Esercizio 9: trasmissione tramite interfaccia seriale

L’UART (Universal Asynchronous Receiver-Transmitter) è un dispositivo hardware che converte flussi di bit di dati da un formato parallelo ad uno seriale e viceversa e, generalmente, è parte di un circuito integrato.

Il componente findamentale di tale dispositivo è uno shift register, il quale ci permette di convertire frame paralleli in seriali e viceversa.

Il protocollo di trasmissione implementato dall’UART è di tipo asincrono ed il frame trasmesso è composto da:

* un bit di start
* 8 bit su cui è espresso il carattere
* 1 bit di controllo
* 1 o più bit di stop

Implementando un protocollo di tipo asincrono, a differenza di uno sincrono, manca la serie di byte iniziali che definisce proprio il sincronismo tra le 2 entità che vogliono comunicare.

Quando non ci sono dati da trasmettere, la linea dati si definisce “a riposo” ed è impostata sul livello logico alto; appena invece ci sono nuovi dati da trasmettere, il valore logico passa da 1 a 0, per poi trasmettere effettivamente i dati.

La trasmissione parte, dunque, con un valore logico basso e, di conseguenza, termina con valore logico alto: ciò implica che ci saranno almeno 2 variazioni del segnale durante la trasmissione.

Al fine di interpretare i dati trasmessi correttamente però, il dispositivo in ricezione deve capire, in un intervallo di tempo, quanti bit ha ricevuto; per tale motivo, i 2 dispositivi devono accordarsi su alcuni parametri della trasmissione:

* Velocità dei bit (baud rate)
* Numero di bit per carattere
* Numero di bit di stop
* Numero di bit di parità

Generalmente, il numero di bit totali trasmessi è pari a 10, con 8 bit per carattere, un bit di start, uno di stop e nessun bit di parità.

Il bit di parità è un metodo di rilevamento (ma non di correzione) di un errore avvenuto su un numero dispari di bit, che prevede l’aggiunta di un bit al termine del carattere, in modo che il numero dei bit 1 sia pari.

Tale errore, noto come parity error, non è però l’unico che potrebbe verificarsi; ci sono infatti altri 2 tipi di errore da tenere in considerazione:

* Overrun error: si verifica quando il destinatario sta ancora elaborando il dato precedente e ne riceve uno nuovo; tale errore porta alla perdita del dato.
* Farming error: dopo la rilevazione di tutti i bit relativi al carattere, il dispositivo in ricezione si aspetta un bit di stop; se questo non arriva, vuol dire che c’è stato un errore di campionamento oppure un errore durante la trasmissione.

Per evidenziare la presenza di errori, tuttavia, il dispositivo in ricezione deve essere sicuro di campionare correttamente i valori ricevuti; a questo scopo, la parte in ricezione dell’uart lavora con un clock interno a frequenza molto più alta (in genere 16 volte), in modo da campionare il valore trasmesso “al centro”, distante da fasi transitorie.

Le 2 frequenze di clock presenti vengono implementate mediante degli stati di wait nelle FSM e gestiti tramite i valori di baud rate e baud divide presenti nell’implementazione.

Un ulteriore modo per valutare se il bit ricevuto è pari a 0 o 1 è quello di campionare il dato in ingresso con il clock a frequenza più alta e contare il numero di volte in cui leggo 0 e 1:

il valore che leggo più volte sarà quello che salvo.

Un’implementazione di questo tipo però è più costosa della precedente e raramente viene implementata; in genere si preferisce ritrasmettere il dato piuttosto che effettuare tali controlli.

Al termine di questa breve introduzione sul funzionamento dell’UART, vediamo ora ingressi ed uscite relative alla parte di ricezione e di trasmissione:

Trasmissione:

Ingressi:

* DBIN: dato da trasmettere, formato parallelo.
* WR: segnale di write che dà il via alle operazioni per la trasmissione del dato sul canale.

Uscite

* TXD: dati trasmessi in formato seriale.
* TBE: indica lo stato del buffer di trasmissione: quando è vuoto è pari a 1, 0 altrimenti.

Ricezione

Ingressi

* RXD: dati in formato seriale in ingresso;
* RD: segnale di strobe, ovvero di confermata lettura del dato in ingresso.
* RDA: indica la disponibilità di nuovi dati in ingresso.

