UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

CURSO DE BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

LABORATÓRIO DE CIRCUITO – CODIFICAÇÃO E SIMULAÇÕES

Boa Vista - RR
2024

ACADÊMICOS:

ABRAHÃO PICANÇO NERES DE OLIVEIRA

LUCAS GABRIEL ROCHA CONSTANCIO

YAN DOS SANTOS TEIXEIRA

LABORATÓRIO DE CIRCUITO – CODIFICAÇÃO E SIMULAÇÕES

Trabalho para a disciplina de Arquitetura e Organização de Computadores do curso Ciências da Computação da Universidade de Roraima - UFRR.

Orientador: Prof.Dr. Hebert Oliveira Rocha

Boa Vista/RR
2024

LABORATÓRIO DE CIRCUITO – CODIFICAÇÃO E SIMULAÇÕES

Trabalho apresentado à disciplina de Arquitetura e Organização de Computadores, do Curso de Ciência da Computação, da UFRR (Universidade Federal de Roraima), como requisito parcial para obtenção de nota na disciplina.

Orientador(a): Prof. Dr. Hebert Oliveira Rocha

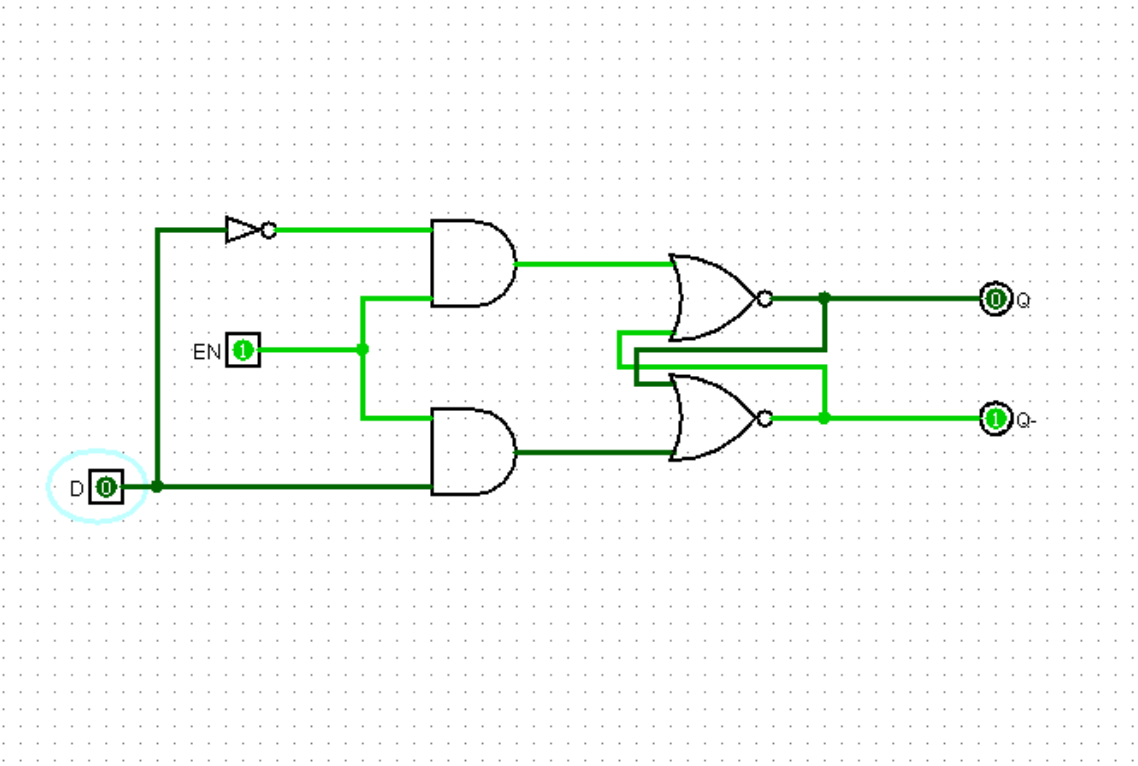
Boa Vista - RR
2024

RESUMO

Este trabalho apresenta o desenvolvimento de diversos circuitos lógicos no simulador Logisim, seguindo princípios de Arquitetura e Organização de Computadores (AOC). Foram implementados componentes lógicos utilizando portas básicas e compostas, empregando hierarquia de encapsulamento para facilitar a organização e a visualização. Cada componente é detalhado em sua funcionalidade, entradas, saídas e lógica interna. Além disso, são apresentadas as estratégias de teste adotadas (tabelas-verdade, testes unitários e resultados práticos no simulador), bem como a análise desses resultados.

Palavras-chave: Lógica Digital; Circuitos Lógicos; Logisim; Arquitetura e Organização de Computadores.

COMPONENTE 01 – Registrador Flip-Flop do tipo D e do tipo JK:
Tipo D:



