Architettura degli elaboratori: Elaborato di SIS

Elia Piccoli, Marian Statache

A.A. 2017/2018

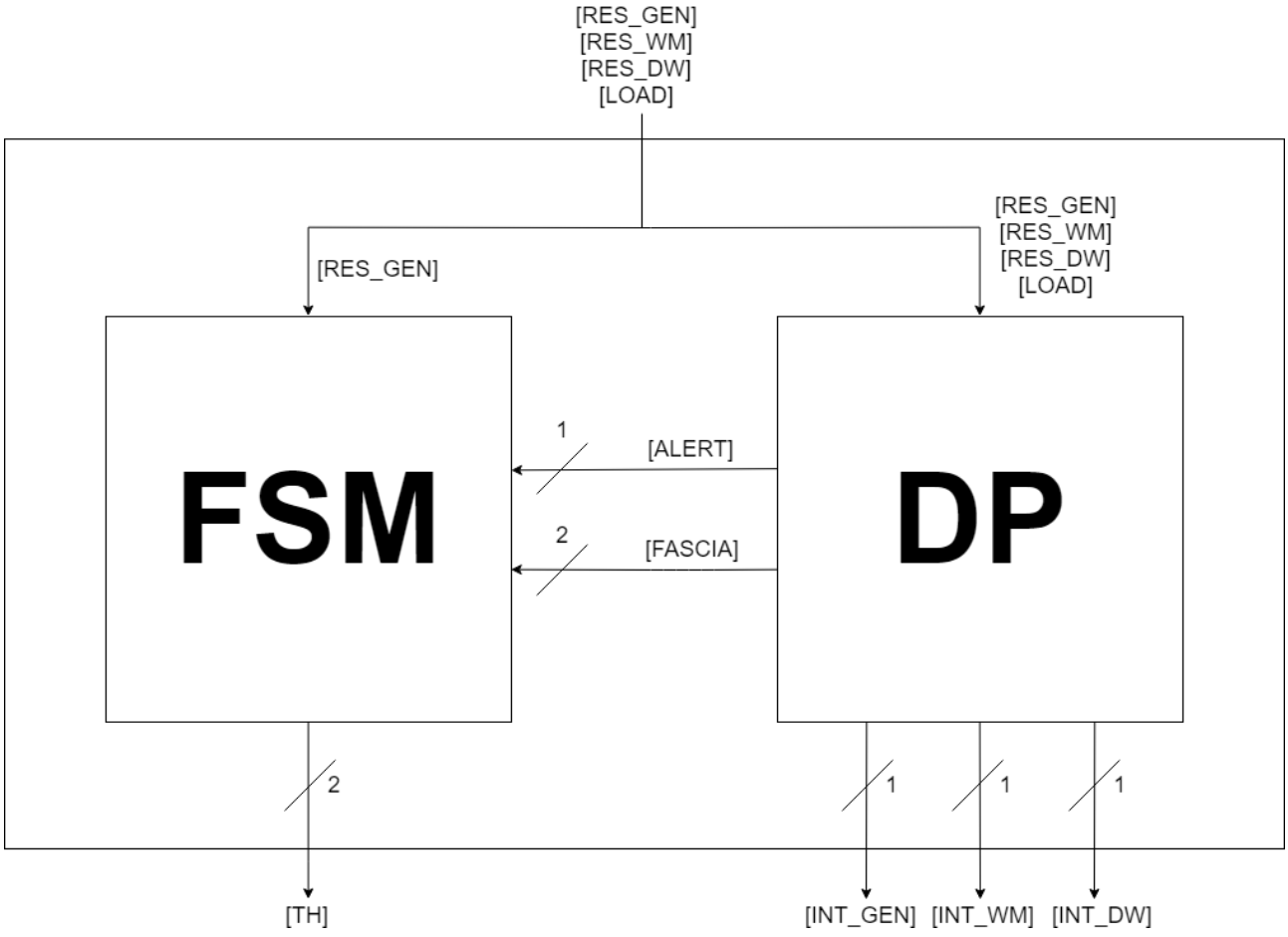
Sommario

Si desidera realizzare un circuito per il monitoraggio del consumo degli elettrodomestici.
Ricevuti in ingresso i reset del generale, lavastoviglie, lavatrice e un elenco dei consumi, restituisce in che fascia di consumo energetico si trova la casa: *Fascia 1*, *Fascia 2*, *Fascia 3*, o sovraccarico (*Overload*); inoltre, restituisce il valore degli interruttori del generale, della lavatrice e della lavastoviglie. Queste ultime vengono definite linee critiche di consumo, ovvero, in caso di 4 o più cicli in Overload, verrà spenta la lavastoviglie; poi al quinto ciclo la lavatrice e, infine, se si giunge al sesto ciclo di clock avendo un consumo fuori fascia, verrà spento l'interruttore generale, e, di conseguenza, anche gli altri due.

Architettura generale del circuito

Il circuito è stato implementato tramite lo sviluppo di una **FSMD,** modello composto da 2 componenti: un controllore realizzato attraverso una FSM (Finite State Machine), e un'unità di elaborazione realizzata col modello DATAPATH.

Nel disegno qui riportato si può osservare come sia organizzato il modello FSMD dell'elaborato, e come vengano gestiti i dati tra i due componenti.