Uscite

* DBOUT: dato ricevuto, formato parallelo.
* RDA: indica la disponibilità di nuovi dati in ingresso.
* PE: parity error.
* OE: Overrun error.
* FE: Farming error.

Entrambe le unità, inoltre, presentano un ingresso per il clock ed uno per il segnale di reset.

Esercizio 9.1

Tale esercizio richiedeva una trasmissione di 1 carattere da 8 bit tra 2 sistemi A e B, utilizzando il componente standard UART fornito dalla Digilent.

Si è provveduto quindi ad implementare 2 componenti, A e B, progettati tramite approccio strutturale ma composti unicamente dal componente UART, dato che la stringa da trasferire viene passata dall’esterno, così come i segnali di write e reset.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Il sistema complessivo comprende inoltre 2 debouncer, utilizzati per ripulire i bottoni per i segnali di write e reset.

Si è deciso di realizzare 2 entità e di non utilizzare direttamente i componenti UART per poterle riutilizzare nell’esercizio 9.2.

Sintesi su fpga:

per la sintesi su board, così come richiesto dalla traccia, sono stati utilizzati gli switch per definire la stringa di input al sistema A e dei led per visualizzare la stringa riconosciuta dal sistema B.

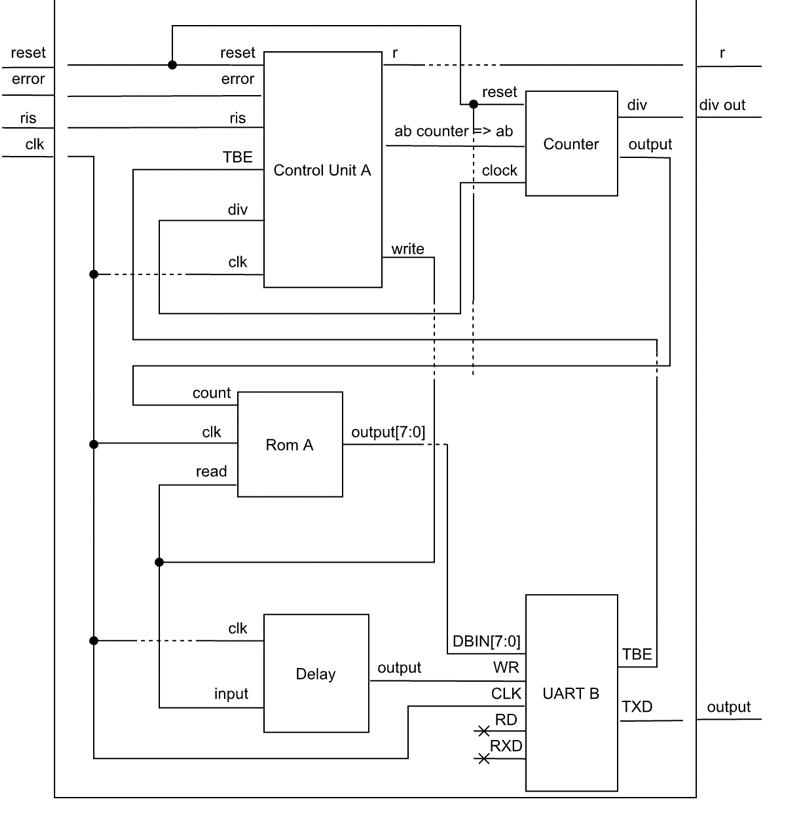
Esercizio 9.2:

L’esercizio è stato impostato sulla falsa riga dell’esercizio 7, dato che la richiesta di leggere un valore in memoria e trasferirlo era analoga. La differenza sta nel fatto che il protocollo di trasmissione non è parallelo ma seriale ed è implementato dall’uart. Cosi come fatto però nell’esercizio 7, viene implementato anche un protocollo di handshaking per la trasmissione di ogni dato (attenzione dato, non bit) al fine di evitare errori di overrun dei dati.

