities*, ossia è caratterizzata da un carico massimo che può essere pilotato con prestazioni soddisfacenti. Come già accennato in sezione 5.2, la cella utilizzata fino a questo momento (*ND2HS*) è ottimizzata per carichi fino a 0.16 fF. Si procederà adesso ad analizzare la cella **ND2HSX8**, ovvero una porta NAND con driving capabilities fino a 1.28 fF.

Lo schema riportato in Figura 5.12 evidenzia come la rete nMOS sia stata quadruplicata mentre la pMOS triplicata; non solo, all'interno della libreria `CMOS013.spi` notiamo anche che le dimensioni dei singoli transistor sono state aumentate al fine di diminuirne la resistenza (ad esempio W_n da $0.64\,\mu m \rightarrow 1.28\,\mu m$ e W_p da $0.77\,\mu m \rightarrow 2.05\,\mu m$). Con un calcolo approssimativo è possibile verificare che le dimensioni della cella si siano incrementate di circa 8 volte:

$$\frac{AREA_{ND2HSX8}}{AREA_{ND2HS}} \cong \frac{1.28 \cdot 8 + 2.05 \cdot 6}{0.64 \cdot 2 + 0.77 \cdot 2} \cong \frac{22.54}{2.82} \cong 8 \tag{5.1}$$

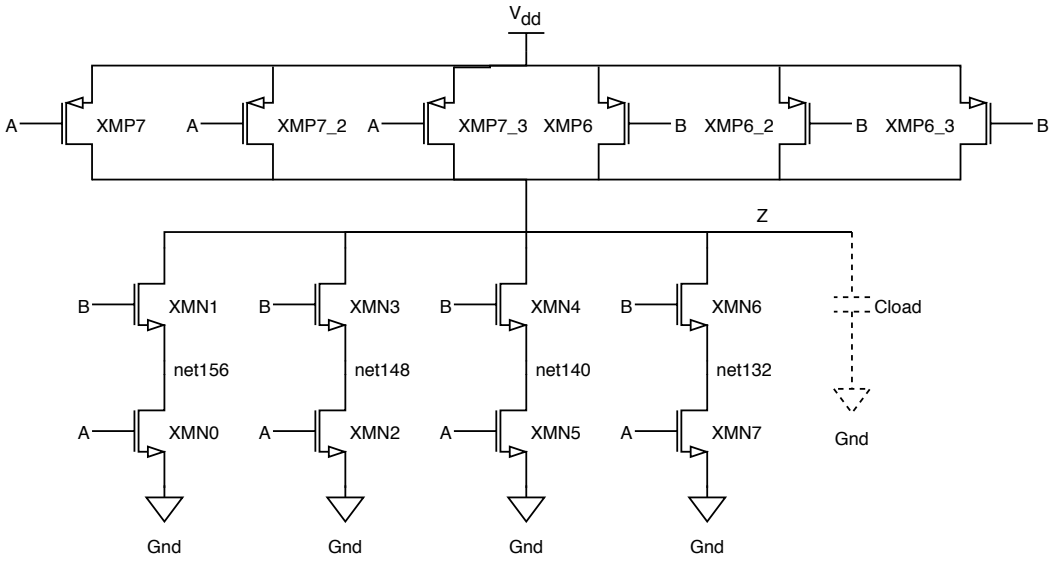

Figura 5.12: **ND2HSX8**

Si andranno ora a testare due differenti capacità di carico: la prima $C_{LOW} = 0.06\,fF$ rientra nelle driving capabilities di entrambe le celle, la seconda da $C_{HIGH} = 60\,fF$ dovrebbe essere pilotata in modo molto più veloce dalla cella *ND2HSX8*. I risultati delle misurazioni (Tabella 5.5) confermano le aspettative, mostrando come **la cella *ND2HSX8* sia circa il 70% più veloce di quella originale in caso di carico $C_{HIGH} = 60\,fF$**.

		RNAND	FNAND	DELAY_HL	DELAY_LH
0.06 fF	ND2HS	6.7692E-11	6.8976E-11	2.0923E-11	3.8361E-11
	ND2HSX8	6.2271E-11	6.3845E-11	1.7465E-11	3.1358E-11
	Avg Δt				**- 10.7%**
60 fF	ND2HS	4.2504E-10	3.4177E-10	2.0371E-10	2.3661E-10
	ND2HSX8	1.1307E-10	1.0687E-10	5.7440E-11	7.2256E-11
	Avg Δt				**- 71%**

Tabella 5.5: Confronto tra i parametri relativi a **ND2HS** e **ND2HSX8** nei due casi testati

La conferma visuale di quanto appena riportato numericamente si può avere osservando, a titolo di esempio, il ritardo di propagazione della porta nella transizione LH (Figura 5.13).

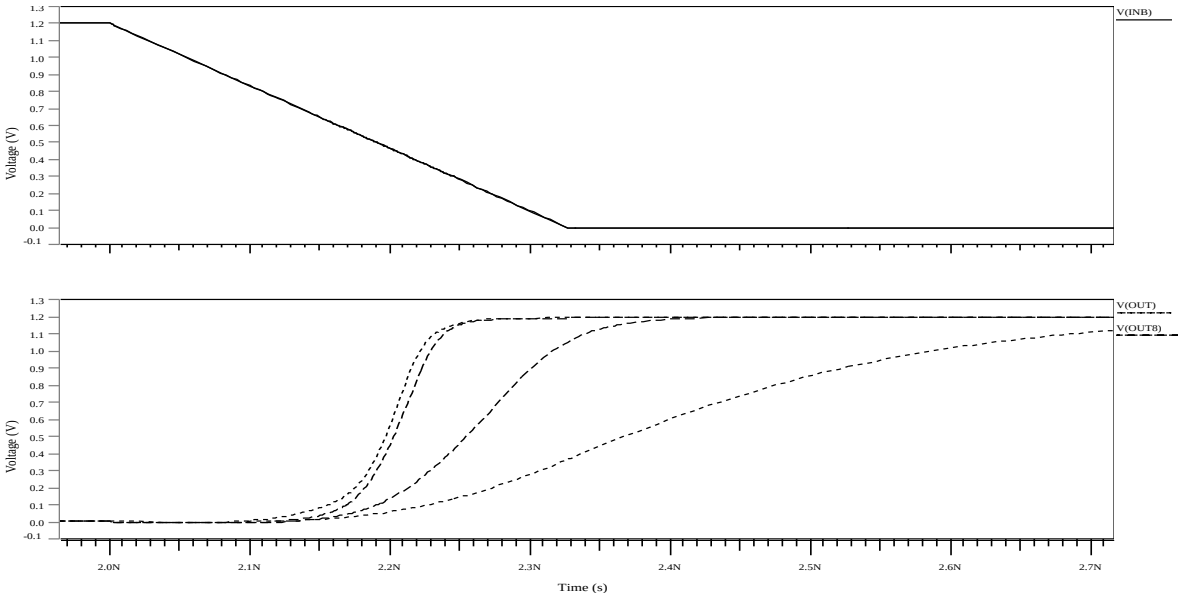

Figura 5.13: Ritardo di propagazione LH per entrambe le celle; **ND2HSX8** è molto più veloce nel caso di $C_L = 60\,\text{fF}$

Si può quindi passare al confronto tra le correnti. Intuitivamente sarebbe logico aspettarsi che le correnti sui nodi V_{DD} e GND siano parecchio maggiori nel caso della cella più grande in quanto ad essi sono collegati molti più transistor; sul carico, invece, ci si aspetterebbe correnti molto simili per il carico C_{LOW}, mentre per C_{HIGH} la cella ND2HSX8 dovrebbe riuscire a gestire più corrente per poterne accelerare la carica/scarica.

I risultati numerici confermano quanto appena previsto e sono riportati in Tabella 5.6 e Figura 5.14.

		ND2HS	ND2HSX8
0.06 fF	IPEAKMAX_C	1.3693E-06	1.5460E-06
	IPEAKMIN_C	-1.4979E-06	-1.7090E-06
	IPEAKMAX_GND	7.7172E-05	6.3745E-04
	IPEAKMIN_GND	-6.0583E-06	-4.2890E-05
	IPEAKMAX_VDD	5.7825E-06	4.5152E-05
	IPEAKMIN_VDD	-7.0514E-05	-5.5288E-04
60 fF	IPEAKMAX_C	2.0627E-04	7.2678E-04
	IPEAKMIN_C	-2.3649E-04	-8.1200E-04
	IPEAKMAX_GND	2.4547E-04	1.0690E-03
	IPEAKMIN_GND	-2.9375E-06	-3.9424E-05
	IPEAKMAX_VDD	3.2629E-06	4.0076E-05
	IPEAKMIN_VDD	-2.1472E-04	-9.2861E-04

Tabella 5.6: Confronto tra i parametri ottenuti nelle due condizioni di carico C_L per i circuiti a diversa driving capability

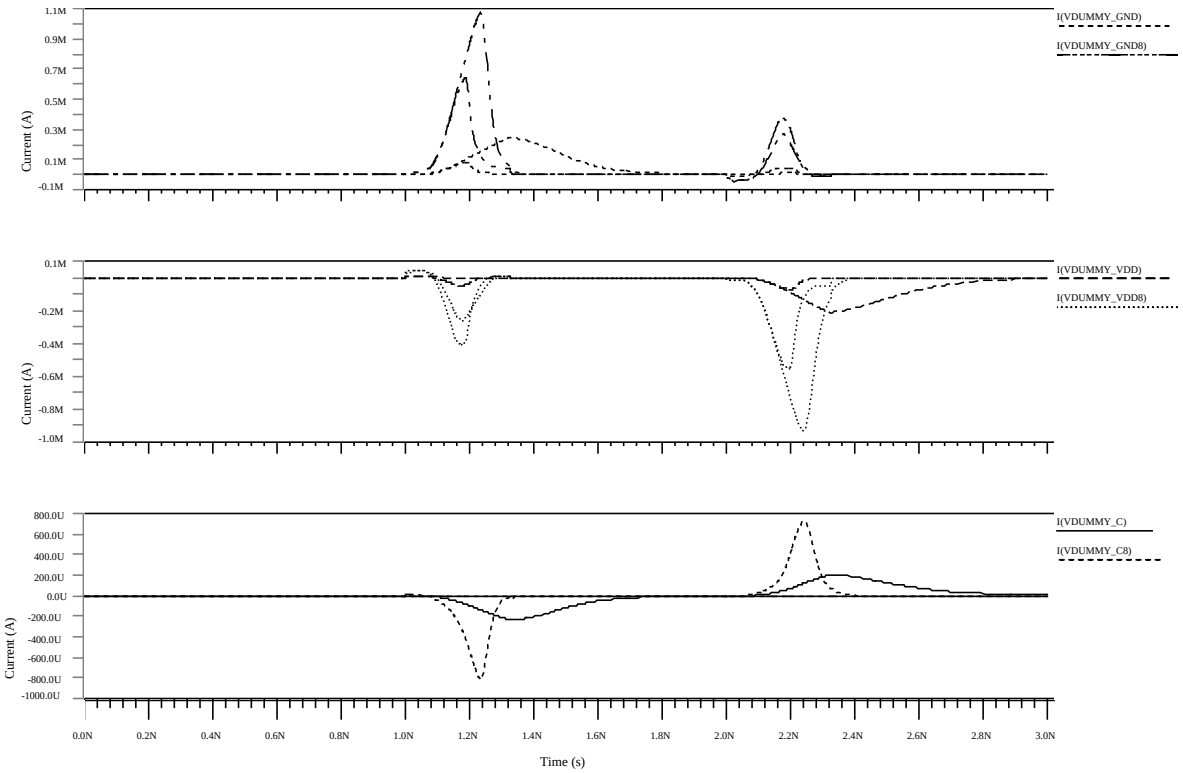

Figura 5.14: Correnti su entrambe le porte

Infine si può confrontare il consumo di potenza delle due celle. Come già spiegato in sezione 5.2, il tool genera automaticamente un'analisi di potenza in DC (trascurando le capacità) che naturalmente mostra come **la cella *ND2HSX8* consumi 49.469 nW, circa 8 volte più della *ND2HS* che invece consuma 6.7908 nW** in quanto banalmente è circa 8 volte più piccola come già spiegato in precedenza. Di conseguenza è opportuno scegliere la cella più grande solo se l'obiettivo è incrementare le prestazioni del dispositivo, a patto di poter accettare un aumento della potenza.

5.3.1 VT

Si possono a questo punto analizzare e confrontare le tensioni di soglia dei dispositivi. Come spiegato in sottosezione 5.1.1, la misurazione delle tensioni di soglia viene fatta in DC, quindi dipende solo da un'analisi statica del circuito. Ci aspettiamo quindi che la tensione di soglia per ND2HS rimanga invariata rispetto ai paragrafi precedenti. Per quanto riguarda ND2HSX8 ci aspettiamo tensioni di soglia differenti, in quanto è cambiata la W dei transistor, ma il leitmotiv dovrebbe essere lo stesso: i pMOS avranno tutti la stessa V_{TH} in quanto topologicamente equivalenti, gli nMOS collegati all'input A (fisso a 1.2 V) avranno una V_{TH} maggiore di quelli collegati a B (PWL). I risultati dell'analisi sono riportati in Tabella 5.7 e confermano quanto previsto.

Cella	V_{TH}	
ND2HS	nMOS - A	3.1371E-01
	nMOS - B	2.7241E-01
	pMOS	-2.4712E-01
ND2HSX8	nMOS - A	3.1893E-01
	nMOS - B	2.7763E-01
	pMOS	-2.4164E-01

Tabella 5.7: Confronto tra le tensioni di soglia nei due circuiti ND2HS e ND2HSX8

5.4 Comparing high speed and low leakage optimization

Oltre alle celle ottimizzate in termini di prestazioni è possibile trovare nelle librerie anche delle celle ottimizzate dal punto di vista dei consumi, e del leakage in particolare. L'obiettivo a questo punto è confrontare prestazioni e consumi delle celle Low-Leakage (LL) con quelle High-Speed (HS) considerate finora. Affinchè il confronto possa essere pertinente verrà utilizzato lo stesso testbench, ovvero la statistica di stimolazione degli input rimarrà invariata.

In Tabella 5.8 viene mostrato un confronto in termini di timing tra le due classi di celle. In media **le celle LL sono state più lente del 10.86% per il carico da 0.06 fF e del 27.4% per quello da 60 fF**.

		RNAND	FNAND	DELAY_HL	DELAY_LH
0.06 fF	ND2HS	6.7692E-11	6.8976E-11	2.0923E-11	3.8361E-11
	ND2LL	7.1556E-11	6.3464E-11	3.1612E-11	5.2657E-11
	Avg Δt				**+ 11.9%**
	ND2HSX8	6.2271E-11	6.3845E-11	1.7465E-11	3.1358E-11
	ND2LLX8	6.3480E-11	5.7371E-11	2.6768E-11	4.4524E-11
	Avg Δt				**+ 9.83%**
60 fF	ND2HS	4.2504E-10	3.4177E-10	2.0371E-10	2.3661E-10
	ND2LL	5.9223E-10	4.0623E-10	2.5767E-10	3.3122E-10
	Avg Δt				**+ 31.5%**
	ND2HSX8	1.1307E-10	1.0687E-10	5.7440E-11	7.2256E-11
	ND2LLX8	1.3767E-10	1.1239E-10	7.8324E-11	1.0277E-10
	Avg Δt				**+ 23.3%**

Tabella 5.8: Confronto tra celle **HS** e **LL** dal punto di vista del timing

La conferma di quanto appena descritto si può avere analizzando il ritardo di propagazione della porta nella transizione LH (Figura 5.15) nei 4 casi, che risulta essere maggiore per porte LL.

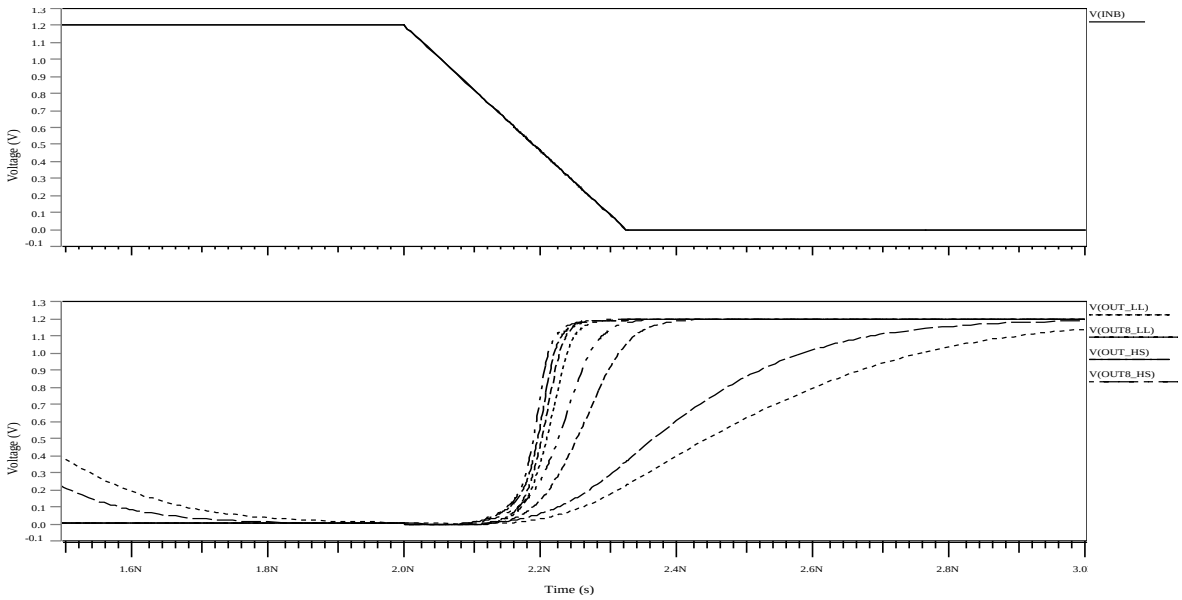

Figura 5.15: Ritardo di propagazione LH per entrambe le celle; **ND2HSX8** è molto più veloce nel caso di $C_L = 60\,\text{fF}$

Per quanto riguarda le correnti, anche le celle LL dovrebbero rispecchiare il trend generale delle celle HS, ovvero correnti rilevanti su V_{DD} e GND nel caso della cella LLX8, correnti sul C_{LOW} paragonabili nei casi LL e LLX8, correnti maggiori sul C_{HIGH} nella versione LLX8. Tuttavia ci si aspetta che complessivamente le correnti delle celle LL siano più basse di quelle HS, data la differenza di velocità rilevata in precedenza. I risultati confermano quanto previsto (Tabella 5.9 e Figura 5.16). Ad esempio si può notare come la cella HSX8 scambi con il C_{HIGH} un totale di 1.5388 mA, circa il **25% in più** della cella LLX8 (1.2319 mA).

		ND2HS	ND2LL	ND2HSX8	ND2LLX8
0.06 fF	IPEAKMAX_C	1.3693E-06	1.0745E-06	1.5460E-06	1.2191E-06
	IPEAKMIN_C	-1.4979E-06	-1.3351E-06	-1.7090E-06	-1.5185E-06
	IPEAKMAX_GND	7.7172E-05	5.1878E-05	6.3745E-04	4.0432E-04
	IPEAKMIN_GND	-6.0583E-06	-5.7358E-06	-4.2890E-05	-3.9616E-05
	IPEAKMAX_VDD	5.7825E-06	5.9895E-06	4.5152E-05	4.5653E-05
	IPEAKMIN_VDD	-7.0514E-05	-4.2315E-05	-5.5288E-04	-3.0301E-04
60 fF	IPEAKMAX_C	2.0627E-04	1.4472E-04	7.2678E-04	5.4325E-04
	IPEAKMIN_C	-2.3649E-04	-1.8652E-04	-8.1200E-04	-6.8863E-04
	IPEAKMAX_GND	2.4547E-04	1.9422E-04	1.0690E-03	8.6710E-04
	IPEAKMIN_GND	-2.9375E-06	-2.0618E-06	-3.9424E-05	-3.6823E-05
	IPEAKMAX_VDD	3.2629E-06	3.2626E-06	4.0076E-05	3.9684E-05
	IPEAKMIN_VDD	-2.1472E-04	-1.5176E-04	-9.2861E-04	-6.8365E-04

Tabella 5.9: Confronto tra i valori di corrente ottenuti nelle due condizioni di carico C_L per i circuiti LL e HS

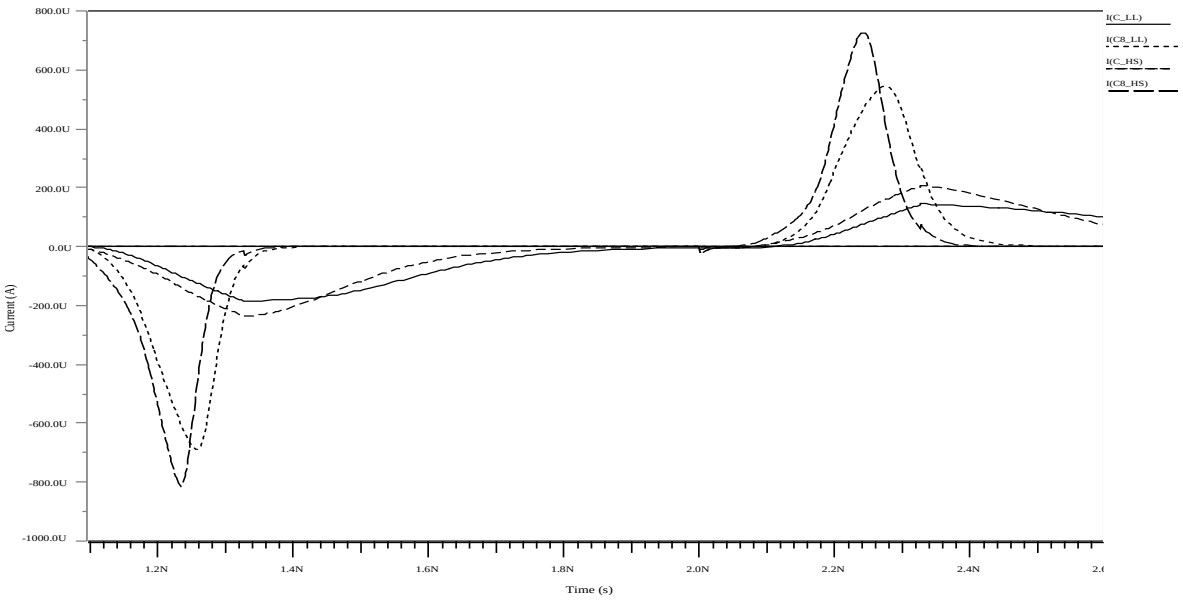

Figura 5.16: Correnti di carico su entrambe le classi HS e LL

Infine si può confrontare il consumo di potenza delle due celle. Ci si aspetta che le versioni X8 delle due celle consumino circa 8 volte in più delle versioni base, e che in generale le celle LL abbiano consentito un risparmio di potenza statica. I risultati ottenuti sono riassunti in Tabella 5.10 e mostrano che se si è disposti ad accettare una degradazione delle prestazioni nelle misure descritte in precedenza, **la scelta di celle LL è la soluzione migliore per risparmiare praticamente il 90% di potenza statica**.

