Dipartimento di informatica

Corso di laurea triennale in informatica

Elaborato SIS - Laboratorio Architettura degli Elaboratori

Davide Cerullo (VR471650) Edoardo Bazzotti (VR472656)
01/02/2022

INDICE

Specifica del circuito

Si progetti il circuito sequenziale che controlla un macchinario chimico il cui scopo è portare una soluzione iniziale a pH noto, ad un pH di neutralità. Il valore del pH viene espresso in valori compresi tra 0 e 14.
Il circuito controlla due valvole di erogazione: una di soluzione acida e una di soluzione basica.
Se la soluzione iniziale è acida, il circuito dovrà procedere all'erogazione della soluzione basica fintanto che la soluzione finale non raggiunga la soglia di neutralità (pH compreso tra 7 e 8).
Analogamente, se la soluzione iniziale è basica, il circuito procederà all'erogazione di soluzione acida fino al raggiungimento della soglia di neutralità.
Per pH acido si intende un valore strettamente inferiore a 7, mentre per basico si intende una soluzione con pH strettamente maggiore a 8.
Il pH viene codificato in fixed-point, con 4 bit riservati per la parte intera e gli altri per la parte decimale.
Le due valvole hanno flussi differenti di erogazione.
La valvola relativa alla soluzione basica eroga una quantità di soluzione che permette di alzare il pH della iniziale di 0.25 ogni ciclo di clock.
La valvola relativa alla soluzione acida eroga una quantità di soluzione che permette di abbassare il pH della soluzione iniziale di 0.5 ogni ciclo di clock.
Il circuito ha 3 ingressi nel seguente ordine:

- RST (1 bit) • START(1 bit)
- pH (8 bit, 4 parte intera e 4 per la parte decimale) Gli output sono i seguenti e devono seguire il seguente ordine:
- FINE_OPERAZIONE (1 bit)
- ERRORE_SENSORE (1 bit)
- VALVOLA_ACIDO (1 bit)
- VALVOLA_BASICO (1 bit)
- PH_FINALE (8 bit)
- NCLK (8 bit)

Input e output devono essere definiti nell'ordine sopra specificato (da sinistra verso destra). Le porte con più bit devono essere descritte utilizzando la codifica con il bit più significativo a sinistra.
Il meccanismo è guidato come segue:

- Quando il segnale RST viene alzato, il sistema torna da un qualsiasi stato allo stato di Reset, mettendo tutte le porte in output a zero.
- Per procedere, Il sistema riceve in input il segnale di START, con valore 1, e il segnale del pH iniziale per un solo ciclo di clock. Il sistema potrà quindi procedere con la fase di elaborazione.
- Se la soluzione iniziale è acida, viene aperta la valvola della soluzione basica, mettendo a 1 il relativo output. Analogamente, se la soluzione iniziale è basica, viene aperta la valvola della soluzione acida mettendo a 1 la porta VALVOLA_ACIDO.
- Il sistema mantiene aperte le valvole per il tempo necessario al raggiungimento della soglia di neutralità (calcolata dal sistema).
- Una volta terminata l'operazione, il sistema deve chiudere tutte le valvole aperte, riportare il pH finale sulla porta in output PH_FINALE e alzare la porta di FINE_OPERAZIONE.
- La porta NCLK riporta quanti cicli di clock sono stati necessari per portare la soluzione a neutralità.
- Se il valore del pH non è valido (> 14) il sistema deve riportare l'errore alzando l'output ERRORE_SENSORE
  Lo schema generale del circuito deve rispettare la FSMD riportata di seguito:

