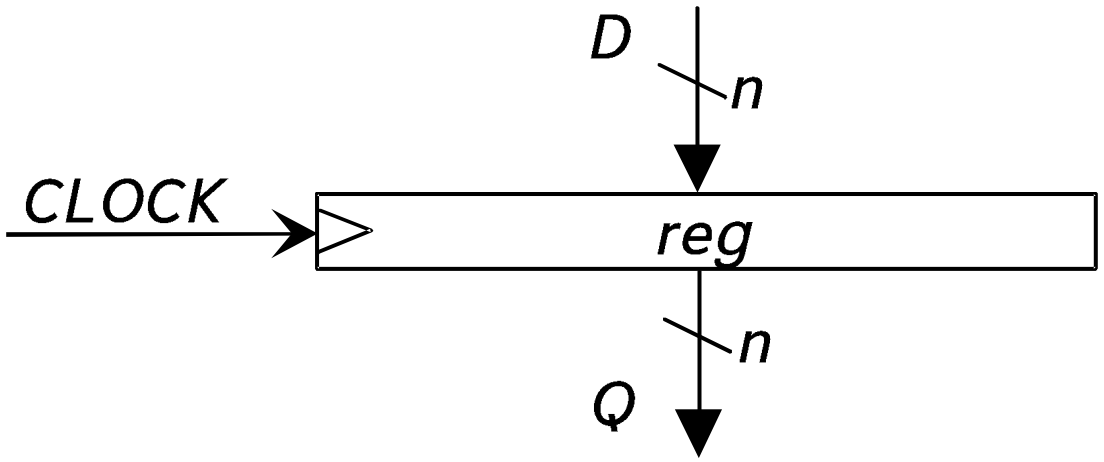
Componenti del Datapath e Modello FSMD

Dal libro di testo:
Fummi, Lora, Sami, Silvano, *Progettazione Digitale*, 3 ed., McGraw-Hill

