**Progetto di reti Logiche**

Interfaccia seriale asincrona UART

J. Singh, C. C. Rambaldi Migliore

# Introduzione

Il progetto consiste nella realizzazione di un trasmettitore e ricevitore UART dotati di segnalazione hardware del controllo di flusso (RTS e CTS).

L’interfaccia asincrona UART permette la comunicazione tra dispositivi full duplex, avendo una linea per la trasmissione e una per la ricezione.

La trasmissione UART si basa su 4 linee digitali: TX (Trasmissione), RX (Ricezione), RTS (Request to send) e CTS (Clear to send). La linea TX del dispositivo trasmittente è collegata verso RX del dispositivo ricevente e trasporta i bit in maniera seriale. La linea RTS del dispositivo ricevente è collegata verso CTS del dispositivo trasmittente e assume valore logico 1 quando il dispositivo ricevitore è pronto e in grado di ricevere dati; questi segnali non sono intesi per interrompere la trasmissione a metà, difatti sono sincronizzati con la fine della trasmissione per interrompere il flusso correttamente. RTS e CTS implementano l’hardware flow control per permettere all’utilizzatore del ricevitore di interrompere eventuali trasmissioni successive.

La linea di trasmissione assume normalmente valore 1. Quando il trasmettitore intende iniziare un trasferimento dati abbassa la linea a 0 (start bit) quindi trasmette 8 bit (7 bit di dati e 1 bit di parità), seguiti da un ulteriore 1 (stop bit). Nel caso in cui il trasmettitore non trasmetta lo stop bit, verrà segnalato un errore dal ricevitore. Il bit di parità viene calcolato durante la trasmissione e trasmesso alla fine.

Vengono mandati 7 bit di dati, 1 di parità, 1 start bit e 1 stop bit. Quindi una trasmissione dura il tempo di 10 bit.

Il bit di parità pari assume valore 0 quando il numero di uni è pari, 1 viceversa, il ricevitore controllerà la coerenza e nel caso segnalerà un errore.

Il trasmettitore e il ricevitore operano allo stesso baudrate ma in maniera asincrona, a questo proposito c’è bisogno di introdurre un “clock veloce” a frequenza multipla del baudrate (16 volte) che verrà utilizzato dal ricevitore per campionare in maniera corretta il bit in arrivo dal trasmettitore.

Il clock veloce viene utilizzato in questo modo: quando il ricevitore si accorge che è stato trasmetto il primo bit a 0, comincia a contare fino a 8, campiona il bit e da lì in poi continuerà a contare fino a 16 per poi campionare il bit dopo; in questo modo si posizionerà circa al centro del bit e sarà sicuro di campionare sempre il bit corretto (se non si utilizzasse questo metodo, potrebbe succedere che uno o più bit vengano persi o venga campionato più volte lo stesso bit).



## Assunzioni

RTS e CTS vengono utilizzati come controllo di flusso per bloccare la prossima trasmissione, non quella corrente. Una volta che RTS e CTS tornano a livello alto la trasmissione riprende a funzionare. Si richiede all’utilizzatore del receiver di segnalare la propria non disponibilità a ricevere i dati nella prossima trasmissione, almeno 2 baudrate prima della fine della trasmissione corrente.

Il “campionamento nel mezzo” del ricevitore avviene mediante un clock di frequenza baudrate\*16.

Il bit di parità è sempre presente ed è di parità pari.

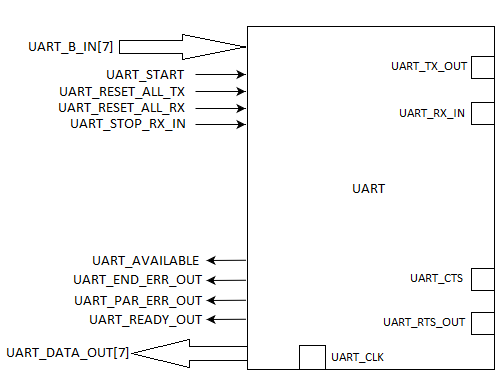
# Specifica

La UART si comporrà di due principali moduli: trasmettitore e ricevitore.