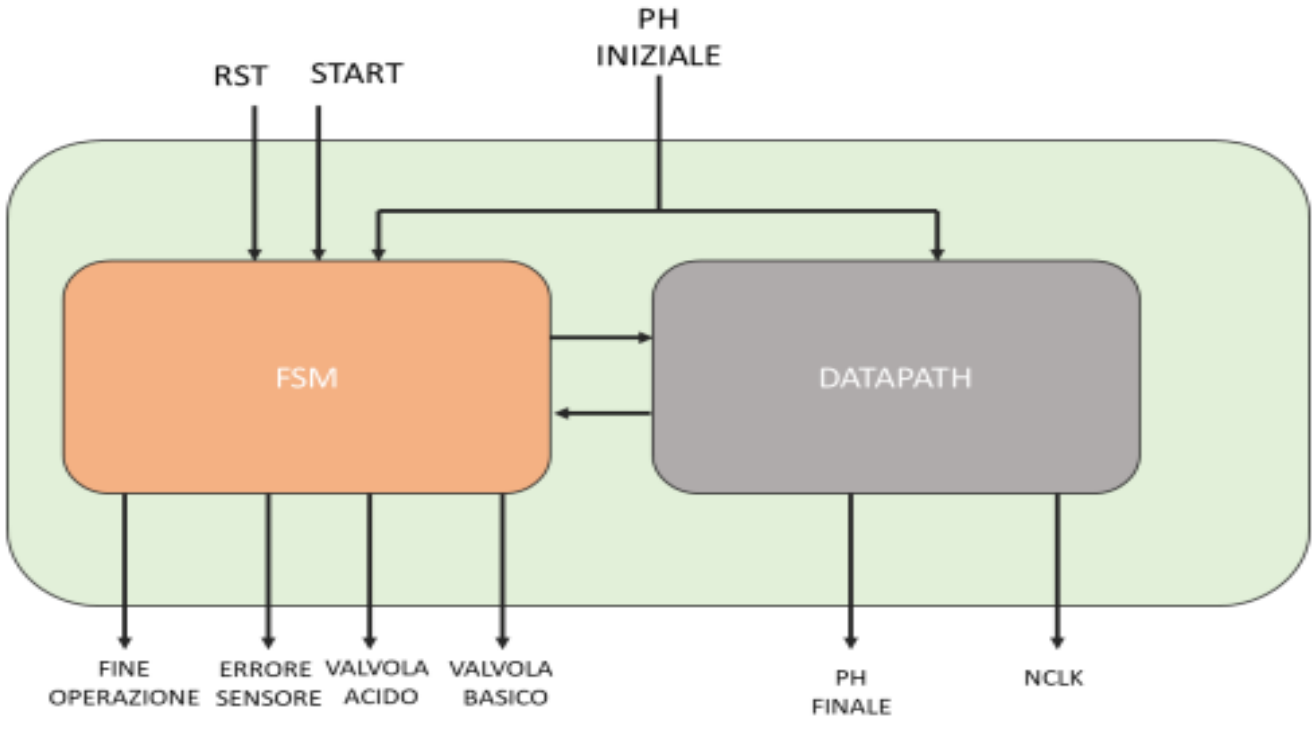


Diagramma degli stati del controllore

Abbiamo realizzato una FSM a 4 stati (RESET, ACIDO, BASICO, FINE), che ha come input: Rst, Start e PH iniziale; le uscite sono: Fine Operazione, Errore Sensore, Valvola Acido, Valvola Basico; abbiamo deciso anche di implementare 6 segnali di controllo (in grado di comunicare tra FSM e datapath): Segnale Inizio, Segnale Fine, Segnale Giusto, Reset, Valvola Acido, Valvola Basico.

Descrizione degli stati:

- RESET: È lo stato iniziale da cui parte il circuito, per procedere in uno degli altri stati è necessario che il valore si Start sia ad 1 e di Reset a 0, inoltre c'è un controllo sul PH inserito per evitare che sia maggiore di 14, se così dovesse essere il programma resta nello stato di RESET e restituisce in output 1 sull'Errore Sensore; nel caso in cui il PH inserito sia valido, lo stato di RESET controlla:
    - Se il valore del PH è maggiore di 8, passa allo stato BASICO e apre la ValvolaAcido ponendola ad 1.
    - Se minore di 7, passa allo stato ACIDO e apre la ValvolaBasico ponendola ad 1.
    - Compreso tra 7 e 8 va in BASICO ma non apre nessuna valvola, perché deve passare soltanto per restituire il PH finale ed il clock (gli output del datapath).

