**Esercizio 10**

Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch progettato deve operare come segue:

1. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in un rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (ed. nodo 1 più prioritario, con priorità decrescenti fino al nodo 4).
2. (Opzionale) rimuovendo l’ipotesi di lavorare secondo uno schema a priorità fra i nodi e considerando una rete di 8 nodi, lo switch deve gestire eventuali conflitti generati da collisioni secondo un meccanismo a scelta (ad es. perdendo uno dei messaggi in conflitto).
3. (Opzionale) Si implementi un protocollo di handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere) per l’invio di ciascun messaggio fra due nodi.

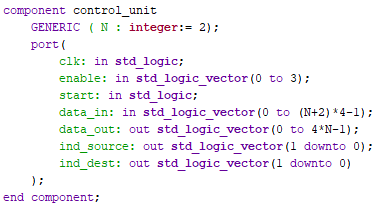
Approccio utilizzato

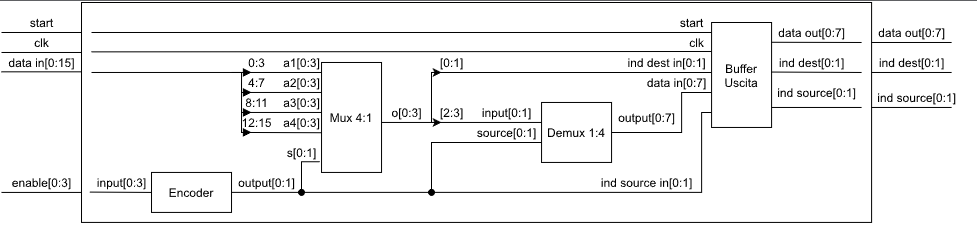
Per la realizzazione dello switch multistadio utilizzando il modello omega network, è stato utilizzata un’architettura composta da una parte operativa ed una parte di controllo. Il numero di nodi sia in ingresso che in uscita è 4 e la priorità è fissa. Si suppone che per iniziare la comunicazione ci sia un segnale di START in ingresso al sistema e che ogni nodo invii un segnale di ENABLE in ingresso al sistema per abilitare la comunicazione. Inoltre si presuppone che possa avvenire una sola comunicazione in base alla priorità, quindi se lo switch è occupato ad instradare un messaggio, allora eventuali altri pacchetti in arrivo andranno perduti.

Si suppone che il pacchetto dati da inviare da una sorgente allo switch sia composto da 2 bit data e 2 bit indirizzo destinazione in ingresso, mentre in uscita dallo switch sia composto dai soli 2 bit data.

Parte di controllo:

(immagine vhdl interfaccia parte di controllo)





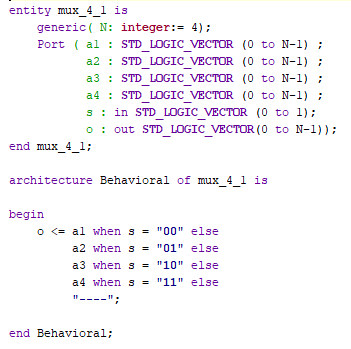
La parte di controllo è realizzata con un approccio strutturale ed ha come componenti: MUX 4:1, DEMUX 1:4, Encoder 4:2 e Buffer. I primi 3 componenti sono stati realizzati utilizzando un approccio Dataflow, generalizzando rispetto al parallelismo dati per il MUX e il DEMUX. La parte di controllo realizza la rete di priorità fissa dei nodi. Questa riceve in ingresso: il pacchetto per ogni linea, 4 segnali di ENABLE provenienti dai rispettivi 4 nodi sorgente e il segnale di START. In uscita produce il pacchetto composto da: 2 bit dato, 2 bit indirizzo sorgente, 2 bit indirizzo destinazione.

A partire dai segnali di ENABLE, l’encoder produce in uscita la codifica della linea abilitata in ingresso utilizzando come schema a priorità fissa dalla prima all’ultima linea. Una volta trovato l’indirizzo sorgente si abilitano le linee dato relative ad esso tramite il MUX 4:1: i primi due bit sono relativi all’indirizzo destinazione, mentre i restanti sono bit di tipo dato. Quest’ultimi attraversano, poi, il DEMUX per avere lo stesso parallelismo dell’Omega network.

