**PROGETTO RETI LOGICHE**

***Interfaccia Seriale UART***

di Flavio Renzi e Federico Viola

**SOMMARIO**

**Introduzione3**

**Specifica4**

Architettura del sistema7

Serializzatore7

Deserializzatore8

Baud Generator9

Verifica10

Test-Bench10

Test 1: trasmissione singolo byte10

Test 2: trasmissione 8 byte11

Test 3: selezione delle frequenze di funzionamento11

Scenari d’uso12

INTRODUZIONE

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

L’interfaccia rappresenta un sistema che permette la comunicazione tra dispositivi, enormemente utilizzato ancora oggi negli apparati industriali.

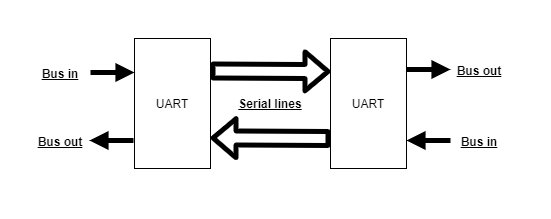
Il nostro progetto ha l’obiettivo di studiarne le proprietà fondamentali e gli aspetti particolari legati alla sua composizione, alla costruzione, al suo utilizzo e funzionamento.

Lo scopo è quello di progettare un trasmettitore ed un ricevitore UART dotati di segnalazione hardware del controllo del flusso.

Per comunicare sono necessari due dispositivi UART, uno utilizzato come trasmettitore dei dati e l’altro come ricevitore. Entrambi possono svolgere questo doppio ruolo. Questo è possibile grazie al fatto che l’interfaccia è Full-Duplex: esiste una porta da cui parte una linea per la trasmissione e una porta 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.

Affronteremo il problema della comunicazione tra due device, in particolare dello scambio di informazioni attraverso una linea di trasmissione seriale. Osserveremo come i dati vengano presi dal bus d’ingresso e portati correttamente su quello d’uscita, assumendo per ipotesi che le linee su cui viaggiano i dati siano collegate a dei generici buffer che noi non specificheremo.

Viene dato per scontato il corretto utilizzo di questa interfaccia, ipotizzando che i dati siano trasmessi in maniera ideale senza considerare problematiche legate a possibili errori di trasmissione e alla loro gestione.



**Questa immagine mostra concettualmente il problema affrontato e la sua finalità**

SPECIFICA

*INTERFACCIA DEL SISTEMA*

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

* Per il ruolo di trasmettitore, l’interfaccia presenta una porta a cui è collegato un segnale **BUS\_IN** 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 da una 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 su una porta **RX** e attraverso un deserializzatore, li porterà in parallelo su **BUS\_OUT** 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 **RX\_READY** (1 bit) in uscita per segnalare che è un byte è pronto sul bus di uscita.
* Un segnale **TX\_READY** (1 bit) in uscita per segnalare quando il dispositivo è libero di effettuare una nuova trasmissione.
* 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, dualmente, 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 **CLOCK** (1 bit) in ingresso.
* Un segnale **RESET** (1bit) in ingresso.
* Un segnale **SEL** 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 il segnale in entrata sarà composto da 3 bit.
* Un segnale di “buffer full” **BUFF\_FULL** (1 bit) gestito dal livello applicativo che segnala se i moduli sono pronti a ricevere un nuovo dato.

Attraverso un evento RESET = 1 è possibile portare i segnali nel loro stato di default. Inizialmente la linea di trasmissione ha valore logico 1, come anche TX\_READY, mentre il segnale di START ha valore 0. Quando si verifica l’evento START = 1 è possibile trasferire i dati se anche CTS = 1. Questo corrisponde ad avere il segnale di RTS = 1 in uscita dall’altra interfaccia UART con la quale si vuole instaurare una comunicazione. RTS, infatti, è correlato ad una richiesta di trasferimento dati (lato ricevitore) e che quindi abilita la trasmissione, essendo collegato alla porta CTS (lato trasmettitore).

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 BUFF\_FULL, inizialmente a valore logico 0. BUFF\_FULL è strettamente legato al segnale RTS dello stesso dispositivo perché quando il suo valore è alto, RTS = 0.