- BASICO: La FSM arriva in questo stato quando è stata inserita una soluzione basica (maggiore di 8) o neutra (caso del PH iniziale già neutro), attivando il datapath il quale avendo già il PH_INIZIALE, riceve da parte dell'FSM il valore della valvola aperta (in questo caso Valvola Basico).
  Dallo stato BASICO possiamo andare allo stato:
    - FINE: Se il Segnale Fine restituito dal Datapath vale 1, quindi il PH è diventato neutro (compreso tra 7 ed 8 inclusi).
    - RESET: Se il valore di Reset viene messo ad 1, vengono posti a 0 tutti gli output, a meno che il PH_INIZIALE non sia stato già neutro, in questo caso passiamo allo stato di FINE (in modo da dare l'output del PH).
    - BASICO: Se il PH risulta ancora basico (maggiore di 8).

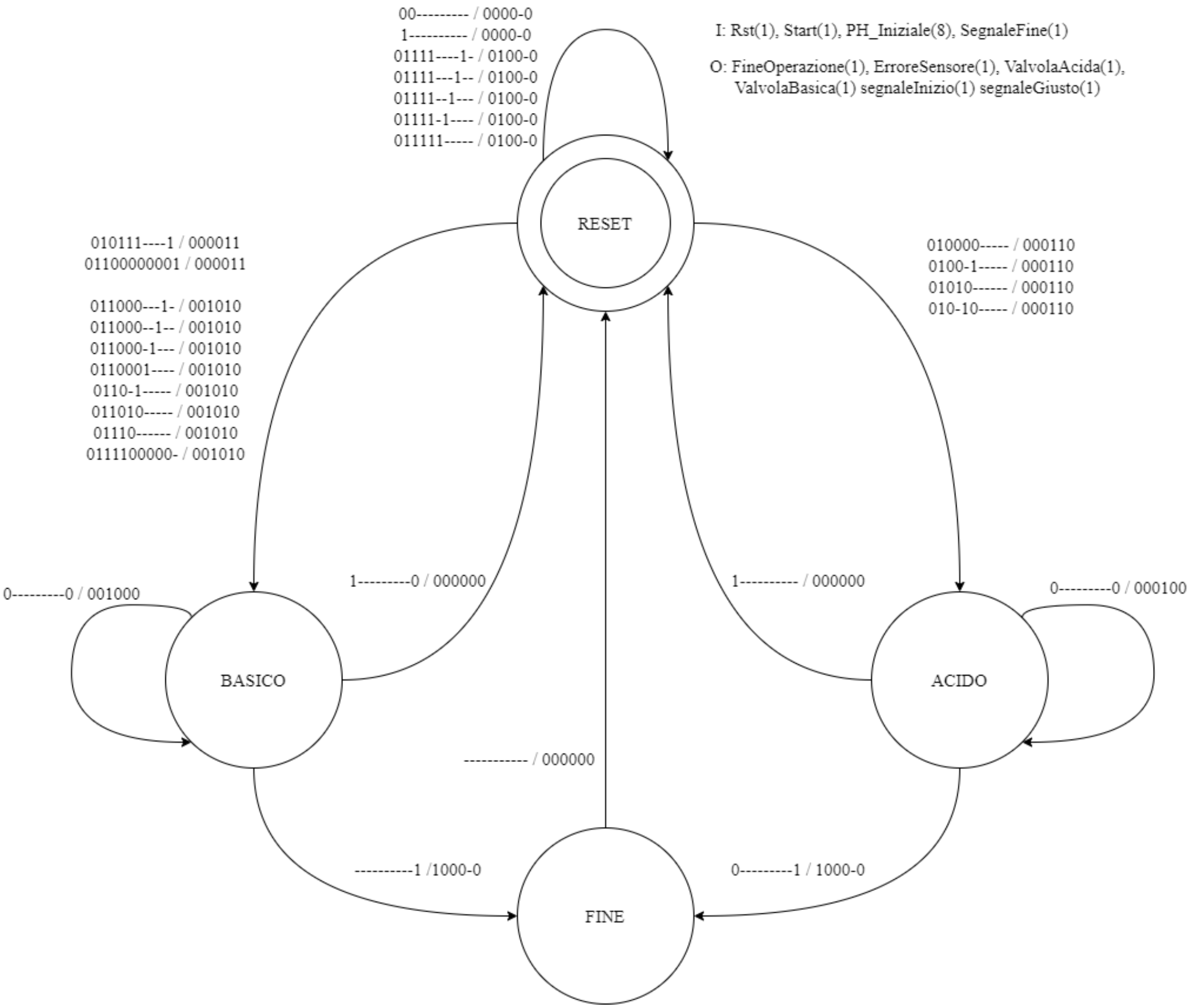
- ACIDO: La FSM arriva in questo stato quando è stata inserita una soluzione acida (minore di 7), attivando il datapath il quale avendo già il PH_INIZIALE, riceve da parte dell'FSM il valore della valvola aperta (in questo caso Valvola Basico).
  Dallo stato ACIDO possiamo andare allo stato:
    - FINE: Se il Segnale Fine restituito dal Datapath vale 1, quindi il PH è diventato neutro (compreso tra 7 ed 8 inclusi).
    - RESET: Se il valore di Reset viene messo ad 1, vengono posti a 0 tutti gli output.
    - ACIDO: Se il PH risulta ancora acido (minore di 7).

