UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
CURSO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

HENDRICK SILVA FERREIRA & GUILHERME MIRANDA DE ARAÚJO

LABORATÓRIO DE CIRCUITOS

BOA VISTA, RR
2024

LABORATÓRIO DE CIRCUITOS

Avaliação de Barramento de Circuitos Digitais, apresentado como requisito de obtenção de nota parcial da disciplina de Arquitetura e Organização de Computadores - DCC 301.

Orientador: Prof. Dr. Hebert Oliveira Rocha

BOA VISTA, RR

2024

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O presente relatório é referente aos circuitos desenvolvidos para a Avaliação de Barramento da disciplina Arquitetura e Organização de Computadores DCC 301. Os testes e descrições das lógicas binárias e aritméticas dos circuitos estão dispostos de acordo com as questões da avaliação, por isto não há necessariamente uma relação de complexidade crescente entre as questões e, por isto, a ordem recomendável de leitura do desenvolvimento dos componentes é: Porta Lógica XOR, Multiplexador, Extensor de Sinal, *Flip-Flop JK* e *Flip-Flop D*, Somador de 8 bits com Valor 4, Somador de 8 bits, Contador Síncrono, Banco de Registradores, Memória RAM, Memória ROM, Banco de registradores, Máquina de Estados, ULA 8 bits, Detector de paridade ímpar, Decodificador de 7 segmentos e, por fim, Detector de número primo. O Software utilizado para realizar os testes foi o Software Logisim, mais especificamente a versão 2.7.1(2024).

2. COMPONENTES

2.1. *REGISTRADOR FLIP-FLOP* JK E *FLIP-FLOP* D

Figura 1 - *Flip-Flop* JK

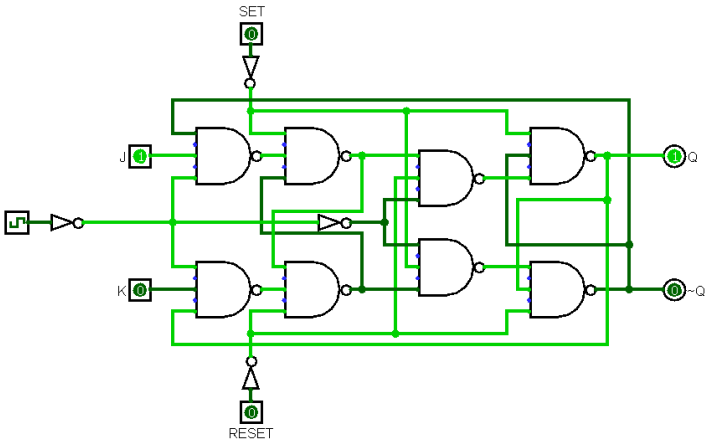
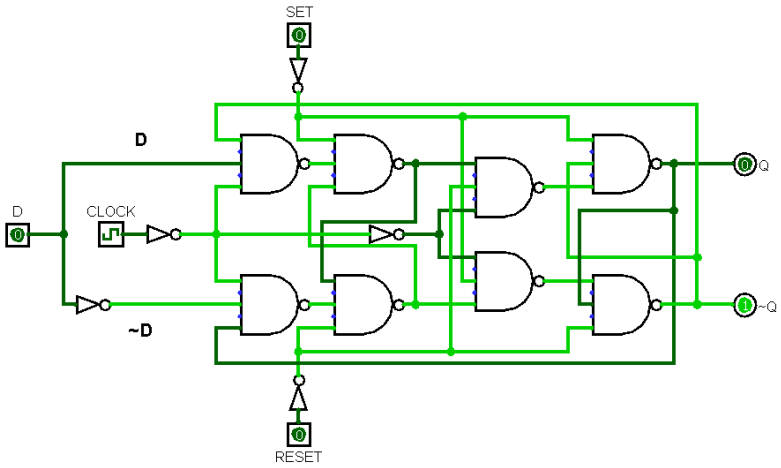


Figura 2 - *Flip-Flop* D



Os *Flip-Flops* são componentes da arquitetura de computadores que possuem em sua tabela-verdade uma característica destoante de outros componentes, a presença do último valor de saída efetuado pelo componente como próximo valor de saída em determinados casos, mesmo sem conhecimento do valor específico do mesmo, ou seja, desta forma é possível armazenar valores anteriores e trabalhar com os mesmos, podendo alterná-los, negá-los ou mantê-los para o funcionamento de demais circuitos.

2.1.1. Descrição pinos e lógica

O *Flip-Flop* JK possui quatro entradas: J, K, *Preset* e *Clear*, fora o Clock, sendo as entradas *Preset* e *Clear* correspondentes às respectivas entradas *SET* e *RESET* na Figura 1, além disto possui dois valores de saída: Q e ~Q. Por outro lado o *Flip-Flop* D possui apenas três entradas D, *Preset* e *Clear*, fora o Clock, e, de forma parecida com o componente *Flip-Flop* JK desenvolvido, as entradas *Preset* e *Clear* são correspondentes às respectivas entradas *SET* e *RESET* na Figura 2, e também

possui dois valores de saída: Q e ~Q. O modelo escolhido de *Flip-Flop* JK e *Flip-Flop D* foi o modelo Chefe-Servente, que divide o *Flip-Flop* em duas partes, como visto na Figura 3, onde existem dois *Flip-Flop* RS, uma versão mais simples de *Flip-Flop*, conectados, onde o Clock do *Flip-Flop* Servente é negado, impedindo o erro no caso de entradas 1 e 1, originalmente presente em *Flip-Flops* RS, como é demonstrado na Tabela 1.

Tabela 1 - Tabela-Verdade Flip-Flop RS Completa

R	S	Qant	Qs
0	0	0	0
0	0	1	1
1	0	0	1
1	0	1	1
0	1	0	0
0	1	1	0
1	1	0	ERRO
1	1	1	ERRO

2.1.2. Testes do componente

Ao realizar os testes foi possível gerar as tabelas-verdade do circuito do *Flip-Flop* JK e do circuito do *Flip-Flop* D e, após isto, simplificá-lo, como pode ser visto nas tabelas Tabela 2, Tabela 3, Tabela 4 e Tabela 5, onde o valor Qant está relacionado ao valor de Q anterior e Qs está relacionado saída do valor de Q atual.

Tabela 2 - Tabela-Verdade Flip-Flop JK Completa

J	K	Qant	Qs
0	0	0	0
0	0	1	1
1	0	0	1
1	0	1	1
0	1	0	0
0	1	1	0
1	1	0	1
1	1	1	1

Tabela 3 - Tabela-Verdade Flip-Flop JK Simplificada

J	K	Qs
0	0	Qant
1	0	1
0	1	0
1	1	~Qant

Tabela 4 - Tabela-Verdade Flip-Flop D Completa

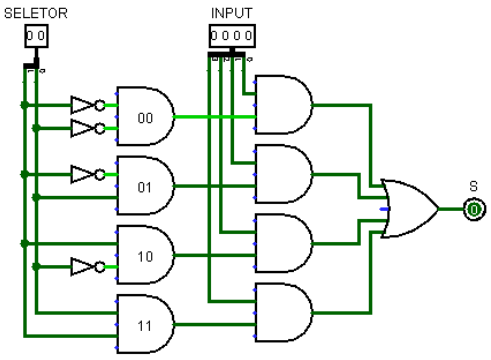
D	Qant	Qs
0	0	0
0	1	0
1	0	1
1	1	1

Tabela 5 - Tabela-Verdade Flip-Flop D Simplificada

D	Qs
0	0
1	1

2.2. MULTIPLEXADOR COM 4 OPÇÕES DE ENTRADA

Figura 3 - Multiplexador



O multiplexador é um componente extremamente comum em arquiteturas de computadores, pois é respondável por controlar o fluxo de dados em circuitos digitais.

2.2.1. Descrição pinos e lógica

O multiplexador possui duas entradas, *INPUT* e o SELETOR, possuindo, respectivamente 4 e 2 bits cada, além disto possui uma saída, S. O componetnte é responsável pelo desvio de fluxo em um circuito, ou seja, o valor de cada saída em um multiplexador será igual ao input apenas caso o valor do seletor desvie a corrente no sentido do mesmo.

2.2.2. Testes do componente

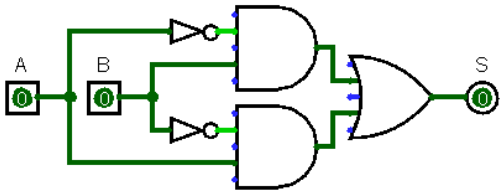
Ao realizar os teste com o multiplexador foi possível gerar um tabela-verdade como pode ser visto nas tabelas Tabela 6, sendo “X”, “Y”, “Z” e “W” os valores dos bits de INPUT, que variam de acordo com a entrada do dada pelo funcionamento do circuito.

Tabela 6 - Tabela-Verdade Multiplexador

INPUT	SELETOR	S
XYZW	00	X
XYZW	01	Y
XYZW	10	Z
XYZW	11	W

2.3. PORTA LÓGICA DO XOR

Figura 4 - Porta Lógica *Xor*



O componente *Xor* é um componente comummente utilizado em diversas arquiteturas de circuitos digitais, pois o mesmo obriga que apenas um dos valores seja verdadeiro para que a saída também seja.

2.3.1. Descrição pinos e lógica

O componente *Xor*, também conhecido como “ou exclusivo” funciona com a junção de dois componentes do tipo E, onde as entradas são negadas de forma alternada e seus resultados após o componente E são conectados à um único componente OU, desta forma, OU o primeiro valor (neste caso, o input A) é igual a 1 e o segundo valor (neste caso, o input B) é igual a 0, OU o primeiro valor é igual a 0 e o segundo valor é igual a 1, para que a saída final seja verdadeira.

2.3.2. Testes do componente

A tabela verdade do componente *XOR* já era previamente conhecida, pois o mesmo é utilizado em diversas outras arquiteturas, como é demonstrado por Metrópole Digital, na publicação “Circuitos *Exclusive-OR* (*XOR*) e *Exclusive-NOR* (*XNOR*)”, e, após teste, foi possível obter os mesmos resultados, como é demonstrado na Tabela 8.

Tabela 7 - Tabela-Verdade XOR

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

2.4. SOMADOR 8 BITS COM VALOR 4

Figura 5 - Circuito Completo Somador 8 bits

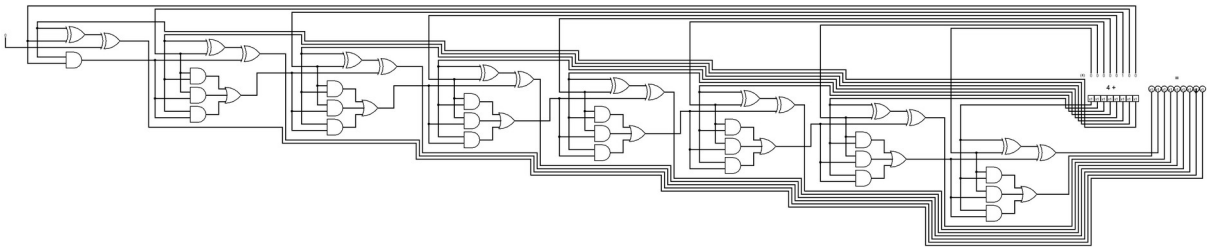
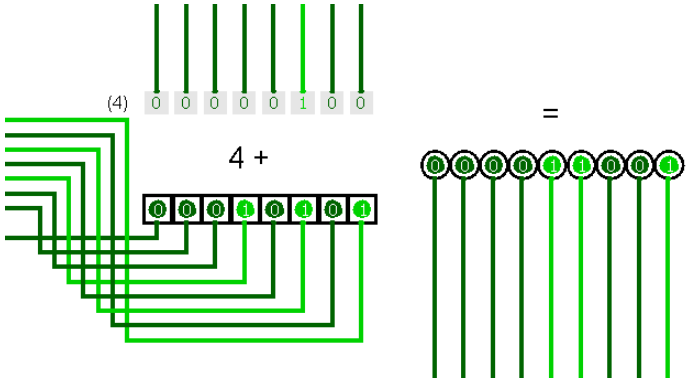


Figura 6 - Valores de Entrada Somador 8 bits



O Somador é um componente aritmético que soma valores binários utilizado para efetuar cálculos pelo computador, sendo que o seu tamanho varia de acordo com a quantidade de bits dos valores de entrada.

