Università degli Studi di Verona

Facoltà di Scienze MM.FF.NN.

AA 2009/2010

Corso di laurea in Informatica

Elaborato SIS #1

Laboratorio di Architettura degli Elaboratori

*A cura di*

*Alessio Bogon – Davide Spadini*

*vr092349 – vr092576*

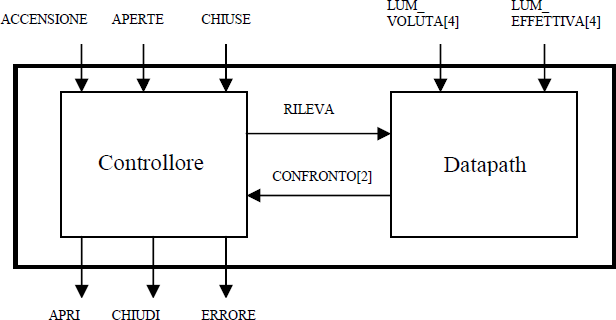
Indice

1. [Specifiche](#Specifiche)
2. [Introduzione](#Introduzione)
3. [Datapath](#Datapath)
   1. [Descrizione](#DatapathDescrizione)
   2. [Realizzazione in SIS](#DatapathRealizzazione)
4. [Controllore](#Controllore)
   1. [Descrizione](#ControlloreDescrizione)
   2. [Realizzazione in SIS](#ControlloreRealizzazione)
5. [Test](#Test)
   1. [Datapath](#TestDatapath)
   2. [Controllore](#TestControllore)
   3. [Centralina](#TestCentralina)
6. [Ottimizzazione](#Ottimizzazione)
   1. [Datapath](#OttimizzazioneDatapath)
   2. [Controllore](#OttimizzazioneControllore)
   3. [Centralina](#OttimizzazioneCentralina)
7. [Mapping tecnologico](#MappingTecnologico)

Specifiche

Si consideri il circuito sequenziale che pilota l'apertura/chiusura automatica delle tende di una sala. Il circuito ha 5 ingressi (ACCENSIONE, APERTE, CHIUSE, LUM\_VOLUTA[4], LUM\_EFFETTIVA[4]) e 3 uscite (APRI, CHIUDI, ERRORE). Il meccanismo automatico è guidato come segue:

* La luminosità desiderata viene impostata mediante un selettore a dieci posizioni che pilota il segnale di ingresso al circuito LUM\_VOLUTA[4], che in ogni istante rappresenta una delle dieci posizioni.
* Un sensore di luminosità assegna in ogni istante al segnale di ingresso LUM\_EFFETTIVA[4] il valore di luminosità presente in sala descrivendolo con una scala di 10 valori.
* Il confronto tra i due segnali sopra descritti viene calcolato in un datapath che comunica al controllore con il segnale CONFRONTO[2] la relazione tra LUM\_VOLUTA[4] e LUM\_EFFETTIVA[4]. (*Si assuma che 01 abbia il significato di maggiore e 10 di minore*).
* Nel caso in cui la luminosità voluta sia inferiore alla luminosità effettiva vengono comandate le tende, con il segnale di uscita del circuito CHIUDI, che permette di chiuderle. Nel caso in cui la luminosità voluta sia superiore alla luminosità effettiva vengono comandate le tende, con il segnale APRI, che permette di aprirle.
* I segnali di ingesso APERTE e CHIUSE valgono 1 se le tende sono completamente aperte o chiuse.
* Il confronto delle luminosità viene effettuato solo se il circuito è acceso (segnale di ingresso ACCESO uguale a 1) tramite il segnale RILEVA da controllore a datapath altrimenti tutti i segnali di uscita del circuito vengono posti a zero.
* Se il confronto richiede di aprire ulteriormente le tende, ma le tende sono già aperte completamente, il circuito non deve forzarne ulteriormente l’apertura. La stessa cosa è richiesta per il caso di chiusura.
* Ogniqualvolta c’e’ un’incongruenza tra i segnali di ingresso (ad esempio, segnali APERTE e CHIUSE uguali a 1 contemporaneamente) il circuito deve andare in uno stato di errore e porre il segnale di uscita ERRORE uguale a 1. Il circuito può essere ripristinato dallo stato di errore spegnendo e riaccendendo il circuito



Introduzione

Questa relazione viene fornita assieme ai sorgenti .blif del progetto.

