UNIVERSITÀ di VERONA | Dipartimento di INFORMATICA

# Laboratorio di Architettura degli Elaboratori

## Elaborato **SIS**

## **A.A 2021/2022**


Studenti:

**Vittorio Maria Stano (VR457793)**

**Stefano Esposito (VR461049)**


23/01/2022

# INDICE GENERALE

## DESCRIZIONE GENERALE DEL PROGETTO

Il dispositivo da noi implementato simula il funzionamento di un 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 **pH 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 4 bit per la parte decimale.

Il sistema procede all'elaborazione del pH solo quando il segnale **START** viene portato a **1**. In qualsiasi momento, se il segnale **RST** viene alzato, il sistema si porta allo stato di Reset, ponendo tutte le porte di output a **0**. Le due valvole hanno flussi differenti di erogazione.

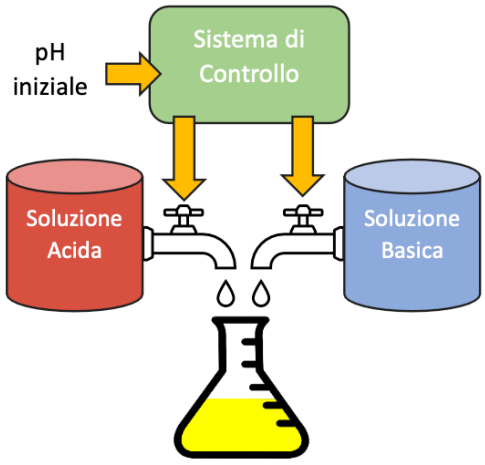
La valvola relativa alla soluzione **basica** eroga una quantità di soluzione che permette di **alzare** il pH della iniziale di **0.25 ogni ciclo di clock** mentre la valvola relativa alla soluzione **acida** eroga una quantità di soluzione che permette di **abbassare** il pH della soluzione iniziale di **0.5 ogni ciclo di clock**.

Il 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**.



## CONTROLLORE - FSM

Il circuito da noi implementato fa affidamento ad un controllore. Il controllore è una macchina a stati finiti (FSM) di Mealy che presenta 4 input e 5 output in totale:

**$I_{FSM}$ = {RST, START, PH_INIZIALE, pHOUT}**

**$O_{FSM}$ = {FINE_OPERAZIONE, ERRORE_SENSORE, VALVOLA_ACIDO, VALVOLA_BASICO, START01}**

Sono stati individuati due stati che vengono descritti di seguito:

- **RESET**: rappresenta lo stato iniziale del circuito. È lo stato di "attesa" del segnale **START** e del pH inserito dall'utente. In caso di pH non valido (> 14) non prosegue allo stato successivo. In tutti gli altri casi si procede allo stato **PH**.
- **PH**: rappresenta lo stato di elaborazione del pH inserito dall'utente. Il sistema rimane in questo stato per tutto il tempo necessario affinché il pH diventi neutro. Quando viene rilevato un pH neutro, oppure quando viene alzato il segnale **RST**, il sistema torna allo stato di **RESET** pronto per ricevere un'altra istruzione.

Nello schema generale del circuito input e output sono rappresentati come: **INPUT / OUTPUT**:

