# Relazione elaborato SIS

A.A. 2016/2017

Giulio Zanchetta **VR413626**

Davide Benini **VR409482**

# Indice

# 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.

Il circuito è composto da un controllore e un datapath con i seguenti ingressi e uscite (nel seguente ordine!).

**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:**

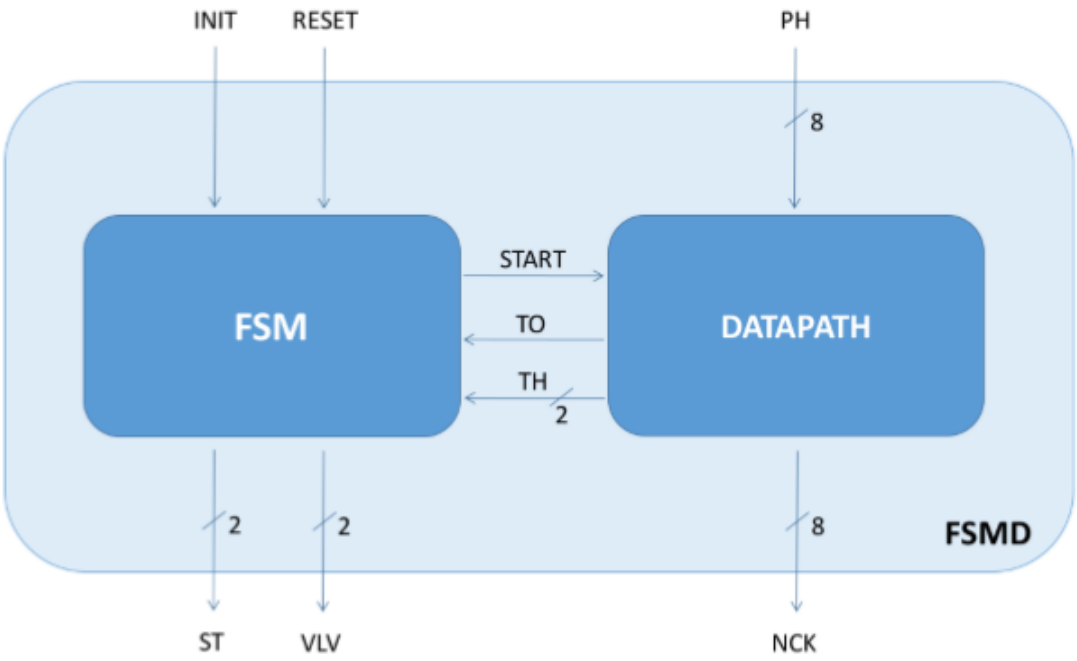
• ST [2]: indica in quale stato si trova la soluzione al momento corrente (01-A, 10-N, 11-B). Si considera la soluzione acida (A) quando PH<6 e basica (B) se PH>8.

• 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 neutro nel caso in cui la soluzione si trovi da più di 5 cicli di clock in stato A o B (01-BS, 10-AS)

Si richiede inoltre che il circuito sia mappato sulla libreria tecnologica synch.genlyb e che venga ottimizzato per area.

# SCHEMA DI RIFERIMENTO




La macchina è definita da un blocco FSM che funge da controllore e da un altro blocco chiamato DATAPATH che ha il compito di elaborare i dati.

L'unione dei due componenti forma l' FSMD unite da un' interfaccia di comunicazione con dei segnali:

• START [1]: segnale di start che comanda il datapath in funzione di INIT e RESET;

• TO [1]: segnale di timeout che indica quando il sistema si trova in uno stesso stato da più di 5 cicli di clock;

• TH [2]: stato in cui si trova il sistema, la codifica del TH segue questo modello:

10: Acido, 00: Neutro, 01: Basico

NB: la codifica del TH non è equivalente a quella del ST per una scelta progettuale del circuito di elaborazione.

# FSM

Il controllore presenta quattro ingressi (INIT[1], RESET[1], TO[1], TH[2]) e tre uscite (START[1], ST[2], VLV[2]).