**Trasmettitore**: si occupa di serializzare i bit che arrivano in input su “UART\_B\_IN[7]”, aggiunge lo start bit all’inizio e il parity bit alla fine, concludendo la trasmissione con la linea alta (livello logico 1). La trasmissione avviene non appena il segnale in input “UART\_START” diventa alto, sempre che l’input “UART\_CTS” sia anch’esso a livello logico alto. La frequenza a cui vengono trasmessi i bit è pari al baudrate scelto. Si comporrà a sua volta dei seguenti moduli: “Start Management”, “Parity Bit Generator”, “PISO”, “Bit Counter”, “Tick Counter”.

**Ricevitore**: Riceve in input su “UART\_RX\_IN” i bit serializzati. “UART\_RX\_IN” a “riposo” è sempre a livello alto, non appena viene visto un livello basso, significa che la trasmissione è iniziata. Da quando inizia la trasmissione, il ricevitore deve cominciare a contare i colpi di clock veloce fino a 8 per arrivare nel mezzo della trasmissione del bit, campionarlo e contare da quel momento in poi 16 colpi prima di campionare un nuovo bit. Andrà poi a parallelizzare in output i bit ricevuti “togliendo” lo start bit, il parity bit e l'end bit. Quando il ricevitore è sicuro di aver ricevuto tutti i bit e averli correttamente parallelizzati, viene messo un bit di "ready" alto per 4 colpi di clock veloce durante i quali l’utilizzatore potrà prelevare i dati e sapere se si sono riscontrati degli errori. Si comporrà a sua volta dei seguenti moduli: “Parity Bit Generator”, SIPO”, “Bit Counter”, “Tick Counter”.

## Interfaccia del sistema



### Segnali:

* UART\_B\_IN[7] **input - 7 bit**: i 7 bit paralleli che verranno poi serializzati dal trasmettitore per permetterne il trasporto in seriale verso il ricevitore;
* UART\_START **input - 1 bit - default 0**: assume valore logico 1 per almeno 1 periodo di baudrate quando l’utilizzatore intende far partire la trasmissione dei dati;
* UART\_RX\_IN **input - 1 bit - default 1**: ingresso seriale del ricevitore;
* UART\_STOP\_RX\_IN **input - 1 bit - default 0**: segnale di controllo dell’utilizzatore del ricevitore della UART per fermare la ricezione dei dati (per esempio nel caso in cui l’utilizzatore del ricevitore abbia il buffer pieno);
* UART\_CLK **input - 1 bit**: clock “veloce” della UART;
* UART\_CTS **input - 1 bit**: Clear To Send (controllo di flusso del trasmettitore);
* UART\_RESET\_ALL\_TX **input - 1 bit - default 0**: reset del trasmettitore;
* UART\_RESET\_ALL\_RX **input - 1 bit - default** **0**: reset del ricevitore;
* UART\_AVAILABLE **output - 1 bit**: segnale di controllo per l’utilizzatore del trasmettitore per capire se la trasmissione dei dati è disponibile o meno (a seconda che il ricevitore sia pronto o meno);
* UART\_DATA\_OUT[7] **output - 7 bit**: i 7 bit parallelizzati dal ricevitore una volta che tutti i 7 bit seriali sono stati ricevuti correttamente;
* UART\_READY\_OUT **output - 1 bit**: assume valore logico 1 quando i dati parallelizzati sono pronti per essere presi dall’utilizzatore;
* UART\_RTS\_OUT **output - 1 bit - default 1**: Request To Send (controllo di flusso del ricevitore);
* UART\_END\_ERROR\_OUT **output - 1 bit**: errore sullo stop bit;
* UART\_PAR\_ERR\_OUT **output - 1 bit**: errore sul bit di parità;
* UART\_TX\_OUT **output - 1 bit - default 1**: l’uscita seriale del trasmettitore;