Sistema A:

il sistema A è stato realizzato attraverso un approccio strutturale partendo dall’entita sviluppata nell’esercizio 9.1, e comprende:

* Control unit: ha il compito di effettuare l’handshaking con il sistemaB, inviare il segnale di read alla memoria e contemporaneamente quello di write all’uart, per definire la trasmissione. La control unit ha inoltre il compito di effettuare una ritrasmissione nel caso in cui il sistemaB notifichi un errore in ricezione.
* Contatore: per scandire il termine della trasmissione ed i valori da trasmettere.
* Rom: contenente i valori da trasferire
* Delay block: per applicare un delay ai segnali quando necessario
* UART: responsabile della trasmissione dei dati sul canale.



Control\_unitA:

Immagine che contiene testo

Descrizione generata automaticamente

la control unit è definita come una macchina a stati, dove alcune uscite sono associate direttamente agli stati, mentre altre sono associate alla transizione tra stati in base agli ingressi:

Q\_start -> tale stato è quello in cui si trova la macchina prima di iniziare la trasmissione. la macchina permane in tale stato finché non vede alzarsi il segnale di reset, il quale, essendo uscita di un debouncer durerà solo 1 colpo di clock, che viene interpretato come segnale per partire con la trasmissione. A questo punto la macchina passa nello stato q\_tarsmitt.

Q\_trasmitt -> in questo stato la control unit verifica che il segnale di ris in uscita dal sistemaB sia basso, ovvero che il sistema non stia elaborando un dato precedentemente inviato (nel caso in cui il dato trasmesso sia il primo, tale segnale sarà sempre basso. Questo controllo è necessario per i dati successivi al primo). Se il segnale di ris viene visto basso, allora il sistema alza il segnale di r e passa nello stato q0, altrimenti permane nello stato q-trasmitt.

Q0 -> tale stato è uno stato di transizione per completare l’handshaking, prima di iniziare la trasmissione vera e propria del dato. Alzando il segnale di r, il sistemaA ha espresso la sua intenzione nel trasmettere dei dati, avendo visto che il sistemaB è disponibile a ricevere ed elaborarli, ma non ha ancora saputo da esso se effettivamente è pronto a ricevere tali dati o sta schedulando operazioni con priorità maggiore. Pertanto, il sistemaA rimane nello stato q0 finché non vede alzarsi il segnale di ris da parte del sistema b, sintomo che quest’ultimo è pronto per elaborare i dati trasmessi. Quando ciò accade, si passa nello stato q1 e si abbassa il segnale di r, altrimenti la macchina permane in q0.

Q1 -> arrivati in questo stato vuol dire che può iniziare la vera e propria trasmissione dei dati. Per questo motivo, a tale stato è associata l’uscita write = ‘1’, che viene passata alla memoria rom per emettere il dato( viene vista da essa come un classico segnale di read) e viene inoltre passato con un colpo di delay (attraverso il delay\_bock) all’uart, per segnalare la possibilità di trasmettere i dati in ingresso. Il motivo del colpo di delay è dato dal tempo che la memoria impiega per leggere il dato e porlo in uscita. Si passa a questo punto nello stato q2.

Q2 -> allo stato q2 è associata l’uscita write =’0’. Il motivo è semplice, ovvero il segnale di write deve durare 1 colpo di clock, per evitare ulteriori letture non richieste alla memoria. Il fatto che il segnale di write però si abbassi, non vuol dure che la trasmissione sia effettivamente iniziata. Per questo motivo si effettua un controllo su tbe, ovvero il segnale in uscita dall’uart che, quando è 0 indica che il canale è occupato e dunque la trasmissione è iniziata, altrimenti inidica un canale ancora libero.

A questo punto, quando il valore di tbe è diventato 0, si passa nello stato q3, altrimenti si permane in q2.

Q3 -> è lo stato in cui avviene il controllo dell’errore. Finche ris è alto, ovvero b sta ancora elaborando i dati,si permane nello stato q3, altrimenti si valuta il segnale di error in entrata dal sistema B. Se questo è apri ad 1, non si incrementa ab\_counter, dunque verrà ritrasmesso lo stesso dato, altrimenti viene incrementato e si procederà dunque alla trasmissione del dato successivo se ce ne sono di nuovi.

Indistintamente si passa poi nello stato q4.

Q4 -> in tale stato semplicemente si valuta il valore del div del contatore. Se questo è pari a 1, si ritorna nello stato di start e si attende un nuovo segnale di reset per trasmettere nuovi dati, altrimenti si ripassa nello stato q1 e si attende che il segnale di ris da parte del sistemaB(ecco ora spiegato perché si effettua quel controllo in q1) si abbassi per procedere ad un nuovo handshaking ed una nuova trasmissione.