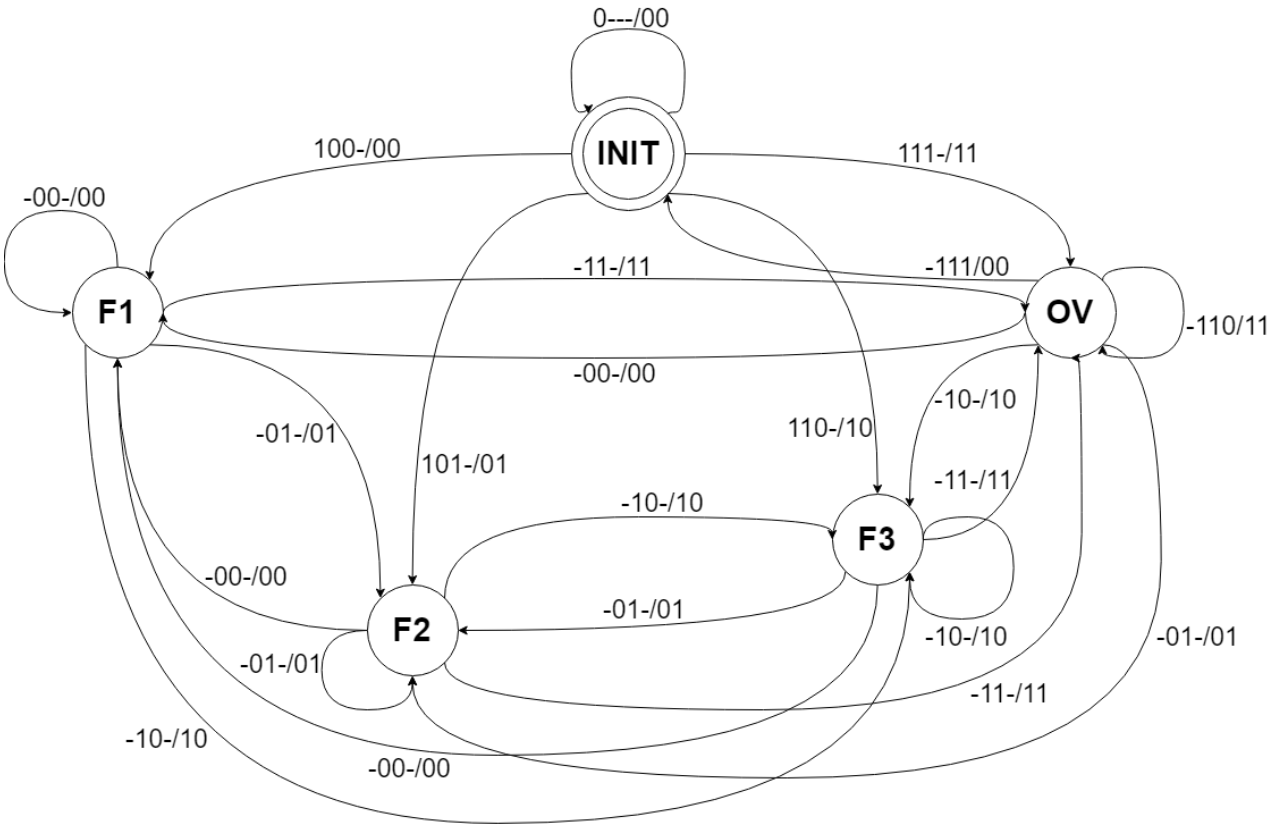


(*FSMD*)

Controllore

Il controllore è stato realizzato tramite una macchina a stati finiti; di seguito si riporta il grafico delle transizioni, con input e output corrispondenti, e una breve analisi del suo funzionamento.

State Transition Graph (STG)



(*Stg del controllore dell'elaborato*)

Input

Input relativi al controllore:

1. RES_GEN[1]: all'accensione, finché vale 0 deve essere mantenuto nello stato INIT. Non appena assume il valore 1 viene acceso l'interruttore generale. Un successivo 0 dopo l'accensione significa che l'interruttore generale non deve essere cambiato rispetto al suo valore precedente, a meno che non si voglia spegnere tutto.
2. FASCIA[2]: dato rielaborato dall'unità di elaborazione che viene utilizzato dal controllore per capire in che fascia si trova il nostro sistema:
    a. 00: Fascia 1
    b. 01: Fascia 2
    c. 10: Fascia 3
    d. 11: Overload
3. ALERT[1]: segnale di controllo inviato dal datapath; se vale 1 si è giunti al 6 ciclo di clock in Overload e il sistema torna allo stato INIT.

Output

Output relativi al controllore:

1. TH[2]: viene codificato lo stato di consumo:
   a. [00]: Fascia 1 → $0 \leq consumo \leq 1.5KW$
   b. [01]: Fascia 2 → $1.5KW < consumo \leq 3KW$
   c. [10]: Fascia 3 → $3KW < consumo \leq 4.5KW$
   d. [11]: Overload → $consumo > 4.5KW$

Architettura del controllore

Il controllo è stato pensato avente i seguenti stati:

1. INIT: è lo stato iniziale della macchina, quando tutti gli interruttori sono spenti.
2. F1, F2, F3: stati in cui la macchina è accesa e ha un consumo regolare, appartenente ad una delle rispettive fasce.
3. OV: è lo stato in cui la macchina va quando il consumo è superiore al limite massimo, e quindi si ha un sovraccarico.

FUNZIONAMENTO

Inizialmente ci troviamo nello stato INIT. Se **RES_GEN** vale 0, resto in questo stato, e gli output sono tutti 0. Se, invece, **RES_GEN** vale 1, accendo tutti gli interruttori, e, a seconda del valore di **FASCIA**, ci si sposta nello stato corrispondente: F1 se **FASCIA** vale 00, F2 se vale 01, F3 se vale 10, e, infine, OV qualora il carico totale superi la soglia (4500W), e, quindi, **FASCIA** vale 11.
Una volta che la macchina è accesa, **RES_GEN** non viene più considerato, e si cambia stato solamente in base a **FASCIA**.

