Sistemas Digitais I

LESI :: 2º ano

# Conceitos sobre Sistemas Sequenciais

**António Joaquim Esteves**

João Miguel Fernandes

**www.di.uminho.pt/~aje**

Bibliografia: secções 7.1, 7.2, 7.4, 7.5, 7.12, 9.2, DDPP, Wakerly

DEP. DE INFORMÁTICA

ESCOLA DE ENGENHARIA

UNIVERSIDADE DO MINHO

# 6. Conceitos sobre Circuitos Sequenciais

*- Sumário -*

- Circuitos combinacionais vs sequenciais
- Estado
- Elementos bi-estáveis
- *Latches* e *flip-flops*
- Projecto de máquinas de estados
- Máquinas de estados em VHDL

# 6. Conceitos sobre Circuitos Sequenciais

*- Circuitos Combinacionais vs Sequenciais -*

- Os circuitos lógicos podem ser classificados de combinacionais ou sequenciais.
- Um circuito combinacional é aquele em que as saídas só dependem das entradas actuais. Exemplo: comando com botão para escolher o canal da TV.
- Um circuito sequencial é aquele em que as saídas dependem das entradas actuais, mas também da sequência de valores por que passaram as entradas. Exemplo: comando para escolher o canal da TV com um botão para ir para o canal próximo/anterior (botão "+/-").
- Não é possível descrever o comportamento dum circuito sequencial simplesmente com uma tabela que relacione as entradas com as saídas.
- Para saber para onde vai evoluir um circuito sequencial, é preciso conhecer em que situação ele se encontra actualmente.
- Ou seja, o estado desse circuito deve ser memorizado.

# 6. Conceitos sobre Circuitos Sequenciais

*- Estado (1) -*

- O estado dum circuito sequencial é o conjunto de variáveis de estado, que guarda a informação relativa ao passado/presente desse circuito, necessária para determinar o seu comportamento futuro.
- No exemplo do comando para escolher o canal da TV, o número do canal actual é o estado actual.
- Conhecido o estado actual, pode sempre prever-se o próximo estado em função das entradas actuais.
- Num circuito digital, as variáveis de estado são valores binários e correspondem a sinais internos desse circuito.
- Um circuito com n variáveis de estado binárias pode ter até $2^n$ estados.
- Um circuito sequencial também pode ser designado de máquina de estados finita (ou seja, máquina com um número de estados finito).

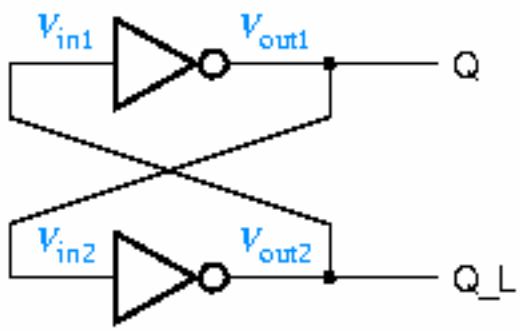
# 6. Conceitos sobre Circuitos Sequenciais

*- Estado (2) -*

- As mudanças de estado acontecem em instantes impostos por um **sinal de relógio**.
- Um sinal de relógio é activo no nível alto se as mudanças de estado acontecerem no bordo ascendente do relógio ou quando o relógio está no nível ALTO. Caso contrário, é activo no nível baixo.
- O periodo do relógio (T) coincide com o intervalo entre transições sucessivas (do relógio) na mesma direcção.
- A frequência do relógio (f) é o inverso do periodo do relógio (f = 1 / T).
- Neste módulo abordam-se dois tipos de circuitos sequenciais:
  - Circuitos sequenciais simples*:* usam portas lógicas elementares (ANDs, ORs,…) e ciclos (ou caminhos) com *feedback* para criar elementos de memória (***latches*** e ***flip-flops***).
  - Máquinas de estados síncronas com um sinal relógio: usam *latches* e *flip-flops* para criar circuitos que funcionam sob o controlo dum sinal de relógio.

# 6. Conceitos sobre Circuitos Sequenciais

*- Elementos bi-estáveis (1) -*

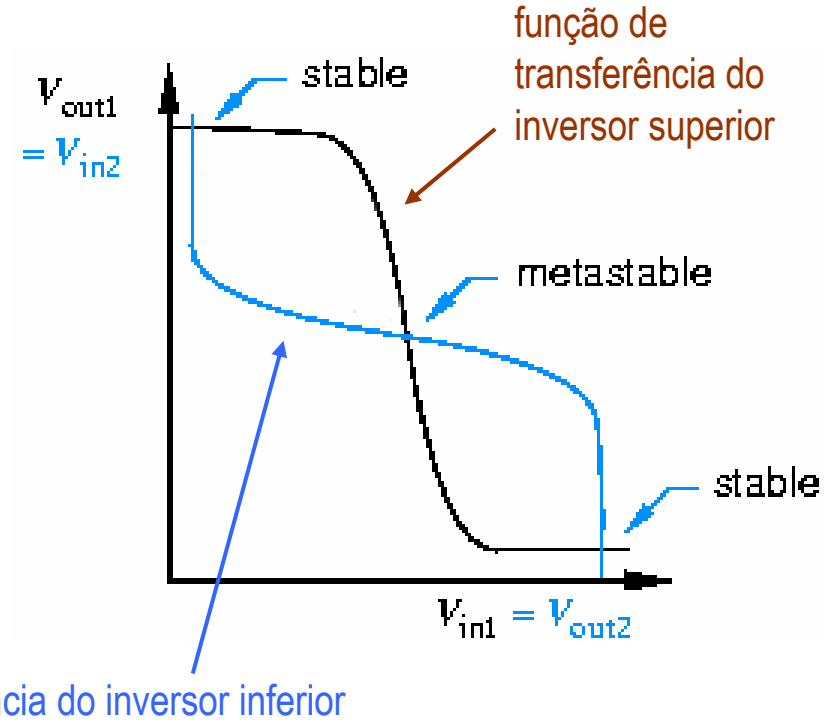



- O circuito sequencial mais simples de todos é um circuito sem entradas e construído com um par de inversors interligados de modo a estabelecer um ciclo com *feedback*.
- A este circuito dá-se o nome de **bi-estável** porque possui dois estados (situações) estáveis:
  - Quando Q está no nível ALTO, o inversor inferior tem a saída no nível BAIXO, forçando deste modo o inversor superior a colocar a sua saída no nível ALTO (como se assumiu inicialmente).
  - Quando Q está no nível BAIXO, o inversor inferior tem a saída no nível ALTO, forçando deste modo o inversor superior a colocar a sua saída no nível BAIXO (como se assumiu inicialmente).
- Pode usar-se uma única variável de estado (sinal Q) para definir o estado do circuito. Logo, há 2 estados possíveis: Q=0 e Q=1.

# 6. Conceitos sobre Circuitos Sequenciais

*- Elementos bi-estáveis (2) -*

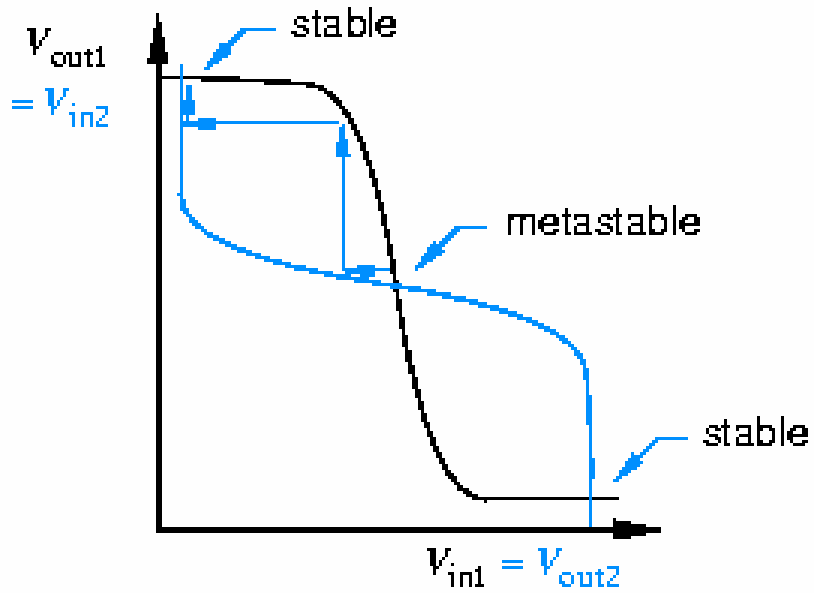
- O elemento bi-estável é tão simples que não possui entradas, o que impede que o seu estado seja controlado.
- Quando o circuito é alimentado, ele assume um estado aleatório e permanece nele indefinidamente.
- Efectuando uma análise do bi-estável segundo uma perspectiva analógica percebe-se melhor o seu funcionamento.
- O bi-estável está em equilíbrio se as tensões de entrada e de saída em ambos os inversors assumirem um valor constante e consistente com (i) as ligações do ciclo de *feedback* e (ii) a função de transferência dos inversores.

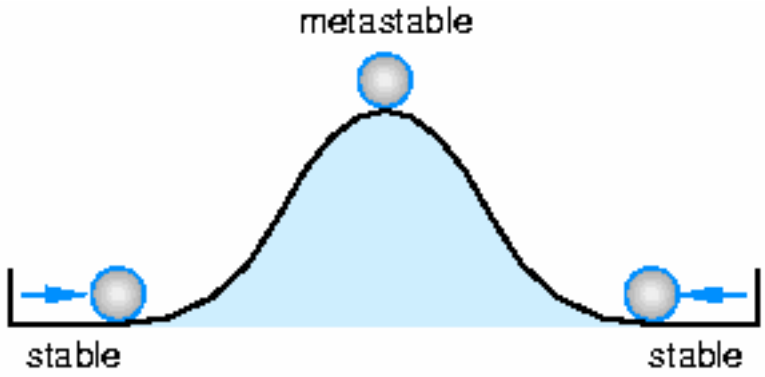


# 6. Conceitos sobre Circuitos Sequenciais

*- Elementos bi-estáveis (3) -*

- O bi-estável está em equilibrio nas posições assinaladas com "stable".
- Há um 3º ponto de equilíbrio, assinalado com "metastable", que ocorre quando $V_{out1}$ e $V_{out2}$ não são nem 0 nem 1 lógico.
- Se não houvesse ruído e o circuito atingisse o ponto meta-estável, poderia permanecer nele indefinidamente.
- O ponto é meta-estável porque o ruído tenderá a levar o circuito para uma das posições estáveis.
- Analogia do ponto de meta-estabilidade com uma bola lançada sobre o pico duma montanha →



# 6. Conceitos sobre Circuitos Sequenciais

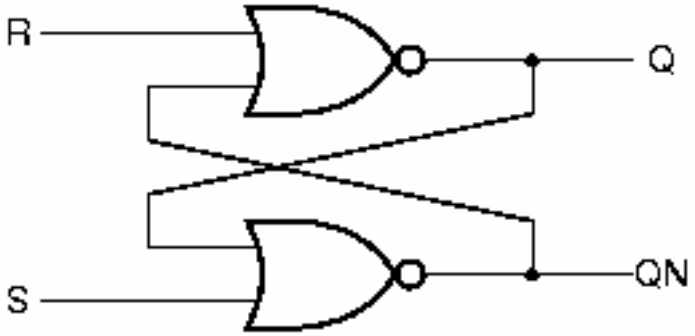
*- Latches e Flip-flops (1) -*

- As *latches* e os *flips-flops* são os blocos elementares com os quais se constroi a maior parte dos circuitos sequenciais.
- Um *flip-flop* é um dispositivo sequencial que amostra as suas entradas e que altera as suas saídas apenas em instantes determinados por um sinal de relógio.
- Uma *latch* é um dispositivo sequencial que observa todas as suas entradas continuamente e altera as suas saídas em qualquer momento, independentemente de qualquer sinal de relógio.

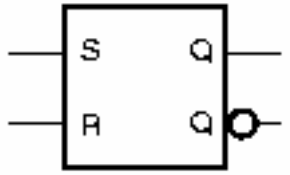
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (2) -*

- Pode construir-se uma *latch* S-R com portas NOR (S=set, R=reset) .
- Normalmente QN é o complemento de Q.
- Se S e R estiverem ambos a 0, o circuito comporta-se como o elemento bi-estável.
- Deve activar-se S ou R para forçar o ciclo de *feedback* a atingir o estado (estável) desejado.
- A entrada S define (*sets* ou *presets*) a saída Q a 1.
- A entrada R define (*resets* ou *clears*) a saída Q a 0.




| S | R | Q | QN |
|---|---|---|---|
| 0 | 0 | last Q | last QN |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

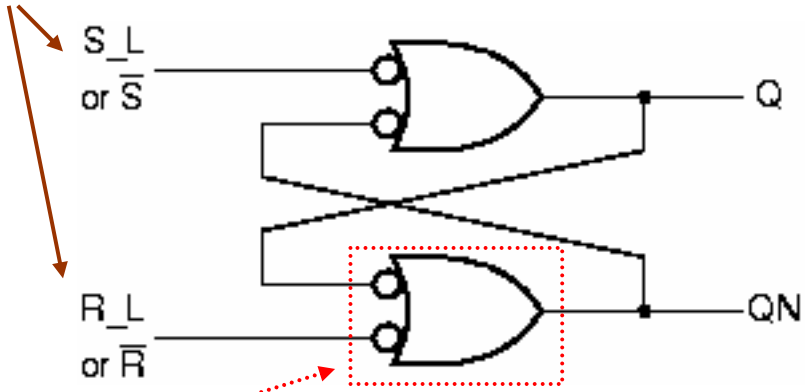


# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (3) -*

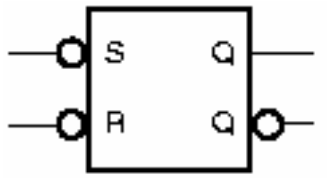
- Pode construir-se uma *latch* S-R, com entradas de *set* e *reset* activas no nível baixo, com portas NAND.
- O funcionamento desta *latch* $\overline{S}$-$\overline{R}$ é idêntico ao da anterior, com 2 diferenças:
  - S_L e R_L são activas no nível baixo, logo a *latch* mantém o seu estado quando S_L=R_L=1.
  - Quando S_L e R_L estiverem ambas activas, ambas as saídas ficam a 1 (e não a 0).

