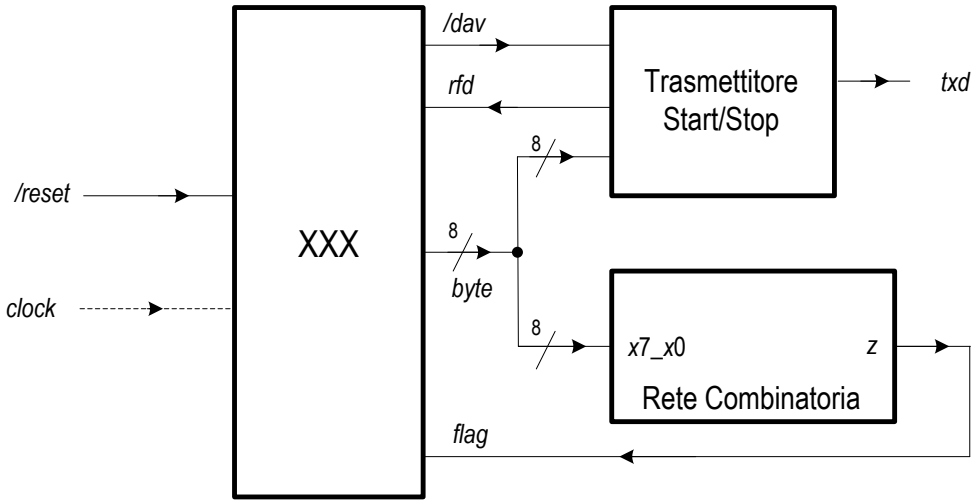
ESERCIZIO 1

L'Unità **XXX** inizia il suo lavoro al reset asincrono e, quando lo finisce, si ferma in attesa di un nuovo reset asincrono.

Il lavoro di **XXX** consiste nell'emettere, per mezzo del trasmettitore seriale start/stop (connesso ad **XXX** come in figura), tutti gli stati di ingresso riconosciuti dalla Rete Combinatoria, connessa a **XXX** tramite le variabili *byte* e *flag*. Si consideri il tempo di risposta della rete combinatoria molto più piccolo del periodo del clock



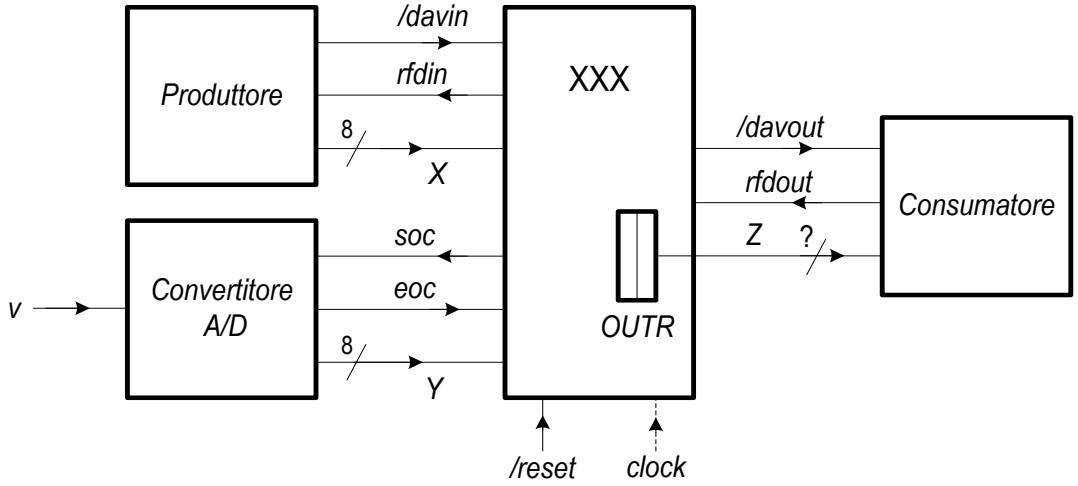
Descrivere e sintetizzare l'Unità XXX.

Una possibile descrizione

```
module XXX(byte,flag,dav_,rfd,clock,reset_);
 input       clock, reset_;
 input       flag,rfd;
 output      dav_;
 output[7:0] byte;

 reg         DAV_; assign dav_=DAV_;
 reg [7:0]   BYTE; assign byte=BYTE;

 reg [2:0]  STAR; parameter [2:0] S0=0,S1=1,S2=2,S3=3,S4=4;

 always @(reset_==0) begin BYTE<=0; DAV_<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    //Ricerca di uno stato di ingresso riconosciuto dalla rete
    S0: begin STAR<=(flag==1)?S2:S1; end
    S1: begin BYTE<=BYTE+1; STAR<=(BYTE=='HFF)?S4:S0; end

    //Trasmissione dello stato di ingresso riconosciuto dalla rete
    S2: begin DAV_<=0; STAR<=(rfd==1)?S2:S3; end
    S3: begin DAV_<=1; STAR<=(rfd==0)?S3:S1; end

    //Arresto
    S4: begin STAR<=S4; end
   endcase
endmodule
```

ESERCIZIO 2

X, Y e Z rappresentano, in complemento a due, tre numeri *x*, *y*, e *z*. L'Unità XXX si comporta all'infinito come segue: Ogni volta che riceve un nuovo byte X dal Produttore, preleva un nuovo byte Y dal Convertitore A/D e invia al Consumatore una nuova configurazione Z, tale che sia $z = x/2 + 2y$. Descrivere e sintetizzare l'Unità XXX.



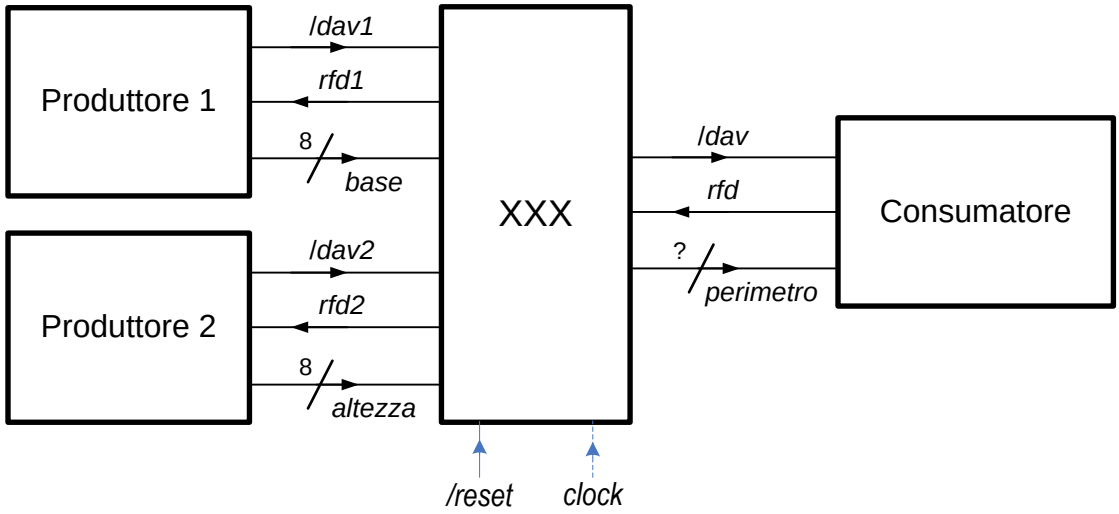
Una soluzione non troppo ottimizzata

```
module XXX(Z,davout_,rfdout, X,davin_,rfdin, Y,soc,eoc, clock,reset_);
 input       clock, reset_;
 input       rfdout,davin_,eoc;
 output      davout_,rfdin,soc;
 input [7:0] X,Y;
 output[9:0] Z;
 reg        DAVOUT_,RFDIN,SOC;
 reg [9:0]  OUTR;
 reg [7:0]  APPX;
 reg [2:0]  STAR; parameter [2:0] S0=0,S1=1,S2=2,S3=3,S4=4,S5=5;
 assign davout_=DAVOUT_; assign rfdin=RFDIN; assign soc=SOC; assign Z=OUTR;

 always @(reset_==0) begin DAVOUT_<=1; RFDIN<=1; SOC<=0; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
   // Prelievo di un novo byte dal Produttore, con appoggio nel registro APPX
    S0: begin APPX<=X; STAR<=(davin_==1)?S0:S1; end
    S1: begin RFDIN<=0; STAR<=(davin_==0)?S1:S2; end
   //Prelievo di un novo byte dal Convertitore e memorizzazione in OUTR
   //della rappresentazione di (x/2 +2y)
    S2: begin RFDIN<=1; SOC<=1; STAR<=(eoc==1)?S2:S3; end
    S3: begin SOC<=0; OUTR<=mia_funzione(APPX,Y);STAR<=(eoc==0)?S3:S4; end
   //Handshake con il Consumatore
    S4: begin DAVOUT_<=0; STAR<=(rfdout==1)?S4:S5; end
    S5: begin DAVOUT_<=1; STAR<=(rfdout==0)?S5:S0; end
   endcase
 //Funzione che calcola la rappresentazione di (x/2 +2y)
 function [9:0] mia_funzione;
  input [7:0] APPX,Y;
  mia_funzione={APPX[7],APPX[7],APPX[7],APPX[7:1]} + {Y[7],Y[7:0],1'B0};
 endfunction
endmodule
```

