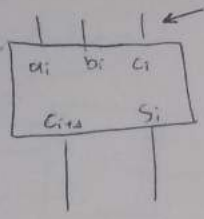
02/04

Copiar Matéria

06/04

Somador Completo de 2 bits

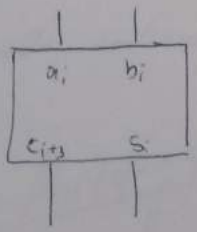


$S_i = a_i \oplus b_i \oplus c_i$

$C_{i+1} = a_i.b_i + a_i.c_i + b_i.c_i$

Def.: Meio Somador

↓ Não tem vem-vem

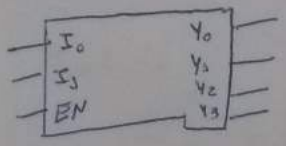


$S_i = a_i \oplus b_i$

$C_{i+1} = a_i . b_i$

6.4) Decodificador (decoder)

Símbolo



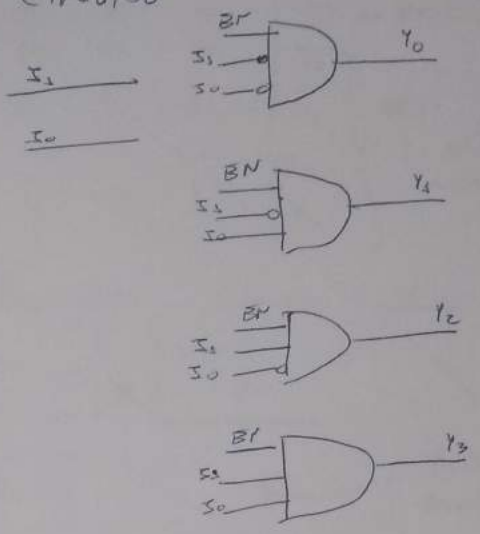
Endereço → codificação

EN	I_1	I_0	Y_3	Y_2	Y_1	Y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0

$X = 0$ ou 1

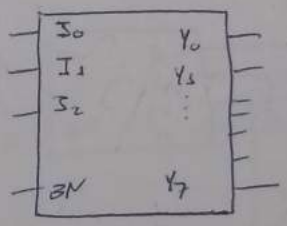
(tabela condensada)

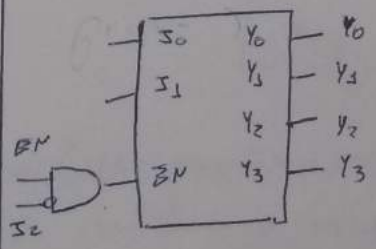
Circuito



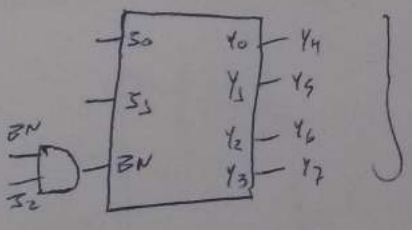
- Cascateamento

Ex: Construir um decoder 3:8 com enable.





Menos significativo



Mais significativo

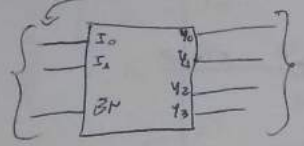
09/04/2018

Cap 6 - Circuitos Combinacionais

- Somador
- Decodificador (Decoder)

Ex: 2:4 com Enable

Endereço



Saída endereçada vai a 1.

Decoder Comercial 74xx138 (Cap 3)

3:8, 3 Enables

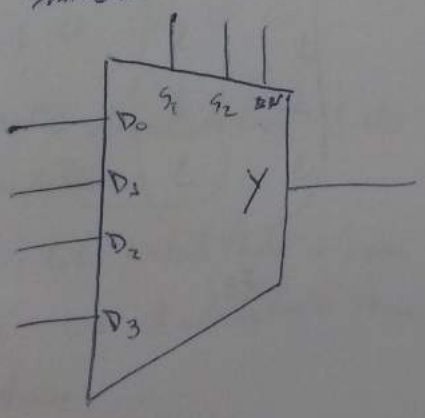
Saídas ativas em zero

(saída endereçada vai a 0, demais em 1)

- ~~Multiplicação~~
- Multiplexador (mux) (6.7 no livro)

Ex: mux 4:1 com Enable

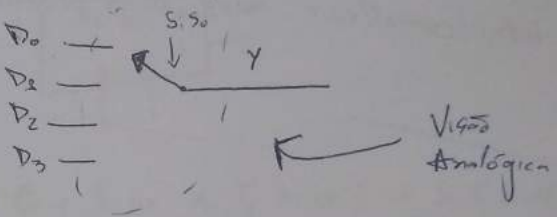
Símbolo:



EN	S1	S0	Y
0	x	x	0 → EN=0 ⇒ Y=0
1	0	0	D0
1	0	1	D1
1	1	0	D2
1	1	1	D3

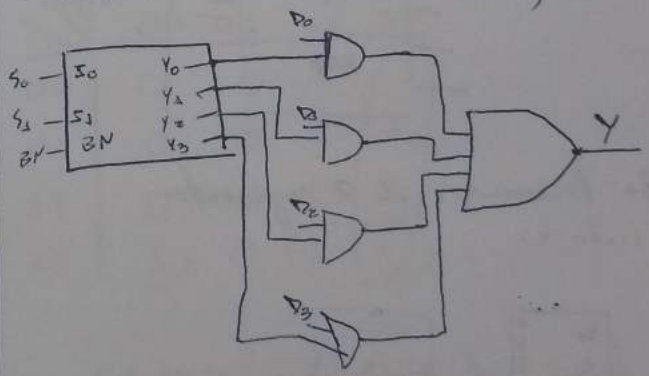
EN=1 · Y: entrada selecionada

Funciona como chave seletora



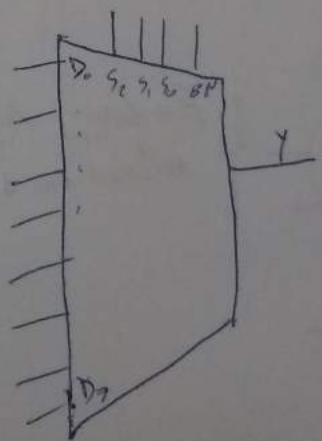
Na prática, o mux copia o nível lógico da entrada endereçada na saída.

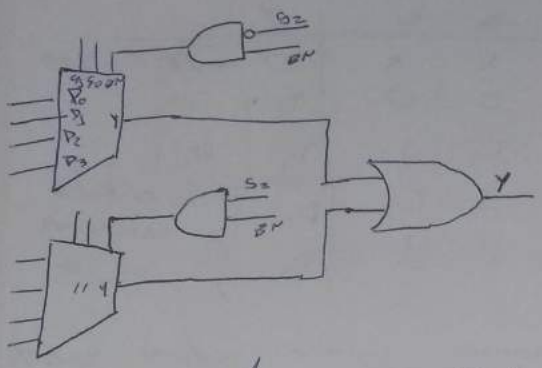
- Circuito (usando decoder 2:4)



- ~~CASCATEAMENTO~~

Ex: Construir mux 8:1 c/ enable usando mux 4:1 c/ enable)



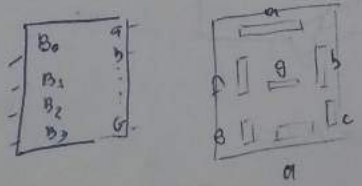


Na lista: comandar mux 16:1 c/ enable.