Si noti inoltre che a tale stato è associata l’uscita ab\_counter=’0’. Il segnale di ab\_counter serve a dare il segnale di conteggio al contatore e dunque trasmettere il dato seguente, cosa che avviene se il dato è stato ricevuto correttamente dal sistemaB e viene abbassato poiché deve durare un colpo di clock per evitare conteggi spuri.

Immagine che contiene testo

Descrizione generata automaticamente

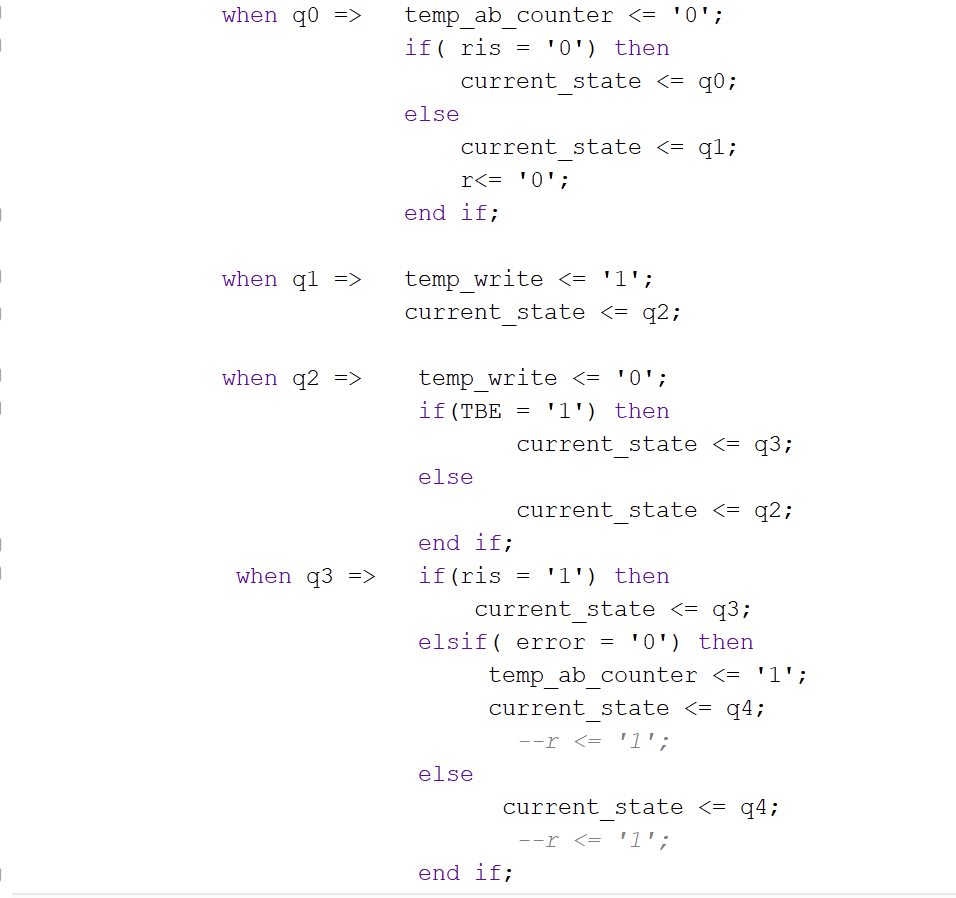


Immagine che contiene testo

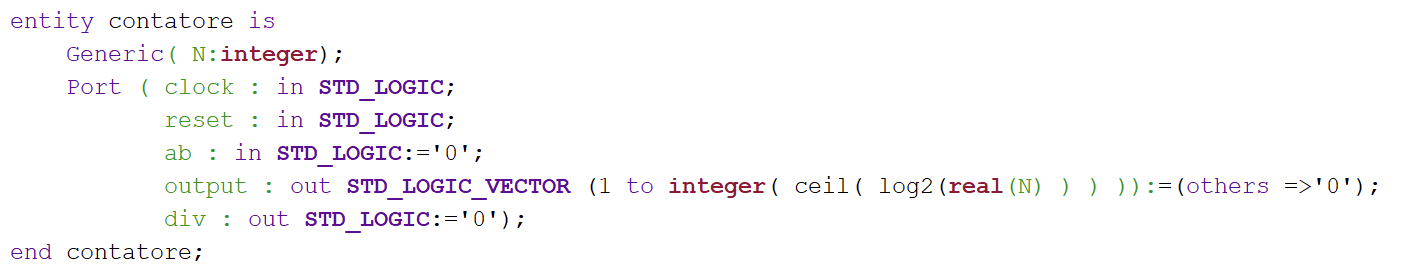
Descrizione generata automaticamente

Considerazioni finali:

La control unit ha come particolarità quella di lavorare sul fronte di discesa del clock, ovvero in opposizione rispetto a tutti i componenti a cui fornisce dei segnali, in modo che questi li vedano quando sono assestati e non durante la fase di transizione.