ESERCIZIO 3

Descrivere l'Unità **XXX** che si evolve ciclicamente come segue: "preleva un byte dal Produttore 1 e un byte dal Produttore 2, elabora i byte ed invia il risultato della elaborazione al Consumatore."

L'elaborazione viene fatta tramite una funzione *mia_rete*(*base*,*altezza*), che interpreta i byte ricevuti da XXX come numeri naturali costituenti la base e l'altezza di un rettangolo e restituisce il perimetro del rettangolo.



Dettagliare la struttura della rete combinatoria che implementa la funzione *mia_rete* di cui sopra.

Tener presente che la correzione del compito sarà, come il Compilatore Verilog, case sensitive.

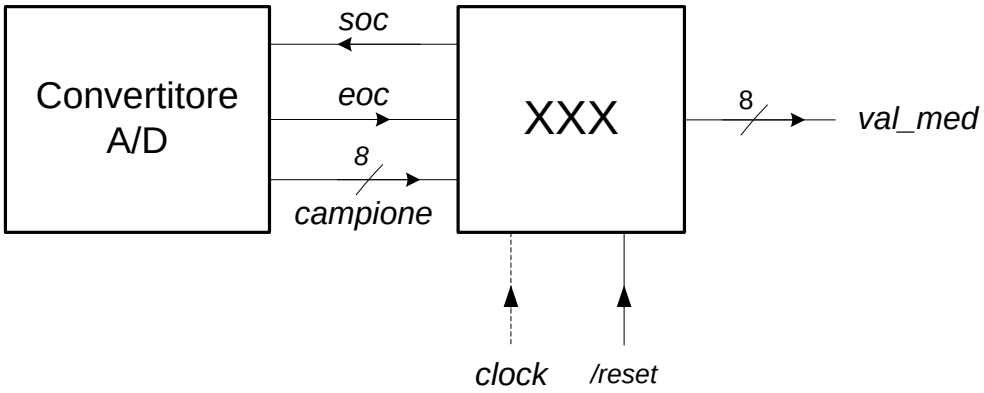
Una possibile soluzione

```
module XXX (dav1_,rfd1,base, dav2_,rfd2,altezza, dav_,rfd,perimetro, clock,reset_);
 input        clock,reset_;
 input        dav1_, dav2_, rfd;
 output       rfd1, rfd2, dav_;
 input  [7:0] base, altezza;
 // Per non avere traboccamenti, il perimetro va calcolato su 10 bit
 output [9:0] perimetro;

 reg          RFD1, RFD2, DAV_; assign rfd1=RFD1; assign rfd2=RFD2; assign dav_=DAV_;
 reg    [9:0] PERIMETRO;        assign perimetro=PERIMETRO;

 reg STAR;
 parameter S0=0,S1=1;

 function[9:0] mia_rete;
  input [7:0] base, altezza;
  mia_rete = {({1'B0,base}+{1'B0,altezza}),1'B0};
 endfunction

 always @(reset_==0) begin RFD1<=1; RFD2<=1; DAV_<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin RFD1<=1; RFD2<=1; DAV_<=1; PERIMETRO<=mia_rete(base,altezza);
              STAR<=({dav1_,dav2_,rfd}=='B001)?S1:S0; end
    S1: begin RFD1<=0; RFD2<=0; DAV_<=0; STAR<=({dav1_,dav2_,rfd}=='B110)?S0:S1; end
   endcase
endmodule
```

ESERCIZIO 4

Descrivere l'Unità XXX in modo che ciclicamente prelevi un novo campione dal Convertitore A/D ed emetta, tramite la variabile *val_med*, il valor medio (approssimato) degli ultimi 4 campioni prelevati. Non ci si preoccupi cosa avviene al reset fino a che non si sono prelevati 4 campioni. Si ricordi che il convertitore A/D fornisce i campioni di *v* rappresentati in binario bipolare.

Sintetizzare l'Unità XXX secondo il modello Parte Operativa/Parte Controllo.

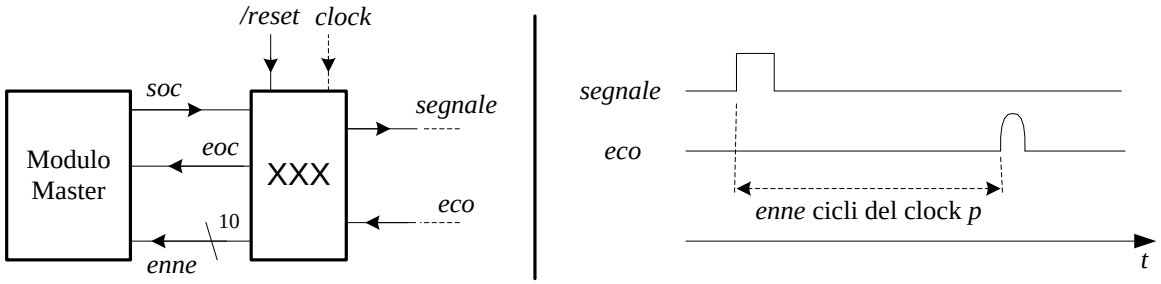


Una possibile soluzione

```
module XXX(campione,soc,eoc, val_med, clock,reset_);
 input         clock,reset_;
 input   [7:0] campione;
 output        soc;
 input         eoc;
 output  [7:0] val_med;

 wire    [9:0] campione_esteso; // su 10 bit e in complemento a due
 assign  campione_esteso={!campione[7],!campione[7],!campione[7],campione [6:0]};

 reg           SOC; assign soc=SOC;

 //4 registri per memorizzare 4 campioni, estesi su 10 bit e in complemento a due
 reg     [9:0] APP3,APP2,APP1,APP0;
 wire    [9:0] sommatoria; //somma degli ultimi quattro campioni estesi
 assign        sommatoria=(APP3 + APP2) + (APP1 + APP0); //Servono 3 sommatori

 reg     [7:0] VAL_MED; assign val_med=VAL_MED;

 reg     [1:0] STAR; parameter S0=0,S1=1,S2=2;

 always @(reset_==0) begin SOC<=0; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin SOC<=1; STAR<=(eoc==1)?S0:S1; end
    S1: begin SOC<=0; APP3<=campione_esteso; STAR<=(eoc==0)?S1:S2; end
    S2: begin VAL_MED<={!sommatoria[9],sommatoria[8:2]}; //Divisione per 4 con
                                                         //ritorno al binario bipolare
              APP0<=APP1; APP1<=APP2; APP2<=APP3; STAR<=S0; end
   endcase
endmodule
```

ESERCIZIO 5

L'Unità *XXX* colloquia con il Modulo Master con un handshake *soc* (Start Of Computation), *eoc* (End Of Computation) del tutto simile all'handshake tipico dei Convertitori A/D. Il numero *enne* che l'Unità *XXX* fornisce al Modulo Master è calcolato in accordo alle seguenti specifiche.