### Relazioni tra segnali:

* Una volta caricati i dati in UART\_B\_IN, viene richiesta l’attivazione del segnale di controllo start. I bit vengono caricati nel PISO al primo fronte di clock dopo che start è stato attivato. Lo start e i bit in ingresso al trasmettitore rimangono per il tempo necessario al caricamento nel PISO, cioè almeno 1 baudrate.
* Per una gestione migliore della trasmissione si è implementato una gestione di flusso attraverso i segnali UART\_RTS\_OUT e UART\_CTS (collegati tra loro su due dispositivi diversi). UART\_RTS\_OUT viene controllato dall’utilizzatore della UART che riceve i dati. Quando l’utilizzatore del ricevitore imposta UART\_STOP\_RX\_IN=1 allora UART\_RTS\_OUT diventa 0, essendo quest’ultimo segnale collegato a UART\_CTS, fa sì che il trasmettitore non cominci nuove comunicazioni. Invece UART\_STOP\_RX\_IN=0 permette al trasmettitore di inviare i dati al ricevitore.
* UART\_AVAILABLE è un segnale che permette all’utilizzatore del trasmettitore di sapere se l’utilizzatore del ricevitore è disponibile alla ricezione dei dati e di conseguenza, di capire se il trasmettitore invierà o meno i dati inseriti.

# Architettura del sistema

## Tick counter

### Segnali

* clock: input – 1bit
* preset: input – 1bit
* enable: input – 1bit
* tick: output – 1bit
* m\_tick: output – 1bit
* qtr\_1\_tick: output – 1bit
* qtr\_3\_tick: output – 1bit

### Comportamento

Si tratta di un contatore johnson modulo N, utilizzato come divisore di clock. Il conteggio avviene ad ogni colpo di **clock** se e solo se **enable** è a 1, il segnale di **preset** (asincrono)mette a 1 l’ultimo registro e a 0 tutti gli altri (così che al prossimo colpo di clock, il contatore conti 1). **Tick** è l’uscita del primo registro, **m\_tick** è l’uscita registro di mezzo, **qtr\_1\_tick** è l’uscita del primo quarto, **qtr\_3\_tick** è l’uscita del terzo quarto. N deve essere un multiplo di 4.

## Bit counter

### Segnali

* tick: input – 1bit
* reset: input – 1bit
* enable: input – 1bit
* count: output – Nbit

### Comportamento

È un contatore a N bit. Viene usato per contare quanti bit sono arrivati al ricevitore. Il **tick** è il clock generale (veloce) diviso per 16. Ad ogni **tick**, viene contato un bit se e solo se **enable** è a 1. Il segnale di **reset** (asincrono)resetta a 0 il conto. **Count** sono gli N bit contati.

## PISO (Parallel Input Serial Output)

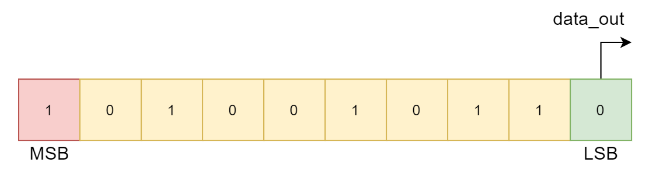
### Segnali

* CLK: input – 1bit
* PRESET: input – 1bit
* ENABLE: input – 1bit
* LOAD: input – 1bit
* XP: input – 8bit
* Y: output – 1bit

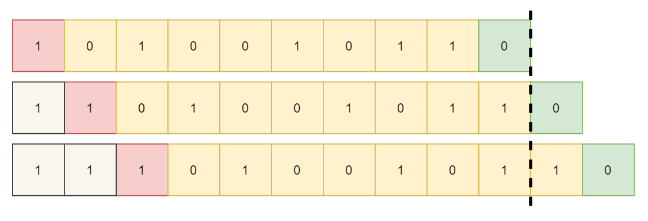
### Comportamento

È uno shifter che converte dati da parallelo a seriale. Quando **LOAD** è a 1, viene caricato **B\_IN** nei registri, mentre quando è a 0, i bit vengono fatti scorrere nei registri. Il **PRESET** (asincrono) imposta a 1tutti i bit nel bistabile. Lo shifter funziona ogni colpo di **tick** sse **enable** è a 1.

