UNIVERSITÀ
di VERONA

Relazione SIS

Laboratorio di Architettura degli Elaboratori

Anno Accademico: 2022-2023

Casalini Marco VR490461

Pancheri Ermes VR487531

Indice

1 Descrizione del progetto

Nelle specifiche viene richiesto di creare un un parcheggio autonomo composto da tre zone, definite un numero di posti: area A con 31 posti, area B con 31 posti e l'area C con 24 posti. Ogni settore ha una sbarra di entrata e una di uscita.

L'utente richiede in quale settore ha intenzione di: entrare, la sbarra del settore si alzerà se il parcheggio non è pieno; uscire, la sbarra si alzerà se il parcheggio non è vuoto.

Il programma avrà due modalità, giorno e notte: durante il giorno si ha il funzionamento ordinario per entrare o uscire dal parcheggio, fino all'inserimento della sequenza "00000" da parte dell'operatore e il programma andrà in standby; mentre durante la notte c'è la possibilità di entrare o uscire a piacimento, di conseguenza le sbarre rimarranno aperte fino a quando l'operatore immette la la sequenza "11111" e il programma riprenderà. Tramite la sequenza di avvio ci saranno tre cicli di clock per aggiornare il numero di auto presenti in ogni settore, per poi riprendere il funzionamento ordinario.

I bit di ingresso saranno cinque:

In/Out[2 bit]	"01" se l'utente ha intenzione di entrare "10" se l'utente ha intenzione di uscire altrimenti viene considerato errore
Sector[3 bit]	"100" se l'utente ha intenzione di entrare nel settore A "010" se l'utente ha intenzione di entrare nel settore B "001" se l'utente ha intenzione di entrare nel settore C altrimenti viene considerato errore

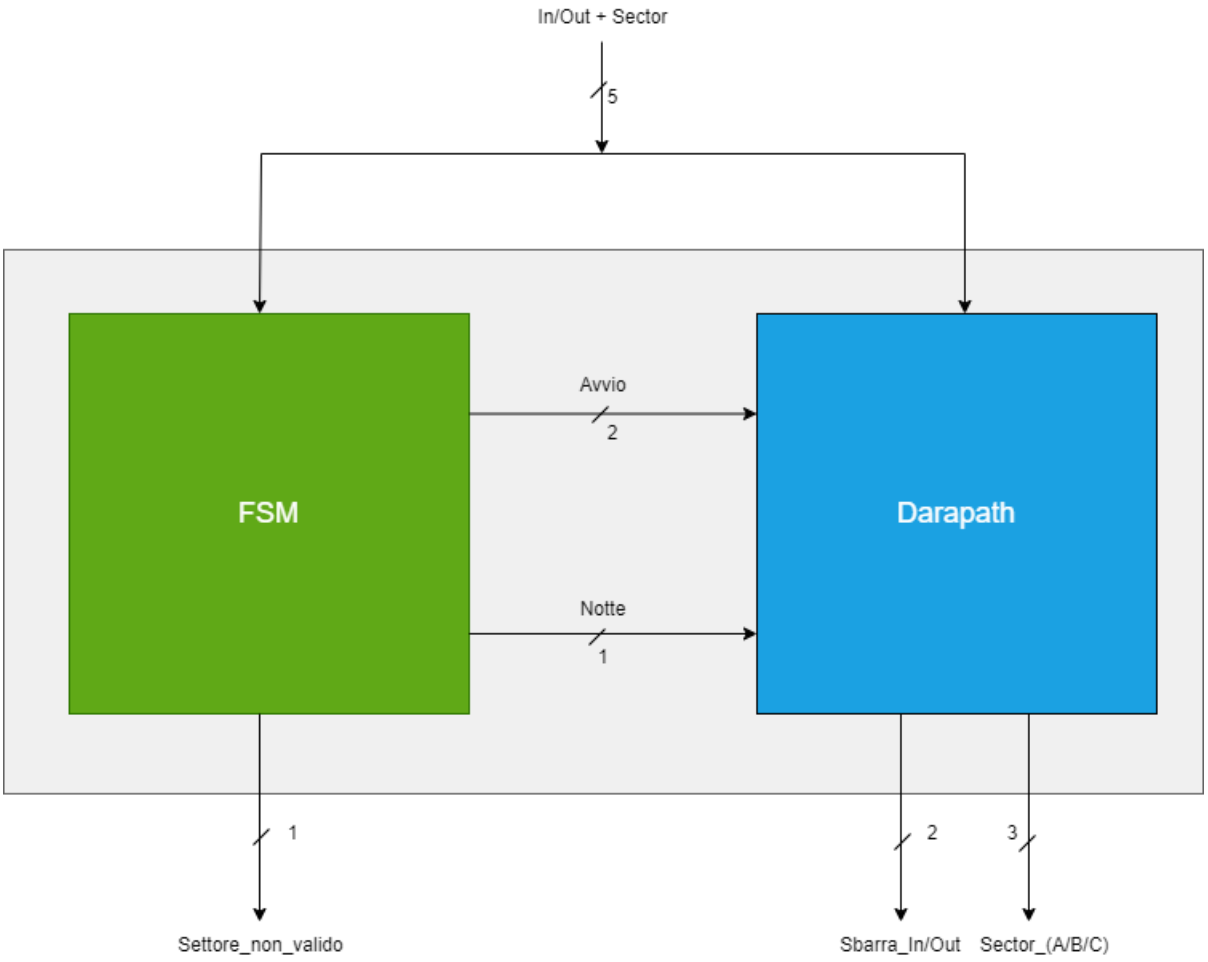
Ovviamente nella fase di avvio l'intero input viene considerato interamente come numero di auto nel parcheggio corrente.