	ND2HS	ND2LL	ND2HSX8	ND2LLX8
Static Power	6.7908E-09	4.4509E-10	4.9469E-08	2.9686E-09
Avg ΔP		**- 93.44%**		**- 94%**

Tabella 5.10: Confronto tra le potenze statiche nei circuiti LL e HS

5.4.1 VT

Uno dei modi per risparmiare facilmente in termini di potenza di leakage, a patto di accettare un degrado delle prestazioni, è quello di aumentare la tensione di soglia dei dispositivi.

Ci aspettiamo quindi di rinvenire un aumento della V_{TH} nelle versioni LL, che però manterranno lo stesso andamento delle celle HS: i pMOS avranno tutti la stessa V_{TH} in quanto topologicamente equivalenti, gli nMOS collegati all'input A (fisso a 1.2 V) avranno una V_{TH} maggiore di quelli collegati a B (PWL).

I risultati dell'analisi sono riportati in Tabella 5.11 e confermano quanto previsto.

		V_{TH}
ND2HS	nMOS - A	3.1371E-01
	nMOS - B	2.7241E-01
	pMOS	-2.4712E-01
ND2HSX8	nMOS - A	3.1893E-01
	nMOS - B	2.7763E-01
	pMOS	-2.4164E-01
ND2LL	nMOS - A	4.1333E-01
	nMOS - B	3.8062E-01
	pMOS	-3.6712E-01
ND2LLX8	nMOS - A	4.1893E-01
	nMOS - B	3.8621E-01
	pMOS	-3.6764E-01

Tabella 5.11: Confronto tra le tensioni di soglia nei 4 circuiti considerati

5.5 Temperature dependency

Come ultimo step si è passati ad analizzare la dipendenza del circuito base ND2LL dalla temperatura, in termini di potenza e tensioni di soglia. I risultati sono mostrati in Tabella 5.12. Come noto, all'aumentare della temperatura si abbassa la V_{TH} nei MOSFET, in quanto l'agitazione termica facilita il passaggio allo stato di conduzione; dato che proprio l'innalzamento della tensione di soglia è una delle tecniche basilari per il contenimento della potenza di leakage, la diminuzione progressiva della V_{TH} provoca un aumento sostanziale della potenza statica dissipata.

Threshold Voltage					**Static Power**	
T [° C]	*XMN0*	*XMN1*	*XMP0*	*XMP1*	*T [° C]*	*P [W]*
-40	4.5753E-01	4.2482E-01	-4.1556E-01	-4.1556E-01	-40	1.1197E-11
0	4.3114E-01	3.9843E-01	-3.8664E-01	-3.8664E-01	0	1.2121E-10
40	4.0476E-01	3.7204E-01	-3.5772E-01	-3.5772E-01	40	7.6913E-10
80	3.7837E-01	3.4566E-01	-3.2880E-01	-3.2880E-01	80	3.2020E-09
120	3.5199E-01	3.1927E-01	-2.9988E-01	-2.9988E-01	120	9.8880E-09
150	3.3220E-01	2.9949E-01	-2.7819E-01	-2.7819E-01	150	1.9907E-08
180	3.1241E-01	2.7970E-01	-2.5650E-01	-2.5650E-01	180	3.6351E-08

Tabella 5.12: Analisi della cella ND2LL in termini di potenza e tensioni di soglia al variare della temperatura nel range $-40\,°\text{C} \div 180\,°\text{C}$

Lo stesso andamento appena descritto viene evidenziato in Figura 5.17.

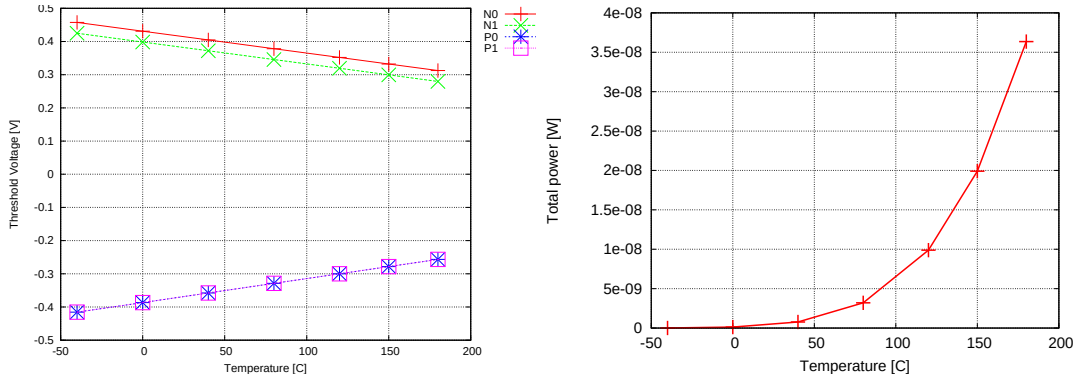

Figura 5.17: Tensioni di soglia e potenza statica al variare della temperatura per cella ND2LL

5.6 Analysis of a memory power components

Ultimo task del laboratorio prevedeva l'analisi dei consumi relativi alle memorie. In particolare veniva assegnata una **SRAM Dual-Ports** di partenza `SRAM_8192-16-16.ps`, della quale si volevano trovare implementazioni alternative per cercare di ridurne i consumi.

Innanzitutto occorre analizzare la struttura delle memorie a nostra disposizione:

- **Nome del file** `SRAM_Ncell-Nbit-Nmux.ps`
  - Ncell: Numero di locazioni della memoria
  - Nbit: Numero di bit in ogni locazione
  - Nmux: Numero di MUX adibiti al BIST (Built-In-Self-Test)

  La prima ipotesi che si è fatta a questo punto è quella di non modificare il valore Nmux rispetto all'architettura di riferimento, per non alterare le self-test capabilities delle strutture alternative. Di conseguenza il ventaglio delle opzioni si è ridotto alle sole memorie `SRAM_xxxx-xx-16.ps`.

- **Area**
  - Core width
  - Core height
  - Footprint width
  - Footprint height

  Le dimensioni del footprint sono comprensive anche dell'overhead dovuto alle interconnessioni, di conseguenza sono le misure più indicative dell'effettivo aumento/diminuzione dell'area totale del circuito.

- **Capacità**
  - Pin Capacitance: capacità associata ai pin della memoria

  Tuttavia si è notato che una volta scelto il numero di MUX di BIST questo valore non cambia, pertanto potremo considerare la capacità come una costante ai fini delle nostre analisi future.

- **Frequenza**: 1 MHz

- **Correnti AC**
  - I_{READ}: corrente consumata dalla singola cella durante operazione di lettura
  - I_{WRITE}: corrente consumata dalla singola cella durante operazione di scrittura
  - I_{PEAK}: massima corrente consumata dalla memoria
  - $I_{DESELECTED}$: corrente consumata dalla memoria quando il suo chip-select non è attivo, ma gli ingressi continuano a switchare
  - $I_{STANDBY}$: corrente di perdita consumata dalla memoria quando viene spenta e gli ingressi non vengono fatti commutare

- **Potenza**
  Per il calcolo della potenza veniva fornita l'Equazione 5.2:

$$I_{AVG} = I_{AC} + \frac{1}{2} \cdot C \cdot V \cdot f \cdot N_{BIT} \cdot N_{PORT} \tag{5.2}$$

  Tuttavia si può notare che nel secondo addendo compaiono tutti termini costanti eccetto N_{BIT}, quindi per semplificare i calcoli si è scelto di adottare una via approssimata, espressa dall'Equazione 5.3, dove il termine 10^{-6} tiene conto del fatto che la frequenza è dell'ordine dei MHz e la capacità dell'ordine dei pF:

$$I_{AVG} \propto I_{AC} + N_{BIT} \cdot 10^{-6} \tag{5.3}$$

A questo punto occorre innanzitutto esaminare tutte le opzioni possibili per gestire un totale di 8192x16 bit (Tabella 5.13):

Opzioni per Memoria Multibanco			
64 x SRAM_512-4-16	32 x SRAM_512-8-16	16 x SRAM_512-16-16	8 x SRAM_512-32-16
32 x SRAM_1024-4-16	16 x SRAM_1024-8-16	8 x SRAM_1024-16-16	4 x SRAM_1024-32-16
16 x SRAM_2048-4-16	8 x SRAM_2048-8-16	4 x SRAM_2048-16-16	2 x SRAM_2048-32-16
8 x SRAM_4096-4-16	4 x SRAM_4096-8-16	2 x SRAM_4096-16-16	1 x SRAM_4096-32-16
4 x SRAM_8192-4-16	2 x SRAM_8192-8-16	1 x SRAM_8192-16-16	

Tabella 5.13: Lista delle opzioni disponibili per creare una **Memoria Multibanco** da 8192x16 bit

Dopodichè è necessario fare alcune scelte a livello di implementazione, che andranno ad influire sul modo in cui la corrente totale verrà calcolata:

- Gli indirizzi vengono incrementati in modo sequenziale
- È attivo un solo blocco alla volta
- Quando un blocco è attivo, gli altri non verranno mai utilizzati finchè il presente blocco non sia stato completamente esaurito

Sotto queste ipotesi possiamo stimare la corrente secondo l'Equazione 5.4:

$$I_{AVG} \propto N_{CELL} \cdot (I_{READ} + I_{WRITE}) + (N_{BLOCK} - 1) \cdot I_{STANDBY} + N_{BIT} = \tag{5.4}$$
$$= N_{CELL} \cdot (I_{READ} + I_{WRITE}) + (N_{BLOCK} - 1) \cdot I_{STANDBY} + N_{CELL} \cdot N_{BIT/CELL}$$

Per quanto riguarda l'area, essa verrà stimata secondo l'Equazione 5.5

$$AREA = N_{BLOCK} \cdot WIDTH_{footprint} \cdot HEIGHT_{footprint} \tag{5.5}$$

Per automatizzare il calcolo, si sono estratti manualmente tutti i parametri di interesse e si sono inseriti in un foglio di lavoro Microsoft Excel, producendo i risultati riportati in Tabella 5.14.

INDEX	#Bit	#Celle	#N	*I(read)*	*I(write)*	*I(st-by)*	I(avg)	I(peak)	Width	Height	AREA	I(avg) * A
1	4	512	64	5.55E-03	5.76E-03	1.09E-01	1.27E+01	29.56	254.20	138.30	2249975.04	2.85E+07
2	4	1024	32	5.82E-03	6.03E-03	1.48E-01	1.67E+01	29.56	259.20	168.90	1400924.16	2.34E+07
3	4	2048	16	6.12E-03	6.33E-03	2.27E-01	2.89E+01	29.56	269.20	228.00	982041.60	2.84E+07
4	4	4096	8	6.44E-03	6.65E-03	3.85E-01	5.63E+01	29.56	274.20	346.10	759204.96	4.28E+07
5	4	8192	4	6.79E-03	7.01E-03	7.00E-01	1.15E+02	29.56	279.20	582.30	650312.64	7.49E+07
6	8	512	32	7.71E-03	8.14E-03	1.68E-01	1.33E+01	36.21	397.60	138.30	1759618.56	2.35E+07
7	8	1024	16	8.07E-03	8.50E-03	2.16E-01	2.02E+01	36.21	402.60	168.90	1087986.24	2.20E+07
8	8	2048	8	8.46E-03	8.90E-03	3.13E-01	3.78E+01	36.21	412.60	228.00	752582.40	2.84E+07
9	8	4096	4	8.92E-03	9.36E-03	5.06E-01	7.64E+01	36.21	417.60	346.10	578125.44	4.42E+07
10	8	8192	2	9.47E-03	9.91E-03	8.93E-01	1.60E+02	36.21	422.60	582.30	492159.96	7.86E+07
11	16	512	16	1.20E-02	1.29E-02	2.86E-01	1.70E+01	49.49	684.30	138.30	1514219.04	2.58E+07
12	16	1024	8	1.26E-02	1.34E-02	3.52E-01	2.91E+01	49.49	689.30	168.90	931382.16	2.71E+07
13	16	2048	4	1.32E-02	1.40E-02	4.84E-01	5.72E+01	49.49	699.30	228.00	637761.60	3.65E+07
14	16	4096	2	1.39E-02	1.48E-02	7.49E-01	1.18E+02	49.49	704.30	346.10	487516.46	5.77E+07
15	16	8192	1	1.48E-02	1.57E-02	1.28E+00	2.50E+02	49.49	709.30	582.30	413025.39	1.03E+08
16	32	512	8	2.07E-02	2.24E-02	5.22E-01	2.57E+01	76.06	1257.80	138.30	1391629.92	3.58E+07
17	32	1024	4	2.16E-02	2.33E-02	6.24E-01	4.79E+01	76.06	1262.80	168.90	853147.68	4.09E+07
18	32	2048	2	2.26E-02	2.43E-02	8.27E-01	9.69E+01	76.06	1272.80	228.00	580396.80	5.63E+07
19	32	4096	1	2.38E-02	2.56E-02	1.23E+00	2.02E+02	76.06	1277.80	346.10	442246.58	8.95E+07

Tabella 5.14: Comparazione tra le diverse opzioni di memoria possibili

A questo punto si sono confrontate le 19 differenti implementazioni in termini di area e corrente. Per quanto riguarda l'area (Figura 5.18) la soluzione #15, ovvero quella originale da `1 x SRAM_8192-16-16` è quella meno costosa; dal punto di vista di correnti (Figura 5.19) la più vantaggiosa è la soluzione #1 da `64 x SRAM_512-4-16`.

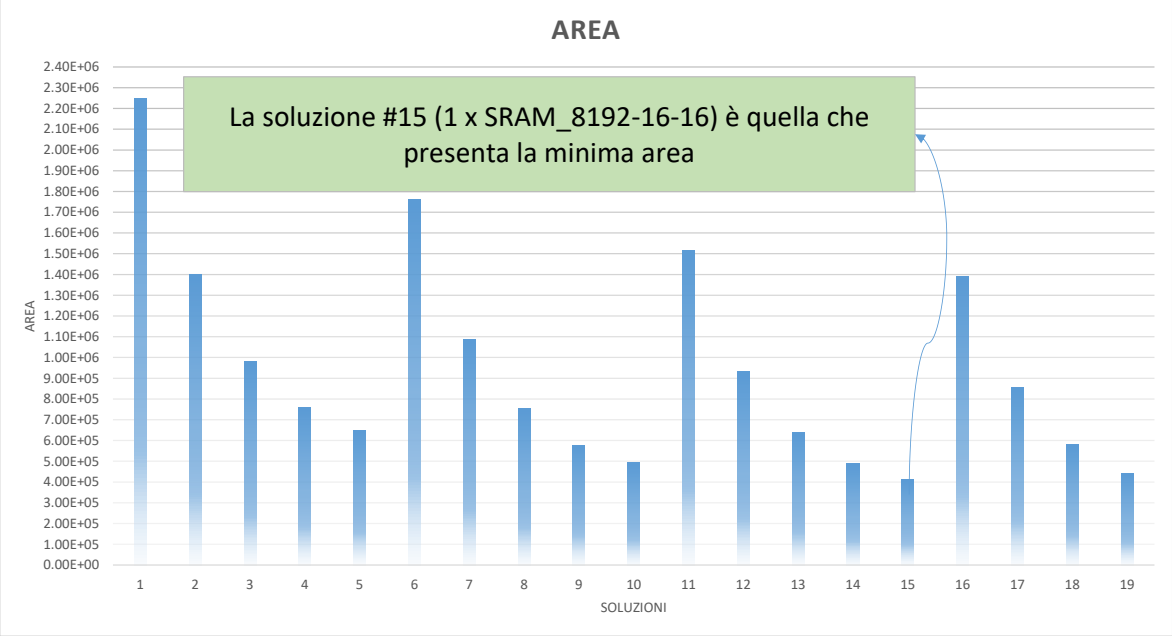

Figura 5.18: Comparazione tra le aree totali relative alle diverse opzioni di memoria possibili

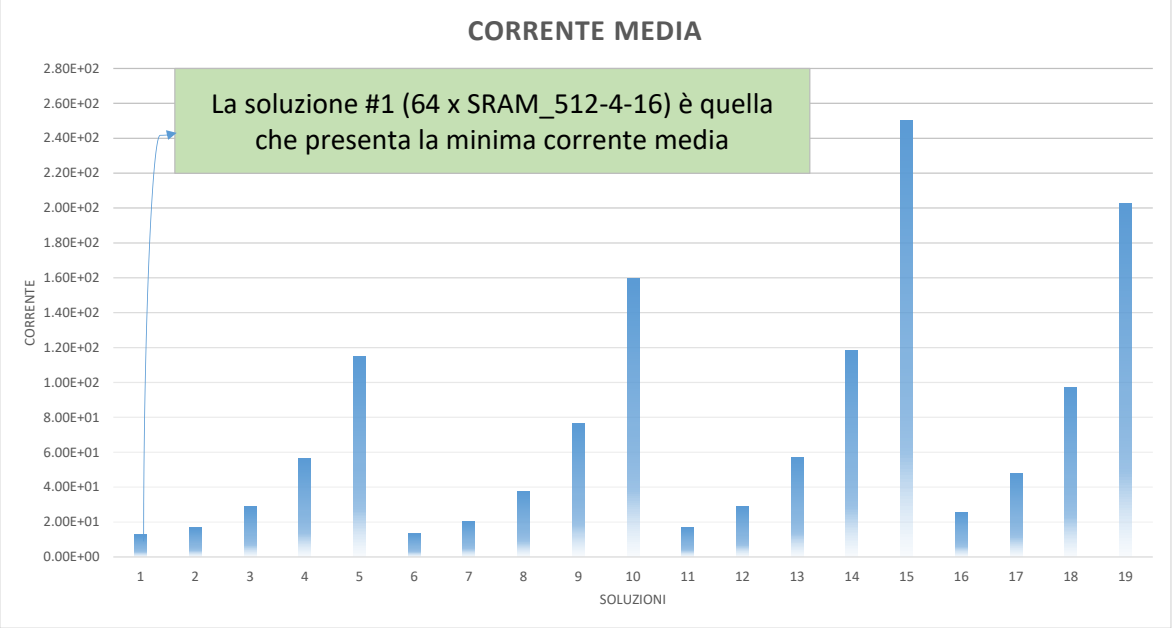

Figura 5.19: Comparazione tra le correnti medie relative alle diverse opzioni di memoria possibili

Per la scelta della migliore alternativa è stato adottato un **approccio multi-obiettivo**, ovvero si è considerata la relazione Area-Corrente riportata in Figura 5.20; è stato calcolato il prodotto *Area* · *Corrente* e, come evidenziato in Tabella 5.14, il miglior compromesso compromesso tra consumo di potenza ed area complessiva è dato dalla soluzione #7, ovvero una memoria multibanco da `16 x SRAM_1024-8-16`.