Fino ad ora non abbiamo considerato l'input di **ALERT**: questo perché è ininfluente in qualsiasi stato, eccetto che per OV. Infatti, se ci troviamo in OV, **FASCIA** vale 11, e l'input di **ALERT** vale 1 - indicatore del superamento dei 6 cicli in Overload consecutivi - il controllore torna in INIT. In alternativa, se **FASCIA** vale sempre 11, e **ALERT** vale 0, l'FSM effettua un'auto-transizione, rimanendo così nello stato OV.

Si può notare che l'output **TH** sarà sempre uguale all'input **FASCIA**, tranne della transizione da OV a INIT, dove l'input **FASCIA** vale 11, mentre l'output **TH** vale 00.

Scelte progettuali

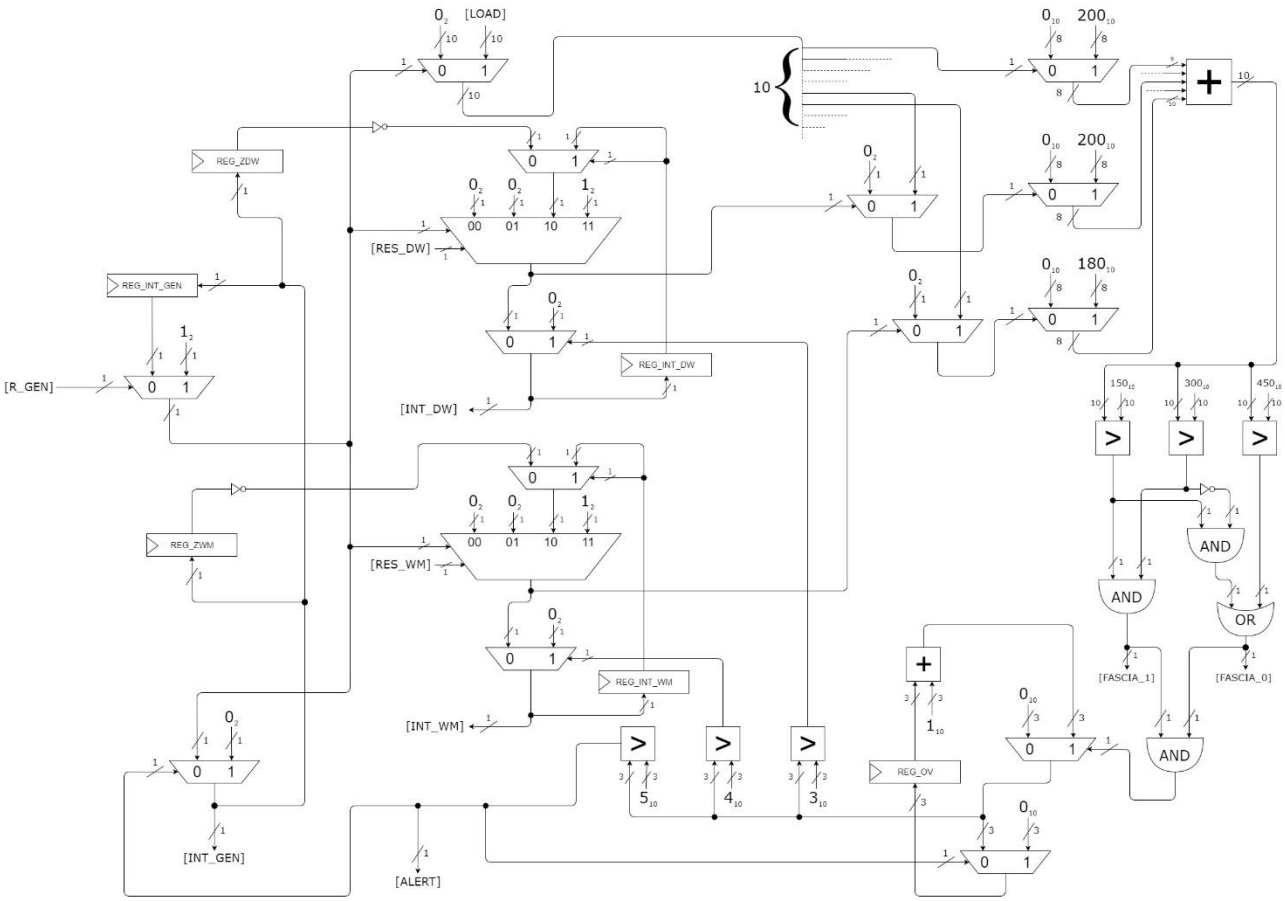
Si sarebbe potuto scegliere di fare una macchina sia con più stati, ad esempio con 7 stati, nella quale avremmo scorporato lo stato OV in 3 stati differenti: uno che si sarebbe occupato dei conteggi di clock fino al terzo, nei quali non viene spento nessun interruttore; un altro per il quarto ciclo in Overload che avrebbe spento l'interruttore della lavastoviglie; e un ultimo che avrebbe spento la lavatrice al quinto ciclo, prima di spegnere eventualmente tutto. Fare tale scelta, però avrebbe reso la nostra FSM significativamente più grande ed ingombrante, con un minimo guadagno da parte del datapath.

Allo stesso modo, avremmo potuto realizzarne una con solo 3 stati, accorpando i 3 stati F1, F2 ed F3 in uno solo visto che la differenza del funzionamento della macchina in questi 3 stati è minima. Tale semplificazione, però sarebbe risultata prima di tutto poco utile, e in secondo luogo avrebbe reso la lettura della FSM un po' meno chiara.

Unità di elaborazione

L'unita di elaborazione è stata implementata tramite il modello DATAPATH schematizzato secondo la seguente struttura.

