Lez. 6 – Progetto Controllore - Datapath

Laboratorio di Architettura degli Elaboratori

Michele Lora

21 dicembre 2023

Outline

Outline

Modelli su più file

Per includere un componente all'interno di un modello blif usare le seguenti istruzioni:

```
.subckt nomeComp paramFormale=paramAttuale ...

.search nomeFileComp.blif
```

Registro a 1 bit

.latch <input> <output> <type> <control> <init-val>

- input è l'ingresso del latch.
- output l'uscita del latch.
- type può essere fe, re, ah, al, as, che corrispondono a "falling edge," "rising edge," "active high," "active low," or "asynchronous."
- control è il segnale di clock per il latch. Può essere un clock del modello, l'uscita di una qualsiasi funzione del modello, o la parola "NIL" per nessun clock interno. Ciò significa che il registro utilizza il clock generale del circuito in cui è inserito.
- init-val è lo stato iniziale del latch, che può essere 0, 1, 2, 3. "2" indica "don't care" e "3" è "unknown." o non specificato.

Registro a 1 bit

```
.model REGISTRO
.inputs A
.outputs O
.latch A O re NIL 0
.end
```

Registro a 4 bit

```
.model REGISTRO4
.inputs A3 A2 A1 A0
.outputs O3 O2 O1 O0
.subckt REGISTRO A=A3 O=O3
.subckt REGISTRO A=A2 O=O2
.subckt REGISTRO A=A1 O=O1
.subckt REGISTRO A=A0 O=O0
.search registro.blif
.end
```

Sommatore

```
.model SOMMATORE
.inputs A B CIN
.outputs O COUT
.names A B K
10 1
01 1
.names K CIN O
10 1
01 1
.names A B CIN COUT
11- 1
1-1 1
-11 1
.end
```

Sommatore a 2 bit

```
.model SOMMATORE2
.inputs A1 A0 B1 B0 CIN
.outputs O1 O0 COUT
.subckt SOMMATORE A=A0 B=B0 CIN=CIN O=O0 COUT=C0
.subckt SOMMATORE A=A1 B=B1 CIN=C0 O=O1 COUT=COUT
.search sommatore.blif
.end
```

Multiplexer a 4 ingressi 1 bit ciascuno

```
.model MUX1
.inputs S1 S0 i3 i2 i1 i0
.outputs out
.names S1 S0 i3 i2 i1 i0 out
001--- 1
01-1-- 1
10--1- 1
11---1 1
.end
```

Multiplexer a 4 ingressi 2 bit ciascuno

```
.model MUX2
.inputs X1 X0 a1 a0 b1 b0 c1 c0 d1 d0
.outputs o1 o0
.subckt MUX1 S1=X1 S0=X0 i3=a1 i2=b1 i1=c1 i0=d1 out=o1
.subckt MUX1 S1=X1 S0=X0 i3=a0 i2=b0 i1=c0 i0=d0 out=o0
.search mux1.blif
.end
```

Multiplexer a 2 ingressi da 3 bit ciascuno

```
.model MUX3
.inputs A2 A1 A0 B2 B1 B0 S
.outputs O2 O1 O0
.names S A2 B2 O2
11- 1
0-1 1
.names S A1 B1 O1
11- 1
0-1 1
.names S A0 B0 O0
11- 1
0-1 1
.end
```

Demultiplexer a 1 bit e 4 uscite

```
.model DEMUX
.inputs S1 S0 IN
.outputs X Y Z W
.names S1 S0 IN X
001 1
.names S1 S0 IN Y
011 1
.names S1 S0 IN Z
101 1
.names S1 S0 IN W
111 1
.end
```

Comparatore a 4 bit

```
.model UGUALE4
.inputs A3 A2 A1 A0 B3 B2 B1 B0
.outputs O
.subckt xnor A=A3 B=B3 X=X3
.subckt xnor A=A2 B=B2 X=X2
.subckt xnor A=A1 B=B1 X=X1
.subckt xnor A=A0 B=B0 X=X0
.names X3 X2 X1 X0 O
1111 1
.search xnor.blif
.end
```