Il sistema di rilevamento dell’errore e ritrasmissione, implementata tramite il segnale di error, potrebbe andare in loop infinito se gli errori persistono e non vengono risolti con una semplice ritrasmissione. Per evitare ciò andrebbe elaborato un sistema di correzione oltre che uno di ritrasmissione.

Componente contatore:



è stato realizzato tramite approccio comportamentale e l’incremento è definito dal segnale di ab\_counter:

Immagine che contiene testo

Descrizione generata automaticamente

Component ROM:

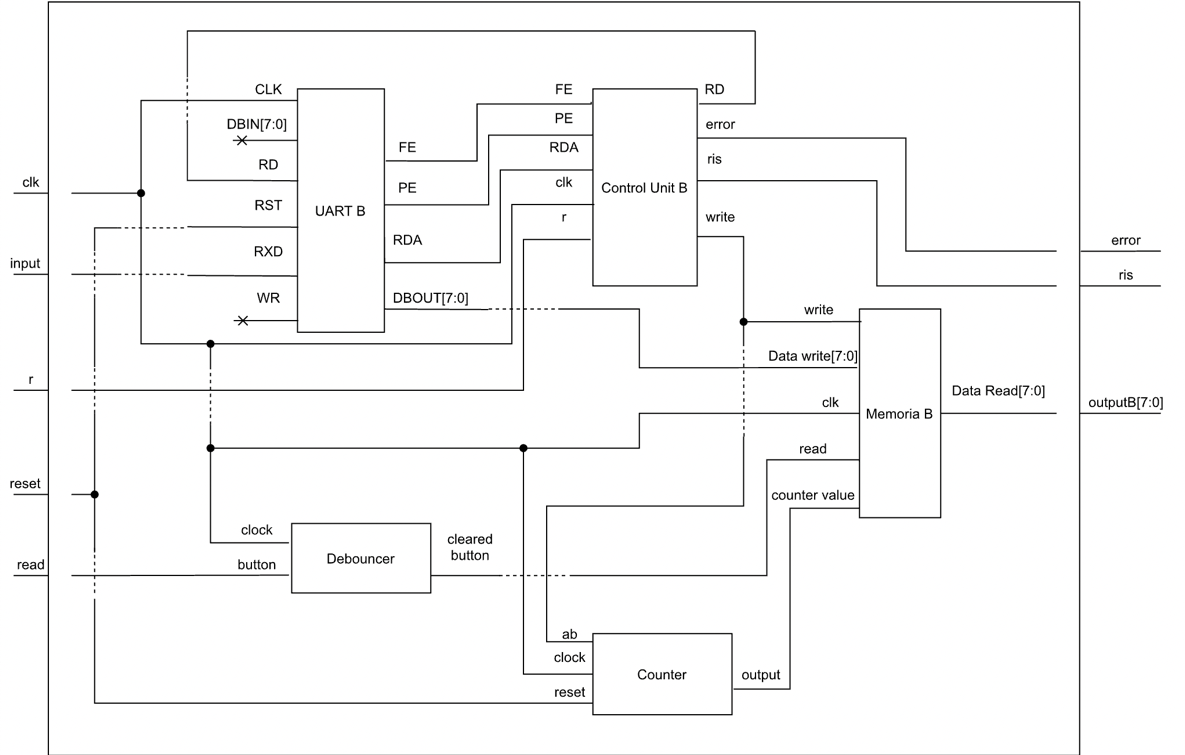
tale componente è analogo a quello utilizzato nei progetti precedenti e pertanto si è ritenuto superfluo riportare nuovamente la sua implementazione.

