**PROGETTO RETI LOGICHE**

***Interfaccia Seriale UART***

di Flavio Renzi e Federico Viola

**SOMMARIO**

**Introduzione3**

**Specifica4**

Architettura del sistema6

Serializzatore7

Parallelo-Serie8

Rising Edge Detector8

Deserializzatore9

Clock Delayer 410

Clock Divider 810

Reception Counter11

Baud Generator12

Clock Generator12

Clock Divider 812

Verifica13

Test Bench13

Casi d’uso13

INTRODUZIONE

Il nostro progetto ha l’obiettivo di studiare l’interfaccia seriale asincrona UART, in particolare gli aspetti legati alla sua composizione, alla costruzione, al suo utilizzo e funzionamento.

Attraverso la realizzazione di un unico componente, che può svolgere sia il ruolo di trasmettitore che di ricevitore, sperimenteremo entrambe le sue proprietà effettuando dei test su due dispositivi identici.

Ogni dispositivo ha una porta di ingresso e una porta di uscita dei dati lungo una certa direzione.

Istanziato, quindi, lo stesso componente due volte testeremo la sua capacità di trasmissione e ricezione prendendo un dato dal bus d’ ingresso dell’interfaccia di uno e trasmettendolo all’altro, che lo riceverà e lo porterà sul proprio bus d’uscita.

Se identifichiamo un device con la lettera A e l’altro con la lettera B, può avvenire contemporaneamente la trasmissione/ricezione di un dato in direzione A🡪B e la trasmissione/ricezione di un altro dato in direzione B🡪A.

SPECIFICA

*INTERFACCIA DEL SISTEMA*

L’interfaccia UART (Universal Asinchronous Receiver Transmitter) è un sistema che permette la comunicazione tra dispositivi, enormemente utilizzato ancora oggi negli apparati industriali.

Per comunicare sono necessari due dispositivi UART, uno utilizzato come trasmettitore dei dati e l’altro come ricevitore. Questo è possibile grazie al fatto che l’interfaccia è Full-Duplex: esiste una porta Tx da cui parte una linea per la trasmissione e una porta Rx collegata a una linea per la ricezione dei dati (ovvero una linea di trasmissione di un’altra interfaccia UART associata). Lo scambio dei dati, quindi, è bidirezionale e può avvenire contemporaneamente.

Il meccanismo UART, nello specifico, funziona in questo modo:

* Per il ruolo di trasmettitore, l’interfaccia presenta una porta a cui è collegato un Data-Bus di 8 bit. I bit arriveranno in parallelo a un serializzatore che avrà il compito di permettere la trasmissione di un bit alla volta attraverso la linea, uscente dalla porta Tx. I bit sono inviati in ordine partendo dal MSB (Most Significant Bit) fino al LSB (Least Significant Bit), oppure viceversa. In questo senso si intende che la comunicazione è seriale.
* Dal punto di vista della ricezione, la nostra interfaccia rileva i dati trasmessi da un altro dispositivo sulla porta Rx e attraverso un deserializzatore, li porterà in parallelo su un Data-Bus collegato a una porta di uscita.

Abbiamo definito 4 porte fondamentali sull’interfaccia. A queste se ne aggiungono altre. Sono necessari:

* Un segnale **START** (1 bit) in ingresso per permettere la trasmissione.
* Un segnale **READY** (1 bit) in uscita per segnalare che è stato ricevuto correttamente un byte.
* I segnali **RTS** (Request-To-Send) in uscita e CTS (Clear-To-Send) in entrata. RTS è collegato al **CTS** di un altro device e, dualmemte, CTS a RTS. Entrambi sono di 1 bit e implementano l’Hardware Flow Control, un meccanismo che permette al trasmettitore e al ricevitore di notificare reciprocamente il proprio stato, in modo da garantire uno scambio affidabile dei dati.
* Un segnale di **Clock** (1 bit) in ingresso di 7,3MHz.
* Un segnale di **Reset** (1bit) in ingresso.
* Un segnale **Frq** per indicare la frequenza di funzionamento del dispositivo (baudrate o bit/s). Tipicamente le frequenze sono otto (9600, 19200, 38400, 57600, 115200, 230400, 460800 e 921600) per cui questo il segnale in entrata sarà composto da 3 bit.
* Un segnale di **BufferFull** (1 bit) gestito dal livello applicativo che segnala se i moduli sono pronti a ricevere un nuovo dato.

