Modello behavioral dei ritardi di un moltiplicatore di Wallace

Di Matteo Brina

Linguaggi di Descrizione dell'Hardware
Prof. Michele Favalli
A.A. 2020-2021

Lo scopo del progetto è quello di costruire, mediante il linguaggio VHDL, un modello strutturale ed uno comportamentale di un moltiplicatore di Wallace a 4 bit. Il modello strutturale avrà ritardi reali, al modello comportamentale verranno assegnati i ritardi derivanti dalla STA in modo che l'uscita si porti prima a 'X' con ritardo EAT e poi al suo valore finale con ritardo LST. A questo punto, simulando in parallelo i due modelli, si dovrebbe osservare che le uscite del modello strutturale commutano nell'intervallo di tempo in cui le corrispondenti uscite del modello comportamentale hanno valore 'X' e che, quindi, i ritardi reali siano compresi tra EAT e LST calcolati mediante STA.

Per prima cosa, partendo dalla descrizione strutturale in Fig.1, è stato necessario costruire lo schema circuitale equivalente che andasse a mostrare i collegamenti tra le singole porte logiche.

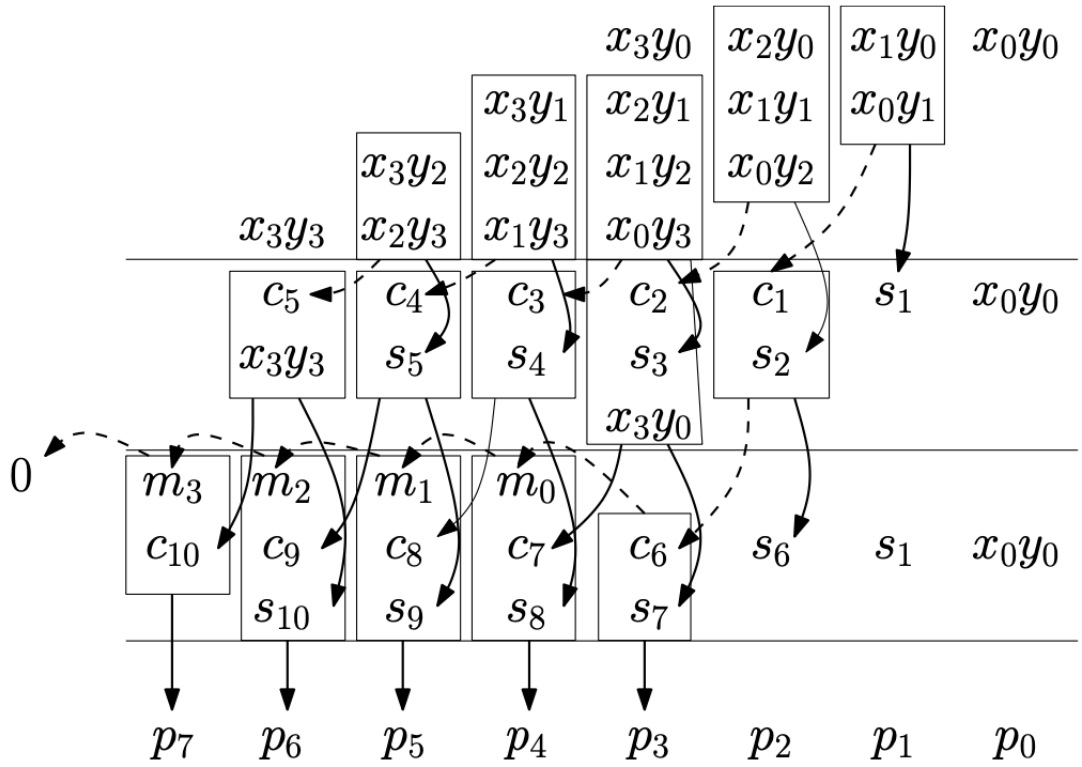

Fig.1: Nella figura le scatole rappresentano FA e HA (il contenuto sono gli ingressi) le uscite sono denotate dalle frecce quelle continue sono i bit di somma e quelle tratteggiate i riporti.