Datapath



(*Schema del modello DATAPATH*)

Input

Input dell'unità di elaborazione:

1. RES_GEN[1]: valore del reset del generale; se vale 1 al primo input accende tutti gli interruttori nonostante il loro reset possa valere 0; ai successivi input, se vale 0, verrà considerato il valore precedente, mentre se vale 1 verrà alzato l'interruttore del generale.
2. RES_WM[1]: valore del reset della lavatrice; se vale 1 verrà alzato l'interruttore della lavatrice, altrimenti verrà preso in considerazione il valore precedente.
3. RES_DW[1]: valore del reset della lavastoviglie; se vale 1 verrà alzato l'interruttore della lavastoviglie, altrimenti verrà preso in considerazione il valore precedente.
4. LOAD[10]: set dei consumi istantanei; ogni bit corrisponde a un differente elettrodomestico di un dato consumo, se il rispettivo bit vale 1 il carico verrà preso in considerazione, quindi è acceso, altrimenti è spento.

Output

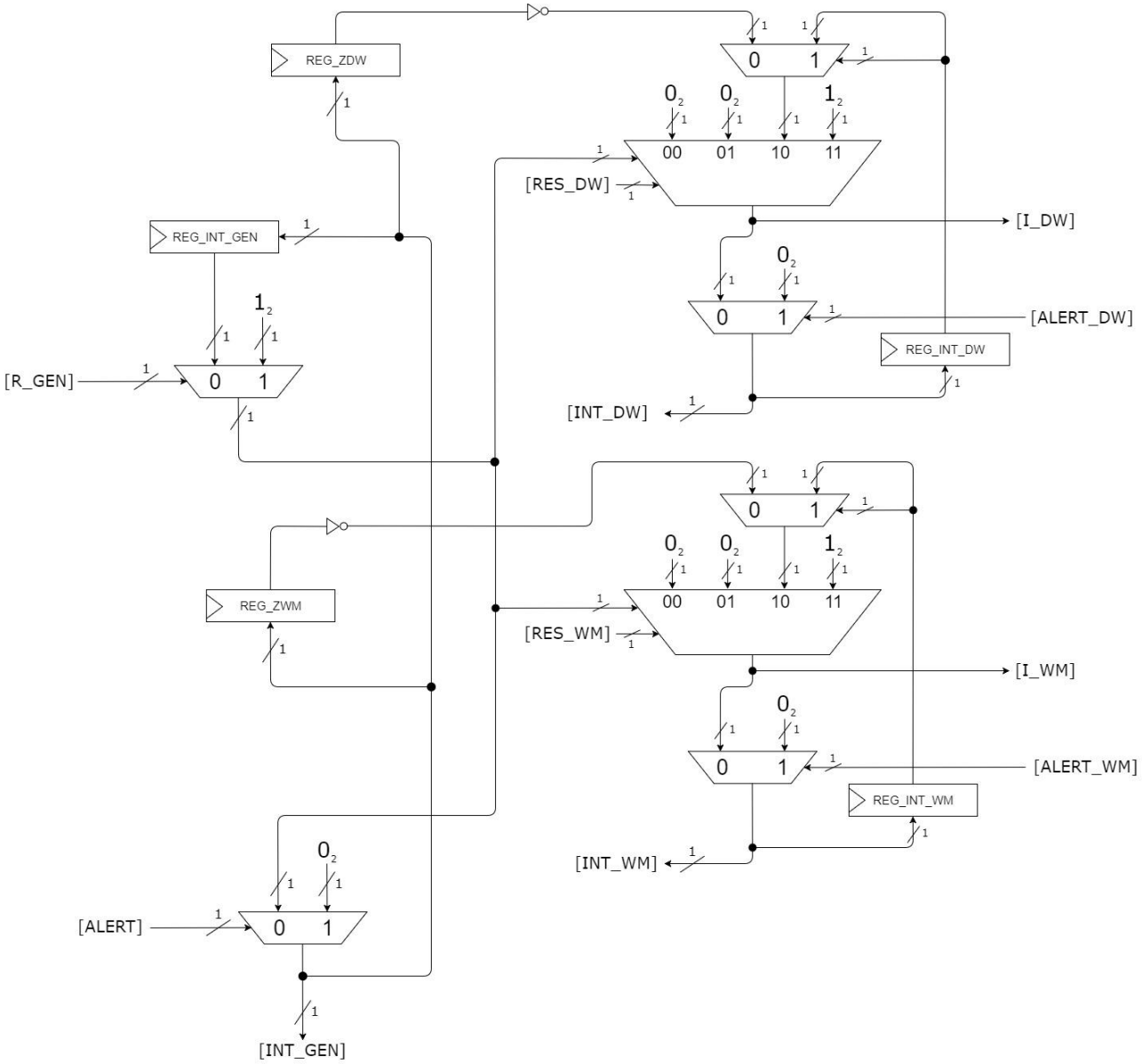
Output dell'unità di elaborazione:

1. FASCIA[2]: attraverso la lettura dei diversi carichi, e l'analisi del consumo totale viene ricavata la fascia di consumo attuale, codificata:
    a. 00 ➔Fascia 1
    b. 01 ➔Fascia 2
    c. 10 ➔Fascia 3
    d. 11 ➔Overload
2. ALERT[1]: segnale di allarme che viene inviato al controllore; quando vale 1 si è raggiunto il sesto ciclo di clock in Overload consecutivo.
3. INT_GEN[1]: segnale che dopo vari controlli rappresenta lo stato dell'interruttore generale: 1/0 →ON/OFF
4. INT_WM[1]: segnale che dopo vari controlli rappresenta lo stato dell'interruttore della lavatrice: 1/0 →ON/OFF
5. INT_DW[1]: segnale che dopo vari controlli rappresenta lo stato dell'interruttore della lavastoviglie: 1/0 →ON/OFF

Analisi Datapath

In seguito viene analizzato nello specifico il funzionamento del DATAPATH.
Per praticità e chiarezza verrà suddiviso in blocchi, che fanno tutti riferimento all'architettura rappresentata precedentemente, analizzando logicamente e progettualmente il significato e il compito di ogni parte dell'unità di elaborazione.