13/04/2018

Encoder (codificador)

3a: Acionador de 7 segmentos:
(lista 3)



Encoder binário com prioridade:

3a: Encoder 4:2

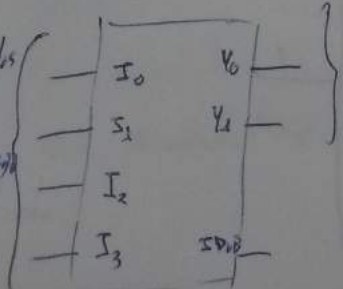
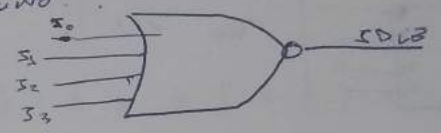


Tabela [I_3 mais prioritário]

I_3	I_2	I_1	I_0	Y_1	Y_0	EOLB	
0	0	0	1	0	0	0	Funcionamento normal (pelo menos uma entrada em 1)
0	0	1	x	0	1	0	
0	1	x	x	1	0	0	
1	x	x	x	1	1	0	
0	0	0	0	0	0	1	

Circuito:



$Y_1 = I_2 + I_3$

I_3I_2 \ I_1I_0	00	01	11	10
00	0	1 *	1	1 *
01	0	1 *	1	1 *
11	0	1 *	1	1 *
10	0	1 *	1	1 *

I_2 I_3

$Y_0 = I_3 + \bar{I}_2 \cdot I_1$

I_3I_2 \ I_1I_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

$\bar{I}_2 I_1$

I_3

S T Q Q S S D

__/__/__

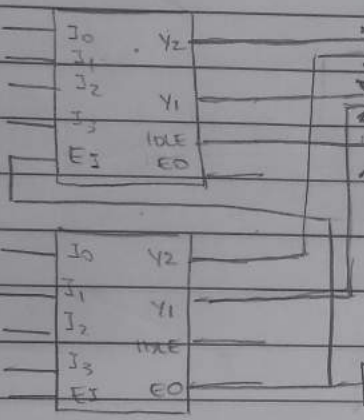
Y_0

$I_3 I_2$ / $I_1 I_0$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

I_3, I_2, I_1, I_3 → Y_0

$\overline{I_2} I_1$

Casacteamento 8:3



EI	I_3	I_2	I_1	I_0	Y_1	Y_0	IDLE	E_0	
0	x	x	x	x	0	0	1	0	•
1	1	x	x	x	1	1	0	0	
1	0	1	x	x	1	0	0	0	⊛
1	0	0	1	x	0	1	0	0	
1	0	0	0	1	0	0	0	0	
1	0	0	0	0	0	0	1	1	□

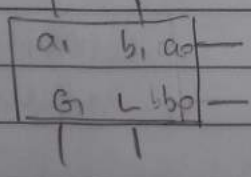
• CI desativada ($E_0 = 0$ p/ desabilitar demais)

⊛ CI habilitado e c/ entrada ativa

□ CI habilitado mas c/ entrada ativa

Comparador de Magnitude

Ex: comparador de (duas palavras de) 2 bits



$G_1 = 1$ se $a[1:0] > b[1:0]$

$L = 1$ se $a[1:0] < b[1:0]$

$G_1 = 0$ e $L = 0 \Rightarrow a[1:0] = b[1:0]$

__/__/__

Saída $G_1 = a_1\bar{b}_1 + a_1a_0\bar{b}_0 + \bar{b}_1a_0\bar{b}_0$

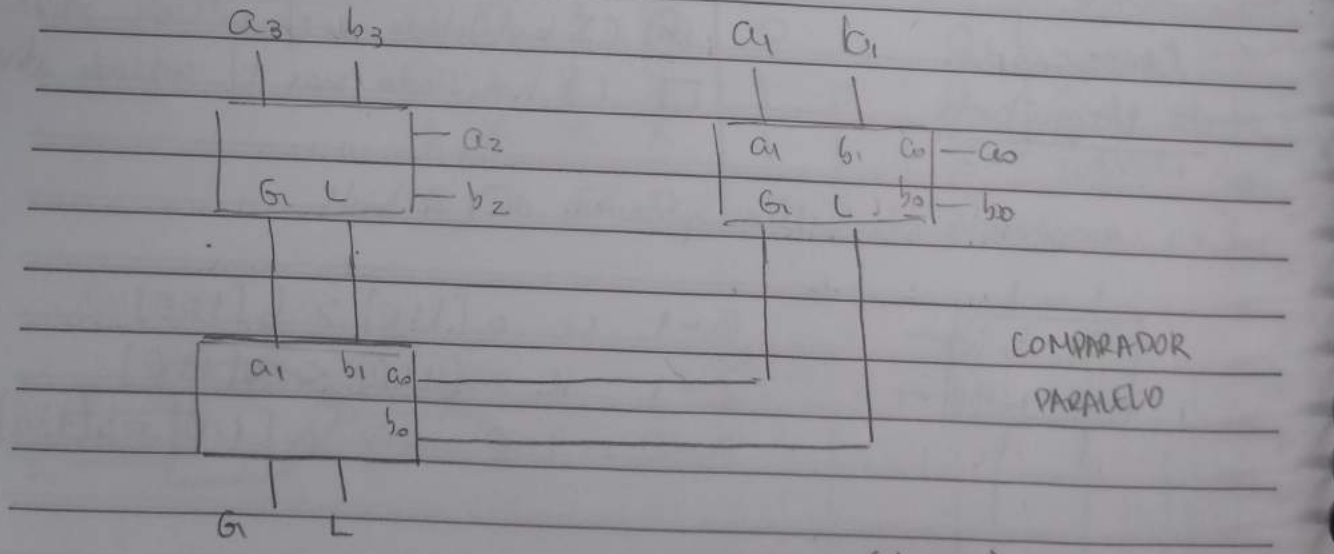
a_0b_0 \ a_1b_1	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	0	1
10	1	0	1	1

Saída $L = \bar{a}_1b_1 + b_1\bar{a}_0b_0 + \bar{a}_1\bar{a}_0b_0$

a_0b_0 \ a_1b_1	00	01	11	10
00	0	1	0	0
01	1	1	1	0
11	0	1	0	0
10	0	1	0	0

Cascateamento

Ex: Comparador de 4 bits



COMPARADOR PARALELO

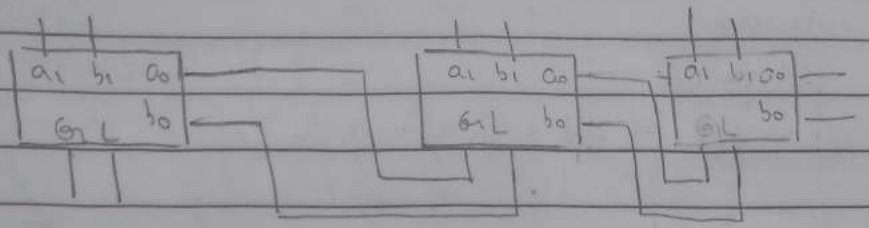
O(logN)

