

# UART

(UNIVERSAL ASYNCHRONOUS RECEIVER - TRANSMITTER)

La UART indica una periferica che converte i dati in ingresso e in uscita in una sequenza seriale.

La trasmissione attraverso un'interfaccia UART è seriale ed avviene in maniera asincrona.

Il primo parametro da definire è il BAUD RATE che serve a determinare la frequenza a cui sono spediti o campionati i dati. Per ricevere correttamente i dati il ricevitore e il trasmettitore devono concordare sulla velocità di trasmissione affinché essa sia considerabile affidabile.

Uno dei vantaggi dati dalla UART è che utilizza solo due fili per trasmettere dati tra i dispositivi.

La UART agisce come interfaccia tra comunicazioni seriali e parallele, ossia prende in ingresso un byte di dati e li invia in maniera sequenziale lungo una linea dati (TX).

A destinazione tali dati, campionati sulla linea dati (RX) vengono assemblati nuovamente in byte.

Questo tipo di interfaccia sono in grado di:

- Creare i pacchetti di dati, aggiungendo all'informazione (ricevuta attraverso data bus dal produttore) i bit di parità (optional) e sincronizzazione per la trasmissione bit a bit sulla linea TX.
- Comprendere in forma sequenziale i singoli bit ricevuti sulla linea RX, in accordo con il BAUD RATE stabilito, eliminando i bit di sincronizzazione e parità ed estraendendo solo la parte relativa all'informazione vera e propria per ricevere l'informazione in forma parallela.

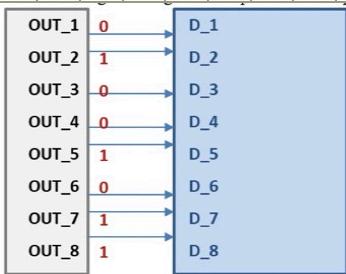
Parametri di configurazione di una UART:

- **BAUD RATE**: Determina la velocità di trasmissione dei dati (Es: 9600 baud = 9600 bps). Quando l'interfaccia UART ricevente riceve un bit di sinc., allora inizia a leggere i bit in ingresso alla frequenza specificata dal BAUD RATE.
- **Numeri di bit**: Rappresenta il DATA chunk nel framing, molto spesso equivale a un byte.
- **Parity bit**: È optional, quindi può essere on/off.
- **Stop bit**: Possono essere presenti più bit di stop, tipicamente il range è 0-2.

La linea di trasmissione dati TX di un'interfaccia UART è normalmente mantenuta attiva alta quando non c'è trasmissione (ovvero quando la linea e il sistema si trovano in IDLE). Il trasferimento di dati viene

avviato mettendo un valore logico a su TX per almeno un bit di durata (start bit). Quando l'interfaccia UART ricevente rileva sulla porta RX la transizione alto-basso, inizia a leggere i bit di ingresso alla frequenza fissata dal BAUD RATE. Tipicamente si inizia per primo l'LSB (Least Significant Bit).

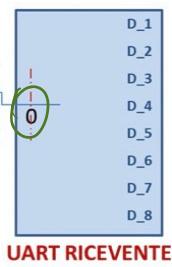
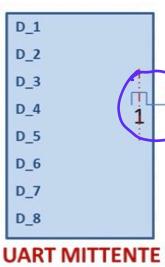
Per segnalare la fine del pacchetto dati, la UART mittente mette su TX un valore logico 1.



1) La UART mittente riceve dal produttore il byte (01001011) da trasmettere. Il bit meno significativo, che da protocollo verrà spedito per primo, è D\_8.



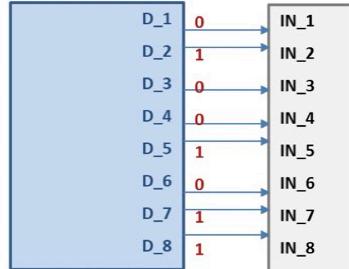
2) La UART mittente crea il pacchetto di dati da trasmettere sequenzialmente. In questo caso il parity (odd parity, ossia il bit di parità dispari) è in rosso perché non presente nella nostra implementazione base. I bit di informazione sono inseriti nel pacchetto a partire da quello meno significativo.



3) Il pacchetto di dati creato dalla UART mittente viene spedito sequenzialmente dalla sua porta TX a quella RX della UART ricevente. La velocità di trasmissione sulla porta TX della UART mittente e quella di campionamento della porta RX della UART ricevente sono fissate dal baud rate. Ossia, i bit del pacchetto sono trasmessi lungo la linea TX-RX spaziatamente (linee rosse verticali) di un tempo compatibile con il baud rate. Con il baud rate scelto ci aspettiamo di ricevere un totale di 9600 bit ogni secondo.



4) La UART ricevente isola l'informazione ricevuta, scartando (bit barrati) i campi del frame che non sono data chunk.



Spiegazione fornita dalle dispense del professore Rubattu.

Bit di start, l'inizio della comunicazione è attivo bassa

Bit di stop, attivo alto

Bit di parità, opzionale

5) La UART ricevente ricrea l'informazione, salvando i bit di informazione in un registro interno alla UART ricevente nell'ordine corretto (01001011), e la trasmette al consumatore.

## Cavale di trasmissione

L'operazione di trasmissione è la più semplice poiché la temporizzazione non è determinata dallo stato della linea e non è neanche vincolata a intervalli di temporizzazione fissi.

Quando il produttore inserisce un byte nel cavale di trasmissione della UART, quest'ultimo può:

- 1) generare un bit di inicio e inviarlo alla linea dati TX;
- 2) spostare, uno a uno, i bit ricevuti dal produttore sulla linea dati TX;
- 3) generare e inviare il bit di parità (se utilizzato) sulla linea TX;
- 4) inviare su TX gli eventuali bit di stop.