I bit di uscita saranno sei:

Settore_non_valido[1bit]	"1" se il settore non è valido, "0 altrimenti"
Sbarra_In/Out[2 bit]	"10" se la sbarra per entrare è alzata "01" se la sbarra per uscire è alzata "00" entrambe le sbarre sono abbassate "11" entrambe le sbarre sono alzate (possibile solo nella fase notturna)
Sector_(A/B/C)[3 bit]	ad ogni bit corrisponde un settore, dove è "1" se il settore è pieno

2 Architettura generale del circuito

Schema generale del circuito:



Come si evince dal grafico i cinque bit di ingresso entrano sia nella FSM che nel datapath. Inoltre l'FSM comunica in uscita con il datapath, inviadndo quattro bit, e come output generale si ha soltanto un bit, mentre il datapath ha 5 bit di uscita dal circuito.

.

3 Final State Machine

Come scritto precedentemente l'FSM ha in input cinque bit e come output cinque bit, quattro comunicano verso il datapath e uno che farà parte dell'output finale del circuito:

- **Giorno [2]** serve ad indicare in quale settore ci stiamo riferendo all'avvio del programma al datapath, quindi utilizziamo le seguenti codifiche: dove con altro si intende il nomr-

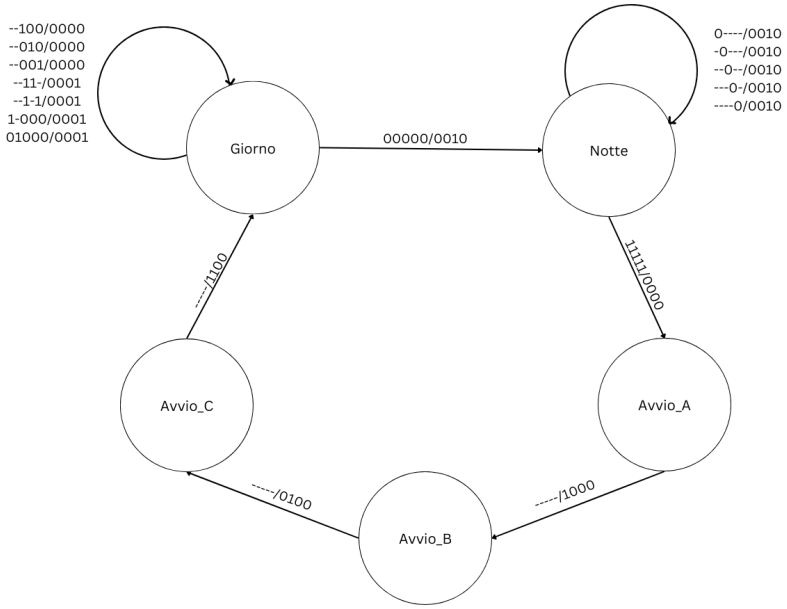
A	01
B	10
C	11
altro	00

  male funzionamento durante il giorno con l'accesso o l'uscita di una singola macchina da un determinato parcheggio (in questo caso sarà compito del datapath indirizzare correttamente il settore).

- **Notte [1]** indica al datapath se il programma deve stare "inattivo", bit alzato a 1, o "attivo", abbasato a 0, questo valore cambia unicamente all'inserimeno di "11111" e "00000" (ovviamente non se inserti nealla fase di avvio).

