ELABORATO SIS

ARCHITETTURA DEGLI ELABORATORI

A.A. 2016/2017 - Corso di Laurea in Informatica

Verona, 15 Febbraio 2017

Lutteri Nicolò (VR408112)

Chiarani Fabio (VR408541)

SOMMARIO

INTRODUZIONE

La relazione viene fornita assieme a tutti i sorgenti del progetto.
Il progetto è così strutturato:

- *./non_ottimizzato* : cartella in cui sono presenti tutti i sorgenti non ottimizzati del progetto, utili a capire com'era stato realizzato;
- *./ (root)*: cartella in cui sono presenti tutti i sorgenti ottimizzati, e per questo, molti potrebbero non essere comprensibili;

I file più importanti contenuti nelle rispettive sottodirectory sono:

- *CONTROLLO.blif* : è il controllore (FSM) prima dell'assegnamento degli stati;
- *CONTROLLOJEDI.blif* : è il controllore (FSM) dopo l'assegnamento degli stati;
- *ELABORAZIONE.blif* : è il datapath del circuito;
- *FSMD.blif* : è il file contenente il circuito completo raggruppando *CONTROLLOJEDI.blif* ed *ELABORAZIONE.blif;*

SPECIFICHE

Si progetti un dispositivo per il monitoraggio di un impianto chimico industriale basato su un circuito sequenziale che riceve come input il pH di una soluzione contenuta in un serbatoio, e fornisce in uscita lo stato della soluzione (compatibilmente con delle soglie pre-impostate) in termini di acido (A), basico (B) e neutro (N). Il sistema deve portare sempre la soluzione allo stato neutro, accettando un transitorio di 5 cicli di clock; pertanto si richiede che al sesto ciclo di clock in cui il sistema sia allo stato A, venga aperta una valvola BS che riporti il sistema a N, e analogamente se allo stato B si apra la valvola AS. Il sistema deve inoltre fornire in uscita il numero di cicli di clock da cui si trova nello stato attuale.

INPUTS:

- ***INIT [1]*** : quando vale 1 il sistema è acceso; quando vale 0 il sistema è spento e deve restituire 0 per tutti i bit di output.
- ***RESET [1] :*** quando posto a 1 il controllore deve essere resettato, ovvero tutte le uscite devono essere poste a 0 e il sistema riparte.
- ***PH [8]*** : valore del pH misurato dal rilevatore. Il range di misura è compreso tra 0 e 14 con risoluzione di 0,1.

OUTPUTS:

- ***ST [2] :*** indica in quale stato si trova la soluzione al momento corrente (01-A, 10-N, 11-B). Si considera la soluzione acida (A) quando PH<6 e basica (B) se PH>8.
- ***NCK [8]*** : indica il numero di cicli di clock trascorsi nello stato corrente
- ***VLV [2] :*** indica quale valvola aprire per riportare la soluzione allo stato

IPOTESI AGGIUNTIVE

Valore del pH ed errore:

Dato che il valore in ingresso del pH è fornito su 8 bit, quindi 255 combinazioni, è stato scelto di utilizzare una precisione di 0,1 in modo che il valore del pH (compreso tra 0 e 14) potesse occuparne 140. Tutti i valori inutilizzati al disopra del 140 (quindi con un pH di valore maggiore a 14) vengono letti come errore dal datapath che trasmetterà come stato *SS1 / SS0* l'errore (*01*) al controllore, il quale, ricevendo questo errore porta allo stato di inizio come se ricevesse il comando di reset.

Questo permette al ciclo di clock successivo di rileggere correttamente il nuovo valore del pH e raggiungere così lo stato adeguato nel controllore.

Il controllore non ha uno stato apposito per la gestione degli errori poiché ogni stato, in caso di errore, va allo stato di inizio dando così la possibilità di eseguire una seconda lettura corretta nel ciclo di clock successivo. Questo permette di risparmiare uno stato nel controllore e quindi diminuire transazioni che serviranno in seguito a rendere migliore la minimizzazione.