Gestione interruttori



(*Blocco calcolo interruttori*)

L'elaborazione inizia con l'analisi del valore di **RES_GEN**, attraverso un multiplexer, che ha come scelta quest'ultimo; viene caricato il valore di **OF**, segnale che indica al DP se la macchina è accesa o spenta. Se **RES_GEN** vale 1, anche **OF** vale 1, altrimenti viene caricato su **OF** il valore precedente di **INT_GEN**, salvato all'interno del rispettivo registro.
Il valore di **OF** viene poi ramificato in 4 parti distinte, 3 delle quali vengono utilizzate per la gestione di questo blocco; la quarta, invece, viene inviata al blocco per il calcolo della fascia.

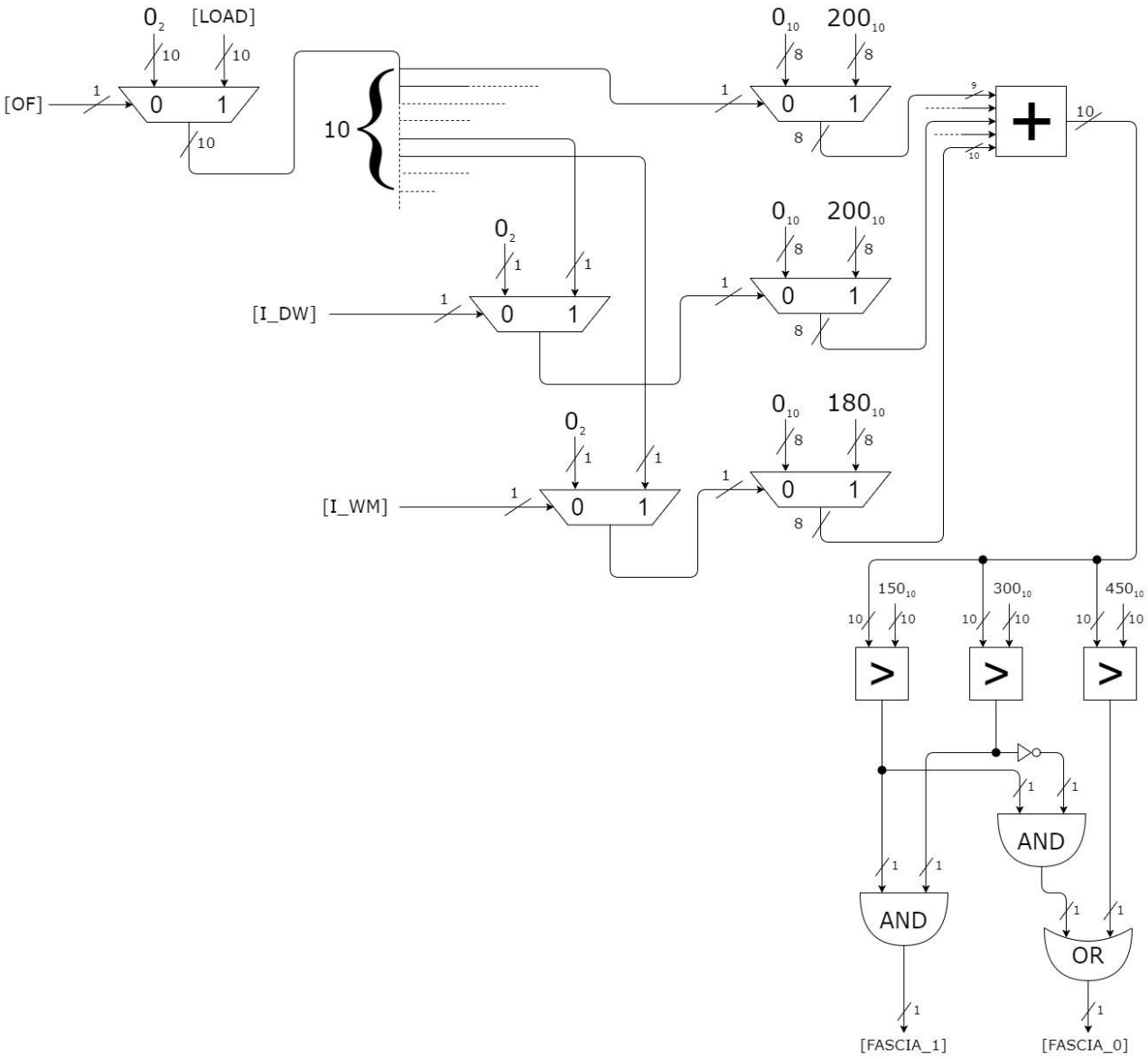
OF viene utilizzato come primo bit di scelta nei 2 multiplexer a 2 bit di scelta che permettono di avere il valore di **I_DW** e **I_WM**. In questi multiplexer il secondo bit di scelta è dato rispettivamente da **RES_DW** e **RES_WM**; se **OF** vale 0, allora la macchina è spenta, di conseguenza gli interruttori avranno valore 0; se **OF** vale 1 e il **RES_DW**/**RES_WM** vale 1, allora **I_DW**/**I_WM** viene settato a 1; se **OF** vale 1 e **RES** vale 0 viene caricato il valore precedente dell'interruttore, a meno che non si tratti dell'accensione iniziale della macchina, come vedremo tra poco. L'uscita di questo multiplexer viene poi inviata al blocco per il calcolo della fascia.