**PARTICOLARE INPUT:**

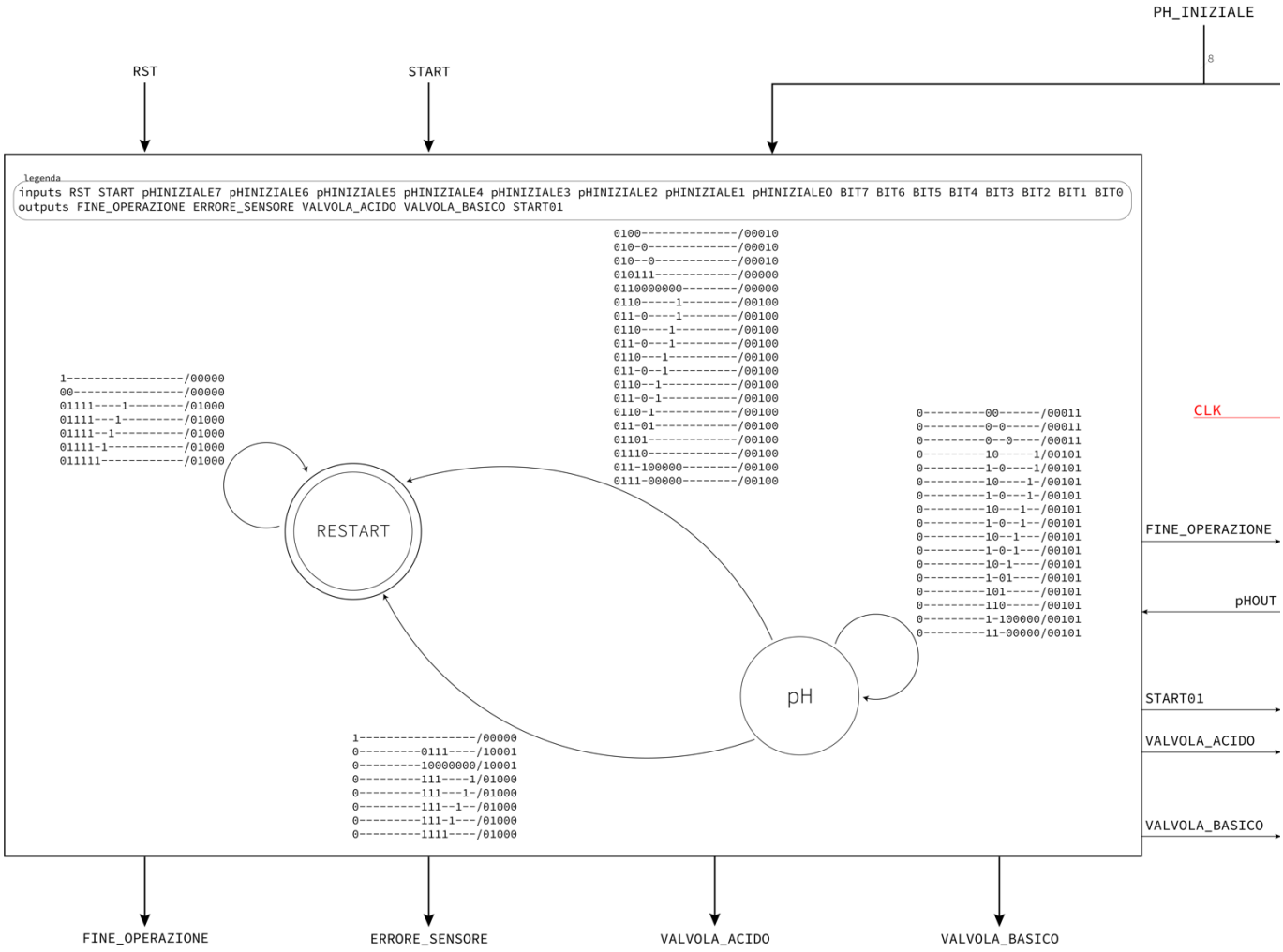
- **RST** (1 bit): rappresenta il segnale di **RESET**. Quando viene alzato, il sistema torna da un qualsiasi stato allo stato iniziale (**RESET**), ponendo tutte le porte in output a zero.
- **START** (1 bit): rappresenta il segnale di "inizio". Una volta ricevuto il segnale con valore 1, il sistema procede con la fase di elaborazione del pH inserito dall'utente.
- **PH_INIZIALE** (8 bit): rappresenta i bit, codificati in fixed_point, del pH inserito dall'utente. Il valore viene elaborato quando anche il segnale START viene alzato.
- **pHOUT** (8 bit): rappresenta i bit, codificati in fixed_point, del pH di ritorno elaborato dal DataPath pronto per essere nuovamente elaborato dalla FSM per determinarne l'acidità, la basicità o la neutralità.