2.4.1. Descrição pinos e lógica

O funcionamento do Somador de 8 bits pode ser entendido a partir do funcionamento de uma secção do mesmo, como pode ser visto na Figura 7, que possui três valores de entrada de 8 bits, A, B, e *Cin*, sendo A e B os valores de entrada atuais e *Cin* o valor excedente da soma da secção anterior, além disto possui duas saídas de 1 bit, S e Cout, onde o valor de S corresponde ao valor da soma e Cout corresponde ao valor excedente da soma atual. Neste circuito o componente *XOR 1* impede que soma de dois valores iguais a 1 provenientes de A e B resultem em 1, pois 1+1 em binário, diferente do sistema decimal comummente utilizado, resulta em 10, não em 2, pois o valor 2 não existe no sistema binário, da mesma forma o componente *XOR 2* impede que, caso o valor do *XOR 1* seja igual a 1 e o valor de *Cin* seja igual a 1, que o valor de S seja igual a 1. O Grupo de E determina o valor de *Cout*, de forma que caso o valor de A **e** B sejam iguais a 1, ou caso A **ou** B sejam iguais a 1 e *Cin* seja igual a 1, então o *Cout* será igual a 1, como pode observado na Tabela 9.

Figura 7 - Secção de Circuito Somador 8 bits

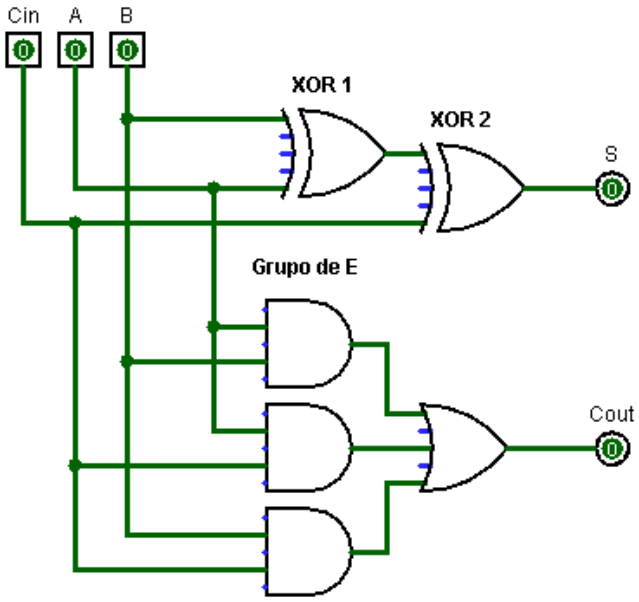


Tabela 8 - Tabela-Verdade Secção de Circuito Somador 8 bits

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

2.4.2. Testes do componente

O circuito do Somador de 8bits é formado por diversas secções de somador, presente na Figura 7, encadeadas, como demonstrado na Figura 5, de forma que 8 valores de 1 bit de entrada (ou uma única entrada de 8 bits) e o valor fixo de 100 (que equivale ao valor 4 em binário), são somados, como demonstrado na Figura 6.

2.5. MEMÓRIA ROM

Figura 8 - Circuito Completo Memoria ROM

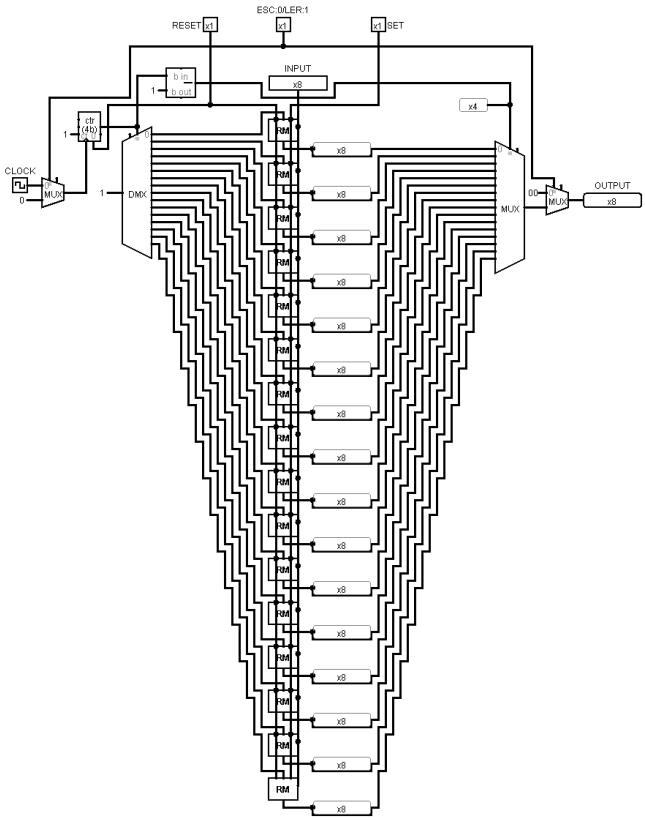


Figura 9 - Entradas e Saídas Memória ROM

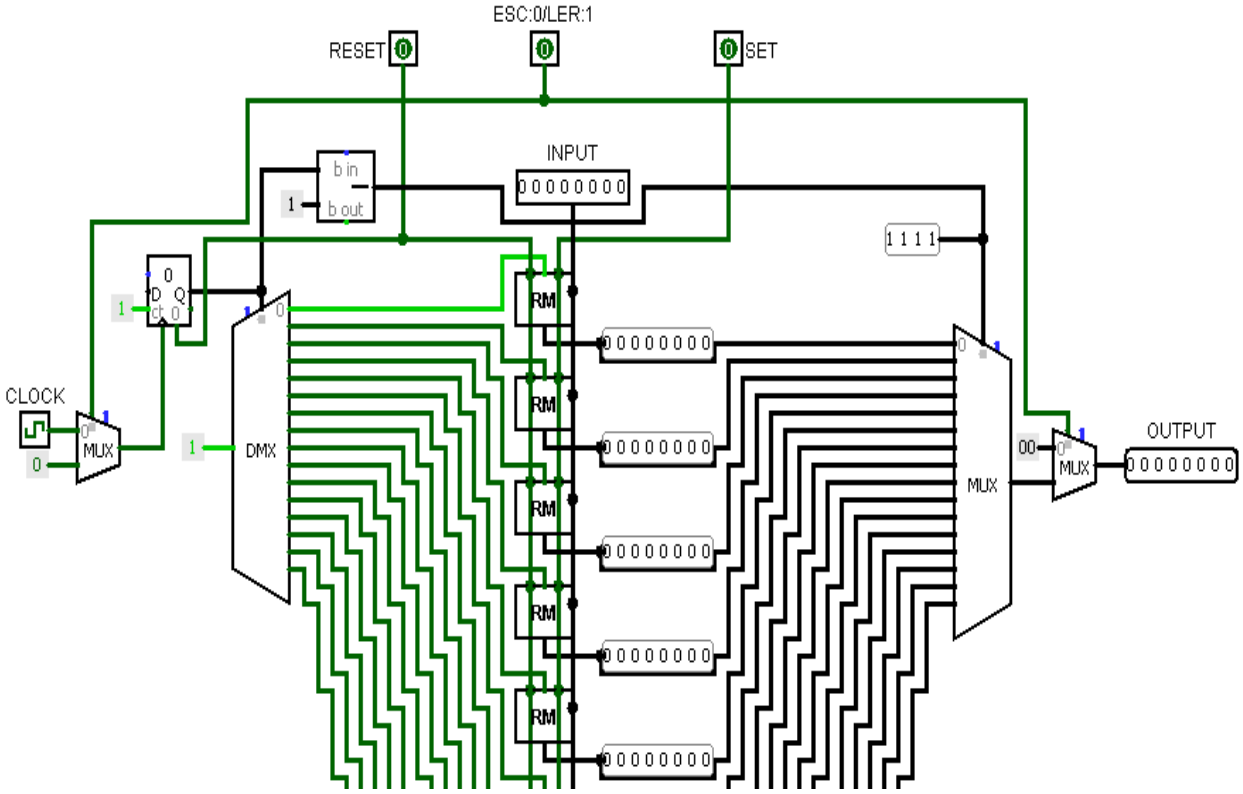
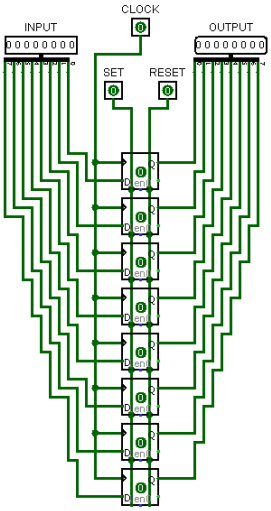


Figura 10 - Valor de Memória



A Memória ROM é uma memória que guarda instruções de forma sequencial, gerando algo parecido com um “histórico”, conceito comum em outras tecnologias, de forma que as instruções anteriores possam ser consultadas.

2.5.1. Descrição pinos e lógica

A Memória ROM, que possui o circuito presente na Figura 8, é uma memória sequencial, ou seja, guarda os valores de entrada em sequencia, neste caso uma pseudo-intrução de 8 bits foi adotada, com 16 espaços de memória disponíveis. O fluxo do circuito se inicia na entrada Ler/Esc, presente na Figura 9, que determina se a operação será de leitura ou escrita de valores na memória, em caso de escrita, o fluxo pode ser observado iniciando no multiplexador conectado com o Clock, que, caso a entrada Ler/Esc seja 0, o Contador Síncrono de 4 bits é incrementado, determinando o seletor no desmultiplexador com entrada constante 1, armazenando o valor de INPUT no Valor de Memória, representado na Figura 10, correto, salvando bit a bit em cada *Flip-Flop D* presente no mesmo. Caso a entrada Ler/Esc seja igual a 1, o valor atual do Contador Síncrono é decrementado em 1, enviando para o valor de saída OUTPUT a última instrução registrada.

2.5.2. Testes do componente

Após testes um problema de escrita de valores pôde ser percebido, pois o valor de entrada do principal do desmultiplexador, assim como o seu Seletor eram energizados ao mesmo tempo, gerando um erro no programa e salvando o mesmo valor em dois Valores de Memória diferentes, no entanto, ao energizar apenas o seletor do desmultiplexador com o valor do Contador Síncrono e atribuindo a entrada principal o valor fixo 1.