I dati, l’indirizzo destinazione e sorgente vengono poi inseriti all’interno di un buffer che li memorizza temporaneamente, ad ogni comunicazione, per poterli inserire all’interno dell’Omega Network.

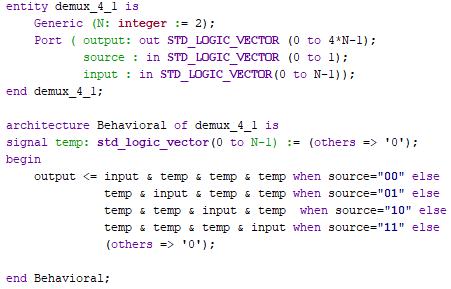
Mux 4:1 :

(immagine vhdl interfaccia mux in control unit)

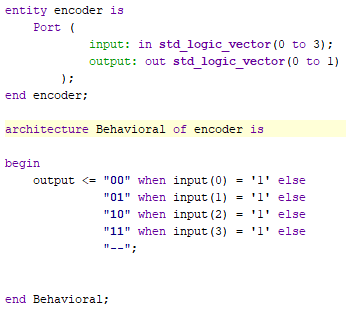


(immagine vhdl mux 4\_1 vhd)

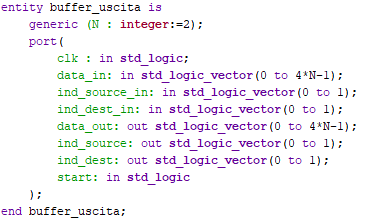
Demux 4:1 :

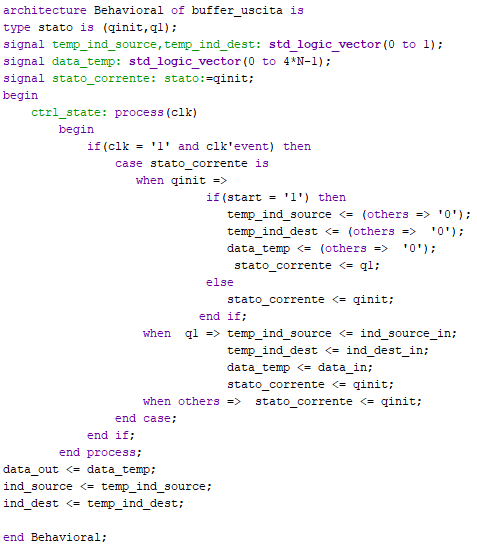


Encoder:



Buffer:



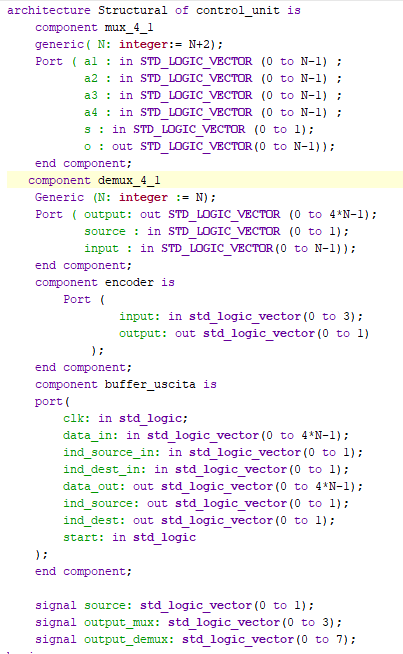


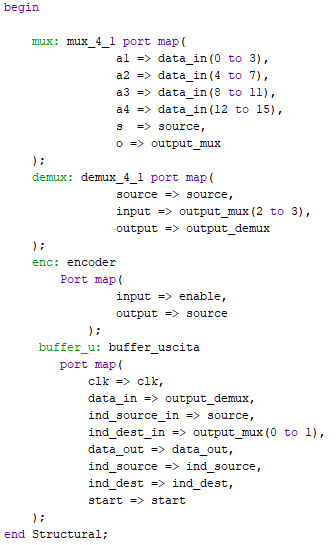
Il buffer è stato realizzato con un approccio Behavioral, cioè come una macchina a stati che ad ogni segnale di start effettua il refresh del pacchetto memorizzato. (Il sistema è stato progettato sul fronte di salita del clock)

Qinit-> Stato iniziale in cui aspetto un nuovo segnale di start. Appena arriva tale segnale effettua il reset del buffer e passa allo stato q1.

