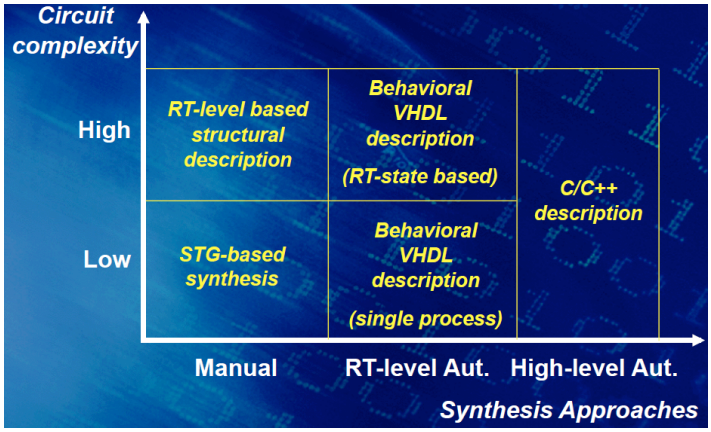
Sequential Logic Design – Approaches

Approaches to Sequential Logic design

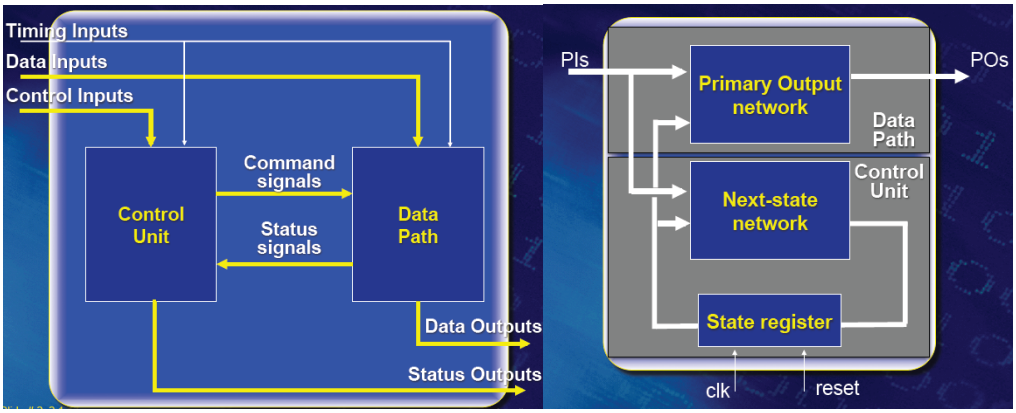
Premessa: ci sono due diverse definizioni di circuito sequenziale a seconda del contesto in cui lo considero, in termini di comportamento un circuito è sequenziale se le uscite dipendono anche dal valore degli ingressi assunti in precedenza precedenti, in termini strutturale una netlist è sequenziale se esiste un anello di reazione (cioè se partendo da un punto si torna al punto iniziale passando attraverso il circuito).



Se pensiamo di avere da una parte la complessità del circuito e dall'altra gli approcci alla sintesi (manuale, tool a livello RT o a un livello più alto) otteniamo soluzioni diverse.

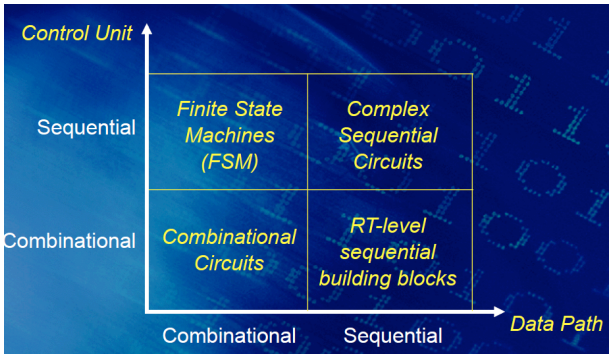
Parlando di sintesi manuale, se progetto circuiti piccoli uso una metodologia che è l'equivalente della copertura della mappa di Karnaugh, se la complessità cresce allora passerò al livello RT. Quando userò tool di sintesi automatica, ci sono due approcci a seconda della complessità del circuito.

Come possiamo stimare la complessità del circuito? Una prima soluzione è **analizzare la control unit del sistema target.**



Fonte: TestGroup

Schema generale di un circuito sequenziale: input -> PI's, clock e reset; output-> POs, all'interno è formato da una primary output network, un next state network e un state register. Facendo il merge di questi, tipicamente il data path è la parte che si occupa di generare gli output, lo state register e il next-state invece compongono la control unit.



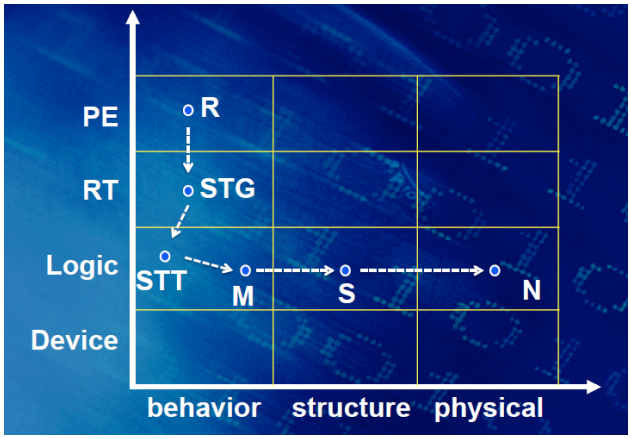
A sua volta, la control unit e il data path posso essere combinatori o sequenziali. In totale ho quindi

4 possibili configurazioni.