**PARTICOLARE OUTPUT:**

- **FINE_OPERAZIONE** (1 bit): il segnale viene alzato quando il circuito ha terminato di eseguire tutte le operazioni.
- **ERRORE_SENSORE** (1 bit): il segnale viene alzato quando il valore del pH inserito dall'utente non risulta valido (> 14).
- **VALVOLA_ACIDO** (1 bit): il segnale viene alzato quando viene rilevato un pH basico.
- **VALVOLA_BASICO** (1 bit): il segnale viene alzato quando viene rilevato un pH acido.
- **START01** (1 bit): dopo aver ricevuto il pH iniziale dall'utente e dopo aver stabilito la natura della soluzione, il segnale viene alzato per permettere al DataPath l'esecuzione delle operazioni.

Nello schema del circuito FSM sono riassunte graficamente tutte le informazioni sugli stati e relativi segnali che abbiamo specificato in precedenza.

## SCHEMA GENERALE DEL CIRCUITO FSM:



## DESCRIZIONE DATAPATH

Il DataPath rappresenta la componente del circuito atta, principalmente, all'esecuzioni di calcoli, confronti e verifiche. Nel nostro caso presenta 5 input e 3 output in totale:

**$I_{DataPath}$ = {PH_INIZIALE, FINE_OPERAZIONE, VALVOLA_ACIDO, VALVOLA_BASICO, START01}**

**$O_{DataPath}$ = {NCLK, PH_FINALE, pHOUT}**

**PARTICOLARE INPUT:**

- **PH_INIZIALE** (8 bit): rappresenta i bit, codificati in fixed_point, del pH inserito dall'utente.
- **FINE_OPERAZIONE** (1 bit): il segnale viene alzato quando il circuito ha terminato di eseguire tutte le operazioni. A questo punto il DataPath procede a mandare in output i segnali NCLK e PH_FINALE.
- **VALVOLA_ACIDO** (1 bit): il segnale viene alzato quando viene rilevato un pH basico. Il DataPath procede a decrementare il pH di 0,25.
- **VALVOLA_BASICO** (1 bit): il segnale viene alzato quando viene rilevato un pH acido. Il DataPath procede a incrementare il pH di 0,50.
- **START01** (1 bit): alla prima elaborazione (START01 = 0) viene preso il PH_INIZIALE. Quando il segnale viene alzato viene preso il pH già elaborato.