2.6. MEMÓRIA RAM

Figura 11 - Memória RAM

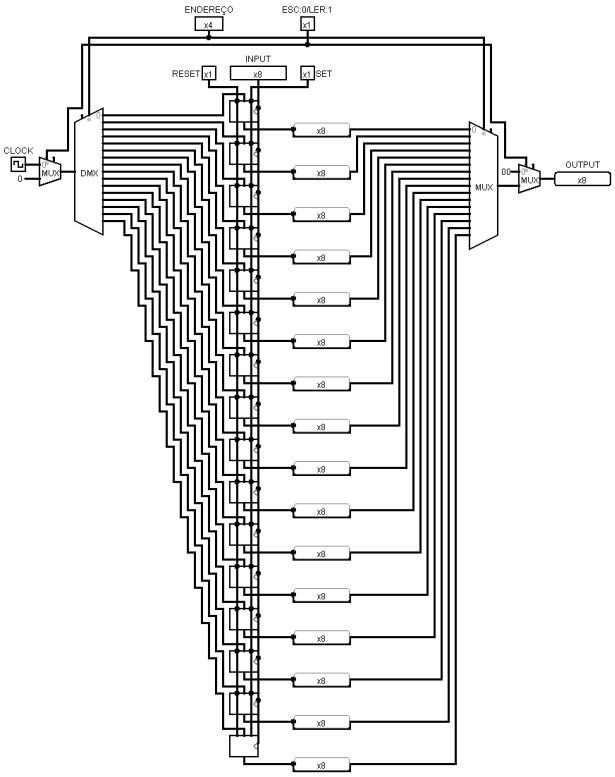
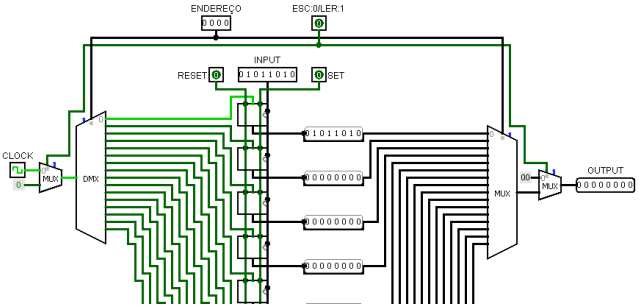


Figura 12 - Entradas e Saídas Memória RAM



Diferente da Memória ROM, a Memória RAM é uma memória que guarda instruções de forma não-sequencial, ou seja, o local onde o valor será armazenado é designado pela entrada de Endereço.

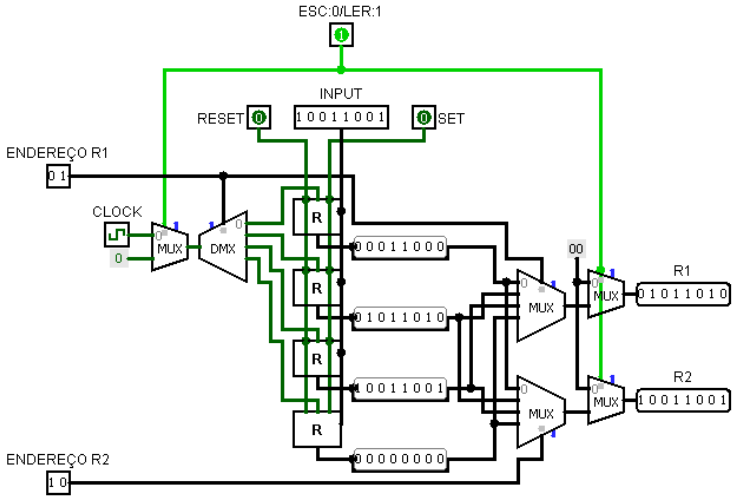
2.6.1. Descrição pinos e lógica

A Memória RAM é uma memória não-sequencial, ou seja, guarda os valores de entrada de acordo com o endereço que lhe é atribuída, neste caso um valor de 8 bits foi adotado, com 16 espaços de memória disponíveis. O fluxo do circuito se inicia na entrada

Ler/Esc, presente na Figura 12, que determina se a operação será de leitura ou escrita de valores na memória, em caso de escrita, o fluxo pode ser observado iniciando no multiplexador conectado com o Clock, que, caso a entrada Ler/Esc seja 0, o valor de Clock é atribuído ao desmultiplexador e o sinal é enviado ao endereço determinado pela entrada Endereço, armazenando o valor de INPUT no Valor de Memória, presente na FIgura 10 designado. Caso a entrada Ler/Esc seja igual a 1 o valor de Endereço é atribuído ao seletor do multiplexador conectado com os Valores de Memória, que irá se conectar com multiplexador que conectado com o OUTPUT, que irá selecionar a saída da memória por conta da entrada Ler/Esc, que é conectada ao seu seletor e irá enviar o valor para a saída OUTPUT.

2.7. BANCO DE REGISTRADORES

Figura 13 - Banco de Registradores



O Banco de Registradores armazena valores para serem utilizado em tempo de execução, realizando operações lógicas e aritméticas com o mesmos.

2.7.1. Descrição pinos e lógica

O Banco de Registradores, representado na Figura 13, tem seu funcionamento parecido com a Memória RAM, usando de demultiplexadores e multiplexadores, com o auxílio de entradas de endereço para armazenas valores em Rgistradores, que possuem o mesmo circuito do Valor de Memória, presente na Figura 10, sendo sua maior diferença a quantidade de entradas e saídas, pois o mesmo possui duas entradas de endereço, ENDEREÇO R1 e ENDEREÇO R2, ambos com 2 bits, uma entrada de valor, INPUT e uma entrada de controle, Ler/Esc, além disto possui duas saídas de valores, R1 e R2. Quando o

valor de Ler/Esc é igual a 0 o valor de ENDEREÇO R1 é utilizado para selecionar o Registrador para ser salvo o valor de INPUT, por outro lado caso o valor de Ler/Esc seja igual a 1, o valor da saída R1 será igual ao valor do Registrador no endereço da entrada ENDEREÇO R1, da mesma forma o valor da saída R2 será igual ao valor do Registrador no endereço da entrada ENDEREÇO R2.

2.8. SOMADOR 8 BITS

Figura 14 - Circuito Completo Somador 8 bits

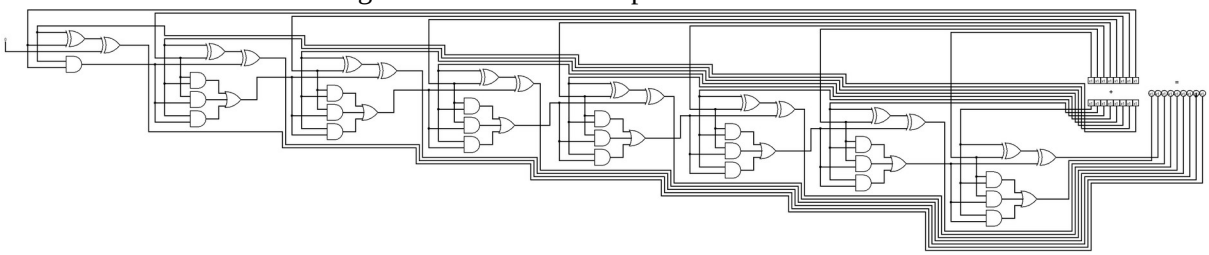
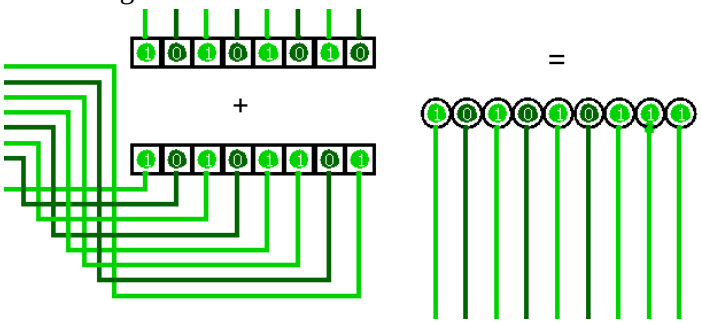


Figura 15 - Valores de Entrada Somador 8 bits



Assim como no ponto 2.4, que também se trata de um Somador 8 bits, o Somador 8 bits de dois valores quaisquer de 8 bits é um componente aritmético que soma valores binários utilizado para efetuar cálculos pelo computador.

2.8.1. Descrição pinos e lógica

Da mesma forma que no ponto 2.4.1 o funcionamento do Somador de 8 bits pode ser entendido a partir do funcionamento de uma secção do mesmo, possuindo como único diferencial neste caso que não há valores fixos de entrada, podendo somar dois valores de 8 bits quaisquer.

2.8.2. Testes do componente

O circuito do Somador de 8bits é formado por diversas secções de somador, presente na Figura 7, encadeadas, como demonstrado na Figura 14, onde dois valores de 8 bits quaisquer são somados, como demonstrado na Figura 15.

2.9. DETETOR DE SEQUÊNCIA BINÁRIA

Um **detector de sequência binária** é um circuito digital que identifica uma sequência específica de bits em um fluxo de dados binários. Quando a sequência desejada é detectada, o circuito gera uma saída indicando que a sequência foi encontrada.

Componentes Principais

1. **Entrada:** Um único bit de dados (X) que chega a cada ciclo de clock.
2. **Estado Atual:** O circuito mantém informações sobre os bits já vistos usando **flip-flops**.
3. **Saída:** Um sinal binário (Z) que é ativado (1) quando a sequência é detectada.
4. **Máquina de Estados Finitos (FSM):** Controla a lógica de transição de estados, dependendo do histórico da entrada.

Funcionamento Geral

- O detector verifica os bits que chegam em série, comparando-os com a sequência-alvo.
- Um **estado inicial** é usado para começar a detecção.
- Cada entrada de bit causa uma **transição de estado** baseada no próximo bit e no estado atual.
- Quando todos os bits da sequência-alvo são correspondidos, a saída Z=1 é ativada.

Exemplo: Detector da sequência "1011"

Estados

Os estados representam o progresso na detecção da sequência:

- **Estado S0:** Nenhum bit da sequência foi reconhecido.
- **Estado S1:** O primeiro "1" foi detectado.
- **Estado S2:** A sequência "10" foi detectada.
- **Estado S3:** A sequência "101" foi detectada.
- **Estado S4:** A sequência "1011" foi detectada.

Transições de Estados

1. Partindo de S0:
    - Entrada X=1: Vai para S1.
    - Entrada X=0: Permanece em S0.
2. Partindo de S1:
    - Entrada X=0: Vai para S2.
    - Entrada X=1: Permanece em S1.
3. Partindo de S2:
    - Entrada X=1: Vai para S3.
    - Entrada X=0: Retorna para S0.
4. Partindo de S3:
    - Entrada X=1: Vai para S4 (sequência detectada).
    - Entrada X=0: Vai para S2.
5. Partindo de S4:
    - Entrada X=1: Permanece em S1 (reinicia a detecção).
    - Entrada X=0: Vai para S0.

Implementação

Um detector de sequência pode ser implementado usando:

1. **Flip-flops:** Para armazenar os estados.
2. **Portas lógicas:** Para determinar as transições entre estados com base na entrada.
3. **Circuitos combinacionais:** Para gerar a saída.