Il progetto è strutturato in due cartelle:

* ./non\_ottimizzato, cartella in cui sono presenti i sorgenti non ottimizzati, utili solo per capire come è stato strutturato il progetto;
* ./ottimizzato, cartella in cui sono presenti i sorgenti ottimizzati, e per questo motivo molti sono incomprensibili.

I file più importanti (contenuti in ogni sottodirectory) sono i seguenti:

* centralina.blif: è il file contenete il circuito completo che raggruppa controllore.blif e datapath.blif;
* controllore\_raw.blif: è il controllore prima dell’assegnamento degli stati;
* controllore.blif: è il controllore dopo l’assegnamento degli stati;
* datapath.blif: è il datapath;
* simulator.script: è uno script creato per testare il circuito.

# Ipotesi aggiuntive

Assumiamo che “incongruenze tra gli ingressi” significhi che gli ingressi APERTE e CHIUSE non possano valere 1 nello stesso momento e che il valore delle luminosità non possa superare il valore decimale 10.

Inoltre il segnale RILEVA vale 1 se e solo se l’ingresso ACCENSIONE vale 1.

Per l’ottimizzazione del circuito sceglieremo di minimizzare per area piuttosto che per ritardo in quanto non è necessario che il dispositivo abbia una bassa latenza, importa invece farlo meno costoso.

Datapath

Analizziamo il comportamento del datapath.

Il datapath deve restituire il confronto tra gli ingressi LUM\_VOLUTA e LUM\_EFFETTIVA solo quando l’ingresso RILEVA è alto e solo se non sono presenti incongruenze tra gli ingressi (ad esempio se LUM\_EFFETTIVA = 1110b, il valore è oltre la soglia consentita (1010b) e quindi va considerato come un errore).

4

>

**LUM\_VOLUTA**

**LUM\_EFFETTIVA**

1010

=

>

>

0 MUX 1

GTO

UGO

RIS1

RIS0

ERR

ERR1

ERR0

**RILEVA**

SEL

00

4

4

2

**CONFRONTO**

# Descrizione

Il datapath fondamentalmente dovrà leggere i 4 bit di LUM\_VOLUTA e i 4 bit di LUM\_EFFETTIVA e restituire il confronto a 2 bit sull’uscita CONFRONTO, solo se l’ingresso RILEVA vale 1.

Le specifiche impongono un controllo degli errori sugli ingressi, quindi il datapath dovrà anche occuparsi di segnalare gli eventuali errori.

L’idea è di portare in uscita il confronto degli ingressi se essi non sono errati e se RILEVA vale 1, in caso contrario l’uscita CONFRONTO varrà 00 (situazione di errore o non devo rilevare).

Abbiamo bisogno dei seguenti componenti:

* 1x : Multiplexer a 1 bit di selezione e 2 bit per ogni ingresso. Esso servirà a decidere se portare in uscita il confronto tra le due luminosità o il codice di errore/circuito spento 00;

MUX

* 3x : Confronta gli ingressi e vale 1 se il primo ingresso è maggiore del secondo, 0 altrimenti. Esso servirà a verificare se LUM\_VOLUTA > LUM\_EFFETTIVA e se vi sono ingressi oltre la soglia di errore;

>

* 1x : Confronta gli ingressi e vale 1 se sono uguali, 0 altrimenti. Esso servirà a verificare se gli ingressi LUM\_VOLUTA e LUM\_EFFETTIVA sono uguali;

=

* 4x : porta AND;
* 4x: porta OR;
* 6x : porta NOT;

Esaminiamo ora tutti i segnali del circuito, interni ed esterni.