S T Q Q S S D

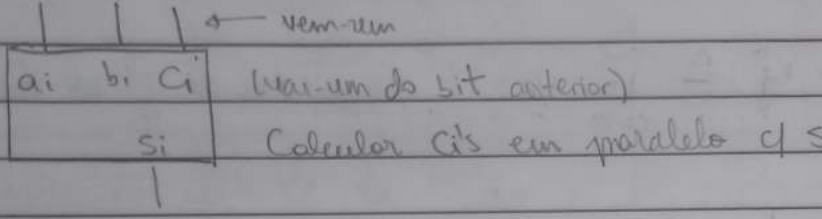
__/__/__

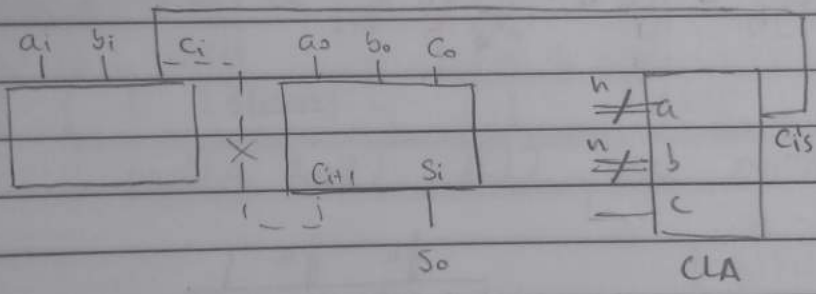
COMPARADOR SÉRIE

O(N)



Preditor de vai-um (Carry Look Ahead - CLA)





Eliminação o atraso de propagação do C_{i+1}

Definições:

$g_i \triangleq a_i \cdot b_i$: gerado de vai-um (vai um gerado na coluna i)

$p_i \triangleq a_i \oplus b_i$: propagador de vai-um (vai um anterior pode ser propagado pl a coluna seguinte).

podia ser OR, mas aproveitar ⊕ do $S_i = a_i \oplus b_i \oplus C_i$

∴ Recorrendo C_{i+1}:

$$C_{i+1} = g_i + p_i \cdot C_i$$

$C_{i+1} = 1$ se é gerado na coluna i ou $C_i = 1$ e propagado pl coluna i+1

spiral

__/__/__

S T Q Q S S D

Recursão

C_0: entrada

$C_1 = g_0 + p_0 \cdot C_0$

$C_2 = g_1 + p_1 C_1 = g_1 + p_1 g_0 + p_1 p_0 C_0$

⋮

$C_4 = g_3 + g_2 \cdot p_3 + g_1 \cdot p_2 \cdot p_3 + g_0 \cdot p_1 p_2 p_3 + C_0 p_0 p_1 p_2 p_3$

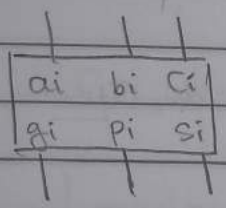
CIRCUITO de 2 NÍVEIS DE ATRASO AND-OR

$$C_n = g_0^{n-1} + p_0^{n-1} \cdot i_0$$

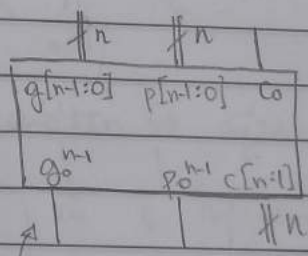
com $p_j^k \triangleq \prod_{i=j}^{k} p_i$ (propagador de j a K)

com $g_j^k = g_k + \sum_{i=j}^{k-1} g_i \, p_{i+1}^k$

Somador Rápido com CLA (Fast Adder)



calcula o bit da soma e o p_i e g_i da coluna.



calcula os vai-uns e o p e g "geral"