Em relação à latch S-R, /S e /R trocaram de posição




⇔ NAND

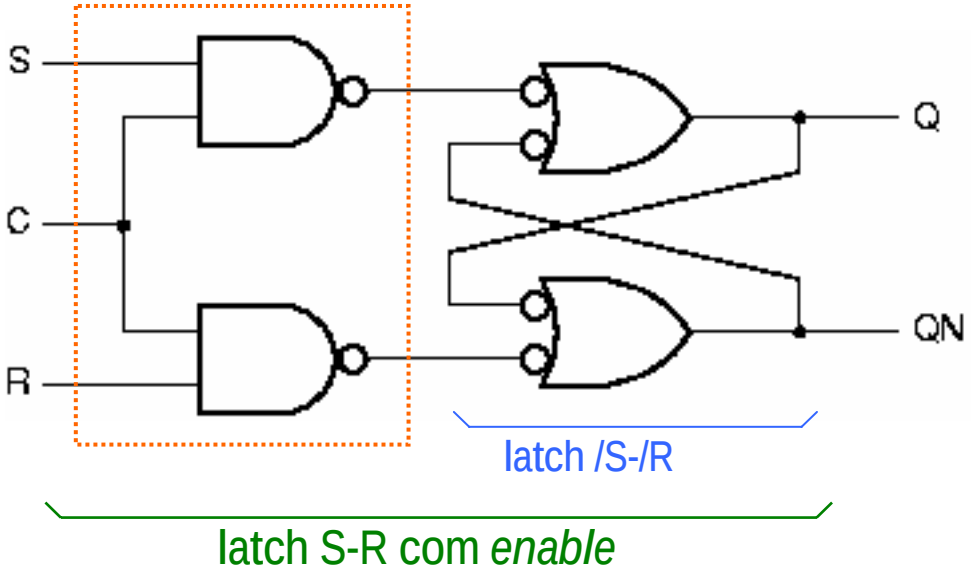
| S_L | R_L | Q | QN |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | last Q | last QN |



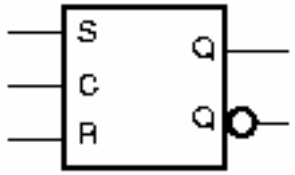
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (4) -*

- Uma *latch* S-R (ou /S-/R) reage ao valor das entradas em qualquer momento.
- Contudo, pode ser alterada para reagir ao valor das entradas apenas quando uma entrada de *enable* (C) estiver activa.
- O circuito alterado comporta-se tal como a *latch* S-R quando C=1.
- E mantém o estado quando C=0.

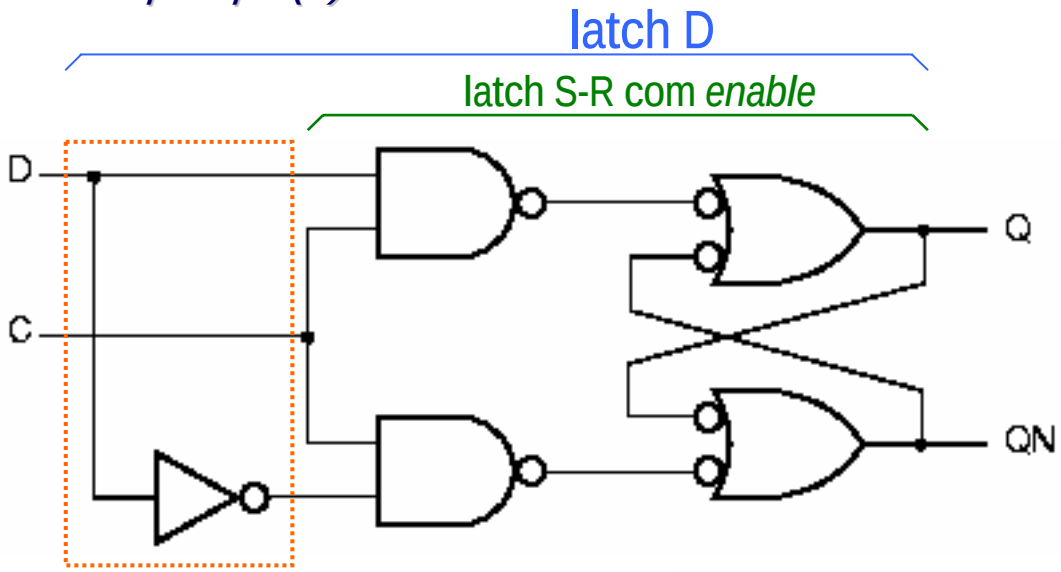



| S | R | C | Q | QN |
|---|---|---|---|---|
| 0 | 0 | 1 | last Q | last QN |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| x | x | 0 | last Q | last QN |



# 6. Conceitos sobre Circuitos Sequenciais

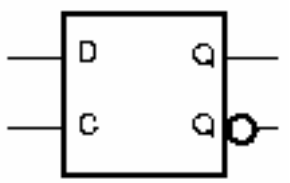
*- Latches e Flip-flops (5) -*




- Quando a finalidade da utilização duma *latch* é guardar um bit de informação, a **latch D** é a mais recomendada.
- Pode construir-se uma *latch* D a partir duma *latch* S-R.

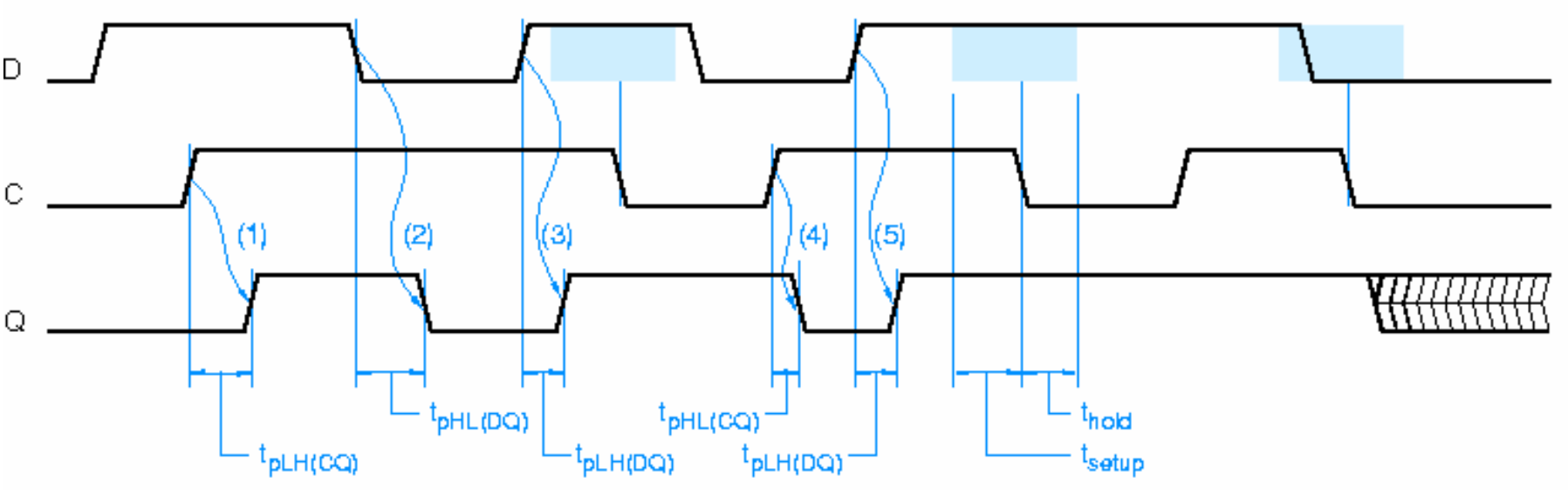
| C | D | Q | QN |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 0 | x | last Q | last QN |

- Esta *latch* elimina a situação problemática da *latch* S-R, que ocorre quando S e R são activadas (a 1) em simultâneo.
- Quando C=1, a *latch* está aberta / transparente e a saída Q acompanha a entrada D. Quando C=0, a *latch* está fechada e a saída Q mantém o último valor.



# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (6) -*

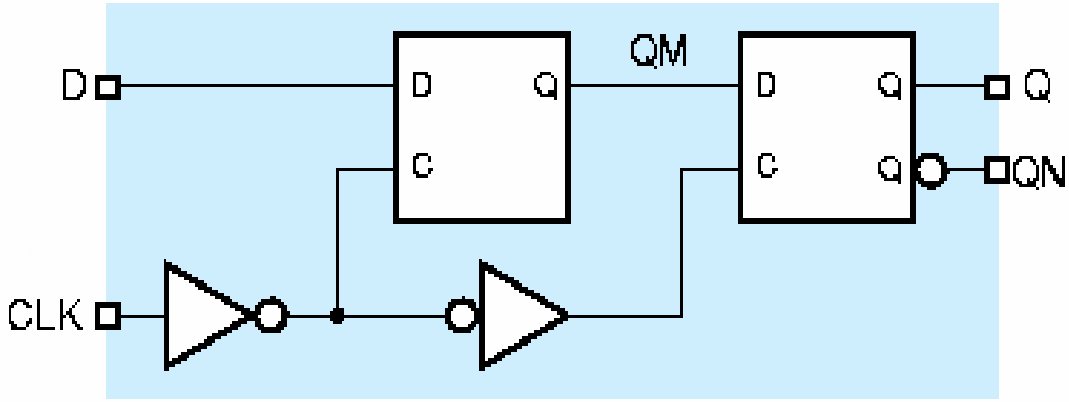



- Há um <u>atraso</u> associado com a propagação dos sinais desde as entradas até à saída Q.
- No intervalo definido pelos ***<u>setup time</u>*** mais o ***<u>hold time</u>***, em torno do bordo descendente de C, a entrada D deve permanecer fixa.
- Se estes 2 tempos não forem respeitados, a saída da *latch* assumirá um valor imprevisível.

# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (7) -*

- Um **flip-flop D positive-edge-triggered** *(FF D sensível à transição positiva)* é um circuito contruído com um par de *latches* D que amostra a entrada D e altera as saídas Q e QN apenas no bordo ascendente do sinal CLK.



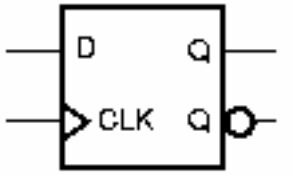

| D | CLK | Q | QN |
|---|---|---|---|
| 0 | ↑ | 0 | 1 |
| 1 | ↑ | 1 | 0 |
| x | 0 | last Q | last QN |
| x | 1 | last Q | last QN |

- A primeira *latch* chama-se mestre e está aberta quando CLK=0.
- Quando CLK muda para 1, a *latch* mestre fecha.
- A segunda *latch*, o escravo, está aberta enquanto CLK=1, mas a saída muda de valor apenas no início desse intervalo, dado que o mestre está fechado nesse intervalo.

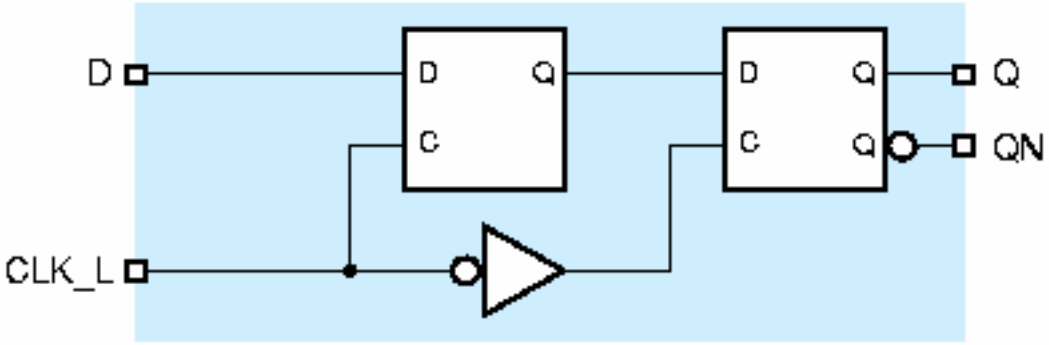
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (8) -*

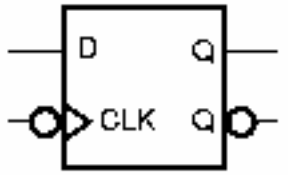
- O triângulo na entrada CLK é um indicador de entrada dinâmica e assinala um comportamento sensível às transições (*edge-triggered*).




- Num flip-flop D negative-edge-triggered *(FF D sensível à transição negativa)* inverte-se a entrada de relógio e a entrada D (as saídas Q e QN) passam a ser amostradas (alteradas) no bordo descendente do CLK.