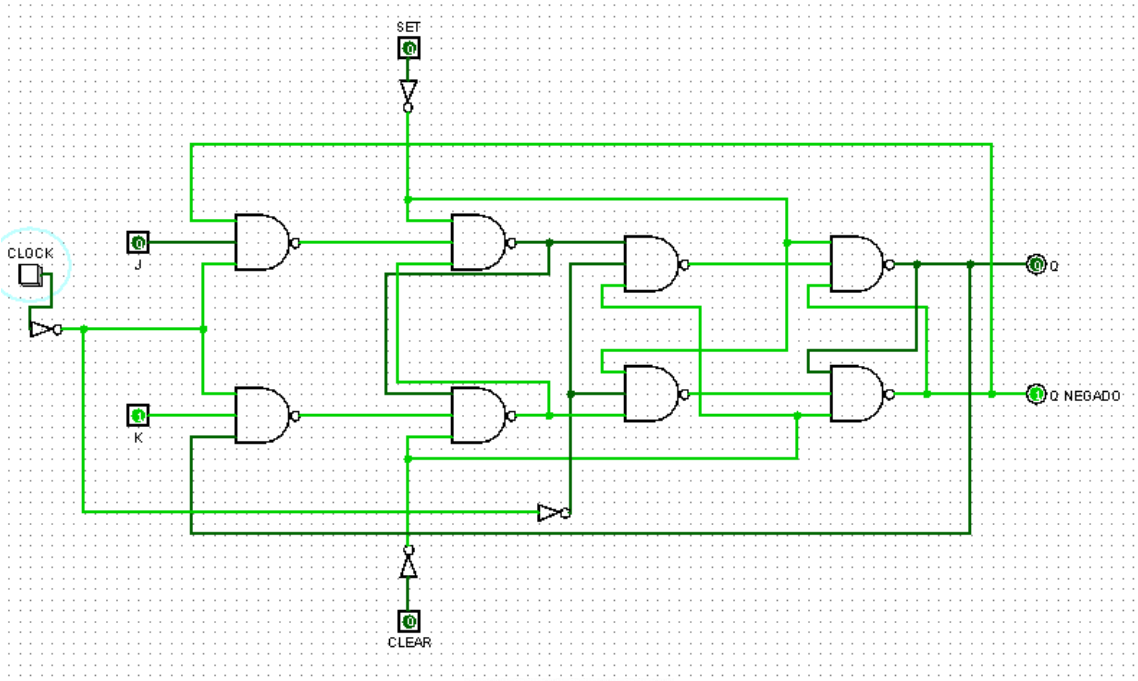
D	Q	Q*
0	0	**0**
0	1	**0**
1	0	**1**
1	1	**1**

O primeiro circuito que fizemos foi o Registrador Flip-Flop do tipo D simples de 1 bit de entrada e 1 bit de saída. Nele temos uma entrada de 1bit que vai para duas portas AND, essas mesmas portas AND recebem outra entrada que é o ENABLE. As duas portas AND vão para duas portas NOR e saem em duas saídas, onde a saída Q armazena o valor do bit de entrada D sempre que o ENABLE está em nível alto (1).

Quando ENABLE é desativado (0), o latch “congela” o valor previamente armazenado em Q, mantendo-o até que o ENABLE seja novamente acionado e o valor de D possa ser atualizado. A utilização de portas NOR interligadas para formar o latch SR interno é padrão, e a lógica de entrada com AND e NOT garante que D seja devidamente convertido em sinais set/reset adequados para atualizar Q.

Esse tipo de circuito é fundamental para memorização de um bit em dispositivos lógicos, sendo a base de registradores, contadores e diversas estruturas de memória mais complexas.

Tipo JK:



SET	RESET	J	K	Qa	$\overline{Qa}$
0	1	X	X	0	1
1	0	X	X	1	0
0	0	0	0	Qa	$\overline{Qa}$
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	$\overline{Qa}$	Qa

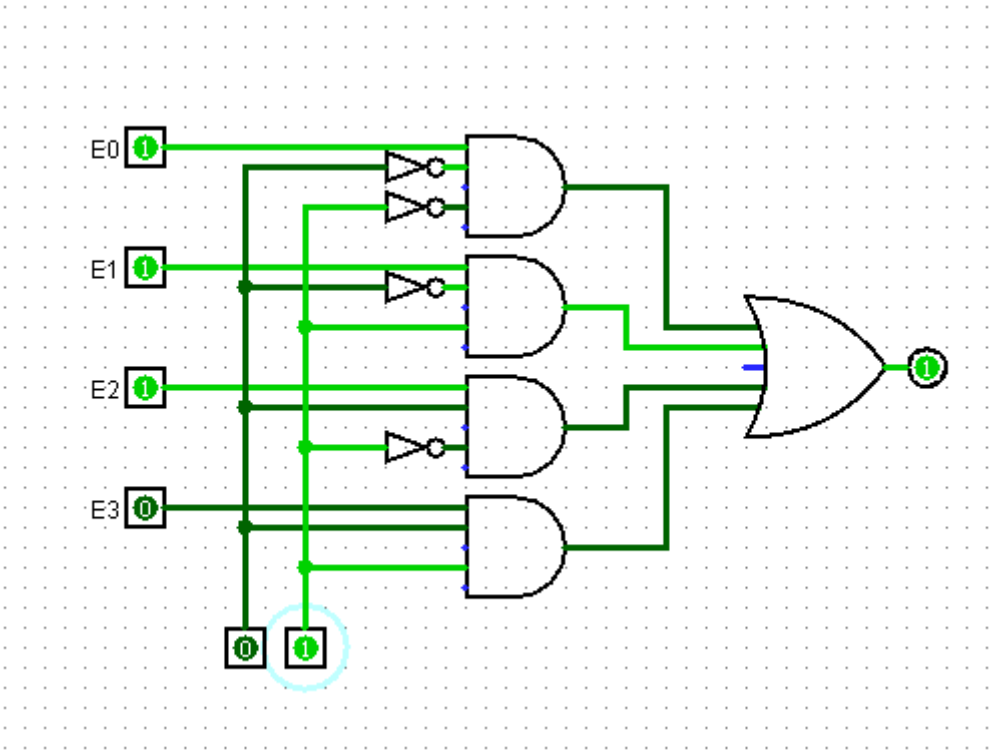
Tabela Verdade do FF JK mestre escravo com as entradas de preset (set) e reset (clear)

Flip-Flop JK fizemos utilizando NANDs e um sinal de clock, que permite implementar todas as funções do Flip-Flop D e SR, com a vantagem adicional do toggle (quando J=K=1). Temos também as entradas SET e CLEAR, que são entradas assíncronas, de nível baixo. Elas permitem forçar o estado do Flip-Flop independentemente do Clock.