Dalla Fig.1 è possibile notare come il moltiplicatore calcola i prodotti parziali, li organizza per pesi e poi applica una serie di passi di riduzione. A ogni passo usa un FA o un HA per accorpare 3 o 2 bit del prodotto parziale con lo stesso peso. I bit non accorpati passano allo stadio successivo con l'uscita di somma del FA e il riporto (che viene promosso come peso). Alla fine i riporti vengono propagati in orizzontale (bit M nella Fig.1).

Per realizzare l'HA e il FA sono state usate le architetture mostrate in Fig.2 e in Fig.3.

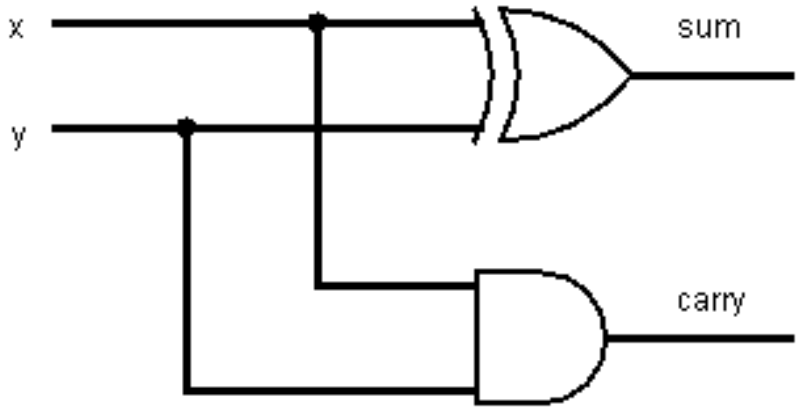

Fig.2: Architettura Half-adder

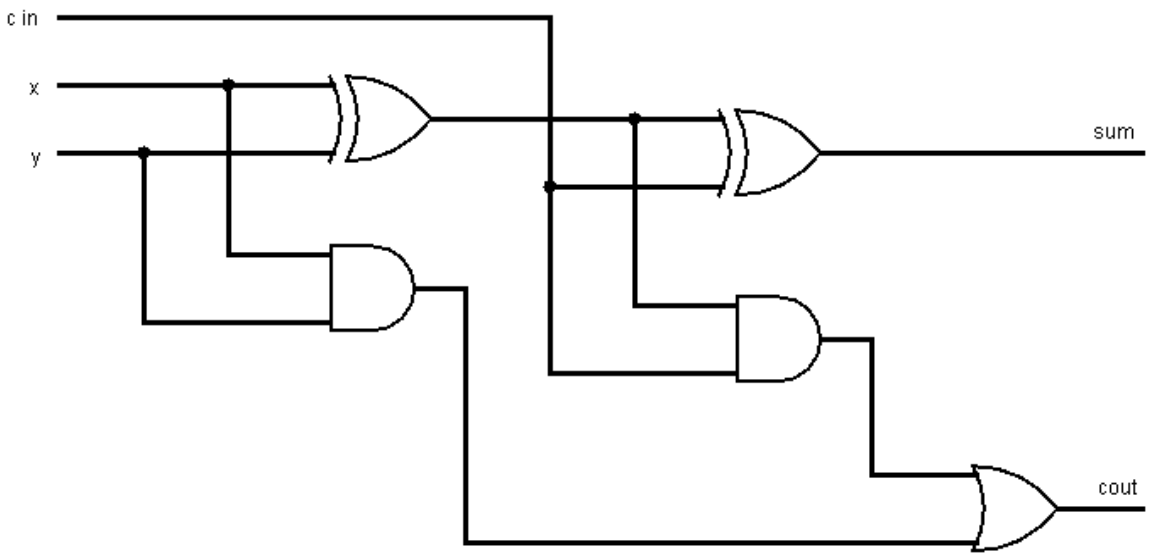

Fig.3: Architettura Full-adder

Lo schema circuitale che deriva dalla combinazione di HA e FA secondo lo schema in Fig.1 è quello mostrato in Fig.4.