* Ingressi:
  + LUM\_VOLUTA: 4 bit, collegato al selettore della luminosità desiderata. Il valore è codificato in modulo. Il valore è valido solo se ≤ 10;
  + LUM\_EFFETTIVA: 4 bit, collegato al sensore di luminosità. Il valore si assume è codificato in modulo. Il valore è valido solo se ≤ 10;
  + RILEVA: 1 bit, collegato all’uscita RILEVA del controllore.
* Uscite:
  + CONFRONTO: 2 bit, collegato all’ingresso CONFRONTO del controllore.
* Segnali interni:
  + Valori costanti (1010 e 00): questi sono implementati nei file uno.blif e zero.blif;
  + GTO (Greater Than – Output): 1 bit, indica se LUM\_VOLUTA > LUM\_EFFETTIVA;
  + UGO(UGuale – Output): 1 bit, indica se LUM\_VOLUTA = LUM\_EFFETTIVA;
  + RIS1, RIS0: 1 bit, indicano il confronto tra LUM\_VOLUTA e LUM\_EFFETTIVA, e valgono rispettivamente:
    - 01 se LUM\_VOLUTA > LUM\_EFFETTIVA
    - 11 se LUM\_VOLUTA = LUM\_EFFETTIVA
    - 10 se LUM\_VOLUTA < LUM\_EFFETTIVA
  + ERR1, ERR0: 1 bit, indicano se gli ingressi LUM\_VOLUTA e LUM\_EFFETTIVA non sono validi;
  + ERR: 1 bit, è la OR tra ERR1 e ERR0. Indica dunque se si è verificata una situazione di errore;
  + SEL: 1 bit, è il bit di selezione del multiplexer. Quando il bit vale 0 (RILEVA=1 e ERR=0) allora l’uscita sarà collegata al confronto tra le due luminosità, altrimenti in uscita avremo il codice di errore/circuito spento 00.

## Multiplexer

Il multiplexer è un componente che serve a portare in uscita uno degli ingressi, opportunamente scelto dall’ingresso di selezione. Quello presente nel circuito è un multiplexer a 2 ingressi da 2 bit ciascuno, un bit di selezione e, naturalmente, 2 bit di uscita. Il componente viene realizzato nel file mux2.blif.

## Comparatore (>) e (=)

Questi componenti eseguono rispettivamente le operazione “maggiore di” e “uguale a” su due ingressi da 4 bit ciascuno. Gli ingressi si suppongono codificati in modulo e gli output varranno 1 se la condizione è soddisfatta, 0 altrimenti.

## Parte di logica combinatoria

La parte di logica combinatoria indicata dal rettangolo tratteggiato è stata ricavata la seguente tabella di verità:

|  |  |  |  |
| --- | --- | --- | --- |
| GTO | UGO | RIS1 | RIS0 |
| 0 | 0 | **1** | **0** |
| 0 | 1 | **1** | **1** |
| 1 | 0 | **0** | **1** |
| 1 | 1 | **-** | **-** |

La descrizione è alquanto banale e dunque non verrà trattata.

Riportiamo invece una tabella semplificata per comprendere meglio il funzionamento del circuito nel suo insieme:

|  |  |  |  |
| --- | --- | --- | --- |
| Ingressi validi | Confronto | RILEVA | Uscita |
| SI | > | 1 | 01 |
| SI | = | 1 | 11 |
| SI | < | 1 | 10 |
| SI | - | 0 | 00 |
| NO | - | - | 00 |

# Realizzazione in SIS

Per la realizzazione del datapath in SIS sono stati creati 12 file blif:

* datapath.blif: collega i componenti tra loro;
* gt4.blif: realizza la funzione “maggiore di”;
* uguale4.blif: realizza la funzione “uguale a”;
* combinatorio.blif: è la parte di logica combinatoria indicata dal rettangolo tratteggiato;
* mux2.blif: è il multiplexer a 1 bit di selezione e 2 bit per ogni ingresso;
* and.blif, or.blif, not.blif, xor.blif, xnor.blif: sono le porte logiche di base necessarie ai vari componenti per funzionare;
* uno.blif, zero.blif: sono le costanti 0 e 1.

Come già detto, tutti i file descritti nella versione grezza possono essere trovati nella cartella ./non\_ottimizzati.

Controllore

Il controllore è la parte che si occuperà di aprire o chiudere le tende, a seconda della luminosità desiderata. Esso deve anche controllare se vi sono incongruenze tra gli ingressi, andando eventualmente nello stato di errore.