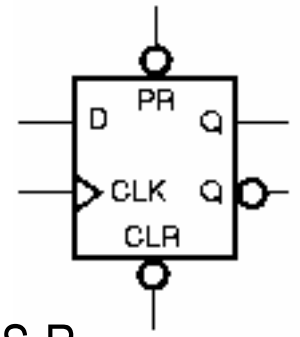

| D | CLK_L | Q | QN |
|---|---|---|---|
| 0 | ↓ | 0 | 1 |
| 1 | ↓ | 1 | 0 |
| x | 0 | last Q | last QN |
| x | 1 | last Q | last QN |



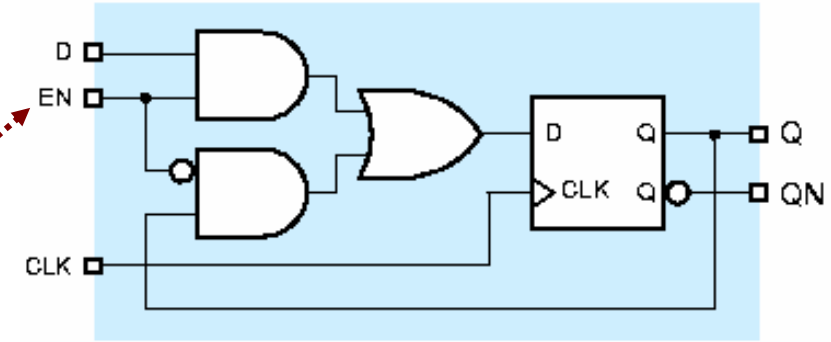

- No intervalo definido pelos *setup time* mais o *hold time*, em torno dos bordos do CLK a que o FF é sensível, a entrada D deve permanecer fixa.

# 6. Conceitos sobre Circuitos Sequenciais

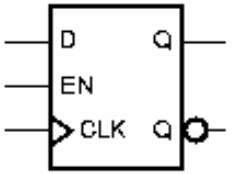
*- Latches e Flip-flops (9) -*




- Alguns *flip-flops* D possuem 2 entradas assíncronas que servem para forçar o seu estado, independentemente das entradas CLK e D → PR e CLR
- Estas entradas têm a mesma funcionalidade do set (S) e do reset (R) da *latch* S-R.
- As entradas assíncronas devem ser usadas nas fases de inicialização e teste dos circuitos.
- Alguns *flip-flops* D têm ainda a possibilidade de manter inalterado o último valor por que passou a saída. Para esse fim adiciona-se ao FF D uma entrada de *enable*.




| D | EN | CLK | Q | QN |
|---|---|---|---|---|
| 0 | 1 | ↑ | 0 | 1 |
| 1 | 1 | ↑ | 1 | 0 |
| x | 0 | ↑ | last Q | last QN |
| x | x | 0 | last Q | last QN |
| x | x | 1 | last Q | last QN |

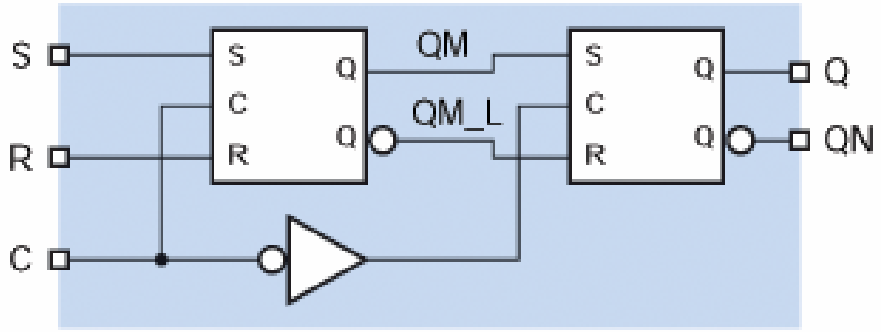



porque não se aplica o *enable* no CLK, usando-se assim apenas um AND ?

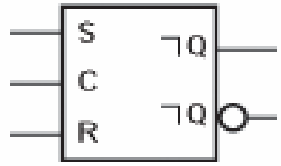
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (10) -*

- As *latches* S-R são úteis em sistemas de controlo, em que é comum ter condições independentes para colocar bits de controlo a 0/1.
- Quando se pretende que um dado bit de controlo seja apenas alterado em certos instantes determinados por um sinal de relógio, então exige-se um ***flip-flop*** S-R.




| S | R | C | Q | QN |
|---|---|---|---|---|
| x | x | 0 | last Q | last QN |
| 0 | 0 | ⊓ | last Q | last QN |
| 0 | 1 | ⊓ | 0 | 1 |
| 1 | 0 | ⊓ | 1 | 0 |
| 1 | 1 | ⊓ | undef. | undef. |

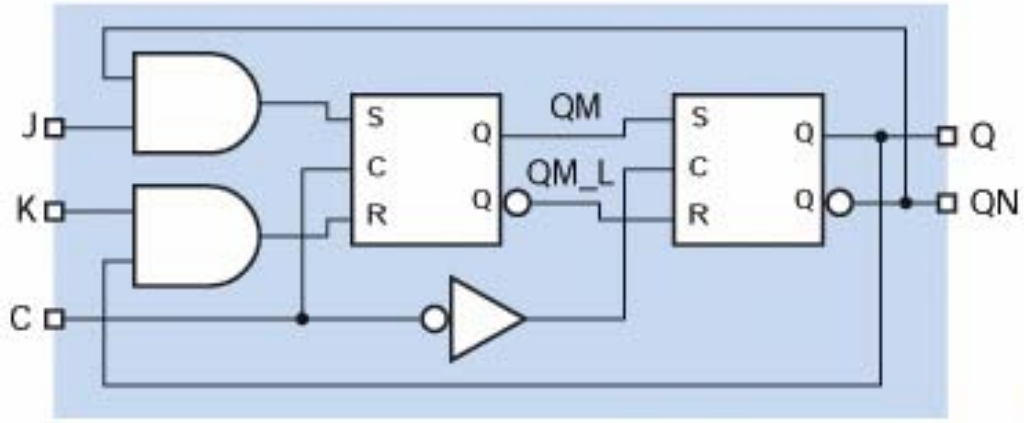



- estrutura mestre/escravo
- não é verdadeiramente sensível à transição
- o último valor guardado em QM (com C=1) só passa para a saída Q quando C mudar 1→ 0
- se S=R=1, antes de C passar 1→0, o valor da saída é imprevisível

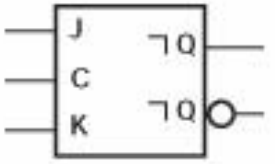
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (11) -*

- O problema que ocorre quando S e R estão ambos a 1 é resolvido no **flip-flop J-K com estrutura mestre/escravo**.
- As entradas J e K são analogas a S e R.
- No entanto, activar J só activa a entrada S da *latch* mestre se Q=0 (QN=1).
- Activar K só activa a entrada R da *latch* mestre se Q=1.
- Deste modo, se as entradas J e K forem activadas simultaneamente, as saídas do *flip-flop* mudam para o estado oposto do actual.



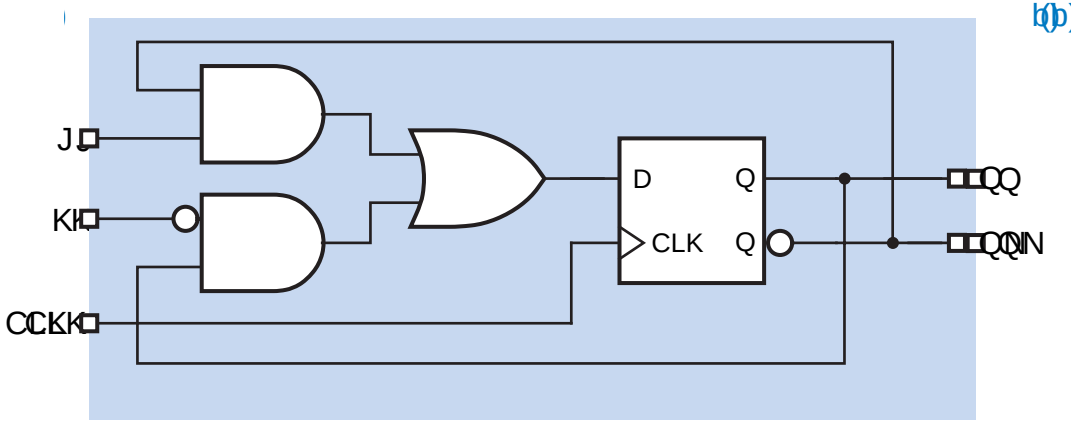

| J | K | C | Q | QN |
|---|---|---|---|---|
| x | x | 0 | last Q | last QN |
| 0 | 0 | ⊓ | last Q | last QN |
| 0 | 1 | ⊓ | 0 | 1 |
| 1 | 0 | ⊓ | 1 | 0 |
| 1 | 1 | ⊓ | last QN | last Q |



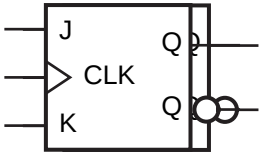
# 6. Conceitos sobre Circuitos Sequenciais

*- Latches e Flip-flops (12) -*

- Com um *flip-flop* D *edge-triggered* pode construir-se um ***flip-flop J-K edge-triggered*** *(sensível às transições)*.
- O *flip-flop* JK amostra o valor das entradas (J e K) e altera as saídas (Q e QN) no bordo ascendente do CLK de acordo com a equação característica: $Q^* = J \cdot Q' + K' \cdot Q$
- No intervalo definido pelos *setup time* mais o *hold time*, em torno do bordo ascendente de CLK, as entradas J e K devem permanecer fixas.
- O *flip-flop* JK tem aplicação comum em máquinas de estado porque gera menos lógica combinacional.



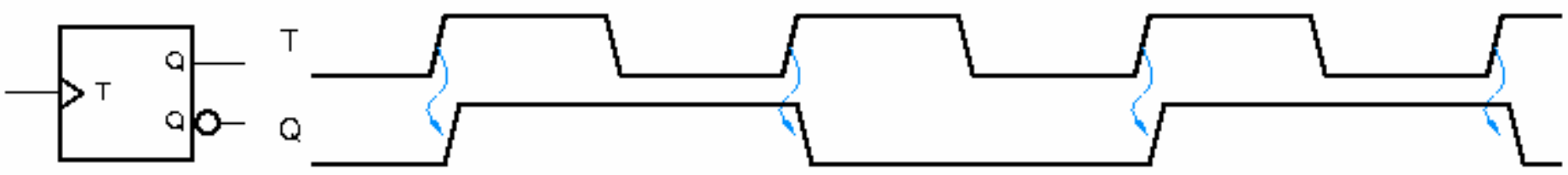

| J | K | CLK | Q | QN |
|---|---|---|---|---|
| x | x | 0 | last Q | last QN |
| x | x | 1 | last Q | last QN |
| 0 | 0 | ↑ | last Q | last QN |
| 0 | 1 | ↑ | 0 | 1 |
| 1 | 0 | ↑ | 1 | 0 |
| 1 | 1 | ↑ | last QN | last Q |



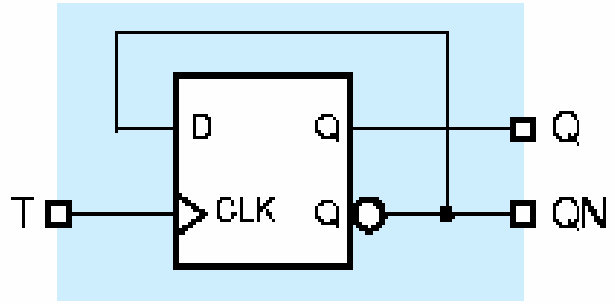
# 6. Conceitos sobre Circuitos Sequenciais

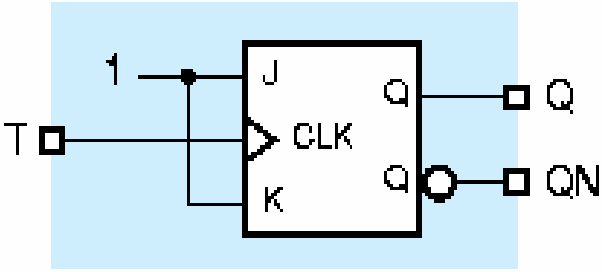
*- Latches e Flip-flops (13) -*

- O ***flip-flop*** T (de *toggle*) muda de estado em cada transição 0→1 do sinal de relógio.




- A saída Q do *flip-flop* possui uma frequência que é metade da frequência da entrada T.
- Pode usar-se um *flip-flop* D ou um J-K para construir o *flip-flop* T.