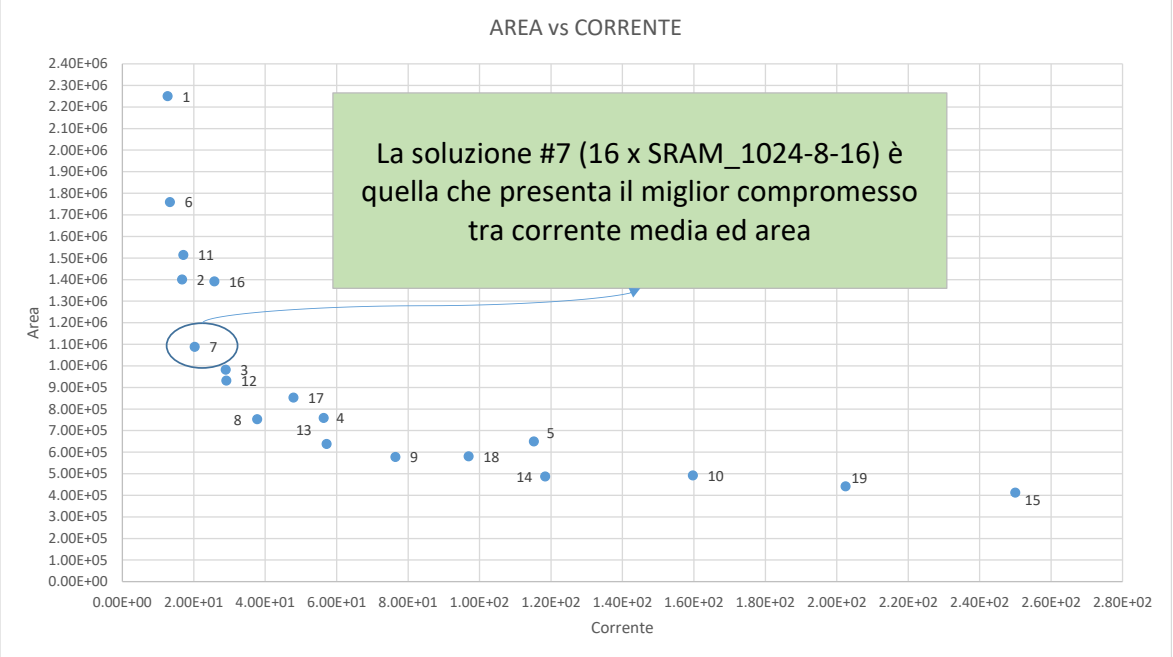

Figura 5.20: Grafico Area-Corrente per tutte le opzioni di memoria disponibili

CAPITOLO 6

Functional Verification

In quest'ultimo laboratorio il focus è stato posto sulla **functional verification**, una delle più importanti, e time-consuming, fasi di progetto.

6.1 VHDL testing

Questa parte di laboratorio è stata utilizzata per prendere confidenza con il processo di verifica tramite l'utilizzo di circuiti relativamente semplici

6.1.1 A given RCA

La prima verifica è stata effettuata su un semplice Ripple Carry Adder, ma a differenza dei precedenti laboratori a riguardo il tesbench ha assunto una struttura differente.

Innanzitutto sono stati utilizzati degli **input pseudo-casuali**, ciò permette di ottenere un risultato molto vicino al comportamento teorico del RCA. In secondo luogo è stata utilizzata un'**architettura di riferimento** per poter comparare l'output del device under test e verificarne la correttezza del risultato. Infine è stato fatto uso dell'istruzione **assert** per riportare un eventuale messaggio di errore nella comparazione dei risultati con l'architettura di riferimente e perciò la presenza di un bug.

Questi tre elementi costituiscono una solida base per un processo di verifica senza l'utilizzo di tool dedicati.

Simulation

Dopo aver opportunamente modificato i path all'interno del file `rca_tb.do`, è stata avviata la prima simulazione, questa, riportata in Figura 6.1, non ha mostrato alcuna anomalia del comportamento atteso.

Successivamente si è andati a modificare il loop per la generazione di numeri pseudo-casuali all'interno del file `rca_tb.vhd`. Aumentando anche di un solo ciclo tale loop, e senza modificare il tempo di simulazione definito all'interno del file `rca_tb.do` l'struzione *assert* è entrata in funzione. La segnalazione del warning è avvenuta tramite il messaggio `-There is a bug-` in corrispondenza dei 30 ns (nella finestra Transcript) e tramite una flag gialla, che

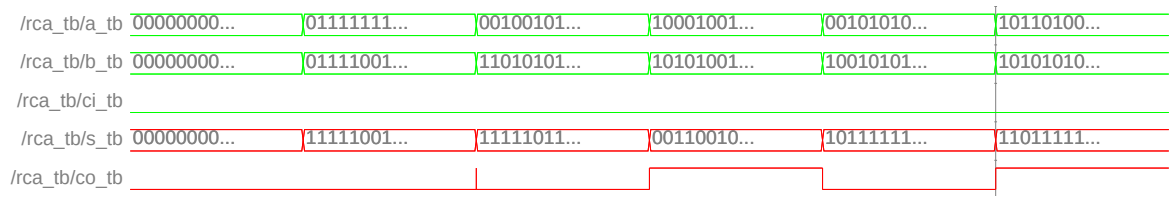

Figura 6.1: Prima simulazione del RCA a 30 ns

purtroppo non siamo riusciti ad esportare in una figura tramite il consueto Print Postscript, sempre in corrispondenza dei 30 ns (ma questa volta nella finestra Wave).

La causa di tale errore risiede all'interno del file `rca.vhd`, in paticolar modo nell'ultimo FA istanziato, al quale era collegato sull'ingresso **Ci** non il penultimo bit del vettore `CTMP`, bensì il bit `CTMP(N_BIT-2)` provocando quindi un errore qualora tali bit fossero differenti e gli ultimi bit dei vettori **A** e **B** ne permettessero la propagazione.

Successivamente netlist e testbench sono stati modificati per testare oltre all'implementazione da 32 bit anche quelle a 64 e 128 bit. Per valutare i tempi associati ad ogni simulazione è stato fatto uso della variabile bash $SECONDS che ha portato ai risultati in Tabella 6.1. Per rendere comparabili i risultati sono stati utilizzati 5 000 000 loop per la generazione degli input e di conseguenza 25 ms come finestra di simulazione, ovviamente senza l'avvio dell'interfaccia grafica.

32 bit	64 bit	128 bit
0h:3m:36s	0h:6m:7s	0h:10m:42s

Tabella 6.1: Tempo impiegato per una simulazione corrispondente a 25 ms per le architetture da 32, 64 e 128 bit del RCA

Come era facilmente prevedibile i tempi di simulazione crescono pressocchè linearmente all'aumentare dei bit dell'architettura.

Per quanto riguarda invece il tempo necessario alla rilevazione del primo errore nei risultati delle tre architetture del RCA, in tutti e tre i casi il comando `assert` viene attivato per la prima volta a 30 ns dall'avvio della simulazione.

6.1.2 A more complex case

In questa seconda parte del laboratorio è stato analizzato un circuito più complesso e non più combinatorio, si tratta di una versione modificata dell'incrementatore e comparatore visto nel Laboratorio 3 la cui struttura è riportata in Figura 6.2.

La prima differenza consiste nell'utilizzo di una descrizione strutturale, in particolare è stato fatto uso dell'adder del Pentium 4. La seconda consiste nell'utilizzo di due multiplexer in ingresso agli adder per permettere l'inizializzazione del counter al valore desiderato.

Il primo passo è stato quello di scrivere un tesbench sulla base di quello utilizzato per il Ripple Carry Adder per testare la versione di riferimento priva di bug collocata all'interno

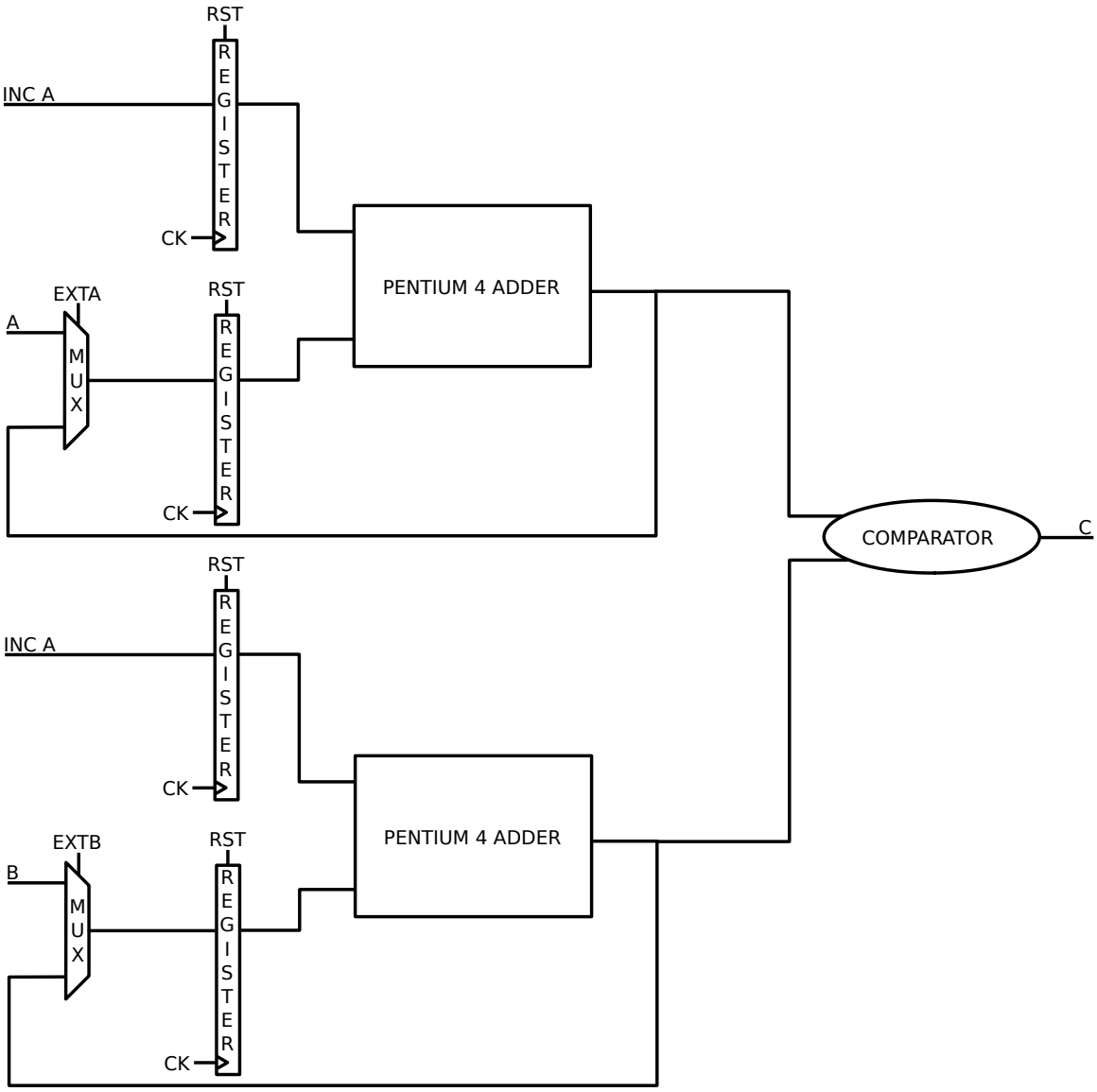

Figura 6.2: Architettura modificata dell'incrementatore e comparatore visto nel Laboratorio 3

della cartella **vREF**. Tale testbench è riportato nella sua interezza in Appendice E.1, oltre alla generazione dei vettori casuali `A` e `B` anche i segnali binari collegati ai due incrementer sono stati gestiti con una generazione casuale. I segnali per pilotare i multiplexer invece sono stati definiti manualmente.

Come per i precedenti laboratori sono stati creati due script, il primo chiamato `script.sh` che si occupa semplicemente di ripulire le librerie create nelle precedenti simulazioni, richiamare il comando `setmentor` e infine chiamare il secondo script, `inccomp_tb.do` all'avvio di *Modelsim*. Tali script sono riportati rispettivamente in Appendice E.2 e Appendice E.3.

Dopo aver proceduto al test della versione di riferimento ci si è concentrati sulle possibili soluzioni per una corretta verification, le soluzioni ponderate sono state quattro:

1. creare un tesbench che richiamasse due architetture, la prima con architettura potenzialmente bugged e la seconda con l'architettura di riferimento, **cambiando manualmente** i nomi dei moduli di quest'ultima per non creare conflitti in compilazione;

2. creare un tesbench che richiamasse due architetture, la prima con architettura potenzialmente bugged e la seconda con l'architettura di riferimento, **cambiando tramite script bash** i nomi dei moduli di quest'ultima per non creare conflitti in compilazione;

3. creare un tesbench che richiamasse due architetture, la prima con architettura potenzialmente bugged e la seconda con l'architettura di riferimento, **senza cambiare** i nomi dei moduli di quest'ultima ma creando delle librerie separate per i suoi componenti;

4. creare un tesbench che richiamasse una sola architettura alla volta e producesse un **file di testo** con i risultati della simulazione da confrontare tramite script con i risultati dell'architettura di riferimento.

Sebbene la terza soluzione sarebbe quella ideale per tale laboratorio, e anche a livello industriale, *Modelsim* non sembra mettere a disposizione la possibilità di selezionare la libreria in fase di compilazione ma solo durante la simulazione grazie all'opzione `-L`, questa soluzione quindi non ha permesso di produrre un architettura corretta per procedere alla fase di verifica.

La quarta opzione è stata scartata in quanto non coerente con le richieste del testo di laboratorio.

Nonostante l'operazione di modifica manuale sarebbe risultata estremamente semplificata tramite l'uso di programmi come *Sublime*, si è infine optato per la seconda soluzione rispetto alla prima poichè era necessario modificare le versioni potenzialmente non funzionanti o sarebbe potuto insorgere un errore in fase di compilazione. Presupponendo infatti che il bug potesse essere presente in qualunque modulo, ad esempio in un solo sottomodulo di un singolo P4 adder che compone l'architettura, in fase di compilazione, a causa della medesima entity, tale modulo avrebbe potuto essere sostituito dalla corrispettiva versione bug-free dell'adder gemello, o viceversa il modulo contenente il bug avrebbe potuto presentarsi in entrambi gli adder. Questo problema non sussiste nella versione di riferimento in quanto i moduli sono per definizione corretti e quindi identici tra loro all'interno dei due adder.

Per fare ciò è stato utilizzato lo script bash `rename.sh` riportato in Appendice E.4. Tale script accede alla cartella `src` e ciclicamente alle cartelle sottostanti modificando ogni singolo file grazie a due funzioni principali: `entity_substitution` e `component_substitution`.

La prima, come suggerisce il nome, tramite il comando `grep` in pipe con un `cut` estrae il nome della entity alla quale viene aggiunta tramite il comando `sed` una label, `_a` per il P4_1 o `_b` per il P4_2. La seconda in modo similare ricerca all'interno del file tutti i component e utilizza `sed` per l'aggiunta della label corrispondente. Per semplicità la cartella `utils` è stata duplicata e rinominata per poter agire sulle due cartelle tramite la stessa metodologia descritta in precedenza.

Ovviamente tale modifica ha portato a rivedere il file `inccomp_tb.do` che ora necessita i comandi per la compilazione dei file di riferimento e di quelli contenenti potenzialmente dei bug. Questo nuovo file è stato chiamato `inccomp_cross_tb.do` ed è riportato per compeltezza in Appendice E.5.

Simulation

Sono state sottoposte al banco di prova le quattro versioni dell'architettura, ognuna assieme alla versione di riferimento. Di seguito saranno analizzate e discusse una alla volta tali versioni.

v1 · È stato riportato in Figura 6.3 l'estratto della simulazione tra 0 e 12 ns ottenute tramite il nuovo testbench riportato in Appendice E.6.

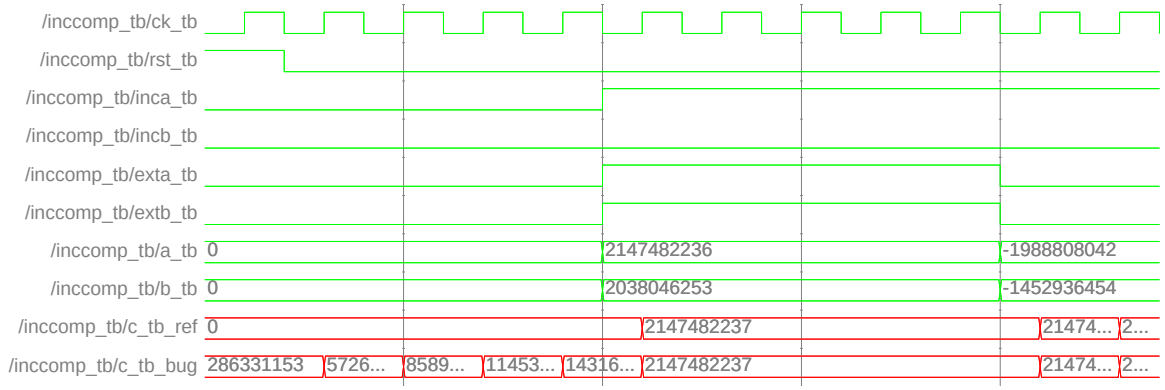

Figura 6.3: Errore nella simulazione della **v1** rispetto alla **vREF**

Com'è possibile notare l'uscita della **v1**, chiamata `c_tb_bug`, differisce fin da subito con l'uscita di riferimento, così sono state aggiunte al banco di prova i segnali di somma dei quattro adder e si è potuto attribuire il bug al secondo sommatore della **v1** come si evince dalla Figura 6.4.

Prima di analizzare i segnali e i moduli interni del `p4a_2` sono stati comparati gli input, ma questi sono risultati essere identici a quelli della **vREF**. All'interno del `p4a_2` il segnale di check `sERROR` ha mostrato anch'esso un errore sul segnale `sSUM` collegato all'uscita, dunque, essendo questi generato dal `sum_generator` e a sua volta dal `carry_select_block` si è proceduto analizzando tale modulo.

Il bug è stato identificato proprio all'interno di quest'ultimo, precisamente nel port map del multiplexer, infatti `sSUM_0` e `sSUM_1` risultavano invertiti, per questo motivo il segnale `SUM`

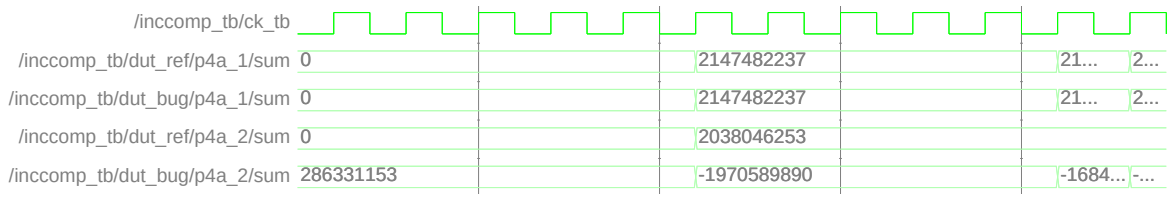

Figura 6.4: Errore nella simulazione del segnale di somma del **p4a_2** della **v1** rispetto a quello della **vREF**

risultava essere sempre quello prodotto utilizzando il `C_IN` errato. Rilanciando la simulazione dopo la correzione del port map i risultati sono stati quelli corretti, coincidenti con la **vREF**.

v2 · Eseguendo il medesimo testbench utilizzato per la **v1** non è stato rilevato alcun errore. Essendo tale testbench estremamente semplice si è provveduto a modificarlo e migliorarlo per ottenere una copertura più ampia delle possibili casistiche.

Per questo motivo si è deciso di testare il funzionamento dei P4 adder singolarmente sino a portarli in overflow. L'intenzione era quella di impostare il valore iniziale di `A` e di `B` a zero e, fissando il segnale `INCA` al valore logico '1', simulare l'architettura fino al raggiungimento della condizione di overflow, replicando successivamente il medesimo test con `INCB` ad '1'. Per fare ciò sono richiesti almeno 2^{32} colpi di clock per ogni adder, questo avrebbe portato a delle simulazioni di diverse ore e non è stata pertanto una via praticabile.

Si è optato quindi per stimolare le criticità del circuito in maniera differente: si sono impostati prima `A` e poi `B` ad un valore molto vicino a quello di overflow, ma entrambi i contatori non hanno prodotto risultati differenti rispetto alla **vREF**. Allo stesso modo si sono testati i carry interni, impostando quindi dei valori, prima di `A` e poi di `B`, prossimi ad una potenza del 2. Questo ha portato a rilevare l'errore riportato in Figura 6.5.