I={ACCENSIONE, APERTE, CHIUSE, CONFRONTO1, CONFRONTO0}

O={RILEVA, APRI, CHIUDI, ERRORE}

10-01\1100

1-010\1010

10000

10100

11000 1000

111--

10011

10110

10111 1000

11001

11011

1----\1001

0----\0000

10-01\1100

1-010\1010

10011

10110

10111 1000

11001

11011

10000

10100

11000 1000

111--

0----\0000

# Descrizione

Il controllore ha i seguenti ingressi:

* ACCENSIONE: è il bit che indica se il circuito è acceso (1) oppure spento (0);
* APERTE: è il bit che indica se le tende sono completamente aperte (1) o meno (0);
* CHIUSE: è il bit che indica se le tende sono completamente chiuse (1) o meno (0);
* CONFRONTO1, CONFRONTO0: sono i due bit di uscita del datapath già descritti sopra;

e le seguenti uscite:

* RILEVA: indica al datapath se effettuare il confronto tra le luminosità (1) o meno (0). Esso è collegato all’ingresso ACCENSIONE, come da specifiche del problema;
* APRI: è il primo bit che comanda il meccanismo di apertura/chiusura delle tende. Se vale 1 le tende si aprono;
* CHIUDE: è il secondo bit che comanda il meccanismo di apertura/chiusura delle tende. Se vale 1 le tende si chiudono;
* ERRORE: segnala se si è verificato un errore.

Il controllore ha tre stati:

* ACCESO: è lo stato principale in cui viene controllato il meccanismo delle tende. A seconda degli ingressi esso sceglierà se aprirle, chiuderle o non alterarle. Questo funzionerà nel seguente modo:
  + Apre le tende quando gli ingressi valgono 10-01, ovvero quando CONFRONTO = 01 (luminosità voluta > luminosità effettiva), le tende non sono completamente aperte e non importa se siano completamente chiuse;
  + Chiude le tende quando gli ingressi valgono 1-010, ovvero quando CONFRONTO = 10 (luminosità voluta < luminosità effettiva), le tende non sono completamente chiuse e non importa se siano completamente aperte;
  + Lascia le tende invariate quando gli ingressi valgono:
    - 10011, 10111, 11011, ovvero confronto = 11 (luminosità voluta = luminosità effettiva);
    - 11001, ovvero quando voglio aprirle ma sono già completamente aperte;
    - 10110, ovvero quando voglio chiuderle ma sono già completamente chiuse.
  + Va nello stato di ERRORE quando vi sono incongruenze tra gli ingressi, ovvero quando CONFRONTO = 00 (situazione di errore tra gli ingressi LUM\_VOLUTA o LUM\_EFFETTIVA) oppure quando gli ingressi segnalano tende completamente aperte e completamente chiuse;
  + Va nello stato SPENTO quando l’ingresso ACCENSIONE vale 0;
* ERRORE: è lo stato che mantiene le uscite APRI e CHIUDI a 0 e l’uscita ERRORE a 1 finché il circuito non viene spento (ingresso ACCENSIONE = 0);
* SPENTO: è lo stato che mantiene tutte le uscite pari a 0 finchè non viene riattivato il circuito (ACCENSIONE = 1). Non appena il circuito viene riattivato, esso fa un controllo su tutti gli ingressi e si comporta in modo del tutto analogo allo stato ACCESO.

# Realizzazione con SIS

Per la realizzazione del controllore in SIS sono stati creati 2 file blif, che ora analizzeremo:

* controllore\_raw.blif: contiene la FSM realizzata con il tool BVE. Per poter testare la FSM con il file centralina.blif dobbiamo prima assegnare gli stati tramite il comando di SIS “state\_assign jedi”.
* controllore.blif: è la FSM dopo aver eseguito il comando “state\_assign jedi”. È necessario avere gli stati codificati affinchè sia possibile testare il circuito completo con centralina.blif.

Come già detto, tutti i file descritti nella versione grezza possono essere trovati nella cartella ./non\_ottimizzati.

Test

Procediamo ora al test dei vari componenti del circuito.