Inizialmente la linea di trasmissione ha valore logico 1, i bit di START e READY sono a 0 e così anche i segnali RTS e CTS. Quando si verifica l’evento START = 1 è possibile trasferire i dati se anche il ricevitore solleva RTS = 1. Questo segnale corrisponde ad una richiesta di trasferimento dati (lato ricevitore) e quindi abilita la trasmissione (essendo collegato alla porta CTS – lato trasmettitore). Il sistema deve poter interrompere o eseguire la trasmissione o la ricezione a seconda del valore di questi segnali.

A questo punto il trasmettitore abbassa la linea di trasmissione a 0 e aspetta un tempo di bit (il tempo di bit è l’inverso della frequenza di funzionamento fissata). Questa operazione equivale a inoltrare uno start bit, ovvero informare il ricevitore che sta per iniziare la trasmissione dei dati vera e propria.

Vengono poi trasmessi 8 bit, seguiti da un bit di stop a valore logico 1.

La trasmissione avviene sempre per blocchi di 8 bit che, a seconda delle impostazioni possono essere 7 bit di dati più un bit di parità oppure 8 bit di dati, senza parità. In totale, con bit di start e stop, sono trasmessi 10 bit.

Sullo start bit il ricevitore si mette in ascolto, ma in che modo?

Poichè la trasmissione è asincrona, il trasmettitore può inviare dati in qualsiasi momento. I fronti di transizione dei bit non sono allineati ad un clock condiviso tra trasmettitore e ricevitore ma avvengono ad una frequenza specifica tra le otto selezionabili. I due dispositivi conoscono il baudrate con cui possono comunicare, il problema è che il segnale di clock di uno e il segnale di clock dell’altro sono sfasati.

Si possono generare degli errori di trasmissione legati a questo sfasamento.

In certe applicazioni le linee elettriche su cui il segnale viaggia sono spesso molto lunghe e gli ambienti possono essere disturbati. Una conseguenza di ciò è il fatto che le transizioni dei bit non sono perfettamente verticali ma delle onde, non si verificano dei fronti verticali netti e puliti.

L’ideale per il ricevitore sarebbe campionare il segnale d’ingresso a metà del tempo di bit.

Il metodo migliore per soddisfare tale richiesta è quello del sovracampionamento: il clock del ricevitore viene impostato a una velocità che è multipla di quella del clock di trasmissione (solitamente 8 o 16 volte più grande). Grazie a ciò è possibile identificare in maniera abbastanza approssimata il momento del fronte di discesa relativo allo start bit, quando il trasmettitore abbassa la linea da 0 a 1.

Da qui, vengono contati 4 cicli, il clock del ricevitore si troverà sincronizzato al centro della trasmissione dello start bit. Questo è il punto di partenza. Ogni 8 clock il ricevitore sarà in grado, così, di campionare i bit al centro del loro tempo di trasmissione, con una buona approssimazione. Dopo 8 bit di dati campionati il ricevitore si aspetterà uno stop bit.

Se i buffer del livello applicativo sono quasi pieni e non possono più leggere un dato in uscita dal ricevitore UART, viene sollevato il segnale BufferFull, inizialmente a valore logico 0.

Se BufferFull=0 il byte può essere trasferito e verrà lanciato l’evento READY = 1 per segnalare che i dati sono stati correttamente inseriti nel bus di uscita.

A questo punto i dispositivi tornano alla propria configurazione iniziale.