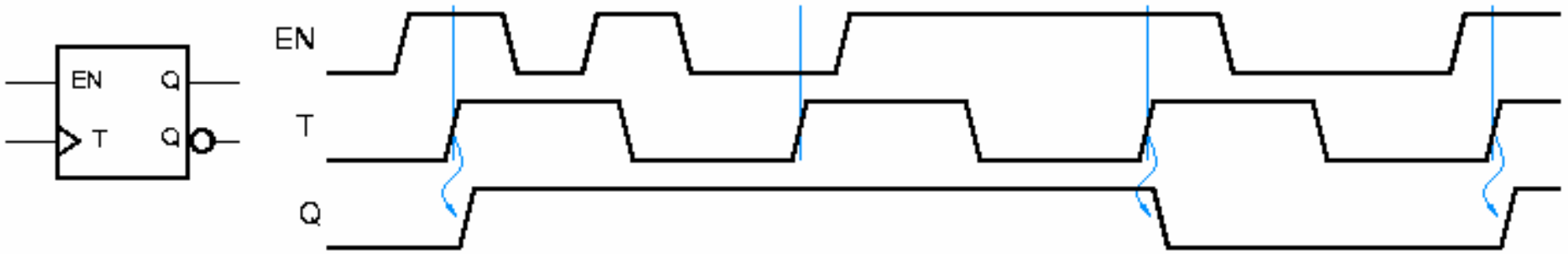


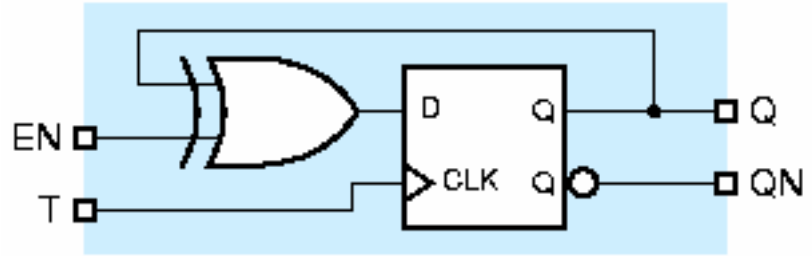
# 6. Conceitos sobre Circuitos Sequenciais

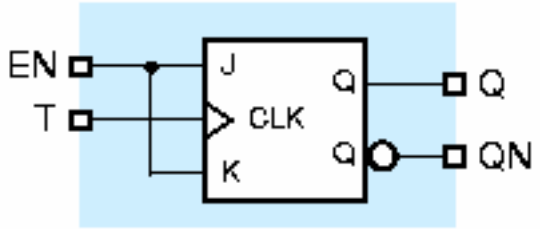
*- Latches e Flip-flops (14) -*

- O *flip-flop* T também pode ter uma entrada de *enable*.
- Neste caso, o *flip-flop* só muda de estado no bordo ascendente do relógio (T) se a entrada de *enable* EN estiver activa.




- Também se pode usar um *flip-flop* D e J-K para obter um *flip-flop* T com *enable*.



# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (1) -*

- Uma máquina de estados finita (FSM) pode ser definida formalmente por um quinteto **<S,I,O,F,G>**, em que:
  - **S** representa o conjunto de estados.
  - **I** representa o conjunto de entradas.
  - **O** representa o conjunto de saídas.
  - **F** representa a função que gera o próximo estado.
  - **G** representa a função que gera as saídas.
- A função F atribui a cada combinação (estado, entradas) outro estado: **F:SxI→S**.
- A função G obtém o valor das saídas para o estado actual.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (2) -*

- Existem 2 tipos de FSM, aos quais correspondem 2 definições diferentes para a função G que gera as saídas.
- Nas máquinas do tipo Moore, a função G baseia-se no estado: $\mathbf{G:S \rightarrow O}$.

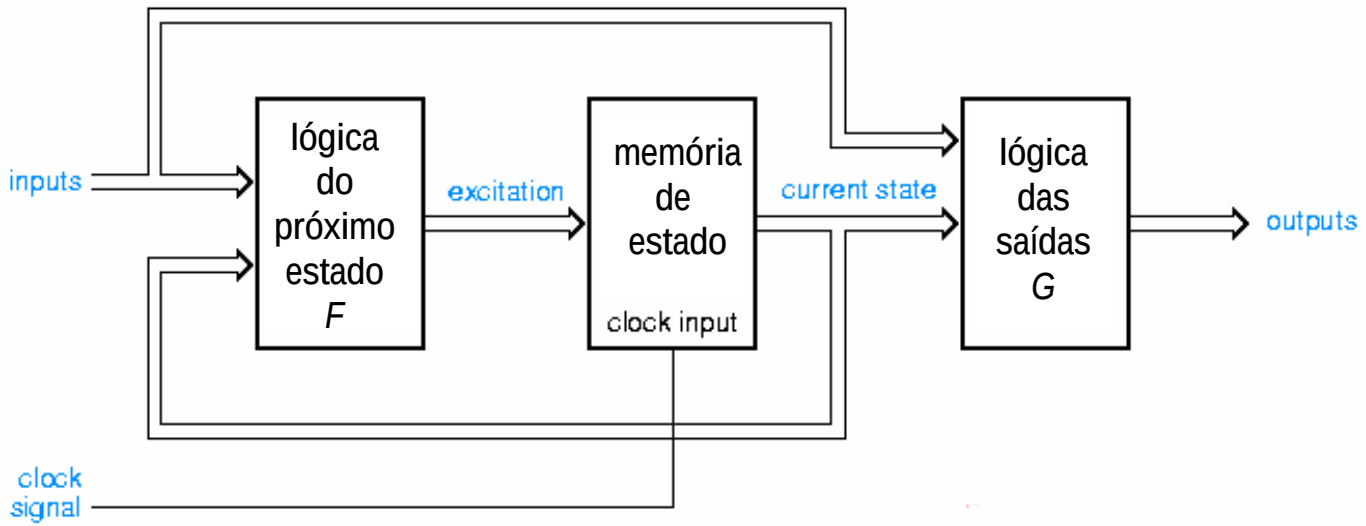
  As saídas dependem apenas do estado da FSM.
- Nas máquinas do tipo Mealy, a função G baseia-se nas entradas e estado: $\mathbf{G:S \times I \rightarrow O}$.

  As saídas dependem do estado e das entradas da FSM.
- O meta-modelo FSM assume que o tempo está dividido em intervalos uniformes e que as transições ocorrem apenas no início de cada intervalo.
- Para definir esses intervalos, a que se chama ciclos de relógio, usa-se um sinal de relógio.
- Cada modelo FSM pode ser implementado com *flip-flops* e portas lógicas.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (3) -*

- Genericamente, uma máquina de estados síncrona e do tipo Mealy possui a seguinte estrutura:

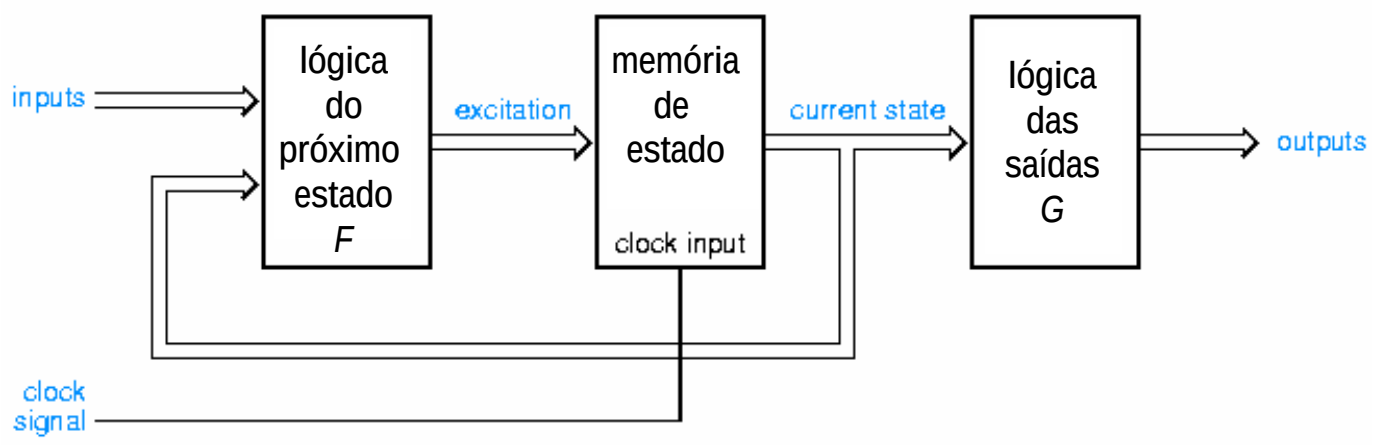



- A **memória de estado** é implementada com um conjunto de *flip-flops* que guarda o estado actual da máquina. Os *flip-flops* partilham o sinal de relógio.
- As funções F e G são circuitos estritamente combinacionais.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (4) -*

- Genericamente, uma máquina de estados síncrona e do tipo Moore possui a seguinte estrutura:




- A única diferença entre as máquinas de Mealy e de Moore reside na forma como as saídas são geradas.
- Regra geral, uma máquina de Mealy tem algumas saídas de Mealy e outras de Moore.
- Para que a máquina seja o mais rápida possível, o bloco G tem que ser o mais simples possível (apenas fios). Para conseguir este objectivo, usam-se variáveis de estado que coincidem com as saídas.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (5) -*

- Os passos envolvidos no projecto duma máquina de estados síncrona são:
  - Analisar a descrição ou especificação em linguagem natural do sistema.
  - Desenhar um diagrama de estados, usando nomes simbólicos para os estados.
  - Construir a tabela de estados e saídas.
  - [opcional] Minimizar o número de estados da tabela.
  - Escolher um conjunto de variáveis de estado e atribuir uma combinação (dessas variáveis) a cada estado.
  - Substituir na tabela o nome dos estados pela combinação (de variáveis) que lhe corresponde.
  - Escolher um tipo de *flip-flop* para a memória de estado.
  - Construir uma tabela de excitação que mostre quais os valores a aplicar na entrada dos *flip-flops* de modo a obter o próximo estado desejado, para cada combinação de estados e entradas.
  - Obter a expressão para cada saída do bloco do próximo estado (excitação dos F/Fs).
  - Obter a expressão para cada saída da FSM.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (6) -*

- Exemplo duma máquina de estados:
  Projectar uma máquina de estados que possui como entradas A e B e como saída Z. A saída Z é 1 se:
  - A apresentar o mesmo valor nos 2 ciclos de relógio anteriores, ou
  - B estiver fixo a 1 desde a última vez que a condição anterior se verificou.

  Nos outros casos, a saída Z é 0.

- Nesta fase, a especificação pode parecer pouco clara.
- O projectista deve transformar uma especificação ambigua, escrita em linguagem natural, numa tabela de estados sem ambiguidades.
- A máquina é do tipo Moore, uma vez que as saídas só dependem do estado actual, ou seja, dependem apenas do que ocorreu nos cliclos de relógio anteriores.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (7) -*

- Processo de construção da tabela de estados e saídas (S=estado actual, S*=próx.estado):

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | | | | | 0 |
| | . . . | | | | | |
| | . . . | | | | | |
| | . . . | | | | | |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | | | | | 0 |
| Got a 1 on A | A1 | | | | | 0 |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK | OK | A1 | A1 | 0 |
| Got a 1 on A | A1 | | | | | 0 |
| Got two equal A inputs | OK | | | | | 1 |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK | OK | A1 | A1 | 0 |
| Got a 1 on A | A1 | A0 | A0 | OK | OK | 0 |
| Got two equal A inputs | OK | | | | | 1 |
| | | S* | | | | |

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (8) -*

má escolha para estado: é preciso distinguir 2 zeros em A de 2 uns em A, no passado

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK | OK | A1 | A1 | 0 |
| Got a 1 on A | A1 | A0 | A0 | OK | OK | 0 |
| Got two equal A inputs | OK | ? | OK | OK | ? | 1 |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK0 | OK0 | A1 | A1 | 0 |
| Got a 1 on A | A1 | A0 | A0 | OK1 | OK1 | 0 |
| Two equal, A=0 last | OK0 | | | | | 1 |
| Two equal, A=1 last | OK1 | | | | | 1 |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK0 | OK0 | A1 | A1 | 0 |
| Got a 1 on A | A1 | A0 | A0 | OK1 | OK1 | 0 |
| Two equal, A=0 last | OK0 | OK0 | OK0 | OK1 | A1 | 1 |
| Two equal, A=1 last | OK1 | | | | | 1 |
| | | S* | | | | |

| Meaning | S | A B 00 | 01 | 11 | 10 | Z |
|---|---|---|---|---|---|---|
| Initial state | INIT | A0 | A0 | A1 | A1 | 0 |
| Got a 0 on A | A0 | OK0 | OK0 | A1 | A1 | 0 |
| Got a 1 on A | A1 | A0 | A0 | OK1 | OK1 | 0 |
| Two equal, A=0 last | OK0 | OK0 | OK0 | OK1 | A1 | 1 |
| Two equal, A=1 last | OK1 | A0 | OK0 | OK1 | OK1 | 1 |
| | | S* | | | | |