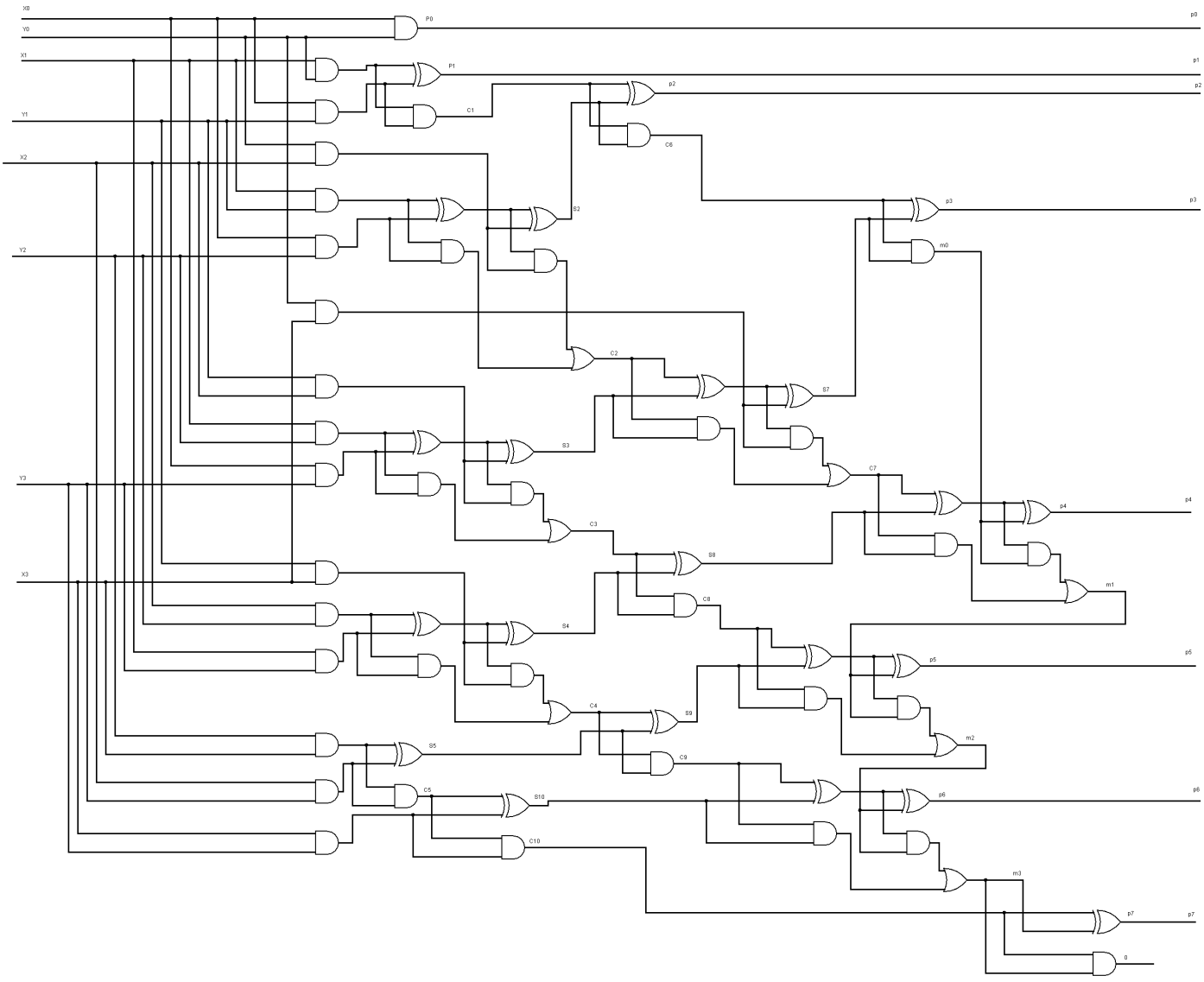

Fig.4: Schema circuitale del moltiplicatore di Wallace.

Successivamente è stato necessario applicare la STA per trovare EAT e LST associati a ogni uscita del circuito. I ritardi dei gate sono stati scelti in modo da essere conformi alle attuali tecnologie, in particolare:

AND	50 ps
OR	60 ps
XOR	80 ps

Per svolgere la STA è stato applicato il seguente algoritmo:

1. La STA inserisce gli ingressi in una lista

2. L'algoritmo procede fino a quando tale lista non é vuota

2.1 estrae un elemento (gate o ingresso) dalla lista

2.2 se l'elemento é un ingresso mette il suo EAT e LST a 0 e lo marca come assegnato

2.3 se é un gate e tutto il suo fan-in é assegnato

• calcola EAT come il minimo EAT del fan-in piú il ritardo del gate

• calcola LST come il massimo LST del fan-in piú il ritardo del gate • marca il gate come assegnato

• carica il suo fan-out nella lista (se non presente)

2.4 uscita dal ciclo

• il minimo EAT é uguale a tRC,min

• il massimo LST é uguale a tRC,max.

