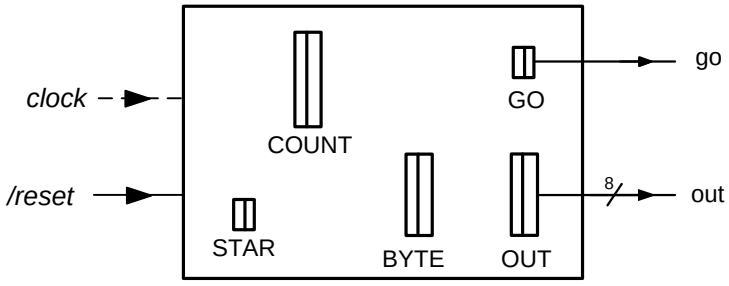
Esercizi finali sulle reti sincronizzate complesse

ESERCIZIO 1

Descrivere e sintetizzare l'Unità *XXX* che emette un byte generato 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*

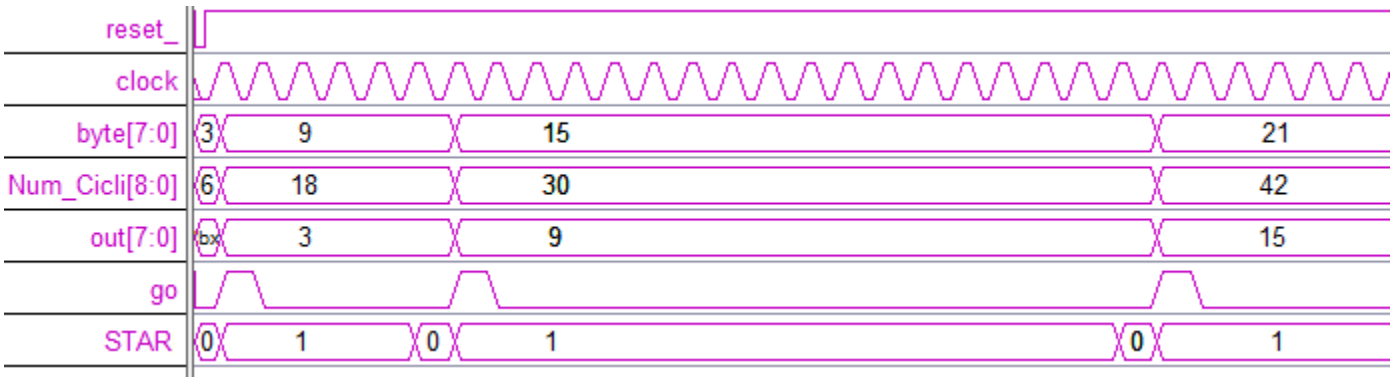
Tracciare il diagramma di temporizzazione come verifica della correttezza della descrizione dell'unità *XXX*



