

Apresentação da Disciplina

Revisão Sistemas de numeração

- Decimal
 - dez símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9
 - regras de posicionamento para valores fora da representação
 - Operações matemáticas
 - Adição e o tamanho da nossa ULA
- binário
 - origem do sistema
 - Voltagem: 0v ou 5v (5 volts nos circuitos digitais do tipo TTL. Com outras tecnologias, CMOS, por exemplo, os valores são outros)
 - Apenas dois símbolos, 0 e 1
 - 0 => 0v
 - 1 => 5v
 - Mesmas regras matemáticas do sistema decimal
- octal
 - Oito símbolos: 0, 1, 2, 3, 4, 5, 6 e 7
 - usado para facilitar o uso de binário por nós
 - Cada 3 bits equivale a um símbolo octal
- hexadecimal
 - São 16 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F
 - Também usado para facilitar nosso uso
 - Cada 4 bits equivale a um em HEXA
 - Oito bits, ou seja, UM BYTE equivale a DOIS HEXA
 - Conversão de hexa para binário e de binário para hexa
- formas de representação de número negativo
 - Decimal:
 - Na verdade o sistema decimal não tem apenas 10 símbolos (0 a 9)
 - Também tem outros dois: + e -
 - Para um número ser negativo, basta inserir o sinal de - no início
 - Para ser positivo, basta colocar o sinal de + (ou não colocar nada. A ausência de sinal é tratada como valor positivo)
 - Em binário:
 - Não há como ter outros símbolos além do 0 e do 1.
 - São apenas dois e deve ser capaz de representar com estes dois
 - Técnicas:
 - sinal magnitude
 - o bit mais significativo determina o sinal
 - se em 0 é positivo, se em 1 é negativo
 - os demais bits determinam o valor
 - vantagens e desvantagens
 - (VANT) É mais fácil de nós, humanos, entendermos. Exemplo 8 bits:
 - +2 => **0** 000 0010
 - -2 => **1** 000 0010
 - (DESV) mais caro para implementar (no sentido de gastar mais componentes eletrônicos)
 - (DESV) há uma dupla representação do zero:
 - 0 000 0000 => Isto seria o 0 ou +0
 - 1 000 0000 => e isto seria um -0 (????)
 - usado para pontos flutuantes, mas não é para números inteiros (nos processadores PC)

- complemento de 2
 - vantagens: facilidade de implementar em *hardware* (será visto)
 - ganha-se um decimal a mais
 - exemplos em 8 bits: 2 e -2
 - sem sinal: 0 a 255
 - com sinal: -128 a 127 (não é de -127 como seria no sinal magnitude)
 - 2 = 0000 0010
 - -2 = 1111 1110
 - -2 é também 254
 - exemplos em 16 bits: 2 e -2
 - sem sinal: 0 a 65535
 - com sinal: -32768 a 32767
 - 2 = 0000 0000 0000 0010
 - -2 = 1111 1111 1111 1110
 - -2 é também 65534
 - exemplos em 32 bits: 2 e -2
 - sem sinal: 0 a 4294967295
 - com sinal: -2147483648 a 2147483647
 - 2 = 0000 0000 0000 0000 0000 0000 0010
 - -2 = 1111 1111 1111 1111 1111 1111 1110
 - -2 é também 4294967294
 - Como o sistema binário não tem um símbolo exclusivo para sinal, podem haver duplo sentido na representação.
 - Afinal, o valor em binário 1111 1111 é o inteiro positivo 255 ou é o inteiro negativo -1?
 - Depende. Muitos processadores (PC por exemplo) permitem que o programador decida se quer ou não usar números negativos (será visto em Arquitetura II)
 - Na linguagem C, por exemplo, ao definir uma variável do tipo char (8 bits), por padrão é um inteiro COM SINAL. Então o binário 1111 1111 atribuído a esta variável seria considerado -1
 - Mas se ao definir a variável se colocou um *unsigned* antes, ali será instruído o hardware a não considerar sinal.
 - Desta forma o valor 1111 1111 será interpretado como 255
 - Você pode testar isso com este simples programinha em C (não é necessário ter conhecimentos da linguagem C para realizar Arq I. É apenas um exemplo):

```
int main()
{
    char a;
    unsigned char b;
    b = 255; // mesmo que 1111 1111
    a = 255; // mesmo que 1111 1111
    if (a < 0) {
        printf("Variavel a eh negativa\n");
    } else {
        printf("Variavel a eh negativa\n");
    }
    /* O if acima irá considerar o a como sendo -1 e não 255 */

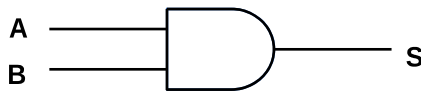
    if (b < 0) {
        printf("Variavel b eh negativa\n");
    } else {
        printf("Variavel b eh negativa\n");
    }
    /* Já este outro if irá considerar o b como 255 e não -1 */
}
```

Portas Lógicas

- Operações envolvendo bits
- Apenas dois valores: Verdadeiro ou Falso (1 ou 0)
- As principais são
 - NOT (Complemento): a entrada é um BIT e o mesmo é invertido
 - Usa-se um traço sobre a informação para simbolizar
 - ou ~ ou apóstrofe
 - Exemplo: $\sim A$ ou A' ou \bar{A}



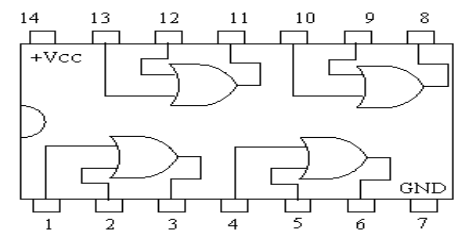
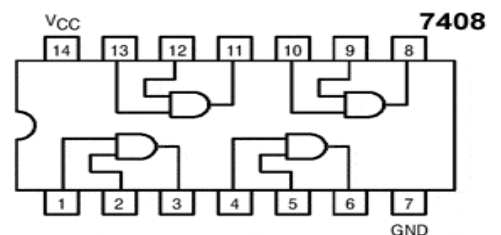
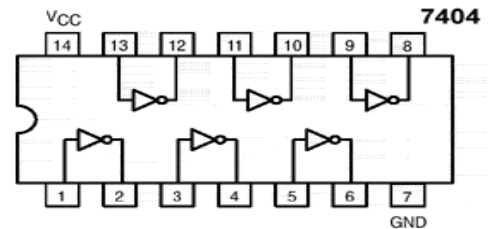
- AND: compara dois bits, tendo uma única saída.
 - Trata-se de uma multiplicação binária
 - representado pelos símbolos '.' ou \wedge
 - Exemplo: $A.B$ ou $A \wedge B$ (ou apenas AB)



- OR: compara dois bits, tendo uma única saída
 - Trata-se de uma soma binária
 - não confunda soma binária com soma numérica
 - representado pelos símbolos '+' ou \vee
 - Exemplo: $A+B$ ou $A \vee B$



- Tabelas verdade



7432

Operação NOT

A	S
0	1
1	0

$$S = \bar{A}$$

Operação AND

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

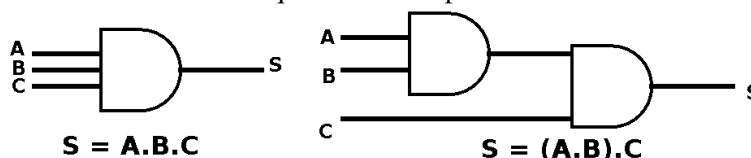
$$S = A.B$$

Operação OR

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

$$S = A+B$$

- Portas lógicas podem ter DUAS ou mais entradas
 - Uma porta com três entradas equivale a duas portas com duas entradas



- Também existem as portas NAND, NOR, XOR e XNOR

- Elas são portas montadas a partir das básicas:

- NAND: AND com um NOT na saída

- NOR: OR com um NOT na saída

- XOR: Este é mais complexo. Tem um símbolo próprio para ele \oplus (em algumas linguagens de programação é usado o ^ como símbolo).

- Para duas entradas: a saída será 0 quando as entradas forem iguais OU

- Para N entradas: a saída será ZERO se a quantidade de entradas em 1 for ZERO ou par.

- Exemplo: A B C em 0 1 1. Tem dois 1's na entrada, ou seja, a quantidade de bits em 1 é PAR. Logo a saída será 0.

Operação XOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



Porta XOR com 2 entradas

- Portas NAND e NOR são chamadas de portas universais
 - Qualquer circuito ou qualquer outra porta pode ser feito usando-se apenas NAND ou apenas NOR.
 - Significa que se fossem produzidos circuitos integrados apenas com NAND, qualquer computador poderia ser montado

Expressões booleanas

- um circuito digital é formado por diversas portas lógicas.
- Para um determinado conjunto de entradas, espera-se uma determinada saída

Obtendo a tabela verdade a partir de uma expressão booleana

- Para avaliar uma expressão:
 - primeiro avalia-se o NOT
 - depois o AND (multiplicação lógica)
 - e, por último, o OR (soma lógica)
 - parêntese alteram a precedência

- Exemplo: $A.B + \bar{C} + A.C$ *1

A	B	C	\bar{C}	A.B	A.C	$A.B + \bar{C}$	$(A.B + \bar{C}) + A.C$
0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	1	1	0	1	1
1	1	1	0	1	1	1	1

Tabela Verdade 1 (obtida pela Figura 1)

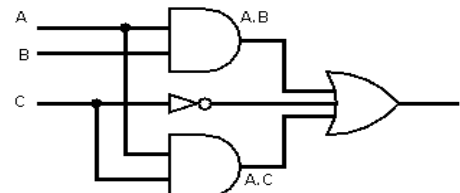
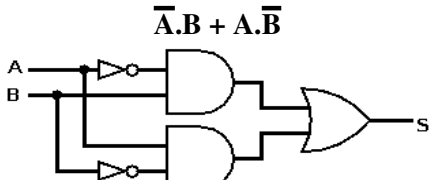
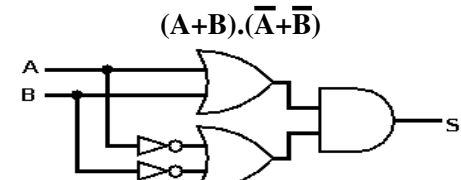


Figura 1 ($A.B + \bar{C} + A.C$)