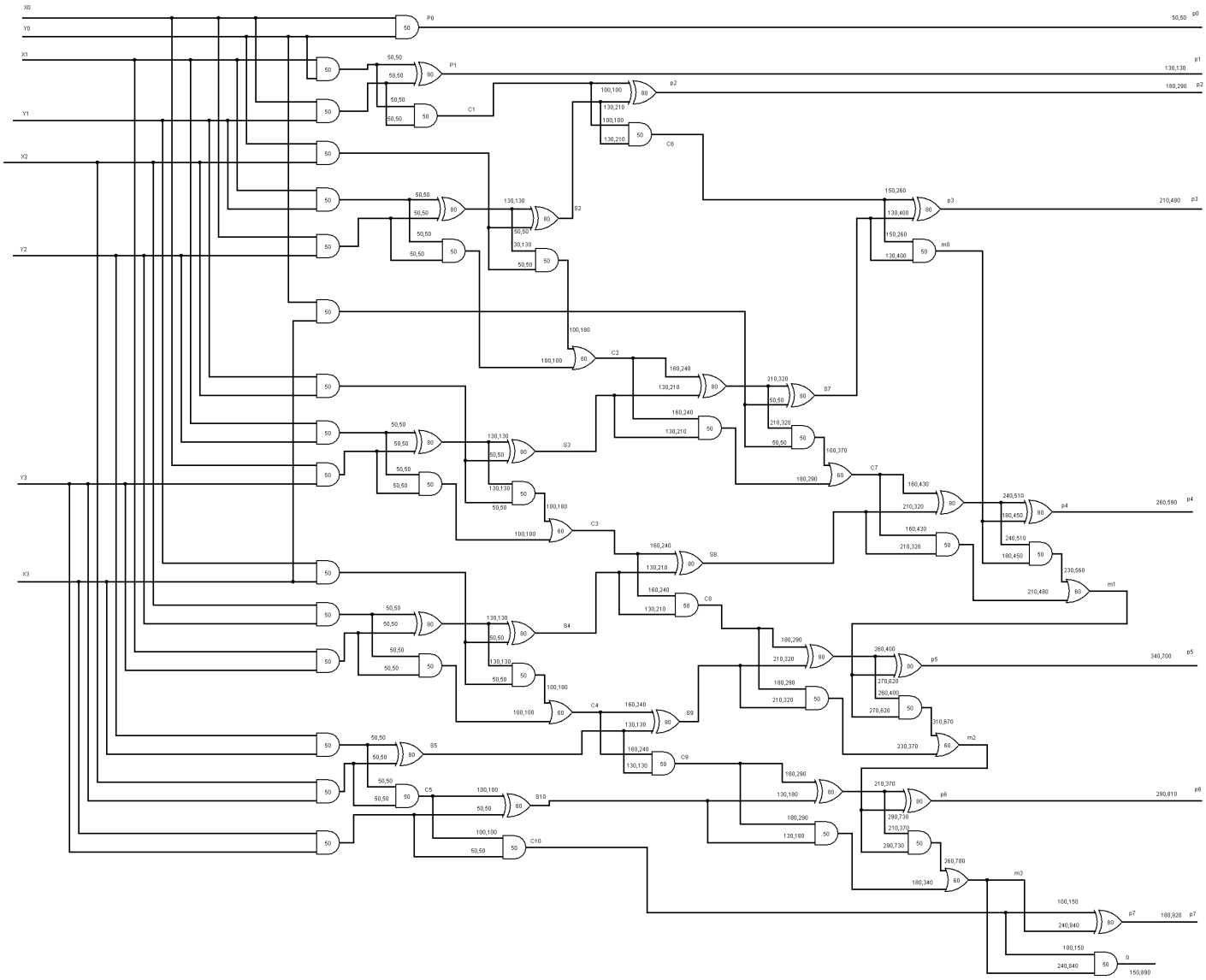

Fig.5: Schema circuitale del moltiplicatore di Wallace dopo aver applicato la STA.