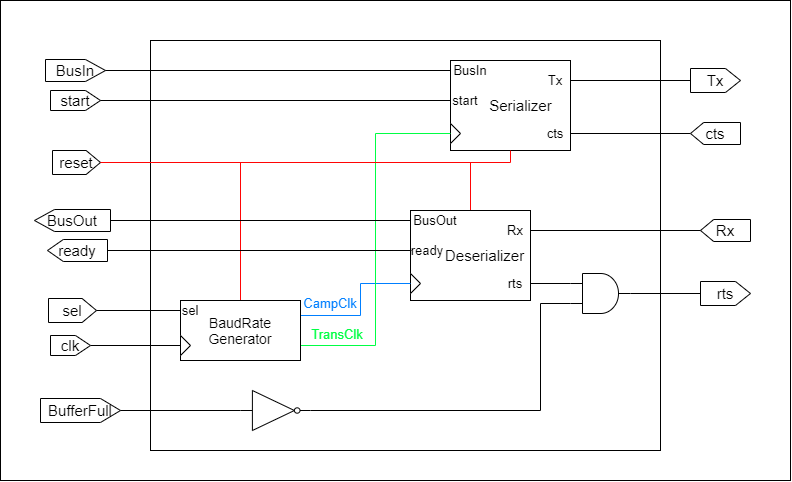
ARCHITETTURA DEL SISTEMA

Come abbiamo già osservato, la struttura di un’interfaccia UART è formata da una serie di elementi che hanno una funzione specifica e svolgono determinati compiti relativi alla gestione dei dati e dei segnali, ai fini di una corretta trasmissione delle informazioni.

E’ chiaro inoltre che sia indispensabile utilizzare due dispositivi identici per instaurare una comunicazione, un flusso di dati in entrambe le direzioni.

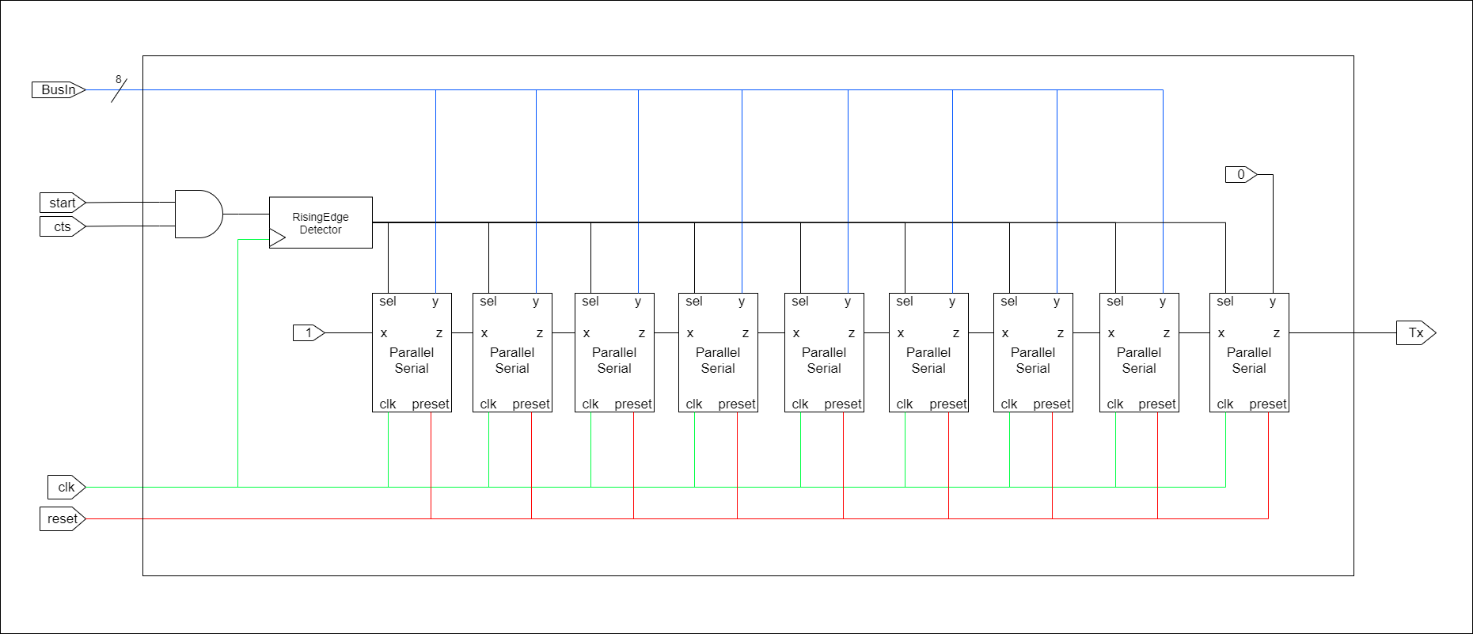
Vediamo nello specifico come è realizzato un dispositivo UART.