- FINE: È lo stato finale in cui termina il circuito, da questo stato passiamo allo stato di RESET, il quale attenderà un nuovo PH.

Descrizione degli output:

- Fine Operazione: viene impostato ad 1 quando il valore del PH rientra nei range e quindi si può passare allo stato di FINE.
- Errore Sensore: Viene impostato ad 1 quando il PH INIZIALE inserito dall'utente è maggiore di 14
- Valvola Acido: viene impostata ad 1, quando siamo ci troviamo nello stato BASICO ed è necessario aggiungere soluzione acida
- Valvola Basico: Viene impostata ad 1 quando ci troviamo nello stato ACIDO ed è necessario aggiungere soluzione basica.

Immagine diagramma degli stati



Datapath

Il Datapath è la parte del circuito che si occupa di aprire e chiudere le valvole (Valvola Acido, Valvola Basico), in modo da modificare il valore del PH per far si che raggiunga la zona di neutralità (maggiore di 7 e minore di 8, compresi).

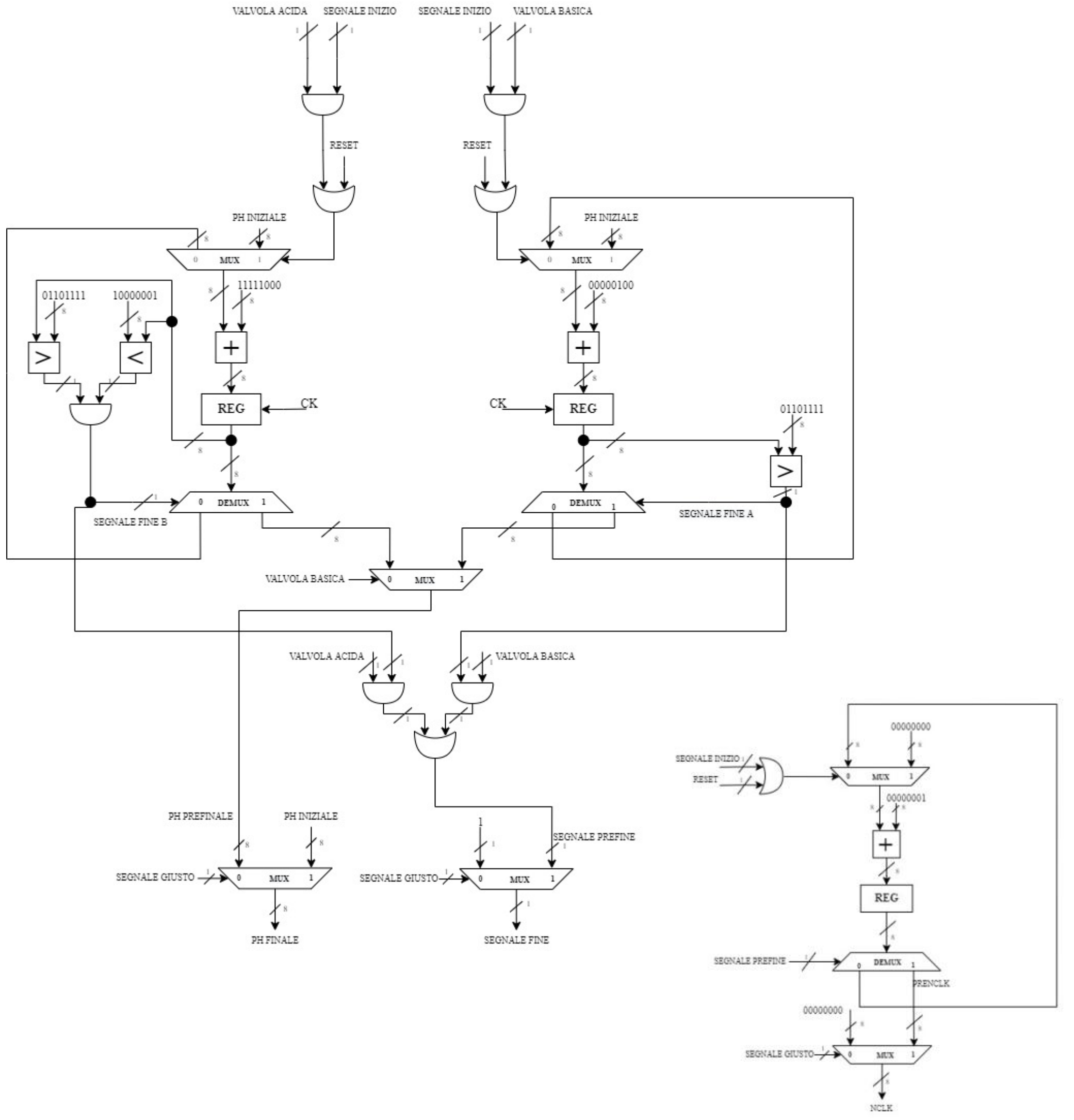
Per poter funzionare il Datapath ha bisogno di:

- PH INIZIALE: 8 bit, fornito dall'utente
- Segnale Inizio: fornito dall'FSM, serve per far partire il datapath (primo ciclo), nei cicli successivi verrà messo a 0
- Segnale Giusto: fornito dall'FSM, serve per comunicare al datapath che il PH INIZIALE inserito è già nei range, quindi non deve modificarne i valori, ma deve limitarsi solo a mandarli in output
- Rst: fornito dall'FSM, serve per bloccare il datapath dall'esecuzione quando viene impostato il segnale di Reset ad 1