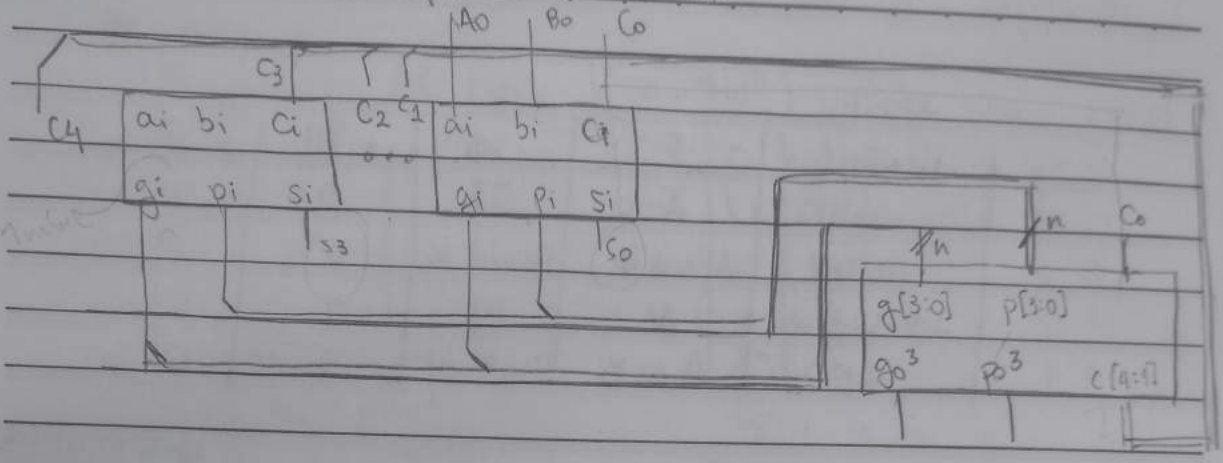
spiral

S T Q Q S S D

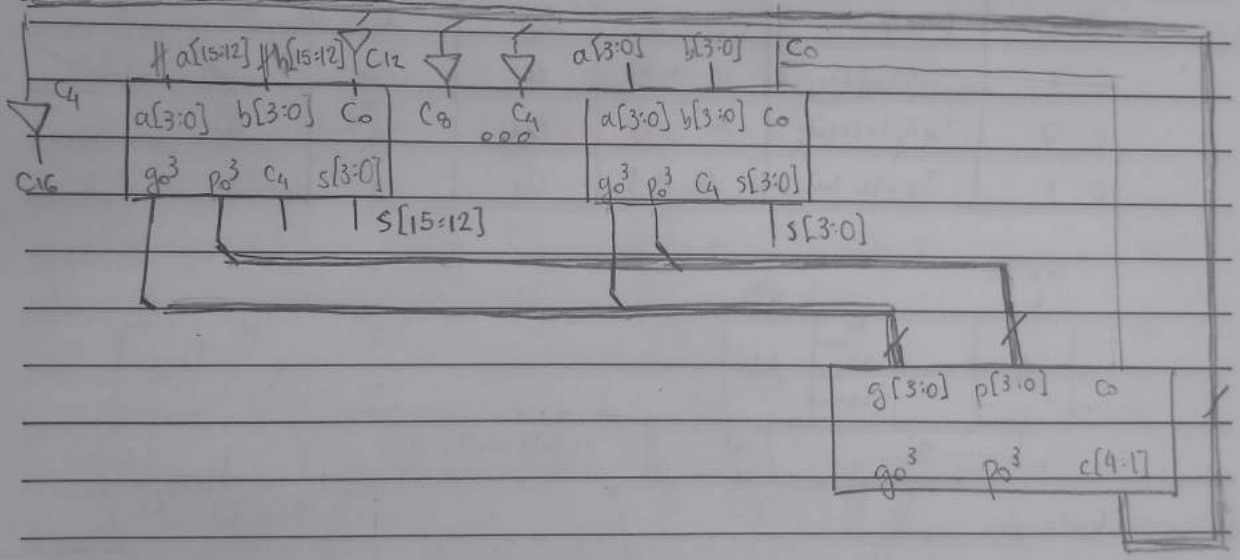
gi e pi independem de ci

__/__/__

Ex: somador rápido de 4 bits

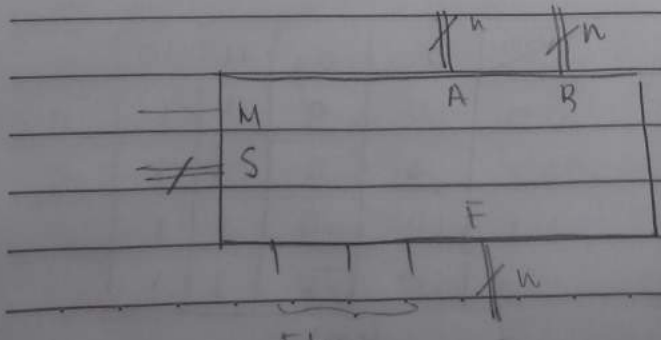


CASCATEAMENTO c/ CLA (Ex: 16 bits)



ULA: Unidade Lógica Aritmética (Gajski)

↳ preparação LAB4



M: define modo de operação (Aritmético ou lógico)

S: bits de seleção da operação

A, B: operandos (n bits)

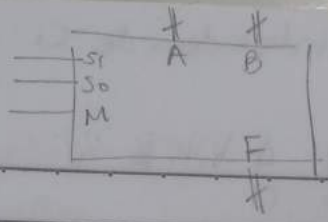
F: saída (n bits)

Flags: Cu (vai-um), Bo, ...

spiral

__/__/__

Exemplo:



S T Q Q S S D

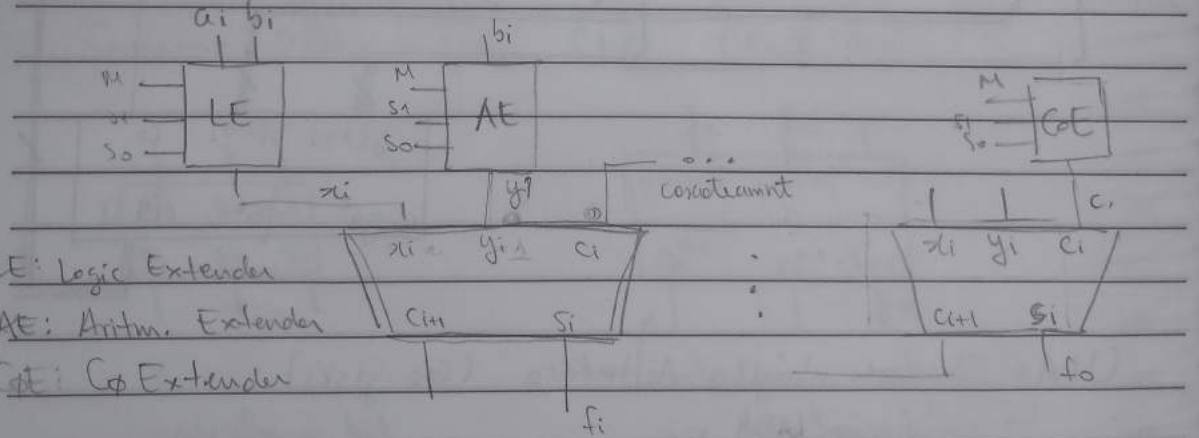
M=0 modo lógico ("bit wise")

S_1 S_0	Função	F	x_i	y_i	C_0
0 0	Complemento	$\overline{A}$	$\overline{a_i}$	0	0
0 1	AND	A and B	a_i and b_i	0	0
1 0	Identidade	A	a_i	0	0
1 1	OR	A or B	a_i or b_i	0	0

M=1 modo aritmético

soma 111111 = (-1)$_{c2}$

S_1 S_0	Função	F	x_i	y_i	C_0
0 0	Decrem.	A − 1	a_i	1	0
0 1	Soma	A + B	a_i	b_i	0
1 0	Subtração	A − B	a_i	$\overline{b_i}$	1
1 1	Increm.	A + 1	a_i	0	1



LE: Logic Extender

AE: Aritm. Extender

CoE: Co Extender

Projeto de LE

S_1S_0 \ a_ib_i	0 0	0 1	1 1	1 0
0 0	1	1	0	0
0 1	0	0	1	0
1 1	0	1	1	1
1 0	0	0	1	1

M = 0

S_1S_0 \ a_ib_i	0 0	0 1	1 1	1 0
0 0	0	0	1	1
0 1	0	0	1	1
1 1	0	0	1	1
1 0	0	0	1	1

M = 1

spiral

S T Q Q S S D

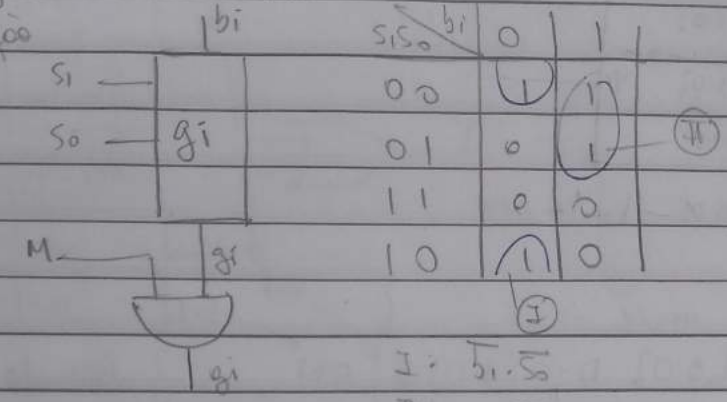
__/__/__

I: Mai II: ai S_1 III: $\overline{M}a_i\overline{S_1}\overline{S_0}$ IV $\overline{M}b_iS_1S_0$

V: $a_ib_iS_0$

Projeto do AE (y_i)