Gli elementi principali che costituiscono il nostro oggetto di studio sono un serializzatore, un deserializzatore e un generatore di frequenza del clock.



**Rappresentazione grafica dell’architettura dell’interfaccia UART attraverso i macroblocchi principali**

*SERIALIZZATORE*



**Rappresentazione grafica dell’architettura del serializzatore attraverso i sottomdouli**

La struttura principale di un serializzatore si basa su un registro parallelo-serie. Questo registro presenta un ingresso parallelo su 9 bit e un’uscita seriale **z**.

Quando **sel** vale 1, ogni bistabile memorizza il valore presente sull’ingresso dati **y** svolgendo di fatto la funzione di un registro parallelo. Non appena il segnale **sel** assume valore 0, gli ingressi dati risultano esclusi grazie ai multiplexer che, in questa condizione, contribuiscono alla formazione di una linea seriale che attraversa tutti i bistabili come nel caso di un registro a scorrimento. Il dato **x** in ingresso più a sinistra è fisso e uguale a 1.

Questo perché quando il dispositivo dovrà trasmettere il dato, esso verrà trasmesso in modo seriale impostando la modalità del registro come “scorrimento” (**sel**=0) e il valore 1 si propagherà fino a raggiungere l’uscita finale, rappresentando il bit di STOP della trasmissione e nei successivi clock mantenendo la linea logica di trasmissione a livello alto.

L’ultimo valore di **y**, più a destra, è fisso e vale 0. Questo valore invece sarà il primo a essere trasmesso e rappresenta il bit di **START** della trasmissione.

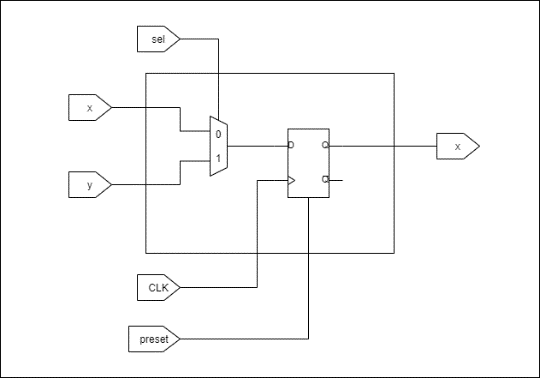
Il valore di **sel** è rappresentato dall’uscita del Rising Edge Detector che ha come ingresso il risultato della porta AND dei segnali di **START** e **CTS**. Quando entrambi saranno a valore 1, viene rilevato il fronte di salita dal Rising Edge Detector il quale porta in uscita il valore 1 fino alle fine del ciclo di clock negato corrente per poi riportare di nuovo uno 0 fino al prossimo evento di inizio di una nuova trasmissione.

Il segnale di **RESET** coincide con il preset di ciascun registro. Esso serve a impostare a 1 i valori iniziali salvati nei registri (**RESET** = 1).