- Valvola Acido: fornito dall'FSM, viene impostata ad 1, quando siamo in presenza di una PH basico ed è necessario aggiungere soluzione acida
- Valvola Basico: fornito dall'FSM, viene impostata ad 1, quando siamo in presenza di un PH acido ed è necessario aggiungere soluzione basica

Gli Output che restituisce il Datapath sono:

- PH FINALE: è il PH risultante dopo varie operazioni che hanno permesso di raggiungere lo stato di neutralità
- NCLK: parametro che indica il numero di cicli che sono stati necessari per portare il PH INIZIALE in una situazione di neutralità



Statistiche del circuito

Vengono ora descritte le statistiche del circuito prima e dopo la miglior ottimizzazione ottenuta.

FSM

Prima dell'ottimizzazione:

```
sis> rl fsm.blif
sis> print_stats
fsm             pi=11   po= 6   nodes=  6       latches= 0
lits(sop)=   0  #states(STG)=   4
sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> print_stats
fsm             pi=11   po= 6   nodes=  8       latches= 2
lits(sop)= 363  #states(STG)=   4
```

Dopo l'ottimizzazione:

```
sis> print_stats
fsm             pi=11   po= 6   nodes= 13       latches= 2
lits(sop)=  50  #states(STG)=   4
```

Dopo varie prove e l'utilizzo di vari script siamo riusciti ad arrivare ad avere questa ottimizzazione, la quale sembra essere la migliore come compromesso tra numero di nodi e di letterali. Si può notare un notevole miglioramento nel numero di letterali, a scapito di un aumento nel numero dei nodi.