# Datapath

Simuliamo il comportamento del datapath con degli ingressi validi, ingressi non validi e nel caso in cui RILEVA = 0.

|  |
| --- |
| **sis> simulate 1 0 0 0 1 1 0 0 0 # LV=1, LE=8**  Network simulation:  Outputs: 1 0  Next state:  **sis> simulate 1 0 0 1 0 0 0 1 0 # LV=2, LE=2**  Network simulation:  Outputs: 1 1  Next state:  sis> simulate 1 0 1 1 1 0 0 0 1 # LV=7, LE=1  Network simulation:  Outputs: 0 1  Next state:  **sis> simulate 1 1 0 1 0 0 0 1 0 # LV=10, LE=2**  Network simulation:  Outputs: 0 1  Next state:  **sis> simulate 1 1 0 1 1 0 0 1 1 # LV=11, LE=3**  Network simulation:  Outputs: 0 0  Next state:  **sis> simulate 0 0 1 0 1 0 0 1 1 # spento**  Network simulation:  Outputs: 0 0  Next state: |

# Controllore

|  |
| --- |
| **sis> read\_blif controllore\_raw.blif**  **sis> simulate 1 0 0 1 0 # LV < LE, chiudi**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 1 0  Next state: ACCESO ((null))  **sis> simulate 1 0 0 0 1 # LV > LE, apri**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 1 0 0  Next state: ACCESO ((null))  **sis> simulate 1 0 0 1 1 # LV = LE, resta fermo**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 0 0  Next state: ACCESO ((null))  **sis> simulate 1 1 0 0 1 # LV > LE, tende APERTE, resta fermo**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 0 0  Next state: ACCESO ((null))  **sis> simulate 1 0 1 1 0 # LV < LE, tende CHIUSE, resta fermo**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 0 0  Next state: ACCESO ((null))  **sis> simulate 1 0 0 0 0 # ingressi luminosità non validi, errore**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 0 1  Next state: ERRORE ((null))  **sis> simulate 1 0 1 1 0 # circuito acceso, errore**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 0 1  Next state: ERRORE ((null))  **sis> simulate 0 0 1 1 0 # circuito spento**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 0 0 0 0  Next state: SPENTO ((null))  **sis> simulate 1 0 0 1 0 # acceso, LV < LE, chiudi**  Network simulation:  Outputs: 0 0 0 0  Next state:  STG simulation:  Outputs: 1 0 1 0  Next state: ACCESO ((null)) |

# Centralina

La simulazione della centralina viene effettuata tramite lo script simulator.script.

|  |
| --- |
| **sis> source simulator.script**  acceso, LUM\_VOLUTA = 1, LUM\_EFFETTIVA = 8 ==> voglio CHIUDERE  Network simulation:  Outputs: 0 1 0  Next state: 00  ---------------------  acceso, LUM\_VOLUTA = LUM\_EFFTTIVA = 2 ==> resto FERMO  Network simulation:  Outputs: 0 0 0  Next state: 00  ---------------------  acceso, LUV\_VOLUTA = 7, LUM\_EFFETTIVA = 1 ==> voglio APRIRE  Network simulation:  Outputs: 1 0 0  Next state: 00  ---------------------  (x2) acceso, tende APERTE, LUM\_VOLUTA = 10, LUM\_EFFETTIVA = 2 ==> resto FERMO  Network simulation:  Outputs: 0 0 0  Next state: 00  Network simulation:  Outputs: 0 0 0  Next state: 00  ---------------------  acceso, LUM\_VOLUTA = 11, LUM\_EFFETTIVA = 3 ==> ERRORE  Network simulation:  Outputs: 0 0 1  Next state: 01  ---------------------  acceso ==> ERRORE  Network simulation:  Outputs: 0 0 1  Next state: 01  ---------------------  spento, ==> SPENTO  Network simulation:  Outputs: 0 0 0  Next state: 10  ---------------------  acceso, LUM\_VOLUTA = 2, LUM\_EFFETTIVA = 3 ==> voglio CHIUDERE  Network simulation:  Outputs: 0 1 0  Next state: 00 |

Ottimizzazione

Il circuito così ottenuto va ottimizzato, cioè si deve ridurre il numero di componenti e/o il ritardo senza però alterarne il funzionamento . Per ottimizzare i circuiti SIS fornisce vari comandi come sweep, eliminate, resub, fx, simplify e full\_simplify: tuttavia nella maggior parte dei casi lo script “script.rugged”, il quale utilizza tutte queste funzioni, permette di ottenere i risultati migliori.