“Maggiore” a 6 bit

```
.model 6gt
.inputs A5 A4 A3 A2 A1 A0 B5 B4 B3 B2 B1 B0
.outputs AgtB
.subckt xor A=A5 B=B5 X=X5
.subckt xor A=A4 B=B4 X=X4
.subckt xor A=A3 B=B3 X=X3
.subckt xor A=A2 B=B2 X=X2
.subckt xor A=A1 B=B1 X=X1
.subckt xor A=A0 B=B0 X=X0
.names A5 A4 A3 A2 A1 A0 X5 X4 X3 X2 X1 X0 AgtB
1-----1----- 1
-1----01---- 1
--1---001--- 1
---1--0001-- 1
----1-00001- 1
-----1000001 1
.search xor.blif
```

“Minore-uguale” a 6 bit

```
.model 6le
.inputs C5 C4 C3 C2 C1 C0 D5 D4 D3 D2 D1 D0
.outputs CleD
.subckt 6gt A5=C5 A4=C4 A3=C3 A2=C2 A1=C1 A0=C0 ...
           B5=D5 B4=D4 B3=D3 B2=D2 B1=D1 B0=D0 AgtB=z
.names z CleD
0 1
.search 6gt.blif
.end
```

Outline

Modellazione di una FSMD in SIS

1. Modellare la FSM del Controllore mediante la tabella delle transizioni e assegnare la codifica agli stati con `state_assign jedi`
2. Salvare il Controllore ottenuto dopo la codifica degli stati in un file diverso da quello originale usando il comando `write_blif nomefile`
3. Modellare il Datapath come una interconnessione di componenti funzionali quali sommatori, moltiplicatori, multiplexer, registri, ecc.
4. Inglobare il Controllore e il Datapath in un unico file blif come se fossero due componenti, collegando opportunamente i segnali di comunicazione. E' importante specificare con la direttiva `.search` il file del Controllore ottenuto dopo la codifica degli stati e non quello scritto a mano.

Esempio: Semaforo con Priorità Temporizzato

Si consideri il classico esempio di un semaforo posto all'incrocio tra una strada principale (direttrice Nord-Sud, NS) e una secondaria (direttrice Est-Ovest, EO). Il semaforo può assumere solo i colori verde e rosso ed è dotato di sensori che rilevano la presenza di traffico. Il circuito che controlla il semaforo ha 2 segnali di ingresso forniti dai sensori (TRAFFICONS e TRAFFICOEO) e due di uscita (LUCENS e LUCEEO) con il seguente significato:

- TRAFFICONS[1]: segnala la presenza di traffico lungo la direttrice NS assumendo il valore 1
- TRAFFICOEO[1]: segnala la presenza di traffico lungo la direttrice EO assumendo il valore 1
- LUCENS[1]: deve essere posto a 1 per accendere la luce verde sulla strada NS. Se viene posto a 0 si accende la luce rossa.
- LUCEEO[1]: deve essere posto a 1 per accendere la luce verde sulla strada EO. Se viene posto a 0 si accende la luce rossa.