- **Settore_non_valido[1]** viene utilizzato appunto per segnalare l'invalidità del settore inserito se il bit sarà alzato a 1, come detto in precedenza, questo è l'unico output prodotto dalla fsm che uscirà dalla fsmd.

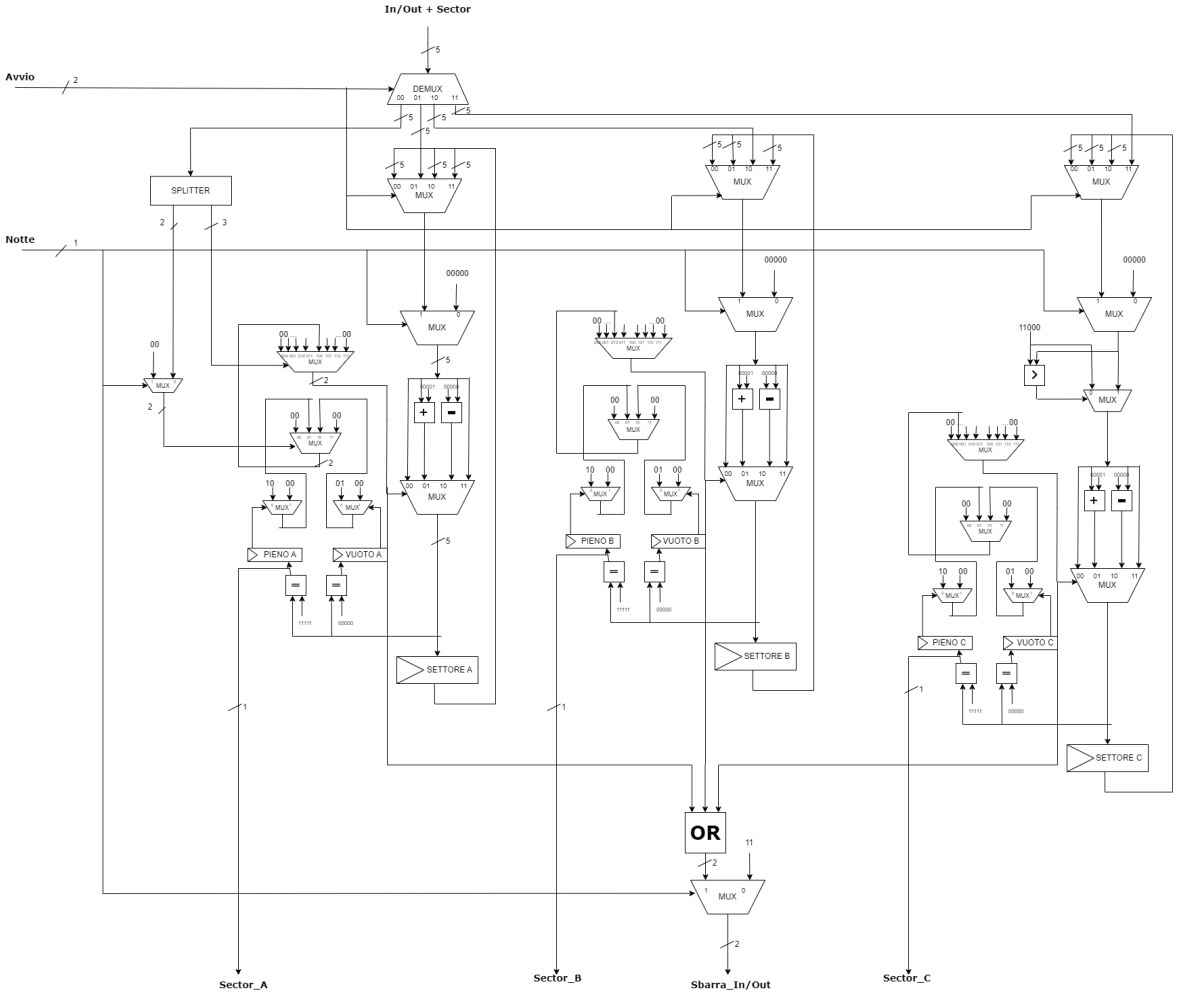
Durante la "Notte" il programma risulterà "inattivo" ovvero ignora qualsiasi tipo di input eccetto "11111", all'inserimento di quest'ultimo il programma continuerà ad ignorare l'input dell'utente, lo indirizzerà soltanto attraverso l'output impostando in automatico il settore, fino ad inserire il conteggio del settore C. Così il programma inizierà il suo effettivo funzionamento dove si filtrano gli input validi, altrimenti il bit di "Settore_non_valido" viene alzato ad 1. Il programma ritornerà nello stato "Notte" con l'inserimento "00000".