**PARTICOLARE OUTPUT:**

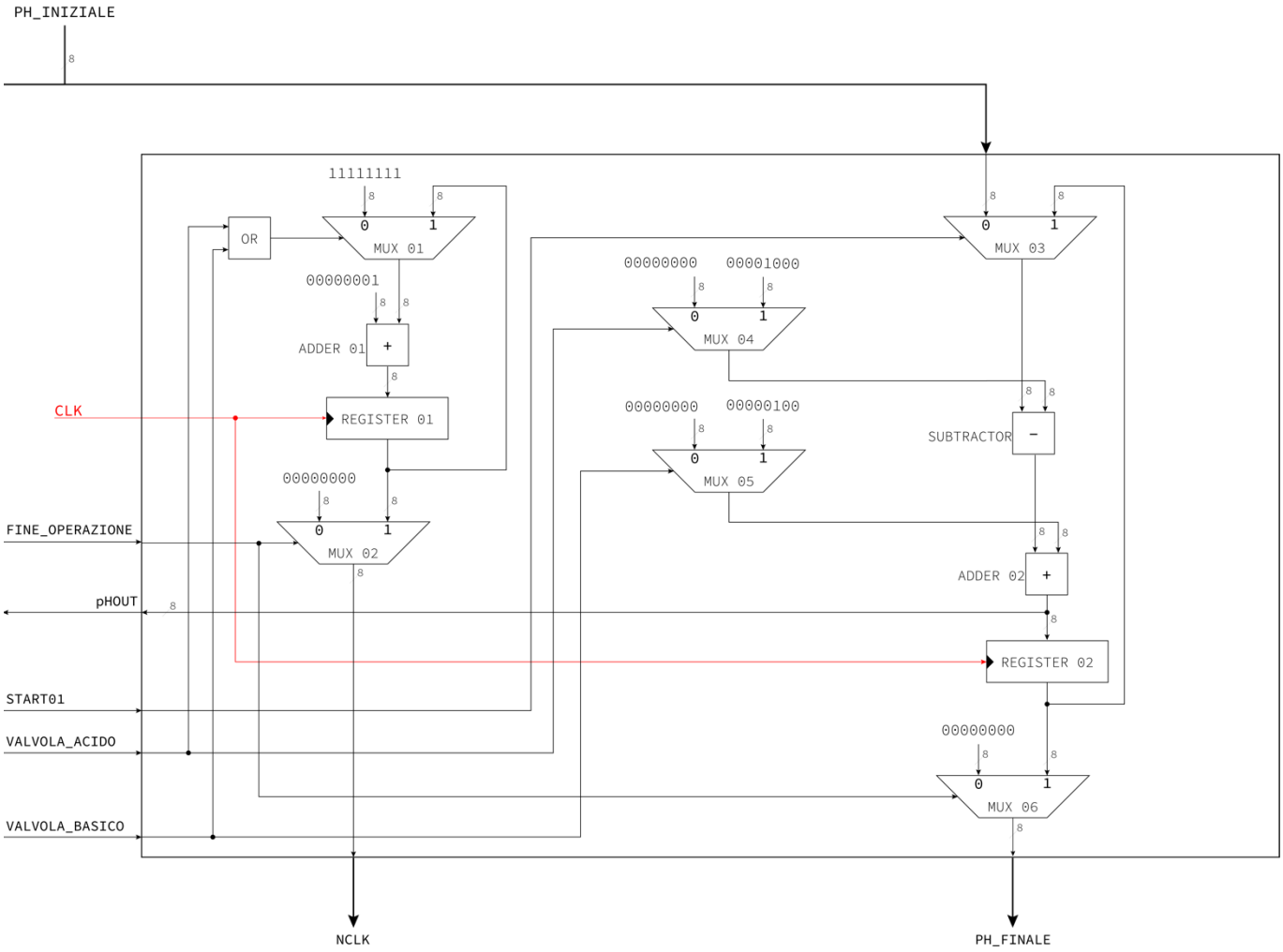
- **NCLK** (8 bit): rappresenta i bit del numero di clock che sono stati necessari per portare il pH inserito ad un valore neutro.
- **PH_FINALE** (8 bit): rappresenta i bit, codificati in fixed_point, del pH neutro elaborato dal circuito.
- **pHOUT** (8bit): rappresenta i bit, codificati in fixed_point, del pH che ogni volta viene passato alla FSM per determinarne l'acidità, la basicità o la neutralità.

Al DataPath arriva direttamente il segnale esterno **PH_INIZIALE** (così come anche nella FSM).

Dalla FSM al DataPath vengono passati direttamente quattro input: **FINE_OPERAZIONE, VALVOLA_ACIDO, VALVOLA_BASICO, START01**.

Dal DataPath alla FSM viene inviato direttamente l'output **pHOUT**.

## SCHEMA GENERALE DEL CIRCUITO DATAPATH:




Per esplicare il funzionamento del DataPath, abbiamo individuato due sezioni distinte: **CONTATORE DEL NUMERO DI CLOCK** ed **ELABORAZIONE DEL PH**.

### CONTATORE DEL NUMERO DI CLOCK

Il contatore ha il compito di restituire al segnale di output **NCLK** il numero di clock che sono stati necessari per portale il **PH_INIZIALE**, definito dall'utente, ad un valore neutro.
In caso la soluzione iniziale sia già neutra, il contatore restituirà il valore 0.

Al DataPath arrivano, dalla FSM, i segnali **VALVOLA_ACIDO** e **VALVOLA_BASICO**. La FSM invierà il segnale di apertura della rispettiva valvola e il contatore dovrà contare il numero di clock.
I due segnali delle valvole entrano in un componente logico **OR** che ha il compito di restituire il valore 1 se almeno una delle due valvole è aperta oppure 0 se entrambe sono chiuse. L'output della porta logica **OR** diventa il selettore di un Multiplexer (**MUX_01**) il quale prende in input il valore 11111111 e il valore che arriverà da un registro (**REGISTRO_01**).