J e K: Entradas de dados do flip-flop. Diferentemente do tipo D, cujo valor de entrada define diretamente a saída, o JK oferece mais flexibilidade:

- Quando J=1 e K=0, o flip-flop é configurado (Set) na próxima borda de clock, resultando em Q=1.
- Quando J=0 e K=1, o flip-flop é resetado (Clear) na próxima borda de clock, resultando em Q=0.
- Quando J=1 e K=1, o flip-flop alterna (inverte) o valor de Q a cada pulso de clock (toggle).
- Quando J=0 e K=0, a saída Q mantém o valor anterior (hold).

COMPONENTE 02 – Multiplexador de quatro opções de entrada:

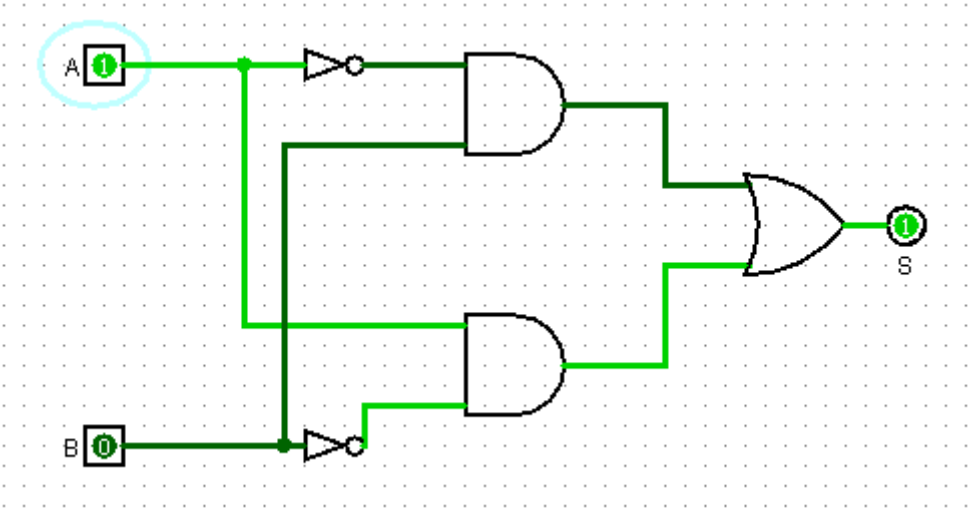


Aqui temos o multiplexador que recebe 4 entradas de 1bit, 2bits de seleção que nele podemos escolher qual entrada será armazenada na saída. Seu objetivo é encaminhar uma dentre quatro entradas (E0, E1, E2, E3) para a saída, com base no valor das duas linhas de seleção (S1, S0).

Seletor	Entrada
00	E0 00
01	E1 01
10	E2 10
11	E3 11

E0	E1	E2	E3	S0	S1	S
1	0	0	0	0	0	1
0	1	0	0	1	0	1
0	0	1	0	0	1	1
0	0	0	1	1	1	1

COMPONENTE 03 – Porta lógica XOR usando os componentes: AND, NOT e OR:

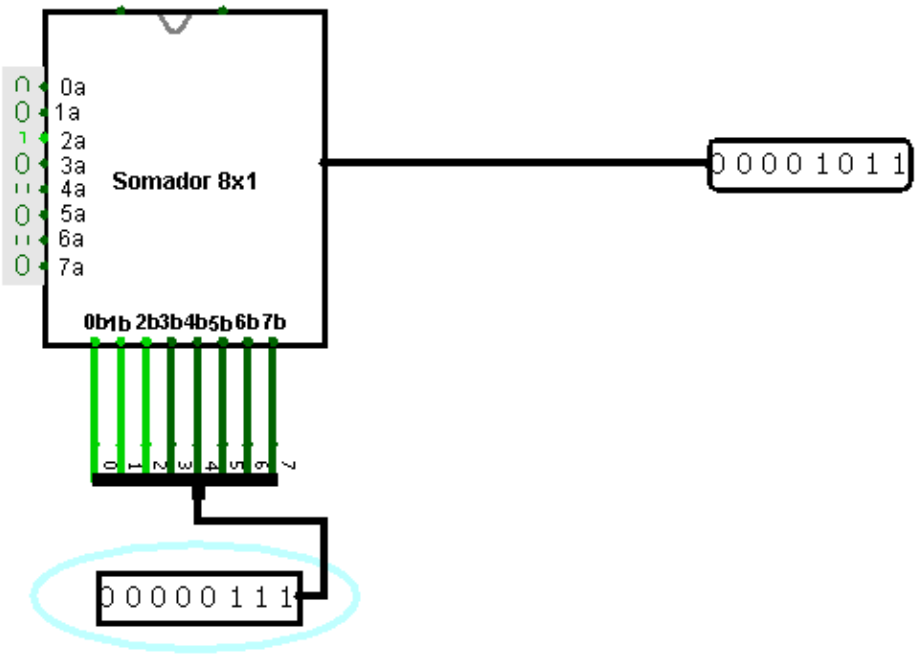


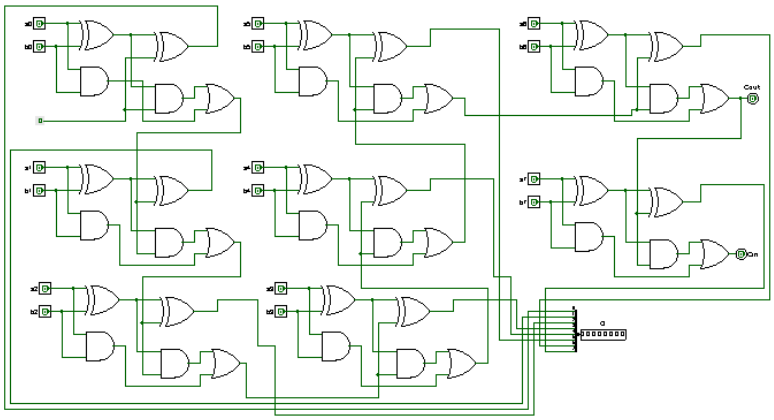
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

A porta lógica XOR acima foi feita utilizando apenas portas AND, NOT e OR e produz saída 1 somente quando as entradas forem diferentes: se A≠B, então a saída é 1; se A=B, a saída é 0. Uma porta NOT recebe A, gerando ¬A (A negado), e outra porta NOT recebe B, gerando ¬B (B negado).