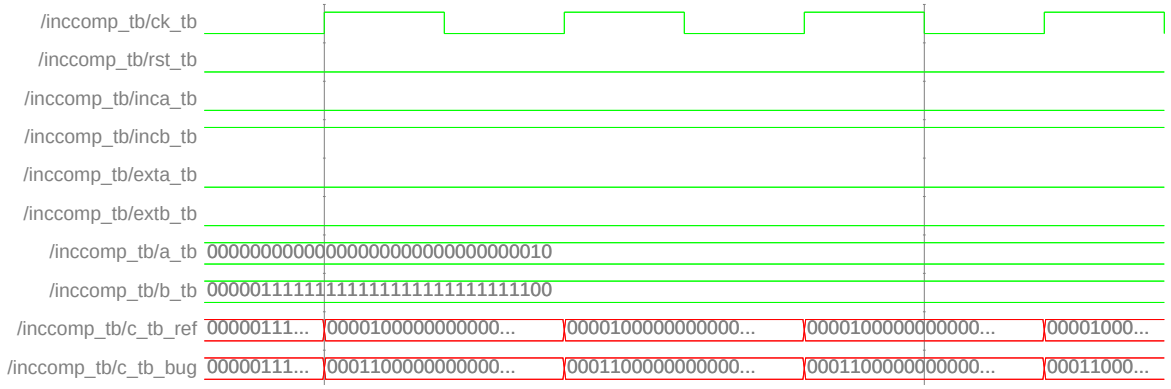

Figura 6.5: Errore nella simulazione della **v2** rispetto alla **vREF**

Tale errore è stato ottenuto impostando `B` ad un valore prossimo a 2^{27} e successivamente incrementandolo. Come per la versione precedente si è analizzato l'interno del `p4_adder_2`, qui `sERROR` mostrava un errore sul segnale `sSUM` nell'istante in cui veniva raggiunto il valore 2^{27}. Procedendo dunque con l'analisi del `sum_generator` si è notato che l'ultimo `carry_select_block` prendeva in ingresso il bit errato del vettore `C_IN`, producendo quindi una somma errata.

Correggendo tale errore e risimulando l'architettura la somma in uscita non presentava più alcun errore.

Per effettuare queste verifica il comando `assert` è stato rimosso dal loop per la generazione dei numeri casuali, che in questa versione non sono stati utilizzati, e inserito dopo il port map delle due architetture, in modo che ad ogni variazione dell'uscita potesse verificarne la correttezza. Il testbench completo è visionabile in Appendice E.7.

v3 · Per testare la **v3** si sono utilizzati in successione tutti i metodi visti in precedenza ma nessuno di questi ha prodotto alcun errore rispetto alle uscite della **vREF**.

A questo punto si è considerata il limitato utilizzo del P4 adder: esso infatti all'interno dell'architettura viene utilizzato esclusivamente per sommare ad un numero randomico, rappresentabile su 32 bit, il valore uno. Si è ipotizzato che questo utilizzo limitato dell'adder non possa permettere il rilevamento di alcuni errori nell'architettura, i quali potrebbero presentarsi solo al di fuori di queste condizioni. Per testare questa ipotesi si è creato un nuovo testbench, riportato in Appendice E.8, per analizzare singolarmente i P4 adder anche se tale operazione (ammesso che l'ipotesi precedente sia vera e che quindi gli adder inseriti all'interno del comparatore in Figura 6.2 non possano produrre errori in tali condizioni) non era richiesta.

Nonostante questo test ulteriore nessun errore è stato rilevato, così si è passati all'analisi della versione successiva.

v4 · L'analisi di quest'ultima versione ha visto anch'essa l'utilizzo, in successione, di tutti i metodi visti finora che per praticità sono riportati di seguito:

- test dell'architettura **inccomp** tramite la generazione in input pseudo-casuali per gli ingressi `A` e `B` con `INCA`, `INCB`, `EXTA` ed `EXTB` definiti manualmente;
- test dell'architettura **inccomp** tramite la generazione in input pseudo-casuali per gli ingressi `A`, `B`, `INCA` e `INCB` con `EXTA` ed `EXTB` definiti manualmente;
- test dell'architettura **inccomp** tramite la stimolazione dell'overflow nei singoli adder;
- test dell'architettura **inccomp** tramite la stimolazione di tutti i carry interni nei singoli adder;
- test delle architetture **p4a_1** e **p4a_2** tramite la generazione in input pseudo-casuali per gli ingressi `A`, `B` e `C_IN_0`;
- test delle architetture **p4a_1** e **p4a_2** tramite la generazione in input pseudo-casuali per gli ingressi `A` e `B` con `C_IN_0` fissato ad uno.

Anche in questo caso, come per la **v3**, non è stato rilevato alcun tipo di errore.

6.1.3 Finite State Machine

Con una metodologia simile a quella del precedente esercizio è stata fornita una versione di riferimento da comparare ad altre quattro versioni potenzialmente contenenti errori. Essendo la macchina a stati impremetata in un singolo file `.vhd` non sono state necessarie grandi modifiche ai file, difatti è stata mofidicata esclusivamente la entity della versione di riferimento in modo tale da poterla richiamare nel testbench e differenziarla dalla versione sotto analisi.

La struttura del contatore sotto forma di macchina a stati finiri è riportata in Figura 6.6.

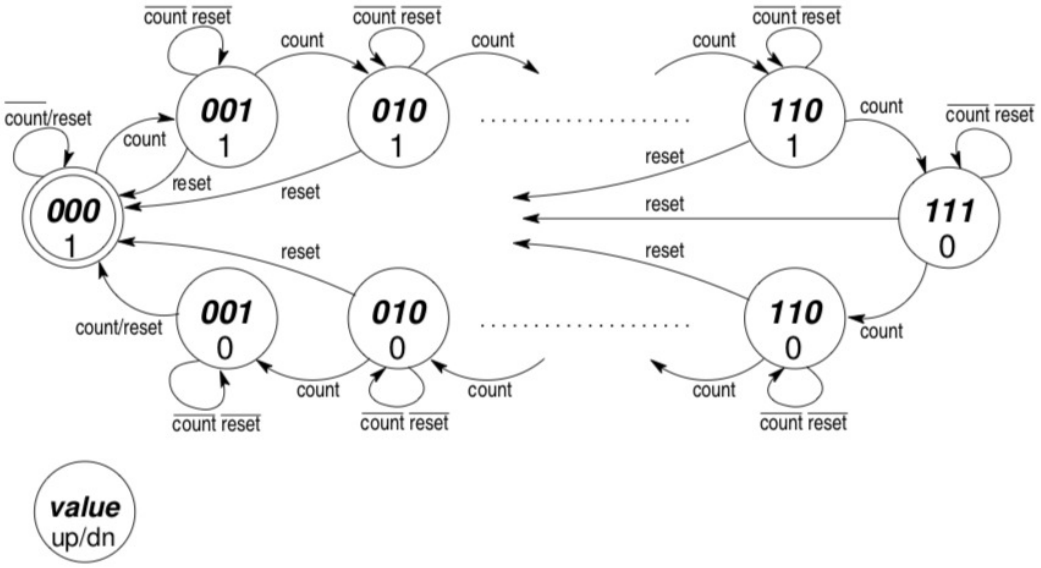

Figura 6.6: Contatore implementato tramite Finite State Machine

Simulation

Esattamente come per l'architettura precedente, sono state sottoposte al banco di prova le quattro versioni messe a disposizione, ognuna assieme alla versione di riferimento. Di seguito saranno analizzate e discusse una alla volta tali versioni.

v1 · Alla prima simulazione, tramite il testbench riportato in Appendice E.9, il quale utilizza la generazione pseudo-casuale per il segnale binario `COUNT` si è ottenuto dopo pochi cicli l'errore riportato in Figura 6.7.

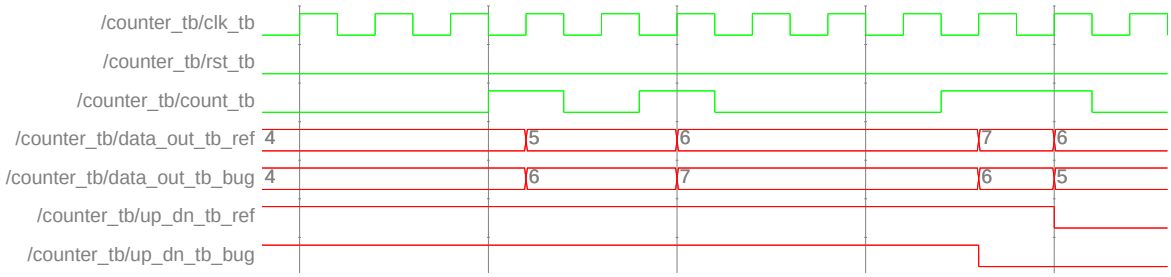

Figura 6.7: Errore nella simulazione della **v1**

Qui l'individazione dell'errore è stata molto più semplice, già dai risultati ottenuti l'ipotesi è stata che l'errore fosse nel passaggio dal quarto al quinto stato della macchina, `idle` escluso. Tale ipotesi si è rivelata essere corretta, all'apertura del file `counter.vhd` infatti, nel `case` per la definizione dello stato futuro, lo stato `u4` riportava come stato successivo in caso di `COUNT = '1'` il passaggio allo stato `u6`, saltando quindi lo stato `u5`. Corretto tale errore la macchina a stati si è comportata come previsto.

v2 · Lo stesso testbench utilizzato per la `v1` è stato eseguito anche per questa versione del contatore ma senza ottenere risultati differenti rispetto alla `vREF`. Si è agito dunque sull'unico input rimanente, se si esclude il clock, ovvero `RST`: si è fatto in modo infatti che ogni stato fosse coinvolto dal segnale di reset per testare la correttezza del comportamento dell'architettura. Si è così ottenuto l'errore riportato in Figura 6.8.

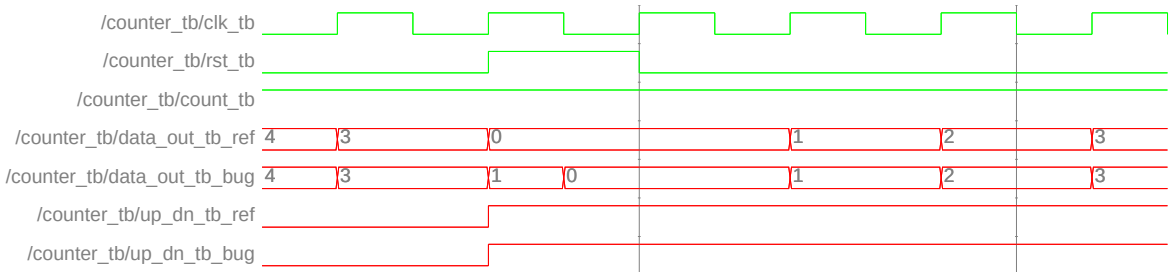

Figura 6.8: Errore nella simulazione della **v2**

All'interno del testbench è stato fissato il segnale `COUNT` al valore logico '1' e tramite il costrutto `wait until` si sono forzate le condizioni di reset in tutti i possibili stati. Il testbench completo è disponibile in Appendice E.10.

v3 · Procedendo con la **v3** si sono utilizzati in successione le tecniche precedenti, ma come era prevedibile non hanno portato ad alcuna differenza tra le uscite di questa versione e la **vREF**.

Si è successivamente testata l'architettura per quattro volte utilizzando un tempo di simulazione abbastanza lungo, circa 100 ms, fissando i segnali `COUNT` e `RST` in tutte le possibili combinazioni (`00`, `01`, `10` e `11`) ma senza che questo portasse ad alcun risultato.

La stessa idea è stata spostata dall'architettura ai singoli stati testandone il funzionamento per lunghi periodi di tempo, ovviamente in questo caso il reset deve rimanere disattivato e il segnale `COUNT` deve essere temporaneamente bloccato per un periodo di tempo definito ad ogni ingresso in un nuovo stato. Il periodo di tempo preso in analisi è stato di 1 µs, questo ha portato all'errore riportato in Figura 6.9.

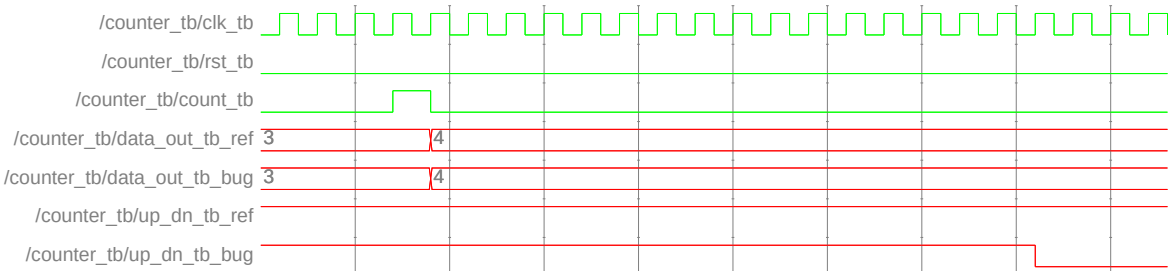

Figura 6.9: Errore nella simulazione della **v3**

Tale errore ha portato ad invertire il segno del segnale d'uscita `up_dn` dopo 16 ns dall'incremento da 3 a 4 del segnale `data_out` e successivamente a invertire l'andamento incrementale dell'uscita, tale comportamento indica che vi è stato un passaggio di stato seppur l'uscita del contatore non sia stata modificata.

Al momento della correzione del codice è stato semplice capire il perchè: infatti si passava dallo stato `u4` allo stato `d4` i quali posseggono lo stesso valore d'uscita `data_out` ma opposto `up_dn`.

Una volta commentate le righe di codice che gestivano il segnale `steady_counter` durante il suo incremento e nello stato `u4` il comportamento del contatore è stato ripristinato al suo funzionamento originale.

La struttura del testbench utilizzato riprende quella della **v2**, per completezza è dunque riportato in Appendice E.11 il testbench utilizzato per la presente versione.

v4 · Per scrupolo tutti i testbench visti finora sono stati applicati alla **v4** sebbene per esclusione tale versione risultava essere bug-free, questo poichè in ambito lavorativo non si può mai essere certi che il proprio progetto sia esente da errori senza prima averlo accuratamente testato.

Come era prevedibile però nessuno dei test precedenti ha portato alla luce dei comportamenti anomali all'inteno della **v4**.

6.2 Scripting and Python

Uno dei linguaggi di scripting più utilizzato a livello industriale è sicuramente Python, questo ci ha spinto ad analizzare, seppur parzialmente, questa parte opzionale dell'ultimo laboratorio.

6.2.1 How to automatically create a VHDL testbench

In questa sezione si intende sfruttare tale linguaggio di scripting per automatizzare la creazione di testbench per moduli generici, riducendo considerevolmente il tempo usualmente necessario per tale scopo.

Si è dunque passati alla modifica del file `TB_generator.py` messo a disposizione per testare un semplice Ripple Carry Adder, qui sono stati inseriti diversi parametri, tra cui il nome della entity e del file, i nomi, la direzione, i tipi e la dimensione dei segnali ed anche il numero di step di simulazioni e il relativo tempo che intercorre tra di essi. Tale file è stato inserito per completezza in Appendice E.12.

In questo modo è stato estremamente semplice e rapido generare 10 000 step di simulazione per il RCA tramite l'utilizzo di vettori casuali sugli ingressi `A` e `B`. Un estratto del testbench ottenuto per il test del RCA è riportato in Appendice E.13.

I successivi punti non sono stati svolti entro la data di consegna di questo report, ma saranno sicuramente visionati successivamente dato il loro valore formativo spendibile in un contesto lavorativo.

Appendice A

Lab 1

A.1 sim_script.tcl

```
    #Compile
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/lfsr.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/tb_prob.vhd
    #Start simulation, without optimization, resolution 100ps
vsim -t 100ps -novopt work.tbprob(test)
    #Tc(CK) = 20
power add *
run 10 ns
power report -file report_power_20.txt
    #Tc(CK) = 200
restart -f
power add *
run 100 ns
power report -file report_power_200.txt
    #Tc(CK) = 2000
restart -f
power add *
run 1000 ns
power report -file report_power_2000.txt
    #Tc(CK) = 20000
restart -f
power add *
run 10000 ns
power report -file report_power_20000.txt
    #Tc(CK) = 200000
restart -f
power add *
run 100000 ns
power report -file report_power_200000.txt
    #End simulation
quit
```

A.2 syn_script_rca.tcl

```
analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab1/fa_syn.vhd /home/lp19.21/Documents/
    lab1/rca_syn.vhd}
elaborate RCA -library WORK
compile -exact_map

create_clock -name clk -period 1
report_timing > report_timing.txt
report_power > report_power.txt

report_power -hier > report_power_hier.txt
report_power -net -verbose > report_power_net.txt

current_instance /RCA/FAI_1
report_power > report_power_FAI_1.txt
report_power -cell > report_power_FAI_1_cell.txt
report_power -net -verbose > report_power_FAI_1_net.txt

current_instance /RCA/FAI_8
report_power > report_power_FAI_8.txt
report_power -cell > report_power_FAI_8_cell.txt
report_power -net -verbose > report_power_FAI_8_net.txt
```

```
## Back to RCA
current_instance
report_power -net -verbose > report_power_RCA_net.txt

quit
```

A.3 tb_mux21_glitch.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity TBMUX21BIS is
end TBMUX21BIS;

architecture TEST of TBMUX21BIS is

    signal A:      std_logic :='0';
    signal B:      std_logic :='0';
    signal S:      std_logic :='0';
    signal Y:      std_logic;

    signal N: std_logic;
    signal Z: std_logic;
    signal X: std_logic;

begin

    --A <= '1';
    --B <= '1';
    --S <= '1', '0' after 1 ns; -- output dovrebbe rimanere a 1 invece ...

    A <= '0', '1' after 1 ns;
    B <= '0';
    S <= '0', '1' after 1 ns; -- output dovrebbe rimanere a 1 invece ...

    PSN: process(S)
    begin
      N <= not(S) after 0.1 ns;
    end process;

    PBS: process(B,S)
    begin
      Z <= B and S;
    end process;

    PAS: process(A,N)
    begin
      X <= A and N;
    end process;

    POUT: process(X,Z)
    begin
      Y <= X or Z;
    end process;

end TEST;

configuration MUX21BISTEST of TBMUX21BIS is
   for TEST
   end for;
end MUX21BISTEST;
```

A.4 syn_script_counter.tcl

```
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/ha.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/fd.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/counter.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab1/tb_counter.vhd

vsim -t 100ps -novopt work.testcount(test)

add wave *
```

```
power add /testcount/UCOUNTER1/*
run 520 ns
power report -file ./reports/report_power_counter.txt

quit
```

A.5 report_power_counter.txt

```
Power Report Interval
520000 ps

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

A.6 report_power_counter_FF.txt

```
Power Report Interval
520000 ps

Power Report                          Node      Tc      Ti     Time At 1     Time At 0     Time At X
-------------------------------------------------------------------------------------------------------
                  /testcount/ucounter1/a       1       0     514000 ps       6000 ps          0 ps
                 /testcount/ucounter1/ck     520       0     260000 ps     260000 ps          0 ps
              /testcount/ucounter1/reset       1       0       2000 ps     518000 ps          0 ps
               /testcount/ucounter1/s(7)       2       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(6)       4       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(5)       8       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(4)      16       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(3)      32       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(2)      64       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(1)     128       0     256000 ps     264000 ps          0 ps
               /testcount/ucounter1/s(0)     257       0     257000 ps     263000 ps          0 ps
                 /testcount/ucounter1/co       2       0       2000 ps     518000 ps          0 ps
```

```
        /testcount/ucounter1/stmp(7)     30      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(6)     52      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(5)     88      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(4)    144      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(3)    224      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(2)    320      0    256000 ps    263600 ps        400 ps
        /testcount/ucounter1/stmp(1)    385      0    256600 ps    263000 ps        400 ps
        /testcount/ucounter1/stmp(0)    258      0    257000 ps    262800 ps        200 ps
        /testcount/ucounter1/ctmp(8)     16      0      2000 ps    517800 ps        200 ps
        /testcount/ucounter1/ctmp(7)     28      0      4000 ps    515800 ps        200 ps
        /testcount/ucounter1/ctmp(6)     48      0      8000 ps    511800 ps        200 ps
        /testcount/ucounter1/ctmp(5)     80      0     16000 ps    503800 ps        200 ps
        /testcount/ucounter1/ctmp(4)    128      0     32000 ps    487800 ps        200 ps
        /testcount/ucounter1/ctmp(3)    192      0     64000 ps    455800 ps        200 ps
        /testcount/ucounter1/ctmp(2)    256      0    128000 ps    391800 ps        200 ps
        /testcount/ucounter1/ctmp(1)    257      0    256800 ps    263000 ps        200 ps
        /testcount/ucounter1/ctmp(0)      1      0    514000 ps      6000 ps          0 ps
    /testcount/ucounter1/stmpsync(7)      2      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(6)      4      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(5)      8      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(4)     16      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(3)     32      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(2)     64      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(1)    128      0    256000 ps    264000 ps          0 ps
    /testcount/ucounter1/stmpsync(0)    257      0    257000 ps    263000 ps          0 ps
```