Resulta da 2ª parte da especificação da máquina de estados

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (9) -*

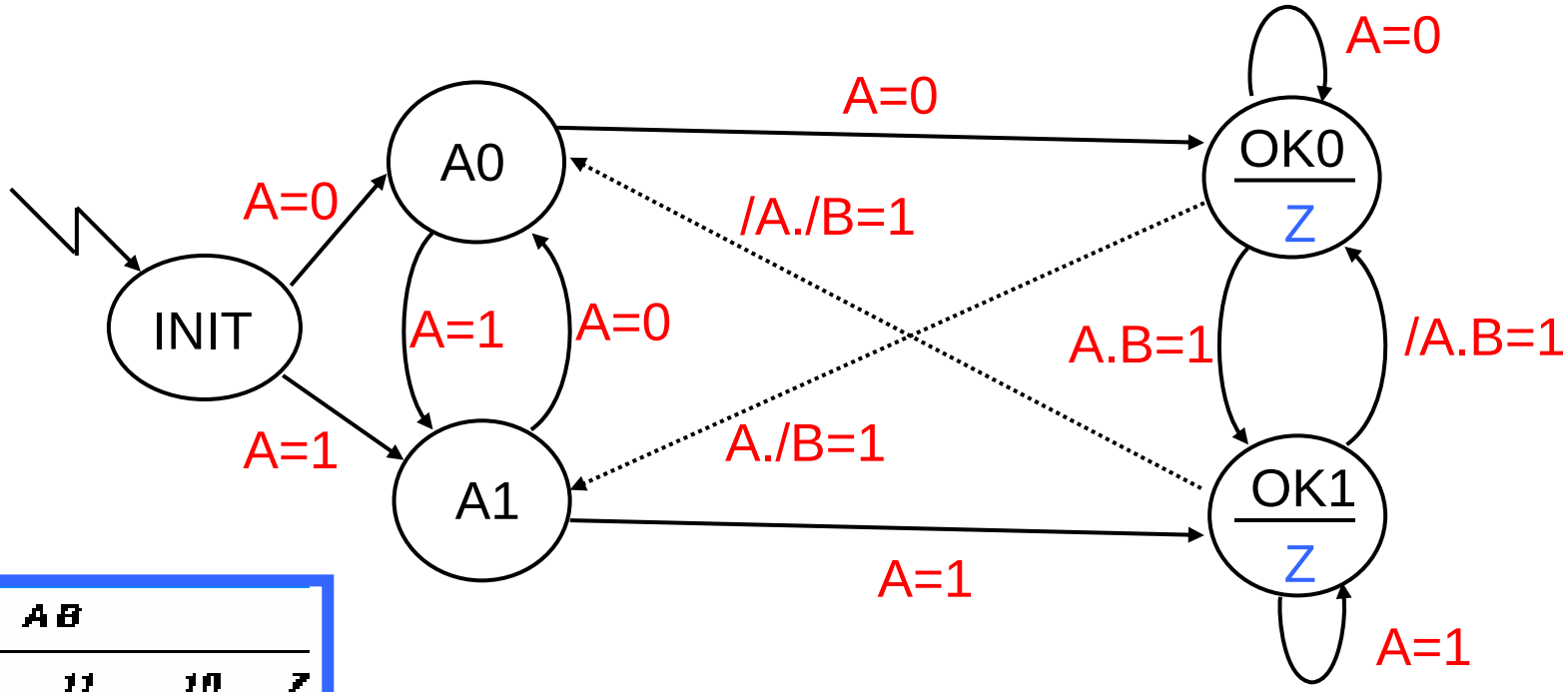



Diagrama de estados

| | AB | | | | |
|---|---|---|---|---|---|
| S | 00 | 01 | 11 | 10 | Z |
| INIT | A0 | A0 | A1 | A1 | 0 |
| A0 | OK0 | OK0 | A1 | A1 | 0 |
| A1 | A0 | A0 | OK1 | OK1 | 0 |
| OK0 | OK0 | OK0 | OK1 | A1 | 1 |
| OK1 | A0 | OK0 | OK1 | OK1 | 1 |
| | | | S* | | |

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (10) -*

- No próximo passo determina-se quantas variáveis binárias são necessárias para representar todos os estados da tabela de estados e saídas.
- De seguida, atribui-se uma combinação (das variáveis de estado binárias) a cada estado.
- A combinação atribuída a um estado particular é o código do estado.

- Com n *flip-flops*, pode codificar-se $2^n$ estados.
- O número de *flip-flops* necessário para codificar os s estados é $\lceil \log_2 s \rceil$.
- No presente problema, como existem 5 estados, exige-se 3 *flip-flops*.

| | AB | | | | |
|---|---|---|---|---|---|
| S | 00 | 01 | 11 | 10 | Z |
| INIT | A0 | A0 | A1 | A1 | 0 |
| A0 | OK0 | OK0 | A1 | A1 | 0 |
| A1 | A0 | A0 | OK1 | OK1 | 0 |
| OK0 | OK0 | OK0 | OK1 | A1 | 1 |
| OK1 | A0 | OK0 | OK1 | OK1 | 1 |
| | S* | | | | |

Tabela de estados e saídas

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (11) -*

- Há várias alternativas de codificação dos 5 estados.

| | Assignment | | | |
|---|---|---|---|---|
| State Name | Simplest Q1–Q3 | Decomposed Q1–Q3 | One-hot Q1–Q5 | Almost One-hot Q1–Q4 |
| INIT | 000 | 000 | 00001 | 0000 |
| A0 | 001 | 100 | 00010 | 0001 |
| A1 | 010 | 101 | 00100 | 0010 |
| OK0 | 011 | 110 | 01000 | 0100 |
| OK1 | 100 | 111 | 10000 | 1000 |

- A atribuição mais simples dos s códigos aos estados consiste em usar os s valores binários iniciais que aparecem na contagem binária ordenada.
- Esta atribuição nem sempre é aquela que conduz às expressões mais simplificadas para as saídas do bloco que gera o próximo estado e para as saídas da FSM.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (12) -*

- A atribuição de códigos aos estados tem um forte impacto no "custo" (área) do circuito e deve ter em consideração a selecção dos elementos de memória e a abordagem a usar na concretização dos 2 blocos de lógica combinacional (próximo estado e saídas).
- Na maior parte dos problemas, para escolher a melhor atribuição de códigos aos estados, seria preciso experimentar todas as atribuições possíveis.
- Não é exequível para fazer isto manualmente!!! No presente exemplo, existem 6.720 formas diferentes de atribuir 5 (das 8) combinações de 3-bits aos 5 estados.
- Em alternativa, o projectista deve guiar-se pela sua experiência e por orientações práticas para obter uma boa atribuição de códigos aos estados.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (13) -*

- Orientações para a atribuição de códigos aos estados:
  - Escolher um código inicial que seja fácil de forçar com o mecanismo de *reset* dos F/Fs (normalmente será 000...0 ou 111...1).
  - Minimizar o número de variáveis de estado que mudam em cada transição de estado.
  - Maximizar o número de variáveis de estado que se mantêm inalteradas num grupo de estados inter-relacionados.
  - Explorar as simetrias existentes na especificação do problema e as simetrias que lhe correspondem na tabela de estados. Por exemplo, se um estado (ou grupo de estados) tiver um significado semelhante a outro estado (grupo), após escolher a atribuição para o primeiro estado (grupo) deve escolher-se uma atribuição similar (diferindo em 1,2,… bits) para o segundo.
  - Decompor o conjunto de variáveis de estado em bits ou campos de bits, cada um relacionado com um aspecto funcional das entradas ou das saídas da FSM.
  - Ponderar a hipótese de usar mais variáveis de estado do que o valor mínimo, por forma a tornar possível a decomposição anterior.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (14) -*

- Algumas das orientações anteriores foram incorporadas na atribuição do tipo "decomposed".
- INIT é 000, um código fácil de forçar através da entrada assíncrona dos flip-flops (CLR).
- INIT nunca mais é acedido após o arranque da máquina. Por isso, usou-se o bit (variável de estado) Q1 para indicar se o estado actual é ou não INIT.

| State Name | Decomposed Q1–Q3 |
|---|---|
| INIT | 000 |
| A0 | 100 |
| A1 | 101 |
| OK0 | 110 |
| OK1 | 111 |

- Q2,Q3 permitem distinguir os outros 4 estados.
- Q2 está relacionado com o facto de a saída ser 1 ou não no estado actual.
- Q3 está relacionado com o valor anterior de A.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (15) -*

- A atribuição do tipo one-hot pode ser aplicada a qualquer máquina de estados.
- Este tipo de atribuição utiliza mais variáveis de estado do que o valor mínimo: usa 1 bit por estado.
- Esta opção conduz normalmente a expressões simples para as saídas dado que cada *flip-flop* só é colocado a 1 nas transições para um único estado.
- A atribuição do tipo almost one-hot segue a estratégia *one-hot* excepto para o estado inicial, em que se usa a combinação 00…0.

| State Name | One-hot Q1–Q5 | Almost One-hot Q1–Q4 |
|---|---|---|
| INIT | 00001 | 0000 |
| A0 | 00010 | 0001 |
| A1 | 00100 | 0010 |
| OK0 | 01000 | 0100 |
| OK1 | 10000 | 1000 |

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (16) -*

- No presente exemplo existem códigos de estado não usados porque o número de estados é menor do que o número de combinações das variáveis de estado.
- Como lidar com os códigos não usados?
- Numa abordagem com risco mínimo, considera-se que a máquina pode atingir um estado com código não usado, por exemplo ao ocorrer uma falha no *hardware*.
- Para qualquer estado com código não usado, implementa-se uma transição explicita para regressar a um estado seguro.
- Numa abordagem com custo mínimo, considera-se que a máquina nunca atinge um estado com código não usado.
- Nos códigos de estado não usados, as entradas na tabela de estados relativas ao próximo estado podem ser definidas como “don’t care”.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (17) -*

- Para obter a versão final da tabela de (transições de) estados e saídas substitui-se o nome dos estados pelo seu código.
- A tabela obtida mostra qual é o próximo estado para cada combinação (estado actual, entradas).
- Para o exemplo em estudo, a tabela de transições de estados e saídas foi obtida efectuando uma atribuição do tipo *decomposed*.

| State Name | Decomposed Q1–Q3 |
|---|---|
| INIT | 000 |
| A0 | 100 |
| A1 | 101 |
| OK0 | 110 |
| OK1 | 111 |

estado actual entradas saída

| | A B | | | | |
|---|---|---|---|---|---|
| Q1 Q2 Q3 | 00 | 01 | 11 | 10 | Z |
| 000 | 100 | 100 | 101 | 101 | 0 |
| 100 | 110 | 110 | 101 | 101 | 0 |
| 101 | 100 | 100 | 111 | 111 | 0 |
| 110 | 110 | 110 | 111 | 101 | 1 |
| 111 | 100 | 110 | 111 | 111 | 1 |
| | Q1*Q2*Q3* | | | | |

próximo estado

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (18) -*

- No próximo passo obtém-se a tabela de excitação. Esta tabela define quais os valores a aplicar nas entradas dos *flip-flops* por forma a que a máquina transite para o próximo estado desejado.
- A estrutura e o conteúdo da tabela de excitação depende do tipo de *flip-flop* escolhido para implementar a memória de estado (D, J-K, T, ...).
- Actualmente, a maioria dos projectos com máquinas de estado utiliza *flip-flops* D, por existirem em todas as tecnologias (CIs SSI, PLDs, FPGAs) e por serem fáceis de usar.

- A equação característica do *flip-flop* D é: Q* = D.
- No caso de se usar *flip-flops* D, a tabela de excitação é identica à tabela de transições de estados, excepto os nomes das colunas que deixam de ser o próximo estado (Q1* Q2* Q3*) para ser a entrada dos F/Fs (D1 D2 D3). →

| | A B | | | | |
|---|---|---|---|---|---|
| Q1 Q2 Q3 | 00 | 01 | 11 | 10 | Z |
| 000 | 100 | 100 | 101 | 101 | 0 |
| 100 | 110 | 110 | 101 | 101 | 0 |
| 101 | 100 | 100 | 111 | 111 | 0 |
| 110 | 110 | 110 | 111 | 101 | 1 |
| 111 | 100 | 110 | 111 | 111 | 1 |
| | D1 D2 D3 | | | | |

# 6. Conceitos sobre Circuitos Sequenciais

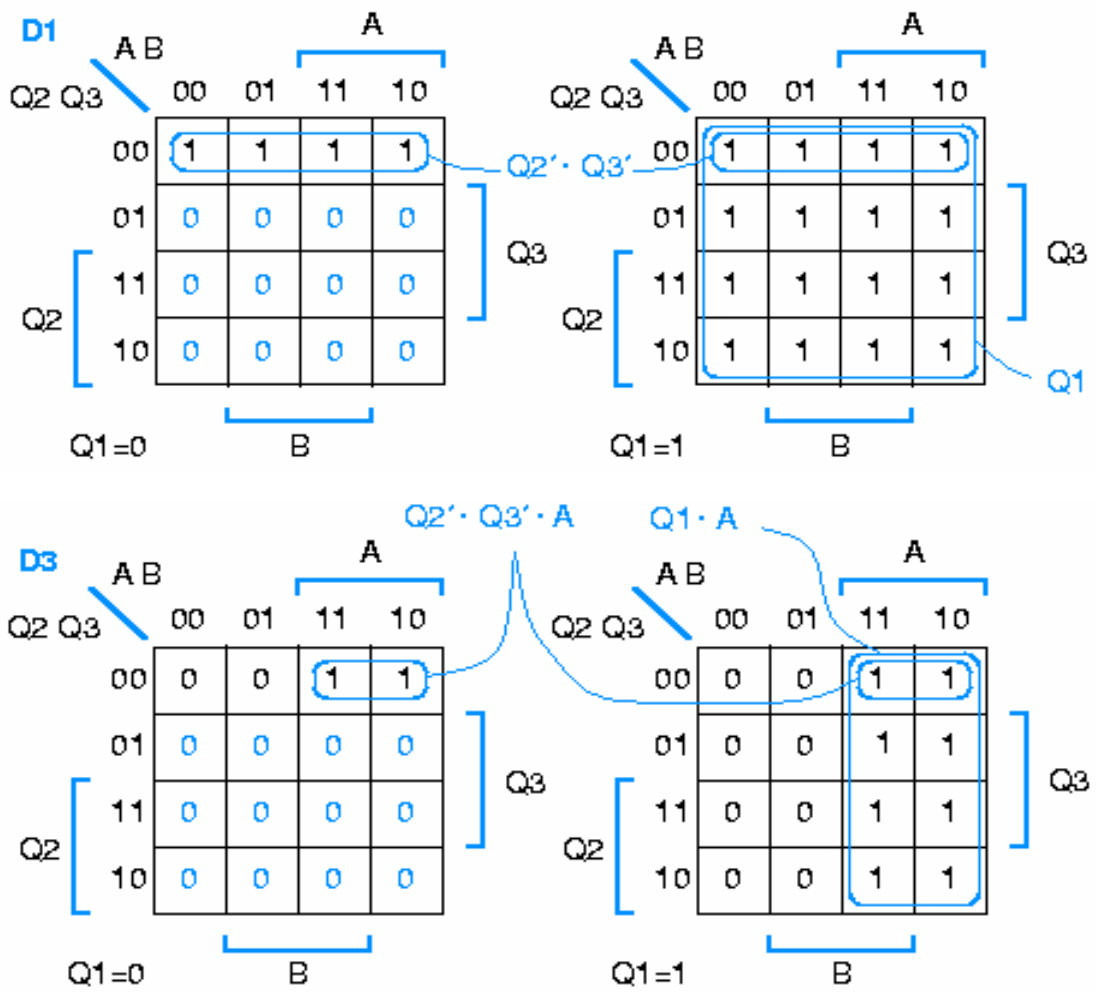
*- Projecto de máquinas de estados (19) -*