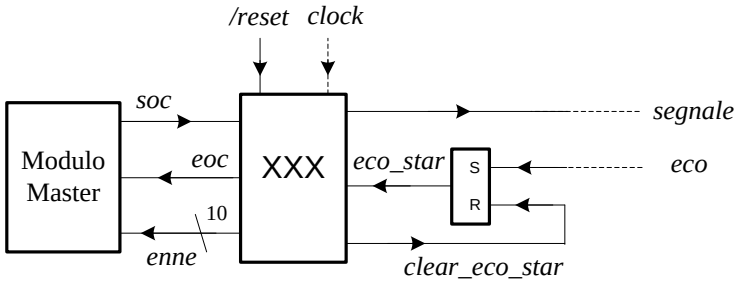


Quando l'Unità *XXX* viene attivata dal Modulo Master emette, tramite la variabile di uscita *segnale*, un impulso di durata pari ad un ciclo di clock e ne raccoglie, tramite la variabile di ingresso *eco*,una eco proveniente da un ostacolo (il ritardo con cui arriva l'eco è proporzionale alla distanza dell'ostacolo). L'Unità XXX calcola pertanto un numero naturale *enne* pari al numero dei periodi di clock che intercorrono tra l'emissione dell'impulso e l'arrivo dell'eco, con una saturazione a 1023 se tale numero tendesse a superare questo limite.

Nota 1: ATTENZIONE: in questo handshake, l'Unità *XXX* gioca il ruolo del Convertitore

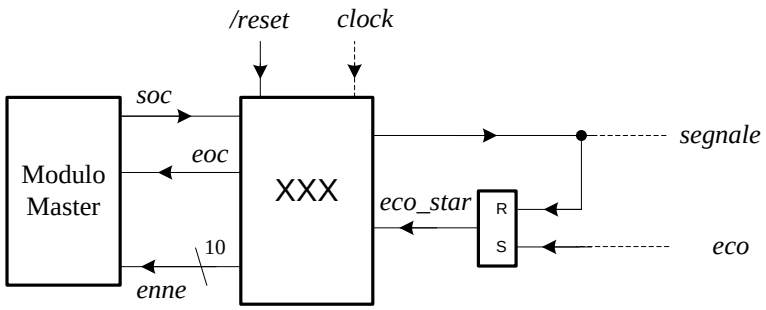
Nota 2: L'eco è un impulso molto distorto e spesso molto breve, che potrebbe non essere visto dall'Unità *XXX*, se non viene inserito un circuito che lo catturi e lo presenti al'Unità *XXX* in modo sicuro.

Una possibile soluzione




```
module XXX(soc,eoc,enne, segnale, eco_star,clear_eco_star, clock,reset_);
 input        clock,reset_;
 input        soc;
 output       eoc;
 output [9:0] enne;
 output       segnale,clear_eco_star;
 input        eco_star;
 reg       EOC,SEGNALE,CLEAR_ECO_STAR;
 assign eoc=EOC; assign segnale=SEGNALE; assign clear_eco_star=CLEAR_ECO_STAR;
 reg [9:0] ENNE; assign enne=ENNE;
 reg [1:0] STAR; parameter S0=0,S1=1,S2=2,S3=3;

 always @(reset_==0) begin SEGNALE<=0; EOC<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin EOC<=1; CLEAR_ECO_STAR<=1; STAR<=(soc==0)?S0:S1; end
    S1: begin EOC<=0; CLEAR_ECO_STAR<=0; SEGNALE<=1; ENNE<=0; STAR<=S2; end
    S2: begin SEGNALE<=0; ENNE<=((eco_star==0)&(ENNE<1023))?(ENNE+1):ENNE;
        STAR<=(eco_star==0)?S2:S3; end
    S3: begin STAR<=(soc==1)?S3:S0; end
   endcase
endmodule
```

Una soluzione più semplice

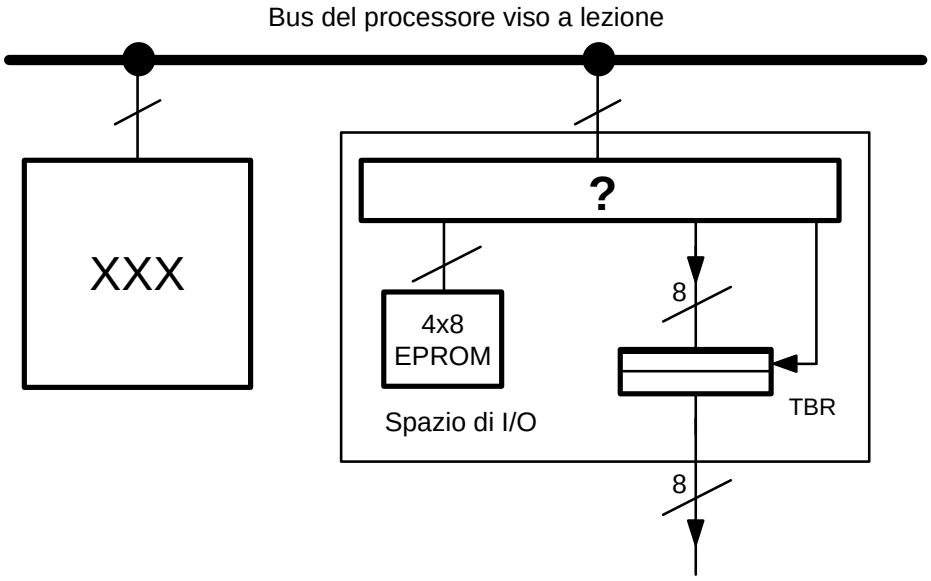



```
module XXX(soc,eoc,enne, segnale, eco_star, clock,reset_);
 input        clock,reset_;
 input        soc;
 output       eoc;
 output [9:0] enne;
 output       segnale;
 input        eco_star;
 reg       EOC,SEGNALE; assign eoc=EOC; assign segnale=SEGNALE;
 reg [9:0] ENNE; assign enne=ENNE;
 reg [1:0] STAR; parameter S0=0, S1=1, S2=2, S3=3;
 always @(reset_==0) begin SEGNALE<=0; EOC<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin EOC<=1; STAR<=(soc==0)?S0:S1; end
    S1: begin EOC<=0; SEGNALE<=1; ENNE<=0; STAR<=S2; end
    S2: begin SEGNALE<=0; ENNE<=((eco_star==0)&(ENNE<1023))?(ENNE+1):ENNE;
        STAR<=(eco_star==0)?S2:S3; end
    S3: begin STAR<=(soc==1)?S3:S0; end
   endcase
endmodule
```

ESERCIZIO 6

Specificare la scatola **?** in modo che la EPROM ed il registro TBR risultino montati nello spazio di I/0 agli indirizzi 0,1,2,3 la EPROM e all'indirizzo 4 il registro TBR e quindi eliminare dal bus tutti i fili inutili.

Descrivere e **sintetizzare** l'Unità **XXX** in modo che, ciclicamente, emetta il contenuto delle locazioni della EPROM tramite il registro TBR, mantenendovi il contenuto di ogni locazione per 20 cicli di clock



Soluzione che rispecchia esattamente il testo

Il bus si riduce alle due variabili di comando */ior*, */iow*, a tre bit di indirizzo e agli otto bit per i dati.