Sistema B:

il sistema B è stato realizzato attraverso un approccio strutturale partendo dall’entita sviluppata nell’esercizio 9.1, e comprende:

* control unit: per coordinare i dispositivi ed effettuare l’handshaking con il sistemaA.
* UART: responsabile della ricezione dei dati da un canale seriale e del loro posizionamento in parallelo in uscita.
* Contatore: per fornire l’indice della prima cella di memoria disponibile per salvare il dato ricevuto
* Memoria: per contenere i dati ricevuti

Siccome i dati salvati non vengono riportati in uscita, è stato introdotto un debouncer interno per il segnale di read dato dall’esterno tramite un bottone, in modo che ad ogni segnale si scorra di una posizione la memoria e viene riportato in uscita il valore corrispondente.



Componente Control\_unitB:

Immagine che contiene testo

Descrizione generata automaticamente

La control unit è stata realizzata come una macchina a stati, dove alcune uscite sono associate direttamente agli stati, altre alle transizioni in funzione degli ingressi:

Q\_start -> tale stato è quello in cui permane la macchina prima di rilevare un valore di r alto da parte del sistemaA, segno di richiesta di iniziare una nuova trasmissione. Quando viene rilevato tale segnale, si passa nello stato q0 e si pone alto il segnale di ris (segno di disponibilità ad elaborare i dati ricevuti).

Q0 -> si permane in questo stato finchè la ricezione dei dati da parte dell’uart non è terminata e quest’ultimi sono presenti nel buffer di uscita. Per capire quando effettivamente ciò avviene, si sfrutta il segnale rda in uscita dall’uart il quale, quando è patri a 1, indica la presenza dei dati nel buffer. Quando tale segnale viene visto alto quindi, si alza il segnale di RD, che viene dato come strobe per la lettura, e si effettua un controllo sui valori di PE ed FE in uscita dall’uart. Se uno dei due è alto vuol dire che c’è stato un errore nella trasmissione e dunque viene alzato il segnale di error ma non quello di write, ovvero viene notificato l’errore ad A ed il valore letto non viene salvato in memoria.

Nel caso di ricezione corretta, invece, viene alzato il segnale di write ed abbassato il segnale di error (il quale che poteva essere alto nella trasmissione precedete). Non abbiamo necessità che il segnale di error duri 1 colpo di clock, ma solo che il suo valore venga aggiornato prima che si abbassi il segnale di ris, poiché è solo in quel momento che tale valore verrà letto dal sistema A.

Da qui, sia in caso di ricezione corretta che non si passa nello stato q1.

Q1 -> a tale stato è associata l’uscita write =’0’, poiché si vuole che tale segnale rimanga alto solo un colpo di clock per evitare conteggi spuri, dato che tale segnale viene dato come abilitazione al contatore. Viene inoltre abbassato il segnale di RD, il quale sostanzialmente comunica all’uart che il dato posto in uscita è stato letto. Così come definito dal protocollo di handshaking implementato nel esercizio 7 e riutilizzato per questo progetto, si attende che il segnale di r sia basso prima di abbassare il segnale di ris e passare nello stato q\_start.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Così come fatto per la control\_unitA, anche questa control unit è impostata sul fronte di discesa del clock, per far si che le unità controllate vedano i segnali stabili e non sulle trasizioni.

Il componente contatore è analogo a quello utilizzato per il sistema A, mentre il componente ROM è analogo a quello presentato nei progetti precedenti; pertanto, non verrà riproposta la loro implementazione.

A causa dell’alto tempo di simulazione e del numero di segnali da visualizzare, si è ritenuto poco opportuno riportare esempi di simulazione (la quale può essere mandata in esecuzione tramite il file del progetto) e si presenterà unicamente il funzionamento sull’fpga.

Sintesi su fpga:

per riuscire a visualizzare i dati salvati nella memoria del sistema B, si è provveduto ad aggiungere il componente visore, in modo da proiettare su di esso le stringhe in uscita dalla memoria. si è aggiunto inoltre un componente debouncer per il segnale di reset: