Università degli Studi di Verona – A.A. 2004/2005
Corso di Laurea in Informatica Multimediale

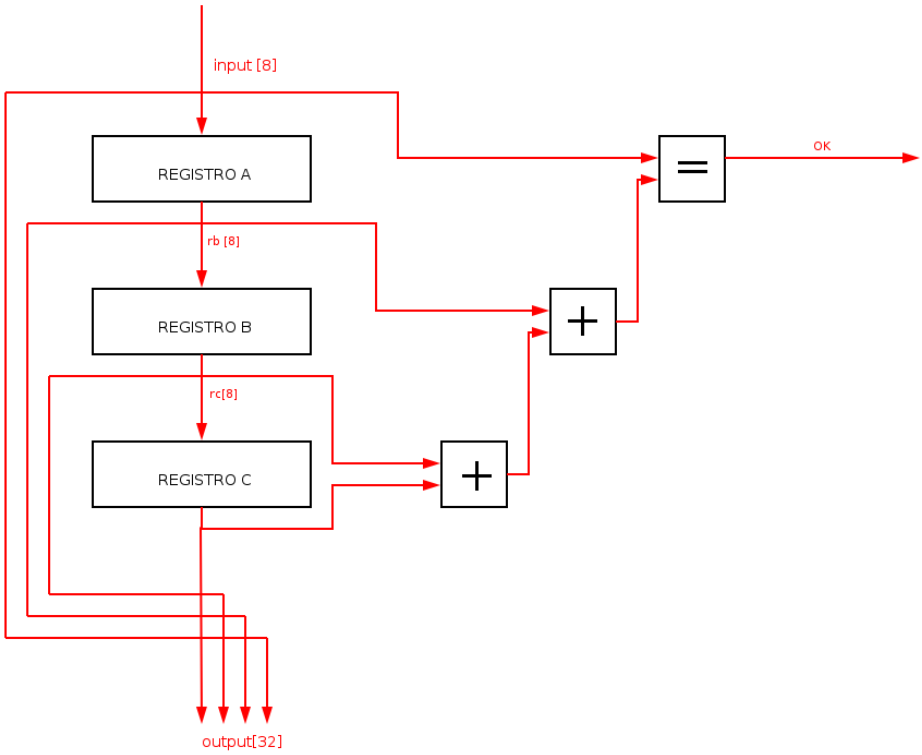
Elaborato SIS – Architettura degli Elaboratori

Pier Mattia Avesani – VR061764

“Si vuole progettare un circuito digitale per interfacciare una linea di trasmissione a 8 bit ad una linea di trasmissione a 32 bit. Sulla linea in ingresso arriva un nuovo byte ad ogni intervallo di clock. Ogni 3 byte arriva un byte corrispondente alla loro somma al fine di rilevare eventuali errori di trasmissione.
Il circuito deve concatenare 4 byte consecutivi sull'uscita a 32 bit. Esso deve anche ricalcolare la somma dei primi 3 byte e confrontarla col 4° byte per evidenziare errori di trasmissione.
La somma dei 3 byte avviene sempre considerandoli come numeri senza segno e ignorando il riporto dopo l'ottava cifra.”

Il circuito è formato da un Datapath e da un Controllore.
Di seguito c'è una spiegazione dettagliata della loro struttura.

Datapath
Il Datapath concatena i 4 byte sull'uscita a 32 bit e controlla che il quarto byte arrivato sia effettivamente la somma dei primi 3.



Questo è la struttura del Datapath realizzato, con un Input a un byte, un uscita a 4 byte e un uscita a un bit “OK” che esce dal comparatore tra i primi tre byte e il Cexam.
Sono presenti tre registri paralleli, due sommatori e un comparatore.

Registri
Sono stati usati tre registri paralleli a 8 bit.
Ne sono stati utilizzati tre e non quattro per la necessità di arrivare a un risultato da parte del Datapath in soli 4 cicli di clock, per rimanere nei tempi dettati dal controllore. Se ne avessi utilizzati quattro, infatti avrei dovuto aspettare il quinto clock per far uscire il dato dall'ultimo registro riempito e poterlo utilizzare.
L'implementazione dei tre registri è contenuta nei file reg8.blif (registro a 8 bit) e reg.blif (registro a 1 bit).