Appendice B

Lab 2

B.1 fsm_adder.vhd

```
library IEEE;
use IEEE.std_logic_1164.all; --  libreria IEEE con definizione tipi standard logic
use IEEE.std_logic_signed.all;

entity fsm_adder is
    port(
            clock:  in std_logic;
            reset:  in std_logic;
            SEL00:  out std_logic;
            SEL01:  out std_logic;
            SEL10:  out std_logic;
            SEL11:  out std_logic
            );
end fsm_adder;

architecture behavioral of fsm_adder is -- enumerated type state machine encoding

    type states is (S1, S2, S3, S4, S5);
    attribute enum_encoding: string;
    attribute enum_encoding of states: TYPE IS "000 001 011 111 110";
    signal current_state: states;
    signal next_state: states;

begin

    P_CURRENT_STATE: process(clock, reset)
    begin
        if reset = '1' then
            current_state <= S1;
        elsif (clock='1' and clock'event) then
            current_state <= next_state;
        end if;
    end process P_CURRENT_STATE;

    P_NEXT_STATE: process(current_state)
    begin
        case current_state is
            when S1 => next_state <= S2;
            when S2 => next_state <= S3;
            when S3 => next_state <= S4;
            when S4 => next_state <= S5;
            when S5 => next_state <= S1;
        end case;
    end process P_NEXT_STATE;

    P_OUTPUTS: process(current_state)
    begin
        case current_state is
            when S1 => SEL00<='0'; SEL01<='0'; SEL10<='0'; SEL11<='0'; -- A+B=01
            when S2 => SEL00<='0'; SEL01<='1'; SEL10<='0'; SEL11<='1'; -- 01+C=02
            when S3 => SEL00<='1'; SEL01<='1'; SEL10<='0'; SEL11<='1'; -- 02+D=03
            when S4 => SEL00<='1'; SEL01<='0'; SEL10<='1'; SEL11<='1'; -- 03+E=04
            when S5 => SEL00<='1'; SEL01<='0'; SEL10<='1'; SEL11<='0'; -- 04+F=SUM
        end case;
    end process P_OUTPUTS;

end behavioral;

configuration CFG_FSM_ADDER of fsm_adder is
    for behavioral
```

```
    end for;
end CFG_FSM_ADDER;
```

B.2 script.sh

```
rm -r work

setmentor
vlib work

#vsim -c -do sim_script.tcl
vsim -do sim_script.tcl
```

B.3 sim_script.tcl

```
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab2/dpadder.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab2/fsm-adder.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab2/top.vhd
vcom -reportprogress 300 -work work /home/lp19.21/Documents/lab2/tb_adder.vhd

vsim -t 100ps -novopt work.tbadd(test)
#add wave *
#add wave -r sim:/tbadd/top_test/*
source wave.do

power add *
run 6.5 ns
power report -file ./reports/report_power.txt

restart -f
power add /tbadd/top_test/fsm/*
run 6.5 ns
power report -file ./reports/report_power_fsm.txt

restart -f
power add /tbadd/top_test/datapath/*
run 6.5 ns
power report -file ./reports/report_power_datapath.txt

#quit
```

B.4 syn_script.sh

```
cp /home/lp19.21/Documents/setup/syn/.synopsys_dc.setup .
rm -r work

mkdir work
setsynopsys

design_vision -f syn_script_basic.tcl
#design_vision -f syn_script_faster.tcl

#dc_shell-xg-t -f syn_script.tcl
```

B.5 top_simple.vhd

```
library IEEE;

use IEEE.std_logic_1164.all;

package CONV_PACK_top is
```

```
-- define attributes
attribute ENUM_ENCODING : STRING;

end CONV_PACK_top;

library IEEE;

use IEEE.std_logic_1164.all;

use work.CONV_PACK_top.all;

entity datapath_adder_DW01_add_0 is

   port( A, B : in std_logic_vector (15 downto 0);  CI : in std_logic;  SUM :
         out std_logic_vector (15 downto 0);  CO : out std_logic);

end datapath_adder_DW01_add_0;

architecture SYN_rpl of datapath_adder_DW01_add_0 is

   component XOR2_X1
      port( A, B : in std_logic;  Z : out std_logic);
   end component;

   component AND2_X1
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component FA_X1
      port( A, B, CI : in std_logic;  CO, S : out std_logic);
   end component;

   signal carry_15_port, carry_14_port, carry_13_port, carry_12_port,
      carry_11_port, carry_10_port, carry_9_port, carry_8_port, carry_7_port,
      carry_6_port, carry_5_port, carry_4_port, carry_3_port, carry_2_port, n1,
      n_1002 : std_logic;

begin

   U1_15 : FA_X1 port map( A => A(15), B => B(15), CI => carry_15_port, CO =>
                           n_1002, S => SUM(15));
   U1_14 : FA_X1 port map( A => A(14), B => B(14), CI => carry_14_port, CO =>
                           carry_15_port, S => SUM(14));
   U1_13 : FA_X1 port map( A => A(13), B => B(13), CI => carry_13_port, CO =>
                           carry_14_port, S => SUM(13));
   U1_12 : FA_X1 port map( A => A(12), B => B(12), CI => carry_12_port, CO =>
                           carry_13_port, S => SUM(12));
   U1_11 : FA_X1 port map( A => A(11), B => B(11), CI => carry_11_port, CO =>
                           carry_12_port, S => SUM(11));
   U1_10 : FA_X1 port map( A => A(10), B => B(10), CI => carry_10_port, CO =>
                           carry_11_port, S => SUM(10));
   U1_9 : FA_X1 port map( A => A(9), B => B(9), CI => carry_9_port, CO =>
                          carry_10_port, S => SUM(9));
   U1_8 : FA_X1 port map( A => A(8), B => B(8), CI => carry_8_port, CO =>
                          carry_9_port, S => SUM(8));
   U1_7 : FA_X1 port map( A => A(7), B => B(7), CI => carry_7_port, CO =>
                          carry_8_port, S => SUM(7));
   U1_6 : FA_X1 port map( A => A(6), B => B(6), CI => carry_6_port, CO =>
                          carry_7_port, S => SUM(6));
   U1_5 : FA_X1 port map( A => A(5), B => B(5), CI => carry_5_port, CO =>
                          carry_6_port, S => SUM(5));
   U1_4 : FA_X1 port map( A => A(4), B => B(4), CI => carry_4_port, CO =>
                          carry_5_port, S => SUM(4));
   U1_3 : FA_X1 port map( A => A(3), B => B(3), CI => carry_3_port, CO =>
                          carry_4_port, S => SUM(3));
   U1_2 : FA_X1 port map( A => A(2), B => B(2), CI => carry_2_port, CO =>
                          carry_3_port, S => SUM(2));
   U1_1 : FA_X1 port map( A => A(1), B => B(1), CI => n1, CO => carry_2_port, S
                         => SUM(1));
   U1 : AND2_X1 port map( A1 => B(0), A2 => A(0), ZN => n1);
   U2 : XOR2_X1 port map( A => B(0), B => A(0), Z => SUM(0));

end SYN_rpl;

library IEEE;

use IEEE.std_logic_1164.all;

use work.CONV_PACK_top.all;

entity fsm_adder is

   port( clock, reset : in std_logic;  SEL00, SEL01, SEL10, SEL11 : out
```

```
        std_logic);

end fsm_adder;

architecture SYN_behavioral of fsm_adder is

   component MUX2_X1
      port( A, B, S : in std_logic;  Z : out std_logic);
   end component;

   component INV_X1
      port( A : in std_logic;  ZN : out std_logic);
   end component;

   component NOR2_X1
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component OAI21_X1
      port( B1, B2, A : in std_logic;  ZN : out std_logic);
   end component;

   component DFFR_X1
      port( D, CK, RN : in std_logic;  Q, QN : out std_logic);
   end component;

   signal SEL01_port, SEL10_port, current_state_0_port, N3, n4, n6, n1, n2,
      n3_port, n5, n7, n8, n_1003 : std_logic;

begin
   SEL01 <= SEL01_port;
   SEL10 <= SEL10_port;

   current_state_reg_0_inst : DFFR_X1 port map( D => n4, CK => clock, RN => n8,
                           Q => current_state_0_port, QN => n6);
   current_state_reg_1_inst : DFFR_X1 port map( D => current_state_0_port, CK
                           => clock, RN => n8, Q => n_1003, QN => n1);
   current_state_reg_2_inst : DFFR_X1 port map( D => N3, CK => clock, RN => n8,
                           Q => n2, QN => n4);
   U3 : INV_X1 port map( A => reset, ZN => n8);
   U4 : OAI21_X1 port map( B1 => n6, B2 => n1, A => n3_port, ZN => SEL11);
   U5 : INV_X1 port map( A => SEL01_port, ZN => n3_port);
   U6 : NOR2_X1 port map( A1 => n2, A2 => n6, ZN => SEL01_port);
   U7 : OAI21_X1 port map( B1 => n6, B2 => n1, A => n5, ZN => SEL00);
   U8 : INV_X1 port map( A => SEL10_port, ZN => n5);
   U9 : NOR2_X1 port map( A1 => n1, A2 => n4, ZN => SEL10_port);
   U10 : INV_X1 port map( A => n7, ZN => N3);
   U11 : MUX2_X1 port map( A => n6, B => n1, S => n4, Z => n7);

end SYN_behavioral;

library IEEE;

use IEEE.std_logic_1164.all;

use work.CONV_PACK_top.all;

entity datapath_adder is

   port( MUX00, MUX01, MUX02, MUX03, MUX10, MUX11, MUX12, MUX13 : in
         std_logic_vector (15 downto 0);  clock, reset, SEL00, SEL01, SEL10,
         SEL11 : in std_logic;  SUM : out std_logic_vector (15 downto 0));

end datapath_adder;

architecture SYN_behavioral of datapath_adder is

   component INV_X1
      port( A : in std_logic;  ZN : out std_logic);
   end component;

   component AOI22_X1
      port( A1, A2, B1, B2 : in std_logic;  ZN : out std_logic);
   end component;

   component AND2_X1
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component NAND2_X1
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component NOR2_X2
```

```
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component datapath_adder_DW01_add_0
      port( A, B : in std_logic_vector (15 downto 0);  CI : in std_logic;  SUM
            : out std_logic_vector (15 downto 0);  CO : out std_logic);
   end component;

   component DFFR_X1
      port( D, CK, RN : in std_logic;  Q, QN : out std_logic);
   end component;

   signal operanda_15_port, operanda_14_port, operanda_13_port,
      operanda_12_port, operanda_11_port, operanda_10_port, operanda_9_port,
      operanda_8_port, operanda_7_port, operanda_6_port, operanda_5_port,
      operanda_4_port, operanda_3_port, operanda_2_port, operanda_1_port,
      operanda_0_port, operandb_15_port, operandb_14_port, operandb_13_port,
      operandb_12_port, operandb_11_port, operandb_10_port, operandb_9_port,
      operandb_8_port, operandb_7_port, operandb_6_port, operandb_5_port,
      operandb_4_port, operandb_3_port, operandb_2_port, operandb_1_port,
      operandb_0_port, N25, N26, N27, N28, N29, N30, N31, N32, N33, N34, N35,
      N36, N37, N38, N39, N40, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11,
      n12, n13, n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25_port
      , n26_port, n27_port, n28_port, n29_port, n30_port, n31_port, n32_port,
      n33_port, n34_port, n35_port, n36_port, n37_port, n38_port, n39_port,
      n40_port, n41, n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53
      , n54, n55, n56, n57, n58, n59, n60, n61, n62, n63, n64, n65, n66, n67,
      n68, n69, n70, n71, n72, n73, n74, n75, n76, n_1004, n_1005, n_1006,
      n_1007, n_1008, n_1009, n_1010, n_1011, n_1012, n_1013, n_1014, n_1015,
      n_1016, n_1017, n_1018, n_1019, n_1020 : std_logic;

begin

   n1 <= '0';
   SUM_reg_15_inst : DFFR_X1 port map( D => N40, CK => clock, RN => n76, Q =>
                           SUM(15), QN => n_1004);
   SUM_reg_14_inst : DFFR_X1 port map( D => N39, CK => clock, RN => n76, Q =>
                           SUM(14), QN => n_1005);
   SUM_reg_13_inst : DFFR_X1 port map( D => N38, CK => clock, RN => n76, Q =>
                           SUM(13), QN => n_1006);
   SUM_reg_12_inst : DFFR_X1 port map( D => N37, CK => clock, RN => n76, Q =>
                           SUM(12), QN => n_1007);
   SUM_reg_11_inst : DFFR_X1 port map( D => N36, CK => clock, RN => n76, Q =>
                           SUM(11), QN => n_1008);
   SUM_reg_10_inst : DFFR_X1 port map( D => N35, CK => clock, RN => n76, Q =>
                           SUM(10), QN => n_1009);
   SUM_reg_9_inst : DFFR_X1 port map( D => N34, CK => clock, RN => n76, Q =>
                           SUM(9), QN => n_1010);
   SUM_reg_8_inst : DFFR_X1 port map( D => N33, CK => clock, RN => n76, Q =>
                           SUM(8), QN => n_1011);
   SUM_reg_7_inst : DFFR_X1 port map( D => N32, CK => clock, RN => n76, Q =>
                           SUM(7), QN => n_1012);
   SUM_reg_6_inst : DFFR_X1 port map( D => N31, CK => clock, RN => n76, Q =>
                           SUM(6), QN => n_1013);
   SUM_reg_5_inst : DFFR_X1 port map( D => N30, CK => clock, RN => n76, Q =>
                           SUM(5), QN => n_1014);
   SUM_reg_4_inst : DFFR_X1 port map( D => N29, CK => clock, RN => n76, Q =>
                           SUM(4), QN => n_1015);
   SUM_reg_3_inst : DFFR_X1 port map( D => N28, CK => clock, RN => n76, Q =>
                           SUM(3), QN => n_1016);
   SUM_reg_2_inst : DFFR_X1 port map( D => N27, CK => clock, RN => n76, Q =>
                           SUM(2), QN => n_1017);
   SUM_reg_1_inst : DFFR_X1 port map( D => N26, CK => clock, RN => n76, Q =>
                           SUM(1), QN => n_1018);
   SUM_reg_0_inst : DFFR_X1 port map( D => N25, CK => clock, RN => n76, Q =>
                           SUM(0), QN => n_1019);
   add_81 : datapath_adder_DW01_add_0 port map( A(15) => operanda_15_port,
                           A(14) => operanda_14_port, A(13) => operanda_13_port
                           , A(12) => operanda_12_port, A(11) =>
                           operanda_11_port, A(10) => operanda_10_port, A(9) =>
                           operanda_9_port, A(8) => operanda_8_port, A(7) =>
                           operanda_7_port, A(6) => operanda_6_port, A(5) =>
                           operanda_5_port, A(4) => operanda_4_port, A(3) =>
                           operanda_3_port, A(2) => operanda_2_port, A(1) =>
                           operanda_1_port, A(0) => operanda_0_port, B(15) =>
                           operandb_15_port, B(14) => operandb_14_port, B(13)
                           => operandb_13_port, B(12) => operandb_12_port,
                           B(11) => operandb_11_port, B(10) => operandb_10_port
                           , B(9) => operandb_9_port, B(8) => operandb_8_port,
                           B(7) => operandb_7_port, B(6) => operandb_6_port,
                           B(5) => operandb_5_port, B(4) => operandb_4_port,
                           B(3) => operandb_3_port, B(2) => operandb_2_port,
                           B(1) => operandb_1_port, B(0) => operandb_0_port, CI
                           => n1, SUM(15) => N40, SUM(14) => N39, SUM(13) =>
```

```
                        N38, SUM(12) => N37, SUM(11) => N36, SUM(10) => N35,
                        SUM(9) => N34, SUM(8) => N33, SUM(7) => N32, SUM(6)
                        => N31, SUM(5) => N30, SUM(4) => N29, SUM(3) => N28,
                        SUM(2) => N27, SUM(1) => N26, SUM(0) => N25, CO =>
                        n_1020);
U3 : NOR2_X2 port map( A1 => n75, A2 => SEL00, ZN => n44);
U5 : NOR2_X2 port map( A1 => n38_port, A2 => SEL10, ZN => n7);
U6 : NOR2_X2 port map( A1 => SEL00, A2 => SEL01, ZN => n43);
U7 : NOR2_X2 port map( A1 => SEL10, A2 => SEL11, ZN => n6);
U8 : NAND2_X1 port map( A1 => n2, A2 => n3, ZN => operandb_9_port);
U9 : AOI22_X1 port map( A1 => MUX12(9), A2 => n4, B1 => MUX13(9), B2 => n5,
                        ZN => n3);
U10 : AOI22_X1 port map( A1 => MUX10(9), A2 => n6, B1 => MUX11(9), B2 => n7,
                        ZN => n2);
U11 : NAND2_X1 port map( A1 => n8, A2 => n9, ZN => operandb_8_port);
U12 : AOI22_X1 port map( A1 => MUX12(8), A2 => n4, B1 => MUX13(8), B2 => n5,
                        ZN => n9);
U13 : AOI22_X1 port map( A1 => MUX10(8), A2 => n6, B1 => MUX11(8), B2 => n7,
                        ZN => n8);
U14 : NAND2_X1 port map( A1 => n10, A2 => n11, ZN => operandb_7_port);
U15 : AOI22_X1 port map( A1 => MUX12(7), A2 => n4, B1 => MUX13(7), B2 => n5,
                        ZN => n11);
U16 : AOI22_X1 port map( A1 => MUX10(7), A2 => n6, B1 => MUX11(7), B2 => n7,
                        ZN => n10);
U17 : NAND2_X1 port map( A1 => n12, A2 => n13, ZN => operandb_6_port);
U18 : AOI22_X1 port map( A1 => MUX12(6), A2 => n4, B1 => MUX13(6), B2 => n5,
                        ZN => n13);
U19 : AOI22_X1 port map( A1 => MUX10(6), A2 => n6, B1 => MUX11(6), B2 => n7,
                        ZN => n12);
U20 : NAND2_X1 port map( A1 => n14, A2 => n15, ZN => operandb_5_port);
U21 : AOI22_X1 port map( A1 => MUX12(5), A2 => n4, B1 => MUX13(5), B2 => n5,
                        ZN => n15);
U22 : AOI22_X1 port map( A1 => MUX10(5), A2 => n6, B1 => MUX11(5), B2 => n7,
                        ZN => n14);
U23 : NAND2_X1 port map( A1 => n16, A2 => n17, ZN => operandb_4_port);
U24 : AOI22_X1 port map( A1 => MUX12(4), A2 => n4, B1 => MUX13(4), B2 => n5,
                        ZN => n17);
U25 : AOI22_X1 port map( A1 => MUX10(4), A2 => n6, B1 => MUX11(4), B2 => n7,
                        ZN => n16);
U26 : NAND2_X1 port map( A1 => n18, A2 => n19, ZN => operandb_3_port);
U27 : AOI22_X1 port map( A1 => MUX12(3), A2 => n4, B1 => MUX13(3), B2 => n5,
                        ZN => n19);
U28 : AOI22_X1 port map( A1 => MUX10(3), A2 => n6, B1 => MUX11(3), B2 => n7,
                        ZN => n18);
U29 : NAND2_X1 port map( A1 => n20, A2 => n21, ZN => operandb_2_port);
U30 : AOI22_X1 port map( A1 => MUX12(2), A2 => n4, B1 => MUX13(2), B2 => n5,
                        ZN => n21);
U31 : AOI22_X1 port map( A1 => MUX10(2), A2 => n6, B1 => MUX11(2), B2 => n7,
                        ZN => n20);
U32 : NAND2_X1 port map( A1 => n22, A2 => n23, ZN => operandb_1_port);
U33 : AOI22_X1 port map( A1 => MUX12(1), A2 => n4, B1 => MUX13(1), B2 => n5,
                        ZN => n23);
U34 : AOI22_X1 port map( A1 => MUX10(1), A2 => n6, B1 => MUX11(1), B2 => n7,
                        ZN => n22);
U35 : NAND2_X1 port map( A1 => n24, A2 => n25_port, ZN => operandb_15_port);
U36 : AOI22_X1 port map( A1 => MUX12(15), A2 => n4, B1 => MUX13(15), B2 =>
                        n5, ZN => n25_port);
U37 : AOI22_X1 port map( A1 => MUX10(15), A2 => n6, B1 => MUX11(15), B2 =>
                        n7, ZN => n24);
U38 : NAND2_X1 port map( A1 => n26_port, A2 => n27_port, ZN =>
                        operandb_14_port);
U39 : AOI22_X1 port map( A1 => MUX12(14), A2 => n4, B1 => MUX13(14), B2 =>
                        n5, ZN => n27_port);
U40 : AOI22_X1 port map( A1 => MUX10(14), A2 => n6, B1 => MUX11(14), B2 =>
                        n7, ZN => n26_port);
U41 : NAND2_X1 port map( A1 => n28_port, A2 => n29_port, ZN =>
                        operandb_13_port);
U42 : AOI22_X1 port map( A1 => MUX12(13), A2 => n4, B1 => MUX13(13), B2 =>
                        n5, ZN => n29_port);
U43 : AOI22_X1 port map( A1 => MUX10(13), A2 => n6, B1 => MUX11(13), B2 =>
                        n7, ZN => n28_port);
U44 : NAND2_X1 port map( A1 => n30_port, A2 => n31_port, ZN =>
                        operandb_12_port);
U45 : AOI22_X1 port map( A1 => MUX12(12), A2 => n4, B1 => MUX13(12), B2 =>
                        n5, ZN => n31_port);
U46 : AOI22_X1 port map( A1 => MUX10(12), A2 => n6, B1 => MUX11(12), B2 =>
                        n7, ZN => n30_port);
U47 : NAND2_X1 port map( A1 => n32_port, A2 => n33_port, ZN =>
                        operandb_11_port);
U48 : AOI22_X1 port map( A1 => MUX12(11), A2 => n4, B1 => MUX13(11), B2 =>
                        n5, ZN => n33_port);
U49 : AOI22_X1 port map( A1 => MUX10(11), A2 => n6, B1 => MUX11(11), B2 =>
                        n7, ZN => n32_port);
U50 : NAND2_X1 port map( A1 => n34_port, A2 => n35_port, ZN =>
```

```
                        operandb_10_port);
U51 : AOI22_X1 port map( A1 => MUX12(10), A2 => n4, B1 => MUX13(10), B2 =>
                        n5, ZN => n35_port);
U52 : AOI22_X1 port map( A1 => MUX10(10), A2 => n6, B1 => MUX11(10), B2 =>
                        n7, ZN => n34_port);
U53 : NAND2_X1 port map( A1 => n36_port, A2 => n37_port, ZN =>
                        operandb_0_port);
U54 : AOI22_X1 port map( A1 => MUX12(0), A2 => n4, B1 => MUX13(0), B2 => n5,
                        ZN => n37_port);
U55 : AND2_X1 port map( A1 => SEL10, A2 => n38_port, ZN => n5);
U56 : AND2_X1 port map( A1 => SEL11, A2 => SEL10, ZN => n4);
U57 : AOI22_X1 port map( A1 => MUX10(0), A2 => n6, B1 => MUX11(0), B2 => n7,
                        ZN => n36_port);
U58 : INV_X1 port map( A => SEL11, ZN => n38_port);
U59 : NAND2_X1 port map( A1 => n39_port, A2 => n40_port, ZN =>
                        operanda_9_port);
U60 : AOI22_X1 port map( A1 => MUX02(9), A2 => n41, B1 => MUX03(9), B2 =>
                        n42, ZN => n40_port);
U61 : AOI22_X1 port map( A1 => MUX00(9), A2 => n43, B1 => MUX01(9), B2 =>
                        n44, ZN => n39_port);
U62 : NAND2_X1 port map( A1 => n45, A2 => n46, ZN => operanda_8_port);
U63 : AOI22_X1 port map( A1 => MUX02(8), A2 => n41, B1 => MUX03(8), B2 =>
                        n42, ZN => n46);
U64 : AOI22_X1 port map( A1 => MUX00(8), A2 => n43, B1 => MUX01(8), B2 =>
                        n44, ZN => n45);
U65 : NAND2_X1 port map( A1 => n47, A2 => n48, ZN => operanda_7_port);
U66 : AOI22_X1 port map( A1 => MUX02(7), A2 => n41, B1 => MUX03(7), B2 =>
                        n42, ZN => n48);
U67 : AOI22_X1 port map( A1 => MUX00(7), A2 => n43, B1 => MUX01(7), B2 =>
                        n44, ZN => n47);
U68 : NAND2_X1 port map( A1 => n49, A2 => n50, ZN => operanda_6_port);
U69 : AOI22_X1 port map( A1 => MUX02(6), A2 => n41, B1 => MUX03(6), B2 =>
                        n42, ZN => n50);
U70 : AOI22_X1 port map( A1 => MUX00(6), A2 => n43, B1 => MUX01(6), B2 =>
                        n44, ZN => n49);
U71 : NAND2_X1 port map( A1 => n51, A2 => n52, ZN => operanda_5_port);
U72 : AOI22_X1 port map( A1 => MUX02(5), A2 => n41, B1 => MUX03(5), B2 =>
                        n42, ZN => n52);
U73 : AOI22_X1 port map( A1 => MUX00(5), A2 => n43, B1 => MUX01(5), B2 =>
                        n44, ZN => n51);
U74 : NAND2_X1 port map( A1 => n53, A2 => n54, ZN => operanda_4_port);
U75 : AOI22_X1 port map( A1 => MUX02(4), A2 => n41, B1 => MUX03(4), B2 =>
                        n42, ZN => n54);
U76 : AOI22_X1 port map( A1 => MUX00(4), A2 => n43, B1 => MUX01(4), B2 =>
                        n44, ZN => n53);
U77 : NAND2_X1 port map( A1 => n55, A2 => n56, ZN => operanda_3_port);
U78 : AOI22_X1 port map( A1 => MUX02(3), A2 => n41, B1 => MUX03(3), B2 =>
                        n42, ZN => n56);
U79 : AOI22_X1 port map( A1 => MUX00(3), A2 => n43, B1 => MUX01(3), B2 =>
                        n44, ZN => n55);
U80 : NAND2_X1 port map( A1 => n57, A2 => n58, ZN => operanda_2_port);
U81 : AOI22_X1 port map( A1 => MUX02(2), A2 => n41, B1 => MUX03(2), B2 =>
                        n42, ZN => n58);
U82 : AOI22_X1 port map( A1 => MUX00(2), A2 => n43, B1 => MUX01(2), B2 =>
                        n44, ZN => n57);
U83 : NAND2_X1 port map( A1 => n59, A2 => n60, ZN => operanda_1_port);
U84 : AOI22_X1 port map( A1 => MUX02(1), A2 => n41, B1 => MUX03(1), B2 =>
                        n42, ZN => n60);
U85 : AOI22_X1 port map( A1 => MUX00(1), A2 => n43, B1 => MUX01(1), B2 =>
                        n44, ZN => n59);
U86 : NAND2_X1 port map( A1 => n61, A2 => n62, ZN => operanda_15_port);
U87 : AOI22_X1 port map( A1 => MUX02(15), A2 => n41, B1 => MUX03(15), B2 =>
                        n42, ZN => n62);
U88 : AOI22_X1 port map( A1 => MUX00(15), A2 => n43, B1 => MUX01(15), B2 =>
                        n44, ZN => n61);
U89 : NAND2_X1 port map( A1 => n63, A2 => n64, ZN => operanda_14_port);
U90 : AOI22_X1 port map( A1 => MUX02(14), A2 => n41, B1 => MUX03(14), B2 =>
                        n42, ZN => n64);
U91 : AOI22_X1 port map( A1 => MUX00(14), A2 => n43, B1 => MUX01(14), B2 =>
                        n44, ZN => n63);
U92 : NAND2_X1 port map( A1 => n65, A2 => n66, ZN => operanda_13_port);
U93 : AOI22_X1 port map( A1 => MUX02(13), A2 => n41, B1 => MUX03(13), B2 =>
                        n42, ZN => n66);
U94 : AOI22_X1 port map( A1 => MUX00(13), A2 => n43, B1 => MUX01(13), B2 =>
                        n44, ZN => n65);
U95 : NAND2_X1 port map( A1 => n67, A2 => n68, ZN => operanda_12_port);
U96 : AOI22_X1 port map( A1 => MUX02(12), A2 => n41, B1 => MUX03(12), B2 =>
                        n42, ZN => n68);
U97 : AOI22_X1 port map( A1 => MUX00(12), A2 => n43, B1 => MUX01(12), B2 =>
                        n44, ZN => n67);
U98 : NAND2_X1 port map( A1 => n69, A2 => n70, ZN => operanda_11_port);
U99 : AOI22_X1 port map( A1 => MUX02(11), A2 => n41, B1 => MUX03(11), B2 =>
                        n42, ZN => n70);
U100 : AOI22_X1 port map( A1 => MUX00(11), A2 => n43, B1 => MUX01(11), B2 =>
```

```
                           n44, ZN => n69);
   U101 : NAND2_X1 port map( A1 => n71, A2 => n72, ZN => operanda_10_port);
   U102 : AOI22_X1 port map( A1 => MUX02(10), A2 => n41, B1 => MUX03(10), B2 =>
                           n42, ZN => n72);
   U103 : AOI22_X1 port map( A1 => MUX00(10), A2 => n43, B1 => MUX01(10), B2 =>
                           n44, ZN => n71);
   U104 : NAND2_X1 port map( A1 => n73, A2 => n74, ZN => operanda_0_port);
   U105 : AOI22_X1 port map( A1 => MUX02(0), A2 => n41, B1 => MUX03(0), B2 =>
                           n42, ZN => n74);
   U106 : AND2_X1 port map( A1 => SEL00, A2 => n75, ZN => n42);
   U107 : AND2_X1 port map( A1 => SEL01, A2 => SEL00, ZN => n41);
   U108 : AOI22_X1 port map( A1 => MUX00(0), A2 => n43, B1 => MUX01(0), B2 =>
                           n44, ZN => n73);
   U109 : INV_X1 port map( A => SEL01, ZN => n75);
   U110 : INV_X1 port map( A => reset, ZN => n76);

end SYN_behavioral;

library IEEE;

use IEEE.std_logic_1164.all;

use work.CONV_PACK_top.all;

entity top is

   port( clock, reset : in std_logic;  A, B, C, D, E, F : in std_logic_vector
         (15 downto 0);  SUM : out std_logic_vector (15 downto 0));

end top;

architecture SYN_structural of top is

   component fsm_adder
      port( clock, reset : in std_logic;  SEL00, SEL01, SEL10, SEL11 : out
            std_logic);
   end component;

   component datapath_adder
      port( MUX00, MUX01, MUX02, MUX03, MUX10, MUX11, MUX12, MUX13 : in
            std_logic_vector (15 downto 0);  clock, reset, SEL00, SEL01, SEL10,
            SEL11 : in std_logic;  SUM : out std_logic_vector (15 downto 0));
   end component;

   signal SUM_15_port, SUM_14_port, SUM_13_port, SUM_12_port, SUM_11_port,
      SUM_10_port, SUM_9_port, SUM_8_port, SUM_7_port, SUM_6_port, SUM_5_port,
      SUM_4_port, SUM_3_port, SUM_2_port, SUM_1_port, SUM_0_port, sel00, sel01,
      sel10, sel11 : std_logic;

begin
   SUM <= ( SUM_15_port, SUM_14_port, SUM_13_port, SUM_12_port, SUM_11_port,
      SUM_10_port, SUM_9_port, SUM_8_port, SUM_7_port, SUM_6_port, SUM_5_port,
      SUM_4_port, SUM_3_port, SUM_2_port, SUM_1_port, SUM_0_port );

   DATAPATH : datapath_adder port map( MUX00(15) => A(15), MUX00(14) => A(14),
                           MUX00(13) => A(13), MUX00(12) => A(12), MUX00(11) =>
                           A(11), MUX00(10) => A(10), MUX00(9) => A(9),
                           MUX00(8) => A(8), MUX00(7) => A(7), MUX00(6) => A(6)
                           , MUX00(5) => A(5), MUX00(4) => A(4), MUX00(3) =>
                           A(3), MUX00(2) => A(2), MUX00(1) => A(1), MUX00(0)
                           => A(0), MUX01(15) => C(15), MUX01(14) => C(14),
                           MUX01(13) => C(13), MUX01(12) => C(12), MUX01(11) =>
                           C(11), MUX01(10) => C(10), MUX01(9) => C(9),
                           MUX01(8) => C(8), MUX01(7) => C(7), MUX01(6) => C(6)
                           , MUX01(5) => C(5), MUX01(4) => C(4), MUX01(3) =>
                           C(3), MUX01(2) => C(2), MUX01(1) => C(1), MUX01(0)
                           => C(0), MUX02(15) => D(15), MUX02(14) => D(14),
                           MUX02(13) => D(13), MUX02(12) => D(12), MUX02(11) =>
                           D(11), MUX02(10) => D(10), MUX02(9) => D(9),
                           MUX02(8) => D(8), MUX02(7) => D(7), MUX02(6) => D(6)
                           , MUX02(5) => D(5), MUX02(4) => D(4), MUX02(3) =>
                           D(3), MUX02(2) => D(2), MUX02(1) => D(1), MUX02(0)
                           => D(0), MUX03(15) => SUM_15_port, MUX03(14) =>
                           SUM_14_port, MUX03(13) => SUM_13_port, MUX03(12) =>
                           SUM_12_port, MUX03(11) => SUM_11_port, MUX03(10) =>
                           SUM_10_port, MUX03(9) => SUM_9_port, MUX03(8) =>
                           SUM_8_port, MUX03(7) => SUM_7_port, MUX03(6) =>
                           SUM_6_port, MUX03(5) => SUM_5_port, MUX03(4) =>
                           SUM_4_port, MUX03(3) => SUM_3_port, MUX03(2) =>
                           SUM_2_port, MUX03(1) => SUM_1_port, MUX03(0) =>
                           SUM_0_port, MUX10(15) => B(15), MUX10(14) => B(14),
                           MUX10(13) => B(13), MUX10(12) => B(12), MUX10(11) =>
                           B(11), MUX10(10) => B(10), MUX10(9) => B(9),
                           MUX10(8) => B(8), MUX10(7) => B(7), MUX10(6) => B(6)
```

```
                          , MUX10(5) => B(5), MUX10(4) => B(4), MUX10(3) =>
                          B(3), MUX10(2) => B(2), MUX10(1) => B(1), MUX10(0)
                          => B(0), MUX11(15) => SUM_15_port, MUX11(14) =>
                          SUM_14_port, MUX11(13) => SUM_13_port, MUX11(12) =>
                          SUM_12_port, MUX11(11) => SUM_11_port, MUX11(10) =>
                          SUM_10_port, MUX11(9) => SUM_9_port, MUX11(8) =>
                          SUM_8_port, MUX11(7) => SUM_7_port, MUX11(6) =>
                          SUM_6_port, MUX11(5) => SUM_5_port, MUX11(4) =>
                          SUM_4_port, MUX11(3) => SUM_3_port, MUX11(2) =>
                          SUM_2_port, MUX11(1) => SUM_1_port, MUX11(0) =>
                          SUM_0_port, MUX12(15) => E(15), MUX12(14) => E(14),
                          MUX12(13) => E(13), MUX12(12) => E(12), MUX12(11) =>
                          E(11), MUX12(10) => E(10), MUX12(9) => E(9),
                          MUX12(8) => E(8), MUX12(7) => E(7), MUX12(6) => E(6)
                          , MUX12(5) => E(5), MUX12(4) => E(4), MUX12(3) =>
                          E(3), MUX12(2) => E(2), MUX12(1) => E(1), MUX12(0)
                          => E(0), MUX13(15) => F(15), MUX13(14) => F(14),
                          MUX13(13) => F(13), MUX13(12) => F(12), MUX13(11) =>
                          F(11), MUX13(10) => F(10), MUX13(9) => F(9),
                          MUX13(8) => F(8), MUX13(7) => F(7), MUX13(6) => F(6)
                          , MUX13(5) => F(5), MUX13(4) => F(4), MUX13(3) =>
                          F(3), MUX13(2) => F(2), MUX13(1) => F(1), MUX13(0)
                          => F(0), clock => clock, reset => reset, SEL00 =>
                          sel00, SEL01 => sel01, SEL10 => sel10, SEL11 =>
                          sel11, SUM(15) => SUM_15_port, SUM(14) =>
                          SUM_14_port, SUM(13) => SUM_13_port, SUM(12) =>
                          SUM_12_port, SUM(11) => SUM_11_port, SUM(10) =>
                          SUM_10_port, SUM(9) => SUM_9_port, SUM(8) =>
                          SUM_8_port, SUM(7) => SUM_7_port, SUM(6) =>
                          SUM_6_port, SUM(5) => SUM_5_port, SUM(4) =>
                          SUM_4_port, SUM(3) => SUM_3_port, SUM(2) =>
                          SUM_2_port, SUM(1) => SUM_1_port, SUM(0) =>
                          SUM_0_port);
   FSM : fsm_adder port map( clock => clock, reset => reset, SEL00 => sel00,
                          SEL01 => sel01, SEL10 => sel10, SEL11 => sel11);

end SYN_structural;
```