Esempio: Semaforo con Priorità Temporizzato

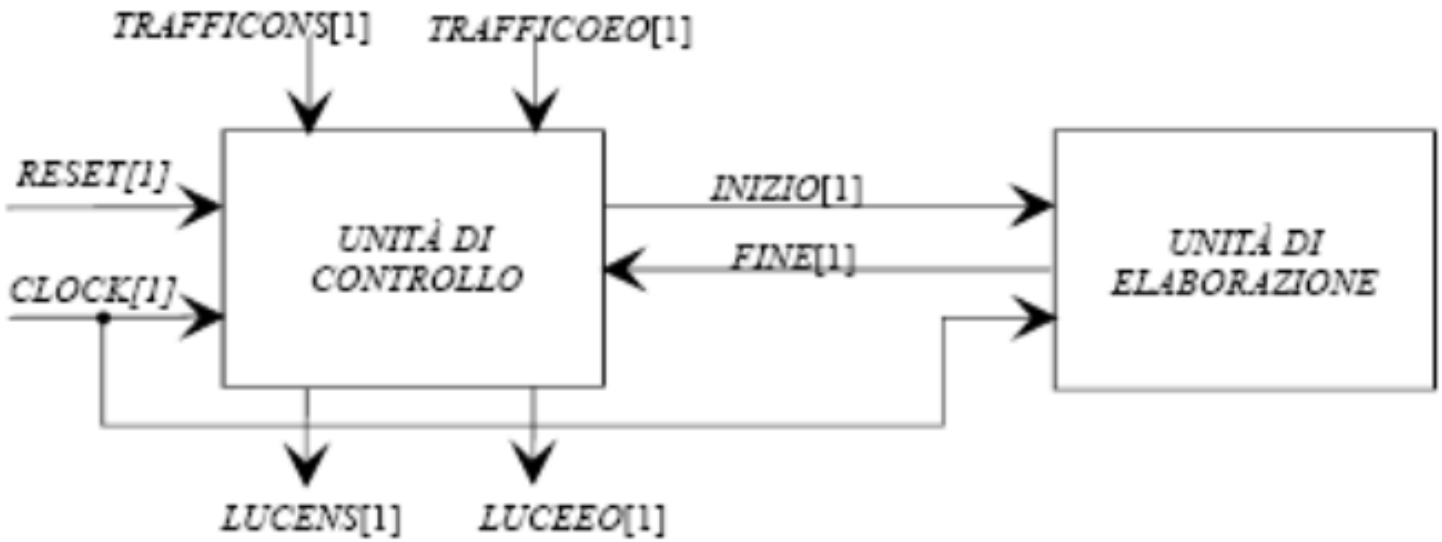
Per evitare incidenti, il circuito di controllo deve garantire che le luci sulle strade NS e EO siano sempre accese in opposizione. Il circuito assegna priorità alla strada NS e commuta dal verde al rosso su NS solo se TRAFFICONS=0 e TRAFFICOEO=1, in caso contrario mantiene il verde su NS. In assenza di traffico sia su NS che su EO il semaforo non modifica la configurazione raggiunta. Non appena giunge traffico su NS, indipendentemente da cosa succede su EO, il semaforo assegna la luce verde a NS e la luce rossa a EO.