Agora temos quatro sinais: A, B, ¬A e ¬B, isso significa que a saída será 1 se "A for 1 e B for 0" ou se "A for 0 e B for 1".

COMPONENTE 04 – Somador de 8 bits que recebe um valor inteiro e soma com o valor 4:





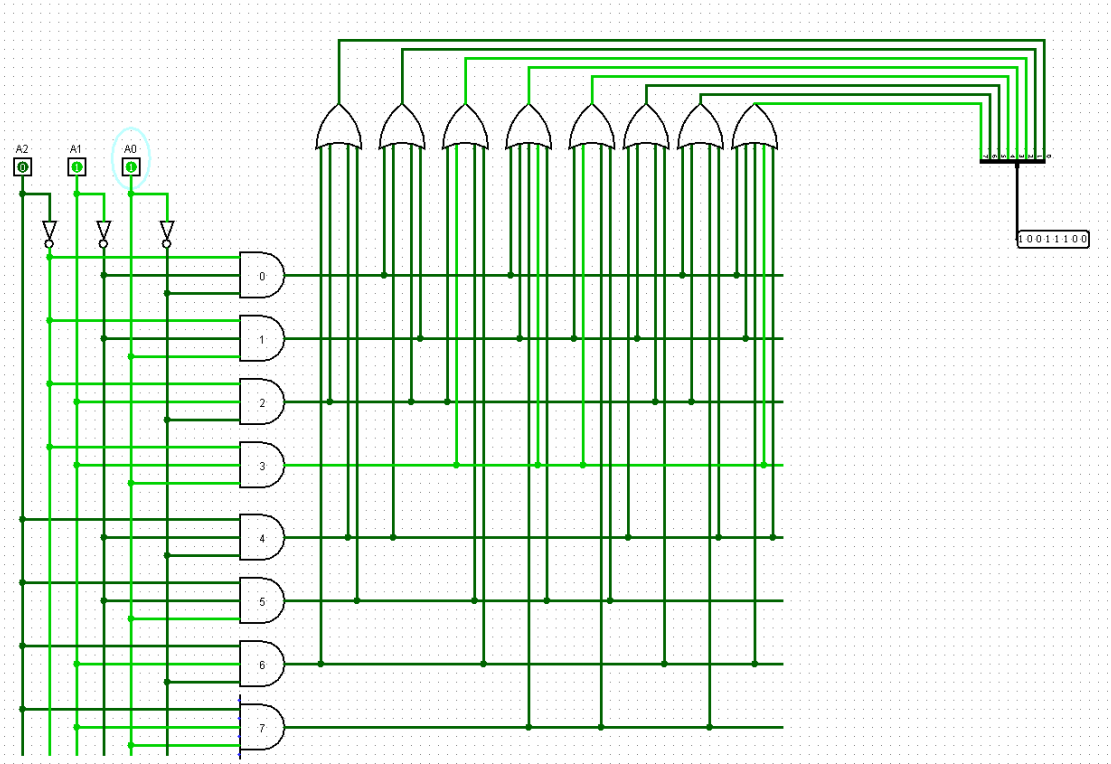
Circuito somador de 8bits.

O circuito que temos acima é um **Somador de 8 bits com valor constante** (no caso, soma-se o valor inteiro 4 a um número de 8 bits fornecido na entrada).

Neste somador temos a entrada A e B, onde: **B (8 bits)**: É a entrada principal, um valor de 8 bits que será acrescido de 4. **A (8 bits)**: Neste caso, não é uma entrada variável. A é configurada internamente com o valor constante 4. Isso significa que, em vez de esperar que A venha de fora, o circuito insere o valor binário “00000100” nas linhas de A.

Este somador de 8 bits é implementado a partir de somadores de 1 bit em série (full adders). Neste caso, o somador é um componente pré-montado (encapsulado) que recebe duas entradas de 8 bits e produz a soma também em 8 bits, além de um sinal de carry-out. O somador pega cada bit de A e B, soma-os junto ao carry que possa surgir do bit menos significativo, resultando na soma final S.

COMPONENTE 05 – Memória ROM de 8 bits:



A memória ROM (Read-Only Memory) é um componente fundamental em sistemas digitais, a que implementamos acima se trata de uma ROM **simples com 3 linhas de endereço (A0, A1, A2) e uma saída de 8 bits**, implementada inteiramente por portas lógicas. Ela armazena 8 palavras (células) de 1 bit cada, já que 3 bits de endereço permitem acessar $2^3 = 8$ endereços distintos. A lógica interna é um arranjo de decodificação de endereços seguida de uma lógica combinacional que determina o valor armazenado em cada posição.