Sommatori
Sono stati utlilizzati due sommatori con due ingressi da 8 bit ciascuno.
Sono stati utilizzati per sommare i dati provenienti dai tre registri, che sono i primi 3 byte arrivati in input. L'uscita va al comparatore.
L'implementazione dei due sommatori è contenuta nei file somm8.blif (sommatore a 8 bit) e sommatore.blif (sommatore a 1 bit).

Comparatore
Il comparatore ha due ingressi da 8 bit, e restituisce un bit che esce dal datapath e dalla FSMD che vale 1 se il Cexam (il quarto byte) e i primi 3 byte sono uguali.
L'implementazione del comparatore è contenuta nei file comp8.blif (comparatore a 8 bit) e xnor.blif (porta xnor a 1 bit).

Il Datapath è stato implementato in SIS nel file DP.blif:

```
.model DP
.inputs i7 i6 i5 i4 i3 i2 i1 i0
.outputs o31 o30 o29 o28 o27 o26 o25 o24 o23 o22 o21 o20
o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4
o3 o2 o1 o0 ok


#gli 8 bit entrano nel primo registro
.subckt reg8 a7=i7 a6=i6 a5=i5 a4=i4 a3=i3 a2=i2 a1=i1
a0=i0 o7=rb7 o6=rb6 o5=rb5 o4=rb4 o3=rb3 o2=rb2 o1=rb1
o0=rb0


#entrano nel secondo registro
.subckt reg8 a7=rb7 a6=rb6 a5=rb5 a4=rb4 a3=rb3 a2=rb2
a1=rb1 a0=rb0 o7=rc7 o6=rc6 o5=rc5 o4=rc4 o3=rc3 o2=rc2
o1=rc1 o0=rc0


#entrano nel terzo registro
```

```
.subckt reg8 a7=rc7 a6=rc6 a5=rc5 a4=rc4 a3=rc3 a2=rc2
a1=rc1 a0=rc0 o7=rd7 o6=rd6 o5=rd5 o4=rd4 o3=rd3 o2=rd2
o1=rd1 o0=rd0

#gli ultimi due entrano nel sommatore
.subckt somm8 a7=rd7 a6=rd6 a5=rd5 a4=rd4 a3=rd3 a2=rd2
a1=rd1 a0=rd0 b7=rc7 b6=rc6 b5=rc5 b4=rc4 b3=rc3 b2=rc2
b1=rc1 b0=rc0 o7=so7 o6=so6 o5=so5 o4=so4 o3=so3 o2=so2
o1=so1 o0=so0


#il primo entra nella somma con gli ultimi due sommati
.subckt somm8 a7=rb7 a6=rb6 a5=rb5 a4=rb4 a3=rb3 a2=rb2
a1=rb1 a0=rb0 b7=so7 b6=so6 b5=so5 b4=so4 b3=so3 b2=so2
b1=so1 b0=so0 o7=sob7 o6=sob6 o5=sob5 o4=sob4 o3=sob3
o2=sob2 o1=sob1 o0=sob0


#la somma e il CEXAM entrano nel comparatore
.subckt comp8 A7=i7 A6=i6 A5=i5 A4=i4 A3=i3 A2=i2 A1=i1
A0=i0 B7=sob7 B6=sob6 B5=sob5 B4=sob4 B3=sob3 B2=sob2
B1=sob1 B0=sob0 OUT=ok


#definisco le uscite
.names rd7 o31
1 1
.names rd6 o30
1 1
.names rd5 o29
1 1
.names rd4 o28
1 1
.names rd3 o27
1 1
.names rd2 o26
1 1
.names rd1 o25
1 1
.names rd0 o24
1 1
.names rc7 o23
1 1
.names rc6 o22
1 1
.names rc5 o21
1 1
.names rc4 o20
1 1
.names rc3 o19
1 1
.names rc2 o18
```

```
1 1
.names rc1 o17
1 1
.names rc0 o16
1 1
.names rb7 o15
1 1
.names rb6 o14
1 1
.names rb5 o13
1 1
.names rb4 o12
1 1
.names rb3 o11
1 1
.names rb2 o10
1 1
.names rb1 o9
1 1
.names rb0 o8
1 1
.names i7 o7
1 1
.names i6 o6
1 1
.names i5 o5
1 1
.names i4 o4
1 1
.names i3 o3
1 1
.names i2 o2
1 1
.names i1 o1
1 1
.names i0 o0
1 1


.end
.search comp8.blif
.search somm8.blif
.search reg8.blif
.search multi8.blif
```