Simplificação



S_1S_0 \ b_i	0	1
0 0	(1)	(1)
0 1	0	1 — II
1 1	0	0
1 0	(1)	0

I

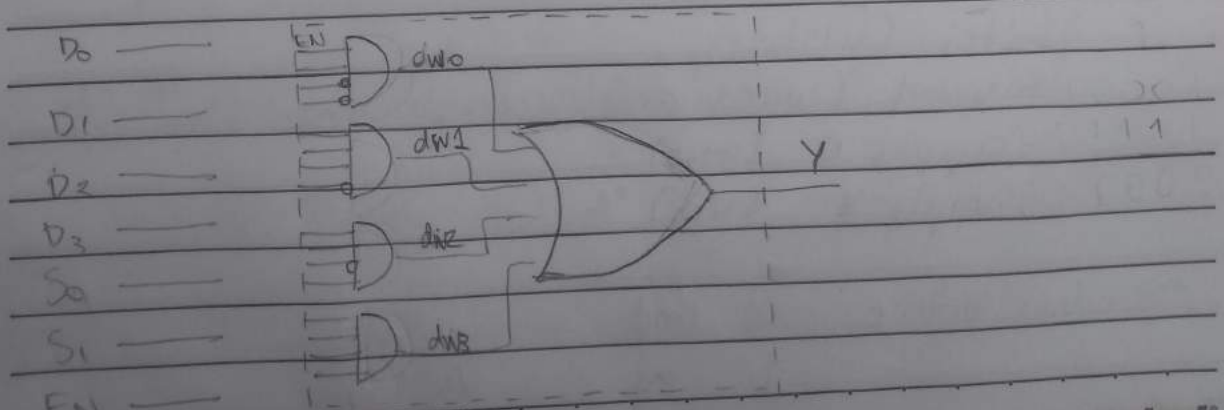
I: $\overline{S_1}\cdot\overline{S_0}$

II: $b_i\cdot\overline{S_1}$

VERILOG: Linguagem de Descrição de Hardware (HDL)

Estrutura: module (parâmetro 1, parâmetro 2, ..., parâmetro n);
// descrição
/* vários comentários sobre ele dig */
end module

Exemplo: Multiplexador 4:1 c/ enable

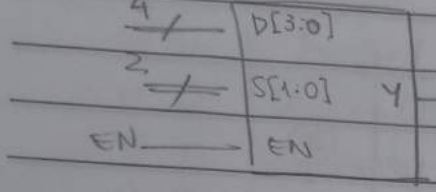


spirati

__/__/__

S T Q Q S S D

Por "assign" (atribui uma expressão ou módulo à um fio (net)).



mux_4_1_xxx

```
module mux_4_1_ ( bus  bit
input [3:0] D; [1:0] S, EN,        } fios do tipo "wire"
output Y);

wire [3:0] dw;
assign dw[0] = EN & D[0] & ~S[1] & ~S[0];
assign dw[1] = EN & D[1] & ~S[1] & S[0];
      :
assign dw[3] = ...          S[1] & S[0];

assign Y = dw[0] | dw[1] | dw[2] | dw[3]   (OR)
end module
```

descrição muito próxima do hardware

wire ⟺ net

outros tipos de net:
- tri (tristate)
- wand (wired-and - open drain)
- supply 0 (GND)
- supply 1 (VDD)

~ operadores bitwise: & and ~ not
| or ^ xor

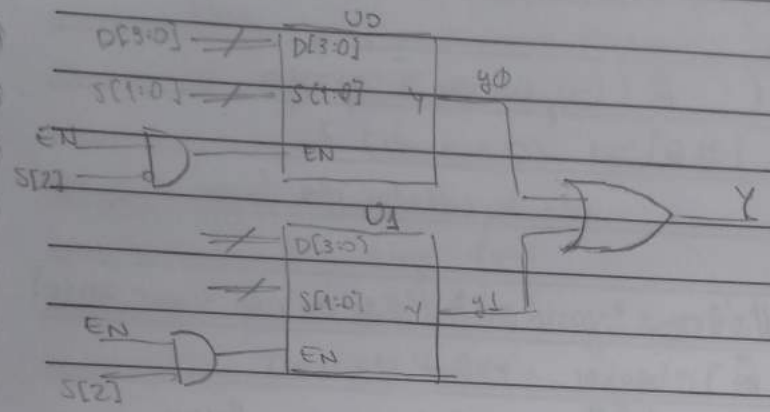
spiral

S T Q Q S S D

Precedentes: ~ → & → |

__/__/__

(*) USAR PARÊNTESES!



Usando um módulo em outro → instanciação!

Ex: module mux_8_1_assign

```
input [7:0] D, [2:0] S, EN
output Y);                    label (opcional)
wire en0, en1, y0, y1;
mux_4_1_assign U0 (D[3:0], S[1:0], en0, y0);
mux_4_1_assign U1 (D[7:4], S[1:0], en1, y1);
assign en0 = EN & (~S[2]);
assign en1 = EN & ( S[2]);
assign Y = y0 | y1;
end module
```

(*) Tudo funciona em paralelo! Não é sequencial.

Descrição "Não Procedural" vistas acima.

spiral

__/__/__

S T Q Q S S D

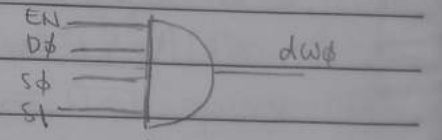
Ex. de módulo procedural

descrição FUNCIONAL (Behavior)

```
module mux_8_1_CG1 (  // CG1: curto e grosso
    input [7:0] D, [2:0] S,
    input EN,                    lista de entradas que fazem bloco ser executado
    output Y);
    reg yr ; // reg = "variável", pode ou não virar wire!
    always @ (*) begin   * vem do input
        yr = EN & D[S] ;   // dentro do always
    end                    // somente reg's podem ser
    assign Y = yr ;        // alterados
end module
```

Outros exemplos de declarações procedurais

mux usando for:




```
reg di, yr ;
integer i

always @ (*) begin
    yr = 0 ;
    for (i=0 ; i<=7 ; i=i+1) begin      i=1 and i=5
        di = EN & D[i] & ( i[1:0] == S[1:0] ) ;
        yr = yr | di ;
    end // for
end // always
assign Y = yr
```

spiral

S T Q Q S S D

__/__/__

mux usando if/else e cascateamento

```
reg yr;                          ← sensitivity list
always @ (*) begin
    if (EN==0) yr = 0;
    else begin                   constante de 3 bits binários
        case (s)
        3'b000: yr = D[0];
        3'b001: yr = D[1];
            :
        3'b111: yr = D[7];
        endcase
    end // else
end // always
```

Em outras bases: 3'b101 → 5 em binário
3'd5 → 5 em decimal
3'h5 → 5 em hexa

Usando Verilog p/ descrever uma SIMULAÇÃO.

Test bench = módulo de teste

```
module mux_4_1_1TB( );  // sem E/S
                        // sinais gerados
reg [3:0] D, [1:0] S;   // na simulação
reg EN;
integer i;
```

descrição do circuito:

```
wire Y
mux_4_1_assign UUT (D,S,EN,Y);
// geração dos sinais
```

spiral

used in test bench

S T Q Q S S D

```
initial begin // roda 1 vez
    for (i=0; i<128; i=i+1) begin   // i varre todas possíveis combinações
        D[3:0] = i[3:0];
        S[1:0] = i[5:4];
        EN     = i[6];
        #10; // avança na simulação 10 unidades de tempo
    end // for
end // initial
```

`timescale 1ns/1ps

diretiva que define unidade de tempo (1ns) e precisão (1ps) para simulação.