Il registro TBR e la logica di raccordo al bus costituiscono una interfaccia parallela di uscita senza handshake (vedi pag. 195) e una maschera che deve riconoscere l'indirizzo 4, ovvero l'unico indirizzo il cui bit più significativo vale 1. Chiamiamo tale interfaccia *Parallel_Out*

Il montaggio classico della EPROM (vedi testo) è ovvio, con la sola accortezza di comandarla tramite il filo del bus */ior*. La sua maschera deve riconoscere gli indirizzi minori di 4, ovvero tutti gli indirizzi il cui bit più significativo vale 0. La EPROM riceverà i due bit di indirizzo meno significativi.

Lo schema a blocchi dello spazio di I/O è il seguente; per comodità i tre fili di indirizzo sono denotati *a*2 (supporta il bit più significativo) e *a*1_*a*0 (supportano gli altri due bit):

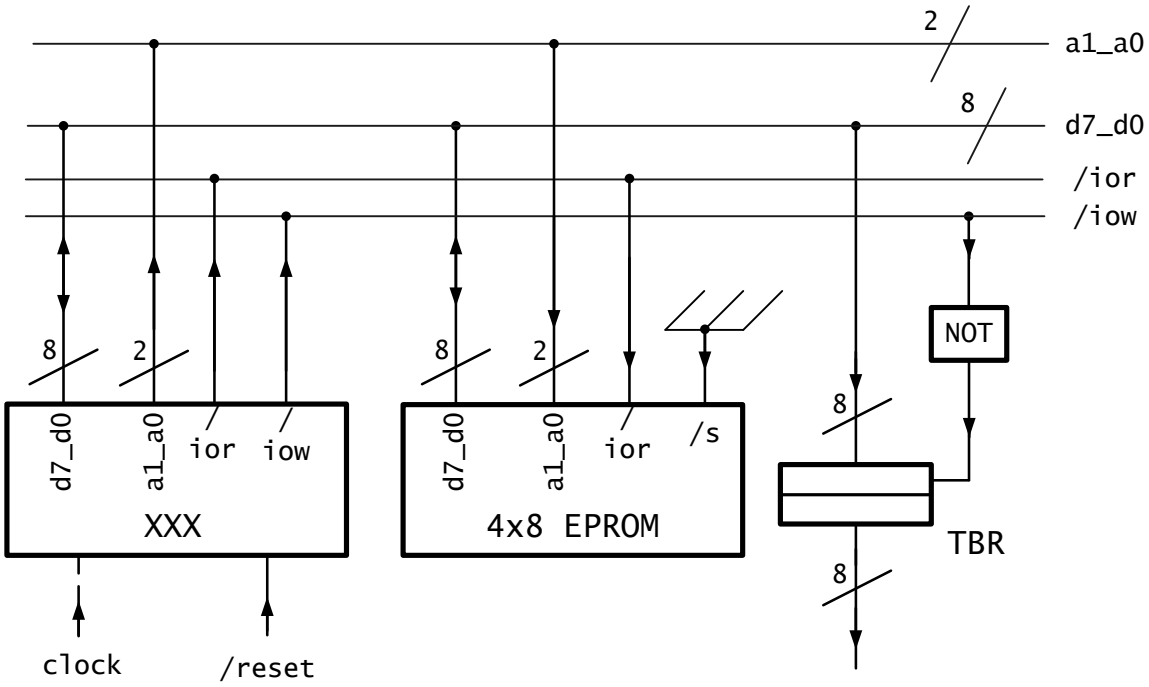
```
module IO_space(d7_d0,a2,a1_a0,ior_,iow_,byte_out);
  input        a2;
  input [1:0]  a1_a0;
  inout [7:0]  d7_d0;
  input        ior_;
  input        iow_;
  output [7:0] byte_out;

  wire sPAROUT_; assign sPAROUT_=!a2;
  Parallel_Out  PAROUT(byte_out,d7_d0,sPAROUT_,iow_);

  wire sEPROM_; assign sEPROM_=a2;
  Eprom         EPROM(d7_d0,a1_a0,sEPROM_,ior_);
endmodule
```

Ciò premesso, l'Unità **XXX** ha la seguente struttura

```
module XXX(d7_d0,a2,a1_a0,ior_,iow_,clock,reset_);
 input         clock,reset_;
 output        ior_,iow_;
 output        a2;
 output [1:0]  a1_a0;
 inout  [7:0]  d7_d0;
 reg           IOR_,IOW_; assign ior_=IOR_; assign iow_=IOW_;
 reg           A2; assign a2=A2;
 reg [1:0]     A1_A0; assign a1_a0=A1_A0;
 reg           DIR;
 reg [7:0]     MBR; assign d7_d0=(DIR==1)?MBR:'HZZ; //FORCHETTA
 reg [4:0]     COUNT;
 reg [2:0]     STAR; parameter [2:0] S0=0,S1=1,S2=2,S3=3,S4=4,S5=5;
 parameter Num_Periodi=20;

 always @(reset_==0) begin DIR<=0; IOR_<=1; IOW_<=1; A1_A0<=3; COUNT<= Num_Periodi;
                           STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin COUNT<=COUNT-1; A2<=0; A1_A0<=(A1_A0)+1; IOR_<=0; STAR<=S1; end
    S1: begin COUNT<=COUNT-1; STAR<=S2; end
    S2: begin COUNT<=COUNT-1; MBR<=d7_d0; IOR_<=1; DIR<=1; A2<=1; STAR<=S3; end
    S3: begin COUNT<=COUNT-1; IOW_<=0; STAR<=S4; end
    S4: begin COUNT<=COUNT-1; IOW_<=1; STAR<=S5; end
    S5: begin DIR<=0; COUNT<=(COUNT==1)?Num_Periodi:(COUNT-1);
              STAR<=(COUNT==1)?S0:S5; end
   endcase
endmodule
```

Soluzione semplificata, prescindendo dalle maschere

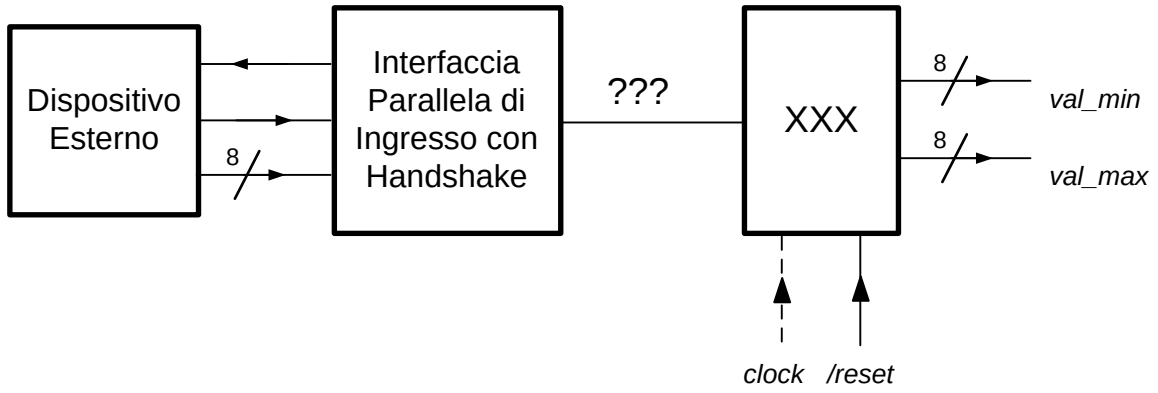
Il registro TBR è l'unico dispositivo accessibile in scrittura e la EPROM è l'unico dispositivo accessibile in lettura. Pertanto, anche se non si rispecchiano esattamente le specifiche del testo, il tutto funziona ugualmente facendo le semplificazioni riportate nella figura sottostante e in base alle quali. il bus si riduce a /ior, /iow, a1_a0 (due bit) e d7_d0 (otto bit);