Esempio: Semaforo con Priorità Temporizzato

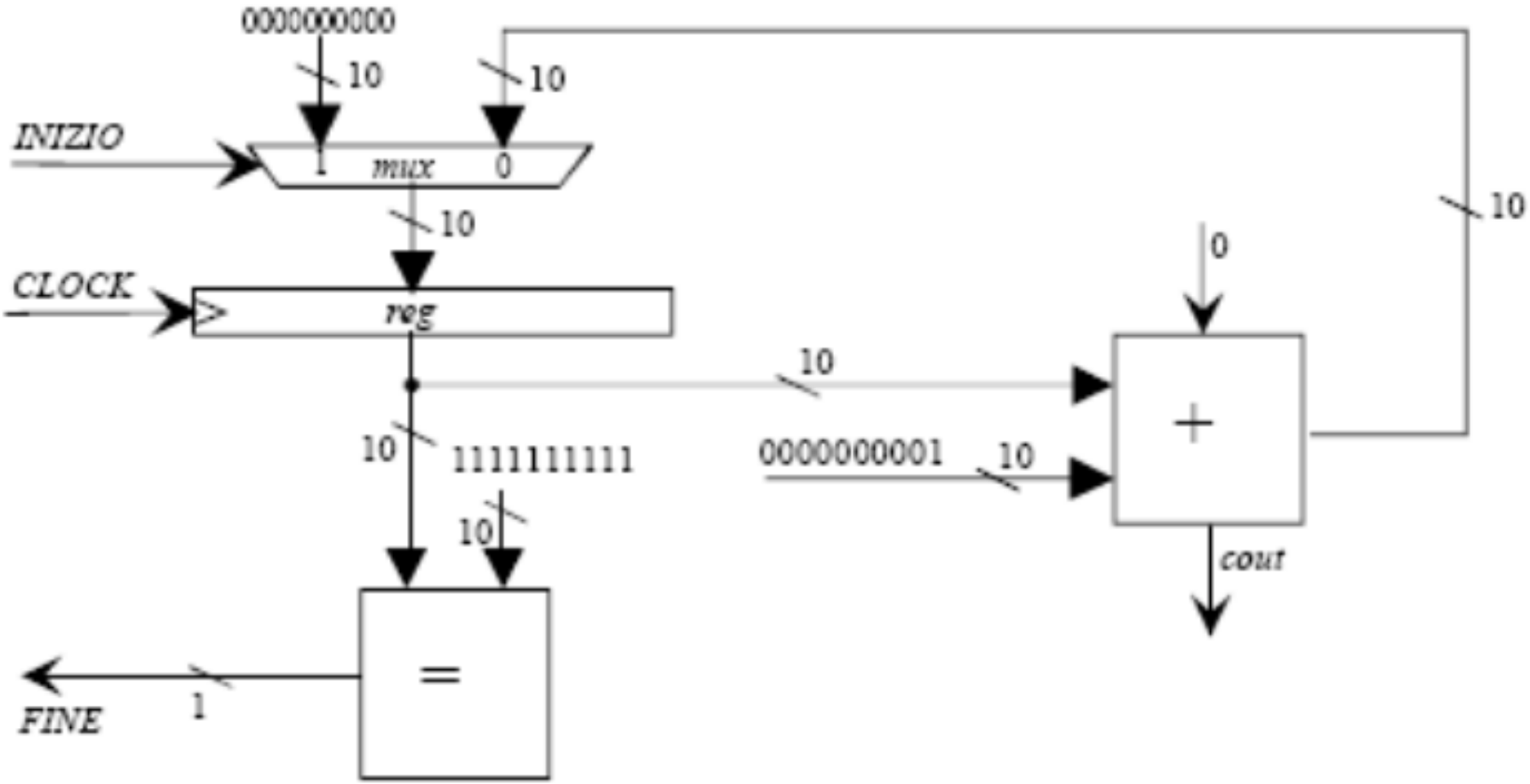
Per evitare incidenti, il circuito di controllo deve garantire che le luci sulle strade NS e EO siano sempre accese in opposizione. Il circuito assegna priorità alla strada NS e commuta dal verde al rosso su NS solo se TRAFFICONS=0 e TRAFFICOEO=1, in caso contrario mantiene il verde su NS. In assenza di traffico sia su NS che su EO il semaforo non modifica la configurazione raggiunta. Non appena giunge traffico su NS, indipendentemente da cosa succede su EO, il semaforo assegna la luce verde a NS e la luce rossa a EO.

Il controllore non commuta mai un semaforo da verde a rosso senza aver atteso 1024 cicli di clock.

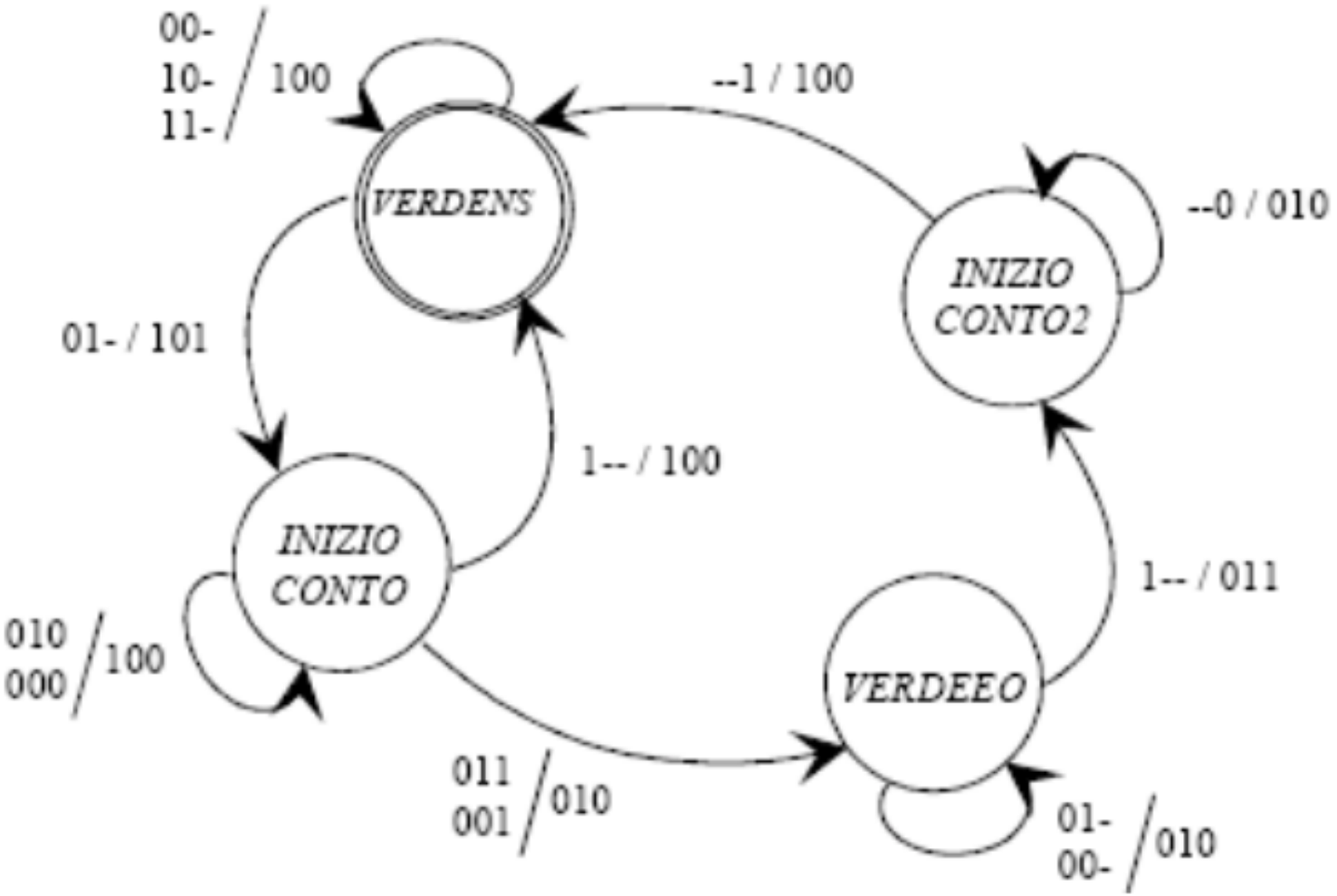
Esempio: Semaforo con Priorità Temporizzato



Esempio: Semaforo con Priorità Temporizzato



Esempio: Semaforo con Priorità Temporizzato



Sintesi del sistema

Sintesi della macchina a stati:

```
sis> read_blif controllo_specifica_manuale.blif
sis> state_minimize stamina
sis> state_assign jedi
sis> stg_to_network
sis> write_blif controllo.blif
```

Sintesi dell'intero sistema:

```
sis> read_blif semaforo_temporizzato.blif
sis> full_simplify
sis> source script.rugged
sis> write_blif ../FSMD.blif
```

Mapping del sistema sulla libreria tecnologica `synch.genlib`:

```
sis> read_library synch.genlib
sis> map -m 0 -s
```

Outline

Struttura di un file

Tre processi:

- blocco sequenziale sincrono che emula il comportamento dei registri del modello di Huffman;
- blocco sequenziale asincrono, sensibile ai cambiamenti di stato, dei segnali di ingresso della FSM, che implementa la macchina a stati. Il processo è solitamente implementato con costrutti `case` ed `if`.
- blocco sequenziale sincrono che implementa le operazioni nel datapath. Il flusso di controllo del blocco sequenziale è solitamente regolato dai valori dei segnali di controllo generati dalla FSM (ed eventualmente, dai segnali d'ingresso nel datapath).