Il sistema deve poter eseguire la trasmissione o la ricezione a seconda del valore di questi segnali.

Se tutto procede correttamente, 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 uno “stop bit” 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 e durante tutto il periodo dell’operazione il segnale TX\_READY viene portato a 0.

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.

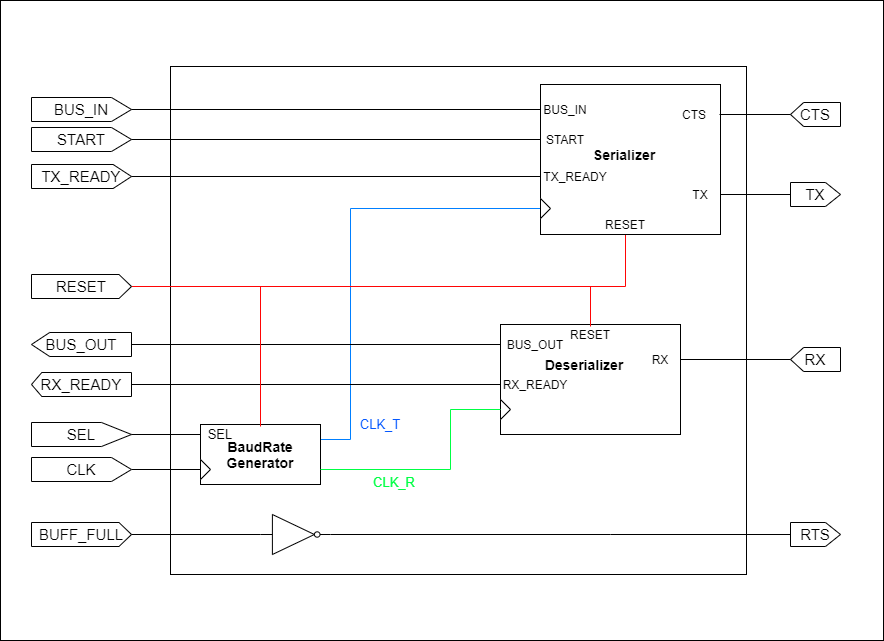
L’ideale per il ricevitore sarebbe campionare il segnale d’ingresso a metà del tempo di bit.

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 1 a 0.

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 BUFF\_FULL = 0 il byte può essere trasferito e verrà lanciato l’evento RX\_READY = 1 (durante la trasmissione si trova a 0) per segnalare che i dati sono stati correttamente inseriti nel bus di uscita.

A questo punto i dispositivi tornano alla propria configurazione iniziale.



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

ARCHITETTURA DEL SISTEMA

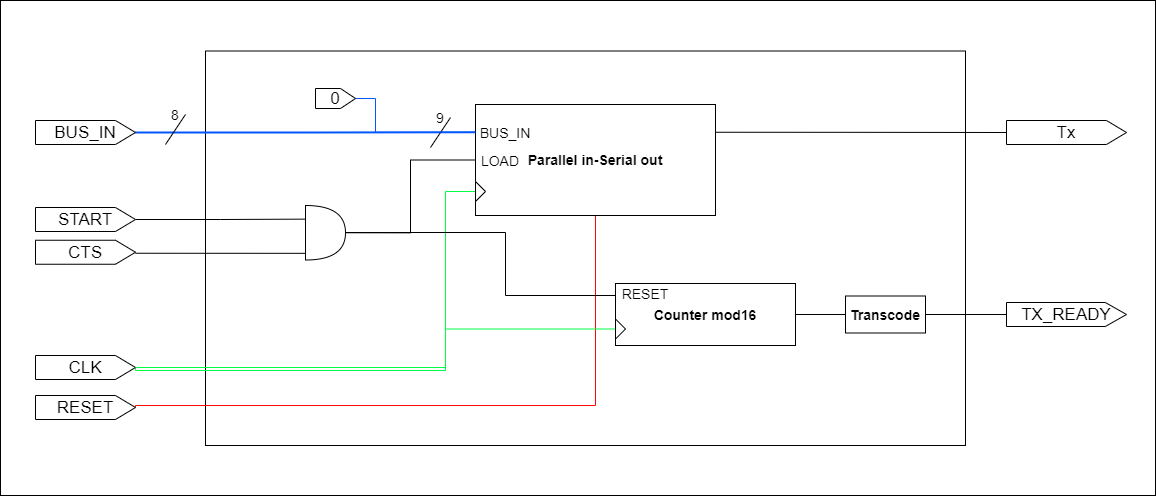
Come abbiamo già osservato, la struttura di un’interfaccia UART è formata da una serie di elementi che hanno una funzione specifica e svolgono determinati compiti relativi alla gestione dei dati e dei segnali, ai fini di una corretta trasmissione delle informazioni.

È 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.

*SERIALIZZATORE*



**Rappresentazione grafica dell’architettura del serializzatore**

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 che porta un bit alla volta su TX.

Il segnale di START entra in una porta AND insieme a CTS, il valore risultante rappresenta il segnale selettore LOAD che determinerà la funzione svolta dal componente.

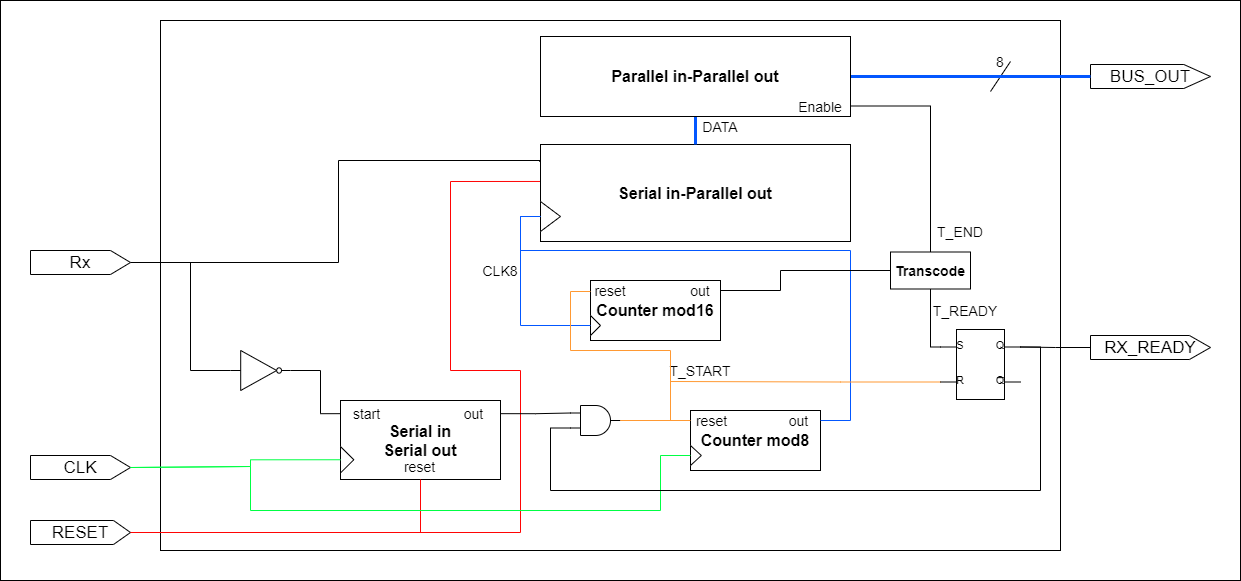
Quando LOAD vale 1, ogni bistabile del registro memorizza il valore presente sull’ingresso dati svolgendo di fatto la funzione di un registro parallelo. Il dato portato in ingresso è di 9 bit: 8 bit del segnale BUS\_IN che rappresentano il byte da trasmettere più un bit costante uguale a 0 che corrisponde allo *start bit*. Il MSB del dato campionato coincide con il MSB del dato proveniente dal bus e il LSB è occupato dallo *start bit.* Non appena il segnale LOAD assume valore 0, gli ingressi dati risultano esclusi grazie ai multiplexer del registro 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 in ingresso al bistabile 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” (**LOAD** = 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.