- Se estiver completa, a tabela de excitação funciona como tabela de verdade das saídas combinacionais D1, D2 e D3, ao exprimi-las em função das entradas A e B e do estado Q1, Q2 e Q3.
- Se estiver completa, a tabela anterior também funciona como tabela de verdade das saídas da FSM, ao exprimir a saída combinacional Z em função do estado Q1, Q2 e Q3.
- Com base na informação da tabela de excitação, pode usar-se mapas de Karnaugh para obter a expressão mínima para cada saída combinacional.
- A tabela de excitação apresentada não está completa porque não especifica o valor do próximo estado e da saída, em todas as combinações (estado, entradas). Ou seja, a informação relativa aos estados não usados não consta da tabela.
- No presente exemplo, o tratamento dos estados não usados seguirá as duas abordagens mencionadas: risco mínimo e custo mínimo.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (20) -*

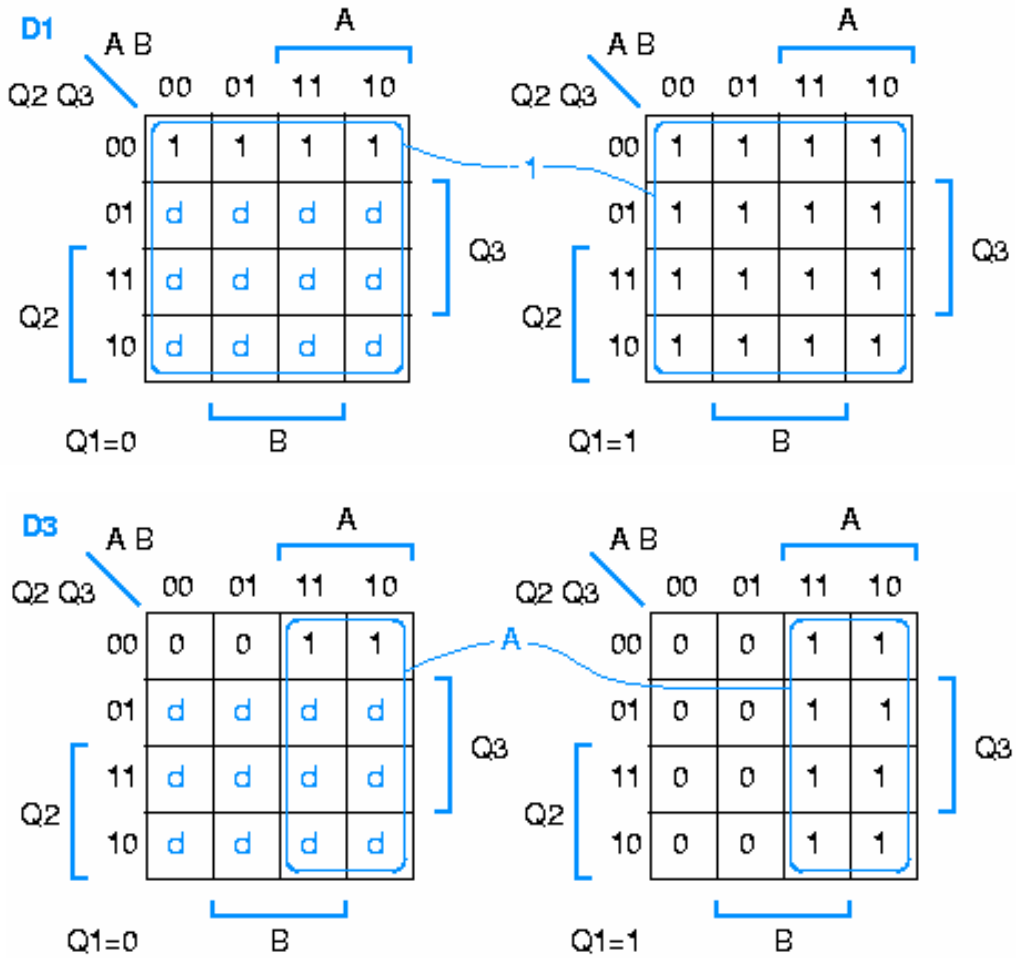
- Na abordagem de risco mínimo, em qualquer estado não usado (Q1Q2Q3 ≡ 001, 010, 011) o próximo estado é INIT ≡ 000.
- A partir dos mapas ao lado (mais o mapa de D2 que falta) obtém-se as expressões:
  D1 = Q1 + Q2'· Q3'
  D2 = Q1·Q3'·A'+Q1·Q3·A+Q1·Q2·B
  D3 = Q1· A + Q2'· Q3'· A
- A tabela de excitação mostra que Z está activa nos estados 110 e 111:
  Z = Q1· Q2· Q3' + Q1· Q2· Q3
  = Q1· Q2



# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (21) -*

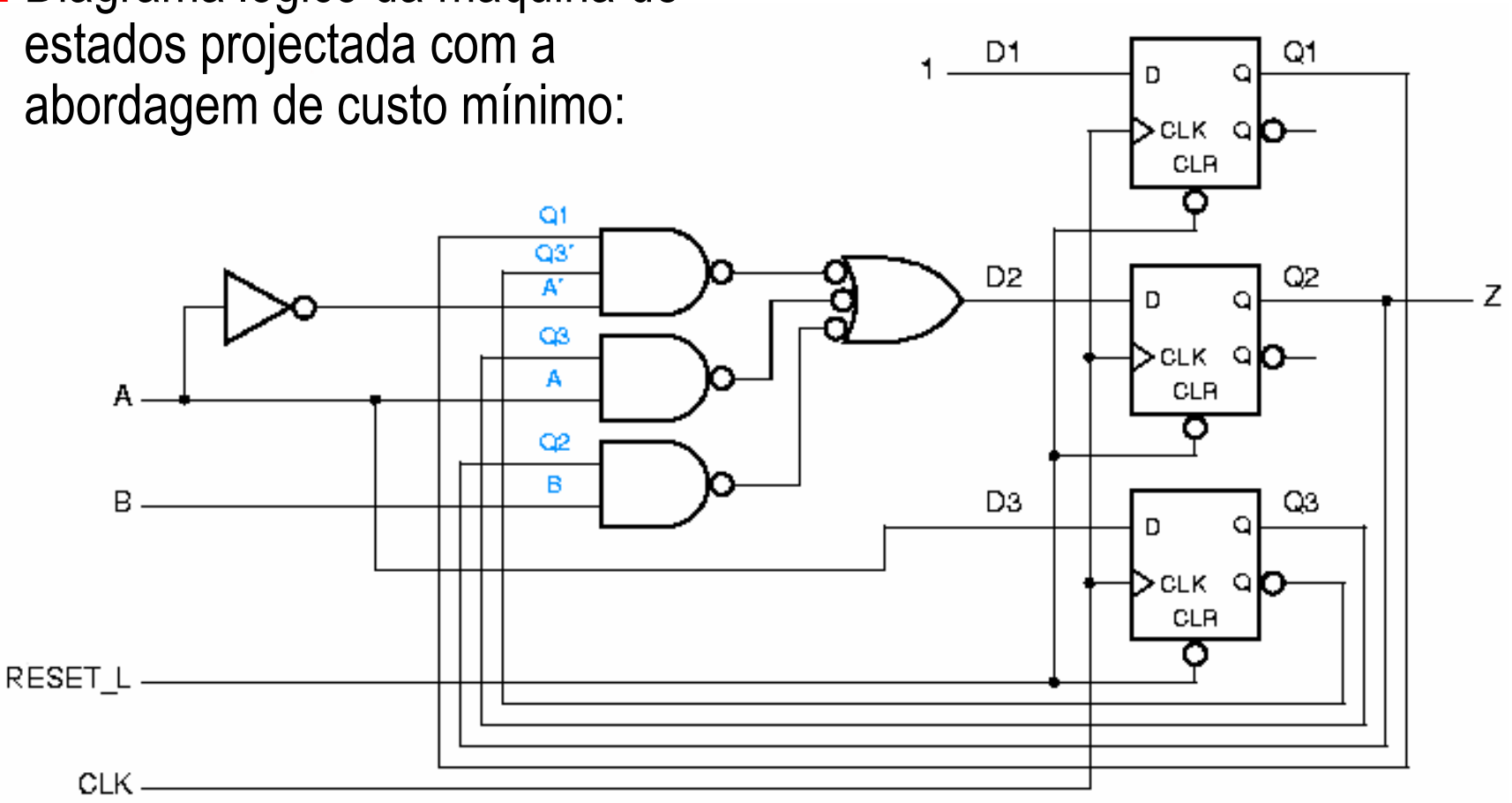
- Na abordagem de custo mínimo, em qualquer estado não usado (Q1Q2Q3 ≡ 001, 010, 011) o próximo estado é *don't-care*.
- A partir dos mapas ao lado (mais o mapa de D2 que falta) obtém-se as expressões:
  D1 = 1
  D2 = Q1·Q3'·A'+Q3·A+Q2·B
  D3 = A
- Como Z está activa nos estados 110 e 111 e é *don't care* nos estados não usados, obtém-se:
  Z = Q2



# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (22) -*

- Diagrama lógico da máquina de estados projectada com a abordagem de custo mínimo:



# 6. Conceitos sobre Circuitos Sequenciais

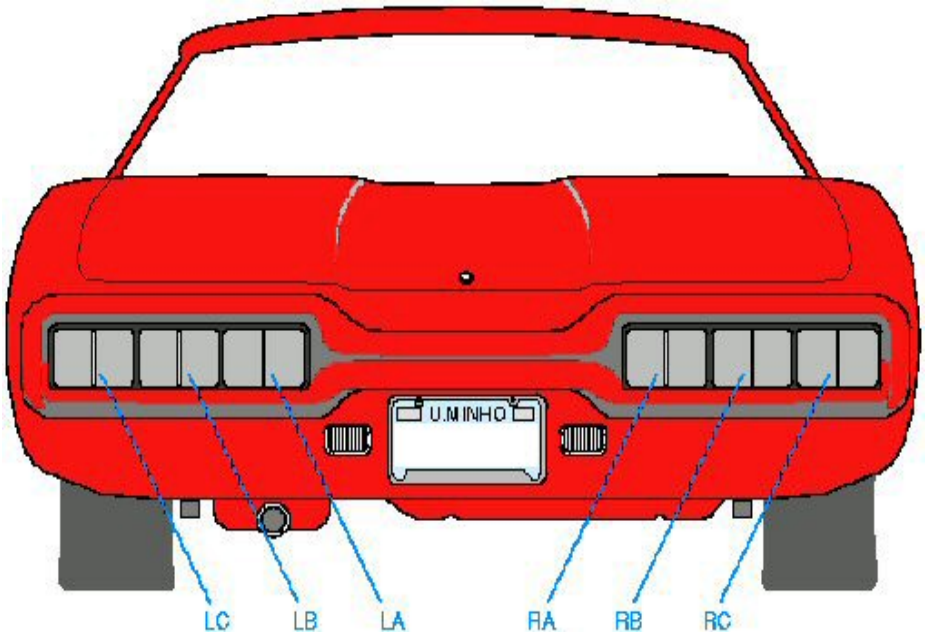
*- Projecto de máquinas de estados (23) -*

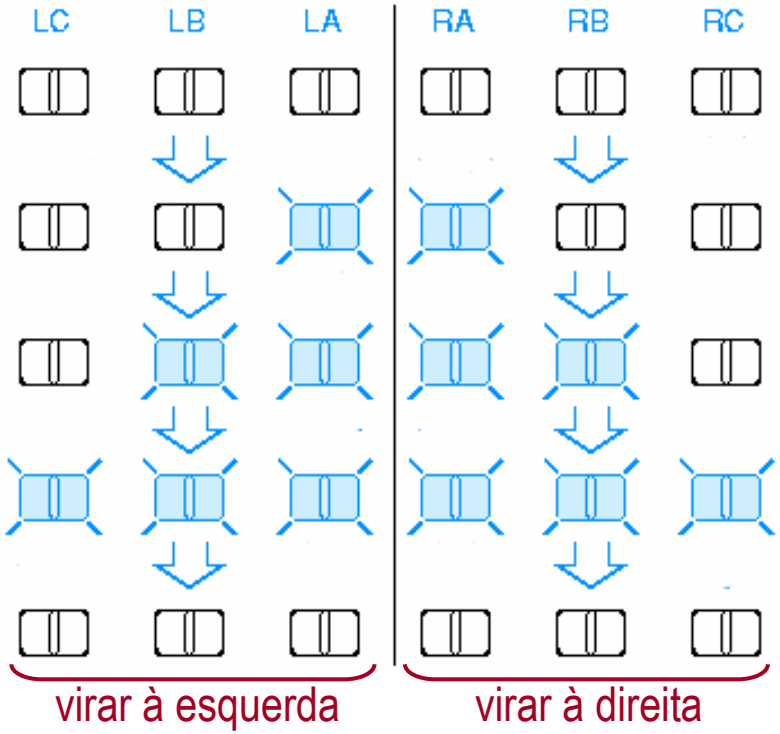
- É frequente usar diagramas de estados para projectar máquinas de estados.
- Projectar um diagrama de estados é similar, mas ainda mais simples, do que escrever uma tabela de estados.
- Contudo, é fácil conceber um diagrama de estados com algumas ambiguidades, que são impossíveis de ocorrer numa tabela de estados.
- Num diagrama de estados incorrectamente projectado, é comum o próximo estado não ser especificado em certas combinações (estado, entradas).
- Também pode ocorrer a situação em que alguma combinação (estado, entradas) permite transitar para mais do que um estado.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (24) -*

- O próximo exemplo é uma máquina de estados que controla as luzes de trás num carro.
- Existem 3 luzes do lado direito (RA, RB e RC) e outras 3 do lado esquerdo (LA, LB e LC).
- A máquina de estados possui 2 entradas LEFT / RIGHT, indicando um pedido para virar à esquerda / direita.
- Possui também uma entrada de emergência HAZ indicando um pedido para as 6 luzes piscarem.