Codice disponibile:
https://www.mheducation.it/

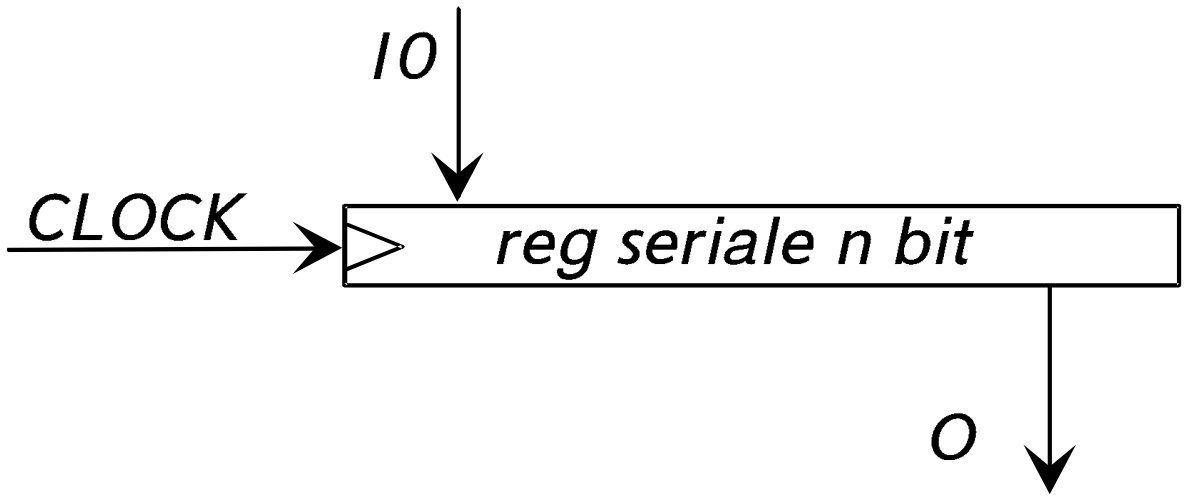
Registri

Registro parallelo/parallelo



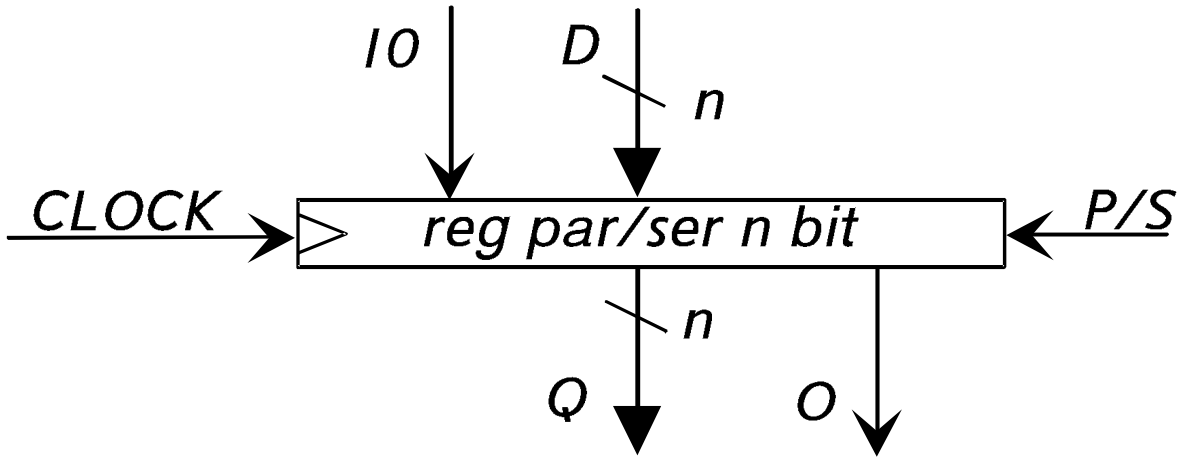

```
module RegistroParalleloParallelo #( parameter N = 8)(
   input [N-1:0] D,
   input clock,
   output [N-1:0] Q);
   reg [N-1:0] dato = 8'b00000000;
   assign Q = dato;
   always @(posedge clock) begin
      dato = D;
   end
endmodule
```

Registro seriale/seriale




```
module RegistroSerialeSeriale (input IO, input clock,
output O);
   parameter N = 8;  reg [N-1:0] dato = 8'b00000000;
   assign O = dato[0];
   always @(posedge clock) begin
      dato = {IO, dato[N-1:1]};
   end
endmodule
```

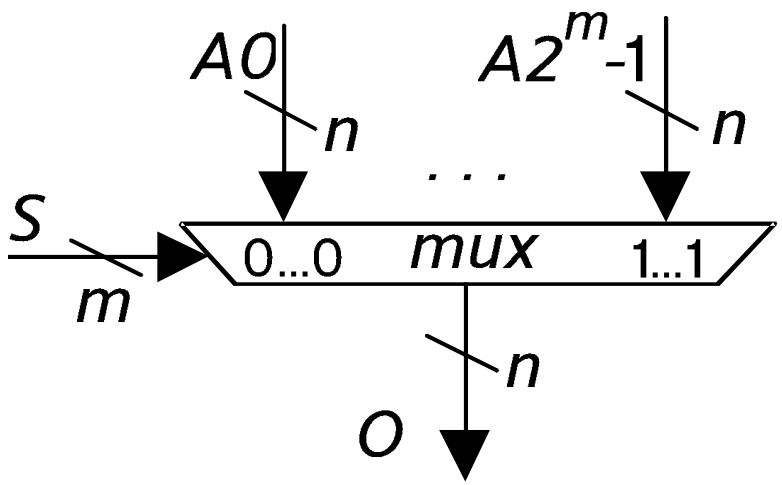
Registro parallelo/seriale




```
module RegistroParalleloSeriale #(parameter N=8)(
   input PS, input IO, input [N-1:0] D, input clock,
   output [N-1:0] Q, output O);
   reg [N-1:0] dato = 8'b00000000;
   assign Q = dato;
   assign O = dato[0];
   always @(posedge clock) begin
      if(PS) dato = D;
      else dato = {IO, dato[N-1:1]};
   end
endmodule
```