I risultati ottenuti dall'applicazione della STA sono, come illustrato in Fig.5, i seguenti:

	EAT	**LST**
P0	50 ps	50 ps
P1	130 ps	130 ps
P2	180 ps	290 ps
P3	210 ps	480 ps
P4	250 ps	590 ps
P5	340 ps	700 ps
P6	290 ps	810 ps

	EAT	LST
P7	180 ps	920 ps
Cout(0)	150 ps	890 ps

È seguita, poi, l'implementazione del modello strutturale del moltiplicatore in linguaggio VHDL. Come prima cosa, si sono realizzati i modelli dei gate utilizzati dal circuito utilizzando descrizioni di tipo Dataflow:

- `y <= a and b after d;`
- `y <= a or b after d;`
- `y <= a xor b after d;`

In seguito è stato realizzato il modello strutturale dell'HA collegando tra loro AND e XOR come mostrato in Fig.2:

```
a0: entity work.and2cmos(dataflow) generic map(d=>dand)
port map(a=>a, b=>b, y=>cout);
x0: entity work.xor2cmos(dataflow) generic map(d=>dxor)
port map(a=>a, b=>b, y=>s);
```

La stessa cosa è stata fatta con il FA, seguendo lo schema in Fig.3:

```
x0: entity work.xor2cmos(dataflow) generic map(d=>dxor)
port map(a=>a, b=>b, y=>w0);
a0: entity work.and2cmos(dataflow) generic map(d=>dand)
port map(a=>a, b=>b, y=>w1);
x1: entity work.xor2cmos(dataflow) generic map(d=>dxor)
port map(a=>w0, b=>cin, y=>s);
a1: entity work.and2cmos(dataflow) generic map(d=>dand)
port map(a=>w0, b=>cin, y=>w2);
o0: entity work.or2cmos(dataflow) generic map(d=>dor)
port map(a=>w2, b=>w1, y=>cout);
```

Dopo aver testato HA e FA mediante opportuni testbench, per costruire il modello strutturale del moltiplicatore è stato necessario collegare tra loro gli AND in ingresso con HA e FA in cascata, come illustrato nello schema in Fig.4

```
    a0: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(0), b=>y(0), y=>p(0));
    a1: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(1), b=>y(0), y=>w(1));
    a2: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(0), b=>y(1), y=>w(2));
    a3: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(2), b=>y(0), y=>w(3));
    a4: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(1), b=>y(1), y=>w(4));
    a5: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(0), b=>y(2), y=>w(5));
    a6: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(3), b=>y(0), y=>w(6));
    a7: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(2), b=>y(1), y=>w(7));
    a8: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(1), b=>y(2), y=>w(8));
    a9: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(0), b=>y(3), y=>w(9));
    a10: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(3), b=>y(1), y=>w(10));
    a11: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(2), b=>y(2), y=>w(11));
    a12: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(1), b=>y(3), y=>w(12));
    a13: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(3), b=>y(2), y=>w(13));
    a14: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(2), b=>y(3), y=>w(14));
    a15: entity work.and2cmos(dataflow) generic map(d=>tand)
    port map(a=>x(3), b=>y(3), y=>w(15));

    h0: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>w(1), b=>w(2), s=>p(1), cout=>c(1));
```

```
    h1: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>w(13), b=>w(14), s=>s(5), cout=>c(5));
    h2: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>c(1), b=>s(2), s=>p(2), cout=>c(6));
    h3: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>c(3), b=>s(4), s=>s(8), cout=>c(8));
    h4: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>c(4), b=>s(5), s=>s(9), cout=>c(9));
    h5: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>c(5), b=>w(15), s=>s(10), cout=>c(10));
    h6: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>c(6), b=>s(7), s=>p(3), cout=>m(0));
    h7: entity work.half_adder(struct) generic map(dand=>tand,
dxor=>txor)
    port map(a=>m(3), b=>c(10), s=>p(7), cout=>carry_out);

    f0: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>w(4), b=>w(5), cin=>w(3), s=>s(2), cout=>c(2));
    f1: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>w(8), b=>w(9), cin=>w(7), s=>s(3), cout=>c(3));
    f2: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>w(11), b=>w(12), cin=>w(10), s=>s(4), cout=>c(4));
    f3: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>c(2), b=>s(3), cin=>w(6), s=>s(7), cout=>c(7));
    f4: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>c(7), b=>s(8), cin=>m(0), s=>p(4), cout=>m(1));
    f5: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
    port map(a=>c(8), b=>s(9), cin=>m(1), s=>p(5), cout=>m(2));
    f6: entity work.full_adder(struct) generic map(dand=>tand,
dxor=>txor, dor=>tor)
```

```
port map(a=>c(9), b=>s(10), cin=>m(2), s=>p(6), cout=>m(3));
```

Il modello è stato testato con un testbench che fornisce 22 vettori di ingresso casuali e tutte le operazioni hanno dato esito corretto.