Se deve essere caricato il valore precedente di **INT_DW**/**INT_WM** abbiamo per entrambi una stessa gestione: il valore **I_DW**/**I_WM** è deciso da un multiplexer nel quale il bit di scelta deriva dal registro che ci ha salvato il valore dell'interruttore del ciclo precedente; nel caso valga 0, esso fa passare il negato del valore precedente di **INT_GEN** salvato su **ZDW**/**ZWM**, distinguendo così il caso dell'accensione della macchina – al ciclo precedente **INT_GEN** valeva 0, quindi il suo negato è 1 e di conseguenza si tratta dell'accensione di tutta la macchina, e restituisco 1 anche se in precedenza **INT_DW**/**INT_WM** valeva 0 – da quello in cui la macchina era già accesa – quindi restituirà 0. Se, invece, il valore di **INT_DW**/**INT_WM** del ciclo precedente vale 1, passerà 1.

Il valore di **I_DW**/**I_WM** viene poi preso in gestione da un multiplexer che ha come bit di scelta **ALERT_DW**/**ALERT_WM**; **ALERT_DW** e **ALERT_WM** sono inviati dal blocco che gestisce l'Overload. Il primo vale 1 nel caso in cui si siano verificati 4 o più cicli consecutivi in Overload, mentre il secondo quando si sono verificati 5 o più cicli. L'uscita di questo multiplexer è **INT_DW**/**INT_WM,** il quale sarà sia output della macchina, sia ingresso del registro che ne salva il valore per il ciclo successivo.
Nel caso in cui **ALERT_DW**/**ALERT_WM** valga 1, viene fatto passare il valore 0, e l'interruttore viene spento, altrimenti viene fatto passare il valore di **I_DW**/**I_WM**.

Infine **OF** viene mandato come uno degli input del multiplexer per l'elaborazione del valore di **INT_GEN**. Questo multiplexer ha come bit di scelta **ALERT**, elaborato dal blocco dell'Overload, e comunica se sono stati raggiunti 6 cicli consecutivi oltre il limite. Se **ALERT** vale 0 **INT_GEN** varrà il valore di **OF**, altrimenti **INT_GEN** varrà 0 ovvero viene spento l'interruttore generale, e quindi tutta la macchina. **INT_GEN** sarà, infine, diviso in due, in modo da mandare il suo valore in output all'FSMD e al rispettivo registro che ne salverà il valore per il ciclo successivo.

Gestione calcolo della Fascia

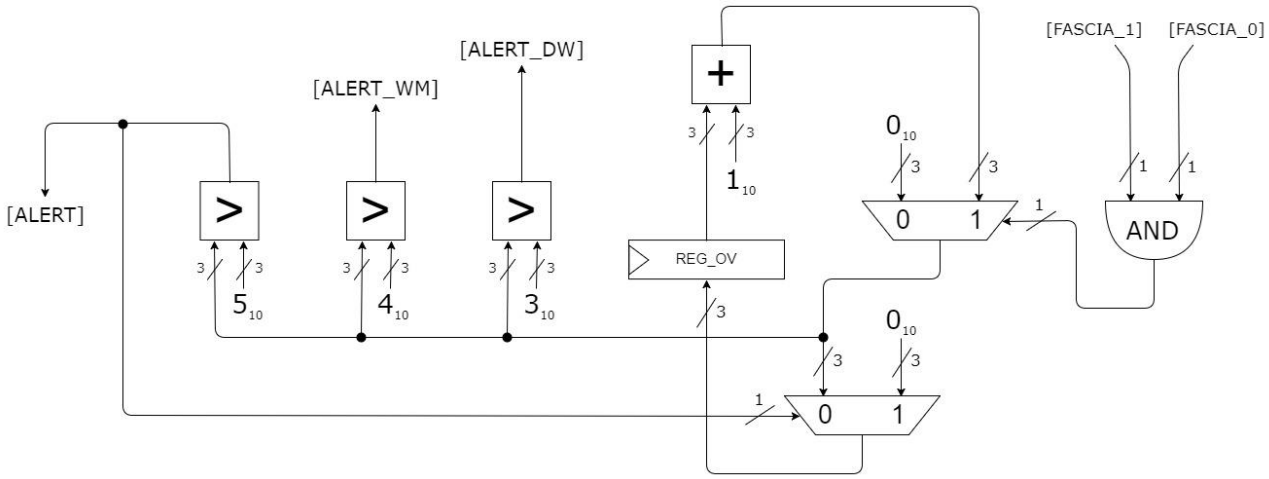


(*Blocco calcolo fascia di consumo*)

Il calcolo della fascia inizia con un multiplexer che ha come scelta il valore **OF** derivante dal blocco che gestisce gli interruttori. Se esso vale 1, passerà i 10 bit del **LOAD**, che sono input della macchina, altrimenti passerà dieci bit a 0. Ognuno di questi bit fungerà da scelta per altrettanti multiplexer, che decideranno se far passare il valore del consumo corrispondente o meno. In seguito questi carichi verranno passati ad un sommatore a 10 ingressi da 10 bit ognuno, che ne farà la somma e restituirà il consumo totale.
Fanno eccezione i due bit del **LOAD** corrispondenti alla lavastoviglie e alla lavatrice, che invece di andare direttamente ai loro multiplexer con il loro consumo, passano prima da un altro multiplexer che ha come bit di scelta **I_DW**/**I_WM**, derivanti dal blocco che gestisce gli interruttori. Infatti, se **I_DW**/**I_WM** vale 0, verrà passato il valore 0, in modo da annullare il carico relativo in anticipo, altrimenti verrà passato il bit del **LOAD** corrispondente.