La simulazione del file DP.blif porta i seguenti risultati:

```
sis> read_blif DP.blif
sis> simulate 0 0 0 0 0 0 0 1

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

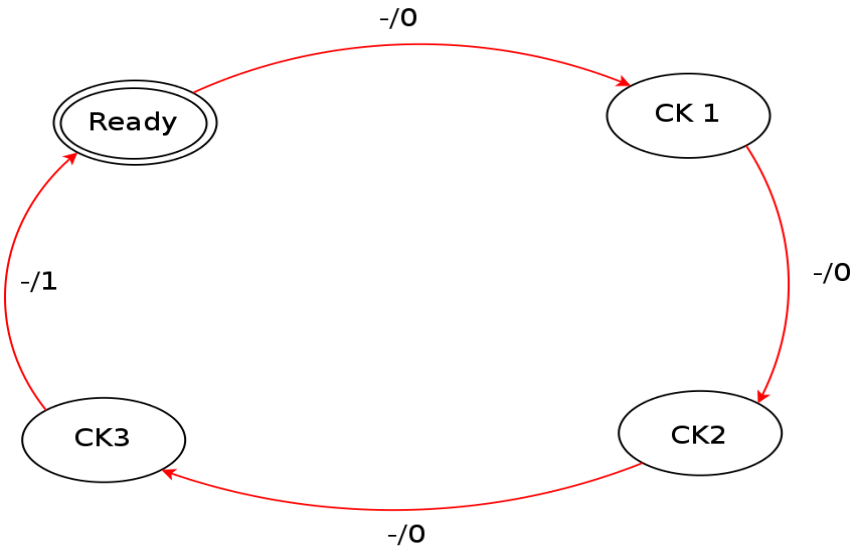
```
0 0 0 0 0 0 0 1 0
Next state: 000000010000000000000000
sis> simulate 0 0 0 0 0 0 1 0

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0 0
Next state: 000000100000000100000000
sis> simulate 0 0 0 0 0 1 0 0

Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0
Next state: 000001000000001000000001
sis> simulate 0 0 0 0 0 1 1 1

Network simulation:
Outputs: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 1 1
Next state: 000001110000010000000010
```

E' proprio quello che si voleva ottenere: al primo ciclo di clock entra 00000001, al secondo 00000010, al terzo 00000100, al quarto il Cexam, che è 00000111, come la somma dei primi 3. Infatti l'ultimo bit in uscita dal Datapath, che è l'OK, vale 1 solo al quarto ciclo di clock, quando effettivamente il comparatore controlla i primi tre byte con il quarto.I primi 32 bit in uscita sono effettivamente i 4 byte entrati in ingresso concatenati tra loro.

Controllore

Il controllore ha il compito di “contare” i 4 cicli di clock, restituendo a valle un bit “Ready” che vale uno quando sono stati compiuti i 4 cicli, cioè quando l'uscita del Datapath contiene i 4 byte concatenati e il valore corretto di OK. Se i cicli da contare fossero stati in numero maggiore si avrebbe dovuto implementare un contatore dentro al Datapath, ma visto che i cicli da contare erano solo 4, il compito di contare è stato dato al controllore.

Lo *State Transition Graph* della FSM di controllo è così strutturato:



Si nota che al quarto ciclo (cioè al quarto clock) restituisce 1 in uscita qualsiasi sia l'entrata.Infatti starà all'operatore a valle controllare i valori di Ready e OK.
La FSM è stata implementata in SIS nel seguente modo:

```
.model FSM
.inputs r
.outputs OUT
.start_kiss

#numero di ingressi
.i 1

#numero di uscite
.o 1

#numero di stati
.s 4

#numero di transizioni
.p 4

#stato di reset
.r INIT

#tabella delle transizioni
- INIT A 0
- A B 0
- B C 0
- C INIT 1

.end_kiss
```

Si nota che è presente un input r non necessario ai controlli della FSM, ma necessario all'implementazione in SIS.

Da questa FSM sono stati assegnati gli stati e è stata costruita la rete attraverso i comandi SIS `state_assign jedi` e `stg_to_network`: il risultato di queste due operazioni è visibile all'interno del file FSMJS.blif .

La simulazione della FSM con l'assegnamento degli stati (il valore di input è zero, poteva essere uno che nulla cambiava, come già spiegato):

```
sis> read_blif FSM.blif
sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> stg_to_network
sis> write_blif FSMJS.blif
sis> simulate 0

Network simulation:
Outputs: 0
```

```
Next state: 10

STG simulation:
Outputs: 0
Next state: A (10)

sis> simulate 0

Network simulation:
Outputs: 0
Next state: 11

STG simulation:
Outputs: 0
Next state: B (11)

sis> simulate 0

Network simulation:
Outputs: 0
Next state: 00

STG simulation:
Outputs: 0
Next state: C (00)

sis> simulate 0

Network simulation:
Outputs: 1
Next state: 01

STG simulation:
Outputs: 1
Next state: INIT (01)

sis> simulate 0

Network simulation:
Outputs: 0
Next state: 10

STG simulation:
Outputs: 0
Next state: A (10)
```

Ottimizzazioni

Si ottimizzeranno i componenti del circuito in modo da ridurne per quanto possibile nodi, letterali, area e ritardo. Come da richiesta, il circuito sarà mappato sulla libreria tecnologica synch.genlib. Per ottimizzare si farà uso dello `script.rugged` e, per quanto riguarda il controllore, anche del comando SIS state_minimize stamina per ridurre il numero degli stati del nostro circuito. Per visualizzare le statistiche si userà il comando SIS `print_stats`.

Carico il sommatore in sis e eseguo il comando `print_stats`, dopo di che si ottimizza e si confrontano i risultati:

```
sis> read_blif somm8.blif
sis> print_stats
somm8           pi=16   po= 8   nodes=  8       latches=
0
lits(sop)=  32

sis> source script.rugged
sis> print_stats
somm8           pi=16   po= 8   nodes=  8       latches=
0
lits(sop)=  32
sis>
```

Si carica il comparatore in SIS e si eseguono le stesse operazioni:

```
sis> read_blif comp8.blif
sis> print_stats
comp8           pi=16   po= 1   nodes=  9       latches=
0
lits(sop)=  40
sis> source script.rugged
sis> print_stats
comp8           pi=16   po= 1   nodes=  7       latches=
0
lits(sop)=  64
```

Si nota che il sommatore è già in condizione ottimale,mentre il comparatore ottimizzato ha 2 nodi in meno, ma 24 letterali in più.

Una volta ottimizzati i componenti si passa all'ottimizzazione dell'intero Datapath, dopo aver controllato lo stato attuale di noti e letterali. Visto che si è deciso di tenere i componenti non ottimizzati, per non far crescere troppo i letterali, si procede a ottimizzare il DP.blif senza ricostruirlo con i nuovi componenti ottimizzati, ma con quelli originali. Si nota che lo script.rugged elimina 26 nodi, lo sweep non ne toglie, quindi non ce ne sono di inutili, e che il simplify e il full_simplify, che implementano Quine McKluskey, non portano nessun miglioramento:

```
sis> read_blif DP.blif
sis> print_stats
DP              pi= 8   po=33   nodes= 49
latches=24
lits(sop)= 128
```

```
sis> source script.rugged
sis> print_stats
DP              pi= 8   po=33   nodes= 23
latches=24
lits(sop)= 128

sis> sweep
sis> print_stats
DP              pi= 8   po=33   nodes= 23
latches=24
lits(sop)= 128

sis> simplify
sis> print_stats
DP              pi= 8   po=33   nodes= 23
latches=24
lits(sop)= 128

sis> full_simplify
sis> print_stats
DP              pi= 8   po=33   nodes= 23
latches=24
lits(sop)= 128
```

Il Datapath ottimizzato è stato salvato in DP_ottimizzato.blif e verrà implementato nella FSMD

Ottimizzazione del Controllore FSM

Si passa all'ottimizzazione della FSM:
Lettura del file FSM.blif, minimizzazione degli stati da stamina, assegnamento degli stati e creazione rete e salvataggio sul file FSMJS.blif, e minimizzazione con lo script rugged con salvataggio in file FSMJDEF.blif