Dois tipos de atribuição em simulação:

```
x = y; // "blocking", a simulação altera x neste ponto
z <= y; // "non-blocking", simulação muda z só no fim do
        // bloco.
```

usadas p/ controlar atrasos na simulação (apenas em blocos "initial").

Cuidados:

- Verilog é case sensitive (maiúsc ≠ minusc)
- Variáveis não declaradas são assumidas como wire

Erros comuns: i0 ≠ iO

c0 ≠ CO

c1 ≠ c]

Arrays

```
reg [7:0] byte1, recent[1:5], mem1[0:255]
```

byte1: 8bit vector

recent and mem1: arrays containing 5 or 256 8bits vetors

spiral

S T Q Q S S D

__/__/__

(*) Não dá p/ acessar valores dentro de array
↳ copiar valor em reg e depois acessar

parameter identifier = cte
↳ local

Verilog built-in gates

and	xor	bufif0	else: z
nand	xnor	bufif1	
or	buf	notif0	
nor	not	notif1	

Conditional operator: selA highest priority

```
module Vtbytarul (A,B,C,selA,selB,selC,Z);
  input [7:0] A, B, C;
  input selA; selB; selC
  output [7:0] Z                          // como se fosse:
  assign Z = A? selA : (                  // if else
             B? selB : (                  // if else
             C? selC : 8'b0));
endmodule
```

P2
- Listas 5, 6 e 7
- Exps 2 e 3

spiral

__/__/__

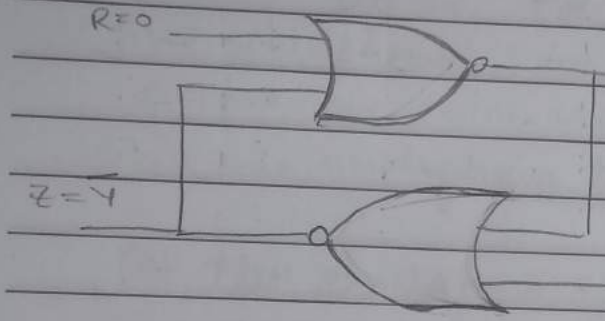
S T Q Q S S D

Cap. 7 - Circuitos Sequenciais (dinâmicos)

Saídas dependem das entradas e do estado atual → memória

Memória com portas lógicas

① Latch RS positivo

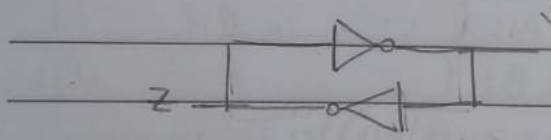


p/ R=0 e S=0

$\begin{cases} Y = \overline{Z} \\ Z = \overline{Y} \end{cases} \Leftrightarrow \begin{cases} Y^{K+1} = Y^K \\ Z^{K+1} = Z^K \end{cases}$

saídas se mantém no estado atual e $Y = \overline{Z}$

Nesta condição, circuito é equivalente a



p/ R=1 e S=0: Y→0 Z→1

p/ R=0 e S=1: Y→1 Z→0

p/ R=1 e S=1: Y→0 Z→0

saídas não memorizáveis

Problema: voltando R:1→0 e S:1→0 valores de Y e Z ficam indefinidos. Na prática Y final é 0 ou 1 (e $Z = \overline{Y}$) dependendo dos atrasos relativos das portas

spiral

S T Q Q S S D

__/__/__

Representação mais Comum

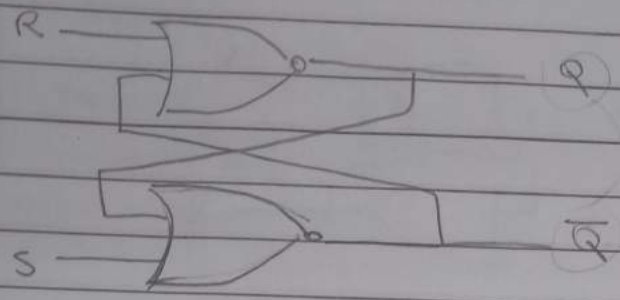
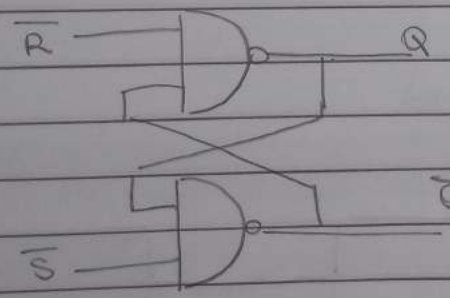


Tabela Característica

R	S	Q^{K+1}	$\overline{Q}^{K+1}$	
0	0	Q^K	$\overline{Q}^K$	(mantém)
0	1	1	0	(set Q)
1	0	0	1	(rest Q)
1	1	0	0	(não usado)

Por dualidade:

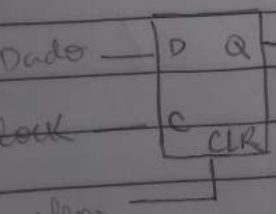
② Latch RS Negativo



$\overline{R}$	$\overline{S}$	Q^{K+1}	$\overline{Q}^{K+1}$	
1	1	Q^K	$\overline{Q}^K$	
1	0	1	0	(set Q)
0	1	0	1	(rest Q)
0	0	1	1	(não usado)

Melhor: eliminando entrada ñ usada

③ Latch tipo D com clear assíncrono



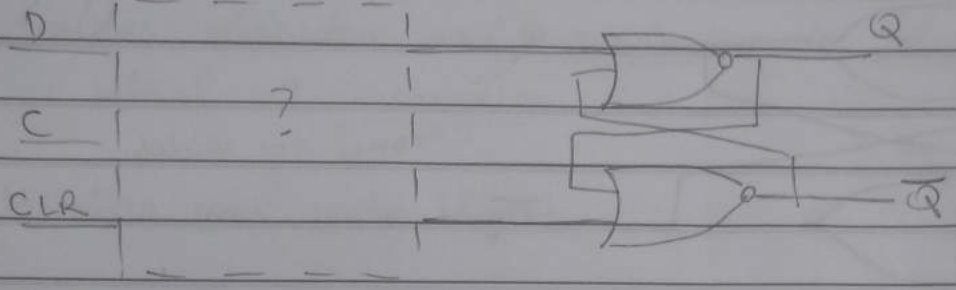
CLR	C	D	Q^{K+1}	
1	x	x	0	(clear assíncrono)
0	0	x	Q^K	
0	1	0	0	cópia D
0	1	1	1	