*PARALLELO-SERIE*

Ogni singolo registro con il relativo multiplexer costituisce un sottomodulo riutilizzabile.

Grazie al segnale **sel** è possibile selezionare come ingresso di ogni bistabile, di tipo D, o il valore proveniente dall’uscita del bistabile immediatamente più a sinistra **x** o un valore proveniente dall’ esterno grazie agli ingressi **y**.

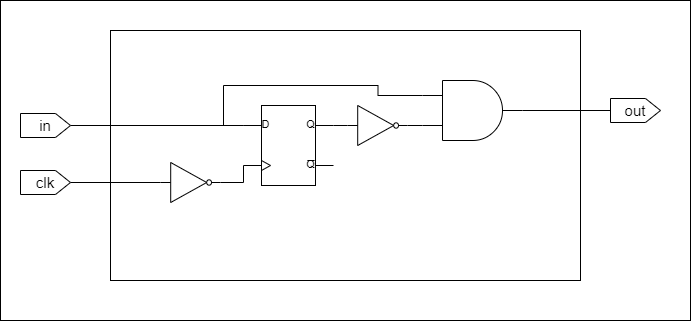


**Sottomodulo Parallelo-Serie**

*RISING EDGE DETECTOR*

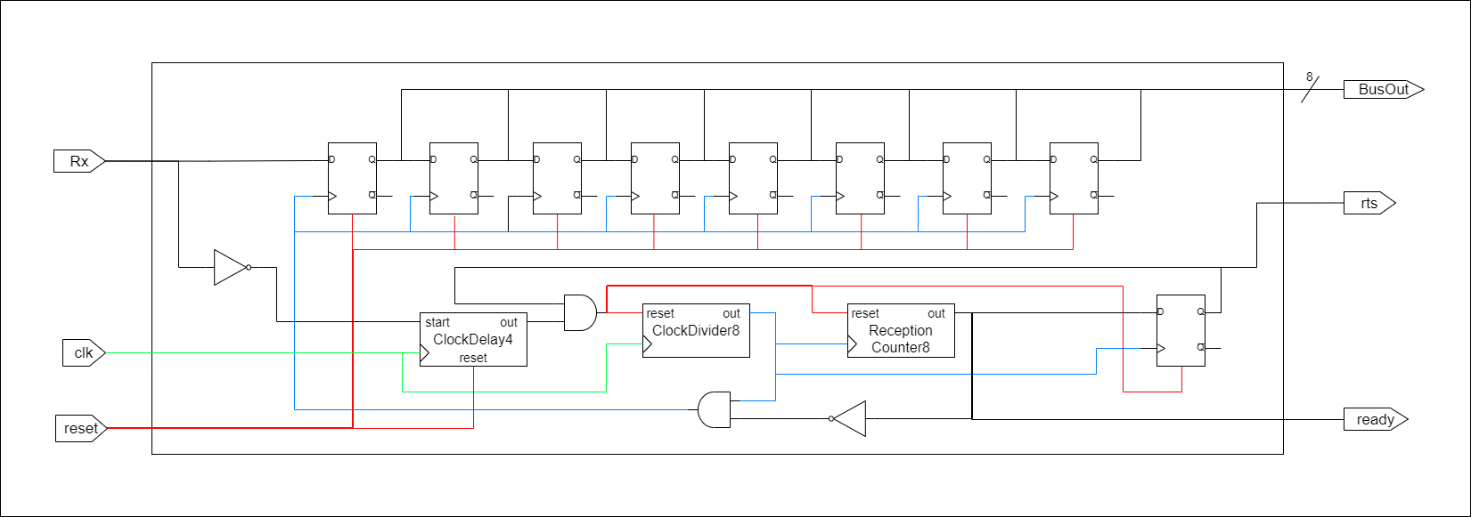
E’ costituito da un bistabile di tipo D. L’uscita negata del bistabile viene messa in ingresso ad una porta AND insieme al segnale d’entrata del flip flop.