Se entrambi sono combinatori, il circuito è combinatorio, *se DP è combinatorio e la CU è sequenziale ho una Finite State Macchine* (FSM, finite perché ho un numero finito di flip-flop). Nel caso in cui DP è sequenziale e CU è combinatorio ho quelli che in questo corso considero i blocchetti elementari a livello RT. DP e CU entrambi sequenziali non li trattiamo in questo corso.

Manual Synthesis

Adesso ci occupiamo di sintesi manuale basata su STG. I passi sono:



- 1 Fonte: TestGroup

- **STEP 0**: Requisiti dell'utente.
- **STEP 1**: nella sintesi manuale di FSM questo è il passo più difficile, solo qui occorre pensare. Il vero problema sarà arrivare al **STG (State Transistions Graph)**. Il primo passo è riuscire a ottenere a livello comportamentale *una descrizione di come si dovrebbe comportare il circuito in funzione dello stato in cui si trova e dei valori degli ingressi*. Per fare questo si ottiene prima lo STG e poi STT. L'obiettivo è arrivare un grafo in cui i cerchi identificano gli stati e gli archi hanno una ben determinata semantica. Questa struttura deriva dalla teoria dei grafi che riprenderemo in termini di programmazione. Il primo passo è ottenere il grafo (STG) e da questo passare a una tabella (STT) che ha le stesse informazioni del grafo ma in forma un po' diversa (un po' come tra la mappa di Karnaugh e la tabella delle verità, in entrambe risiedevano le stesse informazioni ma la forma in cui sono espresse è diversa).
- **STEP 2**: dalla STT bisogna passare al livello logico: devo ottenere una mappa di Karnaugh per ogni variabile e per ogni stato. Il mio obiettivo è realizzare due reti combinatorie per i present state e per i next-step (per facilitarmi la vita suppongo che queste due mappe siano indipendenti).
- **STEP 3**: copro la mappa di Karnaugh come avevo già fatto prima.

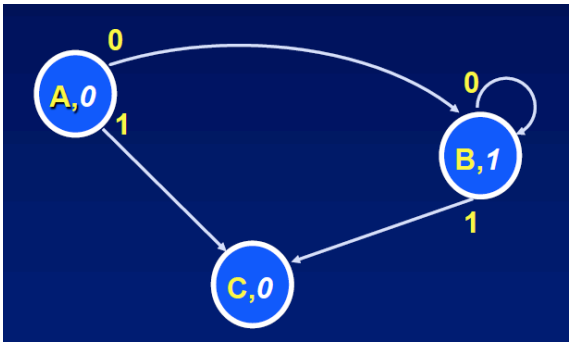
FSM Manual Synthesis

State Transition Graphs (STG's)

Avevamo già parlato di machine di Moore e di machine di Mealy, noi ora ci concentreremo su quelel di Moore.

Nel STG per le macchine di Moore (le uscite cambiano solo quando cambia lo stato) a ogni stato

corrisponde un'uscita. Un STG è uno grafo ordinato (gli archi hanno un verso ben definito) e sia i nodi sia gli archi sono etichettati.



I **nodi** (cerchi) rappresentano l'insieme degli stati S, a ogni stato corrisponde un nodo nel mio STG. Date le specifiche posso capire quanti stati avrà il mio circuito? In genere non è prevedibile a priori (tranne casi particolari). Ogni stato è caratterizzato da un'etichetta (*hints: l'etichetta deve serre la più mnemonica possibile*, così come i segnali e le variabili in programmazione, devono avere una *semantica chiara e "parlante"*), l'etichetta deve essere univoca, e ogni stato ha un suo proprio insieme di uscite., in genere lo faccio mettendo dopo l'etichetta la virgola seguita dai valori che le uscite assumono a quello stato.

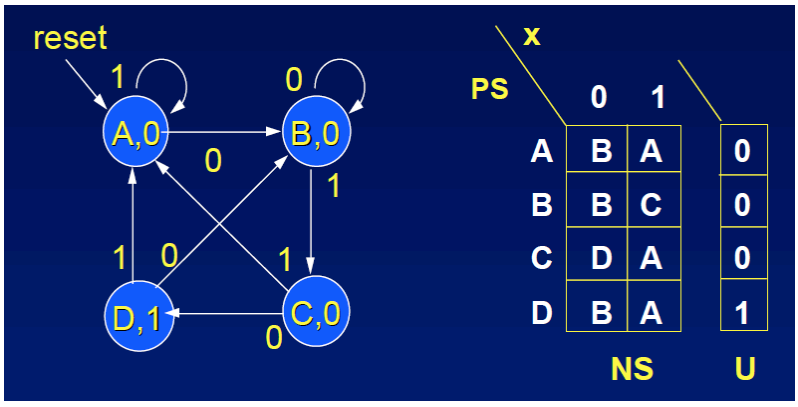
Archi: l'insieme degli archi rappresenta le possibili transizioni fra gli stati, passo da uno stato all'altro quando cambia lo stato, in pratica a ogni colpo di clock. *Esiste un arco tra il nodo S_i e il nodo S_j se c'è un valore degli ingressi che forza la transizione da S_i a S_j*. Sull'arco metto una label che indica il valore di ingresso che fa effettuare la transizione. Ovviamente sono possibili i self-loop, ovvero se sono in B e mi arriva un colpo di clock allora posso rimanere in B. Per ciascun stato deve prevedere tutte le possibili configurazioni degli input (2^n con n=#PI's). L'**out-degree** indica *il numero di archi che escono da uno stato*. In generale non posso sapere quanti archi entrano in un nodo (dipende da cosa fa il circuito). L'STG è un grafo orientato perché devo sapere se vado da A->B o da B->A.