# 6. Conceitos sobre Circuitos Sequenciais

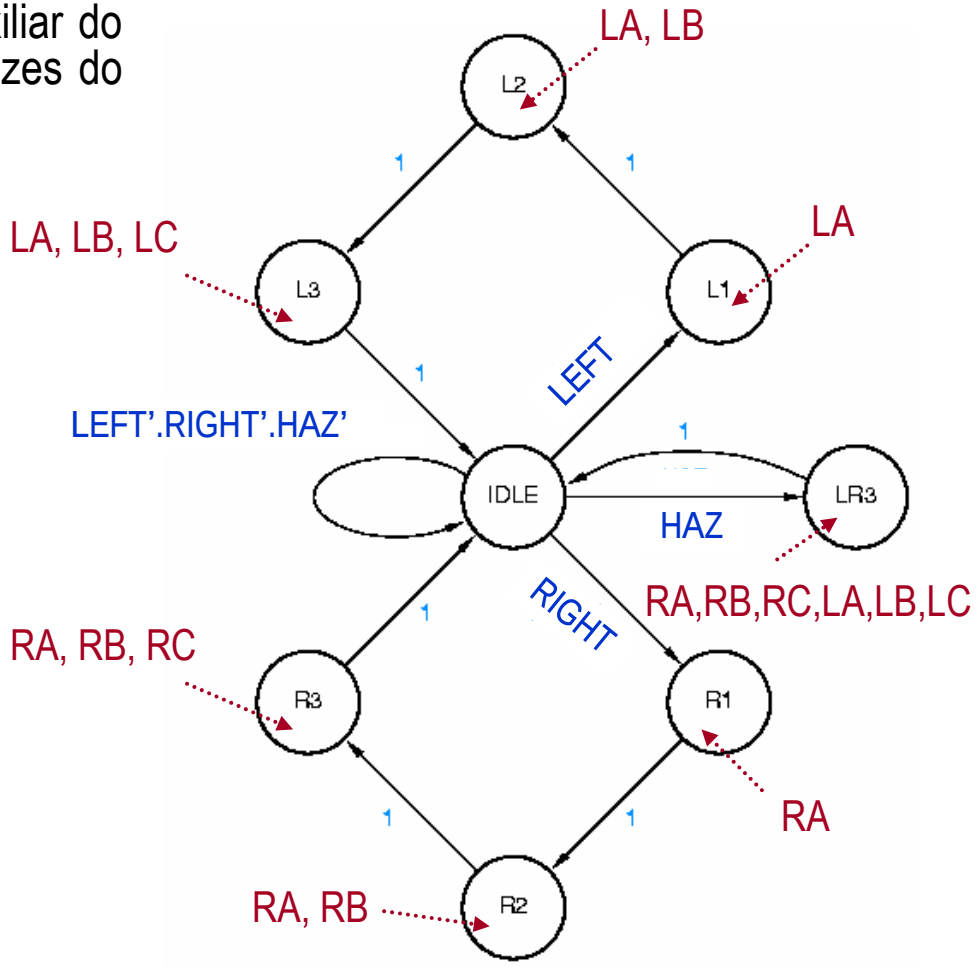
*- Projecto de máquinas de estados (25) -*

- Diagrama de estados e tabela das saídas (auxiliar do diagrama de estados) para o controlador de luzes do carro:

Output Table

| State | LC | LB | LA | RA | RB | RC |
|---|---|---|---|---|---|---|
| IDLE | 0 | 0 | 0 | 0 | 0 | 0 |
| L1 | 0 | 0 | 1 | 0 | 0 | 0 |
| L2 | 0 | 1 | 1 | 0 | 0 | 0 |
| L3 | 1 | 1 | 1 | 0 | 0 | 0 |
| R1 | 0 | 0 | 0 | 1 | 0 | 0 |
| R2 | 0 | 0 | 0 | 1 | 1 | 0 |
| R3 | 0 | 0 | 0 | 1 | 1 | 1 |
| LR3 | 1 | 1 | 1 | 1 | 1 | 1 |

todas as luzes apagadas →

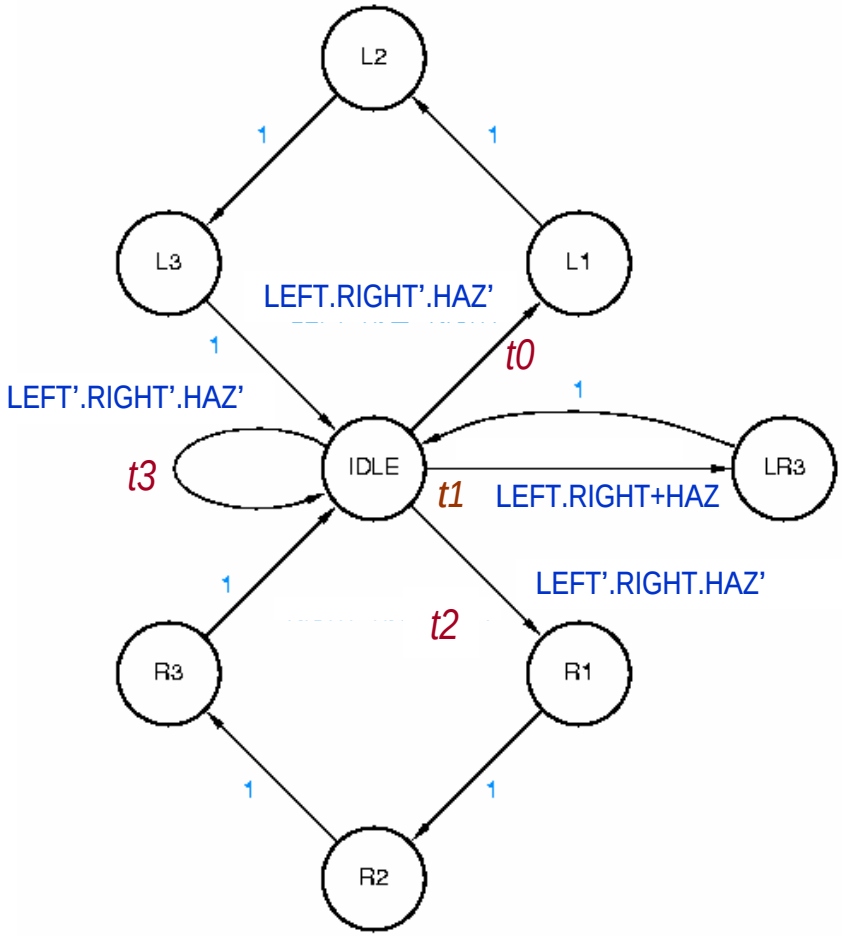



- Este diagrama é limitado: só está correcto se não houver activação simultânea de várias entradas.

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (26) -*

- O diagrama de estados deve ser alterado para:
  - atribuir uma prioridade superior à entrada HAZ.
  - considerar a activação simultânea de LEFT e RIGHT como uma situação de emergência.
- O novo diagrama de estados já não tem ambiguidades porque as condições associadas às várias transições que partem do mesmo estado são mutuamente exclusivas entre si e no conjunto cobrem todas as combinações das entradas.



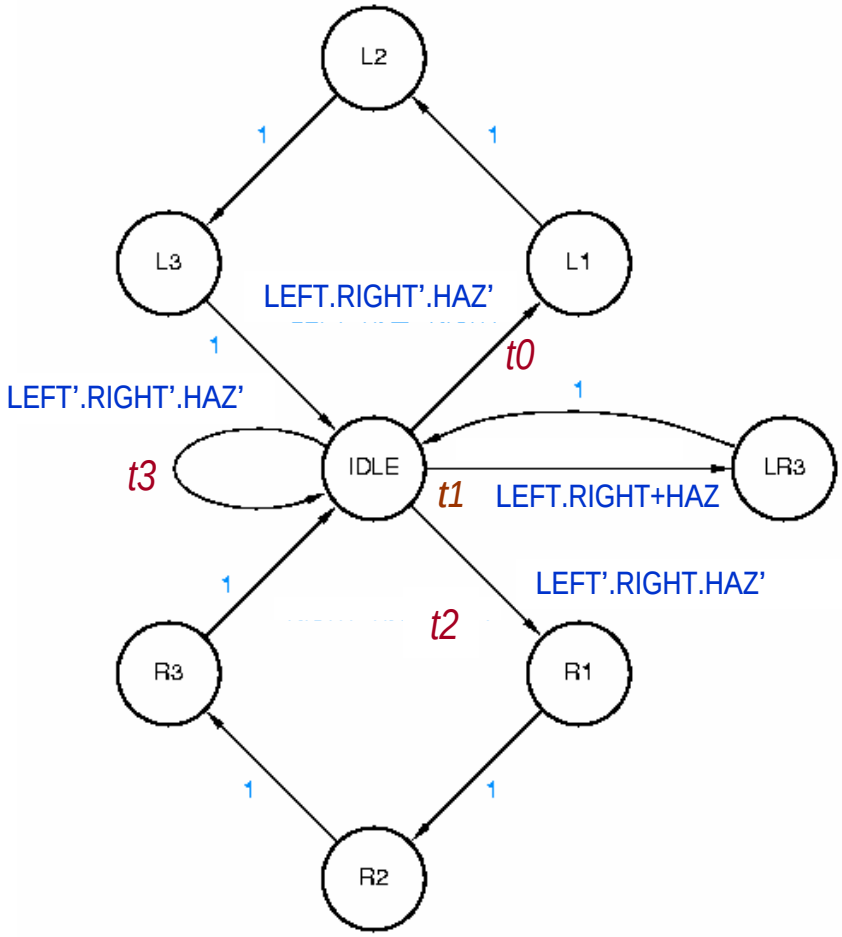
# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (27) -*

- Demonstrar que o novo diagrama de estados não tem ambiguidades:

Transições a partir do estado **IDLE**

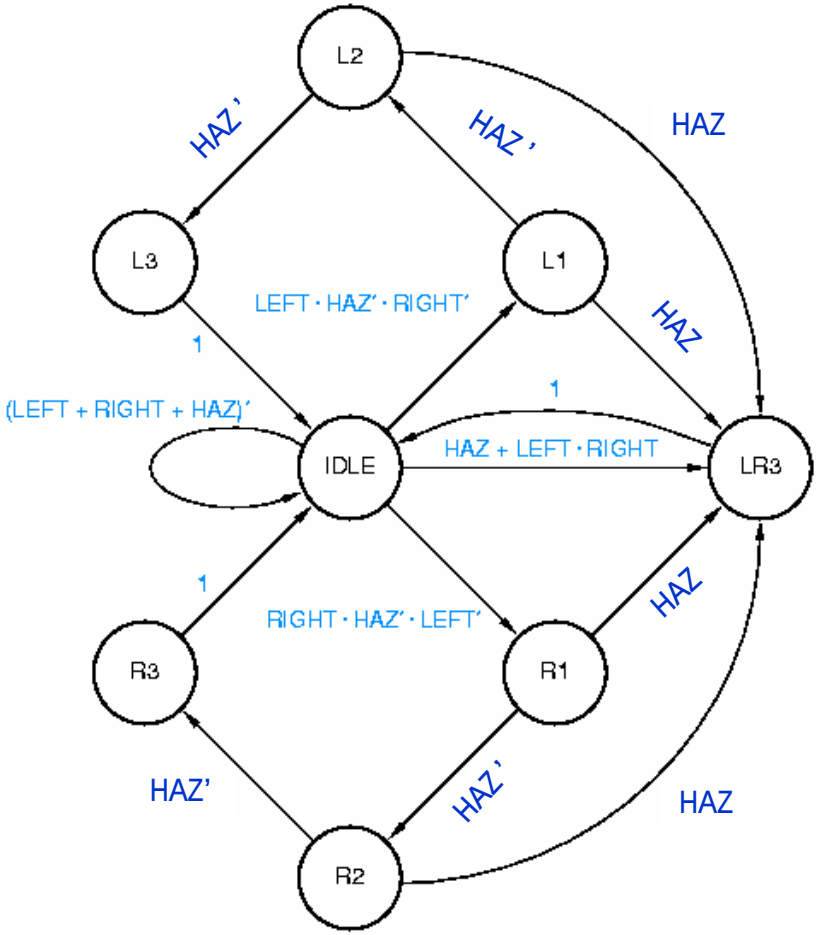
| LEFT | RIGHT | HAZ | Transição | Próximo Estado |
|---|---|---|---|---|
| 0 | 0 | 0 | *t3* | IDLE |
| 0 | 0 | 1 | *t1* | LR3 |
| 0 | 1 | 0 | *t2* | R1 |
| 0 | 1 | 1 | *t1* | LR3 |
| 1 | 0 | 0 | *t0* | L1 |
| 1 | 0 | 1 | *t1* | LR3 |
| 1 | 1 | 0 | *t1* | LR3 |
| 1 | 1 | 1 | *t1* | LR3 |



# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (28) -*

- Antes de sintetizar a FSM, vai introduzir-se uma alteração: permite-se que a FSM vá para o estado LR3 o mais cedo possível, ou seja, antes de terminar uma sequência L1→L2→L3 ou R1→R2→R3 já iniciada, desde que tenha ocorrido um pedido de emergência através da activação de HAZ.
- A FSM possui 8 estados, logo exige-se 3 *flip-flops* para a memória de estado.