Quando rilevato un fronte di salita in input del registro, il risultato della porta AND è 1 fino alla fine del clock corrente. Il clock del Rising Edge Detector è stato negato per fornire il valore in uscita in un momento diverso rispetto al clock che attiva il campionamento degli ingressi nel registro parallelo-serie di 9 bit.



**Sottomudulo Rising Edge Detector**

*DESERIALIZZATORE*



**Rappresentazione grafica dell’architettura del deserializzatore attraverso i sottomdouli**

La struttura del deserializzatore è più complessa di quella di un serializzatore. Essa si basa su un registro serie-parallelo. In questo tipo di registro il caricamento dei bit del dato avviene in modo seriale grazie a un singolo ingresso D, mentre la lettura del valore memorizzato avviene in parallelo. L’architettura di questo registro permette di immagazzinare un bit per volta e di far scorrere i bit già memorizzati per evitarne la sovrascrittura. Ogni uscita Q di un bistabile (tipo D) è connessa in cascata all’ingresso del successivo. In questo modo, a ogni fronte utile del clock ogni bistabile campiona il bit di ingresso e lo mantiene stabile per un ciclo, di fatto realizzando uno scorrimento dei dati in ingresso. Sull’uscita parallela si presentano tutti i bit in uscita da ogni bistabile.

All’interno del deserializzatore deve essere implementata la logica di sincronizzazione per il campionamento dei dati in arrivo attraverso il sovracampionamento.

Il clock in ingresso al deserializzatore è 8 volte più veloce di quello del trasmettitore.

Siccome l’ideale per il ricevitore sarebbe campionare il segnale d’ingresso a metà del tempo di bit del trasmettitore, abbiamo bisogno di 3 elementi: un Clock Delayer 4, un Clock Divider 8, un Reception Counter sincronizzato sull’uscita del Clock Divider 8.

Quando viene il momento del fronte di discesa relativo allo start bit (il trasmettitore abbassa la linea da 0 a 1), il segnale Rx viene negato ed entra nel Clock Delayer 4 (inizialmente contente 4 bit a valore 0). Dopo 4 cicli sull’uscita si presenterà il valore 1 che verrà usato per sincronizzare il conteggio del Clock Divider 8 e di conseguenza anche del Reception Counter.

L’uscita del Clock Divider 8 è importante perché rappresenta il clock del Reception Counter. Esso porta la sua uscita a 0 durante la trasmissione e la mantiene a 1 quando è finita. Questa uscita viene usata per pilotare il segnale di ready e par abilitare il clock dei registri a scorrimento dopo essere stata negata.

Il registro serie-parallelo a 8 bit che riceverà l’intero dato trasmesso è pilotato dall’uscita del Clock Divider 8.

Per non campionare lo stop bit e lasciare tempo al livello applicativo di prelevare il dato sulla porta del bus di uscita del ricevitore, è stato inserito un bistabile di tipo D che ritarda di un clock l’aggiornamento del segnale rts, che richiederà la nuova trasmissione.

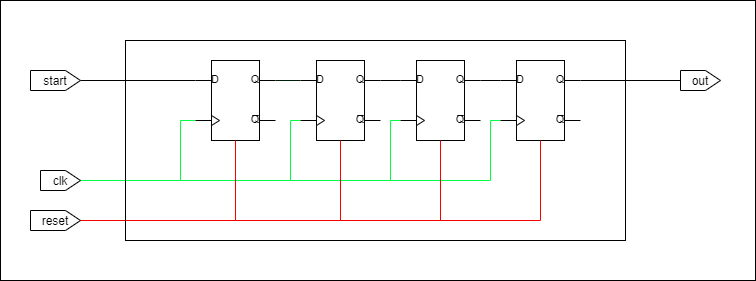
Questo segnale ritardato viene usato anche per dare la possibilità al Clock Delayer 4 di ripristinare la sincronizzazione dei contatori per il campionamento dei dati alla prossima ricezione.