La difficoltà è nel mappare il mio problema e trarne un modello equivalente in teoria dei grafi, poi gli algoritmi già ottimizzati li trovo in letteratura e non devo reinventarli. Il grafo è un disegno, la semantica dipende dal problema che ho davanti (adesso il cerchio è lo stato del mio circuito, ma potrebbe essere una fermata dell'autobus o una città). L'informazione associata al grafo dipende dal problema che sto trattando.

Grafo per macchine di Mealy: è diverso da quelle di Moore, in ciascuno stato non è più vero che le uscite assumono solo un valore ma dipendono a loro volta dagli ingressi.

State Transition Table (STT)

Una volta che ho l'STG minimizzato (un po' come per gli implicanti principali, io voglio fare a mano circuiti che siano minimizzati in termini di area, nei sequenziali devo cercare di non aggiungere stati superflui altrimenti avrei un incremento logaritmico), dovrò ottenere due tabelle, una per le uscite e una per i next-state, dopo di che considero le mappe di Karnaugh



- 2Fonte: TestGroup

per entrambe e le copro (per comodità in modo separato).

Nella STT ho tante righe quanti sono gli stati e quante colonne quante sono le possibili disposizioni degli ingressi. Una mappa di questo tipo corrisponde a: sono in stato I, passo allo stato K quando l'ingresso vale J. È solo una traduzione banale da grafo a tabella. Per le macchine di Moore la tabella dei Primary Output sono a una sola colonna perché i PIs sono dei don't care.

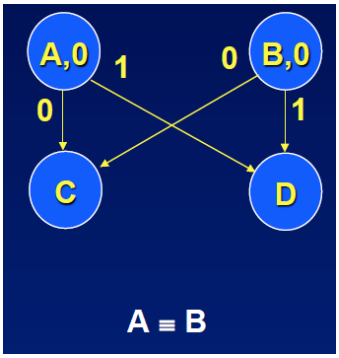
Reset signal e reset state

Devo obbligatoriamente avere un segnale di reset e uno stato di reset, devo progettare il circuito in modo che funzioni veramente (cioè che il reset signal abbi priorità massima). Per convenzione il segnale di reset è asincrono (è un freccia che arriva dal nulla) e da li parte tutto.

Caveat: il segnale di reset, quando diventa attiva, azzera lo state register. Dopo di che le uscite dipendono dallo stato ma in mezzo ho una rete combinatoria per cui non è affatto automatico che quando lo resetto allora le uscite sono tutte azzerate. Stato di reset NON IMPLICA uscite tutte azzerate.

Minimizzazione degli stati

Una volta che a mano ho completato l'STG, devo riguardarlo attentamente alla ricerca degli stati equivalenti.



Def 1. *Due stati sono equivalenti se e solo se per ciascuna combinazione degli ingressi hanno la stessa uscita e lo stesso stato futuro.*

Esempio: A e B sono equivalenti. Il circuito si comporta allo stesso modo sia che si trovi in A sia che si trovi in B. Quando progetto a mano è abbastanza facile cadere in queste situazioni.

Se lavoro sul STG devo fare un'ispezione visuale accurata per scoprire se ho degli stati equivalenti, se lavoro sulla STT posso pensare di accorgermene facendo un match fra le righe (se due righe sono uguali allora gli stati sono equivalenti). Una volta scoperto che lo stato A è equivalente allo stato B, come lo risolvo?

- Ne cancello uno dei due e gli archi pendenti li mando in quello rimasti. Se mi accorgo che con 0 e 1 vado da Y sempre in B, allora Y è un don't care e va sempre in B.
- Faccio un merge fra i due.

Analogamente potrò fare qualcosa di simile se sono sulle STT.

Def. 2 (alternativa): *due stati sono equivalenti ⇔ hanno le stesse uscite e mandano in stati equivalenti (a mano non ci capiteranno questi casi).*

Quando faccio la sintesi manuale cerco già di minimizzare gli stati, se poi me ne scappa qualcuno riesco a recuperalo, esistono anche degli algoritmi molto complicati per la minimizzazione che non vedremo in questo corso.

STG Design Examples

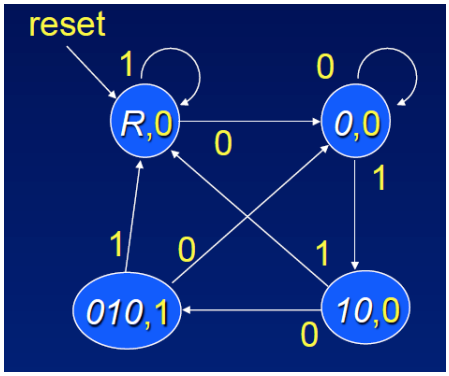
ESERCIZIO 1

" On a serial transmission line X, bits are transmitted synchronously w.r.t. a clock signal CLK, one bit per clock cycle. A circuit to be connected to the serial line is to be designed. It samples the line X at each clock cycle. Whenever the sequence '010' has been received, it asserts its output signal Z for one clock cycle. A '0' cannot be used, at a same time, as the last bit of a sequence and as the first bit of the next one."

A ogni colpo di clock ho un bit, il circuito deve campionare gli ingressi e non appena riconosce la sequenza 010 l'uscita deve andare a 1 per un colpo di clock. 01010 non sono due sequenze distinte: non posso usare due volte lo stesso bit.

Hint 1: in generale è conveniente strutturare l'STG in base all'evoluzione temporale del circuito target.
Hint 2: usare nomi parlanti.

Parto sempre dallo stato di reset. Se mi arriva uno zero vado in uno stato che chiamo "0" dove l'uscita vale ancora zero. Se poi mi arriva l'uno cambio stato e vado in "01" con l'uscita ancora a zero, se poi mi arriva un 1 vado in "010" allora l'uscita varrà 1 e al prossimo colpo di clock dovrò uscire da quello stato. Non ho ancora finito (ho finito se da ogni nodo escono 2^#PIs). Se sono in 010 e mi arriva uno zero allora lo interpreto come un possibile nuovo inizio di sequenza, se mi arriva un 1 non lo devo considerare->torno a reset e ci sto fin quando non ho uno



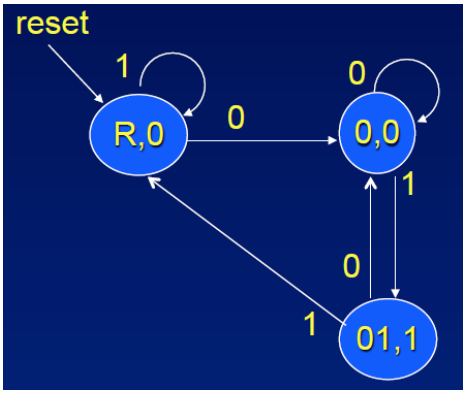
- 3Fonte: TestGroup

0. Nello stato 0 ci sto fin quando non ho un uno. Non ho stati equivalenti e quindi ho finito la parte di STG.

Esercizio 2

"Design a Pulse generator, i.e., a sequential functional block able to generate a 1-clock period pulse whenever a rising edge is detected on its input IN."

Assunzione: cosa succede se il segnale d'ingresso cambia due volte nello stesso clock? Noi assumeremo che il segnale di clock abbia frequenza tale da non far accedere queste cose (per il teorema di Shannon devo avere una frequenza di campionamento almeno doppia rispetto alla frequenza del segnale da campionare). Lo posso considerare come un riconoscitore di sequenza "01", come prima e tolgo un pezzo.



- 4Fonte: TestGroup

Attenzione: la soluzione per cui da reset vado a edge+ se mi arriva un 1 è sbagliata perché devo identificare un fronte di salita, cioè uno zero seguito da un uno.

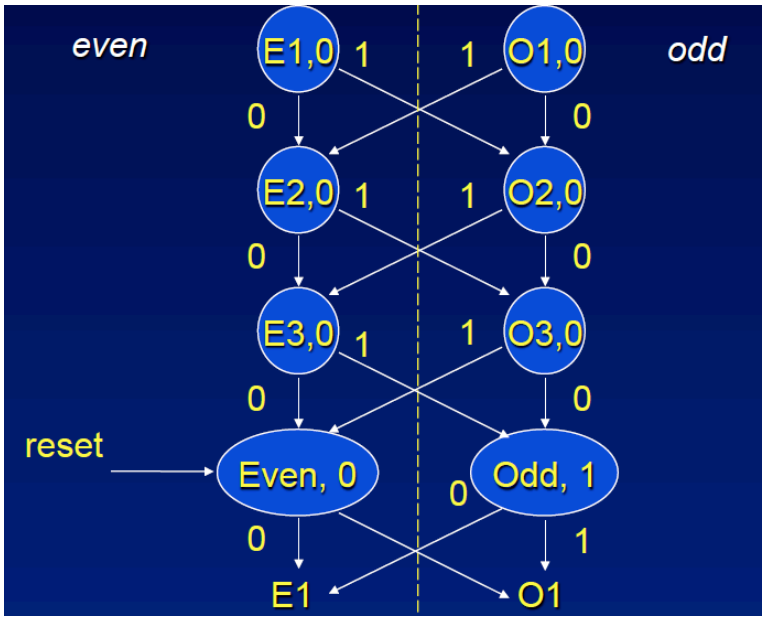
In questo caso è possibile conoscere a priori il numero di possibili stati. A me interessano l'ultimo e il

penultimo bit. Elenco tutti i possibili stati e poi li semplifico. Questo è un approccio a forza bruta che però a volte funziona.

Esercizio 3

"On a serial transmission line X, bits are transmitted synchronously w.r.t. a clock signal CLK, one bit per clock cycle. The line is used to transmit strings of 4 bits. A circuit to be connected to the serial line is to be designed. It has an output ODD, which is asserted, for 1 clock cycle, in correspondence of the 4th bit of each string, if the string itself contains an odd # of 1's. In correspondence of the other bits of the string, ODD must be '0'.(ODD->3 lettere->dispari, even->4 lettere ->pari)."

Hint: in casi di questi genere questo circuito è detto "jumping window", qui la finestra salta di 4 in 4.
Hint: essere ordinati! In particolare fare attenzione a non perdere il sincronismo.



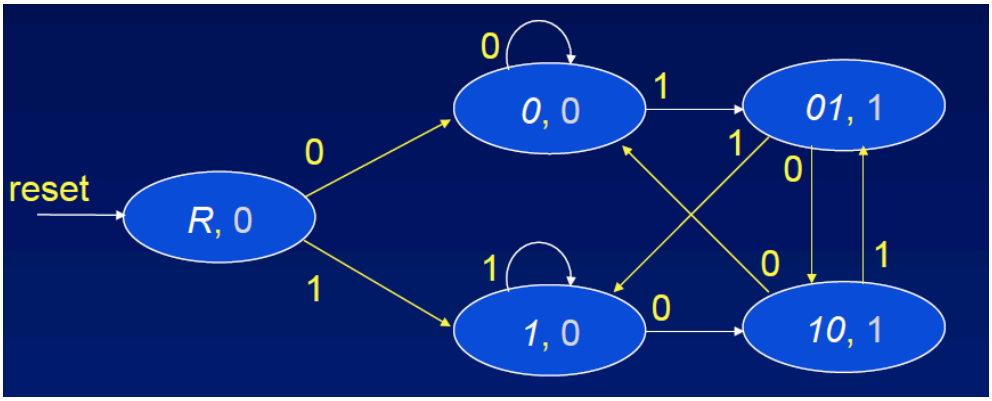
- 5Fonte: TestGroup

Considerazioni: in generale posso sempre costruirmi un albero binario completo (tutti gli stati possibili), ma è da suicida perché poi devo minimizzarlo. Devo capire la logica che ci sta dietro, in questo caso non devo contare il numero di bit, devo solo verificare se il numero di bit è pari o dispari. Non devo contare il numero di pari, devo solo guardare se pari o dispari. Al IV bit sono o in even o in odd. Dopo di che devo vedere se ci sono stati equivalenti. Se la stringa fosse formata da più bit, basta aumentare il numero di righe.

Esercizio 4

"A circuit is to be designed, having:
An input X
A clock signal CLK, which acts as a proper sampling signal of X, i.e., the frequency of CLK is such that it never happens that two transitions of X occur within a same CLK cycle
An output Z, asserted for a clock cycle whenever an edge (raising or falling, indifferently) on the input X is detected."

#soluzione1 oppure posso arrivarci distinguendo il comportamento transitorio da quello a regime.

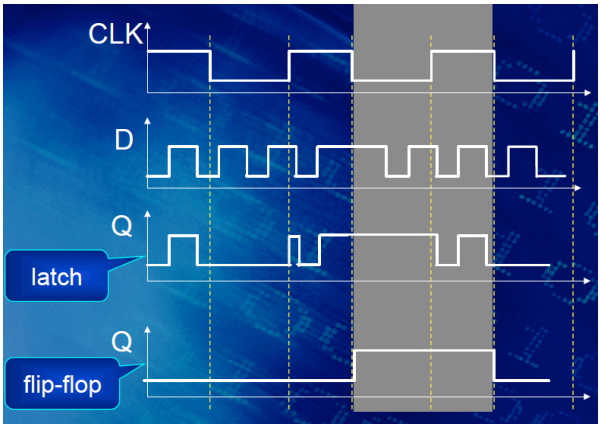


Esercizio 5

"On a serial transmission line X, bits are transmitted synchronously w.r.t. a clock signal CLK, one bit per clock cycle. The line is used to transmit groups of 4 bits: each group corresponding to a BCD digit, transmitted Most Significant Bit (MSB) first (Big-endian). A circuit to be connected to the serial line is to be designed. It has an output U which is asserted, for 1 clock cycle, in correspondence of the 4th bit of each group, if the group itself is a correct BCD digit."

Latches and Flip Flop

Per passare da STG a netlist, bisognerà utilizzare i flip-flop. Adesso facciamo panoramica dei vari flip-flop e latches, anche se poi in realtà ne utilizzeremo solo un tipo.



- 6 Latch vs FF. Fonte: TestGroup

Di fatto stiamo considerando i blocchetti elementari dei circuiti sequenziali (plus porte logiche). Sono dispositivi che memorizzano un bit d'informazione. Supponendo di avere un segnale di clock e un segnale di data, la differenza fra i due è che:

- **Flip-flop**: un evento associato al segnale di temporizzazione (in genere un fronte) fa in modo che il flip-flop vada a campionare l'ingresso e l'uscita rimane al valore campionato dell'ingresso per tutta la durata del clock. Qualunque cosa fa l'ingresso durante il periodo non è importante. Di fatto il flip-flop è una macchina di Moore.
- **Latches**: in inglese chiavistello, il periodo del clock è diviso in due dal suo duty cycle (in genere 50 e 50 ma non è detto), nelle due fasi del clock ve ne è una in cui il latch è aperto, è trasparente, qualunque variazione sull'ingresso viene riportata sull'uscita; nella seconda metà appena c'è la transizione fra aperto e chiuso si campiona il segnale in ingresso è quel valore rimane costante sull'uscita per tutta la seconda fase. In questo caso latches è una macchina di Mealy (dipende dai valori dell'ingresso).

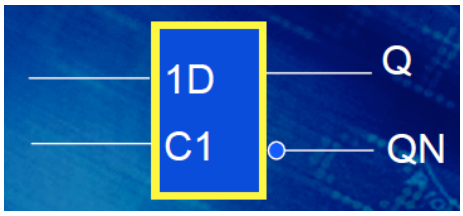
Da questo punto in poi latches e flip-flop sono due cose diverse.

Vantaggi e svantaggi: in genere uso i flip-flop e non i latch perché con i flip-flop riesco ad aumentare la collaudabilità (capire se a fine produzione un circuito sequenziale funziona correttamente è un'operazione molto costosa, se non me ne preoccupo in fase di progetto potrei non riuscire a farlo).

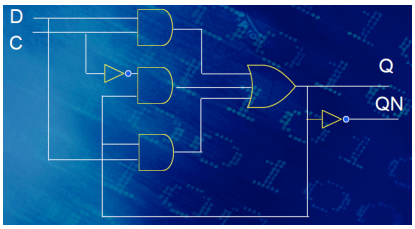
Latches

Ce ne sono 4 diversi tipi:

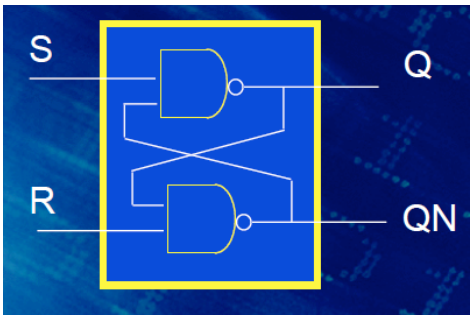
- **D LATCH**: D sta per delay, è un dispositivo con due uscite (Q e QN), segnale di controllo C1 (è il clock, ma si preferisce non chiamarlo così perché è più un controllo che un segnale di temporizzazione) e 1D (data). Per i circuiti sequenziali non ho le tabelle di verità ma le tabelle caratteristiche. Per convenzione Q-1 e QN-1 sono i valori di Q e QN precedenti. Il valore di QN è Q'. Se C è 1, seguo il segnale, se C è 0 il latch è chiuso.



- **IBM LSSD D latch**: l'IBM fin dall'inizio si è progettata e costruito in casa i componenti dei dispositivi che vendeva, fu la prima a capire che bisogna fare attenzione in fase di progetto alla collaudabilità (in genere se voglio collaudare devo pagare in area e in velocità). L'IBM introdusse per prima le design for testability (regole per la testabilità). Se rispetto queste regole riesco a collaudare il circuito. In pratica da sempre l'IBM utilizza questo latch come blocchetto elementare. Questo circuito è asincrono, la sincronicità viene garantita mettendone due in cascata con i clock invertiti (quando uno è aperto l'altro è chiuso). *Hazard-free* vuol dire che il circuito funzioni indipendentemente dai ritardi specifici delle singole porte.



- **SR latch**: non ha il segnale di controllo, Set-Reset latch, si può considerare come blocchetto elementare, ha due ingressi S e R e due uscite Q e QN (con QN=Q'). Su S e R ho 4 possibili combinazioni, di queste una forza Q a zero (reset), una forza Q a uno (set), la terza lascia le uscite invariate e la quarta è proibita. Che poi la configurazione sia 01 o 10 è un dettaglio implementativo, dipende dal componente che uso. Tipicamente può essere implementato come cross nand (avendo un elemento di reazione è sequenziale). Se provo a disegnare la mappa di Karnaugh trovo che la prima combinazione è proibita per due ragioni: se gli ingressi sono entrambi a zero le uscite sono entrambe a uno e non vale più che Q=QN'. Inoltre se gli ingressi vanno da 00 a 11, nella realtà può darsi che primi cambi S o prima cambia R (di sicuro non cambieranno mai contemporaneamente), e allora non so prevedere a priori in quale stato la mia macchina andrà a finire perché dipende dai ritardi interi. Per cui il costruttore non mi garantisce che il dispositivo funzioni correttamente. Analogamente per l'implementazione con le porte NOR.

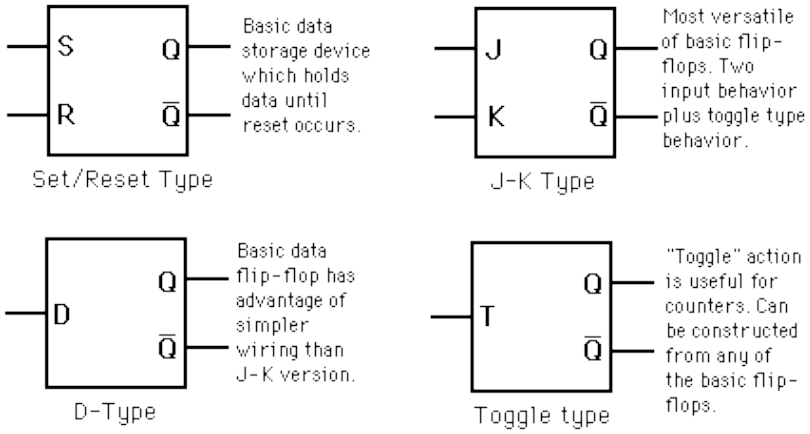


- **Gated set-reset latch**: è quello di prima dove davanti ho due porte con un segnale di controllo. In pratica se il segnale di controllo è a 0 il circuito è bloccato, altrimenti si comporta come prima.

Flip Flop

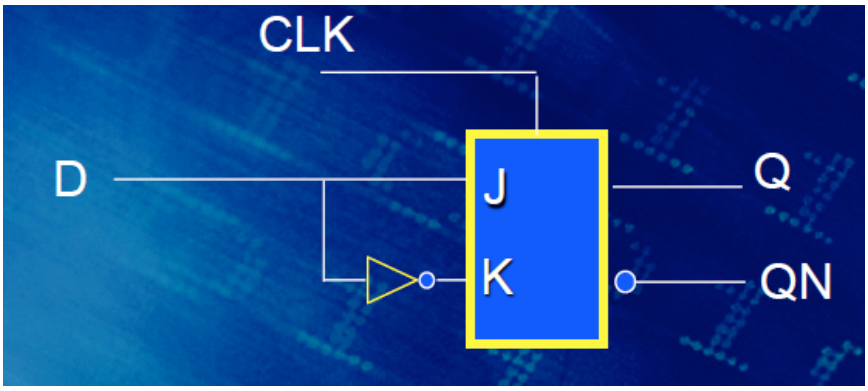
Si utilizza solo il tipo D.

- **D**: data e clock per gli ingressi, come uscite Q e QN. Ciò che ho in ingresso me lo trovo sull'uscita al colpo di clock successivo (per questo delay). *La tabella di transizione è una tabella in cui elenco i valori che devo fornire a D per forzare l'uscita al valore desiderato*. D può essere visto sia come segnale di data, sia come segnale di controllo (se D=0 then Q=0, else Q=1). In questo caso D lo posso utilizzare sia come dato sia come segnale di controllo.
- **SET-reset**: come per il latch.
- **JK**: su di un bit d'informazioni, quali sono le operazioni che posso fare sul singolo bit? Negarlo, mantenerlo, forzarlo a 0 o a 1. Questo flip-flop esegue proprio queste 4 operazioni: forzo l'uscita a 0 o 1, lo complemento o lo tengo invariato. Il JK è il flip-flop più completo. So che l'output vale 0, voglio che al clock successivo l'uscita sia ancora 0, quale valore devo dare all'ingresso per lasciare out a 0? O forzo sempre l'ingresso a 0, oppure se so che vale sempre 0 allora lo mantengo così com'è (don't care).
- **T**: T sta per rimbalzante (toggle), se T=0 l'uscita è invariata, se T=1 allora complemento l'uscita. Il problema è che così com'è all'accensione non riesco a inizializzarlo. In realtà supero il problema mettendo un segnale di reset.

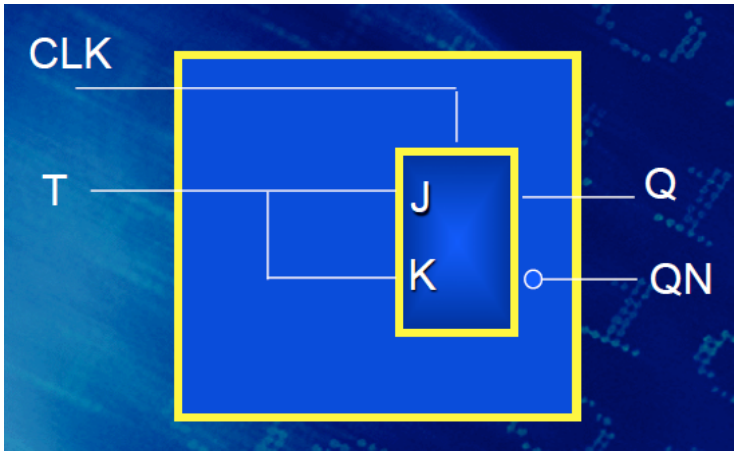


Fonte: http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/flipflop.html

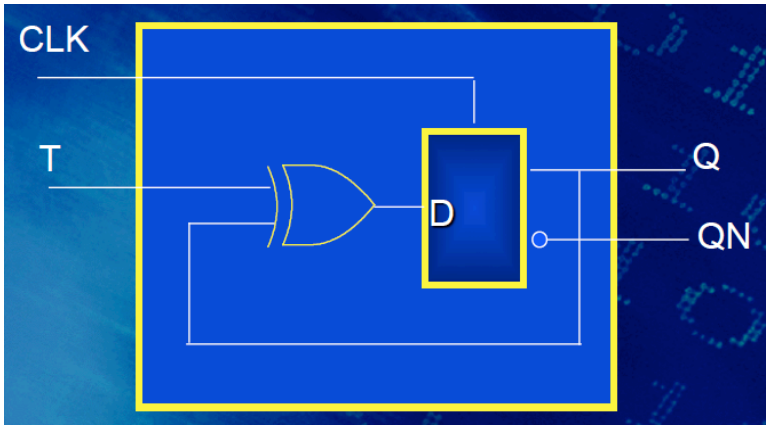
Exerc. 1: *avendo a diposizione un JK, come implemento un flip-flop di tipo D?*



Exerc. 2: *come implemento un T con un JK?*



Exerc. 3: *implementare un T con un tipo D?*

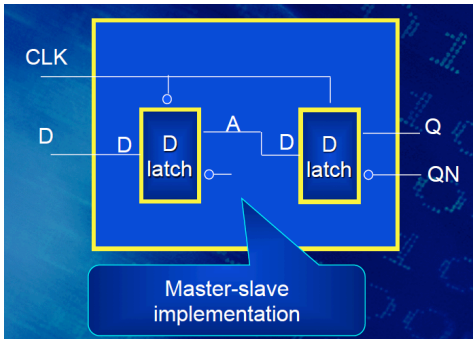


Ho un D dal quale escono Q e QN, dall'esterno ho un segnale d'ingresso T, posso farlo pensando di utilizzare un mux controllato da T, se T=0 carico il valore precedente con un loop su Q, se T=1 devo prendere il complementare e quindi prendo QN. Ok, posso semplificarlo? Se carico un valore o un suo negato allora di fatto è un exor (questo è un passo successivo).

Exerc. 4: *implementare un D con un T?* #todo da fare come esercizio.

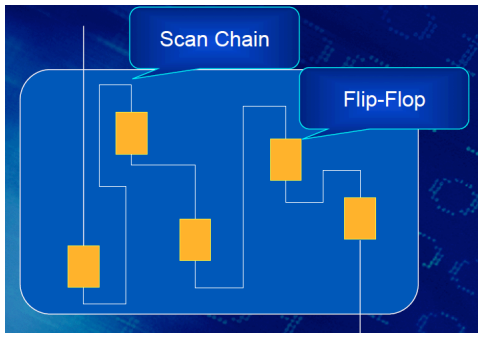
Exerc. 5: *come implementare un D flip-flop triggerato sul fronte positivo usando come blocchetti elementari i latch?*

Soluzione con due D latch in cascata con i clock in controfase (master e slave). Per passare da positive a negative trigger, basta cambiare su quale D latch metto il clock negato.



Il flip flop di tipo D che trovo in libreria è un po' più complicato, ho un reset asincrono, il clock enable, e altri due segnali di controllo che forzano il circuito a 1 o a 0 in modo sincrono.

Scannable flip-flops



- 7Fonte: TestGroup

Ho un circuito sequenziale con vari flip-flop, a livello di collaudo come faccio a forzare le uscite in modo semplice di ogni flip-flop a un valore specifico? È assai complicato perché non so cosa c'è fra i vari flip-flop. Quando sono in fase di collaudo posso riuscire ad avere tutti i flip-flop su di un'unica catena (scan chain), così in fase di test riesco agevolmente a forzare le uscite di tutti. Quindi nelle librerie trovo dei flip-flop con un ingresso in più, per forzare l'uscita del flip–flop in fase di test. Le scan chain sono OBBLIGATORIE per la testability, la regola è usare **solo flip-flop di tipo D e scannable**.

Quindi di fatto utilizzo solo quelli di tipo di D perché così con una sola catena riesco testarli in modo agevole, se invece usassi i JK è vero che avrei più potenze e flessibilità però i costi per la testability sarebbero più alti e sarebbe più complicato farlo.

RT-level sequential blocks

Qui più che sui combinatori guarderemo le caratteristiche fondamentali.

Nota: tutti i blocchi sequenziali saranno in grado di fare il reset sincrono e asincrono e avranno tutti un segnale di enable. Per convenzione il triangolino dentro la black box indica che quell'ingresso e un segnale di temporizzazione.

Registro: è il flip-flop su n-bit.

Contatore

- **UP-DOWN modulo m**: **a ogni colpo di clock, se abilitato, incrementa o decrementa di 1 il suo contenuto**. Poiché non ha una memoria infinita, indico con m il massimo valore che può assumere l'uscita, le uscite saranno comprese fra 0 e m-1 inclusi. Se sono a m-1 e vado avanti torno a 0 (in realtà sono ciclici). Mi serve un segnale di controllo per sapere se incrementare o decrementare (up o down), devo avere il reset, l'enable, il reset sincrono, inoltre non vorrei doverlo inizializzarlo sempre a 0 ma a un certo valore (un offset iniziale), quindi devo avere un segnale di controllo per caricare il valore degli ingressi in parallelo. Tutto questo sarà fatto con tanti flip-flop; in ogni istante posso o lo azzero, o lo caricare un valore iniziale non nullo (ingressi in parallelo) o incrementare/decrementare il suo contenuto. Queste 3 operazioni mutuamente esclusive. In genere la priorità (from top to bottom) è SYNC_RESET, LD_CNT (carica in parallelo) e UP_DN (incrementa o decrementa). In uscita ho anche un segnale di controllo chiamato TC (Terminal Count), che indica la fine del conteggio (se vado up me lo manda quando sono all'm-1, se conto all'indietro se sono a 0). Mi avvisa che al prossimo colpo di clock tornerò alle condizioni iniziali.

- **1-to-m up counter**: Analogo al up-down, l'unica differenza e' che il conteggio ha come estremi 1 e m. Quando arrivo alla fine l'uscita vale m, il TC e' attivo e il prossimo valore sara' 1.

Shift register

Registro a scorrimento, è una variante del normale registro, mi permette di scalare il suo contenuto verso destra o verso sinistra a seconda del segnale di controllo. Inoltre devo chiedere al mio circuito quale valore caricare nella cella che si è liberata. Anche qui ho dei dati in parallelo (qui o carico o sposto a destra o sinistra, non posso shiftare di 2 o più posizioni, solo 1 per volta). RT_LF_n m'indica se mi sposto a dx o sx, su SH_IN indico il valore che devo caricare nella cella libera.

Di questo c'è una versione semplificata, chiamata **serial shift register**, dove ho solo SH_IN e SH_OUT, dove non posso caricare tutti i dati in parallelo. La catena di scan-flip flop concettualmente è uno shift register.

Una versione più raffinata è il **barrel shifter**, a ogni colpo di clock non mi sposto di una posizione ma di n posizioni. Questa versione è abbastanza complicata. Se shifto di n bit devo poter dire quanto vale n.

Pulse generator

Lo abbiamo già progettato, rilevo un fronte del segnale d'ingresso e genero in uscita un segnale lungo quanto il colpo di clock. Serve a interfacciarsi verso qualcosa che è sincrono.

ROM

- **Single port**: ho il dato (D_IN) e l'indirizzo (ADDR), lo vedo come ***insieme di parole***, ciascuna identificata da un indirizzo. Ogni parola è lunga n bit, ***leggo o scrivo tutta la parola***. **Cella: singolo bit, parola: insieme di cello tutte allo stesso indirizzo**. Ho due segnali di controllo (WR_EN e MEM_EN), se devo leggere non m'interessano gli ingressi, se devo scrivere le uscite possono essere qualunque.

- **Dual port**: versione più sofisticata, ho una sola batterie di parole ma ho due porte diverse per accedere (A e B, indipendenti fra di loro). Ciascuna porta ha il suo D_IN, il CLK, D_OUT, WR_EN e ADDR. È come se avessi due utenti diversi, cosa succede se entrambi vogliono accedere alla stessa parola? Se entrambi leggono non capita nulla, se entrambi scrivono o uno scrive e uno legge? In genere il costruttore in questi casi non mi garantisce nulla, sono io che lo sto utilizzando in modo sbagliato. In genere viene usata per scambiare dati fra il produttore e il consumatore (#todo disegnino). Ci sono anche dispositivi di memoria più complicati, ad esempio a n-porte e di tipo FIFO (chi scrive scrive sempre dalla stessa parte, chi legge lo fa sempre dalla stessa parte). Questi ultimi però non sono considerati come elementari.

Cenni alle specifiche dell'homework.

Devo realizzare una piattaforma software che permette di giocare a un numero pari di giocatori. Ogni match fra due giocatori è composto da 3 round. Di fatto si gioca a coppe. Ogni giocatore prende 7 carte in modo random dal mazzo principale, ogni mazzo di carte principale è replicato per ogni giocatore. Ogni giocatore può schierare al massimo 5 carte per match, le carte schierate non le posso usare nei successivi round. Nel match successivo il giocatore ha di nuovo a disposizione le carte estratte.

Ogni carta è caratterizzata da un ID (numero univoco nel mazzo principale). Se la carte è allenabile la posso sostituire con un'altra carta, il valore indica quanto mi costa comprarla. Le carte non si possono vendere, posso solo allenarle. Non posso né fari scambi con altri giocatori né vender carte (funzioni extra, non necessarie). Non posso allenare due volte la stessa carta.