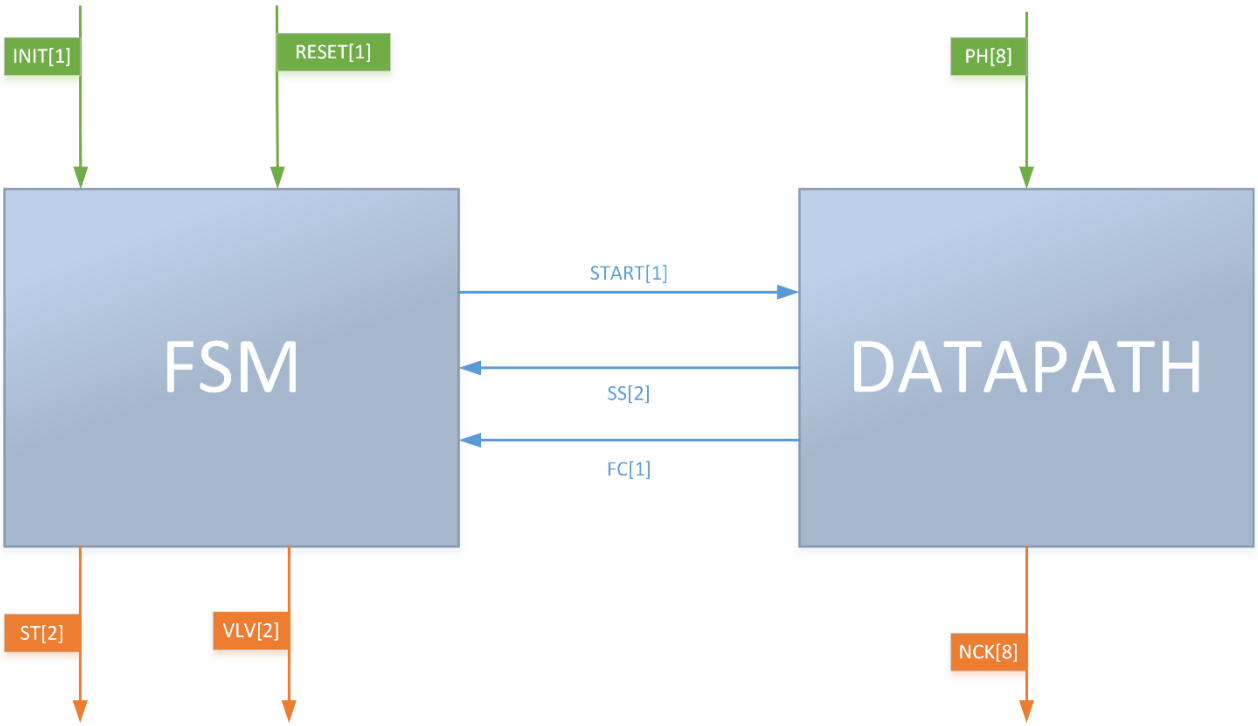
Comparatori del datapath:

Per comparare il valore del pH al valore di *80*, *59* e *140* (in caso di errore) è stato scelto di utilizzare un componente di comparazione 'maggiore' per tutte e tre le verifiche. La scelta di utilizzare lo stesso componente, anziché un componente 'maggiore' e uno 'minore' ci ha permesso di minimizzare, seppur di poco, il nostro circuito del datapath mantenendo comunque il corretto funzionamento.

FSDM

È stato scelto di tenere lo schema della FSMD come indicato nelle specifiche poiché ci risultava più adeguato per lo sviluppo del progetto. La FSMD (*FSMD.blif*) connette il controllore (*FSM*) e l'elaboratore (*datapath*). Entrambi sono sincronizzati con lo stesso ciclo di clock e interagiscono tra di loro con lo stato di ***FC*** *[1]* (stato di fine clock), ***START*** *[1]* (inizio conteggio dei cicli di clock) e ***SS*** *[2]* (stato della soluzione).

L' FSMD è così strutturata:



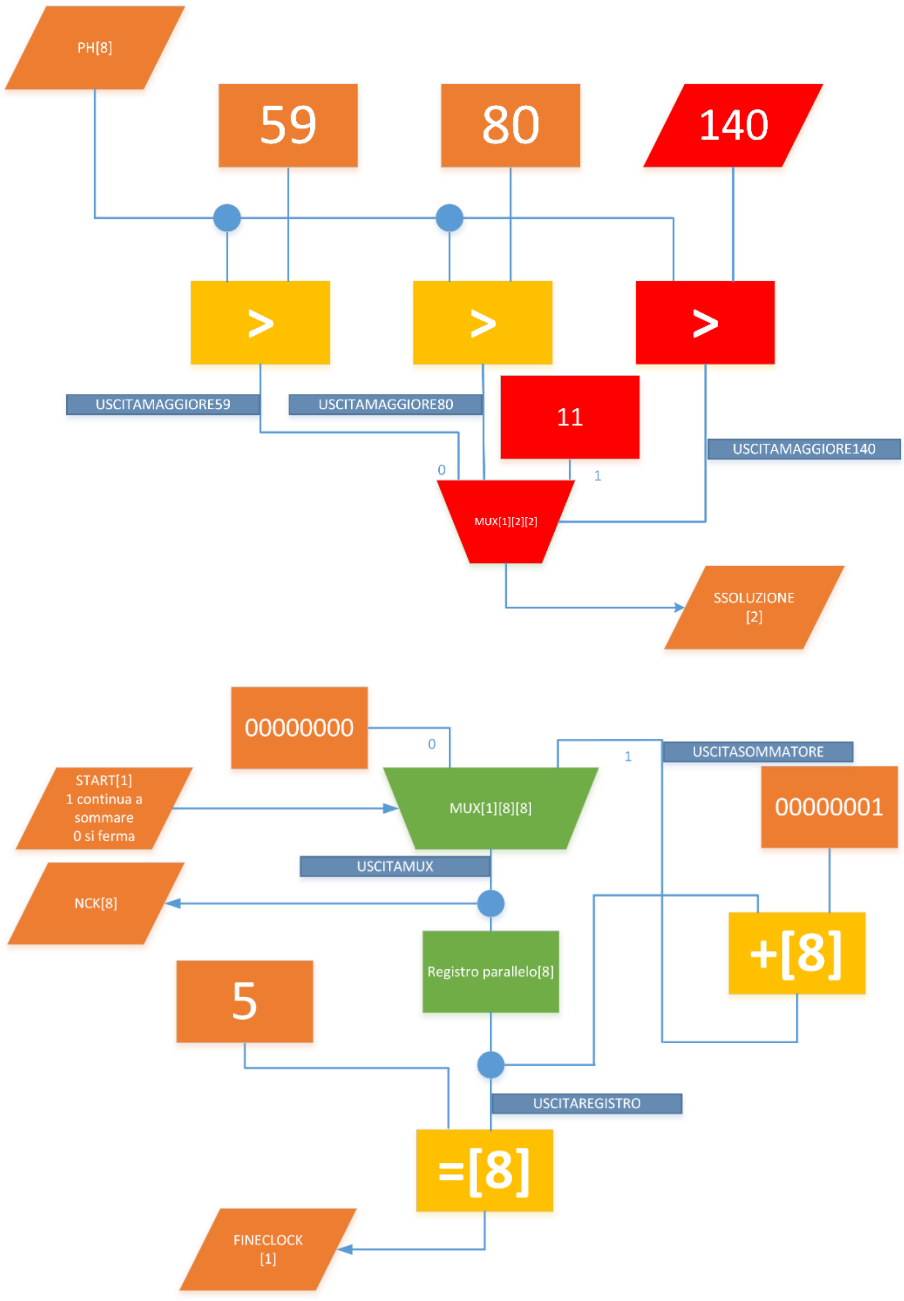
ARCHITETTURA DEL DATAPATH

Il datapath si occupa di verificare il valore del pH dato in input e stabilire se è acido (00), basico (11), neutro (10) o fuori dal range (errore 01). Una volta stabilito il valore del pH, il datapath lo fornisce in out sui valori di *SS1* e *SS0*. Un secondo compito essenziale del datapath è quello di tenere conto dei cicli di clock e fornirli in out sui valori di *NCK*.

Il datapath è così strutturato:

I = { INIT, RESET, PH8, PH7, PH6, PH5, PH4, PH3, PH2, PH1, PH0 }

O = { NCK7, NCK6, NCK5, NCK4, NCK3, NCK2, NCK1, NCK0, SS1, SS0, FC }



Descrizione confronto del pH:

Il datapath riceve il valore del pH su 8bit negli input [*PH7, …, PH1, PH0*]. Dopodiché il pH viene confrontato con degli operatori di comparazione 'maggiori' con i valori 59, 80 e 140.

- Se il valore è minore di 59 il pH è acido con codice [00]
- Se il valore è maggiore di 80 il pH è basico con codice [11]
- Se il valore è compreso tra 59 e 80 il pH è neutro con codice [10]
- Se il valore supera il 140, quindi fuori dal range del pH (0-14), è errore con codice [01]

Questi valori in uscita su *SS1* e *SS0* serviranno per comunicare con l'FSM.

Descrizione del conteggio dei clock:

Nel secondo compito, di contare i cicli di clock, il datapath in base all'input di *START* conteggia (1) o azzera il conteggio (0). Quando il valore di *START* è costante a 1, il datapath con l'utilizzo di un registro parallelo da 8 bit conta tutti i cicli di clock. Per le specifiche che indicano un conteggio massimo di 6 cicli di clock basterebbero 3 bit per il registro, ma siccome l'out di *NCK* è da 8 bit è stato deciso di utilizzare un registro parallelo da 8 bit. Quando il datapath, tramite un operatore di confronto, ha contato 5 cicli di clock, il valore di out *FC* (fine clock) viene impostato a 1 per indicare all'FSM di aprire una valvola.