Il segnale di output rts è il risultato di una porta AND che opera sul segnale negato BufferFull e quello ritardato in uscita dal flip flop.

*CLOCK DELAYER 4*

E’ rappresentato da un contatore one-hot mod4 a 4 bit composto da registri di tipo D in serie.

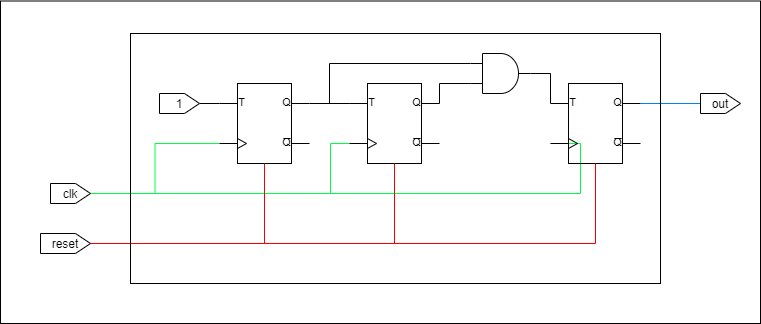
Questo modulo prende in ingresso il segnale Rx negato e ha come uscita lo stesso segnale ritardato di 4 cicli di clock.



**Sottomodulo Clock Delayer**

*CLOCK DIVIDER 8*

E’ rappresentato da un contatore binario mod8 a 3 bit composto da registri di tipo T. Esso porta sull’uscita il valore del bistabile che rappresenta il MSB.

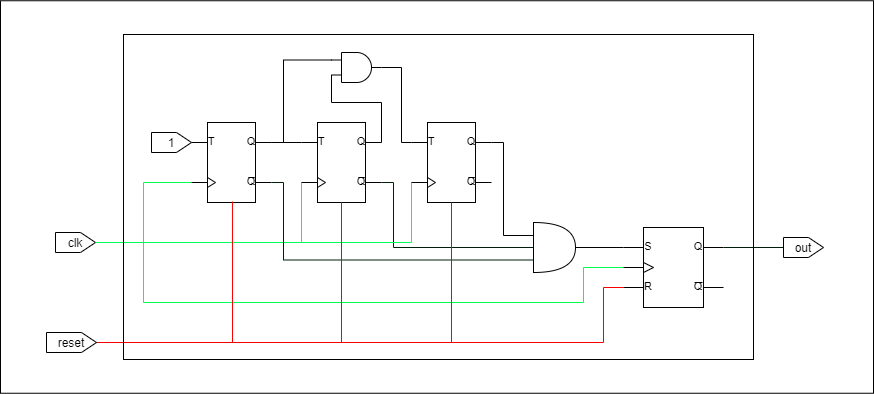


**Sottomodulo Clock Divider**

*RECEPTION COUNTER*

E’ rappresentato da un contatore binario mod8 a 3 bit composto da registri di tipo T.

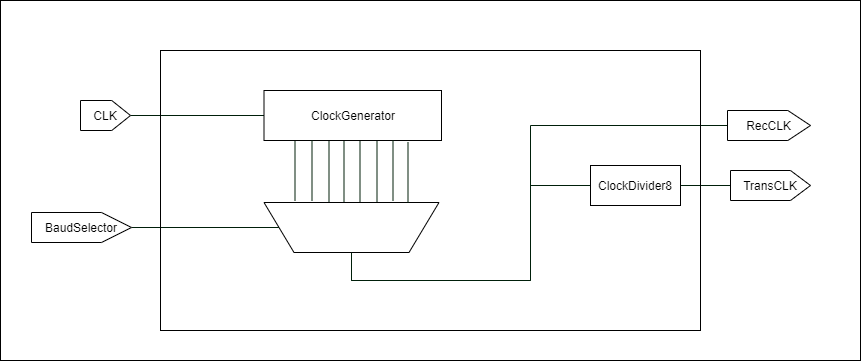
Per mantenere l’uscita alta dopo 8 cicli di clock, l’uscita negata dei primi due flip flop entra in una porta AND con l’uscita dell’ultimo. L’uscita della AND andrà in ingresso a un latch SR. L’uscita viene mantenuta finchè non viene effettuato il prossimo reset dal Clock Delayer 4 nella ricezione successiva.