```
sis> read_blif FSM.blif
sis> state_minimize stamina
Running stamina, written by June Rho, University of
Colorado at Boulder
Number of states in original machine : 4
Number of states in minimized machine : 4

sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> stg_to_network
sis> print_stats
FSM             pi= 1   po= 1   nodes=  3
latches= 2
lits(sop)=  10  #states(STG)=   4
```

```
write_blif FSMJS.blif

sis> read_blif FSMJS.blif
Warning: network `FSMJS', node "r" does not fanout
sis> print_stats
FSMJS           pi= 1   po= 1   nodes=  3
latches= 2
lits(sop)=  10  #states(STG)=   4
sis> source script.rugged
sis> print_stats
FSMJS           pi= 1   po= 1   nodes=  3
latches= 2
lits(sop)=   7  #states(STG)=   4

sis> write_blif FSMJDEF.blif
```

Ottimizzazione dell'intero circuito della FSMD

Si implementa nel file FSMD.blif la nuova FSMJDEF.blif che è stata ottimizzata e si passa all'ottimizzazione dell'intero circuito:

```
FSMD SENZA DP OTTIMIZZATO

sis> read_blif FSMD.blif
Warning: `DP', ".subckt somm8": formal input "CIN" not driven
Warning: `DP', ".subckt somm8": formal input "CIN" not driven
Warning: network `FSMJDEF', node "r" does not fanout
Warning: network `DP', node "CIN" is not driven (zero assumed)
Warning: network `DP', node "[9755]" does not fanout
Warning: network `DP', node "[9788]" is not driven (zero assumed)
Warning: network `DP', node "[9789]" does not fanout
Warning: network `FSMD', node "r" does not fanout
Warning: network `FSMD', node "[9755]" does not fanout
Warning: network `FSMD', node "[9789]" does not fanout
sis> print_stats
FSMD            pi= 9   po=34   nodes= 86
latches=26
lits(sop)= 295
sis> sweep
sis> print_stats
FSMD            pi= 9   po=34   nodes= 56
latches=26
lits(sop)= 243
sis> source script.rugged
sis> print_stats
FSMD            pi= 9   po=34   nodes= 54
latches=26
lits(sop)= 243
sis> full_simplify
sis> print_stats
FSMD            pi= 9   po=34   nodes= 54
latches=26
lits(sop)= 243
```

Si nota che il comando sweep elimina ben 30 nodi inutili. Con lo `script.rugged` e il `full_simplify` si arriva a un risultato finale di 32 nodi eliminati e 52 letterali in meno. Si confronta ora questo risultato con quello ottenuto ottimizzando la FSMD con anche il DP otttimizzato.

```
FSMD CON DP OTTIMIZZATO


sis> read_blif FSMD.blif

Warning: network `FSMJDEF', node "r" does not fanout
Warning: network `FSMD', node "r" does not fanout

sis> print_stats
FSMD            pi= 9   po=34   nodes= 54       latches=26
lits(sop)= 243

sis> sweep
sis> print_stats
FSMD            pi= 9   po=34   nodes= 54       latches=26
lits(sop)= 243
sis> source script.rugged
sis> print_stats
FSMD            pi= 9   po=34   nodes= 54       latches=26
lits(sop)= 243
sis> full_simplify
sis> print_stats
FSMD            pi= 9   po=34   nodes= 54       latches=26
lits(sop)= 243
```

Evidentemente i 30 nodi eliminati dallo sweep sono collegati ai 26 nodi che erano stati eliminati ottimizzando il DP separatamente. Si va ora a confrontare i risultati delle due FSMD mappando il circuito nella libreria tecnologica synch.genlib

```
FSMD SENZA DP OTTIMIZZATO

sis> read_library synch.genlib
sis> map -W -m 0 -s
>>> before removing serial inverters <<<
# of outputs:          60
total gate area:       5610.00
maximum arrival time: (26.80,26.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-26.80,-26.80)
```