Riepilogo componenti:

Il datapath è costituito da 8 principali componenti:

- 3 comparatori 'maggiori': necessari per confrontare l'ingresso del ph con i valori 59, 80, e 240
- 1 comparatore 'maggiore': necessario per verificare se il conteggio dei clock è superiore a 5 cicli
- un registro parallelo: da 8 bit tiene conto del numero dei cicli di clock
- un sommatore: necessario per incrementare di 1 il valore del registro ad ogni ciclo di clock
- 2 MUX: uno per resettare il conteggio dei clock (*NCK*) e uno per la selezione del valore del pH (Errore o nel range)

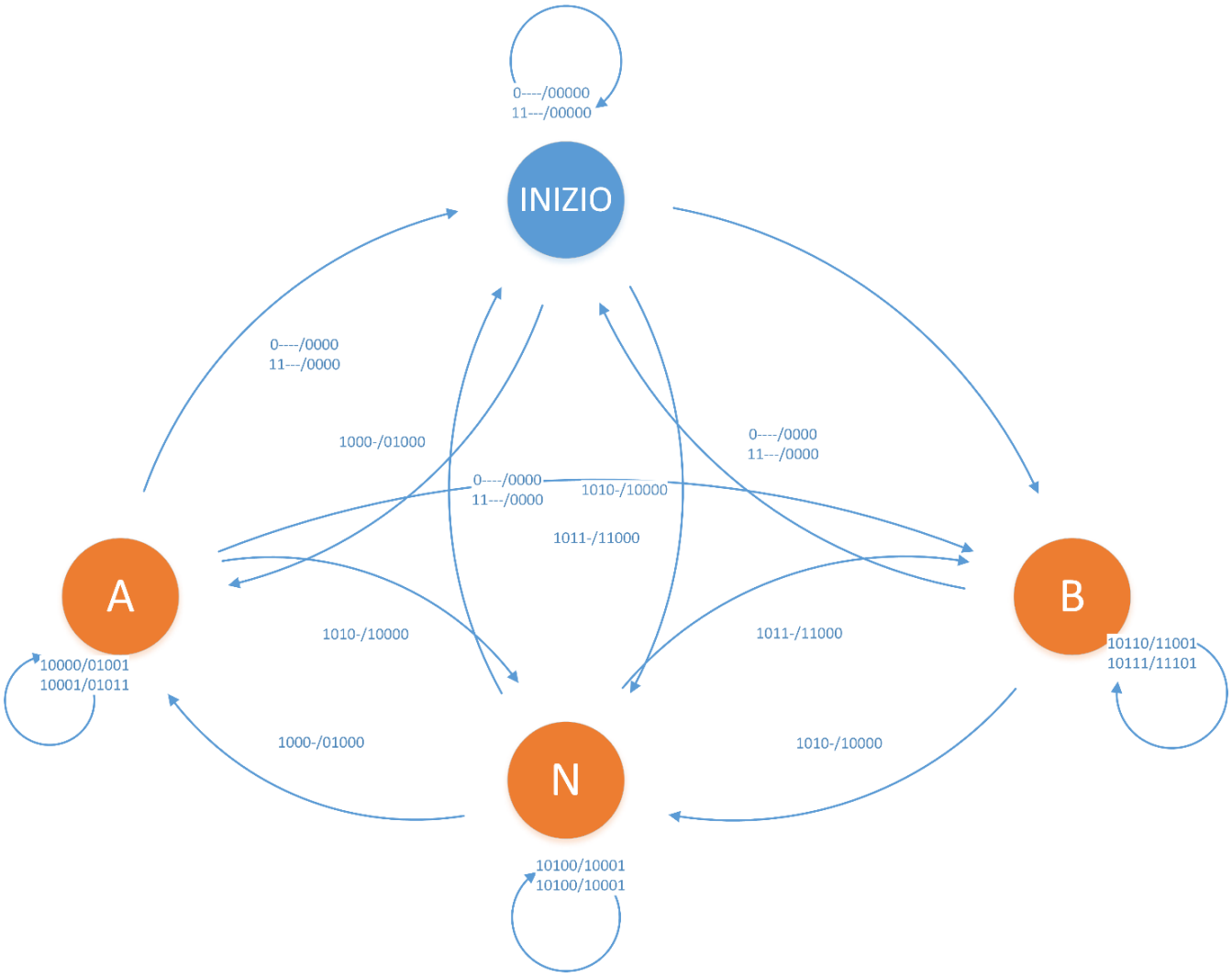
CONTROLLORE

Il controllore si occuperà di portare sempre la soluzione dell'impianto chimico ad un pH neutro. In caso di un rilevamento del pH < 60 (acido) o pH > 80 (basico) per 5 cicli di clock consecutivi, il controllore aprendo le valvole riporta la soluzione ad un pH neutro mantenendo così la soluzione sempre stabile.

Il controllore è così strutturato:

I = { INIT, RESET, SS1, SS0, FC }

O = { ST1, ST0, VLV1, VLV0, START }



Descrizione ingressi:

- INIT : è il bit che indica se il circuito è acceso (*1*) o se è spento (*0*) restituendo un valore *0* su tutti gli output;
- RESET : è il bit che indica se il circuito va resettato (*1*) restituendo un valore *0* su tutti gli output;
- SS1 / SS0 : sono i bit che indicano quale è lo stato di ingresso, che può essere acido (*A-00*), basico (*B-11*) e neutro (*N-10*) provenienti dal datapath;
- FC : è il bit che indica il fine clock, ossia il datapath restituisce (*1 costante*) quando ha contato 6 cicli di clock nello stesso stato. Questo bit serve per far dire al controllore di aprire una valvola per riportare lo stato del pH a neutro.