--100/0000
--010/0000
--001/0000
--11-/0001
--1-1/0001
1-000/0001
01000/0001
Giorno
00000/0010
Notte
0----/0010
-0---/0010
--0--/0010
---0-/0010
----0/0010
-----/1100
11111/0000
Avvio_C
Avvio_A
-----/0100
-----/1000
Avvio_B

4 Datapath

Come detto precedentemente il datapath prende un totale di 8 input: cinque bit provengono dalla FSMD e tre dalla FSM. Mentre i cinque bit di output escono tutti dal circuito generale.

Nel nostro caso quindi il datapath si occuperà di salvare i valori del settore corrente, aumentare o diminuire il numero di auto nel parcheggio scelto, verificare se i settori sono vuoti e infine alzare o tenere abbassata la sabrra di ingresso o uscita, tutto ciò ovviamente indirizzato dagli output del FSM.



Il primo demultiplexer è responsabile di indirizzare i 5 bit di ingresso verso una delle 4 parti del percorso dati. L'ingresso Avvio[2] derivante dalla FSM si occupa della selezione: se è "00" l'input generale di 5 viene portato in uno splitter che lo divide in InOut[2] e Settore[3]; se è è "01", "10" o "11", l'input viene quindi trattato come un numero binario da 0 a 31.

Se nel primo demultiplexer entra la codifica "00", e lo splitter divide l'input in due parti. La parte In/Out[2] entra poi in un multiplexer di un bit selezione, e ne uscirà se tale bi è uguale ad "0", altrimenti verrà fatto passare "00", essendo in fase notturna, le sbarre di tutti i settori saranno aperte di "default". Questi due bit, insieme ai tre del settore faranno successivamente da selettori per due multipexer per ogni settore, i quali verificheranno, grazie anche a due registri da un bit per ogni settore usati uno per tenere traccia se il numero di macchine per parcheggio è vuoto (=00000) e l'altro se è pieno (=11111 e 11000 per C), se la sbarra richiesta

nello stadio diurno si possa alzare per entrare o uscire.

Nelle altre tre codifiche di selezione del demultiplexer si accederà ai tre settori per inserire del registro richiesto il valore dei cinque bit, ma i restanti bit di uscita che non sono considerati nella selezione si porrano in automatico a 0, quindi necessitiamo di porre un multiplexer ogni settore collegato all'input Avvio[2] che filtrerà il passaggio degli 00000 nei settori non coincidenti con il demultiplexer, in questo casi si riattribuisce il valore già contenuto nel proprio registro.

Dopo questo multiplexer si passa ad un altro multiplexer, eccetto per il settore C il quale prima verifca se il valore dei cinque bit sia inferiore di 24, altrimenti lo sostituisce. Quest'ulteriore multiplexer fa passare l'input se non è notte, quindi come bit di selezione rimane Notte[1], e se è 1 passerà "00000", così facendo i registri di notte si resetteranno ad ogni richiesta. Prima di salvare il valore ne registro il segnale viene diviso in quattro, due entreranno in un altro multiplexer con la codifica di "00" e "11", gli atri due vengono uno sommato di uno e l'altro sottratto di 1 per poi finire nello stesso multiplexer con codifica "01" e "10", il bit di selezione sarà il bit risultante dalla serie di multiplexer nominati prima. Il valore finale viene salvaro nel registro del settore specifico, una parte del segnale viene però deviata e divisa per essere conrollata da due operatori di uguaglianza confrontati con il massimo e zero, per appunto dichiarare se il settore è pieno o vuoto (il segnale del risultante del settore vuoto per ogni settore uscirà come output generale, perchè appunto ne stabilisce la pienezza).

Infine sempre la risultante dell'insieme di multiplexer nominati inizialmente di ogni settore vengono confrontate in un OR, per poi entrare in un ultimo multiplexer che darà come output generale se la Sbarra_In/Out è alzato oppure no.