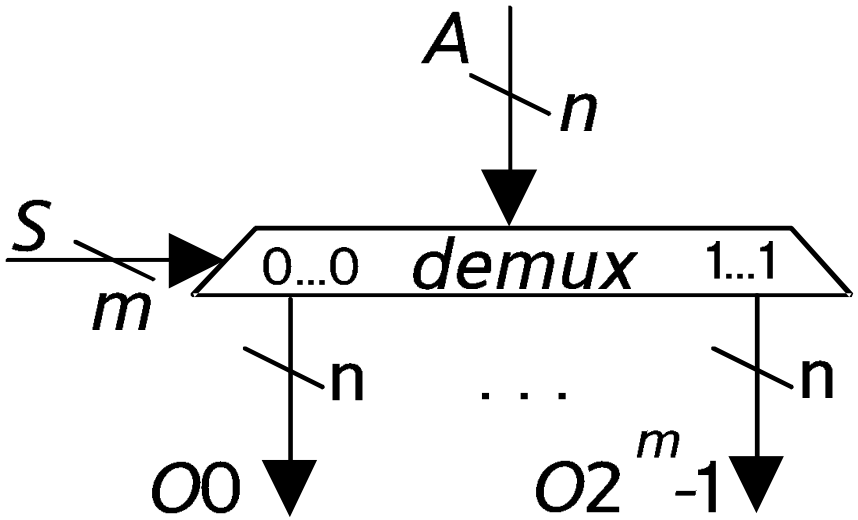
Unità Funzionali

Multiplexer



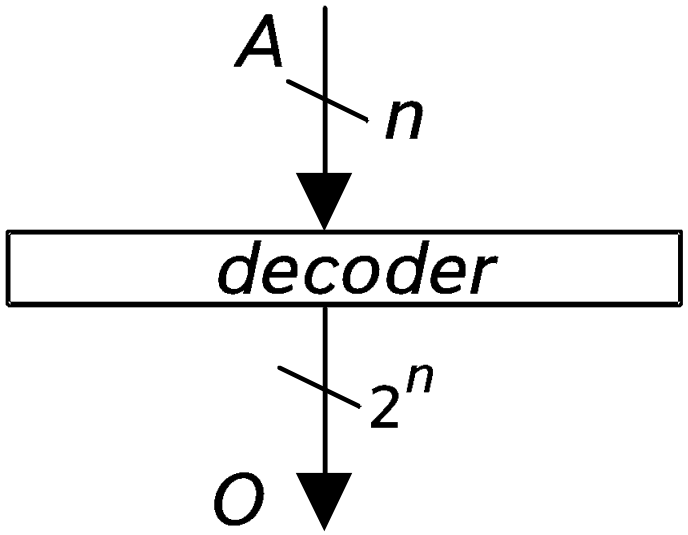

```
module Multiplexer #(parameter N = 8) (
   input [1:0] S, input [N-1:0] A3, input [N-1:0] A2,
   input [N-1:0] A1, input [N-1:0] A0,
   output reg [N-1:0] O);
   always @(A3, A2, A1, A0, S) begin
      case(S)
         2'b00: O = A0;
         2'b01: O = A1;
         2'b10: O = A2;
         2'b11: O = A3;
         default: O = 8'b00000000;
      endcase
   end
endmodule
```

Demultiplexer



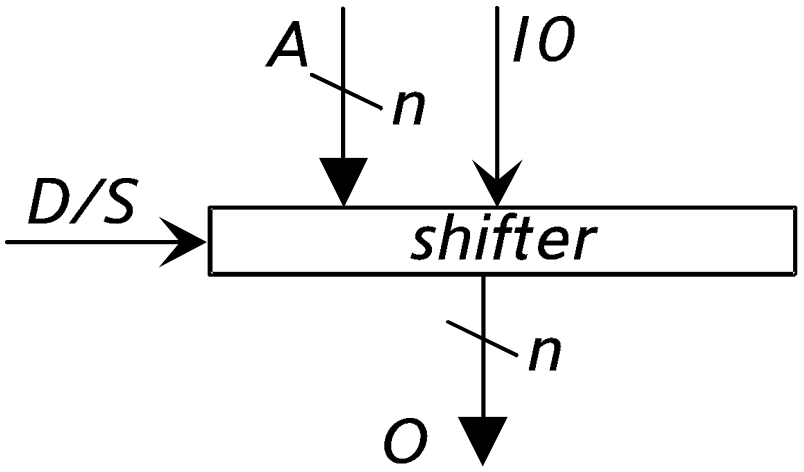

```
module Demultiplexer #(parameter N = 8) (
   input [1:0] S, input [N-1:0] A,
   output reg [N-1:0] O3, output reg [N-1:0] O2,
   output reg [N-1:0] O1, output reg [N-1:0] O0);

   always @(S, A) begin
      case(S)
         2'b00: begin
            O0 = A; O1 = 0; O2 = 0; O3 = 0;
            end
         2'b01: begin
            O1 = A; O0 = 0; O2 = 0; O3 = 0;
            end
         2'b10: begin
            O2 = A; O0 = 0; O1 = 0; O3 = 0;
            end
         2'b11: begin
            O3 = A; O0 = 0; O1 = 0; O2 = 0;
             end
      endcase
   end
endmodule
```