Quando LOAD=1L



QUANDO LOAD=0L sono in shifting



## SIPO (Serial Input Parallel Output)

### Segnali

* tick: input – 1bit
* rx: input – 1bit
* enable: input – 1bit
* reset: input – 1bit
* B\_OUT : output – Nbit

### Comportamento

Converte da seriale a parallelo. **Rx** entra in uno shifter e dopo N colpi di clock ho i bit paralleli pronti su **B\_OUT**. Lo shifter viene resettato a 0 quando **reset** (asincrono)è alto. Lo shifter è sensibile ai colpi di clock solo quando **enable** è a 1.

## START\_MANAGEMENT

### Segnali

* CLOCK\_MNG0: input-1 bit
* CLOCK\_MNG1: input-1 bit
* CTS\_MNG: input-1-bit
* START\_MNG: input-1-bit
* RESET\_START: input-1-bit
* RST\_MNG: input-1-bit
* FF\_OUT\_START: output-1-bit
* LOAD\_CONTROL: output-1-bit

### Comportamento

Gestisce la fase di avvio di una trasmissione, si è preferito creare un componente per questa fase per rendere più leggibile il codice del trasmettitore.

Ci sono 2 D\_FF (Flip Flop di tipo D) in serie, il primo D\_FF è sincronizzato al clock veloce il secondo al baudrate.

Appena **START\_MNG**=1 (se **CTS\_MNG**=1) al successivo colpo di clock veloce la macchina si troverà nello stato ‘10’ quindi porrà **LOAD\_CONTROL**=1 e **FF\_OUT\_START**=1 (utilizzato per enable degli altri componenti del trasmettitore). Successivamente si esce dalla fase di loading ponendo **LOAD\_CONTROL**=0 e quindi il PISO entrerà in modalità shifting. Se fosse stato **CTS\_MNG**=0 non mi sarei portato in stato di loading ‘10’. **RST\_MNG** dopo aver trasmesso 10 bit resetta il secondo D\_FF; invece, **RESET\_START** resetta il primo D\_FF nel caso in cui la trasmissione non sia continua, dopo aver trasmesso 10 bit di informazione.

## Parity Bit Generator

### Segnali

* d\_in: input – 1bit
* tick: input – 1bit
* reset: input – 1bit
* preset: input – 1bit
* enable: input – 1bit
* even\_p: output – 1bit

### Comportamento

Si tratta di un flip flop di tipo T. Se **enable** è alto, ad ogni **tick**,ogni volta che **d\_in** vale 1 **even\_p** cambia di valore. Questo processo permette di capire se il numero di uni che arrivano in input sono dispari (**even\_p** a 1) o pari (**even\_p** a 0). **Reset** imposta **even\_p** a 0, **preset** imposta **even\_p** a 1 (entrambi asincroni).

## Receiver

### Segnali

* RESET\_RECEIV: input – 1bit
* CLOCK: input – 1bit
* RX\_IN: input – 1bit
* STOP\_RX\_IN: input – 1bit
* DATA\_OUT: output – 7bit
* RTS\_OUT: output – 1bit
* END\_ERR\_OUT: output – 1bit
* PAR\_ERR\_OUT: output – 1bit
* READY\_OUT: output – 1bit

### Comportamento

Questo modulo utilizza i sotto moduli “**Tick Counter**”, “**SIPO**”, “**Parity Bit Generator**”, “**Bit Counter**”.