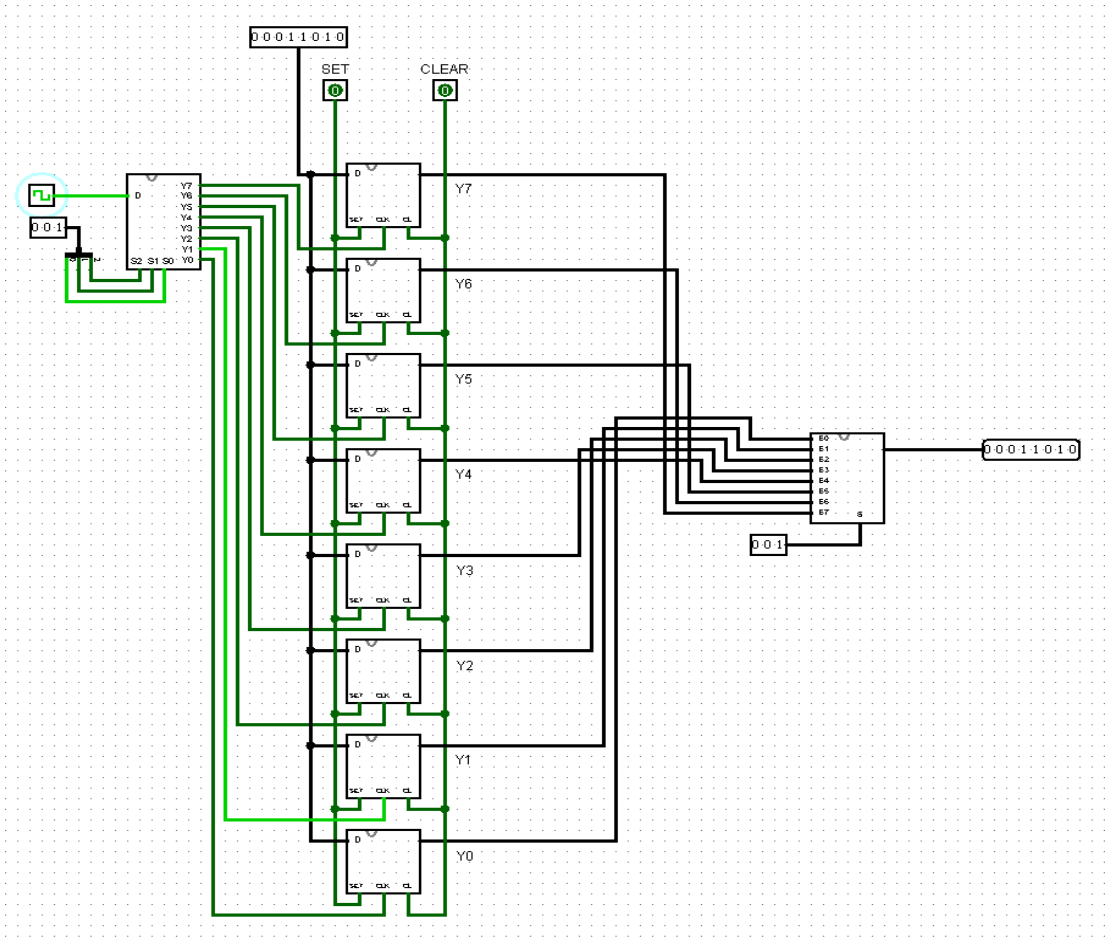
De entrada temos A0, A1, A2: 3 bits de endereço. Cada combinação binária de (A2, A1, A0) seleciona uma das 8 palavras armazenadas na ROM. Já a saída é representada por uma saída de 8 bit que é o valor lido da memória na posição endereçada pelas entradas A0, A1, A2.

Através de 3 portas NOT (para gerar ¬A0, ¬A1, ¬A2) e 8 portas AND, cria-se um decodificador 3-para-8:

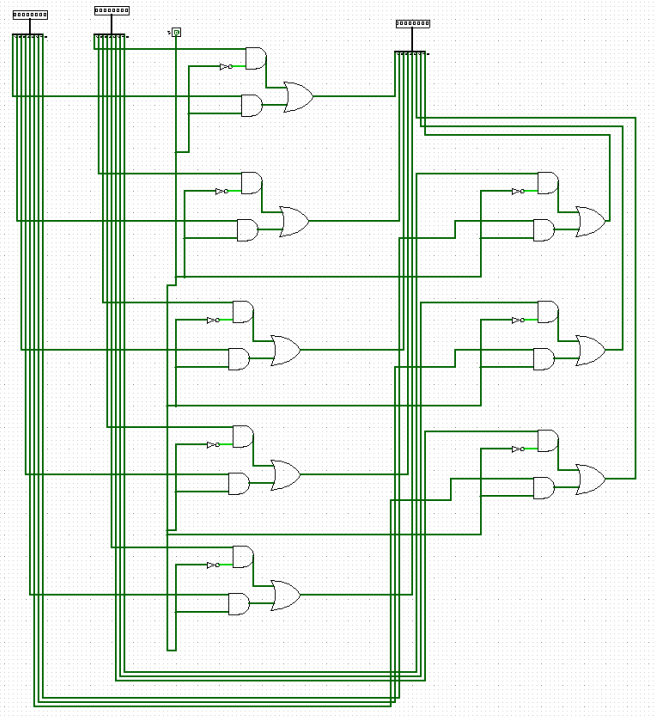
- Linha 0 ativa quando A2=0, A1=0, A0=0
- Linha 1 ativa quando A2=0, A1=0, A0=1
- Linha 2 ativa quando A2=0, A1=1, A0=0
- Linha 3 ativa quando A2=0, A1=1, A0=1
- Linha 4 ativa quando A2=1, A1=0, A0=0
- Linha 5 ativa quando A2=1, A1=0, A0=1
- Linha 6 ativa quando A2=1, A1=1, A0=0
- Linha 7 ativa quando A2=1, A1=1, A0=1

Cada um dos 8 endereços da ROM tem armazenado internamente um valor de 8 bits. Esses 8 bits não são gerados dinamicamente; eles são fixos e definidos no projeto da ROM. Em uma implementação por portas lógicas, a definição do conteúdo da ROM é feita conectando (ou não) cada linha do decodificador às portas OR que formarão cada bit de saída. Ou seja, para cada bit de saída (D0 a D7), você irá combinar as linhas de endereço correspondentes aos endereços que têm um '1' neste bit.