Diagrama FSM (Máquina de Estados Finitos)

O diagrama mostra os estados e transições, ajudando a projetar o circuito lógico.

Aplicações

1. **Detecção de padrões em comunicação digital:** Encontrar sequências específicas em fluxos de dados.
2. **Circuitos de controle:** Responder a padrões de entrada em sistemas embarcados.
3. **Sistemas de criptografia:** Detectar marcas ou assinaturas binárias específicas.

Se precisar de um exemplo prático ou do circuito lógico, posso ajudar a detalhar!

Figura 16 – Detector de sequência binária

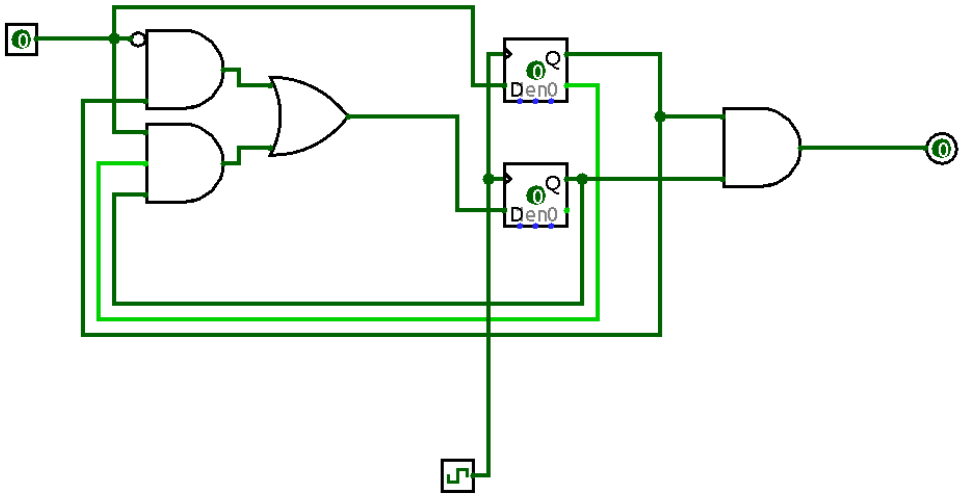
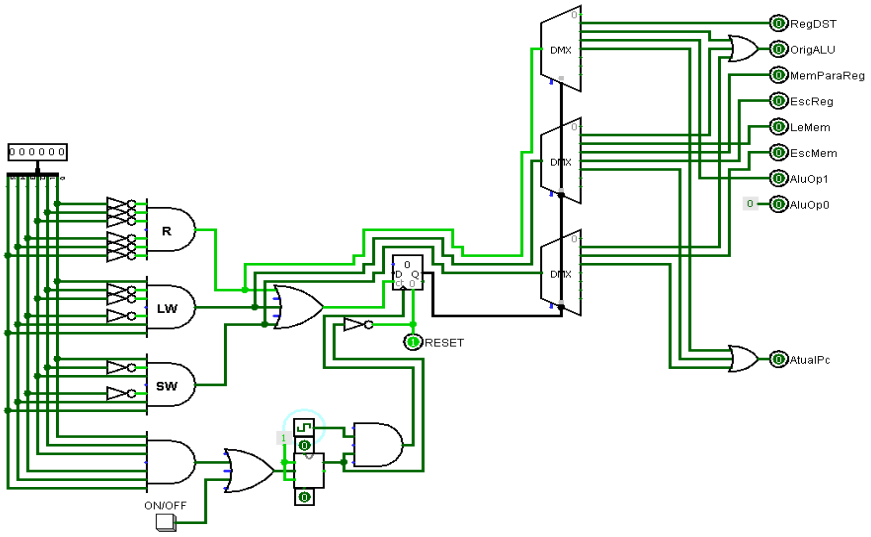


Tabela 9 – Tabela sequência binária

A	X
0	0
1	0

2.10. UNIDADE DE CONTROLE 16 BITS

Figura 16 - Unidade de Controle 16 bits



Unidade de Controle de 16 bits ordena os valores de entradas para o controle de outros componentes de um Processador de 16 bits.

2.10.1. Descrição pinos e lógica

A unidade de Controle 16 bits funciona a partir da saída de valores de "*flags*" periódicos de acordo com o valor de entrada de código de instrução. As *flags* são trilhas que são responsáveis por controlar outros componentes no sistema, desta forma, cada classe de instrução possui diferentes ordens de diferentes *flags* para o seu ideal funcionamento.

2.10.2. Testes do componente

Figura 17 - Unidade de Controle 16 bits Classe R (Caso 1)

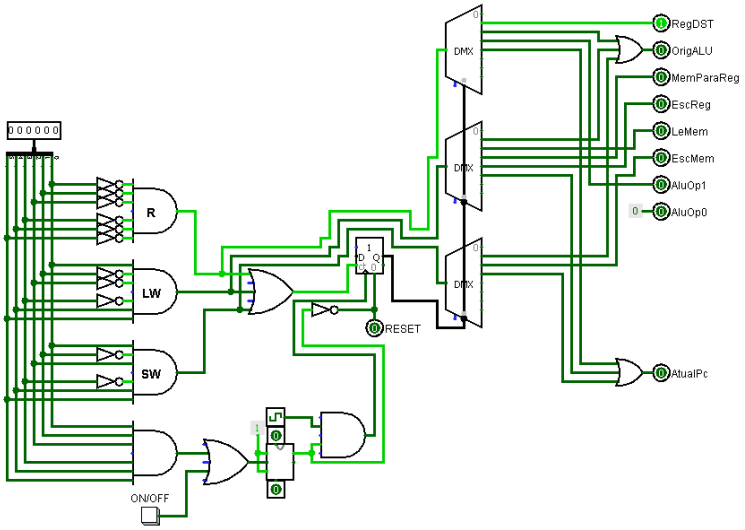


Figura 18 - Unidade de Controle 16 bits Classe R (Caso 2)

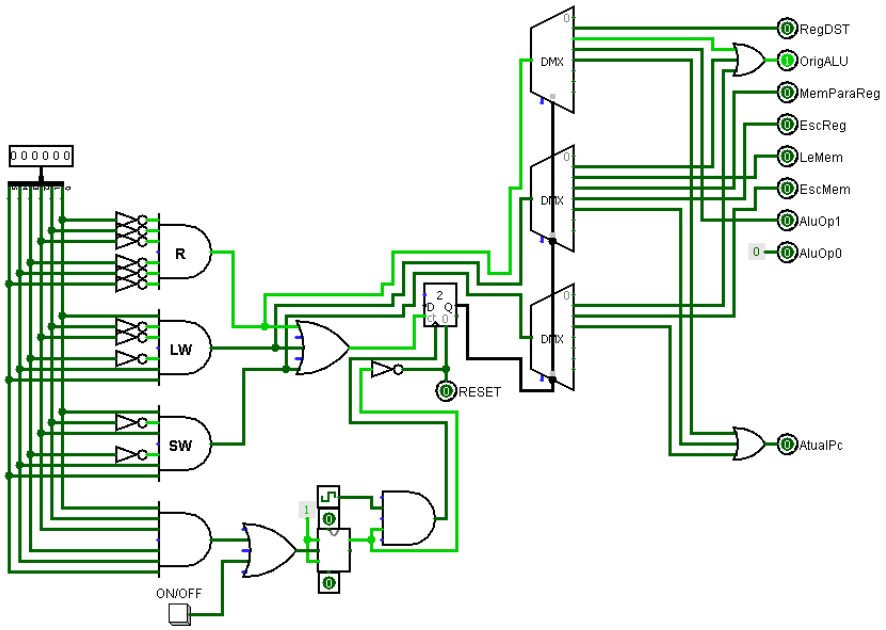


Figura 19 - Unidade de Controle 16 bits Classe R (Caso 3)

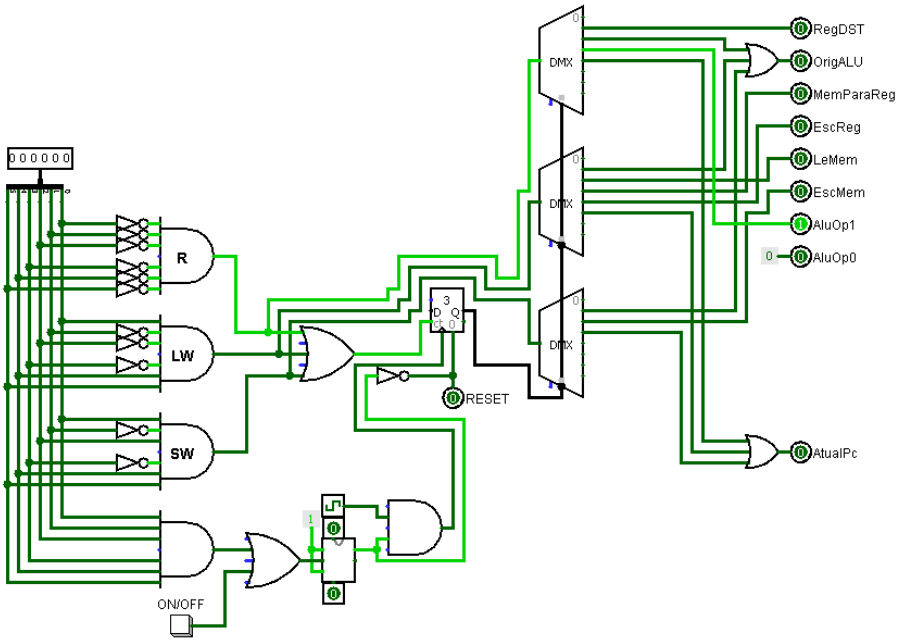
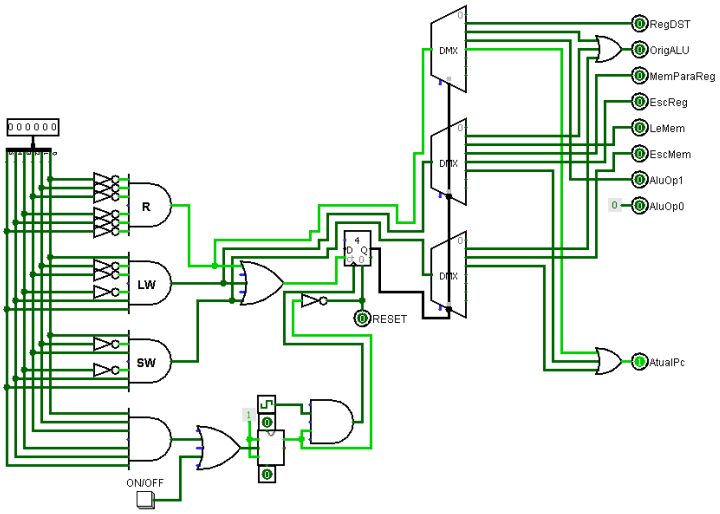


Figura 20 - Unidade de Controle 16 bits Classe R (Caso 4)



Após os testes da Unidade de Controle 16 bits, presentes nas figuras Figura 17, Figura 18, Figura 19 e Figura 20, realizando uma instrução de classe R foi possível perceber que as *flags* RegDST, OrigALU, AluOp1 e AtualPC foram utilizadas.

2.11. ULA 8 BITS

Figura 21 - ULA 8 bits

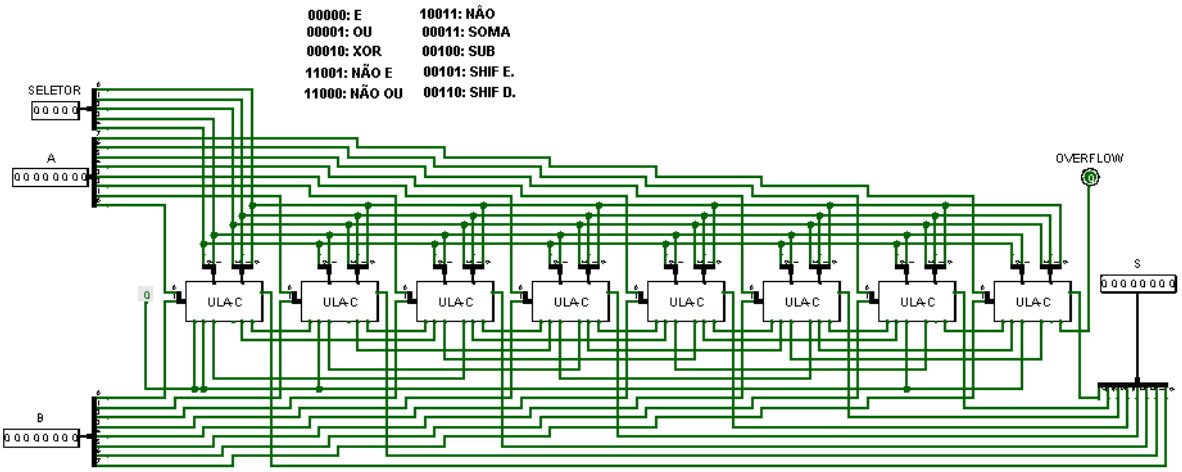
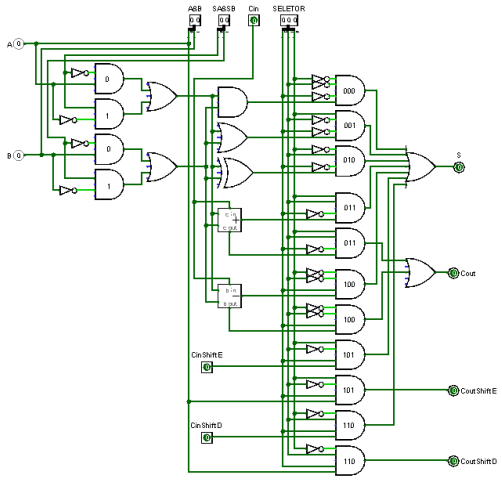


Figura 22 - ULA 1 bit



A ULA 8 bits, Figura 21, é formada por 8 ULAs de 1 bit, Figura 22, de forma que o valor final de OUTPUT é formado pelo conjunto de valores calculados por cada ULA de 1 bit formando um valor de saída de 8 bits.

2.11.1. Descrição pinos e lógica

A ULA de 1 bit possui oito valores de entrada, A, B, SA, SB, Cin, CinShiftE, CinShiftD, que possuem 1 bit e SELETOR, que possui 3 bits. As entradas A, B, SA, SB correspondem aos bits de entrada, sendo A e B os bits para cálculo, SA e SB para selecionar se os valores A e B serão negados, as entradas Cin, CinShiftE, CinShiftD são entradas de controle aritmético, sendo que seus valores advem de outras ULAs de 1 bit, e, por útlimo o SELETOR é responsável por selecionar a operação a ser realizada, além disto, a ULA de 1 bit possui 3 valores de saída, S, Cout, CoutShiftE e CoutShifD, de forma que o valor de saída S corresponde ao valor final da operação aritmética, o valor Cout corresponde ao valor excedente da operação e os valores CoutShiftE e CoutShifD, são utilizados para realizar o *Shift* de dados binários para esquerda e direita respectivamente.

2.11.2. Testes do componente

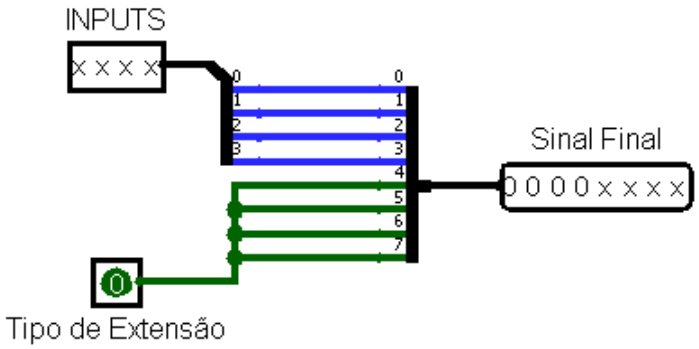
Tabela 10 - Tabela-Verdade Códigos ULA

CÓDIGO	OPERAÇÃO
00000	E
00001	OU
00010	XOR
11001	NÃO E
11000	NÃO OU
10011	NÃO
00011	SOMA
00100	SUBTRAÇÃO
00101	SHIFT ESQUERDA
00110	SHIFT DIREITA

Ao estruturar as ULAs de 1 bit é possível passar uma única entrada de SELETOR que servirá para todos as ULAs de 1 bit e dividir bit a bit os valores de A e B e, após isto, agrupar os valores em uma única saída OUTPUT. Os códigos correspondentes as operações aritméticas estão relacionados na Tabela 10.

2.12. EXTENSOR DE SINAL 4 PARA 8 BITS

Figura 23 - Extensor de Sinal



O Extensor de Sinal de 4 para 8 bits é útil para transformar um valor previamente obtido para um tamanho maior de instrução, quando isto é necessário para o funcionamento do circuito, existindo

dois tipos de extensão: extensão de valor 0 e extensão de valor 1, completando o bits excedentes com o valor do tipo de extensão.

2.12.1. Descrição pinos e lógica

O Extensor de Sinal, Figura 23, possui duas entradas Tipo de Extensão, que possui 1 bit e INPUTS, que possui 4 bits e apenas uma saída, Sinal Final, que possui 8 bits, de forma que os bits da entrada INPUTS ocupam os quatro últimos bits do Sinal Final e o valor repetido da entrada Tipo de Extensão nos quatro primeiros bits do Sinal Final.

2.12.2. Testes do componente

O valor “XXXX” da entrada INPUTS significam que dependem do funcionamento do circuito em conexão com o extensor de sinal, desta forma os resultados do Extensor de Sinal serão “0000XXXX” se o valor de Tipo de Extensão for 0 e serão “1111XXXX” se o valor de Tipo de Extensão for 1.

2.13. MÁQUINA DE ESTADOS

Figura 24 - Máquina de Estados

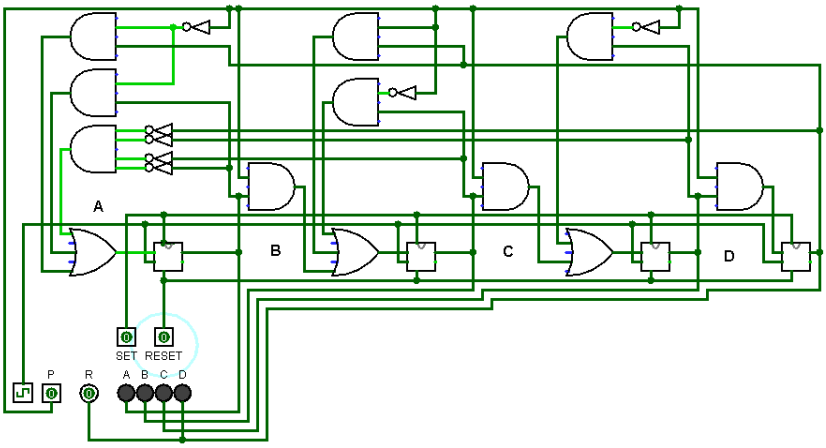
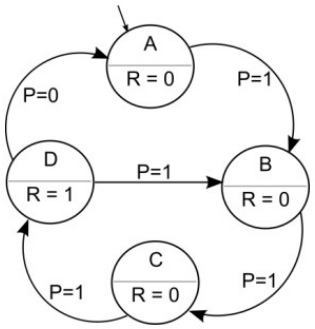


Figura 25 - Conceito Máquina de Estados



Uma Máquina de Estados é, resumidamente, uma máquina que devolve determinados valores dependendo do momento e se move de diferentes formas dependendo de uma determinada entrada. Esta Máquina de Estados não é nomeada, por isto lhe nomeei “Maquina de Estados”.

2.13.1. Descrição pinos e lógica

A Máquina de Estados, presente na Figura 24, é baseada no conceito de máquina de estados da Figura 25, desta forma pressupõe-se as verdades: “Caso não haja estado atual, A torna-se o estado atual”, “Se o estado atual for A e P = 1, o estado atual torna-se B”, “Se o estado atual for B e P = 1, o estado atual torna-se C”, “Se o estado atual for C e P = 1, o estado atual torna-se D”, “Se o estado atual for D e P = 1, o estado atual torna-se B”, “Se o estado atual for D e P = 0, o estado atual torna-se A”, “Se A, ou B, ou C forem verdadeiro e P = , o determinado valor verdadeiro mantém-se verdadeiro”, “Se o estado atual for diferente de D, então R=0” e, por último, “Se o estado atual for D, então R=1”.

2.13.2. Testes do componente

Figura 26 - Máquina de Estados (Caso 1)

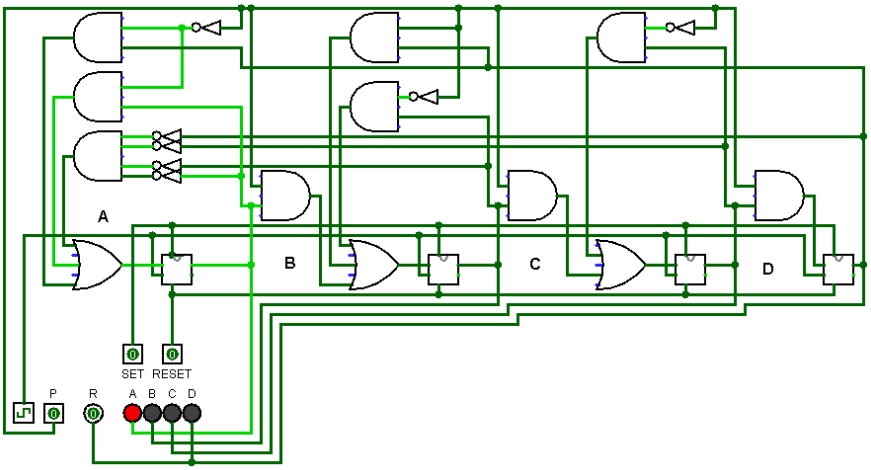


Figura 27 - Máquina de Estados (Caso 2)

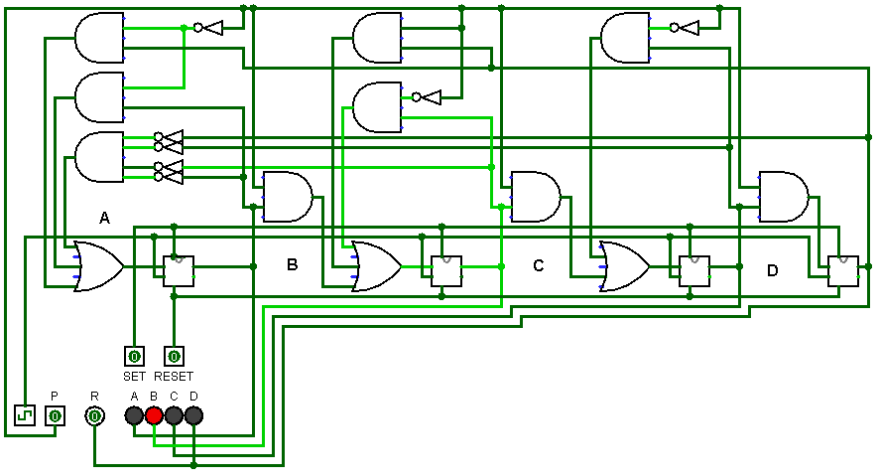


Figura 28 - Máquina de Estados (Caso 3)

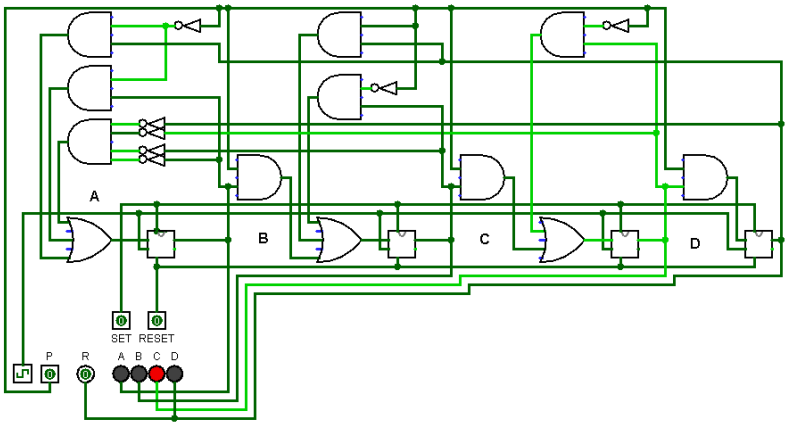


Figura 29 - Máquina de Estados (Caso 4)

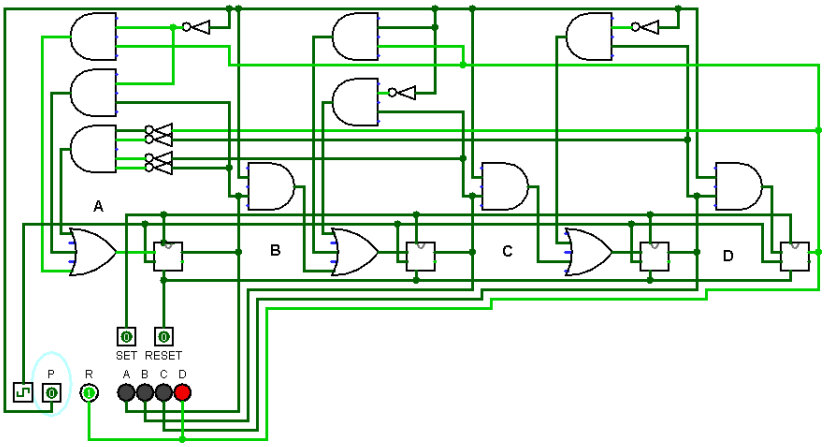


Figura 30 - Máquina de Estados (Caso 5)

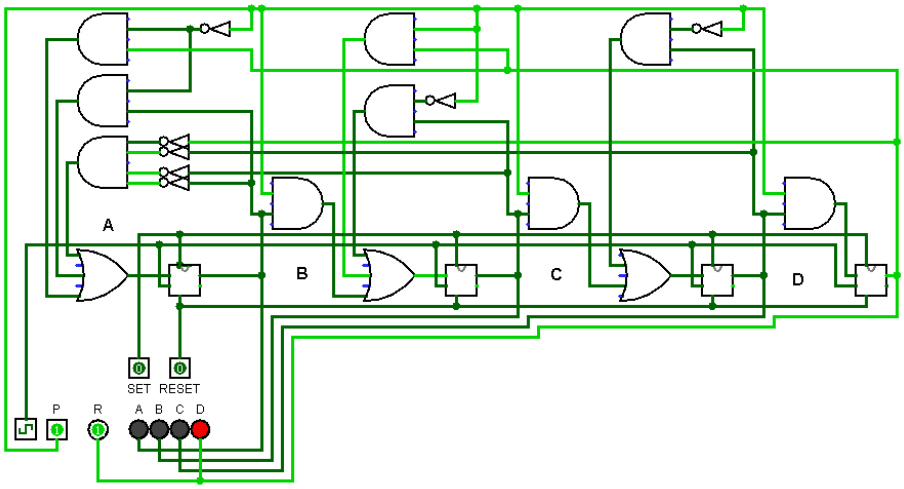


Tabela 11 - Tabela-Verdade Máquina de Estados

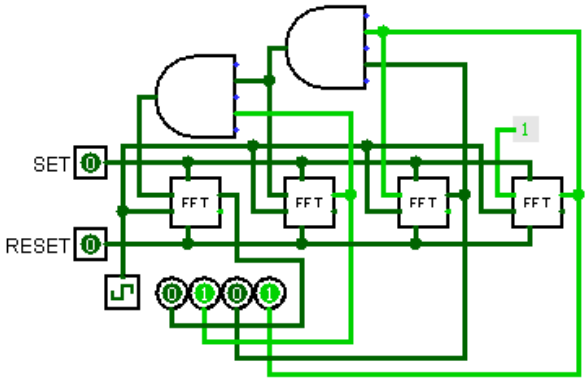
P	A	B	C	D	PRX E	R
0	0	0	0	0	A	0
1	0	0	0	0	A	0
0	1	0	0	0	A	0
1	1	0	0	0	B	0
0	0	1	0	0	B	0
1	0	1	0	0	C	0
0	0	0	1	0	C	0
1	0	0	1	0	D	0

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

Seguindo as regras impostas pelo Conceito de Máquina de Estado da Figura 25 para cada *Flip-Flop D* presente no sistema (representando os estados), há condições para que os mesmos sejam iguais a 1, sendo elas, para o *Flip-Flop D* do estado A: Todos os outros *Flip-Flop D* de estado serem iguais a 0, **ou** o *Flip-Flop D* de A ser igual a 1 **e** P for 0, **ou** o *Flip-Flop D* de D for 1 **e** P for 0. Para o *Flip-Flop D* do estado B: O *Flip-Flop D* de B ser igual a 1 **e** P for 0, **ou** o *Flip-Flop D* de A ser igual a 1 **e** P for 1, **ou** o *Flip-Flop D* de D for 1 **e** P for 1. Para o *Flip-Flop D* do estado C: O *Flip-Flop D* de C ser igual a 1 **e** P for 0, **ou** o *Flip-Flop D* de B ser igual a 1 **e** P for 1. Para o *Flip-Flop D* do estado D: O *Flip-Flop D* de C ser igual a 1 **e** P for 1. Por último, o valor de R é igual ao valor do *Flip-Flop D* do estado D. O funcionamento da Máquina de Estados pode ser descrita por meio da Tabela 10, onde a coluna PRX E equivale ao próximo estado a ser ativado.

2.14. CONTADOR SÍNCRONO

Figura 31 - Contador Síncrono



O Contador Síncrono é um componente que realiza uma contagem bit a bit de um valor com a quantidade de bits delimitada pela quantidade de *Flip-Flops T* presentes em seu circuito.

2.14.1. Descrição pinos e lógica

Figura 32 - *Flip-Flop T*

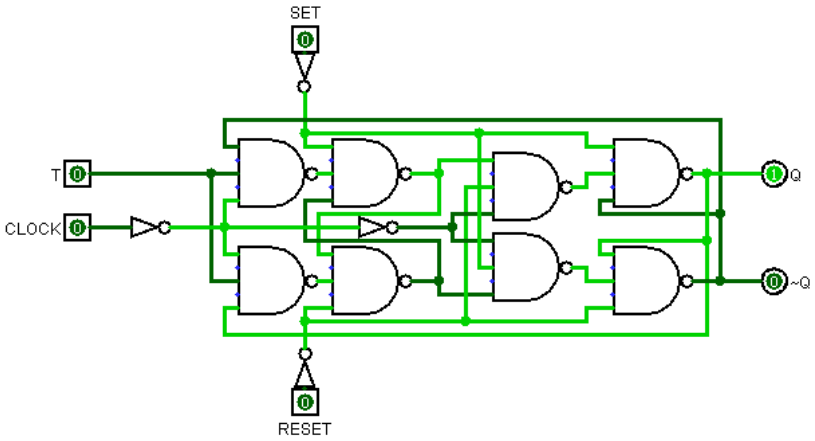


Tabela 12 - Tabela-Verdade *Flip-Flop T* Completa

T	Qant	Qs
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 13 - Tabela-Verdade *Flip-Flop T* Simplificada

T	Qs
0	Qant
1	~Qant

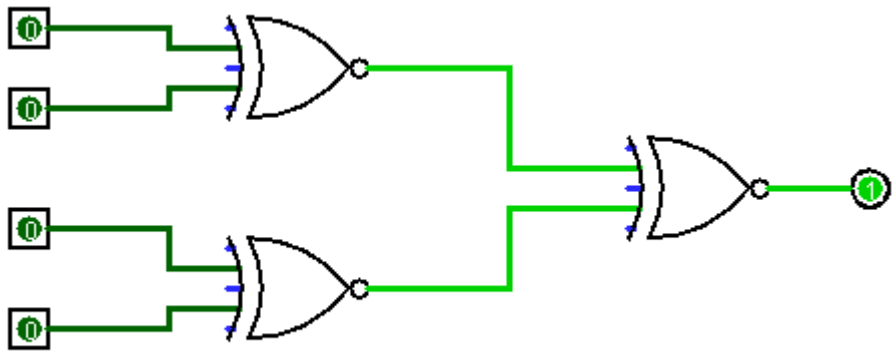
O contador Síncrono desenvolvido possui 4 bits e, consequentemente, quatro *Flip-Flops T*, presente na Figura 32, demonstrado na Figura 31, que, após ser desenvolvido e testado gerou a tabela-verdade Tabela 11 e, após simplificação, gerou a Tabela 12, desta forma o contador atribui o valor 1 a entrada do *Flip-Flop T* apenas quando todos os valores anteriores são iguais a 1, alternando-o, com exceção do último *Flip-Flop T, que sempre alterna o seu valor*.

2.14.2. Testes do componente

Os valores do Contador Síncrono são obtidos pela propriedade de alternação de valores anteriores do *Flip-Flop T*, de forma que é possível atribuir entradas iguais a 1 quando valores também sejam iguais a 1, de forma parecida com um Somador.

2.15 DETECTOR DE PARIDADE ÍMPAR

Figura 32 – Destector de paridade ímpar



Um detector de paridade ímpar é um circuito lógico usado para verificar a paridade (n úmero de bits "1") de uma sequência binária. A paridade ímpar significa que o número de bits "1" deve ser ímpar. Se o número de bits "1" na sequência de entrada for ímpar, o detector indi cará que a sequência tem paridade ímpar. Caso contrário, se o número de bits "1" for par, o de tector indicará que a sequência tem paridade par.