Descrizione uscite:

- ST1 / ST0 : sono i bit che indicano qual è lo stato corrente del pH, che può essere acido (*A-00*), basico (*B-11*) e neutro (*N-10*);
- VLV1 / VLV0 : sono i bit che indicano quale valvola è stata aperta dopo che il controllore ha rilevato un pH acido o basico per 6 cicli di clock. I valori possono essere *AS* (*10*) e *BS* (*01*);
- START: è il bit che indica al datapath di iniziare a contare 5 cicli di clock (*1*), poiché il controllore è in un nuovo stato;

Descrizione stati:

- **INIZIO**: è lo stato iniziale che al primo ciclo di clock, in base all'input di *SS1* e *SS0* andrà nei seguenti stati:

  - Stato neutro (*N*) quando *SS1 / SS0* varranno *10*
  - Stato acido (*A*) quando *SS1 / SS0* varranno *00*
  - Stato basico (*B*) quando *SS1 / SS0* varranno *11*

  Inoltre:
  - Quando *INIT* varrà *0* lo stato non cambierà (quindi rimarrà nello stato *INIZIO*), mettendo in uscita tutti i valori di out a 0
  - Quando *RESET* varrà *1* lo stato non cambierà (quindi rimarrà nello stato *INIZIO*), mettendo in uscita tutti i valori di out a 0

  Ogni volta che lo stato viene cambiato, il valore di *START* in out viene messo a 1, per indicare al datapath di iniziare il conteggio dei 5 cicli di clock. Lo stato di *INIZIO* non apre mai le valvole.

- **N** *(neutro)* : è lo stato in cui il valore del pH è compreso tra 6 e 8. Lo stato N in base all'input di *SS1* e *SS0* andrà nei seguenti stati:

  - Stato neutro (*N*) quando *SS1 / SS0* varranno *10*
  - Stato acido (*A*) quando *SS1 / SS0* varranno *00*
  - Stato basico (*B*) quando *SS1 / SS0* varranno *11*
  - Stato inizio (*N*) quando *INIT* varrà *0* (dando come out tutti i bit a *0*) e quando *RESET* varrà *1* (dando come out tutti i bit a *0*).

  Inoltre:
  - Dopo che sono passati 5 cicli di clock in questo medesimo stato, al 6 ciclo di clock quando il datapath setterà il valore di *FC* a *1*, lo stato non aprirà alcuna valvola poiché è già nel pH neutro e quindi vuol dire che è stabile.

- **A** *(acido)* : è lo stato in cui il valore del pH è minore di 60. Lo stato A in base all'input di *SS1* e *SS0* andrà nei seguenti stati:

  - Stato neutro (*N*) quando *SS1 / SS0* varranno *10*
  - Stato acido (*A*) quando *SS1 / SS0* varranno *00*
  - Stato basico (*B*) quando *SS1 / SS0* varranno *11*
  - Stato inizio (*N*) quando *INIT* varrà *0* (dando come out tutti i bit a *0*) e quando *RESET* varrà *1* (dando come out tutti i bit a *0*).

  Inoltre:
  - Dopo che sono passati 5 cicli di clock in questo medesimo stato, al 6 ciclo di clock quando il datapath setterà il valore di *FC* a *1*, lo stato aprirà una valvola *BS* (*01*).

- **B** *(basico)* : è lo stato in cui il valore del pH è maggiore di 80. Lo stato B in base all'input di *SS1* e *SS0* andrà nei seguenti stati:

  - Stato neutro (*N*) quando *SS1 / SS0* varranno *10*
  - Stato acido (*A*) quando *SS1 / SS0* varranno *00*
  - Stato basico (*B*) quando *SS1 / SS0* varranno *11*
  - Stato inizio (*N*) quando *INIT* varrà *0* (dando come out tutti i bit a *0*) e quando *RESET* varrà *1* (dando come out tutti i bit a *0*).

  Inoltre:
  - Dopo che sono passati 5 cicli di clock in questo medesimo stato, al 6 ciclo di clock quando il datapath setterà il valore di *FC* a *1*, lo stato aprirà una valvola *AS* (*01*).

Realizzazione FSM con SIS:

Per la realizzazione del controllore con SIS sono stati creati due file:

- *CONTROLLO.blif* : contiene la realizzazione della FSM nel linguaggio SIS. Per poter testare il controllore con l'ELABORATORE.*blif* (datapath) nella FSMD (*FSMD.blif*) è necessario assegnare gli stati. Abbiamo utilizzato il comando "*state_assign_jedi*", il quale fa un'assegnazione automatica degli stati al controllore, in modo anche da poter massimizzare la minimizzazione.

- *CONTROLLOJEDI.blif* : contiene la realizzazione della FSM pronta a comunicare con l'FSMD (*FSMD.blif*)

TEST

Test Datapath:

Simuliamo il datapath con ingressi del pH validi al limite degli intervalli e non validi per verificarne il corretto funzionamento:

```
sis> simulate 0 0 1 1 1 0 1 1 0          #pH Acido
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0           #00 (Acido)
Next state: 00000000


sis> simulate 0 0 1 1 1 1 0 0 0          #pH Neutro
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 1 0 0           #10 (Neutro)
Next state: 00000000


sis> simulate 0 1 0 1 0 0 0 0 0          #pH Neutro
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 1 0 0           #00 (Neutro)
Next state: 00000000


sis> simulate 0 1 0 1 0 0 0 1 0          #pH Basico
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 1 1 0           #11 (Basico)
Next state: 00000000

sis> simulate 0 1 1 1 1 1 1 1 1          #pH Maggiore di 14 (140)
Network simulation:
Outputs: 0 0 0 0 0 0 0 1 0 1 0           #01 Valore di errore
Next state: 00000001
```

Test FSM:

Simuliamo l'FSM con ingressi che simulino l'accensione, reset e un cambio di stato con apertura della valvola:

```
sis> simulate 0 1 0 0 0          #Accensione della FSM
Network simulation:
Outputs: 0 0 0 0 0
Next state:
STG simulation:
Outputs: 0 0 0 0 0
Next state: INIZIO ((null))

sis> simulate 1 1 0 0 0          #Reset della FSM
Network simulation:
Outputs: 0 0 0 0 0
Next state:
STG simulation:
Outputs: 0 0 0 0 0
Next state: INIZIO ((null))

sis> simulate 1 0 0 0 0          #SS1 e SS0 a valori '00' (Acido)
Network simulation:
Outputs: 0 0 0 0 0
Next state:
STG simulation:
Outputs: 0 1 0 0 0
Next state: A ((null))

sis> simulate 1 0 0 0 1          #FC settato a 1 per far aprire la valvola
Network simulation:
Outputs: 0 0 0 0 0
Next state:
STG simulation:
Outputs: 0 1 0 1 1               #Valvola BS-01 aperta
Next state: A ((null))

sis> simulate 1 1 0 0 0          #Reset della FSM
Network simulation:
Outputs: 0 0 0 0 0
Next state:
STG simulation:
Outputs: 0 0 0 0 0
Next state: INIZIO ((null))
```

Test FSMD:

```
sis> simulate 0 0 0 0 0 0 0 0 0 0          #Accensione della FSMD
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 0100000000

sis> simulate 1 0 0 0 1 1 1 0 0 0          #Test pH Acido
Network simulation:
Outputs: 0 1 0 0 0 0 0 0 0 0 0 0
Next state: 1100000000

sis> simulate 1 0 0 1 1 1 1 0 0 0          #Test pH Basico
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 0 0 0 0
Next state: 0000000000

sis> simulate 1 0 0 1 1 1 1 0 0 0          #Test conteggio cicli di clock
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 0 1 0 0
Next state: 0000000001

sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 1 0 0 0
Next state: 0000000010

sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 1 1 0 0
Next state: 0000000011

sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 1 0 0 0 0
Next state: 0000000100

sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 0 1 0 1 1 0          #Apertura valvola
Next state: 0000000101
```

```
sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 1 1 0 1 0
Next state: 0000000110

sis> simulate 1 0 0 1 1 1 1 0 0 0
Network simulation:
Outputs: 1 1 0 0 0 0 0 1 1 1 1 0
Next state: 0000000111

sis> simulate 1 1 0 1 1 1 1 0 0 0                    #Reset Finale
Network simulation:
Outputs: 0 0 0 0 0 0 0 0 0 0 0 0
Next state: 0100000000
sis>
```

OTTIMIZZAZIONE

Il circuito ottenuto va ottimizzato, cioè si deve ridurre il numero di componenti e/o il ritardo senza però alterarne il funzionamento. Per ottimizzare i circuiti SIS fornisce vari comandi come *sweep*, *eliminate*, *resub*, *fx*, *simplify* e *full_simplify*: nella maggior parte dei casi lo script '*script.rugged*' che utilizza tutte queste funzioni, permette di ottenere i risultati migliori. Verrà utilizzato inoltre il comando '*print_stats*' dopo ogni modifica per monitorare le statistiche del circuito. Prima di ottimizzare il circuito interamente, eseguiremo le operazioni di minimizzazione prima sul datapath e poi sul controllore perché abbiamo constatato che il circuito viene minimizzato ulteriormente.