Q1-> Stato in cui avviene l’acquisizione del nuovo pacchetto, ed al colpo di clock successivo si sposta in q1, mantenendo memorizzato il pacchetto corrente.

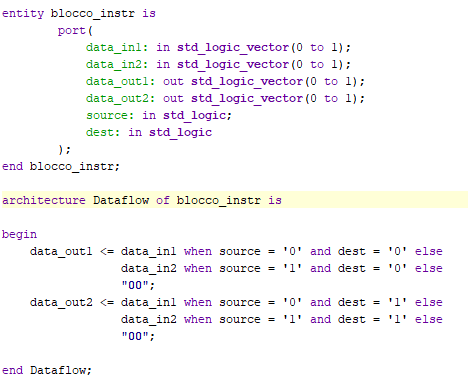
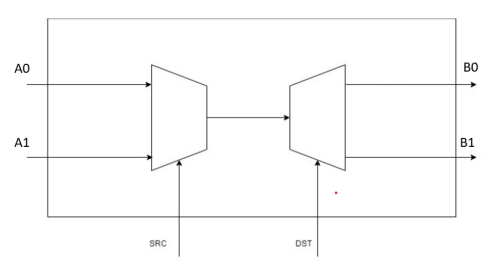
L’architettura Structural complessiva dell’unità di controllo è mostrata in figura:





Parte operativa:

La parte operativa realizza il modello di Omega Network su 4 nodi sorgente e 4 nodi destinazione, è stata realizzata con un approccio strutturale, utilizzando come componenti elementari dei blocchi di instradamento. Questi non sono altro che la composizione di un Mux 2:1 e di un Demux 1:2 : per il primo l’indirizzo di selezione è quello sorgente, mentre per il secondo è quello destinazione. Tale componente è stato realizzato con un approccio Dataflow.