Gli stati individuati che completano la nostra FSM sono quattro:

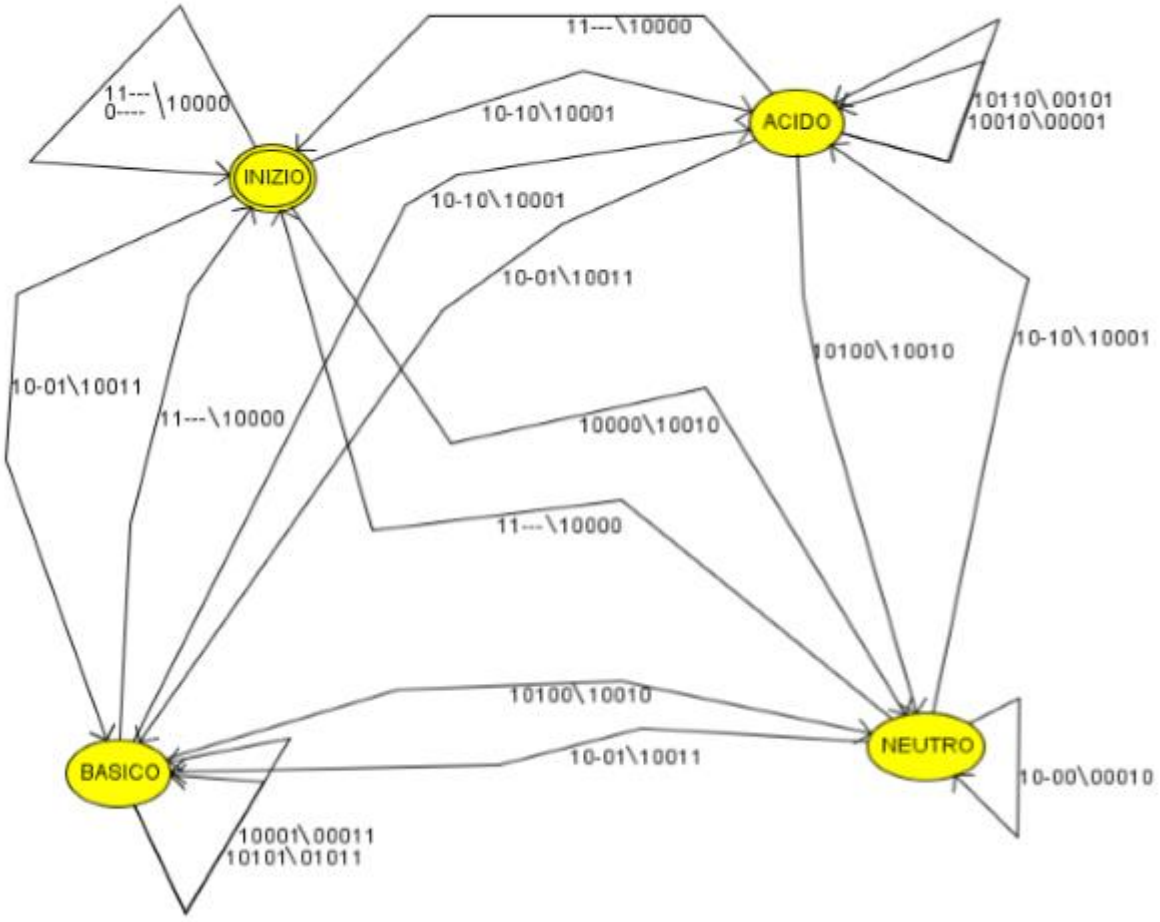
INIZIO: è lo stato di reset; la FSM resta in questo stato finché il sistema non è acceso (INIT=0) e il reset è attivo (RESET=1)