Il segnale RESET coincide con il preset di ciascun bistabile all’interno del registro parallelo-serie. Esso serve a impostare a 1 il valore iniziale della linea di trasmissione (RESET = 1).

Il contatore mod16 e il latch SR vengono resettati quando LOAD = 1. In questo modo il latch manterrà salvato il valore 0 durante tutto il periodo della trasmissione e di conseguenza anche TX\_READY = 0. Questo significa che il canale trasmissivo del dispositivo è occupato. Al decimo conteggio verrà impostato, grazie alla rete di transcodifica, il valore 1 nel latch SR e TX\_READY = 1 ad indicare che ora è possibile inoltrare un nuovo dato.

*DESERIALIZZATORE*



**Rappresentazione grafica dell’architettura del deserializzatore**

La struttura del deserializzatore è più complessa di quella di un serializzatore. La ricezione e il trasferimento del dato sul bus d’uscita si basano su un registro serie-parallelo e su un registro parallelo-parallelo a 8 bit. Il caricamento dei bit del dato avviene in modo seriale, mentre la lettura del valore memorizzato avviene in parallelo. L’architettura del registro serie-parallelo permette di immagazzinare un bit per volta e di far scorrere i bit già memorizzati per evitarne la sovrascrittura.

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.

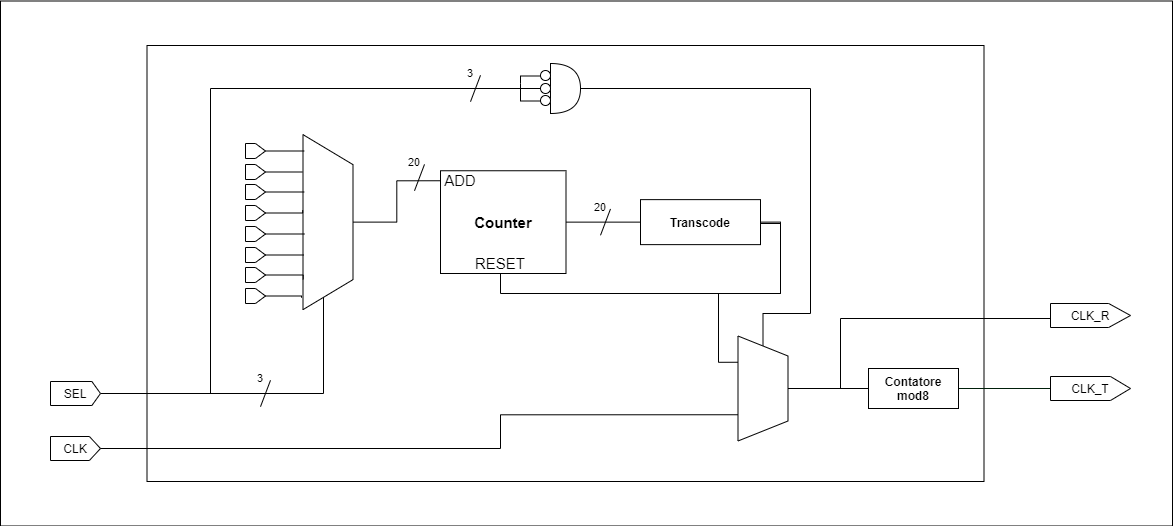
Siccome l’ideale per il ricevitore sarebbe campionare il segnale d’ingresso a metà del tempo di bit del trasmettitore, abbiamo bisogno di 3 elementi: un registro serie-serie a 4 bit, un contatore mod8 e un contatore mod16.

Quando arriva il momento del fronte di discesa relativo allo start bit (il trasmettitore abbassa la linea da 1 a 0), il segnale Rx viene negato ed entra nel registro serie-serie (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 contatore mod8 e di conseguenza anche del contatore mod16.

Il segnale T\_START, ovvero l’uscita del registro in AND con il segnale RX\_READY (si trova a valore 1 quando la trasmissione è terminata e il dato è pronto sull’uscita) permette di resettare i due contatori.

Il valore di output del contatore mod8 e del contatore mod16 sono molto importanti. Il primo rappresenta il clock (prendendo il MSB dei 3 bit) del contatore mod16 e del registro serie-parallelo mentre il secondo, entrando in due reti di transcodifica sui valori 8 e 9, definisce il segnale T\_END, che rappresenta la fine della trasmissione e abilita la lettura del registro parallelo-parallelo, e il segnale T\_READY che imposta il valore salvato nel latch, settato a 0 prima che inizi la trasmissione, a 1 e, di conseguenza, TX\_READY = 1.

*BAUD GENERATOR*



**Rappresentazione grafica dell’architettura del baud generator**

Questo componente ha l’importante funzione di fornire i rispettivi segnali di clock al serializzatore e al deserializzatore avendo bisogno, questi due moduli, di segnali a frequenze differenti.

Il clock in ingresso alla porta dell’UART ha un valore fissato a livello hardware 8 volte il massimo baudrate di trasmissione selezionabile dall’utente. Esso viene modificato in base al segnale SEL di 3 bit in ingresso che rappresenta la frequenza desiderata di funzionamento del dispositivo.

Per poter compiere questa operazione, all’interno del modulo è presente un contatore mod921600 che a ogni ciclo del clock in ingresso incrementa di una quantità definita in base alla selezione dell’utente, cioè un divisore del valore massimo associato a ciascuna frequenza utilizzabile. Tale quantità è codificata su 20 bit. Quando il contatore raggiunge 921600, il valore del clock viene negato ed il contatore si resetta. In questo modo viene generato un segnale di clock CLK\_R di frequenza minore o uguale a quello di partenza e proporzionale ad esso. CLK\_R è il clock del deserializzatore, che deve essere otto volte più veloce di quello del trasmettitore per poter implementare la logica del sovracampionamento. Il clock in ingresso al serializzatore vieni quindi rallentato attraverso l’utilizzo di un contatore mod8, dalla cui uscita viene osservato come clock il MSB.

Se la frequenza selezionata è quella massima non c’è bisogno di modificare il clock iniziale, che viene preso direttamente per definire i due clock secondari.

VERIFICA

Il dispositivo che è stato realizzato può svolgere sia il ruolo di trasmettitore che di ricevitore. L’approccio che è stato deciso di seguire è quello di sperimentare queste due proprietà effettuando dei test su due dispositivi identici.

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.

Testeremo la modalità di comunicazione Full-Duplex supportata dall’interfaccia: se identifichiamo un device\_1 e l’altro come device\_2, può avvenire contemporaneamente la trasmissione/ricezione di un dato in direzione device\_1 🡪 device\_2 e la trasmissione/ricezione di un altro dato in direzione device\_2 🡪 device\_1.

TEST-BENCH

*TEST 1: TRASMISSIONE SINGOLO BYTE*

Dopo un periodo iniziale in cui le due interfacce vengono resettate, è stato trasmesso un byte dal

dispositivo 1 al 2, e viceversa.

Il grafico mostra le variazioni di tutti i segnali entranti e uscenti dalle varie porte di entrambi i dispositivi.

Una volta verificatosi l’evento di START generato dal test-bench e iniziata la trasmissione, è possibile osservare come le due entità, nel ruolo di trasmittente, abbassino la linea logica di TX\_READY comunicando che il canale trasmissivo è occupato mentre, nel ruolo di ricevente, abbassino la linea di RX\_READY nell’istante in cui si raggiunge il centro dello start bit. Questo mostra la corretta sincronizzazione tra i segnali e il funzionamento della logica di sovracampionamento.

Da notare, inoltre, la differenza di frequenza tra i clock associati alla trasmissione e ricezione dei dati per campionare il dato (CLK\_T e CLK\_R).

Immagine che contiene testo, tabellonesegnapunti

Descrizione generata automaticamente

*TEST 2: TRASMISSIONE 8 BYTE*

Questo test permette di osservare la trasmissione in sequenza di 8 byte e la loro corretta ricezione.

Ogni byte viene mantenuto sulla linea di uscita il più a lungo possibile in modo da facilitare l’operazione di lettura del modulo del livello superiore e viene sostituito solo quando uno nuovo è pronto, segnalando l’evento asserendo la linea RX\_READY.