L'Unità XXX ha in tal caso la seguente struttura

```
module XXX(d7_d0,a1_a0,ior_,iow_,clock,reset_);
 input         clock,reset_;
 output        ior_,iow_;
 output [1:0]  a1_a0;
 inout  [7:0]  d7_d0;
 reg           IOR_,IOW_; assign ior_=IOR_; assign iow_=IOW_;
 reg [1:0]     A1_A0; assign a1_a0=A1_A0;
 reg           DIR;
 reg [7:0]     MBR; assign d7_d0=(DIR==1)?MBR:'HZZ; //FORCHETTA
 reg [4:0]     COUNT;
 reg [2:0]     STAR; parameter [2:0] S0=0,S1=1,S2=2,S3=3,S4=4,S5=5;
 parameter Num_Periodi=20;
 always @(reset_==0) begin DIR<=0; IOR_<=1; IOW_<=1; A1_A0<=3; COUNT<= Num_Periodi;
                           STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0: begin COUNT<=COUNT-1; A1_A0<=(A1_A0)+1; IOR_<=0; STAR<=S1; end
    S1: begin COUNT<=COUNT-1; STAR<=S2; end
    S2: begin COUNT<=COUNT-1; MBR<=d7_d0; IOR_<=1; DIR<=1; STAR<=S3; end
    S3: begin COUNT<=COUNT-1; IOW_<=0; STAR<=S4; end
    S4: begin COUNT<=COUNT-1; IOW_<=1; STAR<=S5; end
    S5: begin DIR<=0; COUNT<=(COUNT==1)?Num_Periodi:(COUNT-1);
              STAR<=(COUNT==1)?S0:S5; end
   endcase
endmodule
```

ESERCIZIO 7

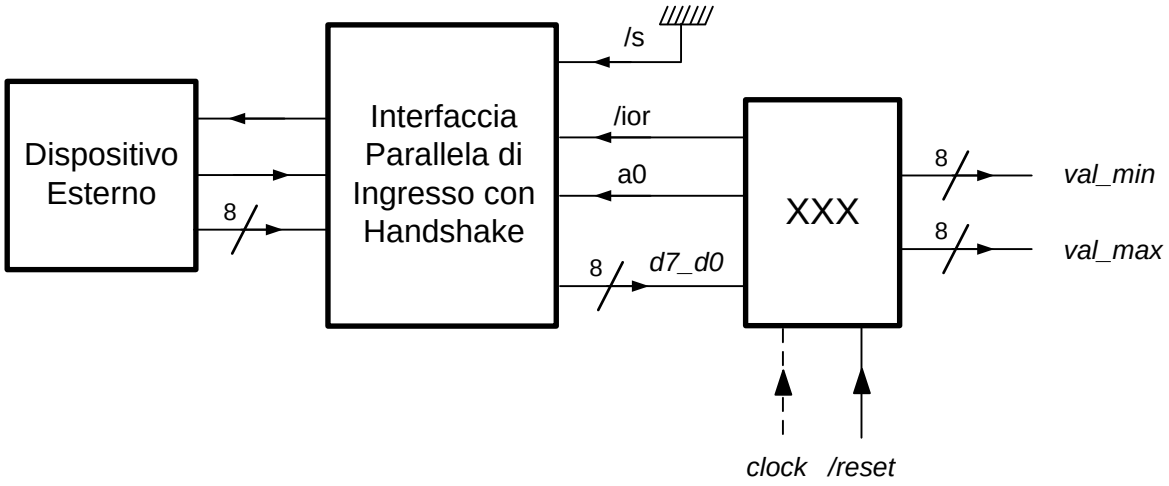


Descrivere l'Unità **XXX** che si evolve all'infinito prelevando nuovi byte dall'Interfaccia Parallela di Ingresso, interpretandoli come numeri in *binario bipolare* e, in tale ottica, presentando in uscita i valori minimo e massimo via via ottenuti.

Sintetizzare l'Unità **XXX** riducendo tutte le reti combinatorie che esso include a reti note.

NOTA: Si supponga che non siano mai necessari stati di wait.
Tener presente che la correzione del compito sarà, come il Compilatore Verilog, case sensitive.

Una possibile soluzione : Si sceglie di gestire l'interfaccia testandone il flag FI




```
module XXX (ior_,a0,d7_d0,val_min,val_max,clock,reset_);
 input        clock,reset_;
 output       ior_,a0;
 input  [7:0] d7_d0;
 output [7:0] val_min, val_max;

 reg A0;              assign a0=A0;
 reg IOR_;            assign ior_=IOR_;
 reg [7:0] VAL_MIN;   assign val_min=VAL_MIN;
 reg [7:0] VAL_MAX;   assign val_max=VAL_MAX;

 reg [2:0] STAR; parameter S0=0,S1=1,S2=2,S3=3,S4=4;
```

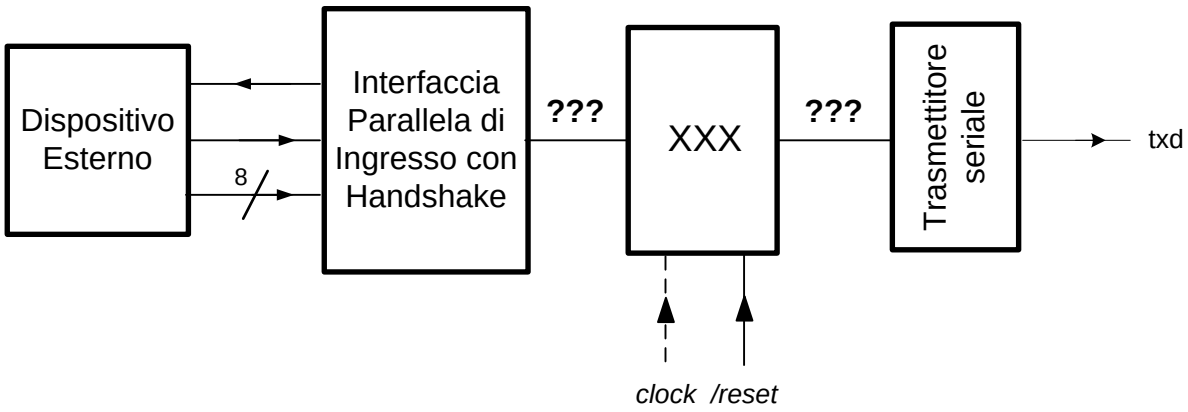
```
 always @(reset_==0) begin VAL_MIN<=255; VAL_MAX<=0; A0<=0; IOR_<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0:   begin IOR_<=0; STAR<=S1; end
    S1:   begin IOR_<=1; STAR<=((d7_d0[0])==0)?S0:S2; end
    S2:   begin A0<=1; STAR<=S3; end
    S3:   begin IOR_<=0; STAR<=S4; end
    S4:   begin VAL_MIN<=(VAL_MIN<d7_d0)?VAL_MIN:d7_d0;
                VAL_MAX<=(d7_d0<VAL_MAX)?VAL_MAX:d7_d0;
                IOR_<=1; A0<=0; STAR<=S0; end
   endcase
endmodule
```

Stante le proprietà della rappresentazione in binario bipolare, la rete combinatoria denotata con < per verificare se un numero è minore di un altro, è un semplice sottrattore per numeri naturali. Più precisamente, detta *b* la variabile prestito in uscita dal sottrattore:

1) il test (`VAL_MIN < d7_d0)?...`, che equivale a `((VAL_MIN - d7_d0)<0) ?...` , diventa `(b==1) ?...`, purchè si attacchino gli ingressi del sottrattore in modo che questa rete calcoli `VAL_MIN - d7_d0`
2) il test (`d7_d0 < VAL_MAX)?...`, che equivale a `((d7_d0 - VAL_MAX)<0) ?...` , diventa `(b==1) ?...`, purchè si attacchino gli ingressi del sottrattore in modo che questa rete calcoli `d7_d0 - VAL_MAX`

ESERCIZIO 8

L'Unità **XXX** preleva un byte dall'interfaccia parallela di ingresso, lo elabora, ed emette il byte elaborato tramite il trasmettitore seriale start/stop.



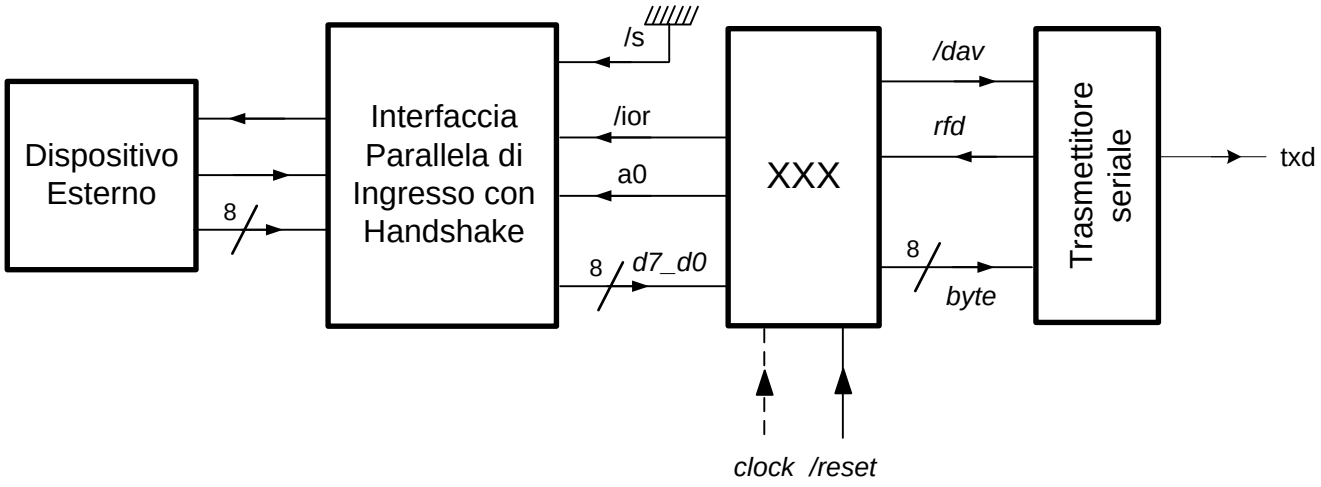
L'elaborazione è la seguente: se il bit più significativo del byte ricevuto è 1, il byte risultato coincide con quello ricevuto; se il bit più significativo del byte ricevuto è 0, il byte risultato coincide con quello ricevuto solo nei bit di posizione pari (bit n. 6, 4, 2, 0), mentre i bit di posizione dispari (bit n. 7, 5, 3, 1) valgono 0.

Descrivere e sintetizzare l'Unità **XXX**.

Stante la semplicità del problema, sarà tenuta in considerazione anche la qualità della soluzione.

NOTA: L'interfaccia parallela è sufficientemente veloce da non richiedere l'uso di stati di wait.

Una possibile soluzione

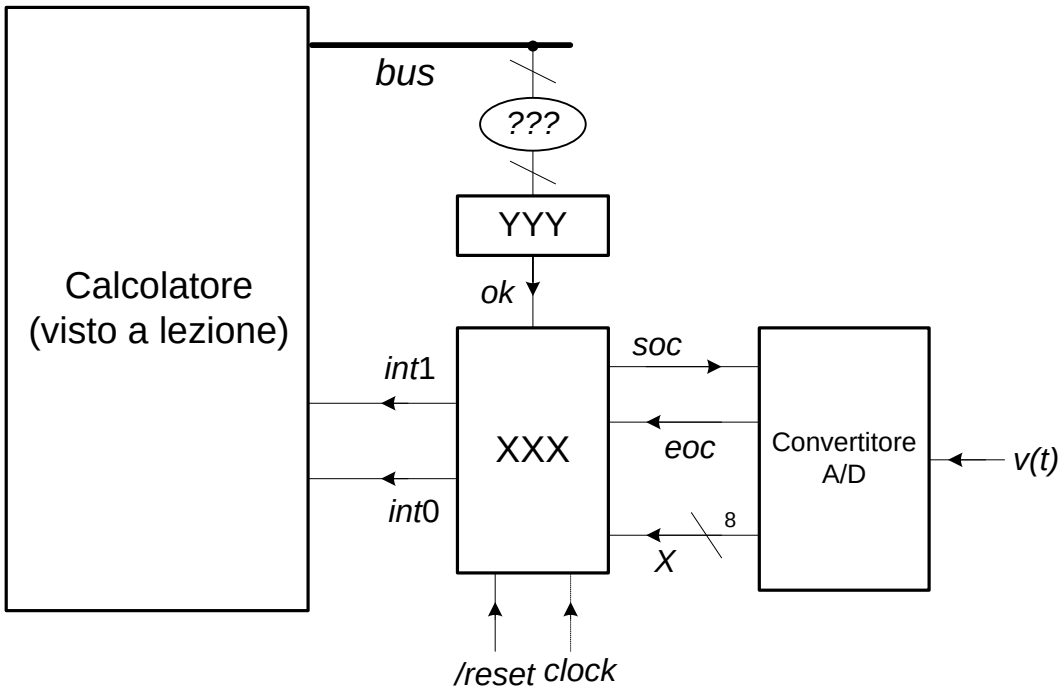



```
module XXX(a0,ior_,d7_d0, dav_,rfd,byte, clock,reset_);
 input         clock,reset_;
 output        a0, ior_;
 input  [7:0]  d7_d0;
 input         rfd;
 output        dav_;
 output [7:0]  byte;
 reg           A0,IOR_;   assign a0=A0; assign ior_=IOR_;
 reg           DAV_;      assign dav_=DAV_;
 reg [7:0]     BYTE;      assign byte=BYTE;
 reg [2:0]     STAR; parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6;
 function [7:0] elaborazione;
  input [7:0] d7_d0;
   begin
    elaborazione[7]=d7_d0[7];
    elaborazione[6]=d7_d0[6];
    elaborazione[5]=d7_d0[5]&d7_d0[7];
    elaborazione[4]=d7_d0[4];
    elaborazione[3]=d7_d0[3]&d7_d0[7];
    elaborazione[2]=d7_d0[2];
    elaborazione[1]=d7_d0[1]&d7_d0[7];
    elaborazione[0]=d7_d0[0];
   end
 endfunction

 always @(reset_==0) begin A0<=0; IOR_<=1; DAV_<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0:  begin IOR_<=0; STAR<=S1; end
    S1:  begin IOR_<=1; STAR<=(d7_d0[0]==1)?S2:S0; end
    S2:  begin A0<=1; STAR<=S3; end
    S3:  begin IOR_<=0; STAR<=S4; end
    S4:  begin IOR_<=1; BYTE<=elaborazione(d7_d0); STAR<=S5; end
    S5:  begin DAV_<=0; STAR<=(rfd==1)?S5:S6; end
    S6:  begin A0<=0; DAV_<=1; STAR<=(rfd==0)?S6:S0; end
   endcase
endmodule
```

ESERCIZIO 9

Con una cadenza pari a 1023 cicli di clock, l'Unità **XXX** compie le seguenti operazioni:

1) Preleva dal convertitore A/D la rappresentazione *X* in complemento a due di un nuovo campione *x*
2) Se il campione prelevato è minore di quello prelevato al ciclo precedente, invia una richiesta di interruzione tramite la variabile *int*0, altrimenti tramite la variabile *int*1
3) Rimuove la richiesta di interruzione quando il sottoprogramma di servizio, tramite l'interfaccia *YYY* gli fa giungere un impulso sulla variabile *ok*.



-) Individuare l'interfaccia YYY, montarla nello spazio di I/O a un offset di vostra scelta e precisare quali istruzioni debbono prevedere i sottoprogrammi di servizio per provocare la generazione di un impulso su *ok*.
-) Descrivere e sintetizzare l'Unità XXX, chiarendo la struttura della rete combinatoria che indica che "è maggiore"

NOTE

Si ammetta che 1023 cicli di clock siano ampiamente sufficienti a svolgere tutte le operazioni senza creare alcun problema di alcun tipo. Si supponga che l'impulso su *ok* e duri a sufficienza da essere sicuramente visto dall'Unità XXX

Una possibile soluzione

L'interfaccia *YYY* è una versione (semplificabile) di una interfaccia parallela di uscita senza handshake, in cui il bit meno significativo fornisca *ok*. Supponendo di montarla nello spazio di I/O all'offset 0x0200, le istruzioni da prevedere nei sottoprogrammi di servizio sono:

```
MOV $0x01,AL
OUT AL,[0x0200]
MOV $0x00,AL
OUT AL,[0x0200]
```

Per montare l'interfaccia nello spazio di I/O all'offset 0x0200, basta una maschera che riceva in ingresso il bus a indirizzi *a*15_*a*0 e metta a 0 la variabile di selezione */s* dell'interfaccia quando l'indirizzo sul bus è 0x0200.

Una possibile descrizione Verilog dell'unità XXX è la seguente:

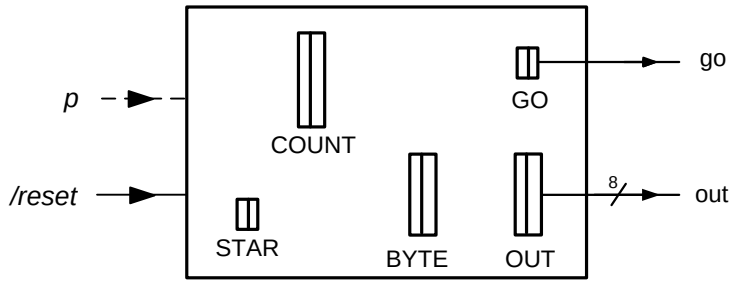
```
//-------------------------------------------------------------------------------
module XXX(soc,eoc,X, int1,int0,ok, clock,reset_);
 input       clock,reset_;
 input       eoc,ok;
 input [7:0] X;
 output      soc,int1,int0;

 reg [2:0] STAR;
 parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;

 reg       SOC;
 reg [1:0] INT;
 reg [7:0] OLD;
 reg [9:0] COUNT;

 assign soc=SOC;
 assign int1=INT[1];
 assign int0=INT[0];

 wire [8:0] differenza;  //si estende per non avere traboccamento
 assign differenza={X[7],X}-{OLD[7],OLD};

 parameter Num_Periodi=1023;
 always @(reset_==0) begin SOC<=0; COUNT<=Num_Periodi; INT<=0; STAR<=S0; end
 always @(posedge clock) if (reset_==1) #3
   casex(STAR)
    S0:  begin COUNT<=COUNT-1; SOC<=1; STAR<=(eoc==1)?S0:S1; end
    S1:  begin COUNT<=COUNT-1; SOC<=0; STAR<=(eoc==0)?S1:S2; end
    S2:  begin COUNT<=COUNT-1; INT<=(differenza[8]==1)?'B01:'B10; OLD<=X;
               STAR<=S3; end
    S3:  begin COUNT<=COUNT-1; STAR<=(ok==0)?S3:S4; end
    S4:  begin COUNT<=COUNT-1; STAR<=(ok==1)?S4:S5; end
    S5:  begin INT<='B00; COUNT<=(COUNT==1)?Num_Periodi:(COUNT-1);
               STAR<=(COUNT==1)?S0:S5; end
   endcase
endmodule
//-------------------------------------------------------------------------------
```

ESERCIZIO 10

Descrivere e sintetizzare l'Unità *XXX* che emette un byte generato un in accordo alla legge di cui sotto. Il byte deve permanere all'uscita *out* di *XXX* per un numero di clock esattamente pari a *numero_clock* = *byte* * 2 e deve essere notificato dal fatto che la variabile *go* passa da 0 ad 1 per un ciclo di clock.

Legge di generazione dei byte: I *byte* generati soddisfano la doppia condizione di essere numeri *dispari* e *multipli di tre*

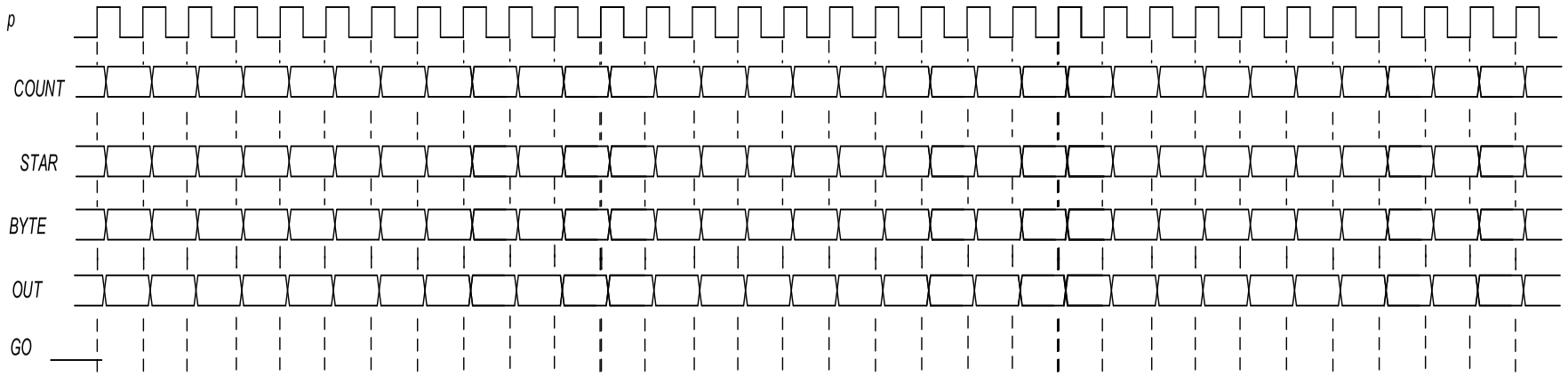
Tracciare il diagramma di temporizzazione come verifica della correttezza della descrizione dell'unità *XXX*

