1 Lezione del 19-11-24

1.0.1 Introduzione alla microprogrammazione

Avevamo visto il concetto basilare di **rete sequenziale sincronizzata complessa**. La sintesi di reti di questo tipo prende il nome di **microprogrammazione**. Bisogna notare che la parola *programmazione* qui è piuttosto fuorviante: l'idea è comunque quella di dare **descrizioni** di hardware.

Il Verilog comprende un sottoinsieme di linguaggio adibito esattamente agli scopi della microprogrammazione, cioè un **linguaggio di trasmissione tra registri**. In ogni statement includiamo:

- μ-istruzioni, cioè assegnamenti a registri operativi;
- μ-salti, cioè assegnament al registro STAR, che possono essere a una o più vie.

Notiamo che qui il μ significa semplicemente "hardware". Si possono ommettere le μ-istruzioni relative a *registri operativi*, che in questo caso implicano conservazione. Per quanto riguarda il registro STAR, invece, l'omissione del μ-salto implicherebbe un salto incondizionato (altrimenti avremmo `STAR<=STAR`, che porterebbe a un **deadlock**). Nella pratica, non ometteremo mai l'aggiornamento di STAR, in quanto porta facilmente a errori, o comunque a codice poco chiaro.

1.1 Temporizzazione di reti complesse

Le temporizzazioni di una rete complessa sono le stesse delle rety di Mealy ritardato: i percorsi sono gli stessi ($T_{in\_to\_reg}$, ecc...) e preleviamo le uscite direttamente dai registri.

1.1.1 Handshake e temporizzazione delle uscite

Solitamente le reti sequenziali sincronizzate comunicano con altre reti sequenziali sincronizzate, idealmente con cicli di clock mutualmente sincronizzati (così da doverci preoccupare solo dei tempi di lettura e scrittura T_{a_monte} e T_{a_valle}). Nel caso le reti che consideriamo siano invece mutuamente asincrone fra di loro, cioè abbiano clock discordi che non si allineano mai (necessariamente), dovremmo adottare soluzioni diverse.

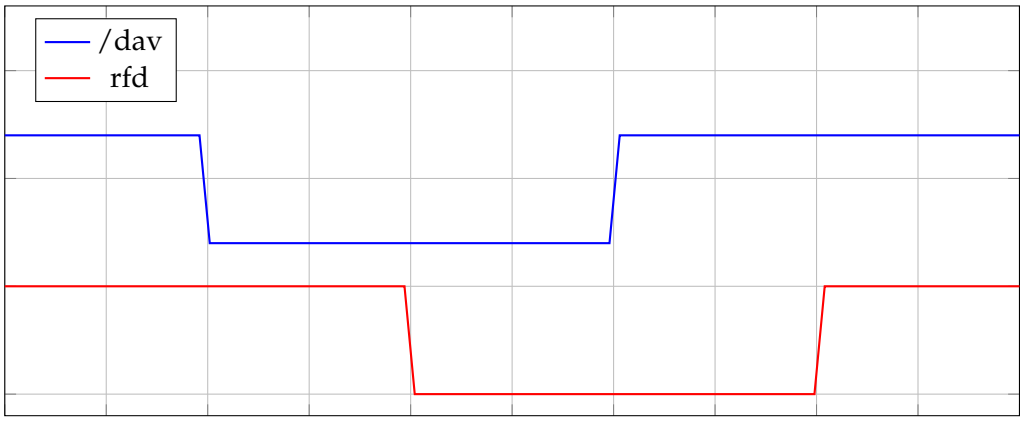
Poniamo una situazione di esempio: una rete, detta **produttore**, mette su una linea un numero rappresentato su 8 bit. Chiamiamo questa linea *linea di trasmissione*. Un altra rete, detta **consumatore**, riceve questo numero e tiene la sua uscita alta per il numero di cicli indicato dal numero ricevuto. Visto che le due reti vedono solamente i rispettivi input e output, come facciamo in modo che il produttore sappia quando il consumatore ha letto il numero con successo, e viceversa che il consumatore sappia quando c'è un nuovo numero da leggere? Il problema si risolve introducendo **linee di handshake** (dall'inglese per *stretta di mano*).

1.2 Handshake /dav-rfd

L'handshake che andiamo ad implementare viene detto **handshake /dav-rfd**. Doteremo quindi il produttore di una linea di uscita `/dav` (*data valid*), e il consumatore di una linea di uscita `rfd` (*ready for data*). La linea `/dav` segnala che ci sono nuovi dati sulla linea di uscita del produttore, mentre la linea `/rdf` segnala che il consumatore ha letto ciò che era sulla linea di trasmissione ed è pronto a ricevere nuovi dati. Entrambe le variabili sono attive basse.

Facciamo una nota sulla circuiteria di **reset**: la linea `/reset` sarà infatti presente e comune alle due reti. A `/reset` bassa, quindi, possiamo mettere `/dav` e `rfd` a 1 (per segnalare che linea di trasmissione non è pronta e il consumatore non è pronto a leggerla).

Viste su un grafico, i valori logici delle variabili `/dav` e `rfd` nel corso di un handshake valido si evolvono come segue:



A seguito dell'handshake, vorremo effettuare una trasmissione vera e propria di dati. Innanzitutto, il produttore metterà un numero sulla linea di trasmissione. In seguito, metterà `/dav` bassa per segnalare che i dati sulla linea di trasmissione sono pronti. A questo punto, il consumatore dovrà rilevare il `/dav`, leggere i dati sulla linea di trasmissione e mettere il suo `rfd` basso. Questo segnalerà, per la rete produttore, che il consumatore **ha letto** i dati con successo ed è pronto ad un nuovo ciclo di trasmissione. Da qui in in poi, il consumatore non potrà più aspettarsi che i dati sulla linea di trasmissione siano validi: in qualsiasi momento il produttore potrebbe aggiornarli e rialzare `/dav` (o viceversa, rialzare `/dav` e poi scrivere sulla linea, ciò che importa è che il consumatore non ha più nulla da leggere fino a un nuovo ciclo di abbassamento del `/dav`). Quando `/dav` si alza, quindi, anche il consumatore riporterà il suo `rfd` alto, e ci troveremo nella situazione di partenza (cioè `/dav` e `rfd` alti). Dobbiamo stare attenti al fatto che `rfd` torna alto **dopo** che lo fa `/dav`: altrimenti il produttore potrebbe perdersi la doppia transizione di `rfd`, e finiremmo in uno stato di deadlock. Possiamo riassumere quest'ultima affermazione come segue: una corretta sincronizzazione delle reti avviene **solamente** se si **alternano** le transizioni delle linee di handshake (a ogni aggiornamento del produttore segue un aggiornamento del consumatore, e così via, senza che nessuno faccia doppi aggiornamenti "di testa propria").