Quando una delle valvole si aprirà e quindi il selettore risulterà pari a 1, il **MUX_01** invierà in output il segnale del **REGISTRO_01** (inizialmente 0), il quale diventerà un input di un sommatore (**ADDER_01**) che, come altro ingresso, prenderà il valore 1. In questo modo, ad ogni ciclo di clock, il valore verrà incrementato di 1 e memorizzato in **REGISTRO_01**.

A sua volta **REGISTRO_01** restituirà l'output al **MUX_01** e ad un altro multiplexer (**MUX_02**). **MUX_02** avrà il compito di inviare all'uscita **NCLK** il numero di clock calcolato in precedenza una volta che il circuito avrà concluso le operazioni. In altre parole, quando il segnale **FINE_OPERAZIONE** viene alzato, **MUX_02** restituirà in output il numero di clock, in tutti gli altri casi restituirà il valore 0.

**** SCELTA PROGETTUALE ****

In **MUX_01** come primo ingresso entra il valore 11111111. In questo modo, quando entrambe le valvole sono chiuse, è possibile sfruttare l'over-flow (11111111 + 1) per azzerare il **REGISTRO_01** e renderlo disponibile per una nuova operazione. Questo risulta fondamentale quando, al termine delle operazioni, viene inserito un altro valore.

## ELABORAZIONE DEL PH

L'elaborazione del pH rappresenta la fase in cui il valore del pH inserito inizialmente dall'utente viene elaborato per essere portato ad un valore neutro.
In particolare, se il pH inserito risulta essere già neutro, il circuito non esegue alcuna operazione. Situazione analoga nel caso si inserisca un pH non valido (> 14).

Al DataPath arriva il segnale **START01**. Questo segnale rappresenta il selettore di un multiplexer (**MUX_03**) il quale come primo ingresso riceve il **PH_INIZIALE** (input diretto del DataPath) e come secondo ingresso il valore di un registro (**REGISTRO_02**) che si occuperà di memorizzare il valore del pH elaborato ad ogni ciclo di clock. Quando **START01** vale 0, l'utente ha inserito il valore del pH e **MUX_03** manda in output il valore del **PH_INIZIALE**.

L'output entra in un sottrattore (**SUBTRACTOR**) il quale, in caso di soluzione basica, riceve, come secondo input, il valore 0,5 da un multiplexer (**MUX_04**), altrimenti riceve il valore 0. In caso di soluzione basica si effettuerà la differenza tra il pH ricevuto da **MUX_03** e il valore 0,5.

L'output del sottrattore entra in un sommatore (**ADDER_02**) il quale, in caso di soluzione acida, riceve, come secondo input, il valore 0,25 da un altro multiplexer (**MUX_05**), altrimenti riceve il valore 0. In caso di soluzione acida si effettuerà la somma tra il pH ricevuto dal sottrattore e il valore 0,25.