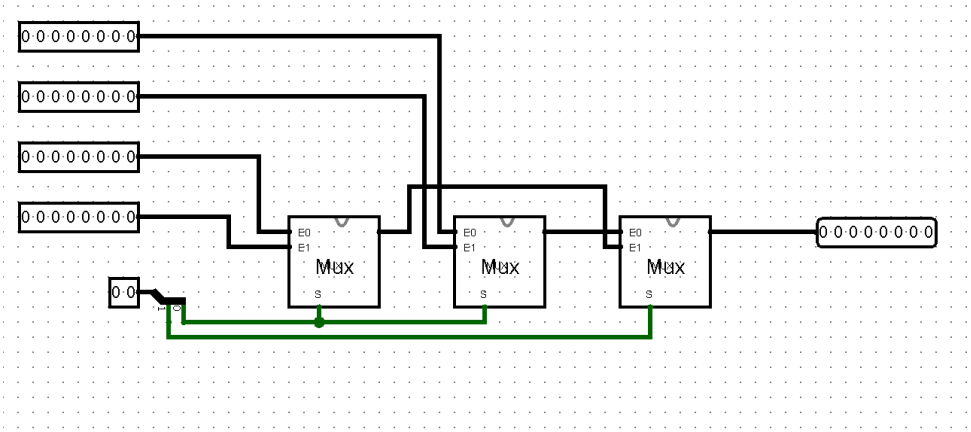
COMPONENTE 06 – Memória RAM de 8 bits:



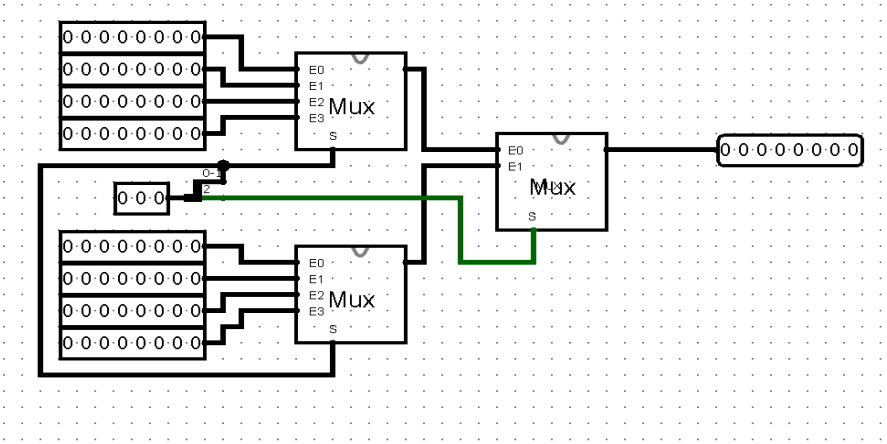
[componentes]



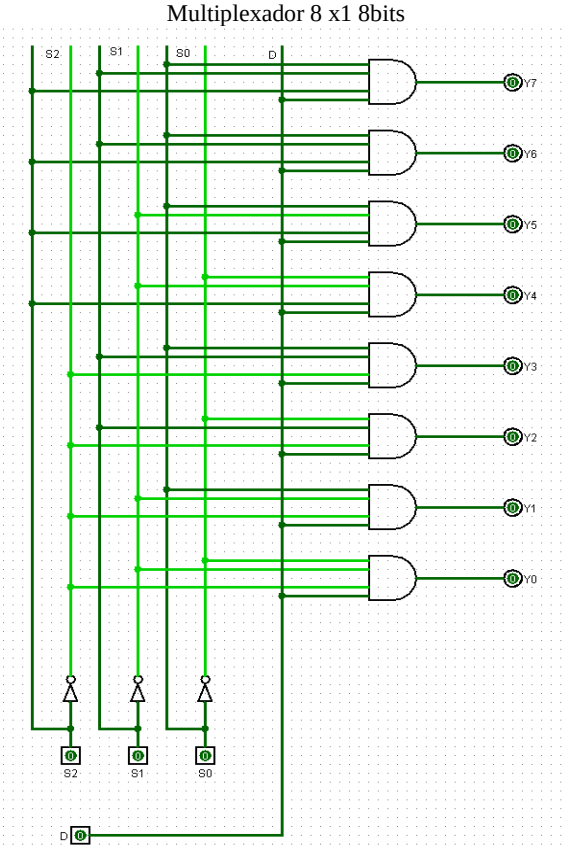
Multiplexador 8 bits.



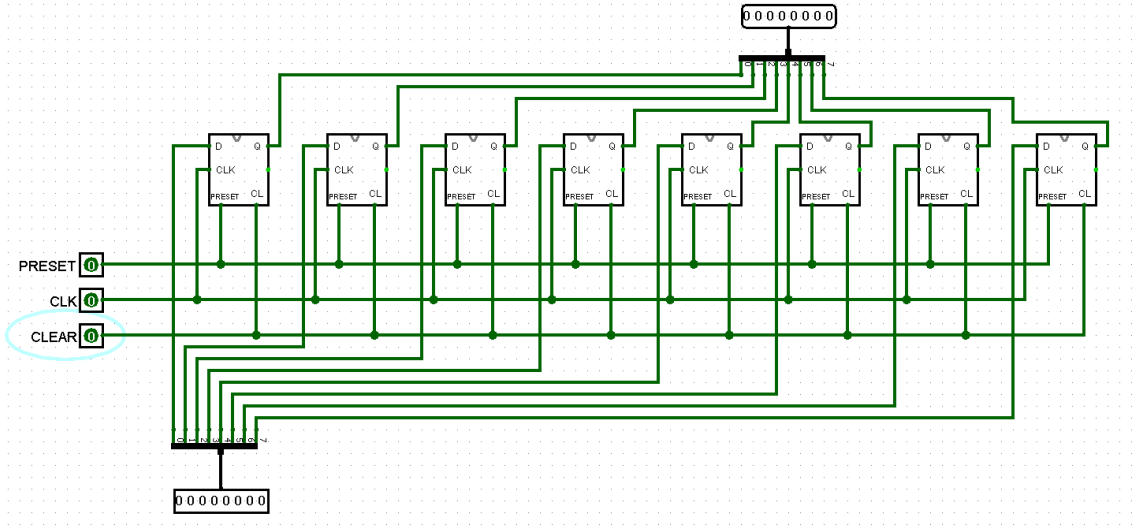
Multiplexador 4x1 8bits



Multiplexador 8 x1 8bits



1 x 8 Demultiplexer



Registrador 8 bits

A memória RAM de 8 bits com 8 endereços foi montada a partir da construção de vários outros componentes digitais encapsulados, tais como demultiplexador, oito registradores 1x8 e um multiplexador 8x1 de 8 bits. Esse circuito forma uma memória RAM endereçável de 8 palavras de 8 bits cada, capaz de armazenar e recuperar dados de forma síncrona.