Verrà utilizzato inoltre il comando “print\_stats” dopo ogni modifica per monitorare le statistiche del circuito. Prima di ottimizzare il circuito interamente, eseguiremo le operazioni di minimizzazione prima sul datapath e poi sul controllore perché abbiamo constatato che il circuito viene minimizzato ulteriormente.

# Datapath

|  |
| --- |
| **sis> set autoexec print\_stats # Visualizza le statistiche**  **sis> read\_blif datapath.blif # Leggo datapath.blif**  Warning: network `DATAPATH', node "UNO" does not fanout  DATAPATH pi= 9 po= 2 nodes= 39 latches= 0  lits(sop)= 153  **sis> source script.rugged**  # OUTPUT  **sis> source script.rugged**  # OUTPUT  **sis> source script.rugged**  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 6 latches= 0  lits(sop)= 62  DATAPATH pi= 9 po= 2 nodes= 9 latches= 0  lits(sop)= 45  DATAPATH pi= 9 po= 2 nodes= 9 latches= 0  lits(sop)= 45  DATAPATH pi= 9 po= 2 nodes= 9 latches= 0  lits(sop)= 45  DATAPATH pi= 9 po= 2 nodes= 7 latches= 0  lits(sop)= 49  DATAPATH pi= 9 po= 2 nodes= 7 latches= 0  lits(sop)= 49  DATAPATH pi= 9 po= 2 nodes= 7 latches= 0  lits(sop)= 49  DATAPATH pi= 9 po= 2 nodes= 7 latches= 0  lits(sop)= 49 |

Utilizzando “script.rugged” per 3 volte il numero di letterali del datapath è passato da 153 prima dell’ottimizzazione a 49 e il numero di nodi da 39 a 7. Ogni altra esecuzione dello script non ha ottimizzato il circuito ulteriormente. Per curiosità abbiamo provato ad ottimizzare mediante il solo comando full\_simplify (algoritmo di Quine-McCluskey) e il numero di letterali è stato ridotto a 74 mentre il numero di nodi è rimasto 39.

# Controllore

|  |
| --- |
| **sis> set autoexec print\_stats**  **sis> read\_blif controllore\_raw.blif**  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 0  lits(sop)= 0 #states(STG)= 3  **sis> state\_minimize stamina # Minimizzazione degli stati**  Running stamina, written by June Rho, University of Colorado at Boulder  Number of states in original machine : 3  Number of states in minimized machine : 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 0  lits(sop)= 0 #states(STG)= 2  **sis> state\_assign jedi # Assegnamento degli stati**  Running jedi, written by Bill Lin, UC Berkeley  CONTROLLORE pi= 5 po= 4 nodes= 5 latches= 1  lits(sop)= 63 #states(STG)= 2  **sis> source script.rugged**  CONTROLLORE pi= 5 po= 4 nodes= 5 latches= 1  lits(sop)= 63 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 5 latches= 1  lits(sop)= 63 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 5 latches= 1  lits(sop)= 20 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 5 latches= 1  lits(sop)= 20 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2  CONTROLLORE pi= 5 po= 4 nodes= 4 latches= 1  lits(sop)= 19 #states(STG)= 2 |

Il primo comando per minimizzare la macchina a stati finiti è stato “state\_minimize stamina”, il quale cerca di minimizzare gli stati utilizzando l’algoritmo di Paull-Unger per fsm completamente specificate. L’esito è stato positivo, infatti il numero degli stati si è ridotto da 3 a 2. Questo perché lo stato SPENTO e ACCESO sono equivalenti, infatti è facilmente verificabile che per questi ultimi la funzione di stato prossimo è equivalente e la funzione di uscita è uguale. Può sembrare che passare da 3 a 2 stati non sia un grande miglioramento, ma in realtà lo è. Infatti con 3 stati il numero di latch necessari a memorizzare lo stato era , mentre con 2 stati sono necessari solo latch.