È stato poi realizzato il modello comportamentale del moltiplicatore che, dopo aver convertito i due vettori in ingresso da std_logic a unsigned e aver eseguito la moltiplicazione tra questi, prima porta le uscite a ‘X’ con un ritardo di tipo trasporto pari al rispettivo EAT e, successivamente, sempre con un ritardo di tipo trasporto pari questa volta al LST, posta le uscite al corrispondente valore del prodotto riconvertite da unsigned a std_logic.

```
xu:=unsigned(x);
yu:=unsigned(y);
pu:=xu*yu;
p(0)<= transport std_logic(pu(0)) after 50 ps;
p(1)<= transport std_logic(pu(1)) after 130 ps;
p(2)<= transport 'X' after 180 ps;
p(2)<= transport std_logic(pu(2)) after 290 ps;
p(3)<= transport 'X' after 210 ps;
p(3)<= transport std_logic(pu(3)) after 480 ps;
p(4)<= transport 'X' after 260 ps;
p(4)<= transport std_logic(pu(4)) after 590 ps;
p(5)<= transport 'X' after 340 ps;
p(5)<= transport std_logic(pu(5)) after 700 ps;
p(6)<= transport 'X' after 290 ps;
p(6)<= transport std_logic(pu(6)) after 810 ps;
p(7)<= transport 'X' after 180 ps;
p(7)<= transport std_logic(pu(7)) after 920 ps;
carry_out <= transport 'X' after 150 ps;
carry_out <= transport '0' after 890 ps;
```

Simulando i due modelli in parallelo mediante un testbench di 22 vettori di ingresso casuali è emerso che in tutti i casi la commutazione dei segnali di uscita del modello strutturale avveniva all'interno dell'intervallo delimitato da EAT e LST calcolati con la STA e, cioè, l'intervallo in cui le uscite del modello comportamentale assumevano valore 'X' (un esempio è illustrato in Fig.6). Oltre ciò, si è trovato il critica path del circuito utilizzando il seguente metodo (Fig.7): si parte dall'uscita con il LST massimo e e si procede verso i PI selezionando l'ingresso di ciascuna porta logica con il LST massimo. Quando però si è cercata la sequenza di vettori di ingresso che sensibilizzasse tale cammino, è emerso che si trattava in realtà di un false path e che, quindi, non era di fatto percorribile. Da questo si può concludere che il ritardo massimo reale del moltiplicatore di Wallace è inferiore a quello calcolato teoricamente mediante STA.