B.6 syn_script_basic.tcl

```
analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab2/fsm-adder.vhd /home/lp19.21/
    Documents/lab2/dpadder.vhd /home/lp19.21/Documents/lab2/top.vhd}
elaborate TOP -architecture STRUCTURAL -library WORK
compile -exact_map

create_clock -name "CLK" -period 10 {clock}

#report_clock > ./reports/basic/report_clock.txt

# #write -hierarchy -format ddc -output /home/lp19.21/Documents/lab2/syn/top_simple.ddc
# #write -hierarchy -format vhdl -output /home/lp19.21/Documents/lab2/syn/top_simple.vhdl
# #read_file -format ddc {/home/lp19.21/Documents/lab2/syn/top_simple.ddc}

#current_design fsm_adder
#report_fsm > ./reports/basic/report_fsm_adder.txt
#report_timing > ./reports/basic/report_timing_fsm_adder.txt
#report_power > ./reports/basic/report_power_fsm_adder.txt
#report_power > ./reports/basic/report_power_summary_adder.txt

#current_design top
#report_area > ./reports/basic/report_area.txt
#report_timing -nworst 10 > ./reports/basic/report_timing_nworst.txt
#report_power > ./reports/basic/report_power_summary.txt
#report_power -hier > ./reports/basic/report_power_cells.txt
#report_power -net > ./reports/basic/report_power_net.txt

#list_design > ./reports/basic/list_design.txt
#
#current_instance FSM
#
#report_power -cell > ./reports/basic/report_power_fsm_cells.txt
#report_power -net > ./reports/basic/report_power_fsm_net.txt

set_max_dynamic_power 35
compile -exact_map
report_power > ./reports/basic/report_power_summary_constrained_238.txt
report_constraints > ./reports/basic/report_constraints_238.txt

#quit
```

B.7 syn_script_faster.tcl

```
analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab2/fsm-adder.vhd /home/lp19.21/
    Documents/lab2/dpadder.vhd /home/lp19.21/Documents/lab2/top.vhd}
elaborate TOP -architecture STRUCTURAL -library WORK
compile -exact_map

create_clock -name "CLK" -period 2 {clock}


current_design fsm_adder
report_fsm > ./reports/faster/report_power_fsm_adder.txt
report_timing > ./reports/faster/report_timing_fsm_adder.txt

current_design top
report_area > ./reports/faster/report_area_top.txt
report_timing -nworst 10 > ./reports/faster/report_timing_nworst_top.txt

report_power > ./reports/faster/report_power_summary.txt
report_power -hier > ./reports/faster/report_power_cells.txt
report_power -net > ./reports/faster/report_power_net.txt

#list_design > list_design.txt

current_instance FSM
report_power -cell > ./reports/faster/report_power_fsm_cells.txt
report_power -net > ./reports/faster/report_power_fsm_net.txt

#quit
```

Appendice C

Lab 3

C.1 syn_script_ckg.tcl

```
analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab1/fa_syn.vhd /home/lp19.21/Documents/
    lab3/inccomp.vhd}
elaborate inccomp -library WORK -architecture behavioral
uniquify
compile -exact_map

create_clock -name clk -period 5 {ck}
report_timing > ./reports/report_timing_1_base_comparator.txt
report_power > ./reports/report_power_1_base_comparator.txt
report_power -include_input_nets > ./reports/report_power_1_base_comparator_input_nets.txt
report_power -net -include_input_nets > ./reports/
    report_power_1_base_comparator_input_nets_only_nets.txt

report_cell > ./reports/report_cell_1_base_comparator.txt

# ------------------------------------------------------------------------------

remove_design -designs

analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab1/fa_syn.vhd /home/lp19.21/Documents/
    lab3/inccomp.vhd}
elaborate inccomp -library WORK -architecture behavioral
uniquify
compile -exact_map

create_clock -name clk -period 5 {ck}

set_switching_activity -static_probability 0.5 -toggle_rate 2 -base_clock clk {ck}
set_switching_activity -static_probability 0 -base_clock clk {rst}

report_timing > ./reports/report_timing_2_changeprobability_comparator.txt
report_power > ./reports/report_power_2_changeprobability_comparator_ckrst.txt
report_power -include_input_nets > ./reports/
    report_power_2_changeprobability_comparator_input_nets_ckrst.txt
report_power -net -include_input_nets > ./reports/
    report_power_2_changeprobability_comparator_input_nets_only_nets_ckrst.txt

set_switching_activity -static_probability 0.12 -toggle_rate 0.025 -base_clock clk {INCA INCB}

report_power > ./reports/report_power_2_changeprobability_comparator_inc.txt
report_power -include_input_nets > ./reports/
    report_power_2_changeprobability_comparator_input_nets_inc.txt
report_power -net -include_input_nets > ./reports/
    report_power_2_changeprobability_comparator_input_nets_only_nets_inc.txt

report_cell > ./reports/report_cell_2_changeprobability_comparator_inc.txt

# ------------------------------------------------------------------------------

remove_design -designs

analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab1/fa_syn.vhd /home/lp19.21/Documents/
    lab3/inccomp.vhd}
elaborate inccomp -library WORK -architecture behavioral
uniquify
compile -exact_map -gate_clock

create_clock -name clk -period 5 {ck}

report_timing > ./reports/report_timing_3_ckg_comparator.txt
```

```
report_power > ./reports/report_power_3_ckg_comparator.txt
report_power -include_input_nets > ./reports/report_power_3_ckg_comparator_input_nets.txt
report_power -net -include_input_nets > ./reports/
    report_power_3_ckg_comparator_input_nets_only_nets.txt

set_switching_activity -static_probability 0.5 -toggle_rate 2 -base_clock clk {ck}
set_switching_activity -static_probability 0 -base_clock clk {rst}

report_power > ./reports/report_power_3_ckg_comparator_ckrst.txt
report_power -include_input_nets > ./reports/report_power_3_ckg_comparator_input_nets_ckrst.txt
report_power -net -include_input_nets > ./reports/
    report_power_3_ckg_comparator_input_nets_only_nets_ckrst.txt

set_switching_activity -static_probability 0.12 -toggle_rate 0.025 -base_clock clk {INCA INCB}

report_power > ./reports/report_power_3_ckg_comparator_inc.txt
report_power -include_input_nets > ./reports/report_power_3_ckg_comparator_input_nets_inc.txt
report_power -net -include_input_nets > ./reports/
    report_power_3_ckg_comparator_input_nets_only_nets_inc.txt

report_cell > ./reports/report_cell_3_ckg_comparator_inc.txt

quit
```