Decoder




```
module Decoder #(parameter N = 8)(
   input [N-1:0] A, output reg [(2**N)-1:0] O);
   integer i;
   always @(A)
   begin
      for(i = 0; i < (2**N); i = i + 1) begin
         if(i == A) O[i] = 1'b1;
         else O[i] = 1'b0;
      end
   end
endmodule
```

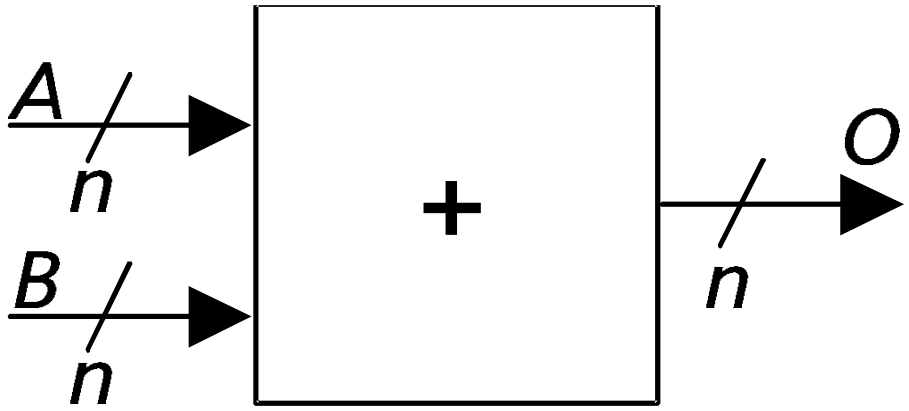
Shifter




```
module Shifter #(parameter N = 8)
(input DS, input [N-1:0] A, input IO,
output reg [N-1:0] O);
   always @(A, IO) begin
      if(DS) O = {IO, A[N-1:1]};
      else O = {A[N-2:0], IO};
   end
endmodule
```

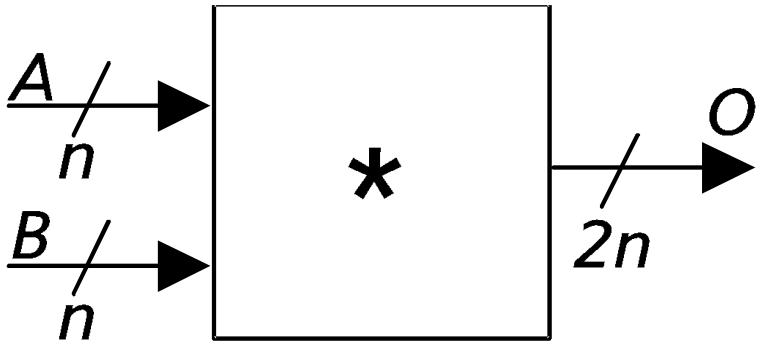
Unità aritmetiche

Somatore




```
module Sommatore #(parameter N = 8)(
input [N-1:0] A, input [N-1:0] B,
output [N-1:0] O);
   assign O = A + B;
endmodule
```