In una UART ad alte prestazioni potrebbe essere presente un buffer First-In-First-Out (FIFO) di trasmissione, affinché il produttore possa inserire più byte consecutivamente (modalità di trasmissione burst: più informazioni di

una di seguito all'altra) invece di dover depositare un'informazione alla volta nel canale di trasmissione. Dal momento che la velocità di funzionamento della CPU e quella di trasmissione sono disaccoppiate, fino a che la trasmissione corrente non viene completa (sia che si parli di una singola informazione che di quelle di tutti i byte memorizzati nell'eventuale FIFO) la UART espone verso il produttore il "flag" di stato occupato.

### Interfaccia del canale di trasmissione della UART

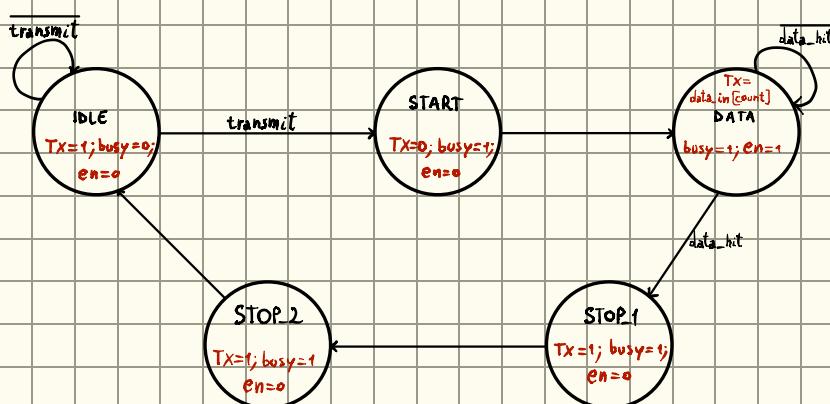
```
module top_tx #(parameter DATA_BW=8, // N bit di informazione
               parameter DATA_BW_BIT=4, // Config. CONTATORE bit sopra
               parameter BAUD_RATE=9600 // Band Rate (in questo caso 9600)
               parameter BAUD_COUNT=10416, // clk cycles (cc) per Band Rate (BR)
               parameter BAUD_BIT=14 // Config. CONTATORE cc per BR
               );

```

```
  input wire clk, rst,
  input wire transmit, // richiesta di trasmissione
  input wire [DATA_BW-1:0] data, // data_chunk da trasferire
  output wire TX, // porta seriale di trasmissione
  output wire busy // "flag" di stato occupato
);
```

### Macchina a stati del canale di trasmissione

Input: transmit, data\_bit. "transmit" serve per la richiesta di trasmissione, "data\_bit" scandisce la completa trasmissione dei bits d'informazione.



Poiché sono il valore del registro in ingresso che memorizza il dato da trasmettere,  $data\_in$  da 8 bit, e il valore di un contatore,  $count$  a + bit, che serve a scandire i bit di informazione spediti su TX.

## Evoluzione degli stati

IDLE: stato di riposo del sistema.

START: trasmissione del bit di start.

DATA: trasmissione di tutti i bit di informazione.

STOP\_1: trasmissione del primo stop bit.

STOP\_2: trasmissione del secondo stop bit.

Al fine di generare il segnale di uscita TX è necessario che la macchina a stati sia in grado di scandire correttamente i bit di informazione su un registro data\_in(data 8 bit), leggibili una volta da 0 a N-1, attraverso un contatore. La scansione avviene dal bit meno significativo. Le altre due uscite sono enable(en) del contatore dei bit di informazione (che avanza solo quando siamo nello stato DATA) e un segnale di "busy" che indica lo stato di "occupato" del canale di trasmissione.

```
module fsm_t #( parameter DATA_BW=8,  
parameter DATA_BW_BIT=4  
)  
  
input wire clk, rst,  
input wire transmit, //transmit=1 → richiesta di trasmissione da parte del produttore  
input wire data_hit, //data_hit=1 → segnalazione della trasmissione dell'ultimo bit di informazione  
input wire [DATA_BW-1:0] data_in, //dati da trasmettere  
input wire [DATA_BW_BIT-1:0] count, //contatore dati da trasmettere  
output reg en, //en=1 → abilitazione trasmissione del dato e conteggio dati da trasmettere  
output reg busy, //busy=1 → trasmissione frame su TX  
output reg TX //TX è il segnale su uscita dalla UART contenente il frame di trasmissione  
);  
  
parameter IDLE=3'b000, START=3'b001, DATA=3'b010, STOP_1=3'b011, STOP_2=3'b100;  
reg [2:0] state, state_nxt;
```

always @ (posedge clk, posedge rst)

if (rst == 1'b1) state<= IDLE;

else state<= state\_nxt;

always @ (transmit, state, data\_hit)

case (state)

IDLE:

begin

if (transmit == 1'b1) state\_nxt = START;  
else state\_nxt = IDLE;

end

START:

state\_nxt = DATA;

DATA :

begin

if (data\_hit == 1'b1) state\_nxt = STOP\_1;  
else state\_nxt = DATA;

end

STOP\_1:

state\_nxt = STOP\_2;

STOP\_2:

state\_nxt = IDLE;

default: state\_nxt = IDLE;

endcase

always @ (state, data\_in, count)

case (state)

IDLE:

begin

```
TX=1'b1;  
en=1'b0;  
busy=1'b0;
```

end

START:

begin

```
TX=1'b0;  
en=1'b0;  
busy=1'b1;
```

end

DATA:

begin

```
TX=data_in[count];  
en=1'b1;  
busy=1'b1;
```

end

STOP\_1:

begin

```
TX=1'b1;  
en=1'b0;  
busy=1'b1;
```

end

STOP\_2:

begin

```
TX=1'b1;  
en=1'b0;  
busy=1'b1;
```

end

default:

begin

Lo UART è disponibile ad accettare

nuove transazioni solo nella fase di  
IDLE dove busy è uguale a zero.

Nella fase di IDLE la linea TX è attiva alta  
quando poi transmit è uguale a 1 si può iniziare  
a trasmettere e quindi il primo bit che viene

trasmesso nella fase di START è 0, ovvero lo  
"start bit".