Datapath

Prima dell'ottimizzazione:

```
sis> rl datapath.blif
Warning: network `datapath', node "printRipA" does not fanout
Warning: network `datapath', node "printRipB" does not fanout
Warning: network `datapath', node "printRip" does not fanout
sis> print_stats
datapath        pi=13   po=17   nodes=189       latches=37
lits(sop)= 706
```

L'ottimizzazione è stata eseguita in modo tale da ridurre il più possibile l'area del circuito (come da richiesta).
Dopo l'ottimizzazione:

```
sis> print_stats
datapath        pi=13   po=17   nodes= 64       latches=37
lits(sop)= 305
```

FSMD

Prima dell'ottimizzazione:

```
sis> rl FSMD.blif
sis> print_stats
FSMD            pi=10   po=20   nodes= 77       latches=39
lits(sop)= 355
```

Dopo l'ottimizzazione:

```
sis> print_stats
FSMD            pi=10   po=20   nodes= 73       latches=39
lits(sop)= 359
```

L'ottimizzazione è stata quasi nulla in quanto l'fsmd è l'unione dell'fsm e del datapath già ottimizzati.

Dopo vari tentativi abbiamo trovato questa ottimizzazione la quale ci è sembrata la migliore, aumento dei letterali e diminuzione dei nodi.

Mapping

Una volta ottimizzato il circuito, abbiamo provveduto a mapparlo, cercando sempre una riduzione per area (map -m 0); la libreria utilizzata p “synch.genlib”.
Di seguito sono riportate 2 immagini, la prima è una mappatura senza ottimizzazione, la seconda invece con l’ottimizzazione.

Mapping pre ottimizzazione:

```
sis> read_library synch.genlib
sis> map -s
warning: unknown latch type at node '{[8085]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[8086]}' (RISING_EDGE assumed)
WARNING: uses as primary input drive the value (0.20,0.20)
WARNING: uses as primary input arrival the value (0.00,0.00)
WARNING: uses as primary input max load limit the value (999.00)
WARNING: uses as primary output required the value (0.00,0.00)
WARNING: uses as primary output load the value 1.00
>>> before removing serial inverters <<<
# of outputs:          59
total gate area:       9936.00
maximum arrival time: (28.00,28.00)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-28.00,-28.00)
total neg slack:      (-886.00,-886.00)
# of failing outputs:  59
>>> before removing parallel inverters <<<
# of outputs:          59
total gate area:       9728.00
maximum arrival time: (28.00,28.00)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-28.00,-28.00)
total neg slack:      (-868.20,-868.20)
# of failing outputs: 59
# of outputs:          59
total gate area:       9472.00
maximum arrival time: (26.60,26.60)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-26.60,-26.60)
total neg slack:      (-822.60,-822.60)
# of failing outputs:  59
```

Mapping post ottimizzazione:

```
sis> read_library synch.genlib
sis> map -m 0 -s
warning: unknown latch type at node '{v13.0}' (RISING_EDGE assumed)
warning: unknown latch type at node '{v13.1}' (RISING_EDGE assumed)
WARNING: uses as primary input drive the value (0.20,0.20)
WARNING: uses as primary input arrival the value (0.00,0.00)
WARNING: uses as primary input max load limit the value (999.00)
WARNING: uses as primary output required the value (0.00,0.00)
WARNING: uses as primary output load the value 1.00
>>> before removing serial inverters <<<
# of outputs:          59
total gate area:       9216.00
maximum arrival time: (26.60,26.60)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-26.60,-26.60)
total neg slack:      (-748.20,-748.20)
# of failing outputs:  59
>>> before removing parallel inverters <<<
# of outputs:          59
total gate area:       9072.00
maximum arrival time: (26.60,26.60)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-26.60,-26.60)
total neg slack:      (-740.20,-740.20)
# of failing outputs:  59
# of outputs:          59
total gate area:       8656.00
maximum arrival time: (25.40,25.40)
maximum po slack:     (-1.60,-1.60)
minimum po slack:     (-25.40,-25.40)
total neg slack:      (-726.80,-726.80)
# of failing outputs:  59
```

Scelte progettuali

Scelte progettuali durante l'implementazione del nostro progetto:

- Segnale Giusto
  Per far si dare in output il PH INIZIALE inalterato, nel caso in cui il suo valore fosse già compreso nel range di neutralità (7-8), abbiamo deciso di implementare un segnale di controllo tra FSM e Datapath, il "Segnale Giusto", il suo valore viene impostato ad 1 quando il PH INIZIALE è già nei range al momento dell'inserimento del valore da parte dell'utente, in questo caso l'FSM passa allo stato BASICO con Segnale Giusto ad 1, nello stato BASICO il datapath rileva il segnale giusto ad 1 grazie a 2 multiplexer, il primo da in output Segnale Fine ad 1, così da terminare il circuito, mentre il secondo si occupa di dare in output il PH INIZIALE, impostandolo come PH FINALE.

- Sommatore8Bit

```
.model sommatore8Bit

#1num è il PH_iniziale, 2num è l'incremento che dobbiamo fare
.inputs 1num7 1num6 1num5 1num4 1num3 1num2 1num1 1num0 2num3 2num2 2num1 2num0 rip
.outputs out7 out6 out5 out4 out3 out2 out1 out0 printRip

#somma 1 e 2° cifre (escludendo le prime 2)
.subckt sommatore2Bit  1num1=1num3 1num0=1num2 2num1=2num3 2num0=2num2 rip=rip out0=out2 out1=out3 printRip=UseRip0

#somma 3 e 4° cifre
.subckt sommatore2Bit  1num1=1num5 1num0=1num4 2num1=2num0 2num0=2num1 rip=UseRip0 out0=out4 out1=out5 printRip=UseRip1

#somma 5 e 6° (riuso 2num1 e 2num0 per passare il valore 0 alla somma)
.subckt sommatore2Bit  1num1=1num7 1num0=1num6 2num1=2num0 2num0=2num1 rip=UseRip1 out0=out6 out1=out7 printRip=printRip

#output per la cifra meno significativa del PH
.names 1num1 out1
1 1

#output per la seconda cifra meno significativa del PH
.names 1num0 out0
1 1

.search sommatore2Bit.blif
.end
```

Per implementare un sommatore ad 8 bit, abbiamo pensato di utilizzare 3 volte un sommatore a 2 bit, questo perché abbiamo deciso di ricevere in input 8 bit e 4 bit, non ne abbiamo chiesti 8 anche per il secondo numero perché quando si andava a sommare o sottrarre un valore al PH, non c'era la necessità di andare a toccare anche le ultime 2 cifre decimali; quindi i 4 bit richiesti vengono sommati come riportato di seguito:
10010100 PH
000010 Valore inserito in input da sommare: 1000
101010 Valore inserito in input da sommare: 1001
101011 Valore inserito in input da sommare: 1101
010132 Es valore inserito da sommare: 3210
Per sottrarre 0,5 al PH abbiamo utilizzato comunque il sommatore, il valore che abbiamo sommato è 1011.
Per eseguire la somma del NCLK abbiamo realizzato un sommatore 8 Bit che si occupasse solo della parte intera.

- Reset
  Implementando il nostro progetto abbiamo riscontrato un caso non presente nella specifica del progetto, ovvero quando una volta inserito in input un PH con valore già nel range, ed al clock successivo si imposta il Reset a 0, abbiamo deciso di dare comunque in output il PH inserito, perché dal punto di vista teorico dall'inserimento del PH INIZIALE (già nei range) il circuito dovrebbe restituire immediatamente il PH FINALE (nessun giro di clock), quindi non ci dovrebbe neanche essere l'occasione di impostare il segnale di Reset a 0.