Função de Paridade

- **Paridade Ímpar:** A quantidade de bits "1" deve ser ímpar. Exemplo: "101" tem parida de ímpar porque possui 2 bits "1" (número par), mas o bit de paridade deve ser 1 par a torná-lo ímpar.
- **Paridade Par:** A quantidade de bits "1" deve ser par. Exemplo: "110" tem paridade p ar, pois tem 2 bits "1".

Como Funciona o Detector de Paridade Ímpar

O detector de paridade ímpar verifica se o número de bits "1" na sequência de entrada é ímpar ou não. Se o número de bits "1" for ímpar, o detector indicará um valor de "1", significando que a paridade é ímpar. Se o número de bits "1" for par, o detector indicará "0", significando que a paridade é par.

Para realizar isso, é necessário analisar todos os bits da sequência e determinar sua soma. O detector de paridade pode ser implementado usando uma operação de **XOR** (ou exclusivo), que é particularmente útil para calcular a paridade.

Aplicações do Detector de Paridade Ímpar

1. **Verificação de Erros em Sistemas Digitais:**
    - Em sistemas de comunicação e armazenamento de dados, um detector de paridade é usado para detectar erros simples de transmissão. Se o número de bits "1" em um pacote de dados for diferente da paridade esperada, isso indica que ocorreu um erro.
2. **Protocolos de Comunicação:**
    - O detector de paridade ímpar é usado em protocolos de comunicação, como o **paridade ímpar** em protocolos de comunicação serial, para verificar a integridade dos dados transmitidos.
3. **Armazenamento de Dados:**
    - Para verificar a integridade dos dados armazenados, os sistemas de memória podem usar bits de paridade para garantir que os dados não tenham sido corrompidos.

Vantagens e Desvantagens

Vantagens:

- **Simplicidade:** O detector de paridade ímpar é simples de implementar, geralmente usando apenas portas lógicas XOR.
- **Detecção de Erros:** Ele pode detectar erros de transmissão simples, como a mudança de um bit.

Desvantagens:

- **Não Detecta Todos os Erros:** O detector de paridade ímpar não pode detectar erros que envolvam um número par de bits corrompidos, pois a soma total de bits "1" pode continuar sendo ímpar mesmo com erros.

- **Não Corrige Erros:** Ele apenas detecta erros, mas não tem a capacidade de corrigir os erros.

Tabela 14 - Tabela verdade detector de paridade ímpar

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Espressão Detector de paridade Ímpar

```
~(~(a ^ b) ^ ~(c ^ d))
```

2.16 MAPAS DE KARNAUGH

O mapa de Karnaugh (ou K-map) é uma ferramenta gráfica utilizada para simplificar e xpressões booleanas. Ele organiza as variáveis e seus valores possíveis em uma tabela, permiti ndo a visualização rápida de padrões e a simplificação das expressões lógicas. É uma técnica que facilita a redução de funções booleanas, tornando os circuitos digitais mais eficientes em t ermos de número de portas lógicas.

Estrutura do Mapa de Karnaugh

- O K-map é uma tabela que organiza as entradas (variáveis) de uma função booleana em uma matriz.

- Cada célula do mapa representa uma combinação possível das variáveis, com a função booleana atribuída a esse valor de entrada.
- As células do mapa contêm os valores de saída (0 ou 1) da função booleana para as combinações de entradas correspondentes.

Passos para Construir um Mapa de Karnaugh

1. **Determinar o número de variáveis**:
    - O número de variáveis determina o tamanho do mapa. Por exemplo:
        - 2 variáveis → Mapa 2x2 (4 células)
        - 3 variáveis → Mapa 2x4 (8 células)
        - 4 variáveis → Mapa 4x4 (16 células)
        - 5 variáveis → Mapa 4x8 (32 células), e assim por diante.
2. **Preencher o mapa**:
    - Preencha as células do mapa com os valores de saída (0 ou 1) correspondentes à função booleana para cada combinação de variáveis.
3. **Agrupar 1s (ou 0s, dependendo da simplificação)**:
    - Para simplificar a expressão booleana, você agrupa os 1s em blocos de potências de 2 (1, 2, 4, 8, etc.). As células de um grupo devem estar em posições adjacentes.
    - Os grupos podem ser formados de forma horizontal, vertical ou, se necessário, circular, conectando as células que estão nas bordas opostas do mapa.
4. **Escrever a expressão simplificada**:
    - Para cada grupo de 1s, escreva a expressão booleana correspondente. O termo de cada grupo será composto pelas variáveis que não mudam no grupo (as variáveis constantes).

Exemplo de Mapa de Karnaugh (para 3 variáveis)

Considere a função booleana f(A,B,C)=ABC+ABC+ABC.

1. **Passo 1: Determinar o Mapa de Karnaugh para 3 variáveis (A, B, C)**:

    O Mapa de Karnaugh para 3 variáveis possui 8 células. As variáveis B e C são as variáveis de controle, e A será a variável mais significativa. O mapa fica assim:

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

2. **Passo 2: Preencher o mapa com os valores de saída**:

    Para a função booleana dada, as células são preenchidas com valores de saída baseados nas combinações das variáveis A, B e C.

3. **Passo 3: Agrupar os 1s**:

   Agora, vamos agrupar os 1s em blocos de potências de 2.

   - O grupo de 4 células que contém 1s pode ser formado entre as células (00, 1 1), (01, 10), (11, 01), (11, 10).
4. **Passo 4: Escrever a expressão simplificada**:

   Para o grupo de 4 células, observe que a variável C está sempre 1, e A e B são amb os 1. Assim, o grupo é simplificado para B·C.

Exemplo de Mapa de Karnaugh (para 4 variáveis)

Para uma função booleana de 4 variáveis, como f(A,B,C,D), você teria um mapa 4x4 com 16 células.

Tabela de Mapa de Karnaugh para 4 variáveis (A, B, C, D):

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

Vantagens dos Mapas de Karnaugh

1. **Simplicidade**:
   - A simplificação de expressões booleanas usando K-maps é intuitiva e visual, ao contrário de métodos algébricos mais complicados.
2. **Redução de Gates**:
   - Ao simplificar as expressões booleanas, você pode reduzir o número de porta s lógicas necessárias para implementar uma função, o que resulta em circuito s mais eficientes e econômicos.
3. **Visualização Rápida**:
   - O K-map permite que você veja rapidamente padrões e relações entre as vari áveis que podem ser agrupados para simplificação.

Desvantagens dos Mapas de Karnaugh

1. **Escalabilidade**:
   - Para funções com muitas variáveis (geralmente mais de 5 ou 6), o mapa pode se tornar difícil de manipular e visualizar devido ao grande número de células.
2. **Limitação em Funções Complexas**:

- Mapas de Karnaugh funcionam bem para simplificar funções com até 6 variáv eis. Para funções com mais de 6 variáveis, o número de células cresce muito, tornando o método imprático.

Figura 32 - Mapas Karnough

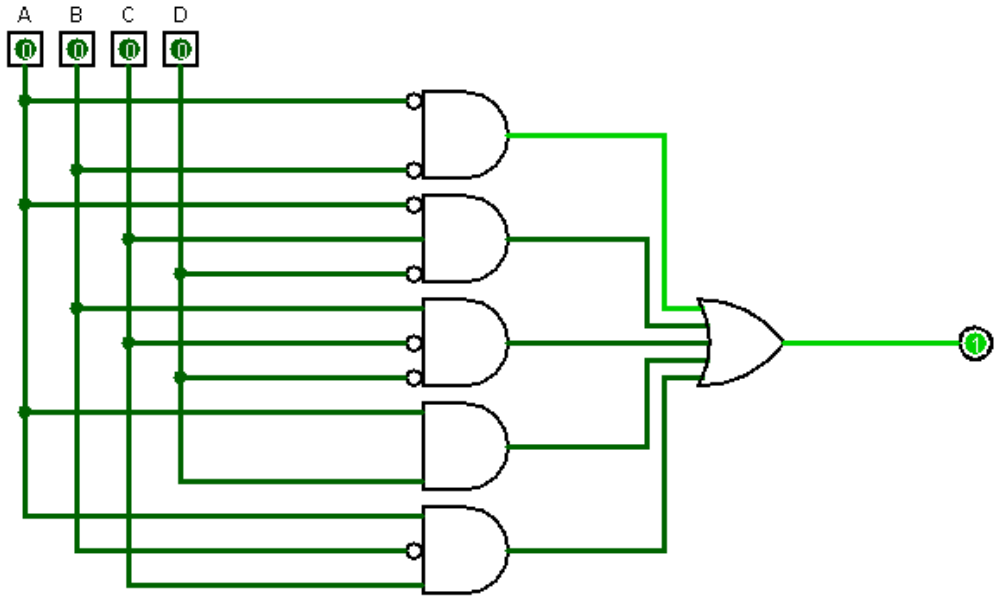


Tabela 14 – Tabela verdade Karnaugh

A	B	C	D	x
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

2.17 DECODIFICADOR DE 7 SEGMENTOS

Decodificador de 7 Segmentos

O **decodificador de 7 segmentos** é um dispositivo utilizado em sistemas digitais para exibir números e, em alguns casos, letras, utilizando um display composto por 7 LEDs organizados em forma de segmentos. Esse decodificador converte um valor binário de entrada em sinais para acionar os segmentos do display e mostrar o número ou caractere correspondente.

Estrutura de um Display de 7 Segmentos

Um display de 7 segmentos é composto por 7 LEDs dispostos em forma de um número "8", e cada LED corresponde a um segmento identificado com as letras de **a** a **g**.

- **a, b, c, d, e, f, g são os 7 segmentos do display.**
- Cada segmento pode ser ligado ou desligado para formar números ou caracteres.

Funcionamento

- Para cada número binário de entrada, o decodificador ativa os segmentos correspondentes para formar o número ou letra no display de 7 segmentos.
- A entrada do decodificador é normalmente um código binário (por exemplo, 4 bits), e o decodificador converte essa entrada para os sinais que controlam quais segmentos serão acesos no display.

Entradas e Saídas

- **Entradas**: O decodificador recebe um número binário de entrada, que geralmente é de 4 bits. Cada combinação de bits (de 0000 a 1001 para os números de 0 a 9) ativa uma sequência de segmentos.
- **Saídas**: O decodificador fornece 7 saídas (uma para cada segmento do display). Essas saídas ativam ou desativam os segmentos correspondentes para formar os números.

Exemplo de Decodificação (para o número 3)

Para exibir o número "3", a entrada para o decodificador seria o número binário **0011** (que corresponde ao número 3 no sistema decimal).

- O decodificador de 7 segmentos acionaria os segmentos **a, b, c, d, g**, resultando no número "3" exibido no display.

Decodificadores Comuns

1. **CD4511**: Um decodificador BCD para 7 segmentos. Ele recebe uma entrada BCD de 4 bits e fornece sinais de controle para os 7 segmentos do display.
2. **74LS47**: Outro exemplo de decodificador BCD para 7 segmentos. Ele converte entra das binárias codificadas em decimal (BCD) para controlar um display de 7 segmentos.

Funcionamento do Circuito

Um **decodificador BCD para 7 segmentos** pode ser implementado usando uma tabela de correspondência ou lógica combinatória, onde você utiliza portas lógicas para implementar a ativação correta de cada segmento para os valores binários de entrada.