Esempio - Semaforo temporizzato: blocchi sincroni

```
module SemaforoTemporizzato(
    input clk, trafficons, trafficoeo,
    output reg lucens, luceeo);
        reg [1:0] stato = 2'b00;
        reg [1:0] stato_prossimo = 2'b00;
        reg inizio = 1'b0, fine = 1'b0;
        reg [2:0] registro = 3'b000;

        always @(clk) begin : UPDATE
            stato = stato_prossimo;
        end
        ...
```

Esempio - Semaforo temporizzato: blocchi sincroni

```
...
always @(posedge clk) begin : DATAPATH
    if(inizio) begin
        registro = 3'b000;
        fine = 1'b0;
    end else begin
        if(registro < 3'b111) begin
            registro = registro + 1'b1;
            fine = 1'b0;
        end else begin
            fine = 1'b1;
        end
    end
end
```

Esempio - Semaforo temporizzato: blocco asincrono

```
always @(stato, trafficons, trafficoeo, fine)
begin : FSM
    case(stato)
    2'b00:
        if(~trafficons && trafficoeo) begin
            lucens = 1'b1; luceeo = 1'b0;
            inizio = 1'b1;
            stato_prossimo = 2'b10;
        end else begin
            lucens = 1'b1; luceeo = 1'b0;
            stato_prossimo = 2'b00;
        end
    ...
```

Esempio - Semaforo temporizzato: blocco asincrono

```
        ...
        2'b11:
            begin inizio = 1'b0;
            if(fine) begin
                lucens = 1'b1; luceeo = 1'b0;
                stato_prossimo = 2'b00;
            end else begin
                lucens = 1'b0; luceeo = 1'b1;
                stato_prossimo = 2'b11;
            end end
        endcase
    end
endmodule
```

Outline

Idea

- Il modello Verilog (più astratto) deve guidare lo sviluppo della sua implementazione in Sis.
- Il comportamento del modello Verilog (più semplice da verificare), può essere utilizzato per verificare il comportamento del modello Sis.
- A parità di ingressi, i modelli Verilog e Sis devono produrre la stessa sequenza di uscite.

Dunque, si modifica il testbench Verilog per produrre i file necessari alla verifica del modello Sis. Per creare file in Verilog:

- dichiarazione di una variabile intera `fd` che identifichi il file (i.e., file descriptor);
- `fd = $fopen(“nome file”, “w”)`: apertura del file per la scrittura.
- `$fclose(fd)`: chiusura del file.
- `$fdisplay(fd, “stringa da stampare”)`: funzione per la stampa.

Esempio: semaforo temporizzato

Si creano due file: uno script `testbench.script` che farà si che Sis esegua il test; un file `output_verilog.txt` che salva i valori di output ad ogni ciclo di clock.
Creazione dei file:

```
module tb_semaforo();
    // File descriptors.
    integer tbf, outf;
...
    initial begin
        ...
        tbf = $fopen("testbench.script", "w");
        outf = $fopen("output_verilog.txt", "w");

// funzione $fdisplay per stampare sul file tbf
$fdisplay(tbf,
                    "read_blif semaforo_temporizzato.blif");
```