Obtendo a expressão booleana a partir de uma tabela verdade

- Para se obter a expressão booleana a partir de uma tabela verdade qualquer, existem duas técnicas:
 - Soma dos produtos
 - monta-se a partir de operações AND (produto) para depois somar todas (Soma)
 - Nesta se escreve a expressão booleana correspondente para cada saída com o resultado em 1
 - Saídas que resultaram em ZERO são ignoradas
 - Se, para a saída que tem 1 uma entrada é 0, ela aparece complementada na expressão.
 - Ao final soma-se todas estas expressões e tem-se a expressão booleana final (que, geralmente, precisa ser SIMPLIFICADA)
 - Produto das somas
 - Monta-se a partir de operações OR (somas) para depois realizar a multiplicação (AND)
 - É ao contrário da soma dos produtos
 - Saídas que resultaram em 1 são ignoradas.
 - Apenas as saídas em ZERO são consideradas
 - Nestas se escreve a expressão. Se uma entrada for UM, então ele aparece complementada (importante: é realmente AO CONTRÁRIO da soma dos produtos)

Exemplo 1: montando a expressão booleana do XOR

A	B	S	Soma dos produtos	Produto das somas
0	0	0	-	$A+B$
0	1	1	$\bar{A}.B$	-
1	0	1	$A.\bar{B}$	-
1	1	0	-	$\bar{A}+\bar{B}$
			$\bar{A}.B + A.\bar{B}$ 	$(A+B).(\bar{A}+\bar{B})$ 

Exemplo 2: usando a tabela verdade montada para a expressão booleana $A.B + \bar{C} + A.C$ da página anterior

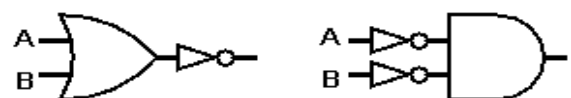
A	B	C	S	Soma dos produtos	Produto das somas
0	0	0	1	$\bar{A}.\bar{B}.\bar{C}$	-
0	0	1	0	-	$A+B+\bar{C}$
0	1	0	1	$\bar{A}.B.\bar{C}$	-
0	1	1	0	-	$A+\bar{B}+\bar{C}$
1	0	0	1	$A.\bar{B}.\bar{C}$	-
1	0	1	1	$A.\bar{B}.C$	-
1	1	0	1	$A.B.\bar{C}$	-
1	1	1	1	$A.B.C$	-
				$(2) \quad S = (\bar{A}.\bar{B}.\bar{C})+(\bar{A}.B.\bar{C})+(A.\bar{B}.\bar{C})+(A.\bar{B}.C)+(A.B.\bar{C})+(A.B.C)$	$(3) \quad S = (A+B+\bar{C}).(A+\bar{B}+\bar{C})$

- Apesar de completamente diferentes, ambas as expressões booleanas significam a mesma coisa.
- E ainda, neste caso a tabela verdade usada foi a montada a partir da expressão: $S = A.B + \bar{C} + A.C$ (1)
-> exemplo da Figura 1

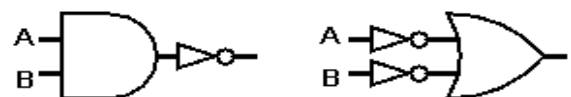
- Então :
 - $A.B + \bar{C} + A.C == (A+B+\bar{C}).(A+\bar{B}+\bar{C}) == (\bar{A}.\bar{B}.\bar{C}) + (\bar{A}.B.\bar{C}) + (A.\bar{B}.\bar{C}) + (A.\bar{B}.C) + (A.B.\bar{C}) + (A.B.C)$
fica como desafio para o leitor implementar a tabela verdade de todas e verificar que a saída é a mesma. No caso a (1) já está fornecida. Falta montar a (2) e a (3).
- Qual destas expressões booleanas que representa a saída que eu quero, vale mais a pena de ser implementada em hardware?
 - Resposta: aquela que representa o menor consumo de recursos. Apenas com estas 3, seria a primeira por usar poucas portas lógicas (um NOT, Dois AND de duas entradas e um OR de três entradas)
 - OU: Nenhuma delas, pois talvez exista ainda uma outra expressão booleana que use ainda menos recursos.
- Desta forma é necessário, sempre, tentar minimizar uma expressão booleana, simplificando-a.

Simplificação de Expressões Booleanas

- As expressões booleanas podem ser simplificadas por postulados, ou seja, regras de matemáticas de simplificação.
- Tome-se cuidado que a “soma” binária (OR) e a “multiplicação” binária (AND) não são as mesmas somas e multiplicação que conhecemos. Por Exemplo:
 - $A+A = 2A$ em nossa soma aritméticas
 - $A+A = A$ nas operações booleanas (o que pode ser facilmente comprovado pela tabela verdade do A fazendo OR com ele mesmo.
 - As simplificações se aplicam a quaisquer entradas. Usou-se X, Y e Z para não confundir com as entradas A, B, C, D... tipicamente usadas
 - Regras para o OR (adição lógica)
 - $X + 0 = X$ -> uma entrada OR 0 sempre resulta na entrada
 - $X + 1 = 1$ -> algo feito um OR com 1 sempre dará 1
 - $X + X = X$ -> Um bit feito um OR com ele mesmo dará o mesmo bit
 - $X + \bar{X} = 1$ -> Um bit feito um OR com ele mesmo negado dará 1
 - Regras para o AND (multiplicação lógica)
 - $X.1 = X$ -> um bit AND 1 dará o mesmo bit
 - $X.0 = 0$ -> um bit AND zero dará sempre 0
 - $X.X = X$ -> Um bit AND ele mesmo dá ele mesmo
 - $X.\bar{X} = 0$ -> um bit AND ele complementado dá sempre ZERO
 - Regra de complementação:
 - $\bar{\bar{X}} = X$ -> um bit complementado DUAS vezes resulta nele mesmo
 - Associativa
 - $X + (Y + Z) = (X + Y) + Z = (X + Z) + Y \dots$
 - $X.(Y.Z) = (X.Y).Z \dots$ -> X ordem na qual se realiza AND ou OR para N entradas não importa.
 - Distributivas
 - $X.(Y + Z) = X.Y + X.Z$
 - $X + (Y.Z) = (X + Y).(X + Z)$
 - Absorção:
 - $X.(X + Y) = X$
 - $X + (X.Y) = X$
 - Teoremas De Morgan
 - $\overline{X + Y} = \bar{X} . \bar{Y}$
 - $\overline{X . Y} = \bar{X} + \bar{Y}$
 - Expressões auxiliares
 - $\bar{\bar{X}} + X.\bar{Y} = \bar{X} + \bar{Y}$
 - $\bar{\bar{X}} + \bar{X}.\bar{Y} = \bar{X}$
 - $X + \bar{\bar{X}} . Y = X + Y$
 - $\bar{\bar{X}} + X . Y = \bar{X} + Y$
 - $X + \bar{\bar{X}} . \bar{Y} = X + \bar{Y}$



Teorema De Morgan numero 1



Teorema De Morgan numero 2

	A	B	C	D	E
1	$\bar{X} = X$	$X + 0 = X$	$X + 1 = 1$	$X + X = X$	$X + \bar{X} = 1$
2	$X.(Y+Z) = X.Y + X.Z$	$X.1 = X$	$X.0 = 0$	$X.X = X$	$X.\bar{X} = 0$
3	$X+(Y.Z) = (X+Y).(X+Z)$	$X.(X+Y) = X$	$X+(X.Y) = X$	$\overline{X+Y} = \bar{X} . \bar{Y}$	$\overline{X . Y} = \bar{X} + \bar{Y}$
4	$\bar{X} + X.\bar{Y} = \bar{X} + \bar{Y}$	$\bar{X} + \bar{X}.\bar{Y} = \bar{X}$	$X + \bar{X} . \bar{Y} = X + \bar{Y}$	$X + \bar{X} . Y = X + Y$	$\bar{X} + X . Y = \bar{X} + Y$

Tabela de referência para expressões de simplificação. Não estão na tabela as regras associativas por serem, digamos, bem óbvias. Cada regra poderá ser referenciada pelo seu número desta tabela. Regra 2D $\Rightarrow X.X = X$

- Usando estas regras é possível reduzir uma expressão booleana (em tempo: existem outras)
 - Exemplo: Simplificando a expressão (2) anterior, que foi obtida através da tabela verdade:

$(\bar{A} . \bar{B} . \bar{C}) + (\bar{A} . B . \bar{C}) + (A . \bar{B} . \bar{C}) + (A . \bar{B} . C) + (A . B . \bar{C}) + (A . B . C)$	usando Associativa para mudar a ordem das expressões, ficando fácil enxergar padrões
$(\bar{A} . \bar{C} . \bar{B}) + (\bar{A} . \bar{C} . B) + (A . \bar{B} . \bar{C}) + (A . \bar{B} . C) + (A . B . \bar{C}) + (A . B . C)$ $\text{----> } \bar{A} . \bar{C} \Rightarrow X \quad \bar{B} \Rightarrow Y \quad B \Rightarrow Z \quad X.Y + X.Z = X.(Y+Z) \quad (2A \text{ ao contrário})$ $\bar{A} . \bar{C} . (B+\bar{B}) + \bar{A} . \bar{B} (C+\bar{C}) + \bar{A} . B (C+\bar{C})$	Isolei o Interpretei o $\bar{A} . \bar{C}$ como sendo X e o \bar{B} como sendo Y e o B como sendo Z. Com isso consegui aplicar a 2A ao contrário da igualdade. Idem para os demais grupos que tinham padrões.
$\bar{A} . \bar{C} . (1) + \bar{A} . \bar{B} (1) + \bar{A} . B (1)$	Segundo 1E, $X + \bar{X} = 1$
$\bar{A} . \bar{C} + \bar{A} . \bar{B} + \bar{A} . B$	Segundo 2B $X.1 = X$
$\bar{A} . \bar{C} + \bar{A} . (\bar{B} + B)$	Decompondo usando 2A novamente
$\frac{\bar{A} . \bar{C} + \bar{A} . (1)}{\bar{A} . \bar{C} + \bar{A}}$	Aplicado 1E em $(\bar{B} + B)$
$\frac{A + \bar{A} . \bar{C}}{A + \bar{C}}$	Por fim, aplicando 4C (usando A para X e C para Y). $X + \bar{X} . \bar{Y} = X + \bar{Y}$. Então $A + \bar{A} . \bar{C} = A + \bar{C}$