```
total neg slack:      (-110.00,-110.00)
# of failing outputs:  60
>>> before removing parallel inverters <<<
# of outputs:          60
total gate area:       5610.00
maximum arrival time: (26.80,26.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-26.80,-26.80)
total neg slack:      (-110.00,-110.00)
# of failing outputs:  60
# of outputs:          60
total gate area:       5610.00
maximum arrival time: (26.80,26.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-26.80,-26.80)
total neg slack:      (-110.00,-110.00)
# of failing outputs:  60
sis> sweep
sis> source script.rugged
sis> map -W -m 0 -s
>>> before removing serial inverters <<<
# of outputs:          60
total gate area:       5362.00
maximum arrival time: (35.80,35.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-35.80,-35.80)
total neg slack:      (-110.80,-110.80)
# of failing outputs:  60
>>> before removing parallel inverters <<<
# of outputs:          60
total gate area:       5346.00
maximum arrival time: (36.00,36.00)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-36.00,-36.00)
total neg slack:      (-111.00,-111.00)
# of failing outputs:  60
# of outputs:          60
total gate area:       5218.00
maximum arrival time: (35.80,35.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-35.80,-35.80)
total neg slack:      (-110.80,-110.80)
# of failing outputs:  60
```

FSMD CON DP OTTIMIZZATO

```
sis> read_library synch.genlib
sis> map -W -m 0 -s
```

```
>>> before removing serial inverters <<<
# of outputs:          60
total gate area:       5658.00
maximum arrival time: (34.80,34.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-34.80,-34.80)
total neg slack:      (-108.80,-108.80)
# of failing outputs:  60
>>> before removing parallel inverters <<<
# of outputs:          60
total gate area:       5610.00
maximum arrival time: (35.40,35.40)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-35.40,-35.40)
total neg slack:      (-109.40,-109.40)
# of failing outputs:  60
# of outputs:          60
total gate area:       5258.00
maximum arrival time: (34.60,34.60)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-34.60,-34.60)
total neg slack:      (-108.60,-108.60)
# of failing outputs:  60
sis> sweep
sis> source script.rugged
sis> map -W -m 0 -s
>>> before removing serial inverters <<<
# of outputs:          60
total gate area:       5610.00
maximum arrival time: (47.60,47.60)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-47.60,-47.60)
total neg slack:      (-122.00,-122.00)
# of failing outputs:  60
>>> before removing parallel inverters <<<
# of outputs:          60
total gate area:       5562.00
maximum arrival time: (47.60,47.60)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-47.60,-47.60)
total neg slack:      (-122.00,-122.00)
# of failing outputs:  60
# of outputs:          60
total gate area:       5242.00
maximum arrival time: (46.80,46.80)
maximum po slack:     (-0.40,-0.40)
minimum po slack:     (-46.80,-46.80)
total neg slack:      (-121.20,-121.20)
# of failing outputs:  60
```

Si nota che l'ottimizzazione della FSMD senza DP ottimizzato dopo la mappatura del circuito porta a un notevole miglioramento: si va da un area di 5610.00 a un area di 5218.00, anche se c'è un leggero incremento del ritardo massimo: da un ritardo massimo di 26.80 a un ritardo massimo di 35.80

Invece i risultati ottenuti con la mappatura della FSMD con DP ottimizzato sono molto diversi:
Prima della ottimizzazione: Area 5258.00 Ritardo 34.60
Dopo la ottimizzazione: Area 5242.00 Ritardo 46.80

CONCLUSIONI

I "`warning`" nell'esecuzione di SIS sono legati a due fatti. Il primo è la necessità di avere almeno un ingresso nella FSM quando la si implementa in SIS. La nostra FSM infatti potrebbe funzionare benissimo senza inputs, ma SIS ne necessita almeno uno. Quindi il primo `warning` non è altro che un avviso di SIS che informa che all'input della FSM non è stato collegato nulla. Il secondo fatto che lancia i `warning` è l'assenza di un ingresso nel *carry in* dei due sommatori e l'assenza di un uscita nel *carry out.* Non dovendo tenere conto dei resti e dei riporti infatti, il *carry in* e il *carry out* sono inutili nel nostro DP, e sono stati lasciati senza connessioni.
Analizzando l'area e il ritardo delle ultime due ottimizzazioni si è scelto di tenere la FSMD senza DP ottimizzato, visto che la mappatura automatica fatta in SIS sembra essere più pulita e porta a dei risultati di simulazione migliori.
Tuttavia dal *print_stats* delle due FSMD dopo l'ottimizzazione totale si arriva a risultati leggermente migliori in termini di nodi e letterali, quindi si presume che con un altra libreria tecnologica area e ritardo darebbero ragione al circuito con DP ottimizzato in precedenza.