spiral

__/__/__

S T Q Q S S D

Construindo com latch RS positivo



R

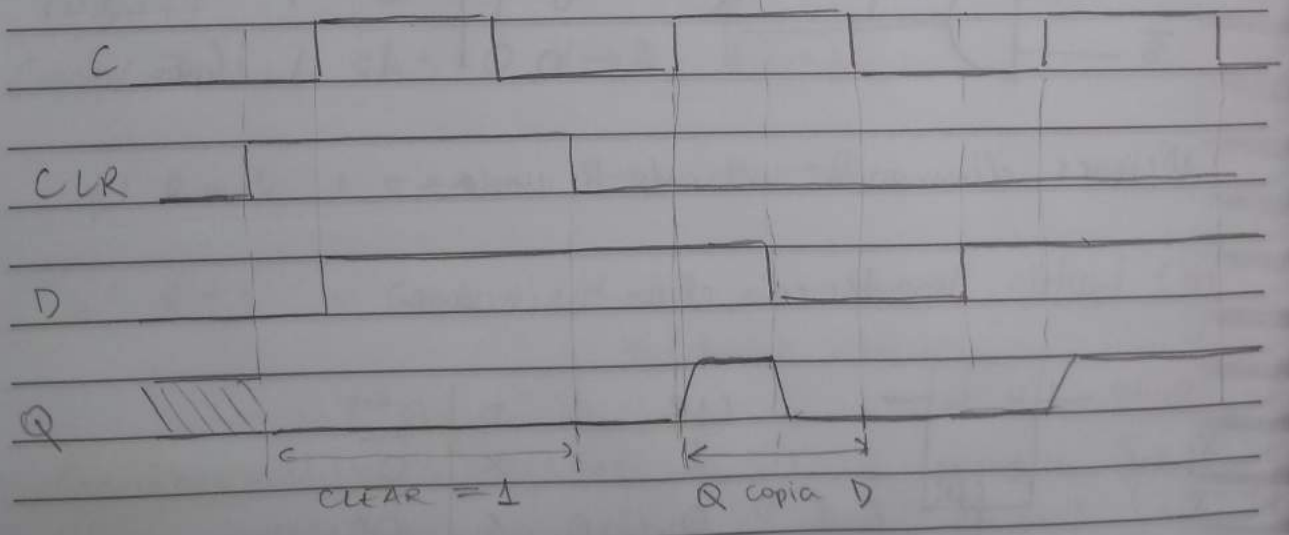
CD \ CLR	0	1
0 0	0	1
0 1	0	1
1 1	0	1
1 0	1	1

S

CD \ CLR	0	1
0 0	0	0
0 1	0	0
1 1	1	0
1 0	0	0

$R = CLR + C.\overline{D}$ $S = \overline{CLR}.C.D$

Exemplo de funcionamento (no tempo)

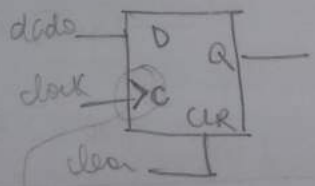


spiral

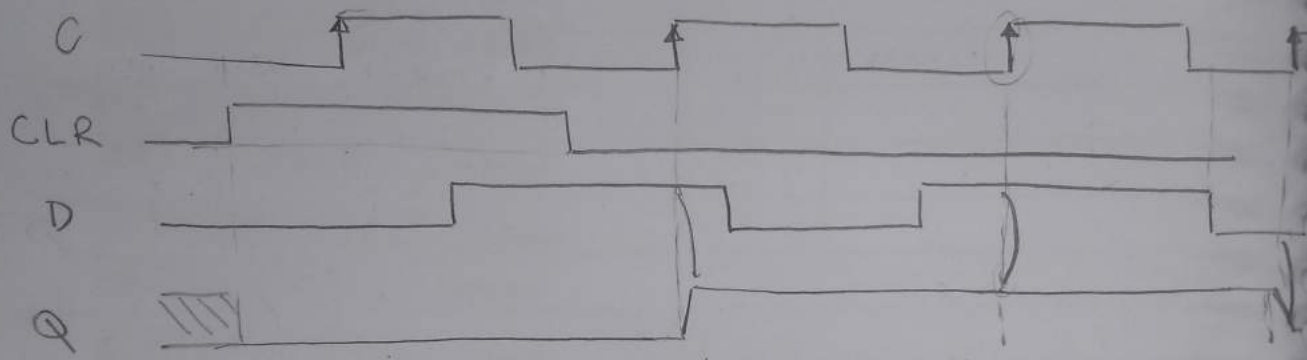
Quando C=1, Q varia com D temos Q sensível ao nível de C.

Solução: flip-flop (sensível à borda de C)

Ex: flip flop tipo D

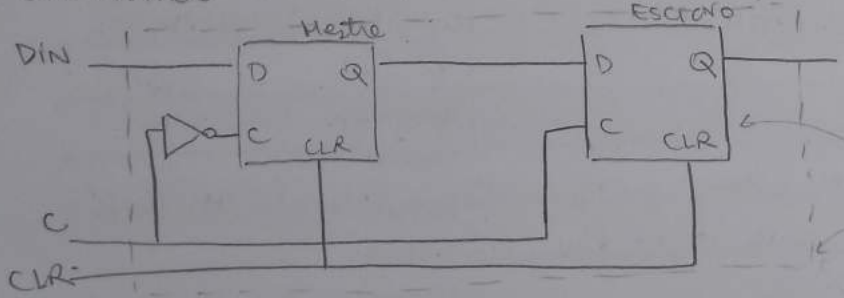


CLR	C	D	Q	
1	x	x	0	(clear assíncrono)
0	↓	0	0	Q copia D só no instante de borda!
0	↓	1	1	
0	0	x	Q	Q se mantem se C constante
0	1	x	Q	



Tempo discretizado nos instantes de borda.

Construindo o f.f. tipo D c/ latch tipo D (mestre-escravo)

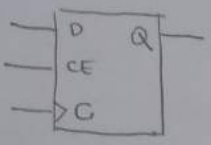


clock = 0 → { Q_{IN} copia D_{IN} ; Q_{OUT} mantém

clock = 1 → { Q_{IN} mantém ; Q_{OUT} copia Q_{IN}

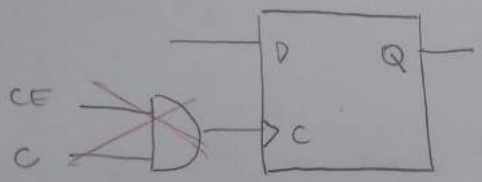
∴ somente na transição do clock de 0 para 1 é que Q_{OUT} é atualizado com D_{IN}.

Flip Flop D com clock enable (CE)



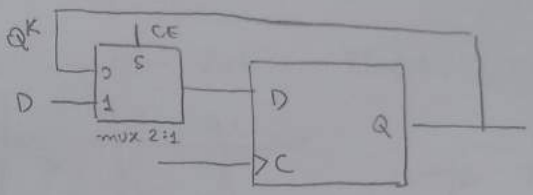
CE	C	D	Q^{k+1}	
0	↓	X	Q^k	→ mantém saída salva
1	↓	0	0	altera memória conforme dado D
1	↓	1	1	