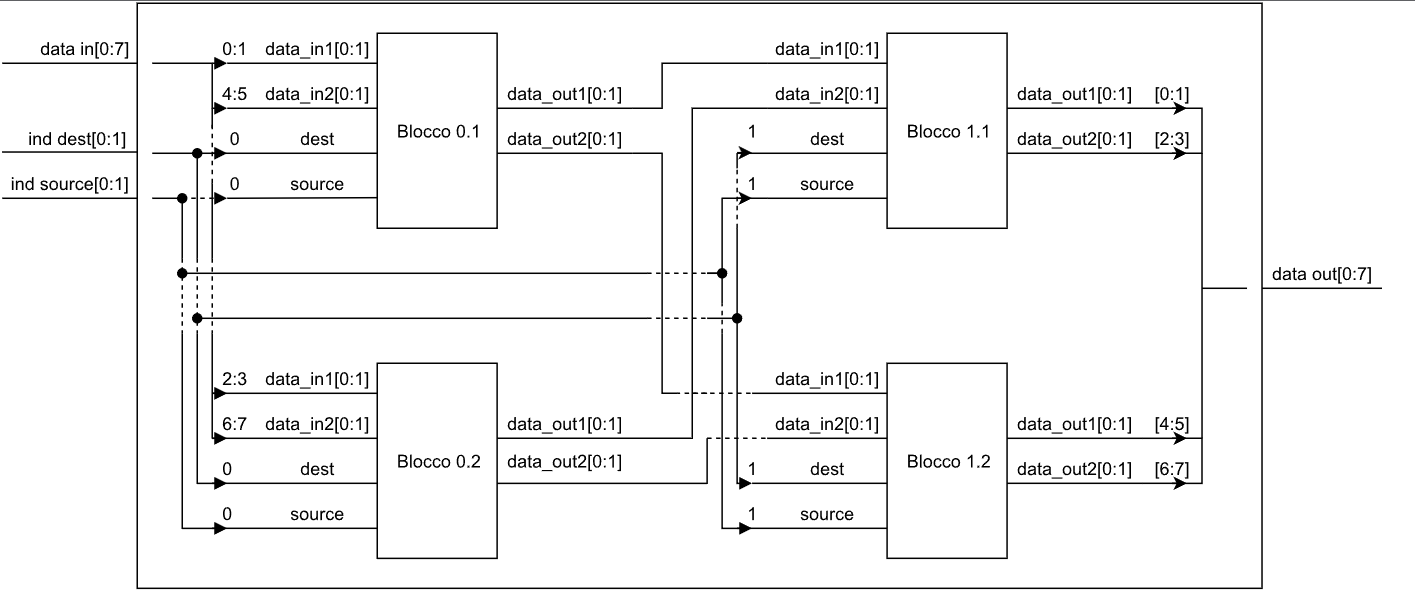


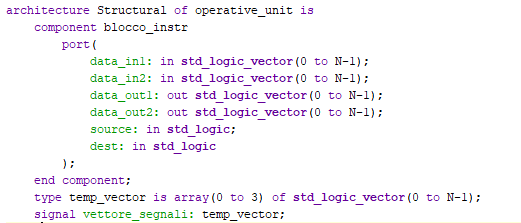
Sfruttando l’approccio strutturale, si collegano i componenti elementari in accordo con il perfect shuffling, si realizza l’Omega Network. Secondo questa tecnica, avendo M carte, dopo un numero di iterazioni del mescolamento perfetto pari a log2 M si riottiene l’ordinamento di partenza, analogamente, per connettere completamente M nodi serviranno log2 M stadi. Nel nostro caso abbiamo 4 nodi da connettere, quindi 2 stadi con 2 switch ciascuno. Le stringhe contenenti gli indirizzi sono lunghe 2 bit, in quanto ci sono 4 possibili indirizzi, avendo assunto che la comunicazione si verifichi da sinistra verso destra, i bit di destinazione vanno dati ai vari stadi (uno per ciascuno) a partire dal bit più significativo, i bit di sorgente invece vanno dati al contrario, dal bit meno al più significativo.

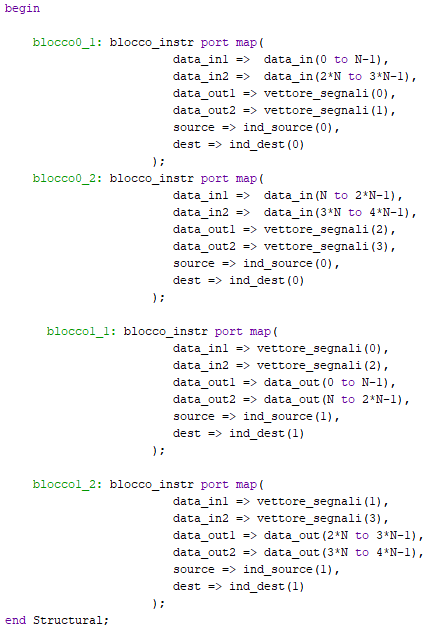
Immagine che contiene testo

Descrizione generata automaticamente

Il generic N permette di scegliere il parallelismo della comunicazione all’interno dell’Omega Network.