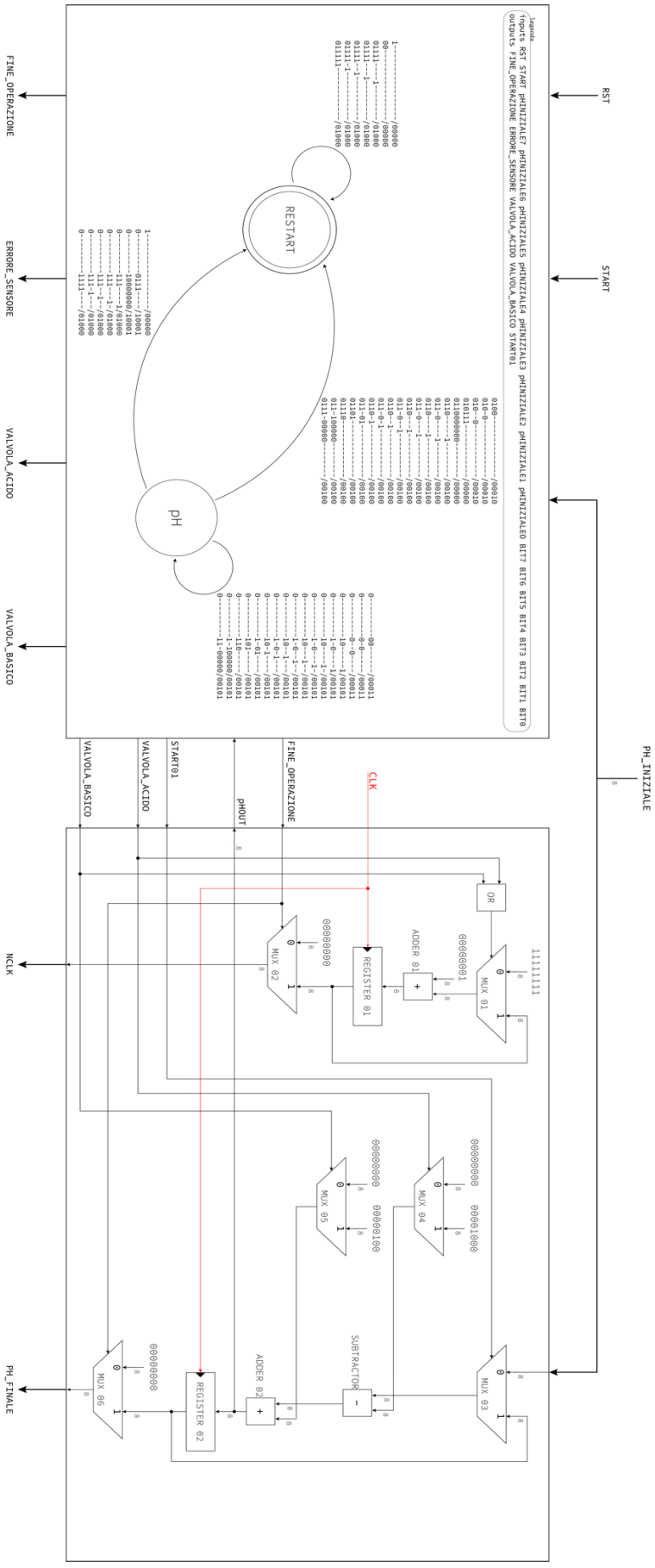
L'output di **ADDER_01** viene inviato tramite il segnale **pHOUT** alla FSM la quale verificherà la natura del pH e, se necessario, riattiverà le valvole. L'output di **ADDER_01** viene memorizzato anche all'interno di un registro (**REGISTRO_02**).

**REGISTRO_02** memorizza il pH ad ogni ciclo di clock e restituisce il valore a **MUX_03** (per una nuova elaborazione) e ad un altro multiplexer (**MUX_06**), il quale avrà il compito di inviare al segnale di output **PH_FINALE** il pH alla fine delle operazioni. In altre parole, quando il segnale **FINE_OPERAZIONE** viene alzato **MUX_06** restituirà in output il pH neutro, in tutti gli altri casi restituirà il valore 0.

**** SCELTA PROGETTUALE ****

Non abbiamo ritenuto necessario azzerare **REGISTRO_02** in quanto, in tutti i casi, il valore che viene memorizzato corrisponde al valore che sarà elaborato. L'unico caso in cui il valore del **REGISTRO_02** sarà pari a 0 coincide con l'avvio del circuito.