Nel test-bench è stato aggiunto un segnale CHECK che genera un impulso ogni volta che un dato viene trasmesso in modo corretto, confrontando il valore di input con il relativo output.

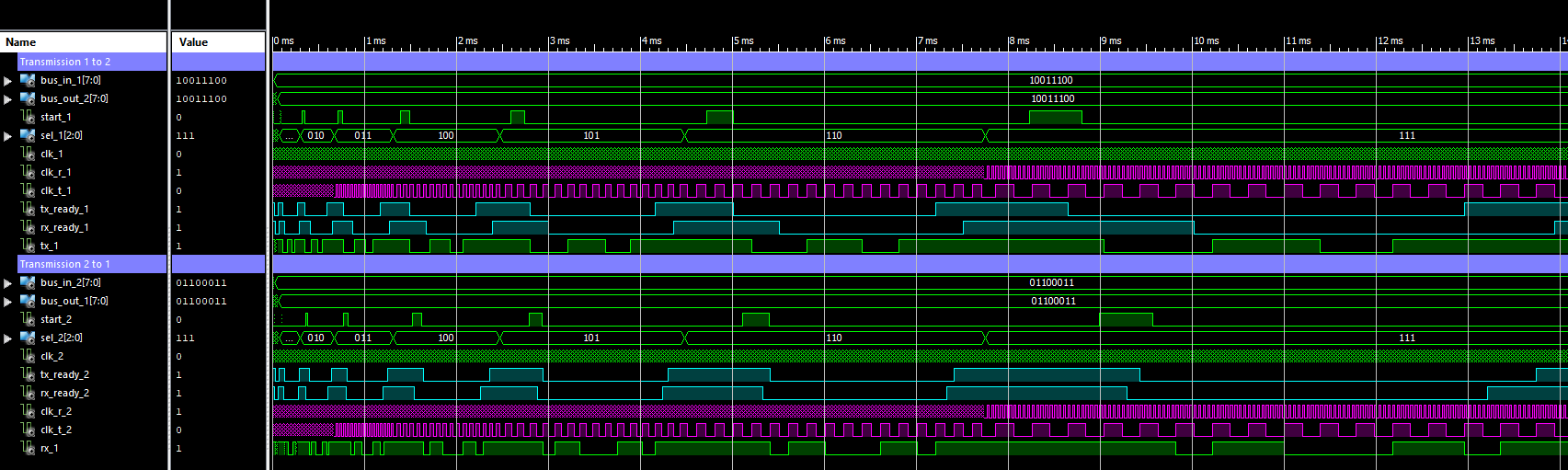
Immagine che contiene testo, computer

Descrizione generata automaticamente

*TEST 3: SELEZIONE DELLE FREQUENZE DI FUNZIONAMENTO*

Attraverso questo test viene verificata la capacità dell’interfaccia di trasmettere un dato utilizzando tutte le possibili frequenze di funzionamento selezionabili.

Facendo variare il segnale di selezione delle frequenze è possibile osservare come il periodo di tutti i segnali venga dilatato.



**SCENARI D’USO**

Abbiamo progettato l’interfaccia per essere utilizzata all’interno di un modulo che abbia collegati alle linee parallele dei bus di entrata e uscita di essa, un buffer per la ricezione e uno per la trasmissione.

Per poter trasmettere, questo modulo dovrà portare il dato dal proprio buffer sulla linea del bus d’ingresso e mantenerlo fino a quando il segnale di start non viene deasserito.

Per garantire il corretto campionamento di un dato, il segnale di start deve rimanere altro per almeno un periodo di clock della trasmissione.

Questo sistema maggiormente complesso potrebbe implementare ulteriori funzioni avanzate come il controllo su possibili errori di trasmissione, la gestione di richieste di ritrasmissione, controllo di congestione e di flusso dei dati.

Il dispositivo UART che è stato realizzato può supportare quindi vari casi d’uso futuri, attraverso la presenza di segnali che permettono lo sviluppo di funzionalità specifiche per lo scenario di utilizzo desiderato.