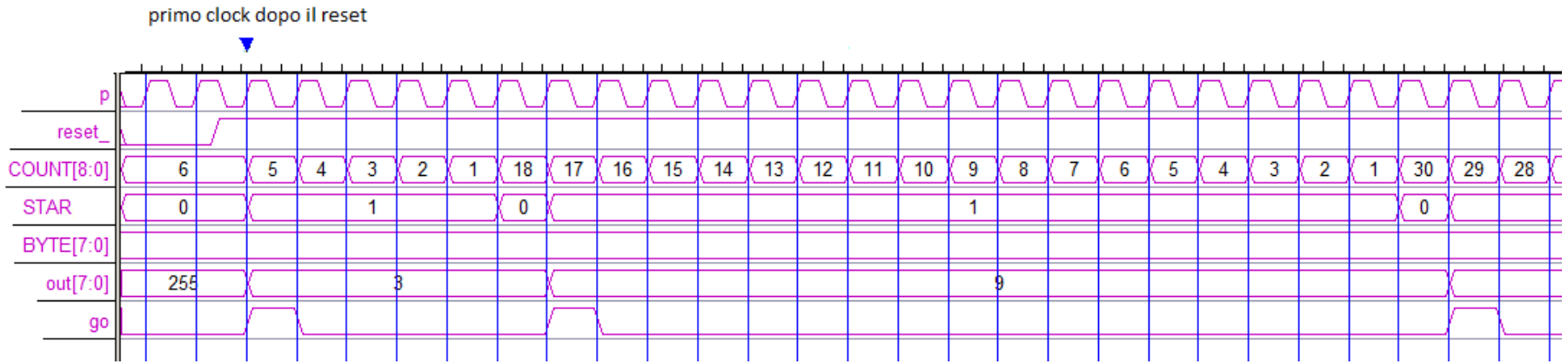


Unità *XXX*

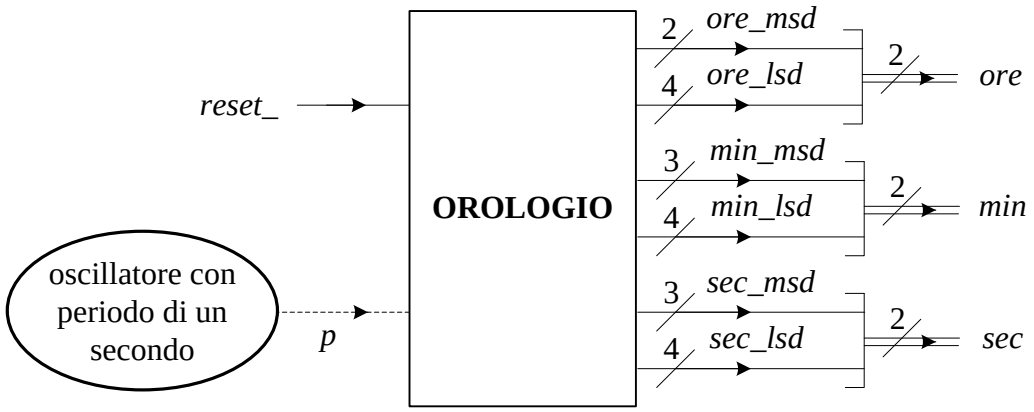
Una possibile soluzione

```
module XXX(out,go,clock,reset_);
input        clock, reset_;
output       go;
output [7:0] out;
reg        GO;          assign go=GO;
reg [7:0]  OUT,BYTE;    assign out=OUT;
reg [8:0]  COUNT;
reg STAR;   parameter S0=0,S1=1;
wire[7:0]  new_byte = (BYTE==255)?3:(BYTE+6);
wire[8:0]  Num_Cicli = {BYTE,1'B0};
always @(reset_==0) begin GO<=0; BYTE<=3; COUNT<=6; STAR<=S0; end
always @(posedge clock) if (reset_==1) #3
 if else #3
  casex(STAR)
   S0:  begin OUT<=BYTE; GO<=1; BYTE<=new_byte; COUNT<=(COUNT-1); STAR<=S1; end
   S1:  begin  GO<=0;  COUNT<=(COUNT==1)?Num_Cicli:(COUNT-1);  STAR<=(COUNT==1)?S0:S1;
end
  endcase
endmodule
```

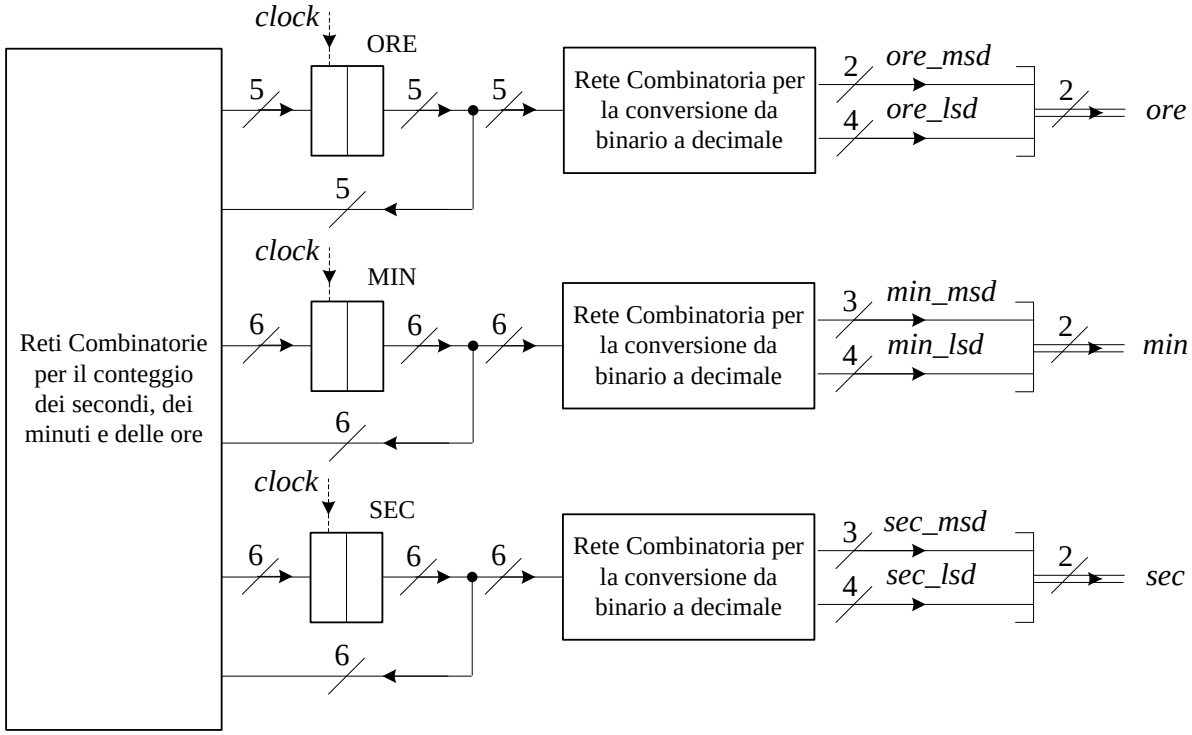

p
COUNT
STAR
BYTE
OUT
GO


primo clock dopo il reset
p
reset_
COUNT[8:0]
STAR
BYTE[7:0]
out[7:0]
go
6 5 4 3 2 1 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 30 29 28
0 1 0 1 0
255 3 9

ESERCIZIO 11 : Un OROLOGIO



Una possibile soluzione




```
module OROLOGIO(ore_msd,ore_lsd, min_msd,min_lsd, sec_msd,sec_lsd, clock,reset_);
 input        clock,reset_;
 output [1:0] ore_msd; output [3:0] ore_lsd;
 output [2:0] min_msd; output [3:0] min_lsd;
 output [2:0] sec_msd; output [3:0] sec_lsd;
 reg [4:0] ORE; reg [5:0] MIN; reg [5:0] SEC;
 assign sec_msd=SEC/10, sec_lsd=SEC%10;
 assign min_msd=MIN/10, min_lsd=MIN%10;
 assign ore_msd=ORE/10, ore_lsd=ORE%10;
 always @(reset_==0) begin ORE<=0; MIN<=0; SEC<=0; end
 always @(posedge clock) if (reset_==1) #3
    begin SEC<=(SEC==59)?0:(SEC+1);
          MIN<=((MIN==59)&(SEC==59))?0:((MIN!=59)&(SEC==59))?(MIN+1):MIN;
          ORE<=((ORE==23)&(MIN==59)&(SEC==59))?0:
               ((ORE!=23)&(MIN==59)&(SEC==59))?(ORE+1):ORE; end
endmodule
```