5 Statistiche del circuito

Prima di vedere l'ottimizzazione della FSMD, vediamo quella della FSM:

```
sis> state_assign jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> print_stats
FSM            pi= 5   po= 4   nodes=  7       latches= 3
lits(sop)=  92  #states(STG)=   5
```

Ora vedremo ed analizzeremo le statistiche del circuito prima e dopo l'ottimizzazione per area

```
sis> rl FSMD.blif
Warning: network `Controllore.blif', node "CI0" does not fanout
Warning: network `Controllore.blif', node "CI1" does not fanout
Warning: network `Controllore.blif', node "CI2" does not fanout
Warning: network `Controllore.blif', node "CI3" does not fanout
Warning: network `Controllore.blif', node "CI4" does not fanout
Warning: network `Calcolatore', node "COUT1" does not fanout
Warning: network `Calcolatore', node "COUT2" does not fanout
Warning: network `Datapath', node "COUT1" does not fanout
Warning: network `Datapath', node "COUT2" does not fanout
Warning: network `Datapath', node "[871]" does not fanout
Warning: network `Datapath', node "[882]" does not fanout
Warning: network `Datapath', node "[899]" does not fanout
Warning: network `Datapath', node "[910]" does not fanout
Warning: network `FSMD', node "COUT1" does not fanout
Warning: network `FSMD', node "COUT2" does not fanout
Warning: network `FSMD', node "[871]" does not fanout
Warning: network `FSMD', node "[882]" does not fanout
Warning: network `FSMD', node "[899]" does not fanout
Warning: network `FSMD', node "[910]" does not fanout
sis> print_stats
FSMD            pi= 5   po= 6   nodes=246       latches=24
lits(sop)=1755
```

Come si evince dalla foto il letterali sono inizialmente 1755 e i nodi sono 246. Dopo alcuni tentativi di ottimizzazione siamo giunti ad una conclusione che avesse il minor numero di letterari rispetto ai nodi, in modo tale da ridurre l'area: abbiamo eseguito due volte lo script rugged, un full_simplify e un fx.

```
sis> source script.rugged
sis> print_stats
FSMD            pi= 5   po= 6   nodes= 94       latches=24
lits(sop)= 414
sis> source script.rugged
sis> print_stats
FSMD            pi= 5   po= 6   nodes= 85       latches=24
sis> full_simplify
sis> print_stats
FSMD            pi= 5   po= 6   nodes= 85       latches=24
lits(sop)= 409
sis> fx
sis> print_stats
FSMD            pi= 5   po= 6   nodes= 93       latches=24
lits(sop)= 397
```

Così abbiamo ottenuto un totale di 397 letterali e 93 nodi.

6 Mapping del circuito

Si puo ora procerdere con il mapping del circuito, viene quindi caricata la libreria *sync.genlib*, per poi procedere a mappare l'area del circuito:

```
sis> read_library synch.genlib
sis> map -m 0
warning: unknown latch type at node '{[4]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[5]}' (RISING_EDGE assumed)
warning: unknown latch type at node '{[6]}' (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
sis> map -s
>>> before removing serial inverters <<<
# of outputs:          30
total gate area:       8640.00
maximum arrival time: (48.40,48.40)
maximum po slack:     (-8.60,-8.60)
minimum po slack:     (-48.40,-48.40)
total neg slack:      (-1022.40,-1022.40)
# of failing outputs:  30
>>> before removing parallel inverters <<<
# of outputs:          30
total gate area:       8624.00
maximum arrival time: (48.40,48.40)
maximum po slack:     (-8.60,-8.60)
minimum po slack:     (-48.40,-48.40)
total neg slack:      (-1022.60,-1022.60)
# of failing outputs:  30
# of outputs:          30
total gate area:       8064.00
maximum arrival time: (47.20,47.20)
maximum po slack:     (-8.60,-8.60)
minimum po slack:     (-47.20,-47.20)
total neg slack:      (-996.40,-996.40)
# of failing outputs:  30
```

come si vede dalle statistiche dell'ultimo comando abbiamo ottenuto un'area totale per i gate di 8640.00 e un ritardo massimo di 48.00.

7 Scelte progettuali

- Riduzione dei bit dei settori nel FSM: abbiamo ridotto i bit da tre a due per avere una riduzione dell'output ed essere facilitati per usarli nelle varie selezioni di multiplexer e dempuliplexer, abbiamo anche preso in considerazione l'idea di aggiungere un output dalla FSM verso il datapath per le sbarre in out ridotte ad un solo bit, ma sarebbe stato troppo difficoltoso gestirlo nel datapath.

- Tre stati di avvio: Nella FSM abbiamo creato tre stati per ogni settore per facilitare l'avvio della macchina e non dare "troppo peso" al datapath.

- Nessun output dal datapath verso la FSM: avendo ingrandito la FSM, abbiamo pensato di non fargli gestire gli input provenienti dal datapath, inoltre, grazie all'organizzazione di quest'ultimo ci è più facile gestire gli output richiesti.