Prima di utilizzare il componente, bisogna resettarlo tramite il segnale di input **RESET\_RECEIV**, da qui in poi il componente è pronto per essere utilizzato.

Il **CLOCK** viene mandato in ingresso al “tick counter” che mi fornirà in uscita il tick nel mezzo della trasmissione del bit, alla frequenza di baudrate. Normalmente **RX\_IN** è a 1 e il componente è fermo, non computa. Quando **RX\_IN** va a 0, significa che il trasmettitore ha iniziato la trasmissione. Questo evento toglie i reset ai sotto moduli e mette i loro enable alti, da qui in poi il “SIPO” shifterà i bit. Quando il “Bit Counter” arriva a contare 10 bit ricevuti, durante i successivi 4 colpi di **CLOCK:**

* **READY\_OUT** diventa alto
* **DATA\_OUT** contiene i bit trasmessi parallelizzati
* **END\_ERR\_OUT** sarà alto se stop\_bit è uguale a 0
* **PAR\_ERR\_OUT** sarà alto se il parity bit trasmesso è diverso da quello generato

Esauriti i 4 colpi di **CLOCK**, i moduli vengono resettati in attesa della prossima trasmissione.

**RTS\_OUT** è il segnale **STOP\_RX\_IN** negato; quando l’utilizzatore del ricevitore ha bisogno di fermare la trasmissione (es: buffer pieno), dovrà mettere **STOP\_RX\_IN** alto. È meglio mantenere un certo anticipo, ad esempio si consiglia di fermare la trasmissione quando il buffer ha ancora 2/3 “posti” liberi.

## Transmitter

### Segnali

* B\_IN: input – 7bit
* RESET\_ALL\_TX: input--1bit
* CLK: input – 1bit
* START: input – 1bit
* CTS: input – 1bit
* AVAILABLE: output-1bit
* TX\_OUT: output– 1bit

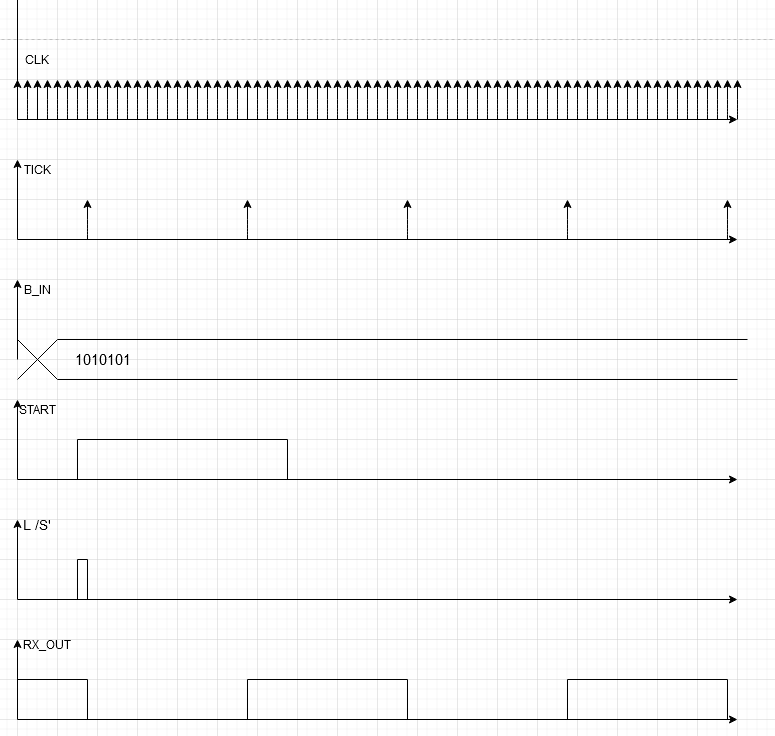
### Comportamento:

Questo modulo utilizza i sotto moduli “Tick Counter”, “PISO”, “Parity Bit Generator”, “Bit Counter”, ”START\_MANAGEMENT”. La linea di trasmissione mantiene livello 1 logico, quando si vuole iniziare a trasmettere viene mandato uno start bit a livello logico 0 su **TX\_OUT**. Quando l’utilizzatore vuole iniziare la trasmissione, invia in ingresso i bit paralleli su **B\_IN** e mette alto **START**. Il segnale di START viene memorizzato in un registro perché lo si immagina come un possibile segnale temporaneo. Da quando arriva **START**, viene effettualo il load di **B\_IN** nel “PISO” e successivamente viene iniziata la trasmissione alla frequenza di baudrate. Il **CLK** viene mandato in ingresso al “tick counter” che mi fornirà in uscita il tick alla frequenza di baudrate. Il bit di partita viene calcolato dal componente “parity bit generator” e trasmesso come “b7” (scelta effettuata grazie ad un mux).

Fintanto che il **CTS** è alto i dati vengono mandati, nel caso il **CTS** assuma valore logico 0, la trasmissione corrente continua fino al normale termine, poi non vengono più iniziate nuove trasmissioni. Si utilizza un contatore per contare i bit mandati perché una volta trasmessi tutti i bit dobbiamo resettare lo stato del transmitter per cominciare eventualmente una nuova comunicazione.

Di seguito un esempio di funzionamento: ipotizziamo che sia **AVAILABLE** e quindi che il receiver abbia RTS=1.

L’utente inserisce i dati nel transmitter e alza il segnale di START, successivamente viene effettuato il caricamento dei dati nel PISO. Non è richiesto il segnale START alto tutto il tempo, è infatti sufficiente mantenerlo per 1 ciclo di baudrate. Il primo bit mandato dal transmitter è lo start bit.



Si osserva che prima bisogna inserire i bit poi successivamente inserire **START**, per un corretto funzionamento. **START** non è richiesto per tutto il tempo della trasmissione, ma almeno per 1 periodo del baudrate. Nel caso l’utente voglia effettuare una trasmissione continua, deve inserire i dati prima del termine della trasmissione e continuare a tenere **START** alto.

Per quanto riguarda la gestione del flusso tramite CTS, è richiesta una notifica di “interruzione prossima trasmissione” almeno 2 baudrate prima della fine della trasmissione corrente.

# Verifica

Il componente è stato verificato sia tramite una simulazione behavioral, che tramite una simulazione post-route.

È stato scelto un clock minimo di 50ns, valore deciso rispetto al massimo valore di baudrate della UART cioè 921600 (che moltiplicato per 16 fa circa 15 MHz e quindi circa 67ns) e confermato tramite simulazione post-route.

## Immagine che contiene testo Descrizione generata automaticamenteImmagine che contiene testo Descrizione generata automaticamenteTest-bench

UART\_TB

TRANSMITTER\_TB --->

TRANSMITTER\_TB

TRANSMITTER\_TB --->

Immagine che contiene testo, monitor, interni, schermo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

<-- RECEIVER\_TB

## Casi d’uso

* Usare solo il receiver, inviargli una stringa di bit con end bit a 0 e verificare che il segnale di errore sia alto alla fine della trasmissione.
* Usare solo il receiver, inviargli una stringa di bit con parity bit non corretto e verificare che il segnale di parity error sia alto alla fine della trasmissione.
* Usare solo transmitter e verificare che dopo aver schiacciato start il primo bit sia a 0 e l’ultimo bit (end bit) sia a 1 e tutti i bit siano trasmessi correttamente.
* Usare il transmitter e receiver con il controllo di flusso (CTS e RTS) e verificare il funzionamento corretto.
* Usare transmitter e receiver, controllare che i bit in input al transmitter siano gli stessi in output al receiver e i segnali di errore siano bassi.