Una volta ottenuto il valore del consumo totale, viene passato a tre 'maggiori', i quali serviranno per individuare i due bit della fascia di consumo corrispondente. Attraverso una combinazione nostrana di porte AND, OR e NOT, siamo riusciti ad ottenere i due bit della fascia di consumo a partire dalle uscite dei tre 'maggiori'.

Gestione Overload



(*Blocco gestione Overload)*

Questo blocco parte sdoppiando i segnali di output dei due bit della fascia di consumo, e portandoli assieme ad una porta AND. Il segnale di uscita di questa porta ci dirà se siamo in sovraccarico o meno: infatti, solo qualora entrambi i due bit valgano 1 – codifica della fascia di Overload – l'uscita della porta AND varrà anch'essa 1.
In seguito, questo segnale giungerà ad un multiplexer: se non saremo in Overload, passerà il valore 0, valore del contatore dei cicli di clock in Overload consecutivi; altrimenti, se saremo in Overload, passerà il valore salvato precedentemente in un registro ed incrementato di uno da un sommatore – da notare che anche qualora si inserisca come primo input un **RES_GEN** uguale a 1 ed un **LOAD** che porta al sovraccarico, anche se precedentemente non avevamo salvato nulla in quel registro, esso parte di default da zero, che incrementato di 1 fa passare 1 al multiplexer, conteggiando in modo giusto anche in questo particolare caso. A questo punto, l'uscita di questo multiplexer, prima di andare al registro per essere salvato, passa attraverso un altro multiplexer, il cui bit di scelta sarà **ALERT**, che ha la funzione di verificare che non abbiamo superato i 6 cicli di Overload consecutivi: trovandosi in questa situazione, infatti, questo multiplexer reimposta il contatore dei cicli a 0, evitando così di continuare a contare in caso di un immediato riavvio con un **LOAD** nuovamente troppo elevato, e di mandare il contatore in overflow.
Infine, prima di questo multiplexer, il valore del contatore viene sdoppiato e mandato a tre diversi 'maggiori', che controlleranno rispettivamente il raggiungimento dei 4, 5 e 6 cicli di Overload, da cui usciranno **ALERT_DW**, **ALERT_WM** e **ALERT**. **ALERT** sarà ramificato in tre segnali, e sarà output, verrà usato come scelta del multiplexer visto poco fa, e verrà mandato al blocco che si occupa degli interruttori.

Scelte progettuali

In riferimento al blocco che si occupa del calcolo della fascia di consumo, avremmo impiegato meno tempo scegliendo di utilizzare un multiplexer a 3 bit di scelta, che utilizzavano le tre uscite dei 'maggiori' per dare la giusta fascia in uscita. Tale scelta però avrebbe sicuramente aumentato l'area del circuito in modo non marginale. Si è optato quindi per una soluzione più intelligente, che, grazie ad un po' di ragionamento, ha permesso di risparmiare un po' sull'area totale.

Inoltre, sempre a livello del calcolo della fascia, si è scelto di convertire tutti i singoli consumi – e di conseguenza anche quello totale – in daW (decaWatt), visto che ogni consumo in Watt è multiplo di 10. La massima somma dei consumi diventa, così, pari a 954daW, per il quale sono sufficienti 10bit; al contrario, se avessimo mantenuto tutto in Watt avremmo avuto bisogno di ben 14 bit, cambiamento non da poco, considerato che abbiamo dieci diversi consumi e un consumo totale, quindi 154 bit totali contro 110. Non avremmo potuto, però, nemmeno utilizzare un multiplo più grande dei daW, poiché saremmo stati obbligati ad usare la notazione con la virgola, oppure ad effettuare arrotondamenti.

Per quanto riguarda il blocco che si occupa del calcolo delle fasce, inizialmente avevamo impostato tutti i multiplexer che danno i carichi con 10 bit per le entrate e l'uscita, proprio come quelli necessari per il consumo totale. Abbiamo notato, però che i carichi singoli richiedono al massimo 8 bit, quindi abbiamo scelto di impostare questi multiplexer a 8 bit, risparmiando così sul numero di letterali, e, come appurato in seguito all'ottimizzazione, anche per quanto riguarda l'area. Gli ingressi del sommatore, invece, saranno tutti sempre a 10 bit, con la differenza che i primi due saranno impostati in automatico a 0. In aggiunta, per un ulteriore miglioramento, abbiamo notato che per la somma dei primi due numeri sarebbero bastati 9 bit – probabilmente anche per la somma col terzo, ma abbiamo voluto mantenere la macchina più generale, in caso di modifiche – quindi abbiamo scritto un nuovo sommatore a 9 bit richiamato dal sommatore a 10 ingressi, che sommasse i primi due consumi solamente su 9 bit.

Realizzazione del circuito in Blif

Per poter realizzare il circuito abbiamo creato la nostra FSM, descritta tramite la transizione degli stati scritti nel file blif. Una volta realizzata, attraverso *state_minimize stamina* abbiamo ottenuto la fsm equivalente minima, formata da 3 stati: *INIT*, *FASCIA*, *OV*, rinominati in seguito da noi. A questo punto attraverso il comando *state_assign*, *sis* ha assegnato una codifica agli stati e ha mappato il controllore trasformandolo in circuito sequenziale.