Unità *XXX*

Una possibile soluzione

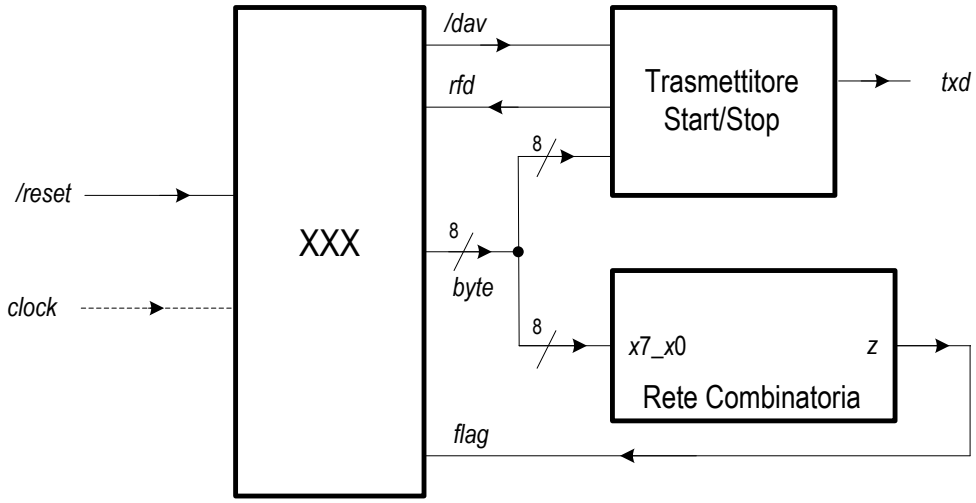
```
module XXX(out,go,clock,reset_);
input        clock, reset_;
output       go;
output [7:0] out;
reg        GO;          assign go=GO;
reg [7:0]  OUT,BYTE;    assign out=OUT;
reg [8:0]  COUNT;
reg STAR;   parameter S0=0,S1=1;
wire[8:0]  Num_Cicli = {BYTE,1'B0};
always @(reset_==0) begin GO<=0; BYTE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



ESERCIZIO 2

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



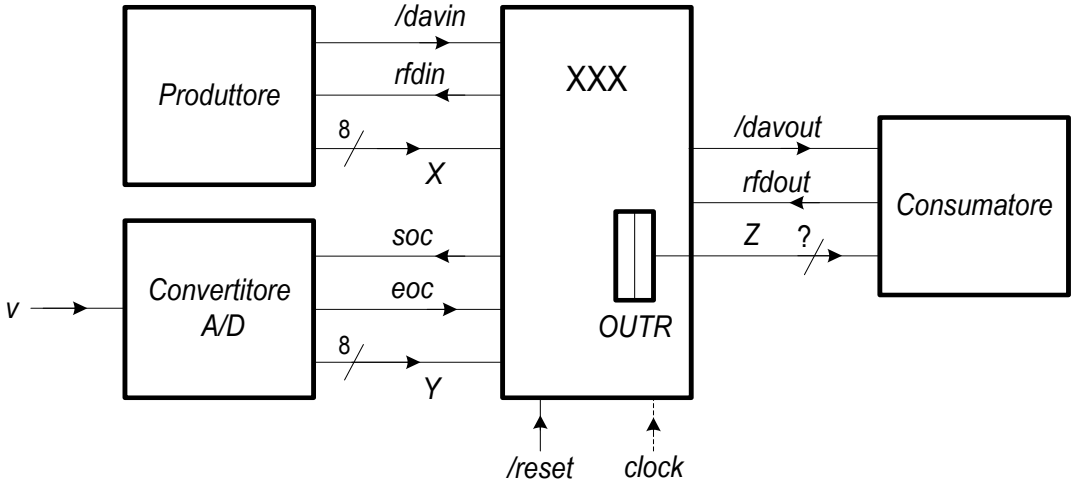
Descrivere e sintetizzare l'Unità XXX.

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 3

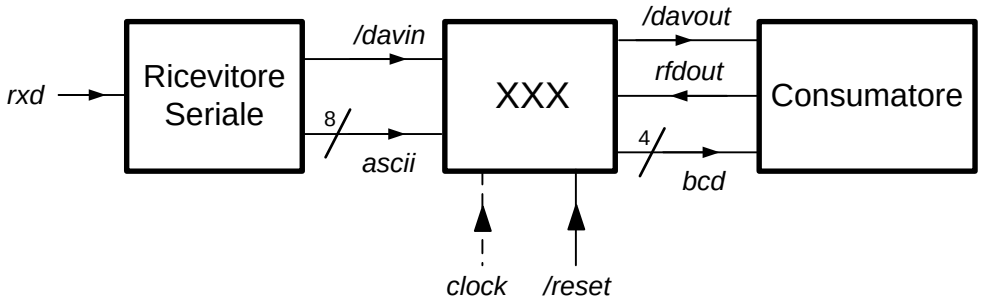
X, Y e Z rappresentano, in complemento a due, tre numeri *x*, *y*, e *z*. L'Unità XXX si comporta all'infinito come segue: Ogni volta che riceve un nuovo byte X dal Produttore, preleva un nuovo byte Y dal Convertitore A/D e invia al Consumatore una nuova configurazione Z, tale che sia $z = x/2 + 2y$. 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 4



Descrivere il circuito XXX che si evolve come segue:

1) preleva un byte dal ricevitore seriale e lo interpreta come la codifica *ascii* di un carattere;
2) se la *codifica* è di una cifra decimale, invia al consumatore i quattro bit che esprimono tale cifra in codifica *bcd* e torna al punto 1, altrimenti torna immediatamente al punto 1

Per verificare se la codifica *ascii* di un carattere è o non è quella di una cifra decimale si usi una funzione *mia_rete*(*ascii*) che genera 1 se il test ha successo, 0 altrimenti.

Trovare l'espressione algebrica (possibilmente minima) per *mia_rete*(*ascii*) e **disegnare** la porzione di parte operativa relativa al registro BCD che supporta la variabile di uscita *bcd*

NOTA SEMPLIFICATIVA: L'intervallo di tempo tra l'arrivo di un byte e l'altro è talmente grande da non generare alcun problema di nessun tipo a nessuno circuito.

Una possibile soluzione

```
module XXX (ascii,davin_, bcd,davout_,rfdout, clock,reset_);
 input          clock,reset_;
 input          davin_, rfdout;
 output         davout_;
 input   [7:0] ascii;
 output  [3:0] bcd;
 reg            DAVOUT_;     assign davout_=DAVOUT_;
 reg     [3:0] BCD;         assign bcd=BCD;
 reg            TEST;
 reg     [2:0] STAR;        parameter S0=0,S1=1,S2=2,S3=3,S4=4;
 function mia_rete;
  input [7:0] ascii;
  casex(ascii)
    'B00110???: mia_rete=1;
    'B0011100?: mia_rete=1;  // ovvero (è un'utile sottigliezza) 'B0011?00?: mia_rete=1
      default : mia_rete=0;
  endcase
 endfunction
always @(reset_==0) #1 begin DAVOUT_<=1; STAR<=S0; end
always @(posedge clock) if (reset_==1) #3
  casex(STAR)
   S0:    begin BCD<=ascii[3:0]; TEST<=mia_rete(ascii);
          STAR<=(davin_==0)?S1:S0; end
   S1:    begin STAR<=(davin_==0)?S1:S2; end
   S2:    begin STAR<=(TEST==1)?S3:S0; end
   S3:    begin DAVOUT_<=0; STAR<=(rfdout==1)?S3:S4; end
   S4:    begin DAVOUT_<=1; STAR<=(rfdout==1)?S0:S4; end
  endcase
endmodule
```

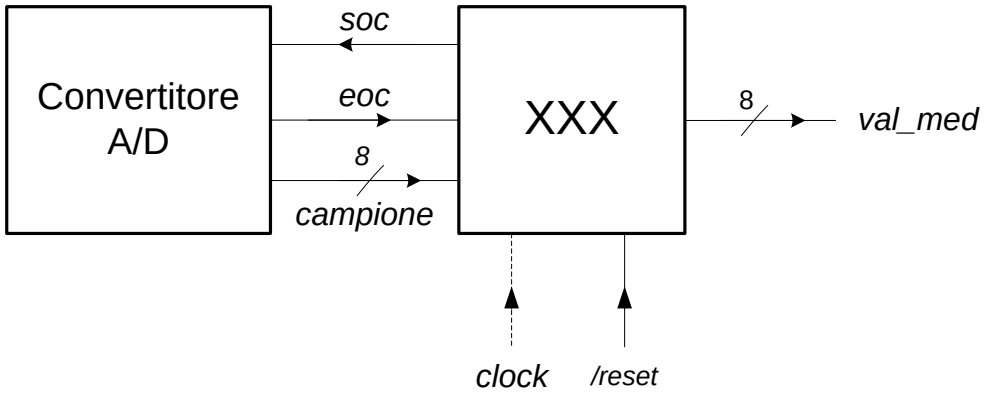
Forma minima di *mia_rete*, indicando con c_i la variabile `ascii[i]`:

$$mia\_rete = \overline{c}_7\overline{c}_6c_5c_4\overline{c}_3 + \overline{c}_7\overline{c}_6c_5c_4\overline{c}_2\overline{c}_1$$

ESERCIZIO 5

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.

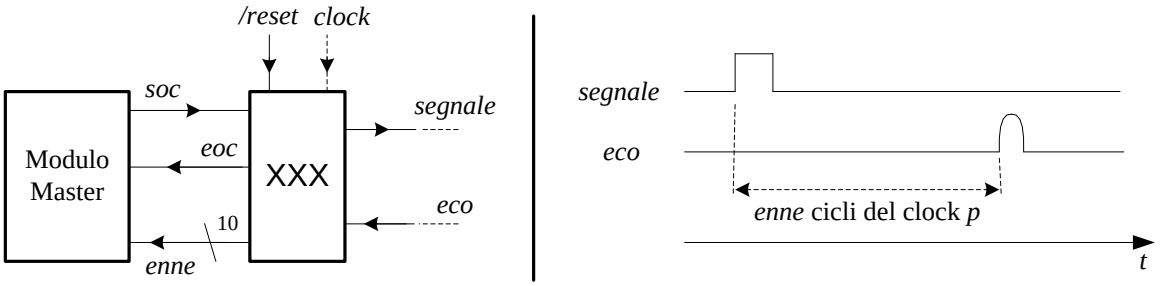


Una possibile soluzione

```
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 6

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.

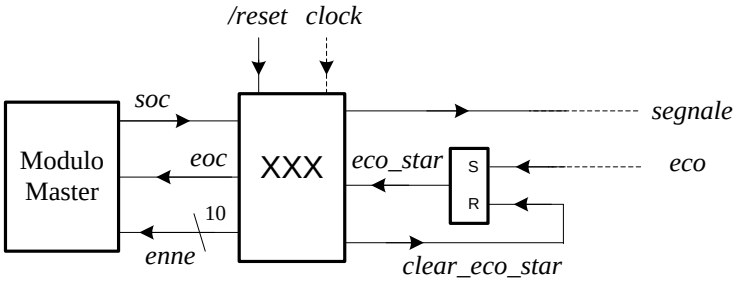


Quando l'Unità *XXX* viene attivata dal Modulo Master emette, tramite la variabile di uscita *segnale*, un impulso di durata pari ad un ciclo di clock e ne raccoglie, tramite la variabile di ingresso *eco*,una eco proveniente da un ostacolo (il ritardo con cui arriva l'eco è proporzionale alla distanza 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

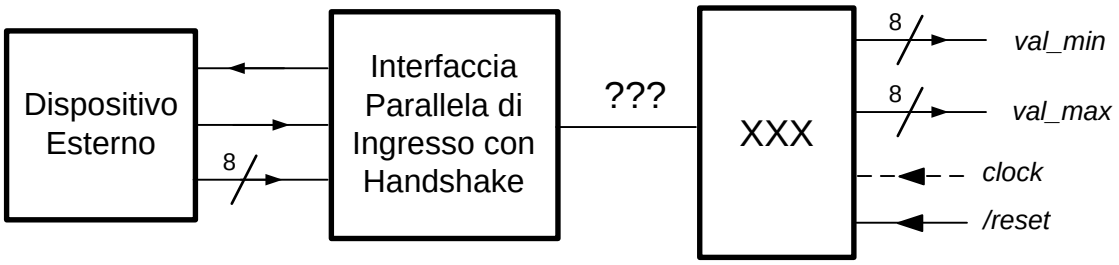
Nota 2: L'eco è un impulso molto distorto e spesso molto breve, che potrebbe non essere visto dall'Unità *XXX*, se non viene inserito un circuito che lo catturi e lo presenti al'Unità *XXX* in modo sicuro.

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
```

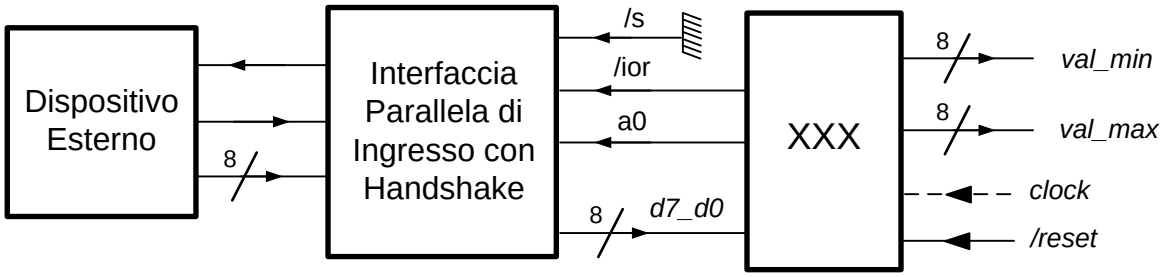
ESERCIZIO 7



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.

Una P Soluzione : Si sceglie di gestire l'interfaccia testandone il flag FI

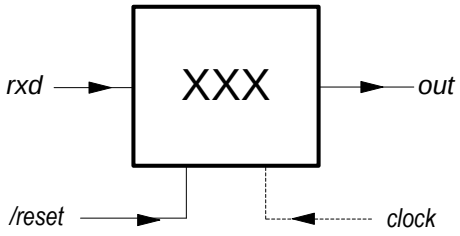



```
module XXX (ior_,a0,d7_d0,val_min,val_max,clock,reset_);
 input        clock,reset_;
 output       ior_,a0;
 input  [7:0] d7_d0;
 output [7:0] val_min, val_max;
 reg A0;              assign a0=A0;
 reg IOR_;            assign ior_=IOR_;
 reg [7:0] VAL_MIN;   assign val_min=VAL_MIN;
 reg [7:0] VAL_MAX;   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
```

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

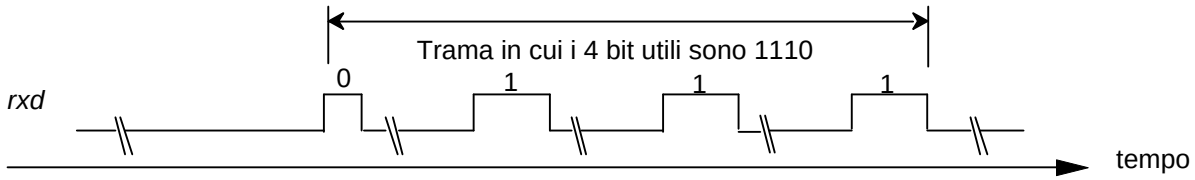
L'Unità *XXX* è, rispetto alla variabile di ingresso *rxd*, un ricevitore seriale *di trame con 4 bit utili*. Ogni volta che ha ricevuto una trama, *XXX* ne confronta i due bit più significativi con i due bit meno significativi: se il confronto dà esito positivo (coincidenza), *XXX* mette ad 1 la variabile *out* per un ciclo di clock, altrimenti lascia tale variabile a 0. Torna quindi ad aspettare una nuova trama, e così via all'infinito.



Il formato delle trame è illustrato sotto ed è *estremamente diverso* da quello delle trame viste a lezione e non ci sono ne' il bit di START ne' il bit di STOP. Il bit ricevuto per primo è comunque, come nelle trame viste a lezione, il bit meno significativo dei 4 bit utili.

In dettaglio:

- Tra un bit utile e un altro e tra una trama e un'altra, *rxd* sta a 0 per un tempo imprecisato, ma sufficientemente lungo da non creare alcun problema di alcun tipo;
- L'arrivo di un bit utile è notificato dalla circostanza che *rxd* va a 1;
- La durata della permanenza di *rxd* a 1 indica se un bit utile vale 1 oppure 0, in accordo alle seguenti specifiche:
  a) *rxd* permane a 1 esclusivamente per 5 o per 10 cicli di clock
  b) Se *rxd* permane a 1 per 5 (cioè per **0**101) cicli di clock, allora il bit utile vale **0**
  c) Se *rxd* permane a 1 per 10 (cioè per **1**010) cicli di clock, allora il bit utile vale **1**



NOTE: **1)** Usare un registro a 4 bit di nome DURATA per memorizzarvi la permanenza di *rxd* ad 1. Un registro a 4 bit di nome BUFFER per memorizzarvi i quattro bit utili della trama via via che arrivano. Un registro di nome COUNT per contare e verificare che i quattro bit della trama siano arrivati.
2) Descrivere *XXX* e **sintetizzare e disegnare lo schema** della parte operativa relativa al registro BUFFER

Una possibile soluzione

```
module XXX(out,rxd, clock,reset_);
  input        clock,reset_;
  input        rxd;
  output       out;
  reg [3:0] DURATA;
  reg [3:0] BUFFER;
  reg [2:0] COUNT;
  reg       OUT;  assign out=OUT;
  reg [1:0] STAR; parameter S0=0,S1=1,S2=2,S3=3;
  // Come evidenziato nel testo, il bit DURATA[3] coincide con il bit utile
  wire bit_utile; assign bit_utile=DURATA[3];
  always @(reset_==0) begin OUT=0; DURATA<=0; COUNT<=4; STAR<=S0; end
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
     S0: begin OUT<=0; DURATA<=DURATA+rxd; STAR<=(rxd==0)?S0:S1; end
     S1: begin DURATA<=DURATA+rxd; STAR<=(rxd==1)?S1:S2; end
     S2: begin BUFFER<={bit_utile,BUFFER[3:1]};
               COUNT<=COUNT-1; DURATA<=0; STAR<=(COUNT==1)?S3:S0; end
     S3: begin OUT<=(BUFFER[3:2]== BUFFER[1:0])?1:0; COUNT<=4; STAR<=S0; end
    endcase
endmodule
```

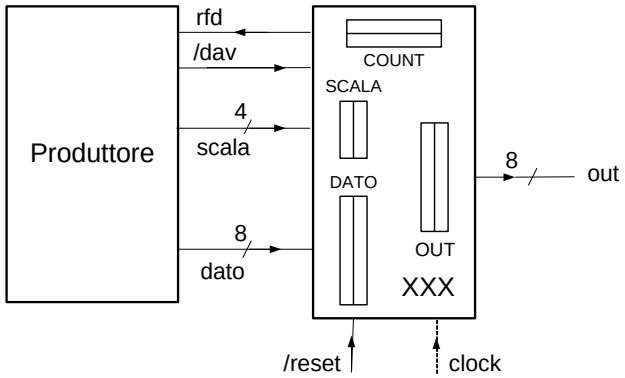
ESERCIZIO 9

Descrivere e sintetizzare l'Unità XXX che preleva dal Produttore un'informazione costituita da una coppia di numeri naturali, uno a 4 bit (*scala*) e l'altro ad 8 bit (*dato*) e presenta in uscita, senza modificarlo, il numero *dato*. Torna poi a prelevare un'altra coppia di numeri e così via all'infinito.

Ogni *dato* deve permanere in uscita per un tempo pari a (*scala* x 8) cicli di clock, dopo di che deve essere sostituito dal *dato* prelevato da XXX al round successivo.

NOTE

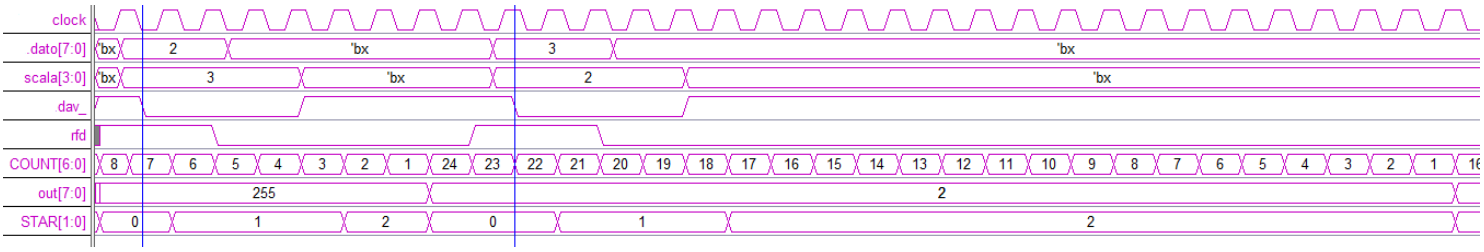
1. Si supponga che: i) il valore di *scala* sia maggiore di 0 e ii) il produttore sia sufficientemente veloce da chiudere ogni handshake in un tempo non noto, diverso da round a round, ma in ogni caso abbastanza inferiore a 8 cicli di clock (caso peggiore per XXX).
2. Si supponga che al reset iniziale l'Unità XXX si comporti come se avesse ricevuto *scala*=1 e *dato*=255.
3. Si consiglia di usare, oltre al registro STAR e al registro RFD non riportati in figura, i quattro registri che vi sono riportati.
4. La permanenza di ogni *dato* in uscita deve essere esattamente (*scala* x 8) cicli di clock e quindi si suggerisce di fare una piccola simulazione di verifica.



UNA SOLUZIONE (la più semplice)

```
module XXX(rfd,dav_,scala,dato,out,clock, reset_);
 input reset_, clock;
 input[3:0] scala;
 input[7:0] dato;
 input dav_;
 output rfd;
 output[7:0] out;
 reg RFD; assign rfd = RFD;
 reg[7:0] OUT; assign out = OUT;
 reg[7:0] DATO;
 reg[3:0] SCALA;
 reg[6:0] COUNT;
 reg[1:0] STAR; parameter S0=0, S1=1, S2=2;
 wire[6:0] Num_Periodi; assign Num_Periodi={SCALA,3'B000};
 always @(reset_==0) #1 begin COUNT<=8; OUT<=255; STAR<=S0; RFD<=1; end
 always @(posedge clock) if(reset_==1) #3
 casex(STAR)
   S0: begin COUNT<=COUNT-1; RFD<=1; DATO<=dato; SCALA<=scala;
       STAR<=(dav_==0)?S1:S0; end
   S1: begin COUNT<=COUNT-1; RFD<=0; STAR<=(dav_==1)?S2:S1; end
   S2: begin COUNT<=(COUNT==1)?Num_Periodi:(COUNT-1); OUT<=(COUNT==1)?DATO:OUT;
       STAR<=(COUNT==1)?S0:S2; end
  endcase
endmodule
```

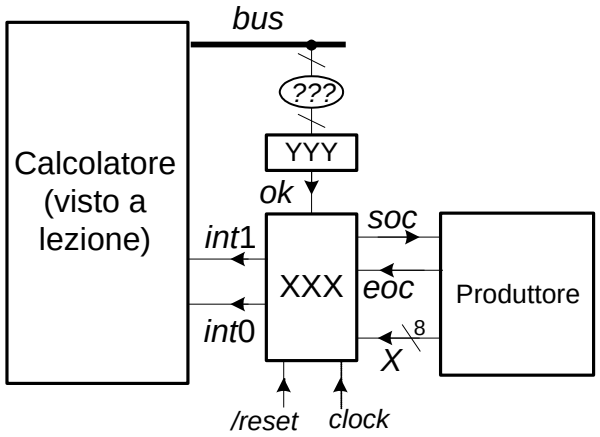
Al reset *out* sta a 255 per 8 cicli di clock. Poi XXX ascolta il Produttore. Nella simulazione il Produttore invia *dato*=2 e *scala*=3 (linea rossa sul *dav_* che va a 0) e quindi *out* sta a 2 per 24 cicli di clock. Nel foglio non entra un diagramma più lungo. Comunque si vede che la seconda volta, il produttore invia *dato*=3 e *scala*=2



ESERCIZIO 10

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

1) Preleva dal Produttore la rappresentazione *X* in <u>complemento a due</u> di un numero *x*
2) Se il numero 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"

NOTA: Si ammetta che 1023 cicli di clock siano 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 <u>essere sicuramente visto</u> 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:

```
outport(0x0200,1)
outport(0x0200,0)
```

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 (*il confronto fra il nuovo e il vecchio numero viene effettuato passando alla rappresentazione in binario bipolare*):

```
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;    assign soc=SOC;
 reg [1:0] INT;    assign int1=INT[1]; assign int0=INT[0];
 reg [7:0] OLD;
 reg [9:0] COUNT;  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<=({!X[7],X[6:0]}<{!OLD[7],OLD[6:0]})?'B01:'B10;
               OLD<=X; STAR<=S3; end
    S3:  begin COUNT<=COUNT-1; STAR<=(ok==0)?S3:S4; end
    S4:  begin COUNT<=COUNT-1; INT<='B00; STAR<=(ok==1)?S4:S5; end
    S5:  begin COUNT<=(COUNT==1)?Num_Periodi:(COUNT-1); STAR<=(COUNT==1)?S0:S5; end
   endcase
endmodule
```