Dalla fase DATA inizio a essere effettivamente  
trasmesso l'informazione infatti:

TX=data\_in[count]; ovvero viene trasmessa  
l'informazione registrata nel registro di  
input.

In questa fase l'uscita en è attiva alta e  
abilite il contatore dei dati trasmessi.

Nelle due fasi di stop viene trasmesso su

TX un bit di valore 1. Dovolmente l'enable  
dei dati trasmessi viene disabilitato.

```
TX = i'b1;  
en = i'b0;  
busy = i'b0;
```

```
end
```

```
endcase
```

```
endmodule
```

La frequenza di clock di funzionamento deve essere quella fissata dal BAUD RATE. Se si immaglia come frequenza generale del sistema lo stesso che si ha sulla Basys 3 (100 MHz  $\Rightarrow$  clock di sys) e si impone il baud rate di 9600 si avrà bisogno di un modulo generatore di baud rate, basato su un contatore. Questo contatore dovrà abilitare la trasmissione di un singolo bit su TX in modo che la velocità di trasmissione sia di 9600 bps.

L'enable per l'avanzamento della trasmissione va generato dunque a intervalli di BAUD\_CNT.

$$\text{BAUD\_CNT} = \frac{\text{frequenza clock sistema}}{\text{BAUD RATE}}$$

Il modulo generatore di baud rate determinerà sia il clock di avanzamento della macchina a stati, che pilota la trasmissione, sia del contatore che gestisce la trasmissione dei bit di informazione visto che viene spedito sulla linea dati TX un singolo bit del data chunk alla frequenza imposta dal baud rate.

generazione del clock

```
always @ (posedge clk, posedge rst)  
if (rst == i'b1) baud_clk <= i'b0;  
else if (count < BAUD_CNT/2) baud_clk <= i'b1;  
else baud_clk <= i'b0;
```

Il top module del canale di trasmissione è:

```
module top_tx #(parameter DATA_BW=8, // N bit di informazione
               parameter DATA_BIT=4, // Config. CONTATORE bit supe
               parameter BAUD_RATE=9600 // Baud Rate (in questo caso 9600)
               parameter BAUD_COUNT=10416, // clk cycles (cc) per Baud Rate (BR)
               parameter BAUD_BIT=14 // Config. CONTATORE cc per BR
               );
    input wire clk, rst,
           input wire transmit, // richiesta di trasmissione
           input wire [DATA_BW-1:0] data, // data_chunk da trasferire
           output wire TX, // porta seriale di trasmissione
           output wire busy // "flag" di stato occupato
    );
    // gestione del registro di input
    wire en_in; /* segnale di abilitazione campionamento dati da trasmettere: ALTO se il produttore richiede una
    trasmissione (transmit=1) e la UART non è impegnata su un'altra trasmissione (busy=0) */
    wire [DATA_BW-1:0] data_stored; // dati campionati
    // gestione contatore dati trasmessi
    wire en; // FSM>CONTATORE segnale di abilitazione dati trasmessi
    wire [DATA_BIT_BW-1:0] count; // CONTATORE>FSM conteggio dati
    wire hit; // CONTATORE>FSM segnala (hit=1) la trasmissione dell'ultimo bit di informazione
    wire baud_clk; // Segnale di sincronizzazione generato sulla base del baud rate
    // input register: registro di campionamento dati da trasmettere
    assign en_in = (transmit==1'b1 && busy==1'b0) ? 1'b1 : 1'b0;
```

```
register #(N(DATA_BW)) DATA_IN(.D(data), .Load(en_in), .clk(clk), .rst(rst), .Q(data_stored));
```

// input register: registro di componimento richiesta da parte del produttore

/\* questo registro interno viene SETTATO quando viene effettuata una richiesta da parte del produttore e RESETTATO quando la richiesta viene servita e si inizia la trasmissione dei dati su TX \*/

```
reg transmit_int;
```

```
always @ (posedge clk, posedge rst)
```

```
if (rst == 1'b1) transmit_int <= 1'bo;
```

```
else if (transmit == 1'b1) transmit_int <= 1'b1;
```

```
else if (en == 1'b1) transmit_int = 1'bo;
```

// clock generator: baud rate counter

// baud rate 9600 bps, system clk 100MHz → baud rate counter MAX=10416, N\_BIT=14

```
baud_rate_clk_gen #(BAUD_CNT(BAUD_COUNT), BAUD_BIT(BAUD_BIT)) BR_GEN (.clk(clk), .rst(rst), baud_clk(baud_clk)),
```

// contatore dei dati trasmessi in uscita su TX

```
counter #(MAX(DATA_BW), N_BIT(DATA_BW_BIT)) CNT_DATA (.clk(clk), .rst(rst), en(en), count(count)),
```

```
assign hit = (count == DATA_BW - 1) ? 1'b1 : 1'bo;
```

// macchina a stati di controllo della funzionalità del sistema

```
fsm_tx #(DATA_BW(DATA_BW), DATA_BW_BIT(DATA_BW_BIT))
```

```
FSM_TX (.clk(baud_clk), .rst(rst), .transmit(transmit_int), .data_hit(hit), .data_in(data_stored), .count(count),  
.en(en), .TX(TX), .busy(busy));
```

```
endmodule
```

Il produttore trasmette i dati a una frequenza diversa dalla UART, è necessario quindi dissociare la richiesta di trasmissione ( sincronizzata su clock di sistema ) e segnale transmit cui è sensibile la macchina a stati ( sincronizzata sulla frequenza di baud rate ). A questo scopo è stato inserito nel modello del canale di trasmissione della UART un registro interno di componimento della avvenuta richiesta. Il registro viene SETTATO quando viene

effettuata una richiesta e RESETATO (oltre che sul reset) quando la richiesta viene servita e inizia la trasmissione dei dati su TX.