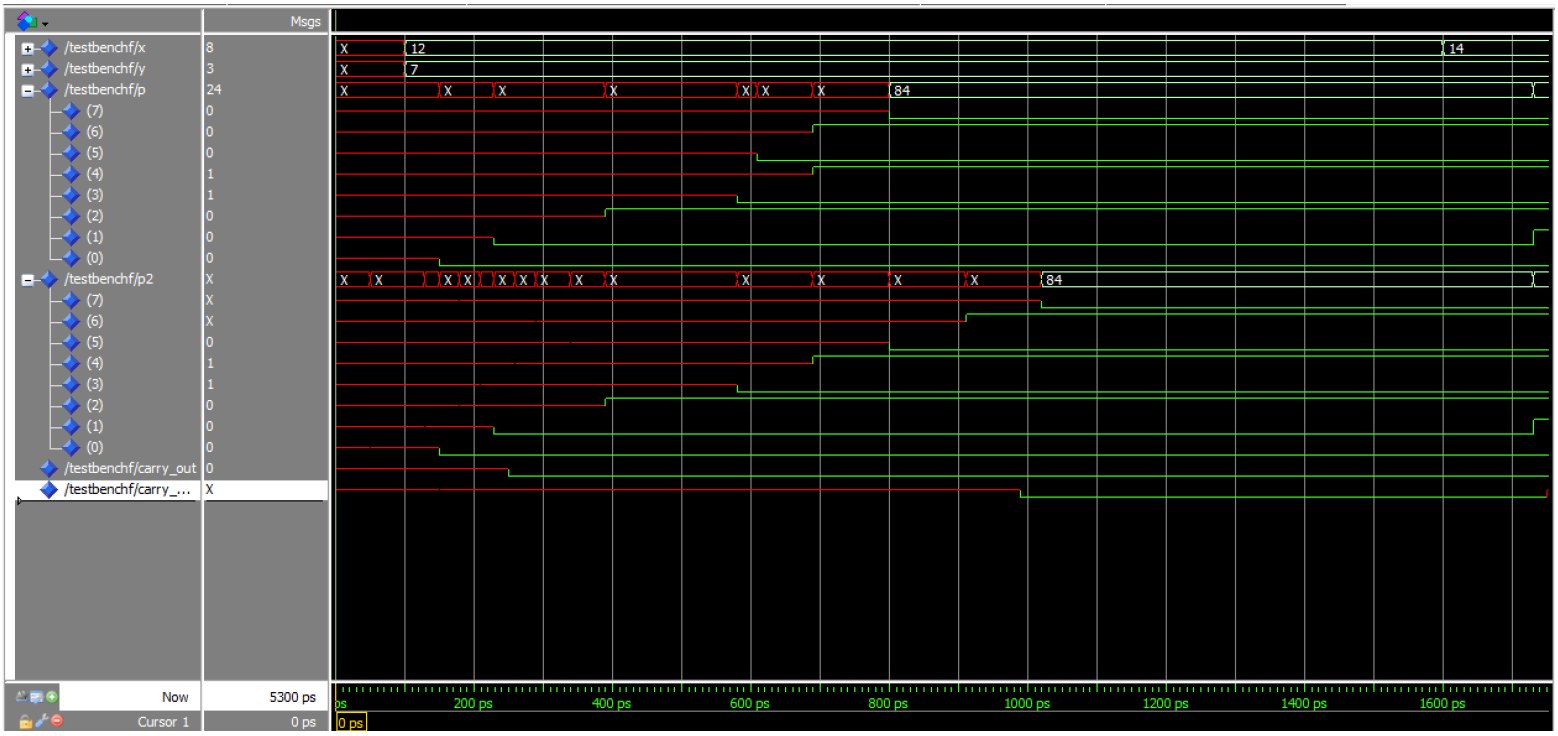

Fig.6: Esempio di simulazione del modello strutturale in parallelo con quello comportamentale del moltiplicatore di Wallace