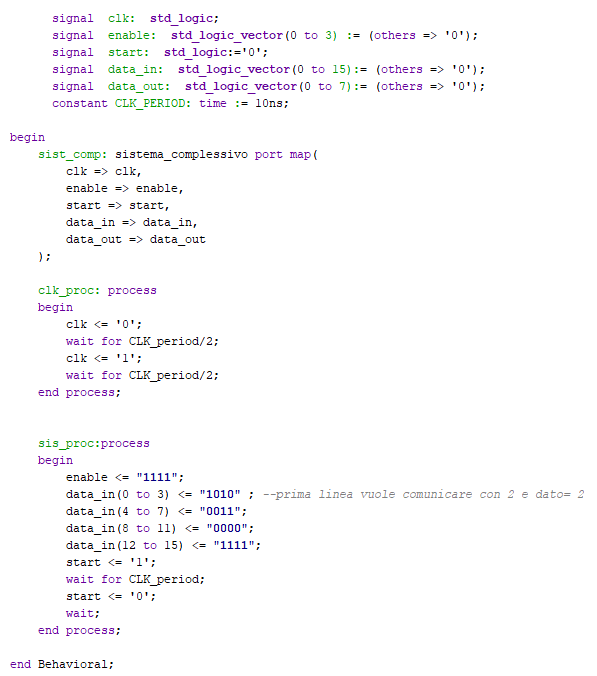




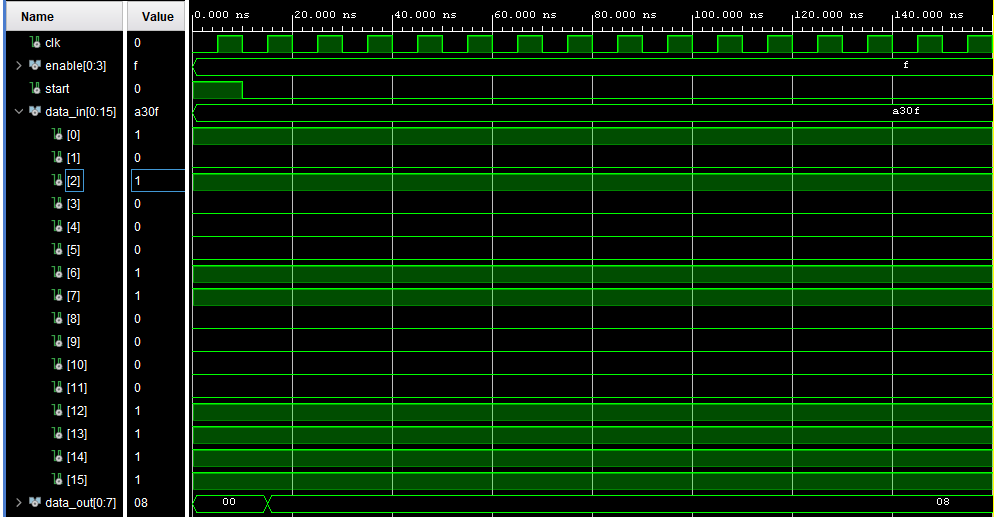
Simulazione:

Per effettuare la simulazione, nel testbench abbiamo utilizzato un sistema con periodo di clock pari a 10ns. Vengono inserite nel sistema i segnali di ENABLE e DATA\_IN per rappresentare la volontà di comunicazione dei nodi sorgente. Si suppone che il segnale di start si alzi per un solo periodo di clock.

(Codice testbench)



(Simulazione immagine)



Dalla simulazione si ottiene che ponendo tutti i bit di ENABLE alti, solo la prima linea riesce a comunicare perché è quella a priorità più alta. Questa comunica con la linea 2 di uscita il dato 2. Essendo tutti i blocchi dell’Omega Network combinatori, la presentazione del risultato in simulazione è istantanea ed avviene non appena è caricato il Buffer con il pacchetto. Nella realtà vi è la presenza di ritardi dovuti alle porte combinatorie.

Considerazioni finali:

Il sistema è funzionante solo per una comunicazione alla volta, secondo una priorità fissa. In realtà sarebbe possibile effettuare anche più comunicazioni all’interno dell’Omega Network, basta che vengano gestite le collisioni: tra nodi sorgente che vogliono comunicare con una stessa destinazione; e tra nodi sorgente che comunicano con diverse destinazioni, ma durante il percorso per raggiungere i nodi destinatari si scontrano.

Una possibile soluzione per poter gestire tali collisioni sarebbe quella di:

-complicare il blocco di instradamento con una coda ed un interfaccia che permettano di memorizzare i dati in arrivo sulle due linee di ingresso;

-realizzare un protocollo di handshaking asincrono (aumentando il numero di linee) tra il blocco trasmettitore ed il blocco ricevente per ogni linea d’ingresso, in maniera tale da stabilire quando il blocco trasmettitore è pronto ad inviare e quando il blocco ricevente è pronto ad acquisire su ogni linea dati.

Ad ogni ciclo di trasmissione da uno stadio al successivo, il blocco in presenza di collisioni ritarda tramite la coda la trasmissione di uno dei pacchetti ricevuto in base ad una priorità ad esempio. In questo modo si realizza un’architettura sincrona rispetto al clock e quindi aumenta il ritardo di trasmissione dal nodo sorgente a quello destinazione per via della gestione degli handshaking.

Inoltre bisogna gestire le comunicazioni concorrenti in ingresso al sistema switch, in particolare se più nodi inviano pacchetti allo switch è necessario che quest’ultimo li acquisisca evitandone la perdita. Una possibile soluzione sarebbe quella di adottare un sistema di handshaking asincrono tra i vari nodi e lo switch.