Nel sistema sono presenti altri moduli:

- un modulo che acquisisce e memorizza il dato che arriva dal produttore (registro di campionamento richiesto da parte del produttore). L'acquisizione è abilitata se transmit=1'b1, quando c'è una richiesta di trasmissione da parte del produttore, e se busy=1'b0. Il secondo controllo può essere evitato se è previsto un controllore da parte del produttore del flag di occupato.
- un modulo che scandisce la lettura dei bit di informazione da spedire da 0 a N-1 attraverso un contatore (contatore dei dati trasmessi in uscita su TX) abilitato dalla macchina a stati quando si trova nello stato di trasmissione dei dati.

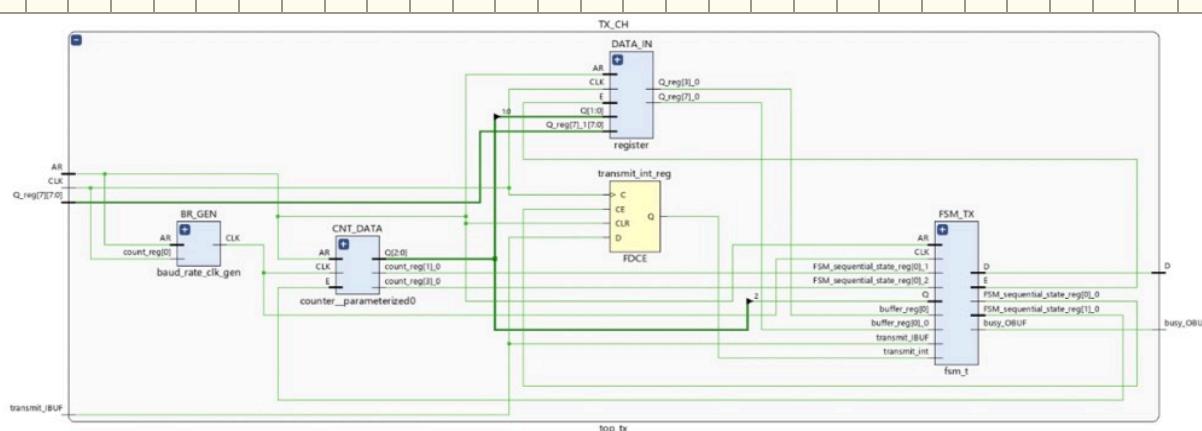


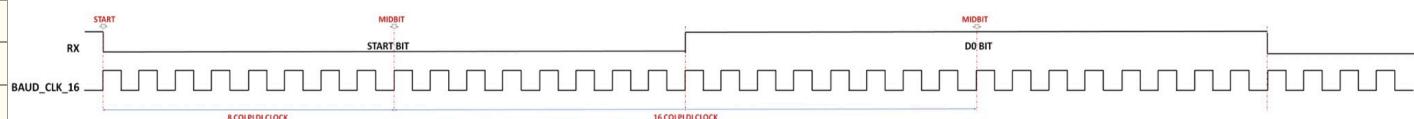
Figura 8 - UART Canale di Trasmissione: Schema a Blocchi Schema a Blocchi (Vivado Post-Sintesi).

## CANALE DI RICEZIONE

La ricezione è più complessa. Un ricevitore oscilloscopio deve sincronizzarsi con il segnale di ingresso senza l'accesso diretto al clock di trasmissione. In maniera assicurata va inserito il campionamento della sequenza di bit se e solo se viene campionato valido il bit di start. Per l'avvio deve essere identificato, in maniera affidabile, il bit di start sulla porta RX. L'identificazione viene fatta sottraendo campionando, ovvero utilizzando una frequenza di campionamento maggiore rispetto a quella di riferimento, il segnale a un multiplo della frequenza di baud rate. Sarà necessario un modulo che si occuperà di fare un "over-clocking" della velocità in bit (tipicamente si opta per un fattore oversample pari a 8x o 16x). Tale modulo deve operare come generatore di baud rate x oversample, ossia oversample volte più veloce rispetto alla velocità del clock di trasmissione.

L'obiettivo del sovra-campionamento è farlo in modo che il campionamento non inserisce errori stati dei glitch o se è presente uno sfasamento fra i due dispositivi.

Lo stato del segnale RX su ingresso viene controllato su ogni impulso di clock compatibile con il baud rate, andando a cercare l'inizio del bit di START, questo bit non avvia subito la ricezione, ma si aspetta un intervallo pari a metà del tempo di bit rappresentato dal MIDBIT. Se dopo tale tempo il MIDBIT non è ancora allo stato logico zero, esso viene generato. Il MIDBIT è determinato da OVERSAMPLE/2, dato che OVERSAMPLE è il fattore di sovra-campionamento.



Quando viene rilevato il bit di start il campionamento della linea di trasmissione può continuare alla velocità neta (dove gli intervalli sono compatti con il baud rate) per acquisire i bit di dati da memorizzare sul registro di uscita, che verrà reso disponibile dopo N periodi di bit, quindi quando il data chunk di lunghezza N è stato ricevuto.

A questo punto la UART può segnalare al consumatore che il dato è pronto, ad esempio generando un interrupt per il processore.

### Circuito ricezione

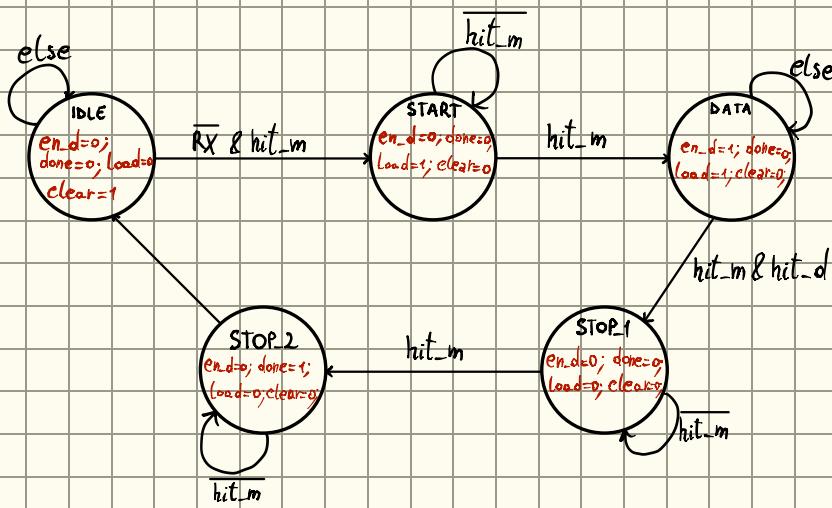
```
module top_rx #(parameter DATA_BW=8,      // N bit di informazione
               parameter DATA_BW_BIT=4,    // Config. CONTATORE bit info
               parameter BAUD_COUNT_x16=651, // CC per BR sovra-campionato (BR_X16)
               parameter BAUD_BIT_x16=10,   // Config. CONTATORE CC per BR_X16
               parameter OVER_SAMPL=16,    // Fattore di sovra-campionamento
               parameter OVER_SAMPL_BIT=5 // Config. CONTATORE OVER_SAMPL
               );
  input wire clk, rst, //segnali di controllo generali del sistema
        input wire RX,   //porta seriale di ricezione
        output wire RD,  //dato re-impacchettato valido
        output wire [DATA_BW-1:0] data_chunck //data_chunk ricevuto
);
```

In questo caso il fattore di sovra-campionamento è OVER\_SAMPL=16. Immaginando la frequenza generale della Basys 3 (100 MHz, clock di 10 ns) e baud rate di 9600 bps da sovra-campionare con fattore 16, si avrà bisogno di un modulo di generatore di baud rate sovra-campionato basato su un contatore. Questo contatore dovrà abilitare la lettura di un nuovo bit su RX facendo sì che la velocità di lettura sia di  $9600 \times 16$  bps.

$$\text{BAUD_CNT\_x16} = \frac{\text{frequenza clock di sistema}}{\text{baud rate} * \text{over-samp}}$$

Il modulo sarà lo stesso di quello del canale di trasmissione, con parametri diversi. Si aggiungerà un modulo, il MIDBIT detector (mb-detector), che identificherà il MIDBIT e lo segnalerà al controllo con hit\_m.

### Sistema di controllo



Input:

- RX: 1 bit, segnale seriale ricevuto su ingresso della VART militante
- hit\_m: 1 bit, segnalazione del raggiungimento del MIDBIT
- hit\_d: 1 bit, completa ricezione dei bit di informazione.

### Evoluzione degli stati

IDLE: stato di riposo del sistema, in cui il sistema rimane in attesa di ricevere e campionare il bit di start. Qualora questo avvenga, la ricezione può essere avviata.

START: completamento della ricezione del bit di start (grazie campionato), e campionamento del primo bit di dato.

DATA: campionamento di tutti gli altri bit di informazione fino a N

STOP\_1: completamento della ricezione del primo stop bit.

STOP\_2: completamento della ricezione del secondo stop bit.

### Vengono generate + uscite

- Load(1bit) - segnala che i singoli bit di data chunk su RX possono essere caricati sul registro interno del canale di ricezione della VART.
- clear(1bit) - ripristina i valori di default sul contatore dati ricevuti e sul registro interno dati ricevuti.
- en\_d(1bit) - abilita il campionamento dati ricevuti.
- done(1bit) - segnala che il frame è stato completamente ricevuto e che c'è un dato valido sul registro di uscita.

```

module fsm_r(
    input wire clk,rst, //segnali di controllo globali
    input wire RX, /*RX è il segnale in ingresso dalla UART trasmittente contenente il frame di trasmissione
    [start,data,stop]*/
    input wire hit_d, /*hit_d=1 --> segnalazione della ricezione dell'ultimo bit di informazione*/
    input wire hit_m, //hit_m=1 --> segnalazione identificazione MIDBIT
    /*load=1 --> segnalazione che i singoli bit di data chunk su RX possono essere caricati sul registro interno del
    canale di ricezione della UART.
    ATTENZIONE: il bit di start è quello che fa passare da IDLE a START, da START in poi i successivi 8 bit
    sono i bit di informazione.
    Il primo bit di informazione viene campionato da START, gli altri da DATA.*/
    output reg load,
    output reg clear, /*clear=1 --> ripristina i valori di contatore dati ricevuti e registro interno dati ricevuti*/
    output reg en_d, //en_d=1 --> abilitazione conteggio dati ricevuti
    output reg done /*done=1 --> frame completamente ricevuto e dato valido sul registro di uscita*/
);

```

parameter IDLE=3'b000, START=3'b001 , DATA=3'b010, STOP\_1=3'b011, STOP\_2=3'b100;

reg [2:0] state, state\_nxt;

//logica sequenziale: registro di stato

```

always @ (posedge clk, posedge rst)
if (rst==1'b1) state<=IDLE;
else state<=state_nxt;

```

//logica combinatoria: determinazione dello stato futuro

```

always @ (state, hit_m, hit_d, RX)
case(state)
IDLE: /*uno 0 su RX in corrispondenza del MIDBIT --> trovato il bit di start viene avviata la ricezione del
frame*/
begin
    if (RX==1'b0 && hit_m==1'b1) state_nxt=START;
    else state_nxt=IDLE;
end

```

START: /\*in corrispondenza del MIDBIT --> inizia la ricezione e il caricamento dei bit del data chunk\*/

```

begin
    if (hit_m==1'b1) state_nxt=DATA;
    else state_nxt=START;
end

```

DATA: /\*hit\_d=1 in corrispondenza del MIDBIT --> caricamento del data chunk terminato\*/

```

begin
    if(hit_d==1'b1 && hit_m==1'b1) state_nxt=STOP_1;
    else state_nxt=DATA;
end

```

STOP\_1: /\*in corrispondenza del MIDBIT --> campionamento del primo dei bit di stop\*/

```

begin
    if(hit_m==1'b1) state_nxt=STOP_2;
    else state_nxt=STOP_1;
end

```

STOP\_2: /\*in corrispondenza del MIDBIT --> campionamento del secondo dei bit di stop\*/

```

begin
    if (hit_m==1'b1) state_nxt=IDLE;
    else state_nxt=STOP_2;
end

```

default: state\_nxt=IDLE;

endcase

//logica combinatoria: determinazione delle uscite

```

always @ (state)
case(state)
IDLE:
begin
    en_d=1'b0; //contatore dati ricevuti disabilitato
    done=1'b0; //dato non valido sul registro di uscita
    load=1'b0; /*sul MIDBIT troviamo start --> non va abilitata la scrittura del registro di uscita*/
    clear=1'b1; //contatore dati e registro di uscita resettati internamente
end

```

START:

```

begin
    en_d=1'b0; //contatore dati ricevuti disabilitato
    done=1'b0; //dato non valido sul registro di uscita
    load=1'b1; /*sul MIDBIT troviamo il primo bit di data chunk --> ya abilitata la scrittura del registro di
uscita*/
    clear=1'b0;
end

```

DATA:

```

begin
    en_d=1'b1; //contatore dati ricevuti abilitato
    done=1'b0; //dato non valido sul registro di uscita
    load=1'b1; /*sul MIDBIT troviamo il resto dei bit di data chunk --> ya abilitata la scrittura del registro di
uscita*/
    clear=1'b0;
end

```

STOP\_1:

```

begin
    en_d=1'b0; //contatore dati ricevuti disabilitato
    done=1'b0; //dato non valido sul registro di uscita
    load=1'b0; /*sul MIDBIT troviamo il primo bit di stop --> non va abilitata la scrittura del registro di
uscita*/
    clear=1'b0;
end

```

STOP\_2:

```

begin
    en_d=1'b0; //contatore dati ricevuti disabilitato
    done=1'b1; //dato valido sul registro di uscita
    load=1'b0; /*sul MIDBIT troviamo il secondo bit di stop --> non va abilitata la scrittura del registro di
uscita*/
    clear=1'b0;
end

```

default:

```

begin
    en_d=1'b0;
    done=1'b0;
    load=1'b0;
    clear=1'b1;
end

```

endcase

endmodule

Il top module, nel caso del canale di ricezione, è il seguente:

```

module top_rx #(parameter DATA_BW=8,          // N bit di informazione
               parameter DATA_BW_BIT=4,    // Config. CONTATORE bit info
               parameter BAUD_COUNT_x16=651, // CC per BR sovra-campionato (BR_X16)
               parameter BAUD_BIT_x16=10,   // Config. CONTATORE CC per BR_X16
               parameter OVER_SAMPL=16,     // Fattore di sovra-campionamento
               parameter OVER_SAMPL_BIT=5  // Config. CONTATORE OVER_SAMPL
               );
  input wire clk, rst, //segnali di controllo generali del sistema
  input wire RX,       //porta seriale di ricezione
  output wire RD,      //dato re-impacchettato valido
  output wire [DATA_BW-1:0] data_chunck //data_chunk ricevuto
);
wire baud_clk_16;
/*segnaile di identificazione del MIDBIT (hit_m=1) */
wire hit_m;

//gestione contatore dati ricevuti
/*Conteggio dati*/
wire [DATA_BW_BIT-1:0] count_data;
/*CONTATORE-->FSM segnala (hit_d=1) la trasmissione dell'ultimo bit di informazione*/
wire hit_d;
/*FSM-->CONTATORE abilita (en_d=1) il conteggio dei dati ricevuti*/
wire en_d;

//gestione del registro di output
/* segnale di abilitazione campionamento dato ricevuto:
ALTO in corrispondenza del MIDBIT (hit_m=1) dei bit di informazione (load=1) all'interno del frame*/
wire en_rx;
/*FSM-->REGISTRO DI USCITA segnala (load=1) la ricezione dei bit di informazione*/
wire load;

//altri segnali di controllo
/*FSM-->CONTATORE e FSM-->REGISTRO DI USCITA ripristino (clear=1) dei valori di default*/
wire clear;

//gestione sovra-campionamento
/*with BAUD_RATE 9600bps, oversampling 16 and system clk 100MHz --> baud rate counter x 16 is 651
represented with 10 bits*/
baud_rate_clk_gen #(.BAUD_CNT(BAUD_COUNT_x16),.BAUD_BIT(BAUD_BIT_x16)) BR_GEN(.clk(clk),
.rst(rst),.baud_clk(baud_clk_16));
/*mid_bit detector*/
mb_detector #(OVER_SAMPL(OVER_SAMPL),OVER_SAMPL_BIT(OVER_SAMPL_BIT))
MB_DET(.clk(baud_clk_16), .rst(rst),.hit_m(hit_m));

//gestione conteggio dati ricevuti
//ATTENZIONE: il contatore dei dati ricevuti su RX deve avanzare ogni MIDBIT
counter_clr #(.MAX(DATA_BW),.N_BIT(DATA_BW_BIT)) CNT_DATA(.clk(hit_m),
.rst(rst),.en(en_d),.clear(clear),.count(count_data));
assign hit_d = (count_data===(DATA_BW-1)) ? 1'b1: 1'b0;

//macchina a stati di controllo della funzionalità del Sistema
fsm_r FSM_RX(.clk(baud_clk_16),
.rst(rst),.RX(RX,.hit_d(hit_d),.hit_m(hit_m),.load(load),.en_d(en_d),.done(RD),.clear(clear)));

//registro di uscita
assign en_rx = hit_m & load;
data_buf #(.(BUF_SIZE(DATA_BW),BUF_SIZE_BIT(DATA_BW_BIT))
R_OUT(.clk(baud_clk_16), .rst(rst),.D(RX),.Id(en_rx),.clear(clear),.AD(count_data),.buffer(data_chunck));
endmodule

```

START & su DATA, dove vengono campionati i pezzi di data chunk, ma solo in corrispondenza del MIDBIT. È necessario quindi gestire bene il segnale di enable di questo registro incrociando il segnale di load e l'istante in cui viene rilevato il MIDBIT.

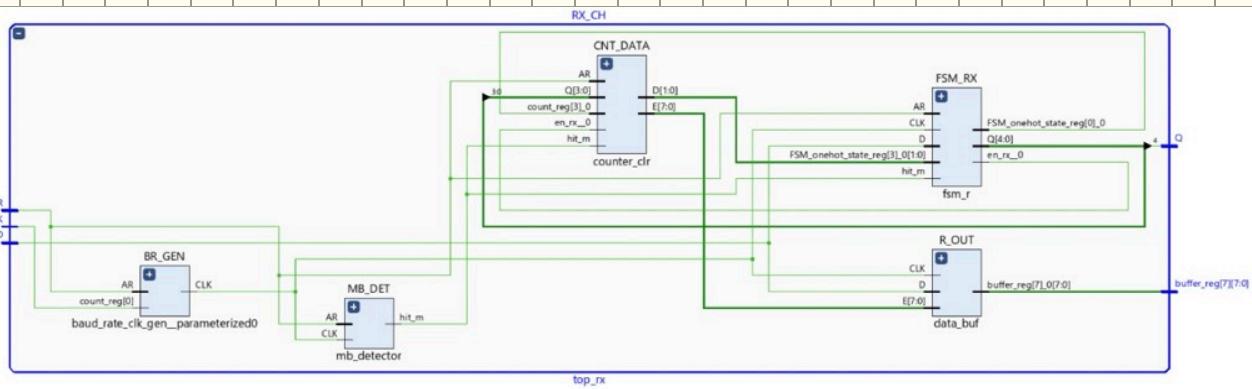


Figura 11 - UART Canale di Ricezione: Schema a Blocchi (Vivado Post-Sintesi).

Anche nel caso della ricezione si

dove ragionare bene sulle frequenze di funzionamento dei diversi blocchi.

Il contatore che scandisce i dati in ingresso

(contatore dati ricevuti su RX) deve avanzare alla velocità di trasmissione, il baud rate, e deve essere aggiornato su ogni MIDBIT. Il segnale di sincronizzazione di questo contatore è dato da hit\_m, generato dal modulo che identifica il MIDBIT (mid\_bit\_detector).

Questo segnale è usato come un clock, ma non ha la classica forma del clock, perché non ha un duty cycle al 50%.

La scrittura sul registro di uscita (registro di uscita) è pilotata dalla macchina a stati e può avvenire quando il controllo si trova su

## Sistema completo

```
module top_DUT #//parametri legati ai dati da trasmettere
    parameter DATA_BW=8, //dimensione dato da trasmettere
    parameter DATA_BW_BIT=4, /*ampiezza in bit contatore per i dati da trasmettere*/
    //parametri legati alla velocità di trasmissione
    parameter BAUD_RATE=9600, //baud rate: 9600bps
    parameter BAUD_COUNT=10416, /*contatore per generare enable baud rate: frequenza
(100MHz)/baud_rate*/
    parameter BAUD_BIT=14, /*ampiezza in bit contatore baud rate*/
    //parametri legati alla velocità di ricezione
    parameter BAUD_COUNT_x16=651, /*contatore per generare enable baud rate sovraccampionato:
frequenza (100MHz)/(baud_rate*over_samp)*/
    parameter BAUD_BIT_x16=10, /*ampiezza in bit contatore baud rate sovraccampionato*/
    //oversampling rate
    parameter OVER_SAMPL=16, //fattore di sovraccampionamento
    parameter OVER_SAMPL_BIT=5 /*ampiezza in bit per la definizione del sovraccampionamento*/
)
    input wire clk, rst, //segnali di controllo generali
    //interfaccia con il produttore: trasmissione
    input wire transmit, /*produttore-UART: dato valido (transmit=1)*/
    input wire [DATA_BW-1:0] data_in, /*produttore-UART: dato da trasmettere*/
    output wire busy, /*UART-produttore: trasmissione in corso (busy=1)*/
    //interfaccia con il produttore: ricezione
    output wire [DATA_BW-1:0] data_out, /*UART-produttore: dato ricevuto*/
    output wire ready /*UART-produttore: dato valido (ready=1)*/
);
wire TX;
top_tx
#(.DATA_BW(DATA_BW),.DATA_BW_BIT(DATA_BW_BIT),.BAUD_RATE(BAUD_RATE),.BAUD_COUNT
(BAUD_COUNT),.BAUD_BIT(BAUD_BIT))
    TX_CH(.clk(clk), .rst(rst), .transmit(transmit),.data(data_in),.TX(TX),.busy(busy));
top_rx
#(.DATA_BW(DATA_BW),.DATA_BW_BIT(DATA_BW_BIT),.BAUD_COUNT_x16(BAUD_COUNT_x16),
.BAUD_BIT_x16(BAUD_BIT_x16),
.OVER_SAMPL(OVER_SAMPL),.OVER_SAMPL_BIT(OVER_SAMPL_BIT))
    RX_CH(.clk(clk), .rst(rst), .RX(TX), .RD(ready),.data_chunck(data_out));
endmodule
```

## Testbench

```
module tb_tx;
parameter DATA_BW=8, DATA_BW_BIT=4,
BAUD_RATE=9600,
BAUD_COUNT=10416, /*BAUD_RATE 9600bps and system clk 100MHz --> baud rate counter is 10416
represented with N_BIT=14*/
BAUD_BIT=14,
OVER_SAMPL=16,
OVER_SAMPL_BIT=5,
BAUD_COUNT_x16=651, /*BAUD_RATE 9600bps, oversampling 16 and system clk 100MHz --> baud
rate counter x 16 is 651 represented with 10 bits*/
BAUD_BIT_x16=10;

reg clk, rst;
reg transmit;
reg [DATA_BW-1:0] input_data;
wire [DATA_BW-1:0] data_out;
wire busy, ready;

top_DUT
#(.DATA_BW(DATA_BW),.DATA_BW_BIT(DATA_BW_BIT),.BAUD_RATE(BAUD_RATE),.BAUD_COUNT
(BAUD_COUNT),.BAUD_BIT(BAUD_BIT),.BAUD_COUNT_x16(BAUD_COUNT_x16),
.BAUD_BIT_x16(BAUD_BIT_x16),.OVER_SAMPL(OVER_SAMPL),.OVER_SAMPL_BIT(OVER_SAMPL_BT
T))
DUT(.clk(clk), .rst(rst), .transmit(transmit),.data_in(input_data),.busy(busy),.data_out(data_out), .ready(ready));

//generazione clock simulazione
always #5 clk=~clk;

//dinamica segnali
initial
begin
clk=1'b0;
rst=1'b0;
transmit=1'b0;
input_data=8'b000010101;
#100
rst=1'b1;
#100
rst=1'b0;
#100
SendSingleByte(8'hC1);
SendSingleByte(8'hBE);
SendSingleByte(8'hEF);
#1000
$stop;
end

//simulazione produttore
```

```

task SendSingleByte;
  input [7:0] data; //dato da trasmettere //ARGOMENTO
begin
  @(negedge clk)
    //by trasmesso alla UART sollevando il transmit contestualmente
    input_data <= data; //Il task assegna transmit e
    transmit <= 1'b1; //input_data
    @(negedge clk) //transmit e input_data sono sincronizzati grazie a questo negedge
    transmit <= 1'b0; //transmit alto un colpo di clock
    /*non si esce dal task (quindi non si spediscono nuove transazioni) fino a che la UART non è di nuovo
disponibile/
    @(negedge busy); //per terminare si attende negedge bus
end
endtask

```

endmodule

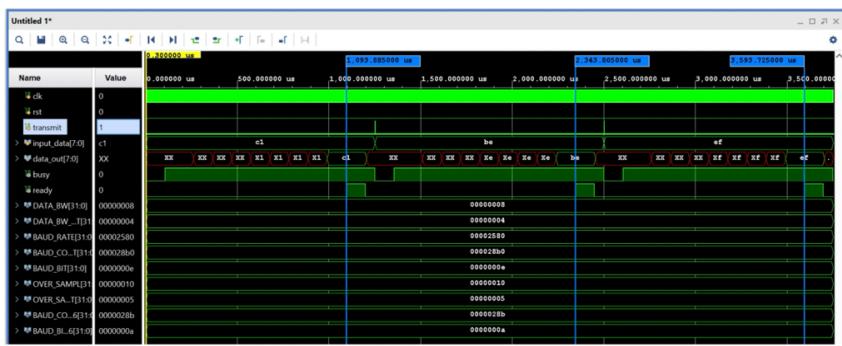


Figura 12 - UART sistema completo: simulazione pre-sintesi.

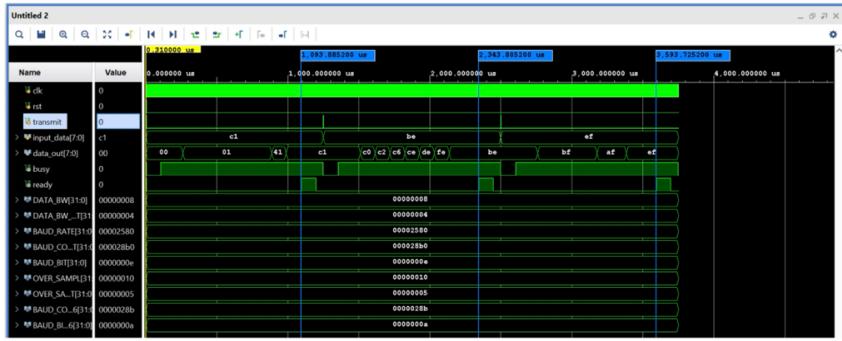


Figura 13 - UART sistema completo: simulazione post-sintesi funzionale

Le due simulazioni danno risultati analoghi, tutti i dati trasmessi arrivano al consumatore con la stessa temporizzazione.

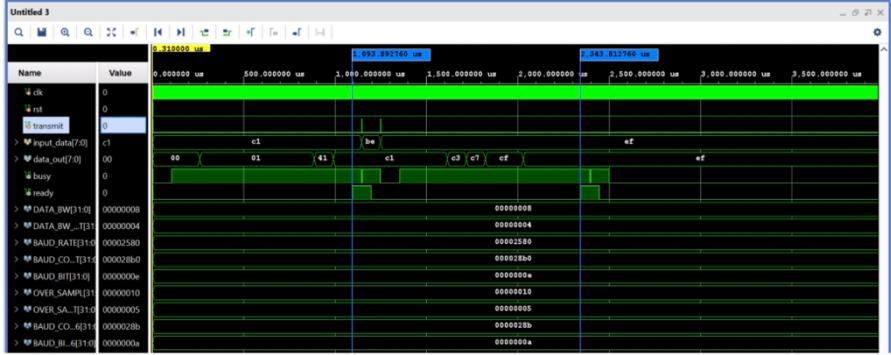


Figura 14 - UART sistema completo: simulazione post-sintesi timing --> GLITCH!

transmit è un input generato dal produttore. Gli altri due sono output, rispettivamente, del canale di trasmissione e del canale di ricezione. Nel segnale di busy sono presenti dei glitch. Questi glitch sono inattesi.

Il Task è un costrutto NON SINTETIZZABILE quindi può essere utilizzato solo nei testbench.  
È simile al concetto di procedura in un qualsiasi linguaggio di programmazione.

Il task:

- può avere argomenti
- può restituire nessun valore, ma può definire il valore dei segnali reg che determineranno gli ingressi del design under test
- può impiegare statement di controllo temporale

In questa figura è mostrato il risultato della simulazione timing post-sintesi. La forma d'onda non sono corrette. Vi è una differenza nel comportamento dei segnali transmit, busy e ready. In questo caso