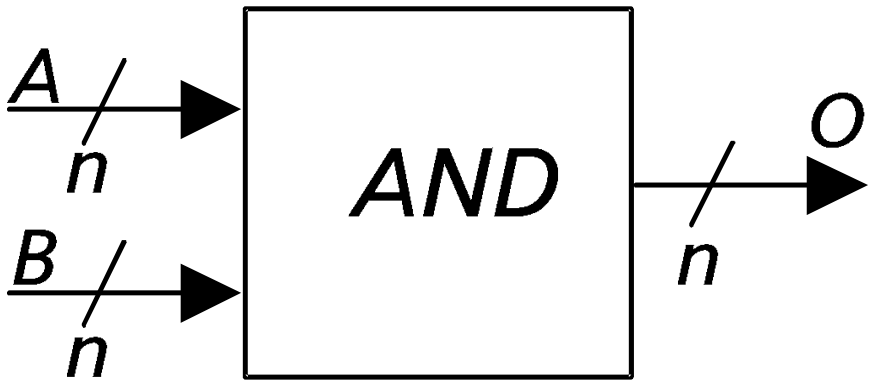
Moltiplicatore




```
module Moltiplicatore #(parameter N = 8)(
input [N-1:0] A, input [N-1:0] B,
output [(N**2)-1:0] O);
   assign O = A * B;
endmodule
```

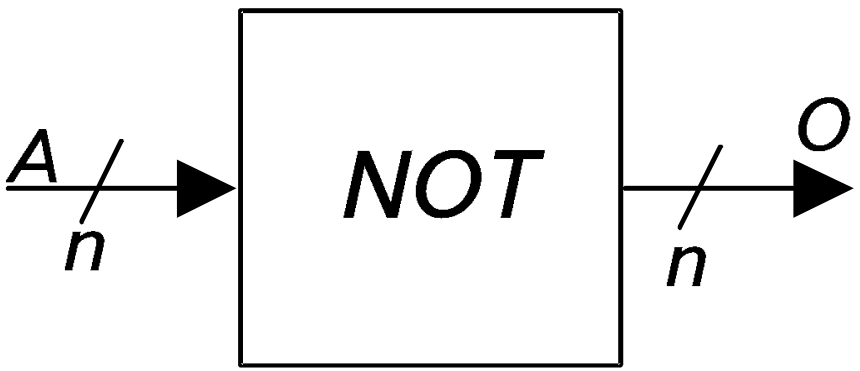
Unità Logiche

And




```
module And #(parameter N = 8)(
input [N-1:0] A, input [N-1:0] B,
output reg [N-1:0] O);
   integer i;
   always @(A, B) begin
      for(i = 0; i < N; i = i + 1) begin
         O[i] = A[i] & B[i];
      end
   end
endmodule
```

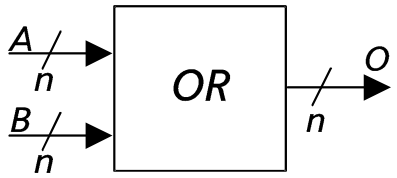
Not




```
module Not #(parameter N = 8)(
input [N-1:0] A
output reg [N-1:0] O);
   integer i;
   always @(A) begin
      for(i = 0; i < N; i = i + 1) begin
         O[i] = ~A[i];
      end
   end
endmodule
```

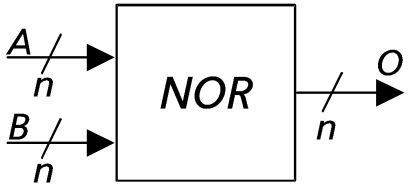
Altri operatori logici

- Or



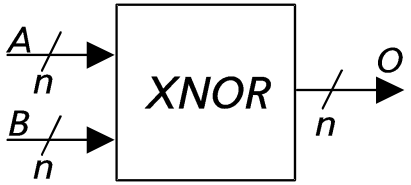
- Nand

- Nor



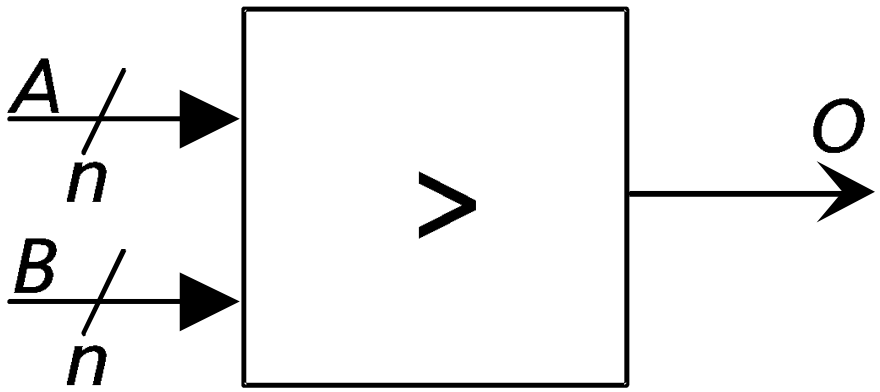
- Xor

- XNor



Operatori Confronto

Operatori di confronto




```
module Maggiore #(parameter N = 8)(
input [N-1:0] A, input [N-1:0] B,
output reg O);

   always @(A, B) begin
      if( A > B ) O = 1'b1;
      else O = 1'b0;
   end
endmodule
```

