Circuitos Digitais

Thiago Figueiredo Marcos

4 de março de 2025

1 Introdução

Um computador digital pode ser descrito como aquilo que computa, ou aquilo que processa informação digital. A informação que é processada por um circuito digital é aquela que é **quantizada** ou **discretada**.

No mundo comum, as informações são analógicas, ou seja, a onda que representa aquela informação possui uma gama de valores diferenciados. Já os sinais digitais podem ser representados por valores que são descontinuos, no tempo, e interpretado por uma lógica binária.

Os circuitos digitais são construidos por componentes eletrônicos e tem como entradas e saidas sinais digitais. Para usarmos informações do mundo analógico é preciso discretar essas informações, afím de convertelas à binária e geralmente é usado uma medida em Volts para determinar se uma informação é ligada ou desligada, e isso se chama a amplitude do sinal. Após convertida a informação e processada no circuito digital é preciso converter o sinal de saída do circuito, que é digital, em analógico novamente.

2 Conversão Analógico - Digital (Discretação)

Sinais discretos são frequências descontinuas no tempo, ou seja, definida apenas para determinados instantes. Representa aproximadamente o mundo real, entretanto, podem ser utilizadas várias técnicas para melhorar a representação, como as de processamento de sinais digitais.

Aqui também vale ressaltar que o processo de discretação de alguma informação está consequentemente ligada a perca de determinadas informações.

2.1 Principais propiedades da discretação

1. **Amostragem**: Discretação do sinal analógico no tempo.
2. **Quantização**: Discretação da amplitude do sinal amsotrado em niveis.
3. **Codificação**: Atribuição de códigos, onde geralmente são binários às amplitudes do sinal quantizado.

3 Conversão Digital - Analógico (Linearização)

Se refere a o processo que transforma um sinal modelado por eventos discretos em um sinal contínuo, ou seja, o processo de integração de vários sinais discretos para simular um evento contínuo.

4 Processamento

O processamento de informação se refere a diversas operaçõs realizadas por um circuito digital para transformar a entrada de dados em uma saida significativa de interesse.

Isso pode ser: calculos, manipular dados como agregação, separação e classificação ou ainda filragem, entre outros. Além disso, no circuito digital é simplificado o armazenamento de informações bem como possui uma menor probabilidade de interferências.

5 Sistemas de numeração

Número nos remete a ideia de quantidade, já o numeral é a representação desta ideia, na prática, nos referimos a palavra número para qualquer tipo de representação numeral.

Exemplo: A quantidade `Quarenta e dois` é representada pelo numeral 42.

Sem o conhecimento da organização posicional dos números, como podemos representar todos eles? Poderiamos pensar em um simbolo para cada número, porém, existe uma infinidade de quantidades.

Há cerca de 3.000 anos atrás os `Egípcios` desenvolveram um sistema de numeração em base 10, na qual utilizamos até hoje. Os números representados por hieróglifos eram mais usados em monumentos e templos, pintados ou talhados em pedra. Há sete símbolos, representando os números 1, 10, 100, 1000, 10 000, 100 000 e 1 000 000.

Algarismos é um conjunto finito de símbolos numéricos que usamos para representar quantidades reais. Todo e qualquer número pode ser representado por uma combinação de algarismos, os mais conhecidos são os indo-arábicos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Sistema de numeração é a forma de atribuir uma representação única para cada número. O sistema de numeração posicional atribui valor ao algarismo conforme a sua posição, mais a esquerda ou mais a direita.
No sistema decimal de numeração posicional possuimos 10 algarismos, 0 .. 9, cada um deles representa seu valor absoluto, ou seja, o valor 0 representa o nada, o valor 1 representa uma única unidade e assim por diante. Dependendo da posição que o valor estiver, seu valor absoluto pode variar, por exemplo: Imagine um número com 4 casas decimais, - - - -, o número que estiver na 'casa' mais a esquerda não representara seu valor absoluto e sim o seu valor absoluto multiplicado por um milhão.

Exemplo: 4237 = 4*1000 + 2*100 + 3*10 + 7*1, observe o seguinte:

4*1000 = 4.000
2*100 = 200
3*10 = 30
7*1 = 7

A soma de todos os valores resulta no valor original 4.237.

Agora podemos sistematizar isso matematicamente, sabemos que um número inteiro A no sistema decimal é presentado por N digitos assim:

$$A_{n-1}A_{n-2}...A_{n2}A_{n1}A_{n0}$$

Cada a_i é um algarismo decimal.

$$a_{n-1} * 10^{n-1} + a_{n-2} * 10^{n-2} + ... + a_2 * 100 + a_1 * 10 + a_0 * 1$$

ou seja,

$$\sum_{i=0}^{n-1} a_i * 10^i$$

Usando a mesma lógica, podemos representar os números racionais no sistema decimal

$$\sum_{i=0}^{n-1} a_i * 10^i + \sum_{i=1}^{\infty} a_{-i} * 10^{-i}$$

6 Truncamento

Vemos que à medida que caminhamos mais a direita depois da virgula, o valor relativo a cada algarismo se torna cada vez menor, podemos fazer uma representação aproximada do número gerado, limitando o número de algarismos após a vírgula por uma constante `M`.

Essa aproximação chama-se truncamento. Com o truncamento há também um erro de aproximaçaõ que pode ser obtido com a diferença do número original com o número truncado.

Ao truncarmos um número com uma constante `M` para qualquer número real com n algarismos à esquerda da virgula, e `M` algarismos à direita, assim:

$$a_{n-1}a_{n-2}...a_1a_0, a_{-1}a_{-2}...a_M$$

então temos que o erro de aproximação de qualquer número N será:

$$\texttt{err} < 10^{-M}$$

ou seja, aumentar `M` implica em diminuir o erro.

7 Bases não decimais

A quantidade de algarismos usados em um sistema de numeração posicional é chama de base, por exemplo: O sistema decimal tem 10, o sistema binário tem 2 e assim por diante.

Acima fizemos uma representação matemática dos números posicionais racionais, a questão é que, em um sistema posicional em uma determinada base **d**, pode ser representada da seguinte forma:

$$\sum_{i=0}^{n-1} a_i * d^i + \sum_{i=1}^{\infty} a_{-i} * d^{-i}$$

Para indicar a base em que um número esta representado é comum a seguinte notação:

$$(a_{n-1}a_{n-2}...a_1a_0, a_{-1}a_{-2}...a_M)_d$$

8 Conversão de bases

Para converter um número $\mathbf{n}_{10}$ para $\mathbf{n}_d$ faremos divisões sucessivas entre o quociente e a base **d**.

Existem algumas conversões que são triviais de realizar, como por exemplo da base 2 para base 16, ou ainda, da base 10 `(até o 15)` para base 16.

Da base 2 para base 16 por exemplo, pode se seguir o padrão de contagem de 4 em 4 bits e ver qual a representação.

9 Operações Binárias

A base 2 é a base númerica mais utilizada na computação hoje em dia, em conjunto com a base 8, 16 e 64. Veremos futuramente que o computador realiza operações aritméticas na sua Unidade Lógica Aritmética `(ULA)`, entenderemos agora como essas operações são realizadas na base binária.

9.1 soma

Antes de falarmos sobre a soma binária, precisamos relembrar o algoritmo da soma decimal, considere dois numeros A e B, sendo eles:

A = $a_{n-1}a_{n-2}...a_1a_0$
B = $b_{n-1}b_{n-2}...b_1b_0$

O resultado da soma de A com B:

C = $c_n\ c_{n-1}\ c_{n-2}$ ... $c_1\ c_0$ com n + 1 algarismos.

O mesmo método se aplica a somas binárias, exemplificado abaixo:

x	Y	Resultado	Vai um
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

9.2 Overflow

Agora considere a seguinte soma, $(111001)_2 + (110011)_2 = (1101100)_2$ repare que a soma de dois números binários de 6 bits, resultou em um número binário de 7 bits, se nossa memoria fosse apenas de 6 bits, ocasionária um `overflow`.

Há também a possibilidade de fazer a representação em uma quantidade maior de bits, basta adicionar o zero a esquerda do número e ter memória suficiente para a representação.

Em complemento de dois, a verificação do overflow, pode ser verificada analisando o bit do sinal do resultado e comparar com o bit do sinal do número que está sendo adicionado.

Um `Overflow / Underflow` só ocorre quando somado dois números de mesmo sinal, seja eles positivos ou negativos. Quando temos este caso, devemos analisar se o resultado tem o mesmo sinal dos operadores, se for diferente, temos um `Overflow / Underflow`.

9.3 Subtração

Para realizar as subtrações binárias, usaremos o complemento de 1 e complemento de 2, para evitar, detalhamentos adicionais.

`Complemento de 1`: O complemento de 1, basicamente, é o processo de inversão do bit, se for 1 fica 0 e vise versa.

`Complemento de 2`: É o complemento de 1 de uma sequência de bit, adicionando uma unidade ao final, exemplificando:

$\bar{B}$: Complemento de 1

$\bar{B} + 1$: Complemento de 2

9.4 Multiplicação

O algoritmo da multiplicação binária é semelhante a ídeia usada no decimal, e muito mais facil, pois os operandos é somente 0 e 1.

Observe que se A tem `n` algarismos e B tem `m` algarismos o produto $A * B$ terá no máximo, `n` + `m` algarismos.

9.5 Números Reais

Como fica as operações nos números reais? como sempre, podemos nos inspirar na base decimal, suponha os seguintes números:

A = $a_{n-1}\ a_{n-2}\ ...\ a_1\ a_0,\ a_{-1}\ ...\ a_{-k}$
B = $b_{n-2}\ b_{n-2}\ ...\ b_1\ b_0,\ b_{-1}\ ...\ b_{-k}$

se $a_{-k} \neq b_{-k}$, ou seja, se os valores depois da virgua de `a` forem diferentes da virgula de `b`. Na soma, subtração, e multiplicação podemos inicialmente ignorar a virgula, realizar a operação, logo em seguida, realocar a virgula `k` algarismos à direita.

10 Representações númericas em computador digital

Em um computador digital, quaquer informação (dados), em última instância é representado por um número. Atualmente os números são representados em base 2 pela facilidade em fazer contas. Um computador digital possui espaço finito para guardar informações e o processamento dessas informações é feito em grupos de bits em vez de bit a bit, para uma melhor eficiência.

Como o processamento desses dados (informações) é feito em grupos de bits, damos um nome a esse grupo, que é chamado de `palavra de dado`, trata-se de uma sequência de bits de tamanho fixo que é processada em conjunto. Uma palavra por exemplo pode ter 16 bits ou 8 bits, depende...

Um conjunto de 8 bits é chamado de 1 byte, com essa relação explicita de 1 byte sendo 8 bits. Um sistema digital, pode padronizar o tamanho de suas `palavras`(operandos). Os mais comuns hoje são os processadores de 64 bits ou ainda os mais antigos de 32 bits, isso significa que na Unidade Lógica Aritmética `(ULA)` dentro do processador, os operandos podem ter no máximo 64 bits, ou 32 bits depende da arquitetura. veremos isso mais profundamente no futuro.

10.1 Estudos de melhoria de processamento de dados

Hoje existem pesquisadores tentando melhorar a capacidade de processamento, na computação clássica, uma melhor capacidade de processamento seria uma palavra maior ou ainda o processamento paralelo de palavras. Existe a computação quântica, que utiliza a sobreposição de estados nas particulas atômicas para definir 0 ou 1, ou ainda os 2 ao mesmo tempo, gerando uma especie de processamento paralelo bem avançado, porém, probabilistico.

A melhora da capacidade de processamento dos dados na era da informação é uma quebra de barreiras e uma evidente elevação da fronteira do conhecimento.

11 Representação Binária

Podemos sistematizar a representação de números inteiros com `N` bits da seguinte forma: $2^n - 1$.

Exemplo: Com 3 bits podemos representar $2^3 - 1$ números inteiros.

$$2^3 - 1 = 7$$

Com esta sistematização podemos prever o próximo número e os anteriores, da seguinte forma:

O número atual é $2^n - 1$ o próximo seria: 2^n, logo, o anterior do atual é $2^n - 2$.

12 Extenções

Imagine a seguinte situação, um número `A` com `n` bits, porém, queremos fazer uma representação com `w` bits sendo `w > n`.

Se o número for um inteiro sem sinal, neste caso, basta adicionar os zeros a esquerda até `n` ficar igual a `w`.

12.1 Números com sinais

Para fazermos representações de números negativos, precisamos reservar um espaço na palavra para representar o sinal. Existem várias técnicas para fazer esta representação, como o sinal-magnitude, complementos de 1 e de 2, veremos adiante.

12.1.1 Sinal-Magnitude

O Sinal-Magnitude é a reserva do bit mais significativo da palavra, ou seja, o bit mais a esquerda, para representar o sinal do número:

O sinal + é representado pelo bit: **0**.
O sinal **-** é representado pelo bit: **1**.

`Exemplo`: A = $\frac{1}{0}$ a_{n-2} a_{n-3} ... a_1 a_0 Agora imagine que o bit $\frac{1}{0}$ é o bit de sinal, e queremos fazer a extensão da palavra. Neste caso nós conservamos o bit mais significativo mais a esquerda e acrescentamos com os zeros a esquerda.

`Exemplo`: A = $(a_{n-1}a_{n-2}...a_2a_1a_0)_2$, temos aqui uma palavra de `n` bits, e queremos extender para `w` bits sendo `w` > `n`

logo, temos o seguinte: A = $(a_{n-1}$ 0 0 0 0 ... k a_{n-2} ... a_2 a_1 $a_0)_2$

Essa técnica vale para qualquer sinal, seja positivo ou negativo.

12.1.2 Inclusões de sinais

A inclusão de sinal trata-se de uma forma simples de converter uma representação binária sem sinal para uma representação com sinal e para realizar a conversão basta converter os níveis lógicos do número:

`Exemplo`: Converter o número $2 - (\mathbf{0}10)$, aqui, vale ressaltar que o bit destacado é a representação do sinal, neste caso, temos um 2 positivo. Para termos o 2 negativo, invertemos os valores, ou seja:

101. Ou seja, basta tirar o complemento de 1 da palavra.

Palavra (010) $->$ complemento de 1 $->$ (101), isso vale para qualquer número, inclusive o zero.

Deve-se tomar o devido cuidado quando for realizar operações com o complemento de 1, pode gerar confusões no percurso, por isso, vamos exemplificar:

Vamos considerar a seguinte subtração: (3 - 2)

A = 3 = 011, aqui o bit 0 significa que o númeoro é positivo.
B = 2 = 010.
Tiramos o complemento de 1 do número B = 010
B = 2 = 010 = $(101)_c1$

Agora sim, podemos fazer a soma A + (-B), ou seja: 011101
Inicialmente a conta parece não fazer muito sentido, porém, lembre-se que deve adicionar 1 no resultado da soma.

Usar o complemento de 1 para fazer a representações de sinais se torna problemática, pois haverá duas representações para o númeor zero, repare que temos o zero positivo: 000 e o zero negativo: 111. Ter duas representações para um mesmo número é algo que gera certo desconforto, por este motivo é mais saudável aos olhos e um alívio a maquina usar o complemento de 2.

As vantagens de se usar o complemento de 2 para representar os sinais é que teremos a representação de mais 1 número e também evitaremos duas representações para o número zero.

As relações ficam da seguinte forma, para uma palavra de 3 bits.

A - 000 = 0_{10}
B - 001 = $+1_{10}$
C - 010 = $+2_{10}$
D - 011 = $+3_{10}$

`Complemento de 2`

A - 111 = -1_{10}
B - 110 = -2_{10}
C - 101 = -3_{10}
D - 100 = -4_{10}

`Complemento de 1`
A - 111 = -0_{10}
B - 110 = -1_{10}
C - 101 = -2_{10}
D - 100 = -3_{10}

Observe que se a palavra estiver em representação de complemento de 2, devemos ignorar o primeiro digito mais significativo, pois este indica o sinal, devemos lembrar também que caso um número seja negativo, deve-se primeiro fazer a conversão para complemento de dois para depois trocar a base, isto é fundamental!

Umas das problemáticas para nós humanos é a comparação. Repare que $(001_2)_{c2} > (101_2)_{c2}$ não é nada intuitivo, porém, para o computador é indiferente.

12.1.3 Extensão em complemento de 2

Para representar uma palavra com sinal magnitude podemos apenas conservar o bit mais significativo, neste caso em complemento de dois, a técnica se altera, considere o seguinte:

A = $((a_{n-1}\ a_{n-2}\ ...\ a_2\ a_1\ a_0)_2)_{c2}$

O número A têm `n` bits e agora queremos `w` bits $\backslash w > n$

Neste caso, podemos apenas pegar o bit mais sinigicativo e extender `w` vezes a esquerda

A = $((a_w\ ...\ a_{n-1}\ a_{n-1}\ a_{n-1}\ a_{n-1}\ a_{n-1}\ a_{n-2}\ ...\ a_2\ a_1\ a_0)_2)_{c2}$

Perceba que esse tipo de extensão é ligeiramente diferente das demais. Por fim, sempre que formos adicionar um sinal na representação do número, precisaremos de um bit a mais para representar o sinal.

13 Álgebra Booleana

A álgebra booleana foi descrita e conceituada por George Boole. Em 1854, publicou seu trabalho no qual fundamenta a álgebra que permite a construção de computadores modernos.

Formalmente, a Álgebra Booleana é um sistema matemático composto por um conjunto de elementos, e que se utiliza somente de dois algarismos para representar os números ou as proposições: O zero (0) e o um (1). Esse sistema de numeração é chamado binário e tem grande utilidade na Lógica e na Teoria dos Conjuntos.

Esse assunto é discutido formalmente na matemática discreta, aqui, pretendemos focar em útilidades da álgebra para aplicações em ciência da computação, como por exemplo, funções, fatorações(`Mapas de Karnaugh`), alguns postulados até chegarmos em circuitos combinacionais e por fim na maquina de estados.

Pretendo discurtir ligeiramente automatos finitos, maquina de turing e linguagens formais. São assuntos nos quais me geram grande entusiasmo.

13.1 Principios Básicos da Álgebra Booleana

São dois príncipios fundamentais:

- `Princípio da não contradição`: Uma proposição não pode ser **verdadeira** e **falsa**;

- `Princípio do terceiro excluído`: Uma proposição só pode assumir um dos valores possíveis: ou é verdadeira ou é falsa, excluindo-se uma terceira hipótese.

13.2 Operações Básicas

Já visto em matemática discreta, porém, iremos reforçar:

`Conjunção ou também Multiplicação booleana:`
$(X\ e\ Y)$ $(X\ and\ Y)$ $(X \wedge Y)$ *ou ainda* $(X * Y)$

`Disjunção ou também Produto booleano:`
$(X\ ou\ Y)$ $(X\ or\ Y)$ $(X \vee Y)$ *ou ainda* $(X + Y)$

`Negação ou também Complemento:`
$(NaoX)$ $(notX)$ $(\neg X)$ *ou ainda* $(\bar{X})$

Todas essas expressões são também representadas em linguagens de programação.

13.3 Tabelas Verdade

Não nos aprofundaremos tanto nas tabelas verdades, são facilmente encontradas e também será tratada melhor em matemática discreta, aqui nos basta saber o seguinte:

Conjunção ($x \wedge y$): Verdadeiro se e somente se (x,y) forem verdadeiros.

Disjunção ($x \vee y$): Verdadeiro se pelo menos um operador (x ou y) for verdadeiro.

Negação ($\bar{x}$): Verdadeiro se e somente se (x) for falso.

Disjunção Exclusiva($x \oplus y$): Verdadeiro se e somente se apenas um dos operandos for verdadeiro. Se os dois operandos forem verdadeiros o valor resultado será falso, pois, só um pode ser verdadeiro.

13.4 Expressões Lógicas

Como na álgebra comum, podemos combinar as operações, formando as expressões lógicas. O resultaod de uma expressão lógica pode ser obtido aplicando-se cada operação, respeitando a ordem de precedência dos operadores.

Aqui vamos ter a convenção de que o valor verdadeiro é: (1) e o valor falso é: (0).

Exemplo de expressão:

Operador	Precedência
$\neg$	1
$\wedge$	2
$\vee, \oplus$	3
$\longrightarrow, \longleftrightarrow$	4

$\bar{1} \vee (0 \wedge 1)$
$\bar{1} \vee (0)$
$0 \vee 0$
0

13.4.1 Variáveis Booleanas

Como na álgebra comum, podemos deixar valores indeterminados, esse valores são chamados de variáveis booleanas.

Exemplo: $\bar{x} \wedge y \vee x \wedge \bar{y}$
Agora podemos substituir os valores de `x` e `y` e calcular a expressão.

13.5 Tabelas Verdade de Expressões Lógicas

Podemos construir as tabelas verdades de expressões lógicas, atribuindo todos os valores possiveis às variáveis.

Considere a seguinte expressão: $\neg X \wedge Y \vee X \wedge \neg Y$

X	Y	$\neg X$	$\neg Y$	$\neg X \wedge Y$	$X \wedge \neg Y$
0	0	1	1	$1 \wedge 0 = 0$	$0 \wedge 1 = 0$
0	1	1	0	$1 \wedge 1 = 1$	$0 \wedge 0 = 0$
1	0	0	1	$0 \wedge 0 = 0$	$1 \wedge 1 = 1$
1	1	0	0	$0 \wedge 1 = 0$	$1 \wedge 0 = 0$

Agora se nós juntarmos as duas últimas expressões, teremos o seguinte:

$(\neg X \wedge Y) \vee (X \wedge \neg Y)$, essa expressão é a já vista **Disjunção exclusiva**. A expressão toda pode ser escrita da seguinte forma: $(X \oplus Y)$, ou seja, a expressão é verdadeira, quando somente um dos valores é verdadeiro.

13.6 Funções Lógicas

Uma função lógica pode ser definida como um conjunto de entradas lógicas que terá como resultante uma expressão que será verdadeira ou falsa.
Considere a seguinte expressão: F(A, B, C) = $A \vee (\bar{B} \wedge C)$

Podemos observar na função F, que se o valor A for 1, a expressão $(\bar{B} \wedge C)$ é insignificante,

observe que $(A \vee (N_{0,1}))$ o valor de N pode assumir 1 ou 0 que o valor da função continua sendo 1, pois, $(1 \vee N)$ é 1.

verdade fornecida será visto com mais detalhe adiante, por enquanto, considere a seguinte tabela:

x	y	f(x,y)
0	0	1
0	1	0
1	0	0
1	1	1

Observe que as variáveis (x, y) quando assumem os valores 0 e 0 o resultado da função é 1. O mesmo ocorre quando assume os valores 1 e 1, ou seja, o resultado é sempre contrário a $x \oplus y$, logo temos que a expressão da função F(x,y) é $\neg(x \oplus y)$.

13.7 Regras Básicas da Álgebra Booleana

Usando as regras da álgebra booleana é possivel simplificar expressões. Todas as regras básicas aqui a serem descritas podem ser demonstradas construindo-se as duas tabelas verdades das expressões em ambos os lados da equivalências.

Propiedade — OU — E

Identidade: $(x \vee 1 = 1)$ — $(x \wedge 0 = 0)$

Elemento Neutro: $(x \vee 0 = 0)$ — $(x \wedge 1 = 0)$

Idempotência: $(x \vee x = x)$ — $(x \wedge x = x)$

Involução: $(\neg(\neg x) = x)$ — $(\neg(\neg x) = x)$

Complemento: $(x \vee \bar{x} = 1)$ — $(x \wedge \bar{x} = 0)$

Comutatividade: $(x \vee y = y \vee x)$ — $(x \wedge y = y \wedge x)$

Associatividade: $((x \vee y) \vee z = x \vee (y \vee z))$ — $((x \wedge y) \wedge z = x \wedge (y \wedge z))$

Cobertura: $(x \wedge (x \vee y) = x)$ — $(x \vee (x \wedge y) = x)$

Combinação: $((x \wedge y) \vee (x \wedge \bar{y}) = x)$ — $((x \vee y) \wedge (x/ \vee \bar{y}) = x)$

Consenso: $((x \wedge y) \vee (\bar{x} \wedge z) \vee (y \wedge z) = (x \wedge y) \vee (\bar{x} \wedge z))$ — $((x \vee y) \wedge (\bar{x} \vee z) \wedge (y \vee z) = (x \vee y) \wedge (\bar{x} \vee z))$

Lei de Morgan: $(\neg(x \vee y) = \bar{x} \vee \bar{y})$ — $(\neg(x \wedge y) = \bar{x} \vee \bar{y})$

13.8 Introdução a Indução Lógica

Para se compreender formalmente como funciona as técnicas de indução é interessante que se tenha consolidado os conhecimentos em técnicas de demonstrações por exaustão, esse contéudo será visto de forma exaustiva na disciplina de matemática discreta. Aqui iremos apenas introduzir a técnica de indução para que se possa demonstrar algumas simplificações das nossas expressões.

13.8.1 Princípio da Indução Matemática

Considere uma sentença `P()` que quando aplicada a uma variável `n`, possa ser considerada **Verdadeira** ou **Falsa**. Inicialmente a dependender da sentença P, ela pode ser verdadeira, porém, como poderiamos provar que a sentençã P é realemnte verdadeira para todo conjunto da variável `(N)`. Para isso usa-se o seguinte ideia:

Sempre que a sentença `P` aplicada a uma variável N que pertence a um determinado conjunto é **verdadeira**, saberemos que para todo e qualquer outro elemento pertencente a este conjunto a sentença `p` será verdadeira.

Formalmente temos o seguinte: ($\forall n \in$ (conjunto qualquer)) a sentença **P(n)** é verdade.

Para tanto precisamos das seguintes provas:

Que a sentença P para algum N do conjunto é verdade.

Supor que para algum K $\in$ conjunto qualquer, a sentença **P(k)** é verdade.

Que **P(K + 1)** é verdade.

13.9 Formas Normais (canônicas)

Toda expressão booleana pode ser escrita de uma forma padronizada, denominada forma normal ou forma canônica.

Temos a `Forma Normal Disjuntiva` (FND), onde se tem a **soma de Mintermos**.

E temos também a `Forma Normal Conjuntiva` (FNC), onde se tem **produto de Maxtermos**.

Essas expressões estão relacionados com a tabela verdade de cada expressão algébrica booleana. Observe o exemplo abaixo

Agora queremos obter a expressão da função (F), no qual nos foi fornecida sua tabela verdade. Podemos utilizar **Mintermos (FND)** ou **Maxtermos (FNC)**. Neste caso como temos somente 4 zeros, valerá mais a pena Maxtermos.

Para utilizarmos os `Maxtermos` devemos deixar o zero intacto e alterar para negado a variável que tenha o valor 1. Observe abaixo linha por linha.

A	B	C	F(A, B, C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabela 1: Tabela verdade de expressões algébricas

A	B	C	F(A, B, C)
0	0	0	$A \vee B \vee C$
0	0	1	1
0	1	0	$A \vee \neg B \vee C$
0	1	1	$A \vee \neg B \vee \neg C$
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabela 2: Maxtermo

Agora como queremos a expressão completa devemos então ter o produto de cada **Maxtermo**, ou seja

$$(A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C)$$

Por outro lado temos também os **Mintermos** para se obter uma expressão de uma tabela verdade, observe que neste caso devemos manter intacto o valor 1 e negar o valor 0 e devemos fazer isso em todas as saídas 1.

Agora que temos os `Mintermos`, devemos obter a **soma** destes, ou seja

A	B	C	F(A, B, C)
0	0	0	0
0	0	1	$\neg A \wedge \neg B \wedge C$
0	1	0	0
0	1	1	0
1	0	0	$A \wedge \neg B \wedge \neg C$
1	0	1	$A \wedge \neg B \wedge C$
1	1	0	$A \wedge B \wedge \neg C$
1	1	1	$A \wedge B \wedge C$

Tabela 3: Mintermos

$$(\neg A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge B \wedge C)$$

Pode haver certas confusões com a Forma Conjuntiva e Disjuntiva, e vou exemplificar melhor por partes.

FND - Está associada a Mintermos, ou seja, expressões com saídas 1. Deve-se ter o `produto de cada variável`, ou seja, produto = variável $\wedge$ variável ... Obtendo-se os produtos, deve-se realizar a soma deles, ou seja, FND = produto $\vee$ produto ... Temos aqui **soma de produtos**.

$$(v * v * v) + (v * v * v) + ...$$

FNC - Está associada a Maxtermos, ou seja, expressões com saídas 0. Deve-se ter a `soma das variáveis`, ou seja, soma = variável $\vee$ variável ..., repare que é o inverso da FND. logo, FND = soma $\wedge$ soma ... Temos aqui **produto de somas**.

$$(v + v + v) * (v + v + v) * ...$$

13.10 Fatorações Lógicas

Bem sabemos que todo e qualquer circuito digital tem uma expressão, logo, a simplificação dessas expressões levam a circuitos mais simplificados.

Para realizar as simplificações das expressões é necessário ter todo o conhecimento registrado até aqui, como as regras da álgebra booleana, equivalências, propiedades, etc...

Veremos duas formas básicas de simplificar expressões:
Fatoração; Mapa de Karnaugh.

Partimos do seguinte `Teorema`: Toda função lógica possui uma expressão que a define e pode ser descrita como **FND** ou também chamada: `Soma de produtos`.

`FNC` e `FND` são formas padrões de representações de expressões booleanas, como exemplificado nas seções anteriores, **FND** possui sua simplicidade por representar as variáveis no estado em que estão, por exemplo uma função F(A, B, C) = 0, 0, 1 oberve que em **FND** temos: $(\neg A \neg BC)$.

Agora em **FNC** a mesma função seria representada da seguinte forma: $(AB\neg C)$, o que pode soar um pouco contra-intuitivo, pois a variável de valor 1 seria negada.

13.10.1 Simplificações Lógicas

Observe a seguinte expressão:

$$(\bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D} + ABCD)$$

Podemos colocar o termo BC em evidência:

$$BC(\bar{A}\bar{D} + \bar{A}D + A\bar{D} + AD)$$

Agora podemos usar a associação nos termos D e $\bar{D}$:

$$BC(\bar{A}(\bar{D}+D)+A(\bar{D}+D))$$

Agora podemos utilizar o complemento, observe que (N + $\bar{N}$) = 1.

$$BC(\bar{A}+A)\ ,$$

logo

$$BC$$

Esse é o processo de simplificações de expressões por propiedades da álgebra booleana.

13.11 Mapas de Veitch - Karnaugh

Há métodos mais otimizado e visual para realizar esse processo de simplificação, foi introduzido em um artigo por Edward W. Veith e públicado em 1952.

Veitch introduziu o método visual, porém, era complicado de se entender e não sistematizado, em 1953 Karnaugh fez algumas contribuições com o método de Veitch introduzindo um conceito conhecido como **"Código de Gray"** onde as variáveis são introduzidas no mapa por diferença de 1 bit, isso facilitava a visualização das simplificações possiveis, também introduziu agrupamentos em potência de 2, isso tornou a simplificação mais eficiênte e sistemática, karnaugh simplificou o método que era complicado de entender, semelhante o que aconteceu com Von Neumann e nosso querido amigo Turing.

13.11.1 Código de Gray

A códificação de gray consiste em transpor bit a bit de coluna a coluna da matriz ou seja, a cada passo na matriz se anda um bit, por exemplo:
Imagine que essa sequência: 00, 01, 11, 10, 00 ... isso significa que a cada coluna que se locomove a direita é de 1 bit, logo, para a esquerda também é de 1 bit.

Agora fica intuitivo imaginar que para cima ou para baixo a diferença também será de 1 bit, ou seja:
O mapa de Karnaught é bastante eficiênte para expressões com 4 variáveis, com 5 variáveis em

ab/cd	00	01	11	10
00				
01				
11				
10				

diante o mapa pode ficar mais complexo, neste caso, é recomendado outros algoritmos como o do `Quine-McCluskey`.

Após a codificação de Gray, associamos cada bit a uma variável, por exemplo em uma função com 4 variáveis F(A, B, C, D), na primeira linha da matriz poderiamos ter algo como (ab), onde os bits 00, 01, 11 e 10 representa o estado atual de cada variável, o mesmo se aplica a variável (cd), conforme exemplificado na tabela acima.

Agora mapeamos a saida S da tabela verdade da função observando os `Mintermos`, e associando as variáveis correspondentes, observe um exemplo com 3 variáveis:
Agora podemos mapear os grupos de `mintermos` com potência de 2.

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

Tabela 4: Tabela Verdade com 3 variáveis

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

Tabela 5: Mapa de Karnaught

Primeiro grupo com 4 `mintermos`: A é a única variável que não se altera com a alternância dos bits.

Agora só temos grupo com 2 `mintermos`: $B\bar{C}$ ou seja, a minha expressão simplificada fica:

$$A + (B * \bar{C})$$

Repare na diferença entre a expressão sem simplificação:

$$(\bar{A} * B * \bar{C}) + (A * \bar{B} * \bar{C}) + (A * \bar{B} * C) + (A * B * \bar{C}) + (A * B * C)$$

14 CMOS e Portas Básicas

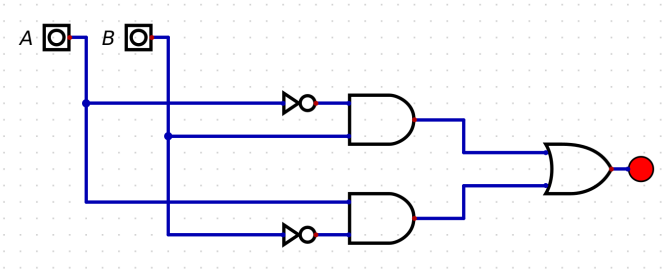
15 Circuitos Combinacionais

Podemos entender os circuitos combinacionais como a construção física das expressões algébricas booleanas, e podemos entender a construção física como a combinação de diferentes portas lógicas para expressar um resultado.

Vejamos um exemplo básico, a expressão: $\neg ab \vee a\neg b$

Pode ser expressa físicamente pelo seguinte circuito

Figura 1: circuitos/universais/xor.dig

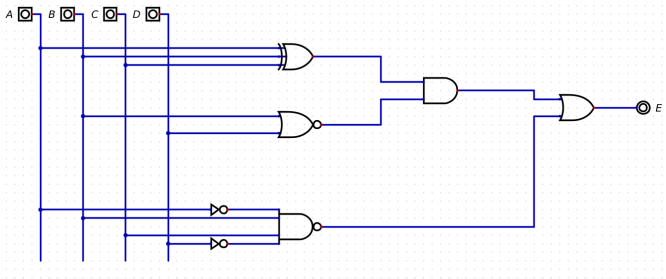


Reapre que o operador (E) na expressão é implícito enquanto no circuito sua construção é explícita. A expressão poderia também ser representada da seguinte forma: $(\neg a \wedge b) \vee (a \wedge \neg b)$, ou seja, são apenas formas diferentes de representar a mesma expressão, e, para melhorar a leitura, corriqueiramente podem ter representações implícitas nas expressões.

Observe agora a expressão: $((a \oplus b \oplus c) \wedge \neg(b \vee d)) \vee \neg(\neg a \wedge b \wedge c\neg d)$

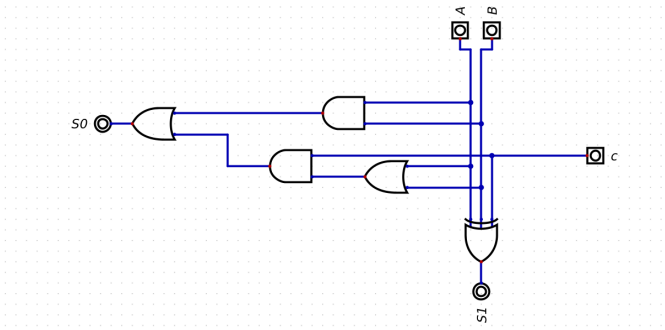
Pode ser expressa físicamente pelo seguinte circuito:

Figura 2: circuitos/combinacionais/circuito-F.dig



Para finalizar e consolidar os conhecimentos em circuitos combinacionais vamos considerar o seguinte circuito: Repare que neste caso especifico temos duas saidas: S0 e S1 e as entradas a, b,

Figura 3: circuitos/combinacionais/somador-completo.dig



c.

Podemos obter as expressões a partir do curcuito acima, observe que para a saida S0 temos: $a \wedge e \vee (a \wedge b) \wedge c$ enquanto para saida S1 temos: $a \oplus b \oplus c$.
Podemos perceber então a relação direta entre a construção de circuitos combinacionais utilizando blocos, ou seja, portas lógicas e também a relação entre as expressões algébricas obtidas por uma análise do circuito.

15.1 Blocos Combinacionais Básicos