- IDLE=000.
- Q2 distingue as partes simétricas R1→R2→R3 e L1→L2→L3.
- Usa-se código gray em Q1Q0 nas sequências IDLE→L1→L2→L3 e IDLE→R1→R2→R3 para minimizar as transições nas variáveis de estado.

| State | Q2 | Q1 | Q0 |
|---|---|---|---|
| IDLE | 0 | 0 | 0 |
| L1 | 0 | 0 | 1 |
| L2 | 0 | 1 | 1 |
| L3 | 0 | 1 | 0 |
| R1 | 1 | 0 | 1 |
| R2 | 1 | 1 | 1 |
| R3 | 1 | 1 | 0 |
| LR3 | 1 | 0 | 0 |

# 6. Conceitos sobre Circuitos Sequenciais

*- Projecto de máquinas de estados (29) -*

as expressões substituem as combinações das entradas

- Tabela de transições de estado
- A tabela foi alterada para ser mais compacta → passou de 64 para 15 linhas

| S | Q2 | Q1 | Q0 | Transition Expression | S* | Q2* | Q1* | Q0* |
|---|---|---|---|---|---|---|---|---|
| IDLE | 0 | 0 | 0 | (LEFT + RIGHT + HAZ)' | IDLE | 0 | 0 | 0 |
| IDLE | 0 | 0 | 0 | LEFT · HAZ' · RIGHT' | L1 | 0 | 0 | 1 |
| IDLE | 0 | 0 | 0 | HAZ + LEFT · RIGHT | LR3 | 1 | 0 | 0 |
| IDLE | 0 | 0 | 0 | RIGHT · HAZ' · LEFT' | R1 | 1 | 0 | 1 |
| L1 | 0 | 0 | 1 | HAZ' | L2 | 0 | 1 | 1 |
| L1 | 0 | 0 | 1 | HAZ | LR3 | 1 | 0 | 0 |
| L2 | 0 | 1 | 1 | HAZ' | L3 | 0 | 1 | 0 |
| L2 | 0 | 1 | 1 | HAZ | LR3 | 1 | 0 | 0 |
| L3 | 0 | 1 | 0 | 1 | IDLE | 0 | 0 | 0 |
| R1 | 1 | 0 | 1 | HAZ' | R2 | 1 | 1 | 1 |
| R1 | 1 | 0 | 1 | HAZ | LR3 | 1 | 0 | 0 |
| R2 | 1 | 1 | 1 | HAZ' | R3 | 1 | 1 | 0 |
| R2 | 1 | 1 | 1 | HAZ | LR3 | 1 | 0 | 0 |
| R3 | 1 | 1 | 0 | 1 | IDLE | 0 | 0 | 0 |
| LR3 | 1 | 0 | 0 | 1 | IDLE | 0 | 0 | 0 |

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (1) -*

- A linguagem VHDL não possui qualquer construtor específico para descrever máquinas de estados.
- A maioria dos conceitos, necessários para descrever máquinas de estados síncronas em VHDL, já foram apresentados.
- O processo e o mecanismo da lista de eventos (usado pelos simuladores para guardar o historial das mudanças que ocorrem nos sinais) são o suporte base para descrever circuitos sequenciais em VHDL.
- Para modelar o comportamento "síncrono com o relógio" exige-se apenas uma característica do VHDL ainda não introduzida:

  o atributo **`event`**, que pode ser anexado ao nome dum sinal para originar um valor que será igual a **`true`** se o sinal tiver mudado de valor e é igual a **`false`** em caso contrário.
- É frequente usarem-se tipos de dados enumerados e o construtor CASE para descrever máquinas de estados.

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (2) -*

- Pode usar-se o atributo **`event`** para modelar um *flip-flop* D
  - com um reset assíncrono (CLR) e
  - sensível às transições positivas do relógio (CLK).
- A influência da entrada de reset sobrepõe-se ao comportamento que depende da entrada de relógio.
- Para qualquer transição do sinal CLK (0→1 ou 1→0), `CLK'event` é igual a `true`.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity VposDff is
  port (CLK, CLR, D: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end VposDff;

architecture VposDff_arch of VposDff is
begin
  process (CLK, CLR)
  begin
    if CLR='1' then Q <= '0'; QN <= '1';
    elsif CLK'event and CLK='1' then Q <= D; QN <= not D;
    end if;
  end process;
end VposDff_arch;
```

- Ao lado apresentam-se mais 2 formas de descrever um *flip-flop* D sensível às transições positivas, mas sem entrada de *reset*. →

```
process
  wait until CLK'event and CLK='1';
  Q <= D;
end process;

Q <= D when CLK'event and CLK='1' else Q;
```

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (3) -*

- Em VHDL, é possível descrever de várias formas uma máquina de estados como a do 1º exemplo apresentado neste módulo.
- A primeira abordagem consiste em construir a tabela de estados e saídas no papel e depois convertê-la manualmente para código VHDL. Esta tabela já foi obtida antes e é repetida no próximo slide.
- A 1ª coisa a fazer nesta abordagem é definir um tipo enumerado Sreg-type, cujos valores permitidos são identificadores correspondentes ao nome dos vários estados: INIT, A0, A1, OK0, OK1.
- Depois declara-se um sinal Sreg desse tipo enumerado, utilizado para guardar o estado actual da máquina.
- O corpo da arquitectura possui duas instruções concorrentes:
  - Um processo que reage apenas ao sinal CLOCK e implementa todas as transições de estado.
  - Uma atribuição selectiva que gera a saída Z da máquina de Moore.

# 6. Conceitos sobre Circuitos Sequenciais

*Máquinas de estados em VHDL (4)*

| | AB | | | | |
|---|---|---|---|---|---|
| S | 00 | 01 | 11 | 10 | Z |
| INIT | A0 | A0 | A1 | A1 | 0 |
| A0 | OK0 | OK0 | A1 | A1 | 0 |
| A1 | A0 | A0 | OK1 | OK1 | 0 |
| OK0 | OK0 | OK0 | OK1 | A1 | 1 |
| OK1 | A0 | OK0 | OK1 | OK1 | 1 |
| | S* | | | | |

```
library IEEE;
use IEEE.std_logic_1164.all;

entity smexamp is
  port ( CLOCK, A, B: in STD_LOGIC;
         Z: out STD_LOGIC );
end;

architecture smexamp_arch of smexamp is
type Sreg_type is (INIT, A0, A1, OK0, OK1);
signal Sreg: Sreg_type;
begin

  process (CLOCK) -- state-machine states and transitions
  begin
    if CLOCK'event and CLOCK = '1' then
      case Sreg is
        when INIT => if    A='0' then Sreg <= A0;
                     elsif A='1' then Sreg <= A1;    end if;
        when A0 =>   if    A='0' then Sreg <= OK0;
                     elsif A='1' then Sreg <= A1;  end if;
        when A1 =>   if    A='0' then Sreg <= A0;
                     elsif A='1' then Sreg <= OK1;  end if;
        when OK0 =>  if    A='0' then Sreg <= OK0;
                     elsif A='1' and B='0' then Sreg <= A1;
                     elsif A='1' and B='1' then Sreg <= OK1;  end if;
        when OK1 =>  if    A='0' and B='0' then Sreg <= A0;
                     elsif A='0' and B='1' then Sreg <= OK0;
                     elsif A='1' then Sreg <= OK1;             end if;
        when others => Sreg <= INIT;
      end case;
    end if;
  end process;

  with Sreg select  -- output values based on state
    Z <= '0' when INIT | A0 | A1,
         '1' when OK0 | OK1,
         '0' when others;

end smexamp_arch;
```

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (5) -*

- Como é feita a atribuição de códigos aos estados? Ou seja, qual a estratégia e qual o número de variáveis de estado usados?
- A ferramenta de síntese tem a liberdade de substituir os identificadores dos estados pelos valores inteiros (ou binários ) que desejar. Regra geral, a estratégia seguida para atribuir os códigos é do tipo mais simples, usando códigos cujo valor segue a ordem pela qual os estados estão listados no tipo enumerado.
- As ferramentas de síntese actuais permitem ao projectista usar uma codificação diferente.

- Uma das formas é usar a interface (gráfica) da ferramenta para escolher o tipo de codificação. A outra consiste em usar uma instrução **`attribute`**.
- O atributo definido pelo utilizador **`enum_encoding`** exige a biblioteca SYNOPSYS e é passado à ferramenta de síntese.

```
library IEEE;
use IEEE.std_logic_1164.all;
library SYNOPSYS;
use SYNOPSYS.attributes.all;
...
architecture smexampe_arch of smexamp is
type Sreg_type is (INIT, A0, A1, OK0, OK1);
attribute enum_encoding of Sreg_type: type is
          "0000 0001 0010 0100 1000";
signal Sreg: Sreg_type;
...
```

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (6) -*

- Outra forma de forçar um tipo de codificação dos estados consiste em definir o registo de estado mais explicitamente.
- Neste exemplo, define-se um sub-tipo **`Sreg_type`** baseado em std_logic com a largura desejada para o registo de estado.
- Depois, é possível definir uma constante desse subtipo para cada estado. Deste modo, força-se o tipo de codificação (valores) e os estados continuam a ser acedidos por nomes simbólicos (nomes).

```
library IEEE;
use IEEE.std_logic_1164.all;
...
architecture smexampc_arch of smexamp is
subtype Sreg_type is STD_LOGIC_VECTOR (1 to 4);
constant INIT: Sreg_type := "0000";
constant A0  : Sreg_type := "0001";
constant A1  : Sreg_type := "0010";
constant OK0 : Sreg_type := "0100";
constant OK1 : Sreg_type := "1000";
signal Sreg: Sreg_type;
...
```

# 6. Conceitos sobre Circuitos Sequenciais

*- Máquinas de estados em VHDL (7) -*

- Exemplo do sistema que controla as luzes de trás dum carro (slide 50): máquina de estados descrita em VHDL.
- Usam-se códigos de estado iguais aos valores das saídas (slide 47) => dispensa-se a lógica para gerar as saídas.

```
entity Vtbird is
  port ( CLOCK, RESET, LEFT, RIGHT, HAZ: in STD_LOGIC;
         LIGHTS: buffer STD_LOGIC_VECTOR (1 to 6) );
end;

architecture Vtbird_arch of Vtbird is
constant IDLE: STD_LOGIC_VECTOR (1 to 6) := "000000";
constant L3  : STD_LOGIC_VECTOR (1 to 6) := "111000";
constant L2  : STD_LOGIC_VECTOR (1 to 6) := "110000";
constant L1  : STD_LOGIC_VECTOR (1 to 6) := "100000";
constant R1  : STD_LOGIC_VECTOR (1 to 6) := "000001";
constant R2  : STD_LOGIC_VECTOR (1 to 6) := "000011";
constant R3  : STD_LOGIC_VECTOR (1 to 6) := "000111";
constant LR3 : STD_LOGIC_VECTOR (1 to 6) := "111111";
begin
  process (CLOCK)
  begin
    if CLOCK'event and CLOCK = '1' then
      if RESET = '1' then LIGHTS <= IDLE; else
        case LIGHTS is
          when IDLE => if HAZ='1' or (LEFT='1' and RIGHT='1') then LIGHTS <= LR3;
                       elsif LEFT='1'                         then LIGHTS <= L1;
                       elsif RIGHT='1'                        then LIGHTS <= R1;
                       else                                        LIGHTS <= IDLE;
                       end if;
          when L1   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L2; end if;
          when L2   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L3; end if;
          when L3   => LIGHTS <= IDLE;
          when R1   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R2; end if;
          when R2   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R3; end if;
          when R3   => LIGHTS <= IDLE;
          when LR3  => LIGHTS <= IDLE;
          when others => null;
        end case;
      end if;
    end if;
  end process;
end Vtbird_arch;
```