**Sottomodulo Reception Counter**

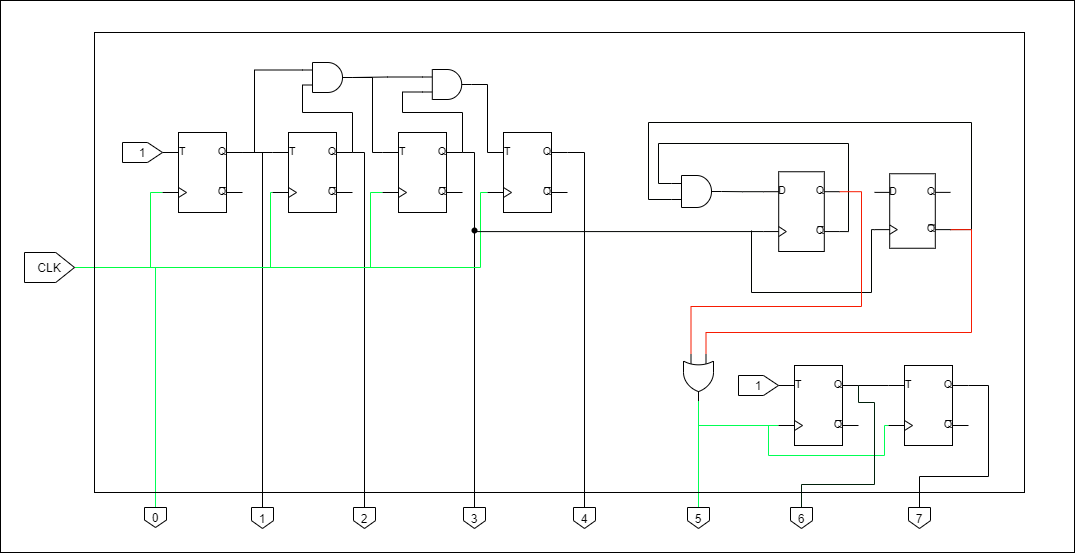
*BAUD GENERATOR*

Il Baud Generator permette di selezionare una della otto frequenze di funzionamento del dispositivo. Un Clock Generator opera sul clock in ingresso all’interfaccia UART del valore di 7372800 Hz estraendo le frequenze desiderate. Le sue otto uscite andranno in ingresso ad un multiplexer di selezione. Il clock di ingresso del serializzatore e del deserializzatore dipenderanno dal baud rate impostato. Il deserializzatore riceverà il clock alla frequenza selezionata direttamente dal mux mentre lo stesso clock, prima di entrare nel serializzatore, viene diviso da un Clock Divider 8.



**Rappresentazione grafica dell’architettura del Baud Generator**

*CLOCK GENERATOR*



**Sottomodulo Clock Generator**

La rete è stata creata osservando i rapporti tra le varie frequenze di baud rate. I valori in uscita dal Clock Generator sono otto volte più grandi delle corrispettive frequenze standard. I primi cinque (921600, 460800, 230400, 115200, 57600)X8 sono ottenuti sfruttando un contatore binario il quale a ogni stadio successivo divide la frequenza a metà. Per ottenere le ultime tre frequenze (9600, 19200, 38400)X8 è stata divisa di un terzo la quarta frequenza e dimezzata in cascata per due volte.

*CLOCK DIVIDER 8*

Questo sottomodulo è già stato descritto in precedenza (vedi pag. 10).

VERIFICA

TEST-BENCH

CASI D’USO