**BASICO**: stato che ci dice che il seguente PH[8] è basico.

**ACIDO**: stato che ci dice che il seguente PH[8] è acido.

**NEUTRO**: è lo stato che ci dice che la soluzione è neutra.

**SCHEMA DELLA FSM:**

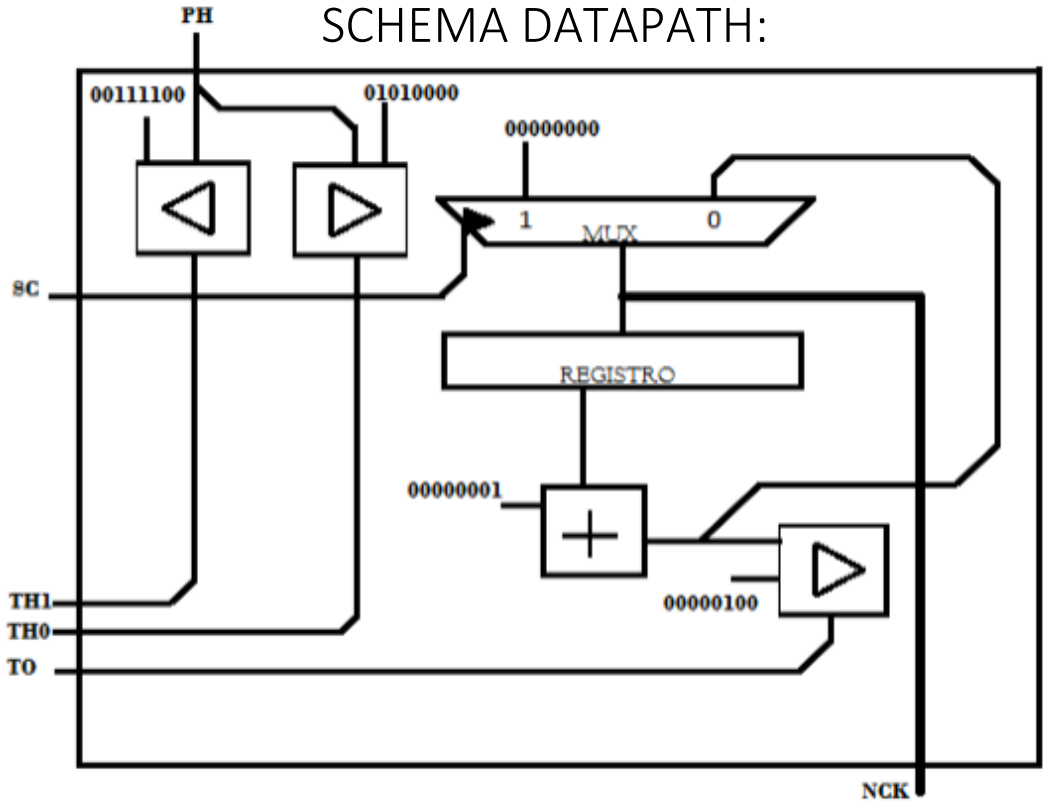


# DATAPATH

Il nostro DATAPATH presenta due ingressi che sono: START[1], PH[8] e tre uscite: TO[1], TH[2], NCK[8].

L' input START del DATAPATH viene posto ad uno ogni qualora volta abbiamo necessità di reinizializzare il valore di NCK e posto a zero quando dobbiamo mantenerlo.

## SCHEMA DATAPATH:




Le componenti che abbiamo utilizzato per realizzare seguono questo elenco:

- costanteuno.blif cabla il valore costante uno;
- costantezero.blif cabla il valore costante zero;
- costanti.blif cabla il valore di tutte le costanti quali sessanta, ottanta, quattro;
- maggiore8.blif è il componente che determina il maggiore tra due numeri;
- minore8.blif è il componente che determina il minore tra due numeri in bit;
- mux8_0const.blif è il multiplexer che permette di resettare o no il registro;
- not.blif per compiere l' operazione di 'NOT';
- reg1.blif componente che salva il valore di un bit;
- registro8.blif componente che salva il valore di un numero salvato in otto bit;
- sommatore.blif compie l' operazione di somma per un bit;
- sommatore8.blif è il componente per sommare un bit ad un numero salvato in otto bit;
- xor.blif definisce il comportamento per la 'XOR';
- xnor.blif definisce il comportamento per la 'XNOR' (XOR negato);
- DATAP.blif riunisce tutti i componenti in ordine per compiere la funzione richiesta

# STATISTICHE DEL CIRCUITO

La prima operazione eseguita è stata la minimizzazione degli stati della FSM:

```
sis> rl stg4.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>
```

Notiamo che il comando state_minimize stamina non è riuscito a minimizzare alcuno stato.