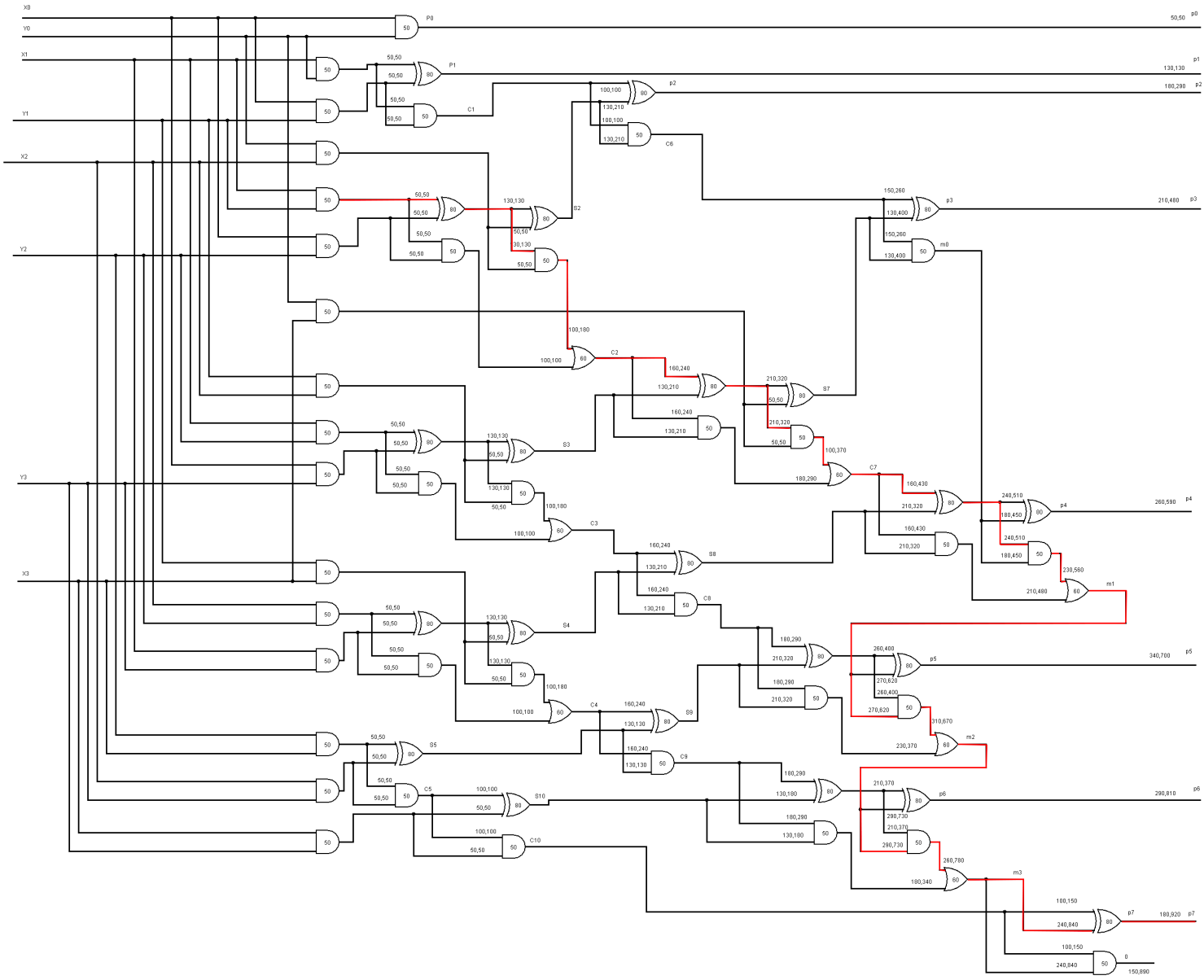

Fig7: Moltiplicatore di Wallace con evidenziato il citical path (rosso).

I ritardi di commutazione dei segnali in uscita registrati nel testbench sono riportati nella tabella in pico secondi. La “H” indica la presenza di un hazard statico e i due dati sono rispettivamente “prima commutazione-seconda commutazione” del segnale:

	X0	X1	X2	X3	X4	X5	X6	X7
13 x 7	50	130	290	480	510	510	590	700
14 x 5	50		290	370	480	H 400-510	H 360-610	
0 x 2		130	210	H 210-450	H 320-450		460	
8 x 3				210	370			
4 x 1			210	210	370			
13 x 14		130	H 210-290	H 370-450	370	480		230
10 x 13			290	H 210-370	480	510	H 340-610	H 230-730
13 x 10				H 210-370	H 320-450	H 350-480	H 370-450	H 480-560
13 x 4		130	290	H 370-450	450	340	H 290-370	400
15 x 0			290		450	370		
1 x 7	50	130	290					
13 x 12	50	130		450	450			230
14 x 2				H 370-450		H 400-480		180
6 x 1		130	H 210-290	480	370			
3 x 4		130	H 210-290	450				
12 x 7			H 210-290	210	320		450	
2 x 1			210	H 210-370	450		340	
9 x 3	50			210	370			
14 x 15			H 210-290	370			480	230
5 x 7	50			H 210-450	480	510	370	180
9 x 11			H 210-290	H 210-450	H 450-560	H 480-590	290	

	X0	X1	X2	X3	X4	X5	X6	X7
15 x 15		130		H 260-450	H 400-500	H 430-610	H 340-430	230

Dalla tabella sono stati calcolati i seguenti indici statistici:

	Moda	Media	Mediana	Min	Max
X0	50	50	50	50	50
X1	130	130	130	130	130
X2	290	274	290	210	290
X3	450	385	410	210	480
X4	450	438	450	320	560
X5	510	490	510	340	610
X6	340, 450, 610	454	450	290	610
X7	230	367	230	180	730