Osservazione: Data la disponibilità di due diversi algoritmi per quanto riguarda *state_assign*, ovvero *jedi* e *nova,* abbiamo testato entrambe le possibilità. Ottenendo i risultati qui sotto riportati abbiamo deciso di utillizzare la FMS con state_assign nova.

```
sis> rl fsm_min.blif
sis> sa jedi
Running jedi, written by Bill Lin,  UC Berkeley
sis> psf
FSM             pi= 4   po= 2   nodes=  4       latches= 2
lits(sop)=  56  #states(STG)=   3
sis> rl fsm_min.blif
sis> sa nova
Running nova, written by Tiziano Villa,  UC Berkeley
Warning: network `SISGAAa00052', node "v0" does not fanout
Warning: network `SISGAAa00052', node "v1" does not fanout
Warning: network `SISGAAa00052', node "v2" does not fanout
Warning: network `SISGAAa00052', node "v3" does not fanout
sis> psf
FSM             pi= 4   po= 2   nodes=  4       latches= 2
lits(sop)=  37  #states(STG)=   3
```

Statistiche del circuito

Una volta realizzata la FSMD e caricata su *sis*, abbiamo visualizzato le sue statistiche.
Successivamente abbiamo iniziato il processo di ottimizzazione fino ad ottenere la versione migliore possibile e rinominata come *FSMD.blif*.

```
sis> rl fsmd_base.blif
Warning: network `fsm_N.blif', node "RES_GEN" does not fanout
Warning: network `fsm_N.blif', node "FASCIA_1" does not fanout
Warning: network `fsm_N.blif', node "FASCIA_0" does not fanout
Warning: network `fsm_N.blif', node "ALERT" does not fanout
sis> psf
FSMD            pi=13   po= 5   nodes=349       latches=10
lits(sop)=2944
sis> rl FSMD.blif
sis> psf
FSMD            pi=13   po= 5   nodes= 90       latches= 8
lits(sop)= 405
```

Script Ottimizzazione

Eseguendo una prima mappatura del nostro circuito, caricando l'FSMD ottimizzata otteniamo le seguenti statistiche:

```
sis> rl FSMD.blif
sis> rlib synch.genlib
sis> map -m 0
sis> print_map_stats
Total Area              = 6504.00
Gate Count              = 210
Buffer Count            = 6
Inverter Count          = 42
Most Negative Slack     = -76.60
Sum of Negative Slacks  = -947.20
Number of Critical PO   = 13
```

Successivamente, abbiamo testato diverse possibilità per quanto riguardava l'ottimizzazione della nostra FSMD mappata, attraverso *script_rugged* e *full_simplify*, giungendo cosi alla stesura del nostro script di ottimizzazione[1], che attraverso una combinazione di questi comandi ci porta alla FSMD migliore possibile.

Statistiche finali elaborato

Lanciando lo script di ottimizzazione si ottiene la FSMD del progetto con le sue statistiche finali.

```
Total Area              = 6248.00
Gate Count              = 203
Buffer Count            = 7
Inverter Count          = 45
Most Negative Slack     = -80.40
Sum of Negative Slacks  = -999.40
Number of Critical PO   = 13
FSMD           pi=13   po= 5   nodes=203       latches= 8
lits(sop)= 481
```

[1] Lo script, con all'interno le combinazioni di comandi, lo si può trovare nella documentazione dell'elaborato

Considerazioni finali

In generale, ciò che ci ha guidati nella realizzazione del nostro elaborato, è stato, oltre al garantirci che il tutto funzionasse in ogni caso possibile e presentabile, la volontà di ottenere una macchina più efficiente ed ottimizzata possibile, ma allo stesso tempo anche flessibile e facilmente modificabile.
Si tratta infatti di una struttura aperta alle modifiche, che richiede poco impegno di risorse affinché vengano reindirizzati e cambiati segnali che permettono di variare il funzionamento della macchina.

Abbiamo, poi, eliminato i messaggi di Warning dovuti ai riporti dei sommatori, creando un sommatore a un bit apposito che non ha il riporto, e usato successivamente per la somma dell'ultimo bit dei sommatori a più bit.

Test dell'elaborato

Inizialmente abbiamo deciso di creare un centinaio di test per conto nostro, mirati a testare il corretto funzionamento, soprattutto dei casi particolari; inoltre, abbiamo anche creato un file .txt con gli output prima di sottoporre gli input alla macchina, in modo da fare un controllo successivo tra i risultati della stessa e quelli che avevamo previsto noi in precedenza[2].

Successivamente, però, non contenti, abbiamo pensato di creare anche un programma in C++, che creasse un numero grande qualsiasi di input casuali – decidibile modificando il valore di *K* nel sorgente "generatore_in_out.cpp" – e che poi calcolasse da solo i rispettivi output, creando, così, i file *test_input_EP_MS* e *test_output_MS_EP.txt*.
Il file di input è già sottoponibile come sorgente per *sis*, che dopo aver letto l'FSMD – comando già presente nel file dei test di input – calcolerà gli output dei *K* test presenti; anche questo è gestito dal nostro programma in C++, che ritorna nella cartella padre, e avvia sis, dicendo all'utente di lanciare due comandi. Il primo, *set sisout risultati.txt*, che farà scrivere gli output calcolati da sis sul file *risultati.txt*, e il secondo, *sr test_input_EP_MS*, che legge il sorgente con i file di input creato prima. Infine, uscendo da sis, il programma riprenderà il suo lavoro confrontando il file contente gli output calcolati da lui stesso e quello contenente gli output prodotti da sis. Se tutti gli output sono corretti, un messaggio a video lo confermerà, altrimenti verrà indicato il numero o i numeri degli output errati.

Per fare il tutto è necessario compilare il sorgente "generatore_in_out.cpp" e lanciare l'eseguibile.

[2] I file in questione sono *test_input* e *test_output.txt*