Il secondo comando è stato “state\_assign jedi”. Esso non ottimizza la FSM ma serve a convertirla in un circuito sequenziale. L’ultimo comando è la chiamata a script.rugged che ha ridotto il numero di letterali da 63 a 19 e il numero di nodi da 5 a 4.

# Centralina

|  |
| --- |
| **sis> set autoexec print\_stats**  **sis> read\_blif centralina.blif**  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  **sis> source script.rugged**  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 9 latches= 1  lits(sop)= 85  CENTRALINA pi=11 po= 3 nodes= 9 latches= 1  lits(sop)= 85  CENTRALINA pi=11 po= 3 nodes= 9 latches= 1  lits(sop)= 85  CENTRALINA pi=11 po= 3 nodes= 13 latches= 1  lits(sop)= 62  CENTRALINA pi=11 po= 3 nodes= 13 latches= 1  lits(sop)= 62  CENTRALINA pi=11 po= 3 nodes= 13 latches= 1  lits(sop)= 62  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66  CENTRALINA pi=11 po= 3 nodes= 11 latches= 1  lits(sop)= 66 |

Lo script.rugged in questo caso non è riuscito a ottimizzare ancor più la centralina, in quanto i singoli componenti sono già stati ottimizzati utilizzando più volte il summenzionato script.

Mapping tecnologico

In quest’ultima fase del progetto vediamo come è possibile mappare il circuito ottimizzato su una libreria tecnologica. La consegna specifica di usare la “synch.genlib” integrata all’interno di SIS.

Si può scegliere di mappare il circuito ottimizzando l’area oppure il ritardo. Il circuito deve pilotare l’apertura e la chiusura delle tende e quindi preferiamo avere un costo minore dello stesso piuttosto che sia più reattivo. Scegliamo dunque di mappare ottimizzando rispetto all’area piuttosto che al ritardo. Le istruzioni necessarie generalmente sono le seguenti:

* read\_library <nome\_libreria>: importa la libreria tecnologica. Nel nostro caso useremo read\_library synch.genlib;
* map <parametri>: mappa il circuito sulla libreria specificata. I parametri principali sono:
  + -m 0: mappa il circuito ottimizzando rispetto all’area;
  + -n 1: mappa il circuito ottimizzandolo rispetto al ritardo;
  + -s: visualizza le statistiche del circuito mappato. Esso mostrerà anche l’area e il ritardo massimo;
  + -W: evita la visualizzazione dei warning.
* reduce\_depth: cerca di ridurre il ritardo a discapito dell’area.

# Realizzazione in SIS

|  |
| --- |
| **sis> read\_blif centralina.blif # Carico la centralina**  **sis> read\_library synch.genlib # Carico la libreria**  **sis> map –W -m 0 –s # Mappa ottimizzando per area, visualizza # output e non mostrare warning**  >>> before removing serial inverters <<<  # of outputs: 4  total gate area: 960.00  maximum arrival time: (14.80,14.80)  maximum po slack: (-12.20,-12.20)  minimum po slack: (-14.80,-14.80)  total neg slack: (-54.00,-54.00)  # of failing outputs: 4  >>> before removing parallel inverters <<<  # of outputs: 4  total gate area: 960.00  maximum arrival time: (14.80,14.80)  maximum po slack: (-12.20,-12.20)  minimum po slack: (-14.80,-14.80)  total neg slack: (-54.00,-54.00)  # of failing outputs: 4  # of outputs: 4  total gate area: 944.00  maximum arrival time: (14.80,14.80)  maximum po slack: (-12.20,-12.20)  minimum po slack: (-14.80,-14.80)  total neg slack: (-54.00,-54.00)  # of failing outputs: 4 |

Il circuito finale ha un area pari a 944 e un ritardo massimo pari a 14.80. La configurazione è stata salvata nel file centralina\_final.blif presente in ognuna delle due sottodirectory.

Il circuito è ora pronto a essere stampato.