Non avendo assegnato alcuna codifica ai nostri stati, affidiamo il compito al comando **state_assign jedi** (predisposto a tale scopo):

```
Running jedi, written by Bill Lin,  UC Berkeley
sis> print_stats
STG             pi= 5   po= 5   nodes=  7       latches= 2
lits(sop)=  37  #states(STG)=   4
sis>
```

Per poi concludere lanciamo il comando per un' ulteriore ottimizzazione: **source script.rugged**

```
sis> source script.rugged
sis> print_stats
STG             pi= 5   po= 5   nodes=  7       latches= 2
lits(sop)=  29  #states(STG)=   4
sis>
```

Passiamo ora alla ottimizzazione del DATAPATH:

Prima dell' ottimizzazione:

```
sis> rl DATAP.blif
Warning: network `datapath', node "[78]" does not fanout
sis> print_stats
datapath        pi= 9   po=11   nodes= 86       latches= 8
lits(sop)= 357
sis>
```

Ottimizzato DATAPATH ci viene:

```
sis> source script.rugged
sis> print_stats
datapath        pi= 9   po=11   nodes= 21       latches= 8
lits(sop)=  91
sis>
```

# Mappatura della FSMD

Ottimizzati FSM e DATAPATH passiamo ora alla nostra FSMD, vediamo prima le statistiche prima dell' ottimizzazione.

```
sis> rl macchinafinale.blif
Warning: network `datapath', node "[82]" does not fanout
Warning: network `FSMD', node "[82]" does not fanout
sis> print_stats
FSMD            pi=10   po=12   nodes=105       latches=10
lits(sop)= 405
sis>
```

In seguito all' ottimizzazione risulta

```
sis> print_stats
FSMD            pi=10   po=12   nodes= 67       latches=10
lits(sop)= 163
sis>
```

Mappiamo ora il nostro circuito sulla libreria synch.genlib in modo da ottenere statistiche più o meno reali su area e ritardo.

```
sis> read_library synch.genlib
sis> map -s
```

Prima dell' ottimizzazione

```
>>> before removing serial inverters <<<
# of outputs:           22
total gate area:        3384.00
maximum arrival time: (17.80,17.80)
maximum po slack:     (-9.00,-9.00)
minimum po slack:     (-17.80,-17.80)
total neg slack:      (-351.60,-351.60)
# of failing outputs:  22
>>> before removing parallel inverters <<<
# of outputs:           22
total gate area:        3384.00
maximum arrival time: (17.80,17.80)
maximum po slack:     (-9.00,-9.00)
minimum po slack:     (-17.80,-17.80)
total neg slack:      (-351.60,-351.60)
# of failing outputs:  22
# of outputs:           22
total gate area:        3256.00
maximum arrival time: (17.80,17.80)
maximum po slack:     (-9.00,-9.00)
minimum po slack:     (-17.80,-17.80)
total neg slack:      (-351.60,-351.60)
# of failing outputs:  22
sis>
```

Dopo ottimizzazione

```
>>> before removing serial inverters <<<
# of outputs:           22
total gate area:        2888.00
maximum arrival time: (19.40,19.40)
maximum po slack:     (-6.00,-6.00)
minimum po slack:     (-19.40,-19.40)
total neg slack:      (-307.40,-307.40)
# of failing outputs:  22
>>> before removing parallel inverters <<<
# of outputs:           22
total gate area:        2888.00
maximum arrival time: (19.40,19.40)
maximum po slack:     (-6.00,-6.00)
minimum po slack:     (-19.40,-19.40)
total neg slack:      (-307.40,-307.40)
# of failing outputs:  22
# of outputs:           22
total gate area:        2648.00
maximum arrival time: (18.40,18.40)
maximum po slack:     (-6.00,-6.00)
minimum po slack:     (-18.40,-18.40)
total neg slack:      (-299.80,-299.80)
# of failing outputs:  22
sis>
```

# SCELTE PROGETTUALI

Riguardo l' FSM le scelte effettuate in base agli stati si sono rivelate corrette in quanto durante la minimizzazione non abbiamo riscontrato quasi alcun cambiamento.

L' assegnazione della codifica degli stati è stata eseguita in automatico dalla funzione state_assign jedi.

Per codificare l' input TH dell' FSM abbiamo deciso di risparmiare componenti mantenendo una codifica arbitraria a seconda di quello che risultava dall' output dei componenti di confronto tra costanti e PH.

Il Datapath è stato costruito secondo le esigenze di cui necessitavamo, unendo più componenti in uno e solo unico circuito di elaborazione dedicato a quella funzione.

Le specifiche su cui ci siamo basati sono rimaste legate a quelle proposte appunto per mantenere consistenza e una buona leggibilità e comprensione dello schema.