- Pode parecer incrível, mas aquela expressão inicial $(A.B + \bar{C} + A.C)$ que teve outras tantas expressões obtidas pela tabela verdade, também pode ser escrita com um simples $A + \bar{C}$!
 - No exemplo de como obter expressões regulares a partir de uma tabela verdade, chegou-se a TRÊS expressões que faziam a mesma coisa:
 - $A.B + \bar{C} + A.C$ (original da Figura 1)
 - $(A+B+\bar{C}).(A+\bar{B}+\bar{C})$ (obtida pela técnica de “produto das somas”)
 - $(\bar{A} . \bar{B} . \bar{C}) + (\bar{A} . B . \bar{C}) + (A . \bar{B} . \bar{C}) + (A . \bar{B} . C) + (A . B . \bar{C}) + (A . B . C)$ (obtida pela técnica de “soma dos produtos”)
 - E, na ocasião se fez a pergunta de qual seria melhor implementada com circuitos lógicos, já tendo dito que “talvez nenhuma”.
 - Amais reduzida é $A + \bar{C}$ depois de fazermos a simplificação
 - sim, isso mesmo, ignora completamente o estado de B.
 Ou seja, o circuito da Figura 1 pode ser simplesmente este da Figura 2
 - Basta fazer a tabela verdade de $A + \bar{C}$ para verificar
- Fica claro, também, que muitos caminhos são possíveis na simplificação.
- Diferentes caminhos levam a expressões mais fáceis ou mais difíceis de serem simplificadas.
- Importante a perícia (e criatividade) para enxergar pontos onde pode ser feita a simplificação

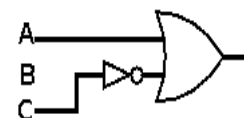


Figura 2

Aula 4 - 16/Março/2017

- Expressões booleanas podem e devem ser simplificadas
 - Para economizar componentes, gastando menos
- Duas formas:
 - regras de simplificação:
 - são muitas, algumas redundantes (derivadas de uma regra maior)
 - requer extrema habilidade para aplicá-las (e uma certa criatividade)
 - se houver a máxima habilidade irá simplificar até uma expressão mínima
 - Lembrando de que podem existir várias expressões mínimas
 - Mapas de Karnaugh
 - Um mecanismo gráfico de simplificação
 - se bem aplicado, sempre reduz para uma das mínimas possíveis
 - Pode ser aplicado para duas, três ou quatro entradas
 - Par mais de quatro entradas o mapa fica muito complexo e se torna oneroso demais (ai valerá mais a pena usar as regras).

Mapas de Karnaugh

- Na verdade não se trata de um mecanismo de simplificação booleana, pois não trabalha sobre uma expressão booleana
- Trabalha diretamente sobre a tabela verdade
- Trata-se, portanto, de um mecanismo para extrair uma expressão booleana a partir de uma tabela verdade
- Três formas de se converter uma tabela verdade em expressão booleana:
 1. Soma dos produtos
 2. Produto das somas
 3. Mapas de Karnaugh
- A diferença é a técnica 3 já gera uma expressão booleana simplificada (se bem aplicado)

Mapas de Karnaugh para duas entradas

- uma entrada fica como coluna e outra na linha
- será uma matriz de 2x2 (dois estados para uma entrada e dois estados para a outra entrada)

		Valores do B	
		0	1
Valores do A	0	Valor da saída quando A foi 0 e B também	Valor da saída quando A foi 0 e B foi 1
	1	Valor da saída quando A foi 1 e B foi 0	Valor da saída quando A foi 1 e B também

A o mapa para duas entradas (A e B por exemplo) terá quatro valores que são justamente as quatro saídas possíveis para todas as combinações de A com B. Basta olhar na tabela verdade e preencher as posições com a saída correspondente

Exemplo

A	B	S
0	0	1
0	1	1
1	0	0
1	1	0

		B	
		0	1
A	0	1	1
	1	0	0

$$S = \bar{A}.\bar{B} + \bar{A}.B \text{ (levantado pela técnica soma dos produtos)}$$

- Após jogar as saídas nas suas devidas posições do mapa, passa-se a etapa de agrupamento
 - tipicamente se agrupa as saídas 1
 - mas pode-se também agrupar as saídas 0
- Um grupo de 1 é composto sempre em potência de 2
 - 2 1's lado a lado
 - 4 1,s lado a lado
 - 8, etc
- No caso deste exemplo tem-se apenas um grupo de saídas 1
- Depois de feito isto (conforme a figura), escreve-se a expressão booleana de cada grupo, somando-as. Só que ao escrever, se uma entrada assumiu todos os estados naquele grupo, esta entrada é OMITIDA.
- No exemplo da figura, o B no único grupo de 1, foi 0 e também 1. logo ele é omitido.
- Aquele grupo destacado tem como expressão apenas \bar{A} (pois está na linha onde o A era zero)
- Como é o único grupo, a expressão obtida para esta tabela verdade é apenas

		B	
		0	1
A	0	1	1
	1	0	0

$$S = \bar{A}$$

- Se for olhar a tabela verdade, constata-se que é isso mesmo! Se o A foi ZERO a saída foi 1, independente do B
- Também deveria-se chegar a este mesmo resultado através de simplificações da expressão obtida pela soma dos produtos:

$$S = \bar{A}.\bar{B} + \bar{A}.B \Rightarrow \bar{A}.(B+B) \Rightarrow \bar{A}$$

Mapas de Karnaugh para três entradas

- Para o mapa com 3 entradas deve-se ter um cuidado especial na montagem da tabela.
- Uma entrada fica como linhas e as outras duas como colunas

A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

		BC			
		00	01	11	10
A	0	1	1	1	1
	1	0	1	1	0

Observe atentamente que no mapa depois de 01 vem 11 e não 10. Isto é muito importante

$$S = \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} + \bar{A}.B.C + A.\bar{B}.\bar{C} + A.\bar{B}.C \text{ (levantada pela soma dos produtos)}$$

- Agora que a tabela verdade está (**corretamente**) no mapa, busca-se os agrupamentos, sendo que:
 - Deve-se buscar sempre o máximo tamanho de grupo
 - Uma saída pode pertencer a mais de um grupo
 - Um grupo pode ser ao longo de uma linha, de uma coluna ou linhas e colunas, mas sempre com quantidade potência de 2 (2, 4 ou 8 integrantes)

	BC			
	00	01	11	10
A 0	1	1	1	1
1	0	1	1	0

- No caso, dois grupos são identificados
 - Um grupo com 4 integrantes em toda a linha A=0
 - Um grupo com 4 integrantes no meio, formando um quadrado
 - Se no segundo grupo fosse pego apenas dois 1's, ignorando os de cima, não estaria-se fazendo o mapa com o máximo de otimização.
 - Sempre deve-se agrupar a **maior quantidade**, não importando quantas vezes uma mesma saída é usada
- Tem-se, portanto, duas expressões (uma para cada grupo) que são somadas (OR)
- lembrando que, se em um grupo uma entrada variou de 0 a 1, ela é OMITIDA.

	BC			
	00	01	11	10
A 0	1	1	1	1
1	0	1	1	0

\bar{A} Em toda esta linha do grupo, o A foi ZERO. Tanto o B como o C alternam em 0 e 1. Logo, ambos são omitidos

	BC			
	00	01	11	10
A 0	1	1	1	1
1	0	1	1	0

C Neste grupo o A foi 0 e também 1. Logo ele é OMITIDO. O B também foi 0 e 1 sendo omitido da mesma forma. Neste grupo apenas o C se manteve em 1

$$S = \bar{A} + C$$

No quadro, em aula, foi feito um contra exemplo onde se chegou na expressão $\bar{A} + A.C$ que, após simplificar por 4E resultou na expressão mínima $\bar{A} + C$. Para este contra exemplo agrupou-se os dois 1's entre si formando um grupo com dois integrantes. A ser registrado nas notas de aula (voluntários?).

Mapas de Karnaugh para quatro entradas

- mesmo princípio do de 3 entradas, mas agora duas nas linhas e duas nas colunas
- Lembrando que após o 01 vem 11 e não 10

Grupos das bordas

- É possível montar grupos (2, 4 ou 8) pegando as bordas

	CD			
	00	01	11	10
AB	00	1	1	1
	01	0	0	0
	11	0	0	0
	10	0	1	1

$\overline{B}D$ pois no grupo o A variou e o C também

	CD			
	00	01	11	10
AB	00	1	0	0
	01	0	0	0
	11	0	0	0
	10	1	0	1

$\overline{B}D$ pois no grupo o A variou e o C também

Mapas de Karnaugh

- Obter expressão simplificada a partir de uma tabela verdade
- PRECISA DA TABELA VERDADE
- Se tem apenas a expressão booleana, teria que montar a tabela verdade primeiro
 - desvantagem, pois o trabalho para se levantar a tabela verdade pode ser maior do que simplificar a expressão que já se tem
- Mas se tem apenas a tabela verdade
 - levantar expressões por soma dos produtos ou produto das somas pode ser trabalhoso e gerar uma expressão muito grande e de simplificação complexa
 - Então o mapa é muito mais eficiente
- Opcionalmente o mapa pode ser montado agrupando-se os ZEROS
 - COM EXTREMO cuidado, o mesmo cuidado de quando se faz o produto das somas.

	BC			
	00	01	11	10
A	0	1	1	1
	1	0	1	0

	BC			
	00	01	11	10
A	0	1	1	1
	1	0	1	0

$$S = \overline{A} + C$$

	BC			
	00	01	11	10
A	0	1	1	1
	1	0	1	0

$$S = \overline{A} + C$$

Opcionalmente pode-se pegar estes dois ZEROS como sendo um grupo. Mas ai só se pega ZEROS. Deve-se pensar em produto das somas, o contrário do anterior, ou seja, baseado em portas OR cada grupo. Então, neste grupo o B sai fora porque variou. O grupo está na linha onde o A é 1 e nas colunas onde o C é zero. Para que a saída seja 0, o A precisa ser NEGADO e o C não (pois já é zero). Então este grupo, só ele, representa o $\overline{A} + C$. Casualmente o mesmo resultado.

- No exemplo anterior agrupando-se os uns ou agrupando-se os zeros resultou na mesma expressão booleana.