Construindo com ff sem CE

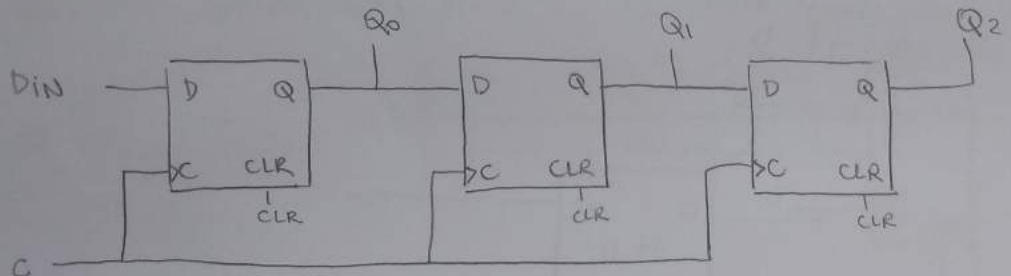


(*) Não se mexe no clock

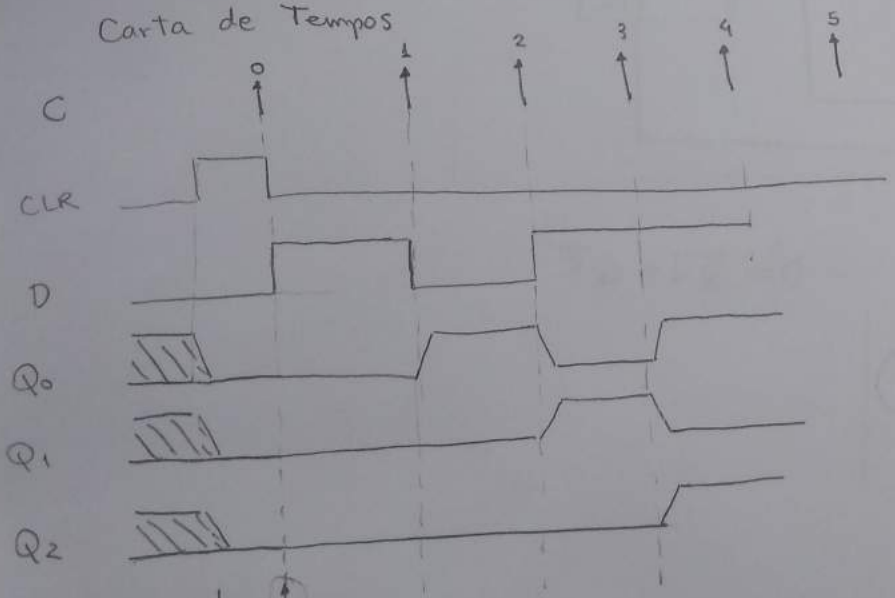
2ª Tentativa



Aplicações: LINHA DE ATRASO. Conversor Série-Paralelo



Carta de Tempos

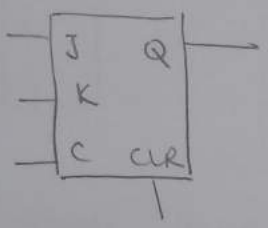


ignorado!
CLR = 1

Dado válido: sempre o imediatamente anterior à borda!

Outros tipos de flip-flop

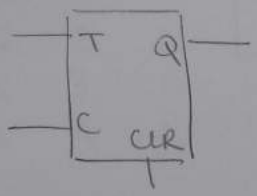
Flip Flop JK



set reset

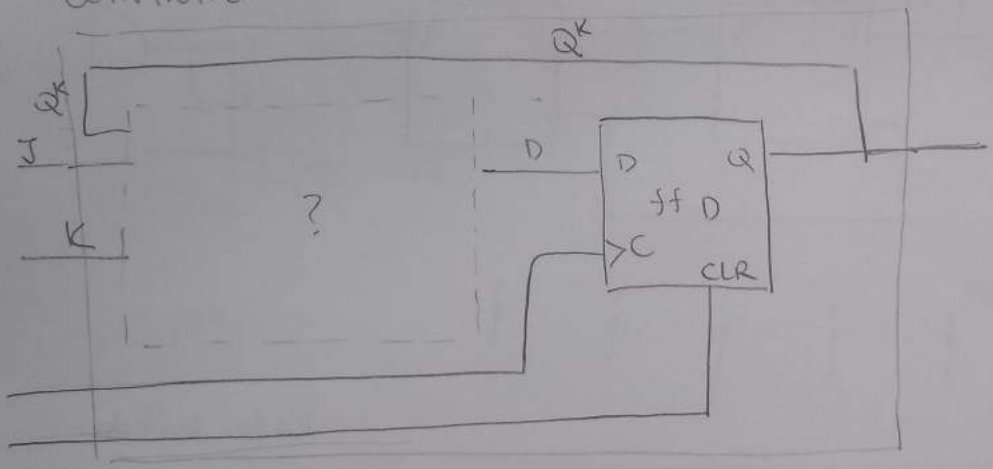
CLR	C	J	K	Q	
1	x	x	x	0	(clear assíncrono)
0	↓	0	0	Q	(mantém)
0	↓	0	1	0	(reset)
0	↓	1	0	1	(set)
0	↓	1	1	$\overline{Q}$	(troca)

Flip Flop tipo T



CLR	C	T	Q	
1	x	x	0	(clear)
0	↓	0	Q	(mantém)
0	↓	1	$\overline{Q}$	(troca)

Construindo JK com ff D

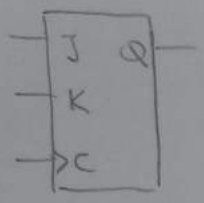


(D)

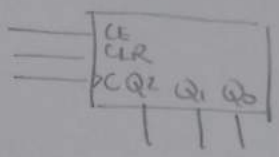
$\overline{Q^K}$ \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$D = \overline{Q}J + Q\overline{K}$$

Remember JK



C	J	K	Q	
↑	0	0	Q	mantém
↑	0	1	0	
↑	1	0	1	
↑	1	1	$\overline{Q}$	troca

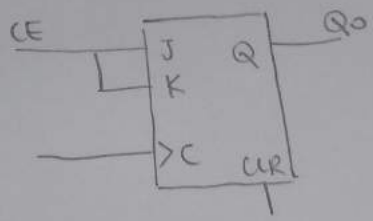


CONTADOR SÍNCRONO (em binário 3 bits)

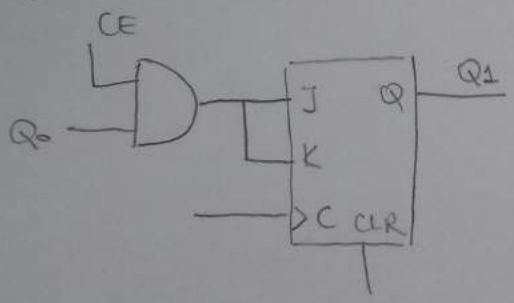
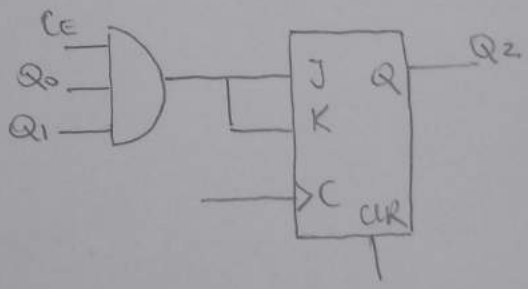
CE	C	Q_2	Q_1	Q_0
1	0	0	0	0
1	1	0	0	1
1	2	0	1	0
1	3	0	1	1
1	4	1	0	0

Usando JK

Q_0: troca a cada clock (se CE = 1)



Q_1: troca quando $Q_0 = 1$ e CE = 1



Q_2: troca quando $Q_0 Q_1 E = 1$