Esempio: semaforo temporizzato

Gli ingressi vanno scritti nel file `testbench.script`, mentre le uscite vanno salvate nel file `output_verilog.txt`.

```
clk = 1'b0;
trafficons = 1'b0;
trafficoeo = 1'b0;
$fdisplay(tbf, "simulate %b %b", trafficons, trafficoeo);
#20
$fdisplay(outf, "Outputs: %b %b", lucens, luceeo);
trafficons = 1'b0;
trafficoeo = 1'b1;
$fdisplay(tbf, "simulate %b %b", trafficons, trafficoeo);
#20
$fdisplay(outf, "Outputs: %b %b", lucens, luceeo);
...
```

Esempio: semaforo temporizzato

Chiudere (SEMPRE) i file prima di terminare l'esecuzione!

```
...
#20
$fdisplay(outf, "Outputs: %b %b", lucens, luceeo);
$fdisplay(tbf, "quit");
// E'fondamentale chiudere i file con $fclose.
$fclose(tbf);
$fclose(outf);
$finish;
```

Utilizzo dei file in Sis

Utilizzare il file `testbench.script` per stimolare l'esecuzione del sistema in Sis, filtrando le uscite (filtrate) nel file `output_sis.txt`:

```
bash> sis -f testbench.script -x |
                          grep Outputs: > output_sis.txt
```

Verificare le differenze tra l'output del modello verilog e l'output del modello sis:

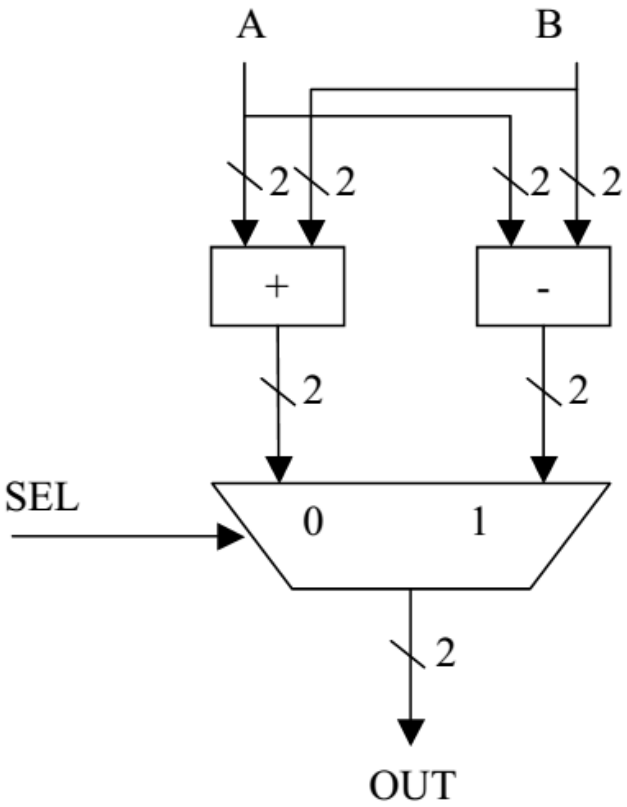
```
bash> diff output_sis.txt output_verilog.sis
```

Se il comando non stampa nulla, i due output sono equivalenti.

Outline

Esercizio 1

Realizzare in formato blif e Verilog il seguente datapath. A e B sono due numeri positivi da 2 bit. Se SEL=0 allora OUT fornisce il valore della somma tra A e B; se SEL= 1 allora OUT fornisce il valore della differenza tra A e B.



Esercizio 2

Realizzare in formato blif e Verilog la FSMD del circuito che controlla l'apertura di una cassaforte. Gli ingressi del circuito sono:

- APRI[1]: vale 1 quando viene premuto il bottone invio per aprire la cassaforte. In tal caso il circuito deve controllare che il valore di NUM sia uguale a "0110" per aprire la cassaforte.
- NUM[4]: è il codice a 4 bit della combinazione.

Le uscite del circuito sono:

- APERTA[1]: viene impostato a 1 quando viene premuto il bottone invio e NUM=0110. Viene impostato a 0 quando viene premuto il bottone chiudi per richiudere la cassaforte oppure dopo che sono trascorsi 4 cicli di clock da quando la cassaforte è stata aperta.