- Isto nem sempre é verdade. Podem resultar em expressões diferentes, mesmo que mínimas
 - Não existe uma única mínima, podem existir mais de uma

A	B	C	S4
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

		C	
		0	1
AB	00	0	1
	01	1	0
	11	0	0
	10	1	1

$S4 = A.\overline{B} + \overline{B}.C + \overline{A}.B.\overline{C}$

		C	
		0	1
AB	00	0	1
	01	1	0
	11	0	0
	10	1	1

Diagram illustrating the Karnaugh map with groupings and corresponding expressions:

- Group 1 (AB=00, C=0 and 1): $A+B+C$
- Group 2 (AB=01, C=0 and 1): $\overline{B}+\overline{C}$
- Group 3 (AB=11, C=0 and 1): $\overline{A}+\overline{B}$

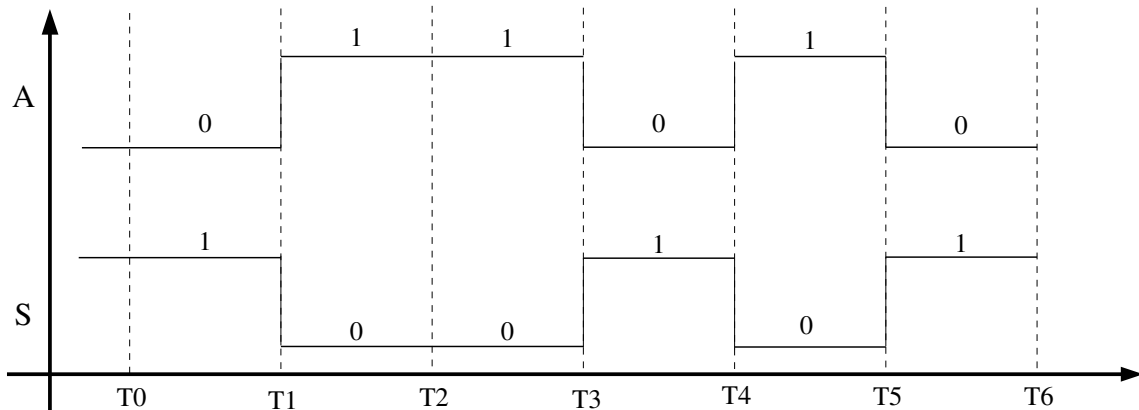
$S4 = (A+B+C).(\overline{B}+\overline{C}).(\overline{A}+\overline{B})$
 E se aplicar a regra 3A no final:
 $S4=(A+B+C).(\overline{B}+\overline{C}.\overline{A})$ (mesma resposta do T12)

Aula 6 - 25/Março/2017

ATIVIDADE SEMIPRESENCIAL - T13

Diagramas de tempo

- Usados para registrar as entradas e saídas ao longo do tempo
- Como é para lógica, uma entrada ou saída pode ser 0 ou 1
- Então, o diagrama do tempo considera apenas 1 ou 0
 - uma forma de onda quadrada
- Uma linha da tabela verdade expressa uma possível combinação de entradas e a devida saída.
- Em um diagrama de tempo, estas entradas podem estar sempre sendo alteradas a cada amostragem
- Exemplo 1 para uma saída $S1 = \overline{A}$



- Exemplo 2: duas saídas: S1 e S2. $S1 = A+B$ $S2 = A.B$

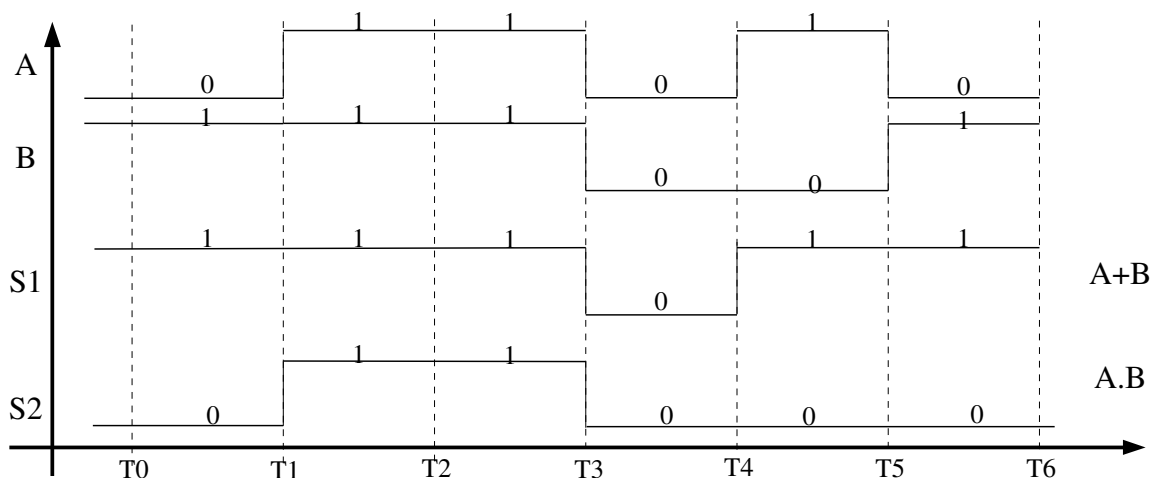


Diagrama de tempo com atraso

- Quando um estado muda, a saída não muda imediatamente
- Existe um atraso, normalmente desconsiderado
- Mas o diagrama de tempo pode considerar
 - Não será usado nesta disciplina
- Exemplo com atraso (no quadro)

Decodificadores

- Liga (ativa) 1 de N saídas
- Cada saída tem um identificador binário
- Entradas suficientes devem ser fornecidas para poder habilitar N saídas
 - Para 4 saídas, precisa-se de duas entradas:

- 00 -> saída 0 01 -> saída 1 10 -> saída 2 11 -> saída 3
- A quantidade de saídas que um decodificador permite é de 2^N sendo N o número de entradas
- A figura 5 apresenta um decodificador para 8 saídas com 3 entradas.
 - Trata-se de um decodificador 3 para 8
- Se a entrada for 000, a saída D0 e apenas ela deve estar ligada
 - e assim sucessivamente
- Isto permite ser representado por uma tabela verdade,
 - sendo um circuito combinacional

Entradas			Saídas							
A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- Cada saída será representada por um único AND de três entradas com algumas delas negadas
 - $D0 \Rightarrow \overline{A2}.\overline{A1}.\overline{A0}$
 - $D1 \Rightarrow \overline{A2}.\overline{A1}.A0$
 - ...
 - $D7 \Rightarrow A2.A1.A0$
- Ao lado, uma implementação do circuito

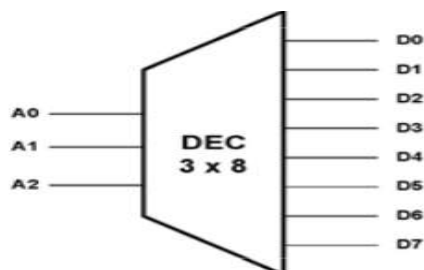


Figura 3: símbolo de um decodificador 3x8

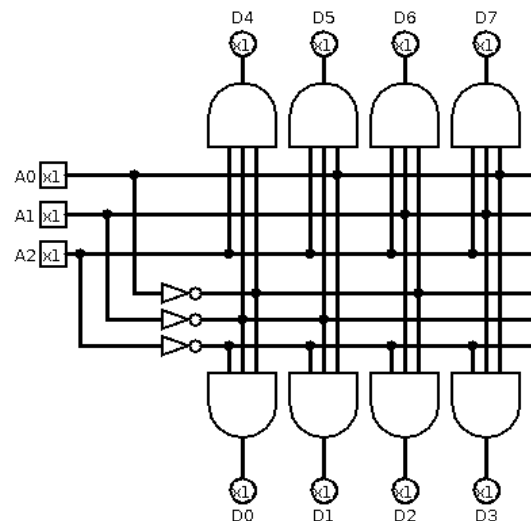
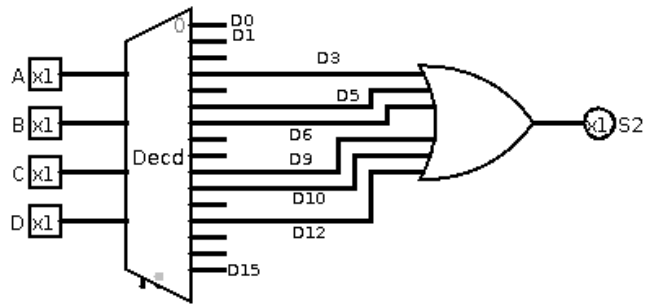


Figura 4: implemesntação de um DEC 3x8

Aplicação de decodificadores

- A primeira é para realmente ativar uma dentre N saídas, cada uma delas com “um endereço”, digamos assim.
- A segunda é bastante interessante: qualquer circuito combinacional pode ser feito com decodificadores!
 - Um decodificador varre todas as possíveis saídas
 - Para implementar uma tabela verdade, basta colocar as saídas que se quer em um OR
 - Exemplo para a questão C, S2 do T14: acende se somente dois interruptores estiverem ligados

A	B	C	D	S2	Saídas de um decodificador 4x16
0	0	0	0	0	D0 - Ignorar
0	0	0	1	0	D1 - Ignorar
0	0	1	0	0	D2 - Ignorar
0	0	1	1	1	D3 - CONSIDERAR
0	1	0	0	0	D4 - Ignorar
0	1	0	1	1	D5 - CONSIDERAR
0	1	1	0	1	D6 - CONSIDERAR
0	1	1	1	0	D7 - Ignorar
1	0	0	0	0	D8 - Ignorar
1	0	0	1	1	D9 - CONSIDERAR
1	0	1	0	1	D10 - CONSIDERAR
1	0	1	1	0	D11 - Ignorar
1	1	0	0	1	D12 - CONSIDERAR
1	1	0	1	0	D13 - Ignorar
1	1	1	0	0	D14 - Ignorar
1	1	1	1	0	D15 - Ignorar



- Qualquer circuito combinacional pode ser montado usando decodificadores
- E, por vezes, de uma forma muito mais fácil
 - Embora se use mais portas lógicas, mas se tu já tem os decodificadores construídos...
- CUIDADO: neste caso as entradas devem ser colocadas exatamente na ordem
 - Para este da votação, C deve ser ligado ao A0 do decodificador

Decodificador com Habilitação

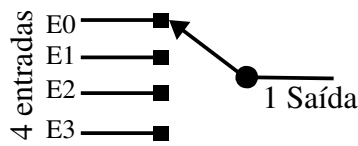
- Possui uma entrada a mais. Se ela estiver em ZERO, nenhuma saída é habilitada
- Se ela estiver em 1, a saída correspondente é habilitada

Codificadores

- Faz o contrário de um decodificador
- Para uma saída, entre N, gera as entradas correspondentes

Multiplexador

- Uma espécie de chave digital
- N entradas podem ser ligadas a uma saída, mas apenas uma das entradas pode ser ligada a cada saída



Ao lado, 4 entradas e uma única saída. Um “interruptor” permite selecionar qual das entradas estará ligada a saída. No caso da figura, a entrada E0 está atualmente conectada. O sinal que E0 gerar será repassado à saída. Um multiplexador seleciona qual das N entradas estará conectada a saída. Cada entrada possuirá um número. Neste caso é de 0 a 3, será endereçado por dois bits

- Assim como no decodificador, o multiplexador possui endereços
 - Para quatro entradas serão necessários dois bits de endereços.
 - se os endereços forem 00, então é a entrada E0 que está ligado na saída.
 - Se E0 for zero, saída será ZERO
 - Se E0 for um, saída será UM
- Decodificador: o endereço LIGA uma das saídas que será 1
- Multiplexador: o endereço CONECTA uma das entradas que será enviado para a saída.
- Basicamente ele copia uma das entradas para a saída
 - Qual entrada?
 - Depende dos endereçamentos
- Uma implementação de um multiplexador de 4 entradas pode ser vista na Figura

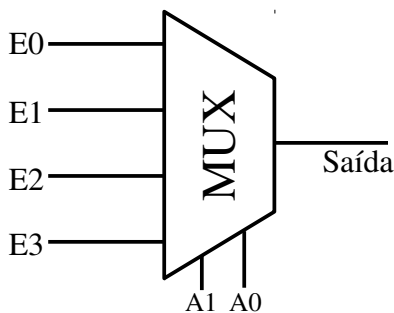


Figura 6: Símbolo de um Multiplexador com 4 entradas

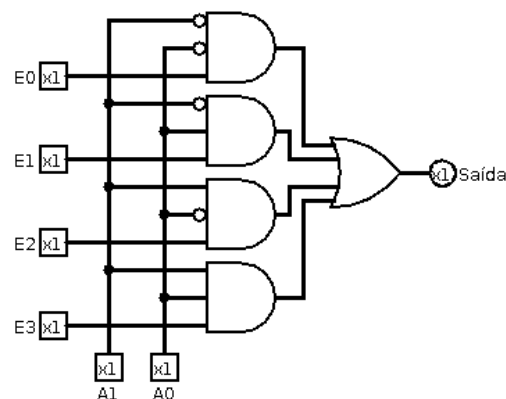


Figura 5: Implementação MUX com portas lógicas

Demultiplexador

- O contrário de um multiplexador
- Liga uma entrada a N saídas
- pensando em interruptores analógicos, como os que acendem a luz de uma casa, não precisaria de um demultiplexador, pois ao ligar uma entrada a uma saída ambas estão ligadas!
 - Porém, no digital e usando portas lógicas, quando a E0 é selecionada, se a E0 coloca 0 a saída é 0, se coloca 1 a saída é 1: a saída copia os bits da entrada
 - Ai vê-se a necessidade de um circuito que faz o contrário

Circuitos Combinacionais aritméticos

- Realizam operações aritméticas : soma, subtração, multiplicação e divisão
- São a base de uma ULA (Unidade Lógica e Aritmética)

Meio somador

- Um meio somador soma apenas dois bits, nada mais
- Seria um somador de dois inteiros, sendo cada inteiro de um único bit
- A resposta é apenas UM bit podendo haver uma sobra
 - Da mesma forma como fazemos ao somar valores decimais
- Esta sobra é chamada de Carry-Out (Vai um)
- A tabela verdade de um meio somador:
 - possui duas entradas, os dois bits a serem somados
 - possui duas saídas:
 - resposta da soma
 - Carry-Out

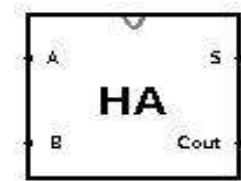


Figura 7: HA: Half-Add (Meio somador)

A	B	S	Carry-Out	Comentário
0	0	0	0	$0+0 = 0$ e não vai um (CO==0)
0	1	1	0	$0+1 = 1$ e não vai um
1	0	1	0	$1+0 = 1$ e não vai um
1	1	0	1	$1+1 = 0$ e vai um (porque em decimal $1+1 = 2$, ou seja, 10 o 0 da direita é a resposta e vai um

Tabela Verdade de um meio somador

- Acredito que com o nível de conhecimento até o momento, torna-se desnecessário fazer mapa de Karnaugh ou simplificar expressões booleanas:
 - a saída S é um XOR de A com B
 - o Carry-Out é um simples AND

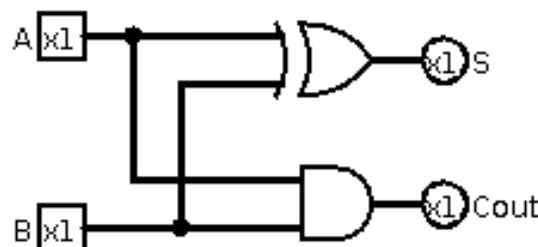


Figura 8: Implementação do meio somador

- Para onde iria este Carry-Out?
 - Para a próxima etapa de soma, podendo ser para outro meio somador
 - mas um meio somador só tem duas entradas, não tem o “vem-um”
 - Mas aí precisamos de um somador completo, que soma três bits
 - melhor deixar isso para a G2 ;-)

Aula 9 - 20/Abril/2017

Correção dos exercícios

Exercício de Revisão

Aula 10 - 04/Maio/2017

PROVA

Circuitos Combinacionais aritméticos

- Realizam operações aritméticas : soma, subtração, multiplicação e divisão
- São a base de uma ULA (Unidade Lógica e Aritmética)

Meio somador

- Visto anteriormente
- Soma um único bit
 - resposta de 1 bit
 - e um possível carry out (sobra para somar com o próximo)

Somador completo

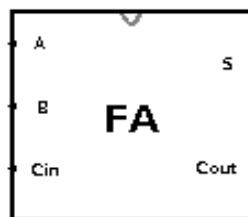
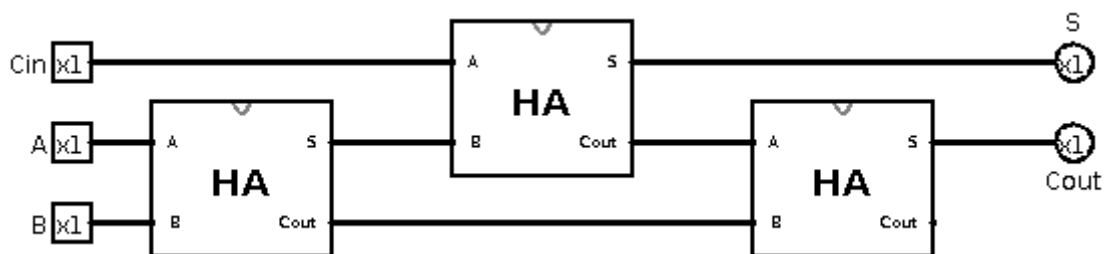


Figura 9: FA: Full Add

- Soma completamente dois bits, recebendo um vem-um de um somador completo anterior
- Pode-se construir um somador completo usando três meio somadores, como mostrado na Figura



- Isto não é eficiente, pois consome muitas portas
- A saída Cout do último meio somador será sempre 0 e é desperdiçado a lógica para calcular ela
- Sem falar que fica estranho usar um S de um meio somador para exportar o Carry da soma
- MAS FUNCIONA

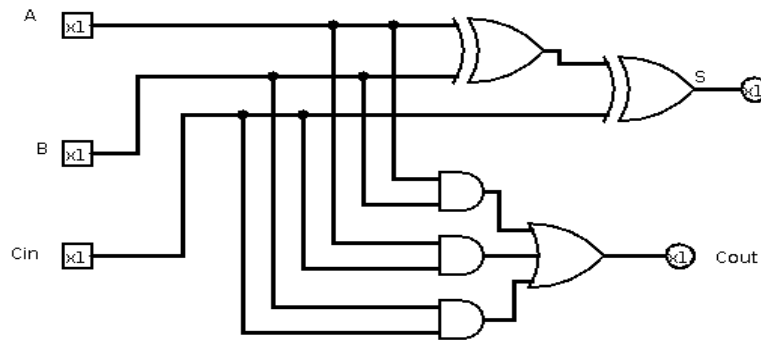
Somador completo com portas lógicas

- O melhor é realizar uma tabela verdade para um somador completo e refazer o circuito
- Ela terá TRÊS entradas (A, B e Cin) e duas saídas (S e Cout)

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Para a saída S a expressão booleana que a representa é bem simples se usar XOR: $(A \oplus B) \oplus Cin$ (visualmente se confere que funciona. Como não vimos simplificações booleanas envolvendo XOR, fica difícil demonstrar matematicamente).

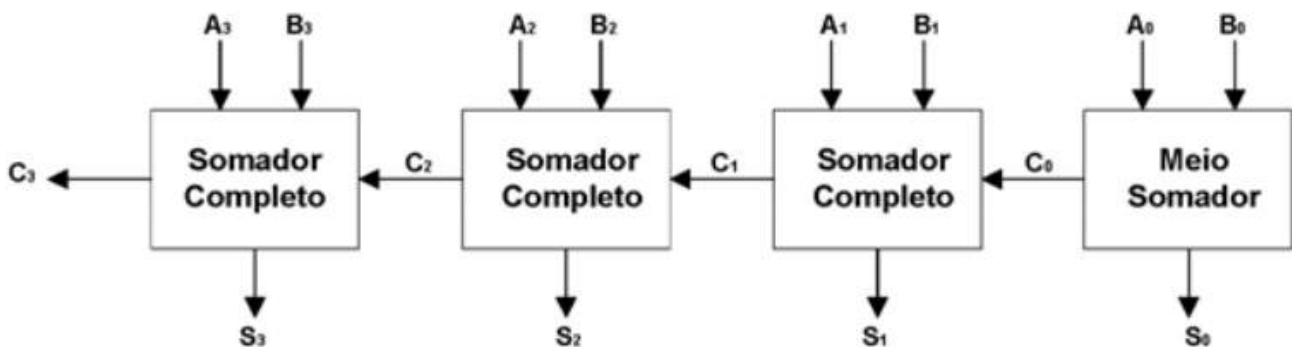
Para a saída Cout uma mapa de Karnaugh devolve $A.B + A.Cin + B.Cin$, ou seja, um OR de três entradas recebendo a saída de três ANDs de duas entradas.



Somador para N bits

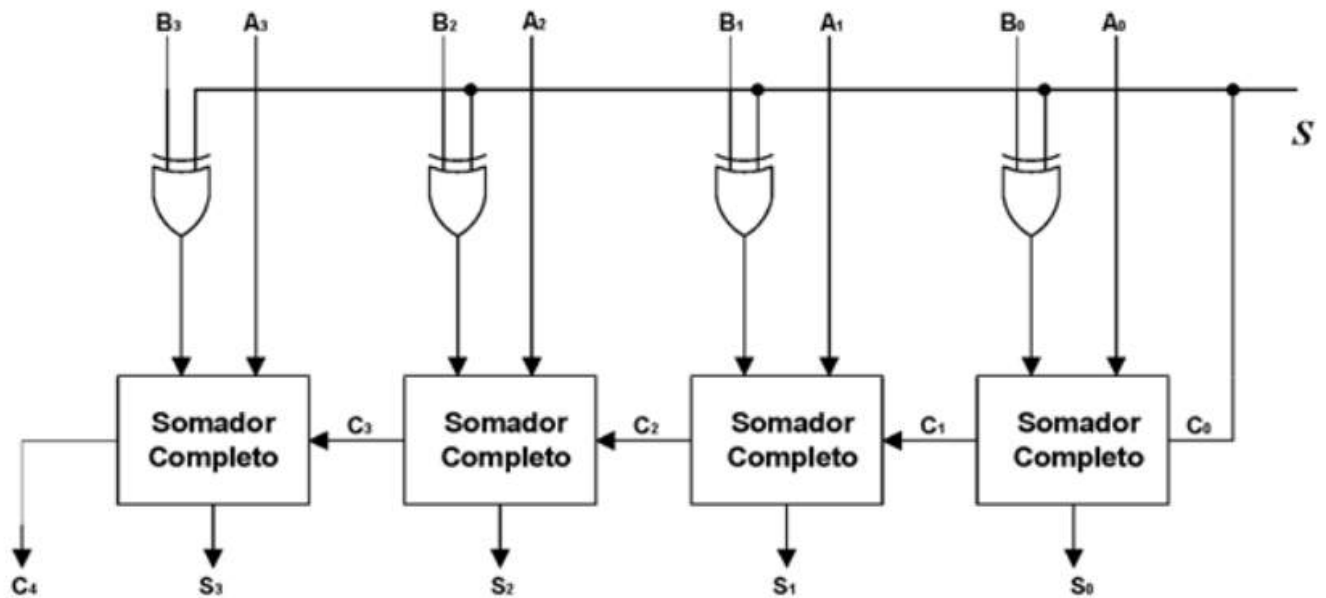
- Para realizar a soma de N bits em um circuito combinacional, basta concatenar N somadores completos.
 - Se é a primeira soma, então não existe um vem um (viria de onde?).
 - Desta forma, a primeira soma pode ser feita por um meio somador.

Exemplo de um somador para 4 bits: neste caso se a resposta deve caber em QUATRO bits, o último carry out deve ser considerado como OVERFLOW



Subtração

- Para subtrair em complemento de dois, basta fazer uma adaptação ao somador completo, fazendo o A ser somando com o complemento de dois de B (o mesmo que $A - B$)
 - complemento de dois é negar o B e somar 1 a ele



$S = 0 \Rightarrow A + B$

$S = 1 \Rightarrow A - B$

- Observe que o S entra como Carry-in no primeiro bit. Logo, se ele for 1, está-se somando 1
- Observe que a porta XOR é uma espécie de NOT com seletor:
 - Se S for 0, o NOT é desligado e o B passa com seu valor original
 - Se S for 1, o NOT é ligado e todos os bits de B são invertidos (complemento)

Aula 12 - 11/Maio/2017

- Demonstração do Logisim
 - pinos com mais de um bit
 - fios com múltiplos bits
 - distribuidor
 - tuneis
 - Encapsulamento de circuitos
 - buffer
- Montagem do somador completo de 4 bits como demonstração
- Exercício T21 (início em aula)

Aula 13 - 13/Maio/2017

ATIVIDADE SEMIPRESENCIAL - T21

Circuitos Lógicos

- Combinacionais
 - para um conjunto de entradas uma saída
 - Nada, além das entradas, influencia a saída
 - Exemplo: nosso sistema de votação do Exercício T13
- sequenciais
 - Para um conjunto de entradas haverá uma saída
 - porém, saídas anteriores são usadas como entradas
 - Ele é realimentado, digamos assim, com valores passados de saídas

Circuitos Combinacionais

- São montados tendo como base tabelas verdade
- Combinações de entradas geram saídas.
- Etapas:
 - Ter uma especificação de problema
 - Determinar símbolos para cada entrada e saída
 - Montar a(s) tabela(s) verdade(s)
 - extrair expressões booleanas simplificadas
 - por mapa de *karnaugh*
 - por soma dos produtos ou produto das somas, devendo simplificar
 - Mapear para portas lógicas, respeitando restrições
 - exemplo: o circuito precisa ser montado usando somente portas NAND
 - Desenhar o circuito
- Todas estas etapas foram exercitadas nos exercícios

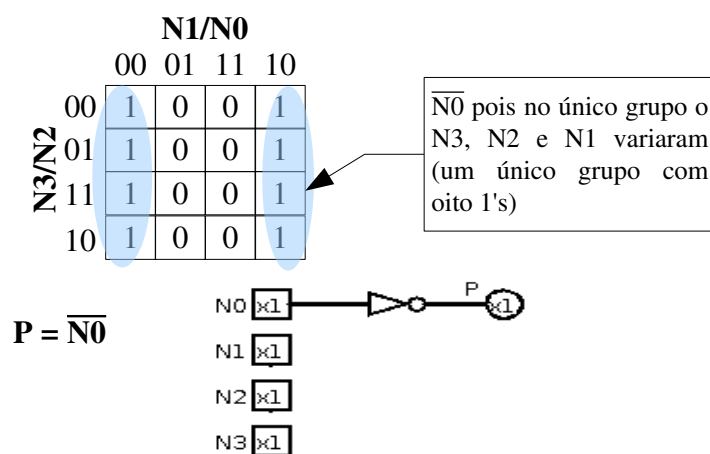
Exemplo: detector de números pares de 4 bits

Especificação: Para um determinado número inteiro de 4 bits (de 0000 a 1111 ou de 0 a 15 em decimal), fornecer uma saída par que terá 1 se o número inteiro for par.

Entradas e saídas: N3, N2, N1 e N0 para os quatro bits do número. Saída P para PAR

Tabela verdade:

	N3	N2	N1	N0	P
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0



Circuitos Sequenciais

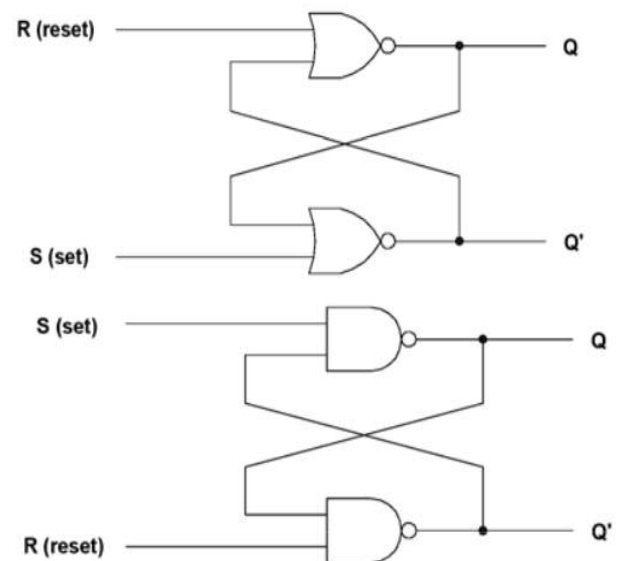
- As saídas são influenciadas pelo “passado”
- Síncronos ou Assíncronos
- Assíncronos: podem mudar a qualquer momento
- síncronos: mudam o estado apenas quando ocorre um sinal de relógio (clock)

Latches

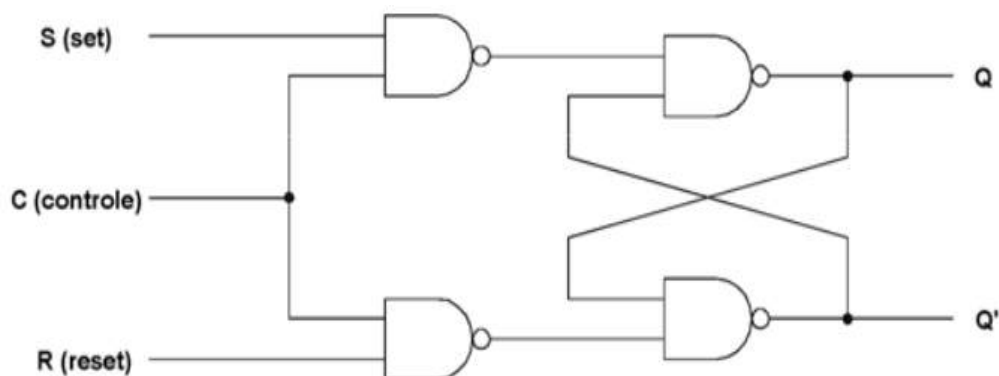
- Duas saídas: Q e \bar{Q}
- duas entradas: R e S
 - R : se colocar em 1 faz Q ser 0. E Q fica em 0 mesmo removendo o 1 depois
 - S : faz Q ser 1

Latches SR

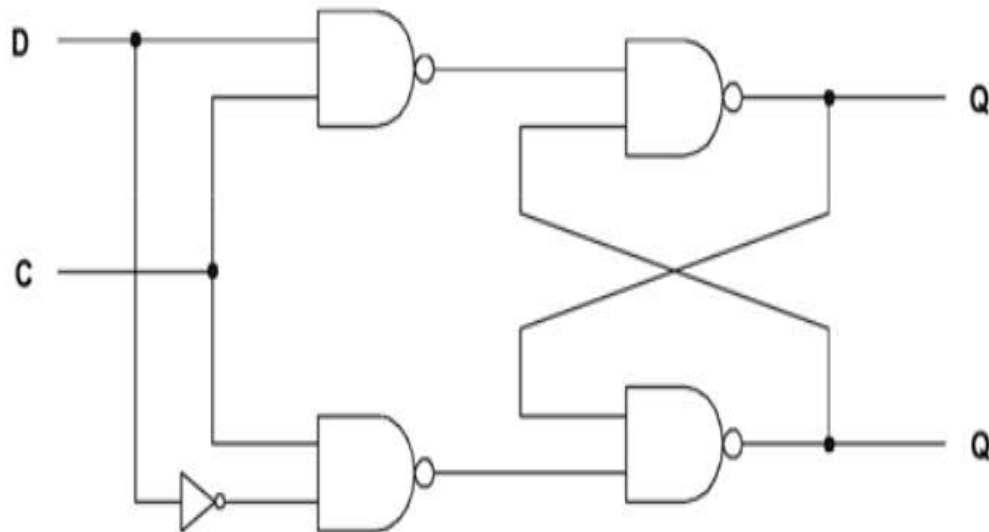
- Com portas NOR ou portas NAND
- Se feito com NOR:
 - R em 1: Reseta (Q em 0)
 - S em 1: Seta (Q em 1)
 - R e S em 0: MANTÉM
 - R e S em 1: estado inconsistente. Não pode
- Se feito com NAND:
 - R em 0: Reseta (Q em 0)
 - S em 0: Seta (Q em 1)
 - R e S em 1: MANTÉM
 - R e S em 0: estado inconsistente. Não pode.



- Pode ser com Controle (Tipo um Habilita)
 - A O Set ou Reset só será considerado se Controle estiver em 1



- Se S ou R estiverem em 1 (caso da versão em NOR), a saída é indefinida
- Latch D resolve isto
 - Ele não tem R e S, mas apenas D (Data) e o C (Controle).
 - Precisa ter o Controle
 - Quando e enquanto C for 1 o que estiver em D é enviado a saída Q



Aula 15 - 25/Maio/2017

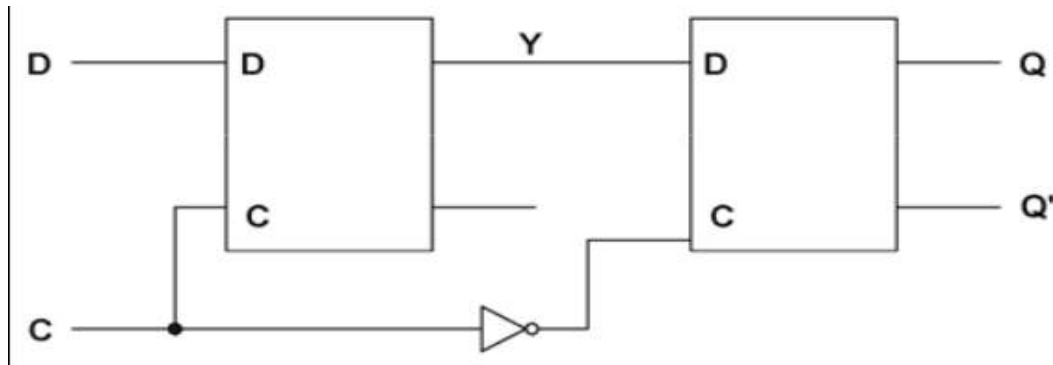
Flip Flops

- Uma memória de 1 bit
- Consegue armazenar dentro do circuito 1 bit enviado
- Baseado em latches
- Funciona através de clock
- Controle é diferente de clock
 - Controle (Habilita) se estiver em 1, a entrada é considerada. Se a entrada mudar enquanto Controle estiver em 1, ela é imediatamente considerada.
 - Clock: considera a entrada apenas quando o clock mudar seu estado
 - passar de 0 para 1 (chamado de Ascendente) ou
 - passar de 1 para 0 (chamado de Descendente).
 - Clock estando em ZERO, se a entrada mudar, não considera
 - Clock estando em UM, se a entrada mudar, também não considera.

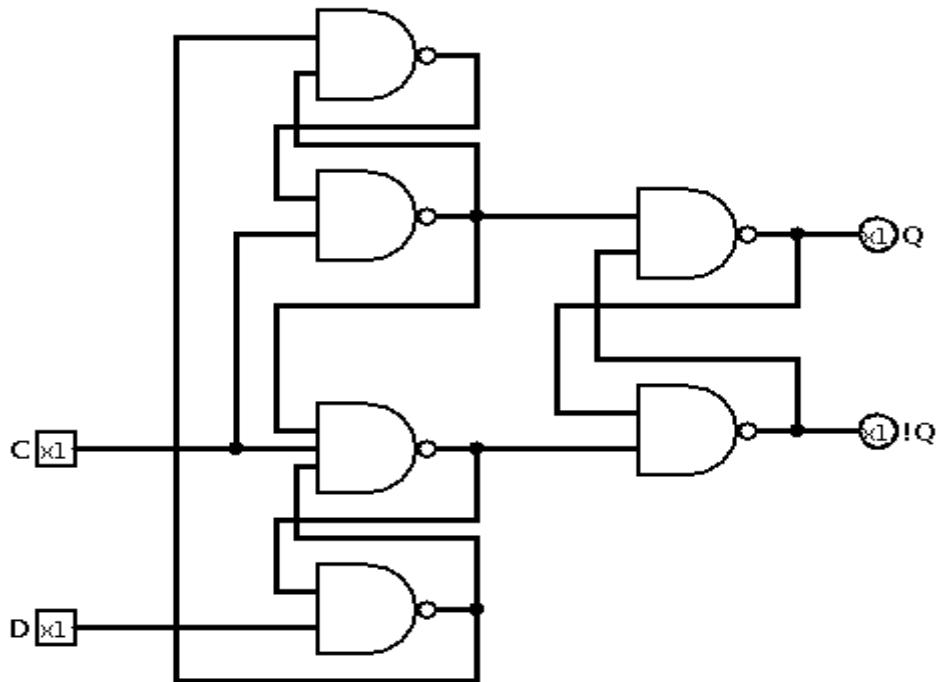
Flip Flop D

- possui duas entradas, D e C (como o latch D)
- Porém só muda a saída quando o C subir (ascendente) ou descer (descendente)
 - Ascendente: este tipo de flip flop só altera a saída Q quando o clock mudar de 0 para 1
 - Descendente: neste tipo a saída só altera-se quando o clock mudar de 1 para 0
- Ambos não alteram a saída se Clock permanecer em 0 ou permanecer em 1

Flip Flop D descendente pode ser implementado com dois latches do tipo D:

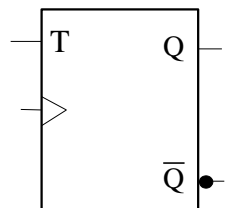


Já um Flip Flop D Ascendente:



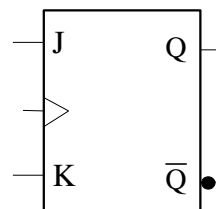
Flip-Flop tipo T

- Quando houver uma variação no clock (ascendente ou descendente)
 - Se $T=0 \Rightarrow$ o que tem na saída é mantido
 - Se $T=1 \Rightarrow$ o que tem na saída é invertido (se era 1, fica 0 e vice-versa)
- Importante que este flip-flop tenha uma entrada de Clear, para inicializá-lo.

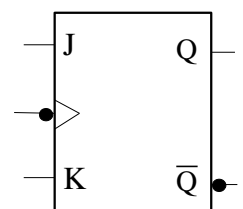


Flip-Flop tipo JK

- Possui Clock e duas entradas: J e K
- Quando houver clock (seja ascendente ou descendente):
 - Se $J = 1$ e $K = 0$, então $Q = 1$ (função SET)
 - Se $J = 0$ e $K = 1$, então $Q = 0$ (função RESET)
 - Se $J = 0$ e $K = 0$ não haverá alteração em Q
 - Se ambos forem 1, Q será INVERTIDO.
- Com o JK é possível construir o T e o D



JK Ascendente



JK Descendente

Pinos auxiliares

- Flip-Flops podem ter outros pinos auxiliares, além dos inerente ao seu tipo
 - **Enable:** se tiver esta entrada, ela estando em ZERO o clock é ignorado. Só funciona se Enable estiver em 1 (alguns podem ser ao contrário, marcado como $\overline{\text{Enable}}$)
 - **Clear:** permite “limpar” a saída, ou seja, forçar que Q seja 0. Serve para inicializar o Flip-Flop com zero. Colocar em 1 para limpar. Enquanto Clear estiver em 1, a saída Q será 0 forçosamente mesmo que clock esteja em funcionamento. Alguns tem esta entrada invertida marcada como $\overline{\text{Clear}}$. Neste caso deve ser colocado 0 para limpar
 - **Preset:** Simplesmente o contrário do Clear. Serve para forçar a saída Q a ser 1. Também pode ser ao contrário ($\overline{\text{Preset}}$).

Resumo dos tipos de Latches e Flip-Flops

- **Latch SR:** não tem clock. Entradas são imediatamente enviadas para a saída. S=0 e R=0 não muda. S=1 e R=1 saída indefinida. Dependendo de como for montado, considerar S e R invertido (ou seja, S=1 e R=1 não muda, S=0 e R=1, SET, etc)
- **Latch D:** possui apenas uma entrada a D e um Controle (Habilita, quase um clock).
 - Controle == 0, nada acontece. Clock = 1, variação do D imediatamente repassada a saída.

A diferença entre Latch e Flip-Flop é que os Flip-Flops só consideram as entradas quando há clock, que pode ser ascendente (subiu de 0 para 1) ou descendente (desceu de 1 para 0). Nos Latches ou não há clock (SR) ou o clock é, praticamente, um “habilita” considerando as entradas enquanto estiver em 1 (caso do latch D).

- **Flip-Flop D:** quase um Latch D, exceto que a mudança é quando o clock SOBE ou quando desce. O valor de D é copiado para a saída Q quando houver alteração no clock (subida ou descida como qualquer flip-flop)
- **Flip-Flop T:** Uma entrada T e um clock. Se T=0, saída não muda. Se T=1, saída inverte o que tinha. Imprescindível ter entrada Clear para inicializar a saída.
- **Flip-Flop JK:** Pode-se dizer que é um Flip-Flop SR misturado com um tipo T. J e K tem a função de Set e Reset. Ambos em ZERO, nada muda (como SR). J=1 e K=0 é Set (como SR), J=0 e K=1 é Reset (como SR). J=1 e K=1 troca estado da saída (como no T).

Uso do Clear: todos os flip-flops podem ter Clear que inicializa Q. Porém o Flip-Flop tipo T precisa ter o Clear, pois é a única forma de forçar a saída para um valor. Clear força a saída Q a ser 0. No D e JK isto não é necessário pois pode-se colocar 0 na saída (D=0 ou J=0,K=1).

Aula 16 - 01/Junho/2017

- Continuação do T22 em aula
- Início do T23

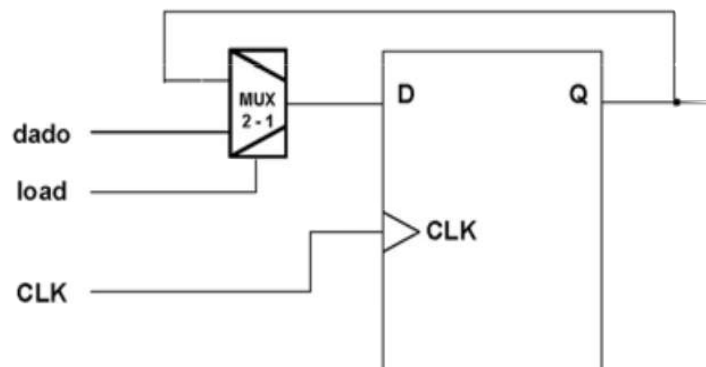
Aula 17 - 03/Junho/2017

ATIVIDADE SEMIPRESENCIAL - T23

Aula 18 - 08/Junho/2017

Registradores

- Um registrador é uma memória interna dos processadores
- Ele armazena uma palavra
 - no caso das arquiteturas de 64 bits, atuais, um registrador armazena 64 bits de dados
- Eles possuem nomes e não endereços: Ex: rax, rbx (64 bits)
- No processador o clock está sempre oscilando
 - Exemplo: 2Ghz faz todo o circuito oscilar 2.000.000.000 vezes por segundo
- Não é interessante a saída ser influenciada pela entrada a cada subida ou descida de clock
 - Isto torna o flip flop D não tão interessante para construir um registrador
 - Ao armazenar um bit, a mudança do clock poderia removê-lo da memória
- Um registrador de um bit pode ser construído a partir de uma leve alteração em um flip-flop tipo D.



- A inclusão de um multiplexador 2x1 (duas entradas e uma saída) resolve o problema
 - D receberá Q se load for 0
 - D receberá dado se load for 1
- Assim, com o load em ZERO, a cada oscilação de clock, o D recebe o próprio Q, regravando o mesmo bit na memória.
- Para gravar outro bit na memória:
 - colocar o bit a ser gravado em dado
 - colocar load em 1
 - esperar o clock
- Após gravar, manter load em 0
- O registrador também pode ser feito apenas com um flip-flop tipo D se ele tiver o pino ENABLE.
 - O LOAD será ligado no Enable simplesmente
 - Se LOAD em 0, não está habilitado e o clock é ignorado
 - Se LOAD estiver em 1, habilita

- Projeto de um registrador de 8 bytes
 - Demonstração
- Chamado de registrador com carga paralela
 - Todos os 8 bits entram ao mesmo tempo para serem armazenados

Registrador de deslocamento

- Servem para realizar entrada de dados sequencial
- Ou para realizar SHIFT
- Ou para converter de paralelo para serial ou de serial para paralelo

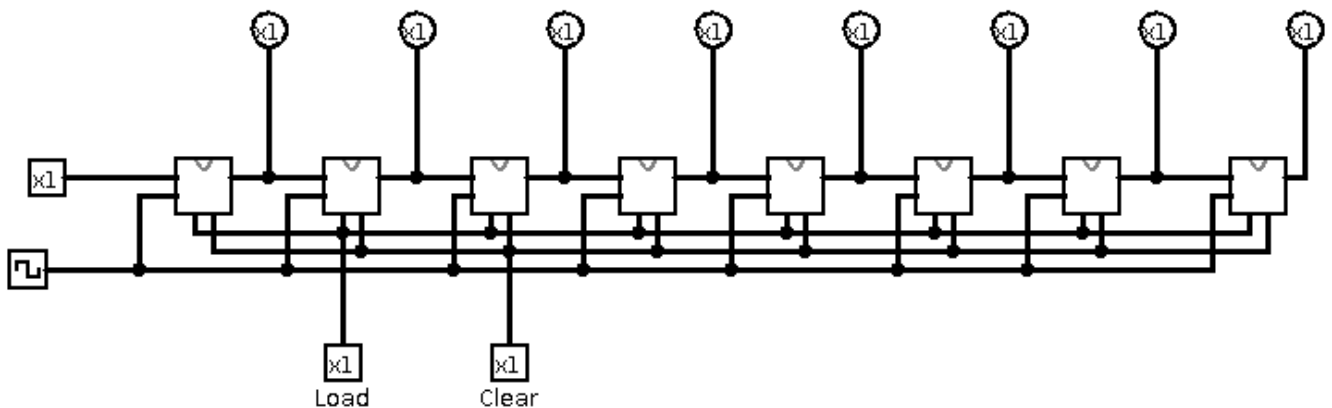


Figura 10: Registrador de deslocamento 8 bits

- Na Figura tem-se um exemplo de um registrador de 8 bits com carga serial (a caixa é um registrador de 1 bit. A esquerda tem a entrada D e o clock, a direita apenas o Q. Embaixo tem os pinos Load e Clear).
- A cada clock, um bit de entrada é lido
- os demais bits são movidos para a direita
- Ao final de 8 clocks, oito bits são carregados
- Também é possível ter um registrador com carga paralela ou serial
 - Um mux define qual a função do registrador
 - Na Figura 7 tem uma parte, um bit, de um registrador de seguimento que pode ter carga paralela ou carga serial
 - Se o pino Serial/Par estiver em 0, a entrada 0 será jogada para o próximo Flip-Flop
 - Ou seja, o D do próximo flip flop receberá o D do flip flop anterior
 - Trata-se, portanto, de carga serial
 - Mas se o pino Serial/Par estiver em 1, então a entrada 1 será selecionada
 - Desta forma o que está no pino Dn será jogado para o D
 - trata-se de carga paralela
- Usando um registrador de deslocamento com carga paralela para realizar multiplicação por dois
- Usando registrador de deslocamento para realizar multiplicação de decimais
 - Demonstração
 - 4x3 em quatro bits (no quadro)

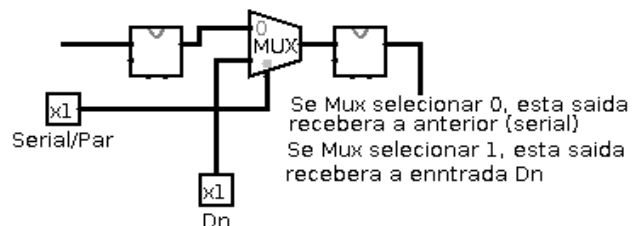


Figura 11: Parte registrador deslocamento Par/Ser

Registradores Contadores

- Servem para contar
- Contar mesmo, 0, 1, 2, 3.. etc
- Existem diversos tipos de contadores
 - Este feito a partir de flip flops tipo T conta de forma decrescente

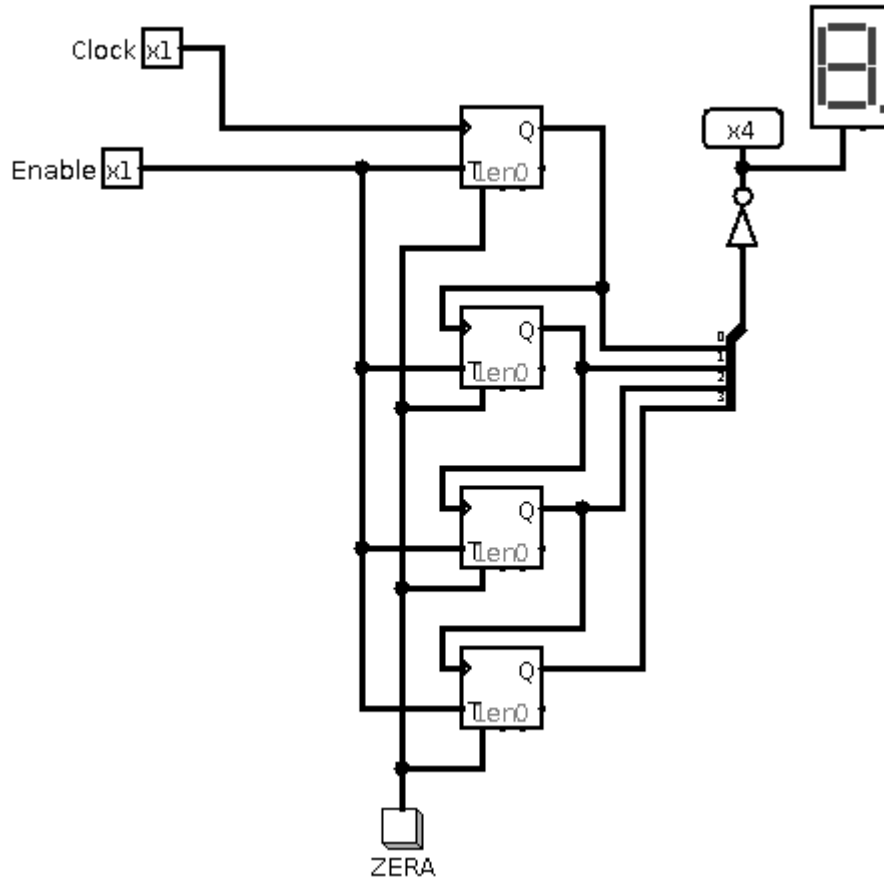


Figura 12: Contador Decrescente (com inversor)

Aula 19 - 22/Junho/2017

Exercício Revisão Moodle

Aula 20 - 29/Junho/2017

PROVA G2

Aula 21 - 06/Julho/2017

Correção G2

Aula 22 - 13/Julho/2017

Prova G3