Ottimizzazione CONTROLLOJEDI.blif:

Prima di iniziare la vera e propria ottimizzazione controlliamo tramite il comando di SIS '*print_stats*' le statistiche del nostro circuito :

```
sis> rl CONTROLLOJEDI.blif
sis> ps
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2                  #Statistiche non ottimizzato
lits(sop)=  59   lits(fac)=  45   #states(STG)=   4
```

Eseguiamo quindi ora lo script '*script.rugged*' per la minimizzazione:

```
sis> source script.rugged                          #Script per l'ottimizzazione
```

A questo punto ricontrolliamo le statistiche e possiamo notare che il nostro circuito è stato migliorato principalmente sotto i valori di lits(sop) e lits(fac) :

```
sis> ps
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2                  #Statistiche Ottimizzato
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
```

Mostriamo quindi di seguito tutto il processo di ottimizzazione:

```
sis> rl CONTROLLOJEDI.blif
sis> ps
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2          #Statistiche non ottimizzato
lits(sop)=  59   lits(fac)=  45   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  59   lits(fac)=  45   #states(STG)=   4
sis> source script.rugged
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  59   lits(fac)=  45   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  59   lits(fac)=  45   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  8   latches= 2
lits(sop)=  32   lits(fac)=  32   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  8   latches= 2
lits(sop)=  32   lits(fac)=  32   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  8   latches= 2
lits(sop)=  32   lits(fac)=  32   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
CONTROLLO       pi= 5   po= 5   nodes=  7   latches= 2          #Statistiche Ottimizzato
lits(sop)=  33   lits(fac)=  31   #states(STG)=   4
```

Ottimizzazione ELABORAZIONE.blif:

Prima di iniziare la vera e propria ottimizzazione controlliamo tramite il comando di SIS '*print_stats*' le statistiche del nostro circuito :

```
sis> rl ELABORAZIONE.blif
sis> ps
ELABORAZIONE     pi= 9    po=11    nodes= 95    latches= 8        #Statistiche non ottimizzato
lits(sop)= 456    lits(fac)= 364
```

Eseguiamo quindi ora lo script '*script.rugged*' per la minimizzazione:

```
sis> source script.rugged                          #Script per l'ottimizzazione
```

A questo punto ricontrolliamo le statistiche e possiamo notare che il nostro circuito è stato notevolmente migliorato sotto quasi tutti gli aspetti:

```
sis> ps
ELABORAZIONE     pi= 9    po=11    nodes= 15    latches= 8        #Statistiche Ottimizzato
lits(sop)=  98    lits(fac)=  81
```

Mostriamo quindi di seguito tutto il processo di ottimizzazione:

```
sis> rl ELABORAZIONE.blif
sis> set autoexec ps
ELABORAZIONE     pi= 9   po=11   nodes= 95   latches= 8
lits(sop)= 456   lits(fac)= 364
sis> source script.rugged
ELABORAZIONE     pi= 9   po=11   nodes= 27   latches= 8
lits(sop)= 145   lits(fac)= 105
ELABORAZIONE     pi= 9   po=11   nodes= 18   latches= 8
lits(sop)= 150   lits(fac)=  96
ELABORAZIONE     pi= 9   po=11   nodes= 18   latches= 8
lits(sop)=  99   lits(fac)=  78
ELABORAZIONE     pi= 9   po=11   nodes= 17   latches= 8
lits(sop)=  99   lits(fac)=  77
ELABORAZIONE     pi= 9   po=11   nodes= 17   latches= 8
lits(sop)=  99   lits(fac)=  77
ELABORAZIONE     pi= 9   po=11   nodes= 12   latches= 8
lits(sop)= 128   lits(fac)=  88
ELABORAZIONE     pi= 9   po=11   nodes= 12   latches= 8
lits(sop)= 107   lits(fac)=  80
ELABORAZIONE     pi= 9   po=11   nodes= 12   latches= 8
lits(sop)= 106   lits(fac)=  81
ELABORAZIONE     pi= 9   po=11   nodes= 19   latches= 8
lits(sop)=  92   lits(fac)=  85
ELABORAZIONE     pi= 9   po=11   nodes= 19   latches= 8
lits(sop)=  92   lits(fac)=  85
ELABORAZIONE     pi= 9   po=11   nodes= 19   latches= 8
lits(sop)=  92   lits(fac)=  85
ELABORAZIONE     pi= 9   po=11   nodes= 15   latches= 8
lits(sop)=  98   lits(fac)=  81
ELABORAZIONE     pi= 9   po=11   nodes= 15   latches= 8
lits(sop)=  98   lits(fac)=  81
ELABORAZIONE     pi= 9   po=11   nodes= 15   latches= 8
lits(sop)=  98   lits(fac)=  81
ELABORAZIONE     pi= 9   po=11   nodes= 15   latches= 8
lits(sop)=  98   lits(fac)=  81
sis>
```

Ottimizzazione FSMD.blif:

Prima di iniziare la vera e propria ottimizzazione controlliamo tramite il comando di SIS '*print_stats*' le statistiche del nostro circuito :

```
sis> rl FSMD.blif
sis> ps
FSMD          pi=10   po=12   nodes= 22   latches=10          #Statistiche non ottimizzato
lits(sop)= 131   lits(fac)= 112
```

Eseguiamo quindi ora lo script '*script.rugged*' per la minimizzazione:

```
sis> source script.rugged                          #Script per l'ottimizzazione
```

A questo punto ricontrolliamo le statistiche e possiamo notare che il nostro circuito è stato notevolmente migliorato sotto quasi tutti gli aspetti:

```
sis> ps
FSMD          pi=10   po=12   nodes= 20   latches=10          #Statistiche Ottimizzato
lits(sop)= 132   lits(fac)= 111
```

Mostriamo quindi di seguito tutto il processo di ottimizzazione:

```
sis> rl FSMD.blif
FSMD          pi=10   po=12   nodes= 22   latches=10
lits(sop)= 131   lits(fac)= 112
sis> source script.rugged
FSMD          pi=10   po=12   nodes= 22   latches=10
lits(sop)= 131   lits(fac)= 112
FSMD          pi=10   po=12   nodes= 21   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 21   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 21   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 21   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 17   latches=10
lits(sop)= 182   lits(fac)= 115
FSMD          pi=10   po=12   nodes= 17   latches=10
lits(sop)= 182   lits(fac)= 115
FSMD          pi=10   po=12   nodes= 17   latches=10
lits(sop)= 182   lits(fac)= 115
FSMD          pi=10   po=12   nodes= 29   latches=10
lits(sop)= 126   lits(fac)= 120
FSMD          pi=10   po=12   nodes= 29   latches=10
lits(sop)= 126   lits(fac)= 120
FSMD          pi=10   po=12   nodes= 29   latches=10
lits(sop)= 126   lits(fac)= 120
FSMD          pi=10   po=12   nodes= 20   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 20   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 20   latches=10
lits(sop)= 132   lits(fac)= 111
FSMD          pi=10   po=12   nodes= 20   latches=10
lits(sop)= 132   lits(fac)= 111
```

MAPPING TECNOLOGICO

Nell'ultima fase del progetto è possibile mappare il circuito con una libreria di SIS chiamata *synch.genlib* come è richiesto dalla specifica. Tale libreria consente di mappare il circuito sia per area che per ritardo, ma la specifica indica di mappare per area.

Mpping tecnologico con SIS:

Le istruzioni necessarie a eseguire il mapping tecnologico del circuito per area sono le seguenti:

```
sis> read_library synch.genlib          #Caricamento della libreria
sis> map -W -m 0 -s                     #Mapping tecnologico del circuito
```

Mapping tecnologico della FSMD con SIS:

Eseguiamo quindi la mappatura del circuito con SIS. Nel nostro caso, dopo i vari test, abbiamo rilevato una sequenza di comandi di ottimizzazione di SIS che ci ha permesso di ottimizzare il circuito nel miglior modo possibile per ricavare la minor area con il mapping:

```
sis> source script.rugged
sis> eliminate -1
sis> fx
sis> eliminate -500
```

Dopo il mapping tecnologico il circuito ha un'area finale di **2696.0** e un ritardo massimo di **30.0**.

Mostriamo quindi di seguito tutto il processo di mapping tecnologico:

```
sis> rl FSMD.blif                                   # Leggo il file .blif
sis> source script.rugged
sis> eliminate -1
sis> fx
sis> read_library synch.genlib                      # Carico la libreria synch.genlib
sis> eliminate -500
sis> map -m 0 -s                                    # Mappo il circuito per area (-m 0)
>>> before removing serial inverters <<<
# of outputs:        22
total gate area:      2920.00
maximum arrival time: (30.00,30.00)
maximum po slack:     (-8.80,-8.80)
minimum po slack:     (-30.00,-30.00)
total neg slack:      (-406.00,-406.00)
# of failing outputs:  22
>>> before removing parallel inverters <<<
# of outputs:        22
total gate area:      2920.00
maximum arrival time: (30.00,30.00)
maximum po slack:     (-8.80,-8.80)
minimum po slack:     (-30.00,-30.00)
total neg slack:      (-406.00,-406.00)
# of failing outputs:  22
# of outputs:        22
total gate area:      2696.00
maximum arrival time: (30.00,30.00)
maximum po slack:     (-8.20,-8.20)
minimum po slack:     (-30.00,-30.00)
total neg slack:      (-404.60,-404.60)
# of failing outputs:  22
sis>
sis> print_map_stats                                # Statistiche del mapping
Total Area          = 2696.00
Gate Count          = 76
Buffer Count        = 10
Inverter Count              = 22
Most Negative Slack  = -30.00
Sum of Negative Slacks      = -404.60
Number of Critical PO= 22
```