Datapath in Verilog

Libreria di componenti

- Registri
  - Registro parallelo/parallelo
  - Registro parallelo/seriale
  - Registro seriale/seriale
- Unità funzionali
  - Multiplexer
  - Demultiplexer
  - Decoder
  - Shifter
- Arithmetiche
  - Sommatore
  - Moltiplicatore
- Logiche
  - Not
  - And, Or
  - Nand, Nor
  - Xor, Xnor
- Confronto
  - Maggiore
  - Maggiore o uguale
  - Minore
  - Minore o uguale
  - Uguale
  - Diverso

Esempio: ALU

```
module Opposto #(parameter N = 8)(
        input [N-1:0] operando,
        output [N-1:0] risultato);
            assign risultato = -operando;
endmodule
```

```
module ALU #(parameter N = 8)(
        input clock,
        input [N-1:0] op1,
        input [N-1:0] op2,
        input stored,
        input [1:0] oper,
        output [N-1:0] O);

            wire [N-1:0] acc, sel, t1, t2, t3, t4, t5, out;
            wire s1, s2;

            RegistroParalleloParallelo registro(out, clock, acc);
            Multiplexer2 mux1(stored, acc, op2, sel);
            Multiplexer2 mux2(s1, op1, sel, t3);
            Multiplexer2 mux3(s2, op1, sel, t4);
            Multiplexer mux4(oper, t4, t3, t2, t1, out);
            Sommatore sum1(op1, sel, t1);
            Opposto opp1(sel, t5);
            Sommatore sum2(op1, t5, t2);
            Maggiore mag(op1, sel, s1);
            Minore min(op1, sel, s2);

            assign O = out;
endmodule
```

Esempio: ALU Behavioral

```
module ALU_behav #(parameter N = 8)(
    input clock,
    input [N-1:0] op1,
    input [N-1:0] op2,
    input stored,
    input [1:0] oper,
    output [N-1:0] O);

        reg [N-1:0] acc, sel, out;

        always @(stored, acc, op2) begin
            if(stored) sel = acc;
            else sel = op2;
        end

        always @(posedge clock) begin
            if(clock) acc = out;
        end

...
```

```
...

    assign O = out;

    always @(op1, sel, oper) begin
        case(oper)
        2'b00:
            out = op1 + sel;
        2'b01:
            out = op1 - sel;
        2'b10:
            if(op1 > sel) out = op1;
            else out = sel;
        2'b11:
            if(op1 < sel) out = op1;
            else out = sel;
        endcase
    end
endmodule
```

Controllore e Datapath

Semaforo temporizzato in Verilog

```
module SemaforoTemporizzato(
    input rst, 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 [3:0] registro = 3'b000;

    always @(clk) begin : UPDATE
        if(rst) stato = 2'b00;
        else stato = stato_prossimo;
    end
```

```
    always @(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
```

Semaforo temporizzato

```
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
    2'b01:
        if(~trafficons) begin
            lucens = 1'b0; luceeo = 1'b1;
            stato_prossimo = 2'b01;
        end else begin
            lucens = 1'b0; luceeo = 1'b1; inizio = 1'b1;
            stato_prossimo = 2'b11;
        end
...
```

```
    2'b10:
        begin inizio = 1'b0;
        if(trafficons) begin
            lucens = 1'b1; luceeo = 1'b0;
            stato_prossimo = 2'b00;
        end
        else if(fine) begin
            lucens = 1'b0; luceeo = 1'b1;
            stato_prossimo = 2'b01;
        end else begin
            lucens = 1'b1; luceeo = 1'b0;
            stato_prossimo = 2'b10;
        end end
    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
```