# SCHEMA GENERALE FSM & DATAPATH



## STATISTICHE DEL CIRCUITO

Vengono ora descritte le statistiche del circuito prima e dopo l'ottimizzazione per area. Si è cercato di minimizzare gli stati della FSM tramite il comando "**state_minimize stamina**" e successivamente con "**state_assign jedi**" ottenendo le seguenti statistiche:

```
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
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 : 2
Number of states in minimized machine : 2
sis> print_stats
FSM             pi=18   po= 5   nodes=  5       latches= 0
lits(sop)=   0  #states(STG)=   2
sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> print_stats
FSM             pi=18   po= 5   nodes=  6       latches= 1
lits(sop)= 344  #states(STG)=   2
```

Dopo l'ottimizzazione risultano i seguenti parametri:

```
sis> print_stats
FSM             pi=18   po= 5   nodes= 12       latches= 1
lits(sop)=  74  #states(STG)=   2
```

In questo modo abbiamo ottenuto una diminuzione del numero di letterali.

Il DataPath da noi implementato presenta le seguenti statistiche:

```
UC Berkeley, SIS 1.3.6 (compiled 2017-10-27 16:08:57)
sis> read_blif DATAPATH.blif
Warning: network `adder', node "cout" does not fanout
Warning: network `subtractor', node "lout" does not fanout
Warning: network `DATAPATH', node "cout" does not fanout
Warning: network `DATAPATH', node "[40]" does not fanout
Warning: network `DATAPATH', node "lout" does not fanout
sis> print_stats
DATAPATH        pi=12   po=24   nodes=102       latches=16
lits(sop)= 770
```

I warning non sono preoccupanti in quanto non interferiscono con il risultato del nostro circuito.

Viene richiesta l'ottimizzazione per area, quindi, è necessario verificare se è possibile ridurre al minimo il numero di letterali.
Dopo l'ottimizzazione risultano i seguenti parametri:

```
sis> print_stats
DATAPATH        pi=12   po=24   nodes= 64       latches=16
lits(sop)= 179
```

La FSMD, formata dall'unione dei due circuiti descritti nei precedenti capitoli, presenta le seguenti statistiche:

```
sis> read_blif FSMD.blif
Warning: network `adder', node "cout" does not fanout
Warning: network `subtractor', node "lout" does not fanout
Warning: network `DATAPATH', node "cout" does not fanout
Warning: network `DATAPATH', node "[42]" does not fanout
Warning: network `DATAPATH', node "lout" does not fanout
Warning: network `FSMD', node "cout" does not fanout
Warning: network `FSMD', node "[42]" does not fanout
Warning: network `FSMD', node "lout" does not fanout
sis> print_stats
FSMD            pi=10   po=20   nodes=108       latches=25
lits(sop)=1114
```

Dopo l'ottimizzazione risultano i seguenti parametri:

```
sis> print_stats
FSMD            pi=10   po=20   nodes= 63       latches=17
lits(sop)= 237
```

La riduzione del numero di letterali e di nodi è stata considerevole.

## MAPPING DEL CIRCUITO

Dopo aver ottimizzato il circuito, è necessario procedere a mappare lo stesso con l'ausilio di una libreria. Nel caso specifico si è considerato opportuno utilizzare la libreria "**synch.genlib**".

Dopo aver eseguito il mapping con i criteri di cui sopra, le statistiche sono le seguenti:

```
>>> before removing serial inverters <<<
# of outputs:          37
total gate area:       5360.00
maximum arrival time: (37.20,37.20)
maximum po slack:     (-7.40,-7.40)
minimum po slack:     (-37.20,-37.20)
total neg slack:      (-856.00,-856.00)
# of failing outputs:  37
>>> before removing parallel inverters <<<
# of outputs:          37
total gate area:       5184.00
maximum arrival time: (33.60,33.60)
maximum po slack:     (-7.40,-7.40)
minimum po slack:     (-33.60,-33.60)
total neg slack:      (-727.00,-727.00)
# of failing outputs:  37
# of outputs:          37
total gate area:       5120.00
maximum arrival time: (33.40,33.40)
maximum po slack:     (-7.40,-7.40)
minimum po slack:     (-33.40,-33.40)
total neg slack:      (-726.20,-726.20)
# of failing outputs:  37
```

Il **TOTAL GATE AREA** risulta essere pari a **5360.00**.
Il **CAMMINO CRITICO** risulta essere pari a **37.20**.