C.2 inccomp_mod.vhd

```
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

    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 ((inca or incb)='1') then     --  PORTA OR AGGIUNTA PER FORZARE CLOCK GATING
            if ((tmpa)>(tmpb)) then
                C <= tmpa;
            else
                C <= tmpb;
            end if;
        end if;
      end if;
    end process;
end behavioral;
```

C.3 Back Annotation Process

- Sintesi con Design Compiler e generazione della netlist Verilog e del file SDF

```
analyze -library WORK -format vhdl {/home/lp19.21/Documents/lab1/fa_syn.vhd /home/lp19.21/
    Documents/lab3/saiftest/inccomp.vhd}
elaborate inccomp -library WORK -architecture behavioral
compile -exact_map

write -hierarchy -format ddc -output inccomp.ddc

ungroup -all -flatten
change_names -hierarchy -rules verilog
write_sdf ../inccomp.sdf
write -f verilog -hierarchy -output ../inccomp_netlist.v
quit
```

- Simulazione con ModelSim, annotazione dell'attività dei nodi in un file VCD

```
# Analyze the netlist and testbench
vlog -reportprogress 300 -work work /home/lp19.21/Documents/lab3/saiftest/inccomp_netlist.v
vlog -reportprogress 300 -work work /home/lp19.21/Documents/lab3/saiftest/tb_inccomp.v
# Loads the technological library and the SDF, resolution is ps
vsim -voptargs=+acc -L /software/dk/nangate45/verilog/msim6.5c -sdftyp /tbinccomp/DUT=../
    inccomp.sdf work.tbinccomp -t ps

#add wave *

# Generates the VCD file and add all the DUT signals
vcd file ../inccomp.vcd
vcd add /tbinccomp/DUT/*

# runs the simulation
run 2000ns
quit
```

- Conversione del file VCD in un file SAIF

```
vcd2saif -64 -input ../inccomp.vcd -output ../backward.saif
```

- Importazione del file SAIF in Design Compiler e report di potenza

```
read_ddc inccomp.ddc

create_clock -name "clock" -period 5 {ck}

# Reads data from backward.saif file...
# THE CLOCK IN THE TESTBENCH MUST BE THE SAME AS THE CLOCK 'CK'
# DECLARED UNDER DESIGN VISION!!
read_saif -input ../backward.saif -instance tbinccomp/DUT -unit ns -scale 1
report_power -include_input_nets > ./reports/report_power_saiftest_input_nets.txt
report_power -net -include_input_nets > ./reports/
    report_power_saiftest_input_nets_only_nets.txt
quit
```

Appendice D

Lab 4

D.1 create_sdf.scr

```
source definitions.scr

# busnormal
variable targetcompilation "../busnormal.vhd"
variable top_hierarchy "busnormal"
analyze -format vhdl $targetcompilation
elaborate -library work $top_hierarchy
compile -exact_map
write -hierarchy -format ddc -output $top_hierarchy.ddc
ungroup -all -flatten
change_names -hierarchy -rules verilog
# write_sdf ../$top_hierarchy.sdf
write_sdf ../sim/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../fromsynopsys$normtag.v

# businvert
variable targetcompilation "../businvbeh.vhd"
variable top_hierarchy "businvbeh"
analyze -format vhdl $targetcompilation
elaborate -library work $top_hierarchy
compile -exact_map
write -hierarchy -format ddc -output $top_hierarchy.ddc
ungroup -all -flatten
change_names -hierarchy -rules verilog
# write_sdf ../$top_hierarchy.sdf
write_sdf ../sim/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../fromsynopsys$invtag.v

# transbased
variable targetcompilation "../transbased.vhd"
variable top_hierarchy "transbased"
analyze -format vhdl $targetcompilation
elaborate -library work $top_hierarchy
compile -exact_map
write -hierarchy -format ddc -output $top_hierarchy.ddc
ungroup -all -flatten
change_names -hierarchy -rules verilog
# write_sdf ../$top_hierarchy.sdf
write_sdf ../sim/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../fromsynopsys$trantag.v

# grayencoder
variable targetcompilation "../grayencoder.vhd"
variable top_hierarchy "grayencoder"
analyze -format vhdl $targetcompilation
elaborate -library work $top_hierarchy
compile -exact_map
write -hierarchy -format ddc -output $top_hierarchy.ddc
ungroup -all -flatten
change_names -hierarchy -rules verilog
# write_sdf ../$top_hierarchy.sdf
write_sdf ../sim/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../fromsynopsys$graytag.v

# tzero
variable targetcompilation "../t0encdec.vhd"
variable top_hierarchy "t0encdec"
analyze -format vhdl $targetcompilation
elaborate -library work $top_hierarchy
compile -exact_map
```

```
write -hierarchy -format ddc -output $top_hierarchy.ddc
ungroup -all -flatten
change_names -hierarchy -rules verilog
# write_sdf ../$top_hierarchy.sdf
write_sdf ../sim/$top_hierarchy.sdf
write -f verilog -hierarchy -output ../fromsynopsys$tzerotag.v

quit
```

D.2 t0encdec.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity t0encdec is
port (  ck : in std_logic;
        rst : in std_logic;
        A : in std_logic_vector(7 downto 0);
        B : buffer std_logic_vector(8 downto 0);
        C : buffer std_logic_vector(7 downto 0));
end t0encdec;

architecture behavioral of t0encdec is

signal A_cod, A_notcod, C_inc, mux : std_logic_vector(7 downto 0);
signal inc, comp : std_logic;

begin

    enc: process(ck,rst)
    begin
        if rst='1' then
            A_cod <= (others => '0');
            A_notcod <= (others => '0');
            inc <= '0';
        elsif ck'event and ck='1' then
            A_notcod <= A;
            inc <= comp;
            if comp = '0' then
                A_cod <= A;
            end if;
        end if;
    end process;

    comp <= '1' when A = (A_notcod + 1) else '0';
    B <= inc & A_cod;

    dec: process(ck,rst)
    begin
        if rst='1' then
            C <= (others => '0');
        elsif ck'event and ck='1' then
            C <= mux;
        end if;
    end process;

    C_inc <= C + 1;
    mux <= B(7 downto 0) when B(8) = '0' else C_inc;

end behavioral;
```

D.3 master_script.sh

```
DATATYPE=ADDRESS
#DATATYPE=DATA

## Synopsys
cd syn
setsynopsys
rm -r work
mkdir work
```

```
design_vision -f create_sdf.scr

## ModelSim
cd ../sim
rm -r work
setmentor
vlib work
vsim -do ../fill_forward.scr

## Conversion
cd ../syn
vcd2saif -64 -input ../sim/frommentor_$DATATYPE.vcd -output ../backward_$DATATYPE.saif

## Synopsys
setsynopsys
design_vision -f backward_all.scr

cd ..
```

Appendice E

Lab 6

E.1 inccomp_tb_vref.vhd

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

entity inccomp_tb is
end entity ; -- inccomp_tb

architecture arch of inccomp_tb is

  component INCCOMP 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 INCCOMP;

  constant Period: time := 1 ns; -- Clock period (1 GHz)
  signal CK_TB    : std_logic :='0';
  signal RST_TB   : std_logic;
  signal INCA_TB  : std_logic;
  signal INCB_TB  : std_logic;
  signal EXTA_TB  : std_logic;
  signal EXTB_TB  : std_logic;
  signal A_TB     : signed (31 downto 0);
  signal B_TB     : signed (31 downto 0);
  signal C_TB     : signed (31 downto 0);

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
      end if;
      return '1';
  end function;

begin

  DUT : INCCOMP
  generic map (
               N_BIT => 32
              )
  port map    (
               CK    => CK_TB,
               RST   => RST_TB,
               INCA  => INCA_TB,
               INCB  => INCB_TB,
               EXTA  => EXTA_TB,
               EXTB  => EXTB_TB,
```

```
                A      => A_TB,
                B      => B_TB,
                C      => C_TB
              );

  CK_TB <= not CK_TB after Period/2;
  RST_TB <= '1', '0' after Period;
  EXTA_TB   <= '0', '1' after Period*5, '0' after Period*10, '1' after Period*55, '0' after Period
      *60;
  EXTB_TB   <= '0', '1' after Period*5, '0' after Period*10, '1' after Period*55, '0' after Period
      *60;

random: process
  variable seed1, seed2: positive;

  variable rand_1: real;
  variable rand_2: real;
  variable rand_A: real;
  variable rand_B: real;

  variable int_rand_1: integer;
  variable int_rand_2: integer;
  variable int_rand_A: integer;
  variable int_rand_B: integer;

  variable sum: signed(31 downto 0);

begin
  INCA_TB   <= '0';
  INCB_TB   <= '0';
  --EXTA_TB    <= '0';
  --EXTB_TB    <= '0';
  A_TB    <= to_signed(0, A_TB'LENGTH);
  B_TB    <= to_signed(0, B_TB'LENGTH);
  wait for 5 ns;


  for I in 1 to 50 loop
    -- Random number generation
    UNIFORM(seed1, seed2, rand_1);
    UNIFORM(seed1, seed2, rand_2);
    UNIFORM(seed1, seed2, rand_A);
    UNIFORM(seed1, seed2, rand_B);

    int_rand_1 := INTEGER(TRUNC(rand_1*4294967296.0 - 2147483649.0));
    int_rand_2 := INTEGER(TRUNC(rand_2*4294967296.0 - 2147483649.0));
    int_rand_A := INTEGER(rand_A);
    int_rand_B := INTEGER(rand_B);

    A_TB    <= to_signed(int_rand_1, A_TB'LENGTH);
    B_TB    <= to_signed(int_rand_2, B_TB'LENGTH);
    INCA_TB <= to_std_logic(int_rand_A);
    INCB_TB <= to_std_logic(int_rand_B);

    wait for 5 ns;
  end loop;

  wait;

end process;

end architecture ; -- arch
```

E.2 script.sh

```
rm -r -f work
setmentor
vlib work

#vsim -c -do inccomp_tb.do
vsim -do inccomp_tb.do
```

E.3 inccomp_tb.do

```
vcom -93 -work ./work ./src/utils/constants.vhd
vcom -93 -work ./work ./src/utils/andgate2.vhd
vcom -93 -work ./work ./src/utils/orgate2.vhd
vcom -93 -work ./work ./src/utils/notgate.vhd
vcom -93 -work ./work ./src/utils/fa.vhd
vcom -93 -work ./work ./src/utils/rca_gen.vhd
vcom -93 -work ./work ./src/utils/mux21_generic.vhd
vcom -93 -work ./work ./src/utils/RegisterN.vhd
vcom -93 -work ./work ./src/utils/comparator.vhd
vcom -93 -work ./work ./src/P4_1/g_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_network_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_network.vhd
vcom -93 -work ./work ./src/P4_1/carry_select_block.vhd
vcom -93 -work ./work ./src/P4_1/carry_logic_network.vhd
vcom -93 -work ./work ./src/P4_1/carry_generator.vhd
vcom -93 -work ./work ./src/P4_1/sum_generator.vhd
vcom -93 -work ./work ./src/P4_1/p4_adder_1.vhd
vcom -93 -work ./work ./src/P4_2/p4_adder_2.vhd
vcom -93 -work ./work ./src/inccomp.vhd
vcom -93 -work ./work ./inccomp_tb.vhd

vsim -t 100ps -novopt work.inccomp_tb

set NumericStdNoWarnings 1
run 0 ps
set NumericStdNoWarnings 0

add wave -radix binary    -color GREEN       sim:/inccomp_tb/CK_TB
add wave -radix binary    -color GREEN       sim:/inccomp_tb/RST_TB
add wave -radix binary    -color GREEN       sim:/inccomp_tb/INCA_TB
add wave -radix binary    -color GREEN       sim:/inccomp_tb/INCB_TB
add wave -radix binary    -color GREEN       sim:/inccomp_tb/EXTA_TB
add wave -radix binary    -color GREEN       sim:/inccomp_tb/EXTB_TB
add wave -radix decimal    -color GREEN       sim:/inccomp_tb/A_TB
add wave -radix decimal    -color GREEN       sim:/inccomp_tb/B_TB

add wave -radix decimal    -color RED         sim:/inccomp_tb/C_TB

run 300 ns
```

E.4 rename.sh

```
#!/bin/bash

entity_substitution() {      #$1=$file_in , #2=a|b
    n=`grep -i "^ENTITY " $1 | cut -d " " -f 2`
    if [[ ! -z "$n" ]]; then     # if not empty
        echo "- Entity: $n"
        sed -i "s/ $n / $n\_$2 /g" $1
        sed -i "s/ $n;/ $n\_$2;/g" $1
    fi
}

component_substitution() {  #$1=$file_in , #2=a|b
    while read line; do
        if [[ ${line^^} =~ "END COMPONENT" ]]; then
            p=`echo $line | cut -d " " -f 3 | cut -d ";" -f 1`
            sed -i "s/ $p;/ $p\_$2;/g" $1
        elif [[ ${line^^} =~ "COMPONENT " ]]; then
            p=`echo $line | cut -d " " -f 2`
            sed -i "s/COMPONENT $p/COMPONENT $p\_$2/g" $1
            sed -i "s/: $p/: $p\_$2/g" $1
            sed -i "s/ $p / $p\_$2 /g" $1
        fi
    done < $1
}

rm -r src
version=$(pwd | cut -d "/" -f 7)
echo "Version: $version"
cp -r /home/repository/lowpower/ese6/2/"$version"/src .

cd src
cp -r utils utils_2
mv utils utils_1

for file_out in $(ls); do
```

```
    if [[ -d $file_out ]]; then
        cd $file_out
        echo "Folder: $file_out"
        for file_in in $(ls); do
            if [[ -f $file_in ]]; then
                location=$(pwd)
                if [[ $location =~ "P4_1" ]]; then
                    entity_substitution "$file_in" "a"
                    component_substitution "$file_in" "a"
                elif [[ $location =~ "P4_2" ]]; then
                    entity_substitution "$file_in" "b"
                    component_substitution "$file_in" "b"
                elif [[ $location =~ "utils_1" ]]; then
                    entity_substitution "$file_in" "a"
                    component_substitution "$file_in" "a"
                elif [[ $location =~ "utils_2" ]]; then
                    entity_substitution "$file_in" "b"
                    component_substitution "$file_in" "b"
                fi
            fi
        done
        cd ..
    elif [[ -f $file_out ]]; then
        sed -i "s/p4_adder_1/p4_adder_1_a/g" $file_out
        sed -i "s/p4_adder_2/p4_adder_2_b/g" $file_out
    fi
done

cd ..
```

E.5 inccomp_cross_tb.do

```
vcom -93 -work ./work ../vREF/src/utils/constants.vhd

vcom -93 -work ./work ./src/utils_1/andgate2.vhd
vcom -93 -work ./work ./src/utils_1/orgate2.vhd
vcom -93 -work ./work ./src/utils_1/notgate.vhd
vcom -93 -work ./work ./src/utils_1/fa.vhd
vcom -93 -work ./work ./src/utils_1/rca_gen.vhd
vcom -93 -work ./work ./src/utils_1/mux21_generic.vhd
vcom -93 -work ./work ./src/utils_1/RegisterN.vhd
vcom -93 -work ./work ./src/utils_1/comparator.vhd
vcom -93 -work ./work ./src/P4_1/g_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_network_block.vhd
vcom -93 -work ./work ./src/P4_1/pg_network.vhd
vcom -93 -work ./work ./src/P4_1/carry_select_block.vhd
vcom -93 -work ./work ./src/P4_1/carry_logic_network.vhd
vcom -93 -work ./work ./src/P4_1/carry_generator.vhd
vcom -93 -work ./work ./src/P4_1/sum_generator.vhd
vcom -93 -work ./work ./src/P4_1/p4_adder_1.vhd

vcom -93 -work ./work ./src/utils_2/andgate2.vhd
vcom -93 -work ./work ./src/utils_2/orgate2.vhd
vcom -93 -work ./work ./src/utils_2/notgate.vhd
vcom -93 -work ./work ./src/utils_2/fa.vhd
vcom -93 -work ./work ./src/utils_2/rca_gen.vhd
vcom -93 -work ./work ./src/utils_2/mux21_generic.vhd
vcom -93 -work ./work ./src/utils_2/RegisterN.vhd
vcom -93 -work ./work ./src/utils_2/comparator.vhd
vcom -93 -work ./work ./src/P4_2/g_block.vhd
vcom -93 -work ./work ./src/P4_2/pg_block.vhd
vcom -93 -work ./work ./src/P4_2/pg_network_block.vhd
vcom -93 -work ./work ./src/P4_2/pg_network.vhd
vcom -93 -work ./work ./src/P4_2/carry_select_block.vhd
vcom -93 -work ./work ./src/P4_2/carry_logic_network.vhd
vcom -93 -work ./work ./src/P4_2/carry_generator.vhd
vcom -93 -work ./work ./src/P4_2/sum_generator.vhd
vcom -93 -work ./work ./src/P4_2/p4_adder_2.vhd

vcom -93 -work ./work ../vREF/src/utils/andgate2.vhd
vcom -93 -work ./work ../vREF/src/utils/orgate2.vhd
vcom -93 -work ./work ../vREF/src/utils/notgate.vhd
vcom -93 -work ./work ../vREF/src/utils/fa.vhd
vcom -93 -work ./work ../vREF/src/utils/rca_gen.vhd
vcom -93 -work ./work ../vREF/src/utils/mux21_generic.vhd
vcom -93 -work ./work ../vREF/src/utils/RegisterN.vhd
vcom -93 -work ./work ../vREF/src/utils/comparator.vhd
vcom -93 -work ./work ../vREF/src/P4_1/g_block.vhd
```

```
vcom -93 -work ./work ../vREF/src/P4_1/pg_block.vhd
vcom -93 -work ./work ../vREF/src/P4_1/pg_network_block.vhd
vcom -93 -work ./work ../vREF/src/P4_1/pg_network.vhd
vcom -93 -work ./work ../vREF/src/P4_1/carry_select_block.vhd
vcom -93 -work ./work ../vREF/src/P4_1/carry_logic_network.vhd
vcom -93 -work ./work ../vREF/src/P4_1/carry_generator.vhd
vcom -93 -work ./work ../vREF/src/P4_1/sum_generator.vhd
vcom -93 -work ./work ../vREF/src/P4_1/p4_adder_1.vhd
vcom -93 -work ./work ../vREF/src/P4_2/p4_adder_2.vhd

vcom -93 -work ./work ../vREF/src/inccomp_ref.vhd
vcom -93 -work ./work ./src/inccomp.vhd
vcom -93 -work ./work ./inccomp_tb.vhd

vsim -t 100ps -novopt work.inccomp_tb

set NumericStdNoWarnings 1
run 0 ps
set NumericStdNoWarnings 0

add wave -radix binary   -color GREEN      sim:/inccomp_tb/CK_TB
add wave -radix binary   -color GREEN      sim:/inccomp_tb/RST_TB
add wave -radix binary   -color GREEN      sim:/inccomp_tb/INCA_TB
add wave -radix binary   -color GREEN      sim:/inccomp_tb/INCB_TB
add wave -radix binary   -color GREEN      sim:/inccomp_tb/EXTA_TB
add wave -radix binary   -color GREEN      sim:/inccomp_tb/EXTB_TB
add wave -radix decimal   -color GREEN      sim:/inccomp_tb/A_TB
add wave -radix decimal   -color GREEN      sim:/inccomp_tb/B_TB

add wave -radix decimal   -color RED        sim:/inccomp_tb/C_TB_REF
add wave -radix decimal   -color RED        sim:/inccomp_tb/C_TB_BUG

run 300 ns
```

E.6 inccomp_v1_tb.vhd

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

entity inccomp_tb is
end entity ; -- inccomp_tb

architecture arch of inccomp_tb is

  component INCCOMP_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 INCCOMP_REF;

  component INCCOMP 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 INCCOMP;

  constant Period: time := 1 ns; -- Clock period (1 GHz)
  signal CK_TB     : std_logic :='0';
  signal RST_TB    : std_logic;
  signal INCA_TB   : std_logic;
  signal INCB_TB   : std_logic;
  signal EXTA_TB   : std_logic;
  signal EXTB_TB   : std_logic;
  signal A_TB      : signed (31 downto 0);
  signal B_TB      : signed (31 downto 0);

  signal C_TB_REF      : signed (31 downto 0);
  signal C_TB_BUG      : signed (31 downto 0);

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
      end if;
      return '1';
  end function;


begin

  -- Reference architecture
  DUT_REF : INCCOMP_REF
  generic map (
                N_BIT => 32
              )
  port map     (
                CK     => CK_TB,
                RST    => RST_TB,
                INCA   => INCA_TB,
                INCB   => INCB_TB,
                EXTA   => EXTA_TB,
                EXTB   => EXTB_TB,
                A      => A_TB,
                B      => B_TB,
                C      => C_TB_REF
              );