Aplicações

- **Displays Digitais**: Usados em relógios digitais, calculadoras, medidores, termômetro s digitais e outros dispositivos que exibem números.
- **Contadores e Indicadores**: Utilizado em sistemas que necessitam exibir contagens ou valores numéricos de forma clara.
- **Sistemas de Comunicação**: Para mostrar status ou valores de forma legível em inte rfaces digitais.

Figura 33 – *Detector de 7 segmentos*

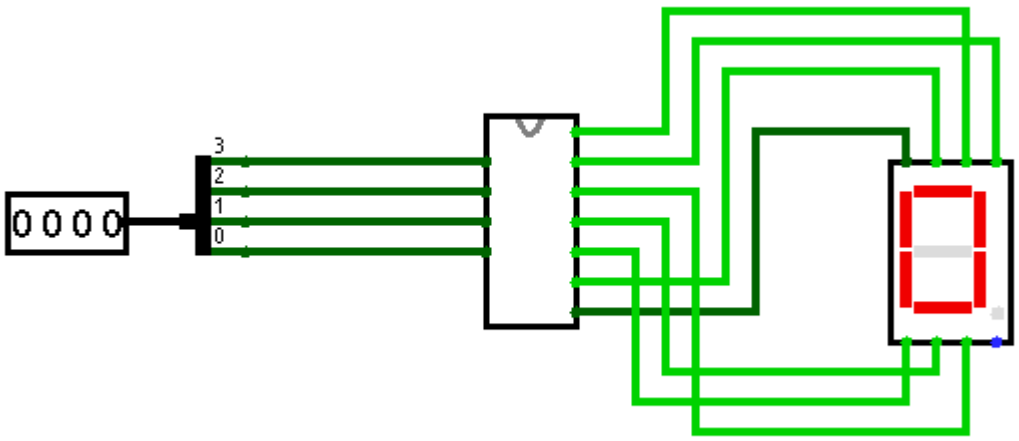
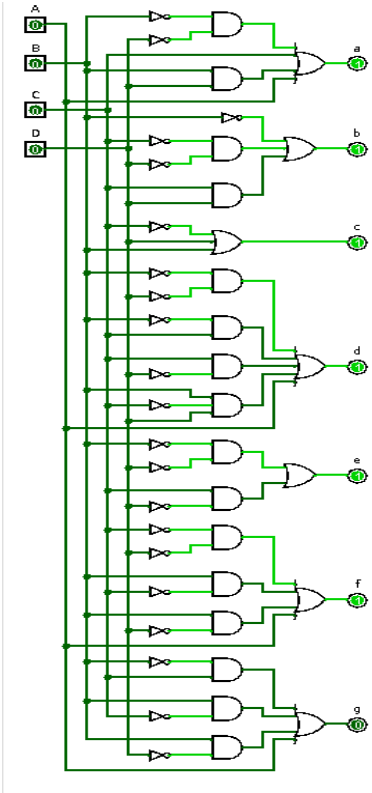


Tabela 15 – Tabela verdade detector 7 segmento

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	0	1	1
1	1	0	1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1

Logo após obtermos a tabela verdade e suas expressões simplificadas, cria-se o circuit o combinacional com portas lógicas AND, OR a NOT, porém, sem que a saída esteja ligada a algo. Logo, acrescentamos o display de 7 segmentos que cada entrada é ligada a saída que do circuito combinacional que representa uma parte do display.

Figura 33 - circuito combinacional com portas lógicas AND, OR a NOT



2.18 DETECTOR DE NÚMERO PRIMO

Um **detector de número primo** é um sistema ou circuito lógico projetado para verific ar se um número fornecido é um número primo ou não. Um número primo é um númer o natural maior que 1 que tem exatamente dois divisores positivos: 1 e ele mesmo. Por exemplo, 2, 3, 5, 7 e 11 são números primos, enquanto 4, 6, 8, 9 e 10 não são.

Como Funciona um Detector de Número Primo?

O funcionamento básico de um detector de número primo envolve testar se o número dado t em divisores diferentes de 1 e ele mesmo. Se o número tiver mais de dois divisores, ele não é primo.

Passos para Detectar se um Número é Primo:

1. **Entrada do Número**:
    - O número a ser verificado é dado como entrada. Pode ser um número inteiro de N bits.
2. **Divisão por Números Menores**:
    - O detector começa dividindo o número de entrada (digamos N) por todos os n úmeros menores que N, geralmente de 2 até N. Isso ocorre porque um númer o não pode ter divisores maiores que sua raiz quadrada (exceto ele mesmo).
3. **Verificação de Resto**:
    - Para cada divisão, verifica-se se o resto da divisão é 0. Se o número for divisí vel por algum número dentro desse intervalo, o número não é primo e o detec tor retorna "não primo".
4. **Resultado**:
    - Se não encontrar nenhum divisor entre 2 e N, então o número é primo, e o det ector retorna "primo".

Algoritmo Básico

Aqui está um pseudocódigo básico para um detector de número primoem python:

```
função é_primo(n):
    se n <= 1:
        retorna FALSO
    para i de 2 até sqrt(n):
        se n % i == 0:
            retorna FALSO
    retorna VERDADEIRO
```

Implementação com Lógica Combinatória (Circuitos)

Em um nível de hardware, a detecção de números primos pode ser implementada por meio de lógica combinatória ou uma máquina de estados finitos (FSM) que executa a verificação de divisibilidade. O processo seria:

1. **Entrada Binária**:
    - O número N é representado em formato binário e dado como entrada para o s istema.
2. **Circuitos de Comparação**:
    - Circuitos de comparação verificam se o número N é divisível por qualquer nú mero menor que N. Isso pode ser feito por circuitos que realizam divisões e v erificam os restos.
3. **Decisão**:
    - Se o número N for divisível por qualquer número, a saída do circuito indicará que o número não é primo.
    - Se não for divisível por nenhum número, a saída do circuito indicará que o nú mero é primo.

Exemplo de Detecção de Número Primo:

Entrada: 11

1. **Divisores possíveis**: Verificar divisores de 2 até 11, ou seja, 2, 3.
2. **Divisão**:
    - 11 dividido por 2: o resto não é 0.
    - 11 dividido por 3: o resto não é 0.
3. Como não há divisores que resultam em um resto de 0, o número 11 é primo.

Entrada: 10

1. **Divisores possíveis**: Verificar divisores de 2 até 10, ou seja, 2, 3.
2. **Divisão**:
    - 10 dividido por 2: o resto é 0.
3. Como 10 é divisível por 2, o número 10 não é primo.

Algoritmos de Detecção de Primos Mais Eficientes

Embora o método acima funcione, há algoritmos mais eficientes para a detecção de número s primos, especialmente para números grandes, como:

1. **Algoritmo de Crivo de Eratóstenes**:
    - Um algoritmo eficiente para encontrar todos os números primos até um deter minado limite. Ele elimina múltiplos de cada número primo, deixando apenas números primos na lista.
2. **Teste de Primalidade de Miller-Rabin**:

- Um teste probabilístico de primalidade que é eficiente para números grandes, usado frequentemente em criptografia.

3. **Algoritmo de Primalidade AKS**:
    - Um algoritmo determinístico que verifica se um número é primo em tempo poli nomial.

Exemplo de Circuito Digital para Detector de Número Primo

Para circuitos digitais, podemos usar uma série de portas lógicas, como portas AND, OR, N OT e XOR, para implementar a lógica de verificação de divisibilidade. O circuito pode:

1. Realizar divisões por diferentes números até N.
2. Comparar o resto das divisões.
3. Usar um comparador para verificar se o resto é 0.
4. Se encontrar um divisor, acionar um sinal para indicar que o número não é primo.

Vantagens do Detector de Número Primo

- **Automatização**: Um detector de número primo automatiza o processo de verificação, o que é útil em sistemas embarcados ou de criptografia.
- **Eficiência**: Detectores eficientes podem lidar com números grandes de maneira rápi da, especialmente quando combinados com algoritmos avançados.

Figura 34 - Detector de número primo

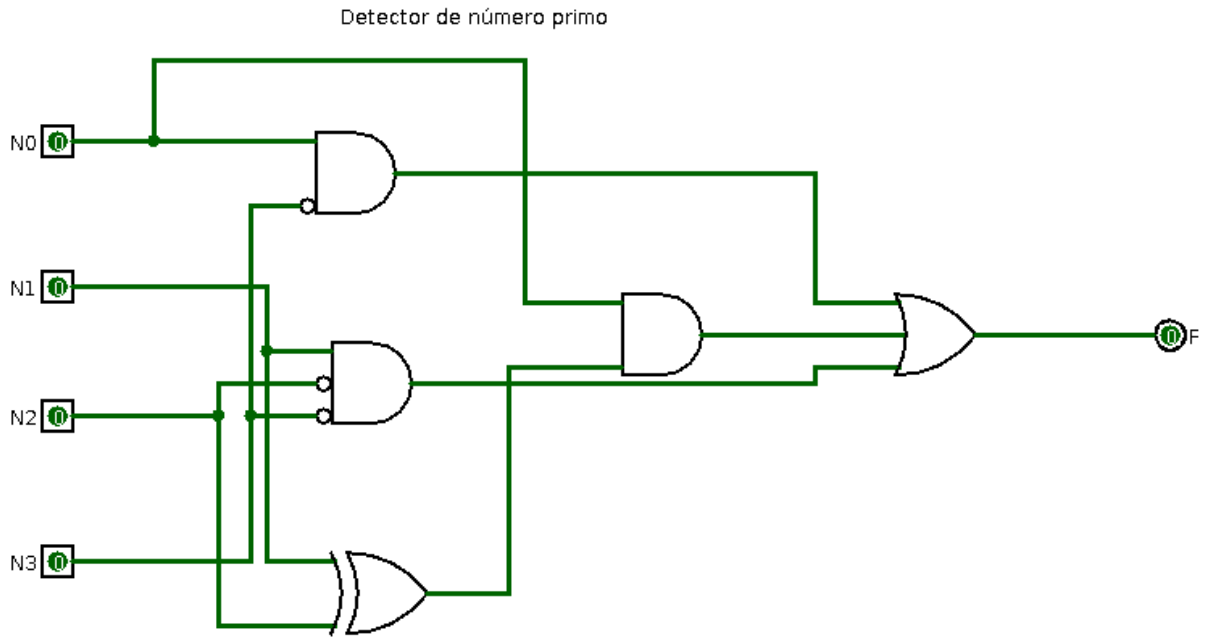


Tabela Verdade detector de número primo

N3	N2	N1	N0	Output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0

3. CONSIDERAÇÕES FINAIS

O trabalho foi uma reaproveitação da atividade de barramento do semestre passado de AOC pois não consegui nota para aprovação da matéria mas pude tá reaproveitando o trabalho, este semestre foi quase o mesmo de antes com alguns diferencias como detector de sequência binária, detector de número primo e decodificador de 7 segmentos. Agradeço ao professor Herbert pela oportunidade e espero que neste semestre eu consigo a aprovação da matéria.

4. REFERÊNCIAS

Canal Tell Moitas, https://www.youtube.com/@tellmoitas9661

Site Filipe Flop, https://www.filipeflop.com/blog/entendendo-o-flip-flops/

Site Metrópole Digital, https://materialpublic.imd.ufrn.br/

Biblioteca UFRR, https://ufrr.br/bibliotecas/