  -- Architecture under test
  DUT_BUG : INCCOMP
  generic map (
                N_BIT => 32
              )
  port map     (
                CK     => CK_TB,
                RST    => RST_TB,
                INCA   => INCA_TB,
                INCB   => INCB_TB,
                EXTA   => EXTA_TB,
                EXTB   => EXTB_TB,
                A      => A_TB,
                B      => B_TB,
                C      => C_TB_BUG
              );

  CK_TB <= not CK_TB after Period/2;
  RST_TB <= '1', '0' after Period;
  EXTA_TB   <= '0', '1' after Period*5, '0' after Period*10, '1' after Period*55, '0' after Period
      *60;
  EXTB_TB   <= '0', '1' after Period*5, '0' after Period*10, '1' after Period*55, '0' after Period
      *60;

random: process
  variable seed1, seed2: positive;

  variable rand_1: real;
  variable rand_2: real;
  variable rand_A: real;
  variable rand_B: real;

  variable int_rand_1: integer;
  variable int_rand_2: integer;
  variable int_rand_A: integer;
  variable int_rand_B: integer;

begin
  INCA_TB   <= '0';
  INCB_TB   <= '0';
  --EXTA_TB    <= '0';
```

```
  --EXTB_TB    <= '0';
  A_TB     <= to_signed(0, A_TB'LENGTH);
  B_TB     <= to_signed(0, B_TB'LENGTH);
  wait for 5 ns;


  for I in 1 to 50 loop
    -- Random number generation
    UNIFORM(seed1, seed2, rand_1);
    UNIFORM(seed1, seed2, rand_2);
    UNIFORM(seed1, seed2, rand_A);
    UNIFORM(seed1, seed2, rand_B);

    int_rand_1 := INTEGER(TRUNC(rand_1*4294967296.0 - 2147483649.0));
    int_rand_2 := INTEGER(TRUNC(rand_2*4294967296.0 - 2147483649.0));
    int_rand_A := INTEGER(rand_A);
    int_rand_B := INTEGER(rand_B);

    A_TB    <= to_signed(int_rand_1, A_TB'LENGTH);
    B_TB    <= to_signed(int_rand_2, B_TB'LENGTH);
    INCA_TB <= to_std_logic(int_rand_A);
    INCB_TB <= to_std_logic(int_rand_B);

    -- Assert
    assert (C_TB_BUG = C_TB_REF) report "There is a bug." severity warning;
    wait for 5 ns;
  end loop;

  wait;

end process;

end architecture ; -- arch
```

E.7 inccomp_v2_tb.vhd

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

entity inccomp_tb is
end entity ; -- inccomp_tb

architecture arch of inccomp_tb is

  component INCCOMP_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 INCCOMP_REF;

  component INCCOMP 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 INCCOMP;

  constant Period: time := 1 ns; -- Clock period (1 GHz)
  signal CK_TB    : std_logic :='0';
  signal RST_TB   : std_logic;
  signal INCA_TB  : std_logic;
  signal INCB_TB  : std_logic;
  signal EXTA_TB  : std_logic;
  signal EXTB_TB  : std_logic;
  signal A_TB     : signed (31 downto 0);
  signal B_TB     : signed (31 downto 0);

  signal C_TB_REF     : signed (31 downto 0);
  signal C_TB_BUG     : signed (31 downto 0);

begin

  -- Reference architecture
  DUT_REF : INCCOMP_REF
  generic map (
                N_BIT => 32
              )
  port map    (
                CK    => CK_TB,
                RST   => RST_TB,
                INCA  => INCA_TB,
                INCB  => INCB_TB,
                EXTA  => EXTA_TB,
                EXTB  => EXTB_TB,
                A     => A_TB,
                B     => B_TB,
                C     => C_TB_REF
              );

  -- Architecture under test
  DUT_BUG : INCCOMP
  generic map (
                N_BIT => 32
              )
  port map    (
                CK    => CK_TB,
                RST   => RST_TB,
                INCA  => INCA_TB,
                INCB  => INCB_TB,
                EXTA  => EXTA_TB,
                EXTB  => EXTB_TB,
                A     => A_TB,
                B     => B_TB,
                C     => C_TB_BUG
              );

  CK_TB <= not CK_TB after Period/2;
  RST_TB <= '1', '0' after Period;

  assert (C_TB_BUG = C_TB_REF) report "There is a bug." severity warning;

begin
  INCA_TB   <= '0';
  INCB_TB   <= '0';
  EXTA_TB   <= '0';
  EXTB_TB   <= '0';
  A_TB    <= to_signed(0, A_TB'LENGTH);
  B_TB    <= to_signed(0, B_TB'LENGTH);
  wait for 5 ns;

  INCA_TB   <= '1';
  A_TB    <= "00000000000000000000000000000000";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000000000100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000000001100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000000011100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000000111100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000001111100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000011111100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000000111111100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
  A_TB    <= "00000000000000000000001111111100";
  EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
```

```
A_TB    <= "00000000000000000000001111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000000000011111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000000000111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000000001111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000000011111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000000111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000001111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000011111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000000111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000001111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000011111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000000111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000001111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000000011111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000001111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000011111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00000111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00001111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00011111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "00111111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "01111111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
A_TB    <= "11111111111111111111111111111100";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 20 ns;
INCA_TB   <= '0';

A_TB    <= "00000000000000000000000000000000";
EXTA_TB   <= '1'; wait for 2 ns; EXTA_TB   <= '0'; wait for 200 ns;

INCB_TB   <= '1';
B_TB    <= "00000000000000000000000000000000";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000000000100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000000001100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000000011100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000000111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000001111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000011111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000000111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000001111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000011111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000000111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000001111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000011111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000000111111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000001111111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000011111111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
B_TB    <= "00000000000000111111111111111100";
EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
```

```
  B_TB    <= "00000000000001111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000000000011111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000000000111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000000001111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000000011111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000000111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000001111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000011111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00000111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00001111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00011111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "00111111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "01111111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  B_TB    <= "11111111111111111111111111111100";
  EXTB_TB   <= '1'; wait for 2 ns; EXTB_TB   <= '0'; wait for 20 ns;
  INCB_TB   <= '0';

  wait;

end process;

end architecture ; -- arch
```

E.8 p4_tb.vhd

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

entity p4_tb is
end entity ; -- p4_tb

architecture arch of p4_tb is

  component p4_adder_ref is
    generic (
              N_BIT : integer;
              CARRY   : integer
            );
    port    (
              A     : in  signed (N_BIT-1 downto 0);
              B     : in  signed (N_BIT-1 downto 0);
              C_IN_0: in  std_logic;
              C_OUT : out std_logic;
              SUM   : out signed (N_BIT-1 downto 0)
          );
  end component p4_adder_ref;

  component p4_adder_1_a is
    generic (
              N_BIT : integer;
              CARRY   : integer
            );
    port    (
              A     : in  signed (N_BIT-1 downto 0);
              B     : in  signed (N_BIT-1 downto 0);
              C_IN_0: in  std_logic;
              C_OUT : out std_logic;
              SUM   : out signed (N_BIT-1 downto 0)
          );
  end component p4_adder_1_a;

  signal A_TB       : signed (31 downto 0);
  signal B_TB       : signed (31 downto 0);
```

```
  signal C_IN_0_TB  : std_logic;

  signal C_OUT_REF  : std_logic;
  signal C_OUT_BUG  : std_logic;
  signal SUM_REF    : signed (31 downto 0);
  signal SUM_BUG    : signed (31 downto 0);

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
      end if;
      return '1';
  end function;


begin

  -- Reference architecture
  DUT_REF : p4_adder_ref
  generic map (
                N_BIT => 32,
                CARRY => 4
              )
  port map    (
                A      => A_TB,
                B      => B_TB,
                C_IN_0 => C_IN_0_TB,
                C_OUT  => C_OUT_REF,
                SUM    => SUM_REF
              );

  DUT_BUG : p4_adder_1_a
  generic map (
                N_BIT => 32,
                CARRY => 4
              )
  port map    (
                A      => A_TB,
                B      => B_TB,
                C_IN_0 => C_IN_0_TB,
                C_OUT  => C_OUT_BUG,
                SUM    => SUM_BUG
              );

random: process
  variable seed1, seed2: positive;

  variable rand_1: real;
  variable rand_2: real;
  variable rand_C: real;

  variable int_rand_1: integer;
  variable int_rand_2: integer;
  variable int_rand_C: integer;

begin
  A_TB    <= to_signed(0, A_TB'LENGTH);
  B_TB    <= to_signed(0, B_TB'LENGTH);
  C_IN_0_TB   <= '0';
  wait for 5 ns;

  for I in 1 to 100000 loop
    -- Random number generation
    UNIFORM(seed1, seed2, rand_1);
    UNIFORM(seed1, seed2, rand_2);
    UNIFORM(seed1, seed2, rand_C);

    int_rand_1 := INTEGER(TRUNC(rand_1*4294967296.0 - 2147483649.0));
    int_rand_2 := INTEGER(TRUNC(rand_2*4294967296.0 - 2147483649.0));
    int_rand_C := INTEGER(rand_C);

    A_TB      <= to_signed(int_rand_1, A_TB'LENGTH);
    B_TB      <= to_signed(int_rand_2, B_TB'LENGTH);
    C_IN_0_TB <= to_std_logic(int_rand_C);

    -- Assert
    assert (SUM_BUG = SUM_REF) report "There is a bug in the sum." severity warning;
    assert (C_OUT_BUG = C_OUT_REF) report "There is a bug in the c_out." severity warning;
    wait for 1 ns;
  end loop;

  wait;
```

```
end process;

end architecture ; -- arch
```

E.9 counter_v1_tb.vhd

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;
  use ieee.math_real.all ;

entity counter_tb is
end entity ; -- counter_tb

architecture arch of counter_tb is

  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;

  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;

  constant Period: time := 1 ns; -- CloCLK period (1 GHz)
  signal CLK_TB   : std_logic :='0';
  signal RST_TB   : std_logic;
  signal COUNT_TB : std_logic;

  signal DATA_OUT_TB_REF : unsigned (8-1 downto 0);
  signal DATA_OUT_TB_BUG : unsigned (8-1 downto 0);

  signal UP_DN_TB_REF : std_logic;
  signal UP_DN_TB_BUG : std_logic;

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
      end if;
      return '1';
  end function;


begin

  -- Reference architecture
  DUT_REF : counter_ref
  port map    (
                CLK       => CLK_TB,
                RST       => RST_TB,
                COUNT     => COUNT_TB,
                DATA_OUT  => DATA_OUT_TB_REF,
                UP_DN     => UP_DN_TB_REF
              );

  DUT_BUG : counter
  port map    (
                CLK       => CLK_TB,
                RST       => RST_TB,
                COUNT     => COUNT_TB,
                DATA_OUT  => DATA_OUT_TB_BUG,
                UP_DN     => UP_DN_TB_BUG
              );

  CLK_TB <= not CLK_TB after Period/2;
  RST_TB <= '1', '0' after Period;
```

```
random: process
  variable seed1, seed2: positive;
  variable rand_count: real;
  variable int_rand_count: integer;

begin
  COUNT_TB   <= '0';

  wait for 5 ns;

    for I in 1 to 1000 loop
    -- Random number generation
    UNIFORM(seed1, seed2, rand_count);

    int_rand_count := INTEGER(rand_count);

    COUNT_TB <= to_std_logic(int_rand_count);

    -- Assert
    assert (DATA_OUT_TB_BUG = DATA_OUT_TB_REF) report "There is a bug in DATA_OUT." severity warning
        ;
    assert (UP_DN_TB_BUG = UP_DN_TB_REF) report "There is a bug in UP_DN." severity warning;
    wait for 1 ns;
  end loop;

  wait;

end process;

end architecture ; -- arch
```

E.10 counter_v2_tb.vhd

```
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 ; -- counter_tb

architecture arch of counter_tb is

  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;

  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;

  constant Period: time := 1 ns; -- CloCLK period (1 GHz)
  signal CLK_TB    : std_logic :='0';
  signal RST_TB    : std_logic;
  signal COUNT_TB : std_logic;

  signal DATA_OUT_TB_REF : unsigned (8-1 downto 0);
  signal DATA_OUT_TB_BUG : unsigned (8-1 downto 0);

  signal UP_DN_TB_REF : std_logic;
  signal UP_DN_TB_BUG : std_logic;

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
```

```
      end if;
      return '1';
  end function;


begin

  -- Reference architecture
  DUT_REF : counter_ref
  port map    (
                CLK         => CLK_TB,
                RST         => RST_TB,
                COUNT       => COUNT_TB,
                DATA_OUT    => DATA_OUT_TB_REF,
                UP_DN       => UP_DN_TB_REF
              );

  DUT_BUG : counter
  port map    (
                CLK         => CLK_TB,
                RST         => RST_TB,
                COUNT       => COUNT_TB,
                DATA_OUT    => DATA_OUT_TB_BUG,
                UP_DN       => UP_DN_TB_BUG
              );

  CLK_TB <= not CLK_TB after Period/2;
  --RST_TB <= '1', '0' after Period;

  -- Assert
  assert (DATA_OUT_TB_BUG = DATA_OUT_TB_REF) report "There is a bug in DATA_OUT." severity warning;
  assert (UP_DN_TB_BUG = UP_DN_TB_REF) report "There is a bug in UP_DN." severity warning;

begin
  COUNT_TB  <= '0';
  RST_TB    <= '1';

  wait for 1 ns;

  RST_TB <= '0';
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 1;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 2;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 3;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 4;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 5;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 6;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until DATA_OUT_TB_BUG = 7;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 1;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
```

```
  wait until DATA_OUT_TB_BUG = 2;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 3;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 4;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 5;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 6;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 7;
  RST_TB <= '1';
  wait for 1 ns;
  RST_TB <= '0';

  wait;

end process;

end architecture ; -- arch
```

E.11 counter_v3_tb.vhd

```
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 ; -- counter_tb

architecture arch of counter_tb is

  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;

  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;

  constant Period: time := 1 ns; -- CloCLK period (1 GHz)
  signal CLK_TB    : std_logic :='0';
  signal RST_TB    : std_logic;
  signal COUNT_TB : std_logic;

  signal DATA_OUT_TB_REF : unsigned (8-1 downto 0);
```

```
  signal DATA_OUT_TB_BUG : unsigned (8-1 downto 0);

  signal UP_DN_TB_REF : std_logic;
  signal UP_DN_TB_BUG : std_logic;

  function to_std_logic(i : in integer) return std_logic is
  begin
      if i = 0 then
         return '0';
      end if;
      return '1';
  end function;


begin

  -- Reference architecture
  DUT_REF : counter_ref
  port map    (
                CLK        => CLK_TB,
                RST        => RST_TB,
                COUNT      => COUNT_TB,
                DATA_OUT   => DATA_OUT_TB_REF,
                UP_DN      => UP_DN_TB_REF
              );

  DUT_BUG : counter
  port map    (
                CLK        => CLK_TB,
                RST        => RST_TB,
                COUNT      => COUNT_TB,
                DATA_OUT   => DATA_OUT_TB_BUG,
                UP_DN      => UP_DN_TB_BUG
              );

  CLK_TB <= not CLK_TB after Period/2;
  --RST_TB <= '1', '0' after Period;

  -- Assert
  assert (DATA_OUT_TB_BUG = DATA_OUT_TB_REF) report "There is a bug in DATA_OUT." severity warning;
  assert (UP_DN_TB_BUG = UP_DN_TB_REF) report "There is a bug in UP_DN." severity warning;

timer_gen: process
  variable t : natural range 0 to 100 := 0;

begin
  COUNT_TB  <= '0';
  RST_TB    <= '1';

  wait for 1 ns;

  RST_TB <= '0';
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 1;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 2;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 3;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 4;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 5;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until DATA_OUT_TB_BUG = 6;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';
```

```
  wait until DATA_OUT_TB_BUG = 7;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 1;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 2;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 3;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 4;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 5;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 6;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait until UP_DN_TB_BUG = '0';
  wait until DATA_OUT_TB_BUG = 7;
  COUNT_TB  <= '0';
  wait for 1000 ns;
  COUNT_TB  <= '1';

  wait;

end process;

end architecture ; -- arch
```

E.12 tb_generator_rca.py

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


##############################################################################################
########################## Insert here all the input parameters ##############################

# Name of the component to be tested (beware of capital/small letters, they must match)
EntityName = "RCA"
# Name of the file with the component (beware of capital/small letters, they must match)
# The name of the file and the name of the entity of the testbench must match
FileName = "rca_tb"

# Flag used to identify if there are or not generics
GenericFlag = 1; # 1 = yes, 0 = no
# List containing the name of the generics
GenericsList = ["N_BIT"]
# List containing the type of each generic (the first one is referred to "IN1", the second one is
    referred to "IN2", ...)
GenericsType = ["Integer"]
# List containing the default value of each generic (the first one is referred to "IN1", the second
    one is referred to "IN2", ...)
GenericsDefaultValue = [32]
```

```
# List containing the assigned value of each generic (the first one is referred to "IN1", the second
    one is referred to "IN2", ...)
GenericsAssignedValue = [32]

# List containing the name of the signals
SignalsList = ["A", "B", "Ci", "S", "Co"]
# List containing the mode of each signal (the first one is referred to "IN1", the second one is
    referred to "IN2", ...)
SignalsMode = ["IN", "IN", "IN", "OUT", "OUT"]
# List containing the type of each signal (the first one is referred to "IN1", the second one is
    referred to "IN2", ...)
SignalsType = ["signed", "signed", "std_logic", "signed", "std_logic"]
# List containing the size of each signal (the first one is referred to "IN1", the second one is
    referred to "IN2", ...)
SignalsSize = ["N_BIT-1", "N_BIT-1", "1", "N_BIT", "1"]
# List containing a fixed size for the inputs, for test purpose. (substitution of
    GenericsAssignedValue inside SignalsSize)
# (the first one is referred to "IN1", the second one is referred to "IN2", ...)
# Example if GenericAssignedValue of Nbit is 16 and SignalsSize is Nbit-1 then insert 15
# Example if GenericAssignedValue of Nbit is 16 and SignalsSize is Nbit then insert 16
# For std_logic insert 1
SignalsSizeAssigned = [31, 31, 1, 31, 1]

# Number of simulation steps
NumberOfSteps = 10000
# Delay between steps
DelayBetweenSteps = "1 ns"
##########################################################################################


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

# Creation of file name
TestbenchFileName = FileName + ".vhd"

# 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\n")
##########################################


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

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


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

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


################################################################################################
################################### 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 ####################################

# Starting process
TestbenchFile.write("TEST: process \n\n")
TestbenchFile.write("begin \n\n")

# Loop for all simulation steps
for i in range(0,NumberOfSteps):
    # 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":
                RandomInputInteger = randint(1,2**SignalsSizeAssigned[i]-1)
                RandomInputBinary = ('{0:0{l}b}'.format((RandomInputInteger),l=SignalsSizeAssigned[i
                    ]))
                TestbenchFile.write("    " + SignalsList[i] + "_i <= \'" + RandomInputBinary + "\';\n
                    ")
            # Check if not single bit
            else:
                RandomInputInteger = randint(1,2**SignalsSizeAssigned[i]-1)
                RandomInputBinary = ('{0:0{l}b}'.format((RandomInputInteger),l=SignalsSizeAssigned[i
                    ]+1))
                TestbenchFile.write("    " + SignalsList[i] + "_i <= \"" + RandomInputBinary + "\";\n
                    ")
    TestbenchFile.write("    " + "wait for " + DelayBetweenSteps + ";\n\n")

TestbenchFile.write("wait; \n\n")
TestbenchFile.write("end process; \n\n")
################################################################################################


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

TestbenchFile.write("end test;")

# Closing testbench file
TestbenchFile.close()
```

E.13 rca_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rca_tb is
end rca_tb;

architecture test of rca_tb is

component RCA
```

```
generic (
        N_BIT: Integer:= 32);
port (
      A: IN signed(N_BIT-1 downto 0);
      B: IN signed(N_BIT-1 downto 0);
      Ci: IN std_logic;
      S: OUT signed(N_BIT downto 0);
      Co: OUT std_logic);
end component;

signal A_i: signed(31 downto 0) := (others => '0');
signal B_i: signed(31 downto 0) := (others => '0');
signal Ci_i: std_logic:= '0';
signal S_i: signed(31 downto 0) := (others => '0');
signal Co_i: std_logic:= '0';

begin

DUT:RCA
generic map (
            N_BIT=> 32)
port map (
          A => A_i,
          B => B_i,
          Ci => Ci_i,
          S => S_i,
          Co => Co_i);

TEST: process

begin

   A_i <= "01011111100100000100010010100000";
   B_i <= "00101011111110101100000000011111";
   Ci_i <= '1';
   wait for 1 ns;

   A_i <= "00110100111110111111000010111011";
   B_i <= "01111110101100011111101100000100";
   Ci_i <= '1';
   wait for 1 ns;

   [...]

   A_i <= "00001110101100101000100110000010";
   B_i <= "01000101111110100010110000100010";
   Ci_i <= '1';
   wait for 1 ns;

wait;

end process;

end test;
```