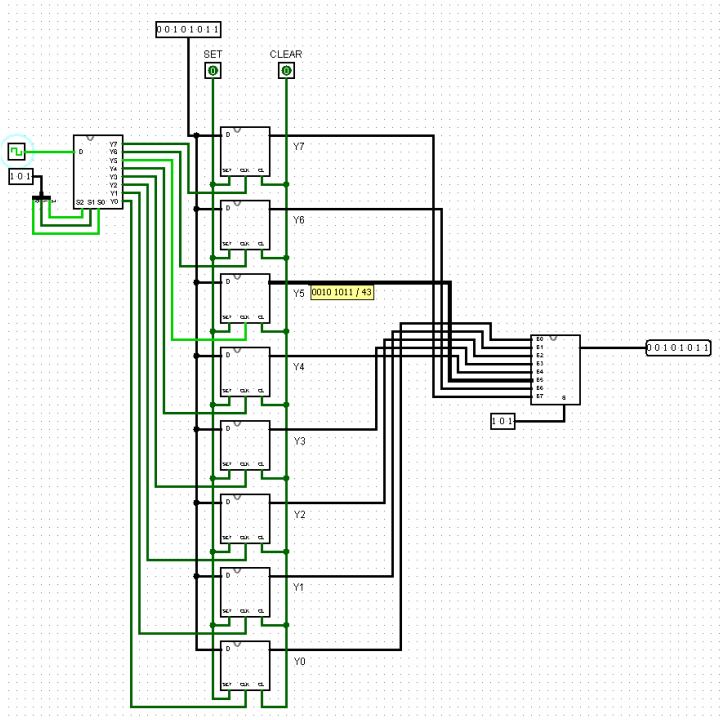
O demultiplexador 1x8 recebe como entrada três linhas de seleção (A2, A1, A0), as quais determinam qual das oito saídas será ativada. Esse componente é utilizado como um decodificador de escrita: quando um endereço é selecionado, apenas a saída correspondente fica em nível lógico ativo, permitindo escrever dados no registrador correspondente.

Em seguida, a memória conta com oito registradores de 8 bits, cada um capaz de armazenar um valor de 8 bits. Todos os registradores recebem o mesmo conjunto de dados na entrada, porém somente aquele que estiver habilitado pela linha ativa do demultiplexador terá seu conteúdo atualizado no momento do pulso de clock associado ao sinal de escrita. Dessa forma, a escolha do registrador alvo para armazenamento é feita pelo endereço aplicado ao demultiplexador.

Para a leitura dos dados armazenados, utiliza-se um multiplexador 8x1 de 8 bits. Esse multiplexador recebe as saídas de todos os oito registradores e, por meio de três linhas de seleção (A2, A1, A0), escolhe qual registrador terá seu conteúdo encaminhado à saída. Ao fornecer o mesmo endereço utilizado para escrita, é possível recuperar o dado previamente armazenado naquele registrador.

Em suma, o demultiplexador seleciona qual registrador irá receber os dados, os oito registradores retêm seus valores até que uma nova escrita ocorra, e o multiplexador permite a leitura do valor armazenado em um endereço específico.

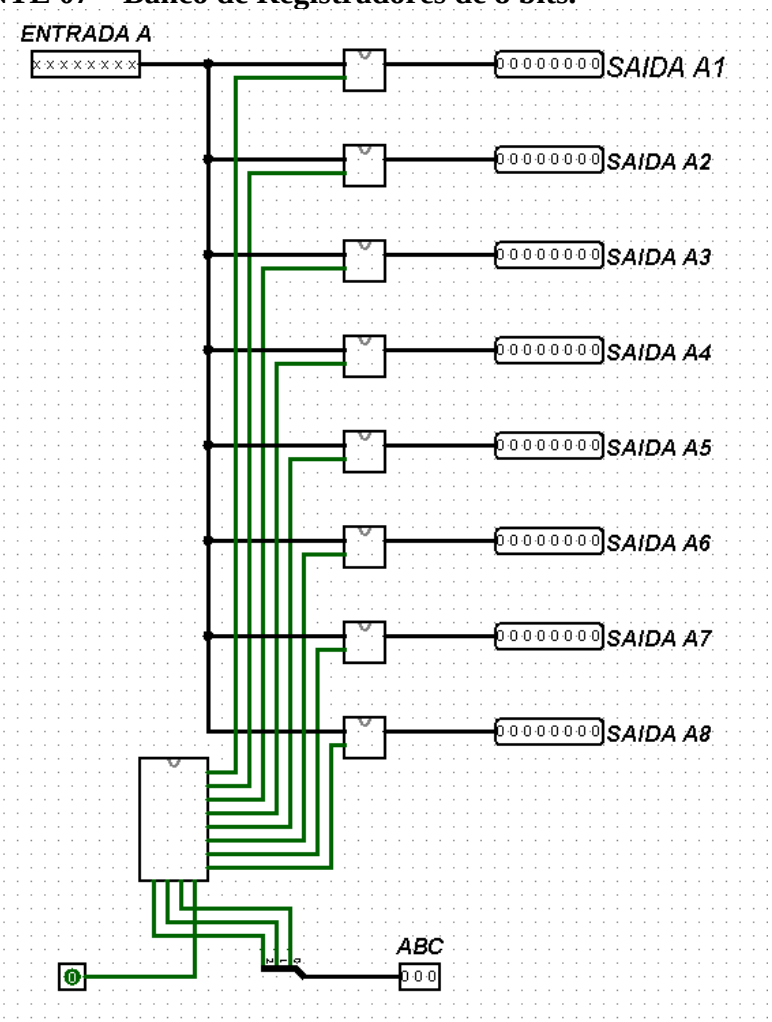
Testes RAM:



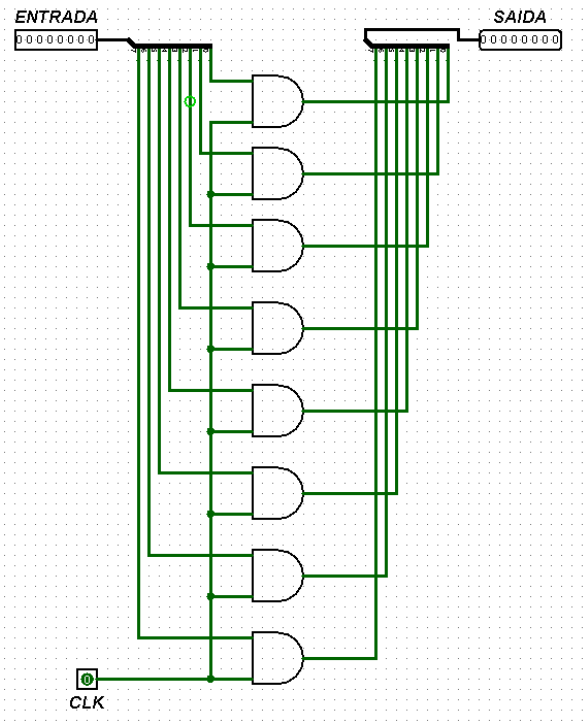
Endereços:

S DEMULTIPLEXADOR	REGISTRADOR	SAÍDA
000	**Y0**	**010**
001	**Y1**	**011**
010	**Y2**	**000**
011	**Y3**	**001**
100	**Y4**	**110**
101	**Y5**	**111**
110	**Y6**	**100**
111	**Y7**	**101**

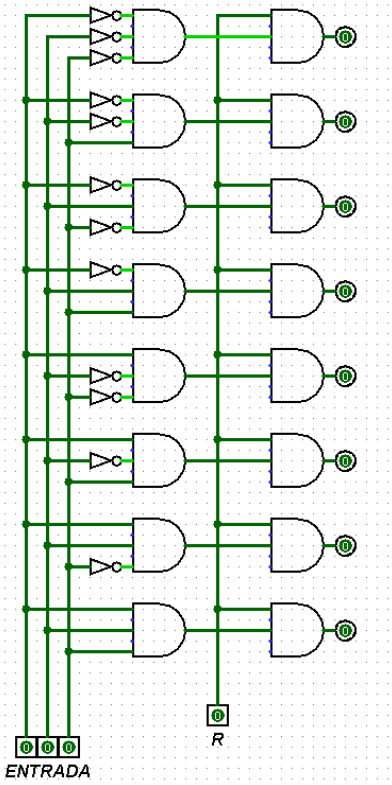
COMPONENTE 07 – Banco de Registradores de 8 bits.



[componentes]



Registrador de 8 bits



Demultiplexador 3-8

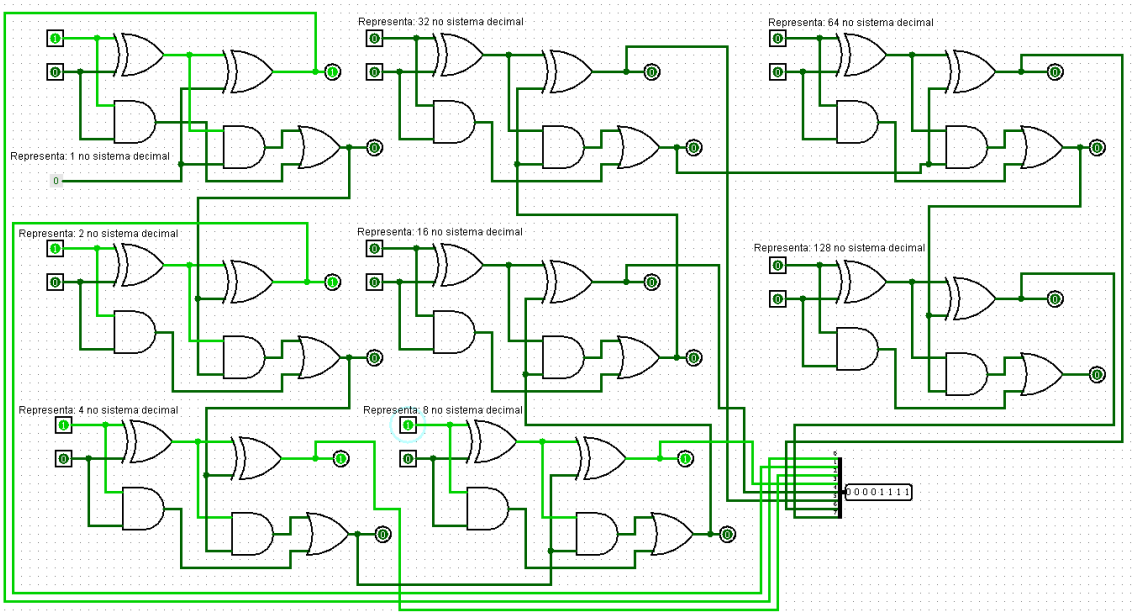
banco de registradores, uma estrutura clássica na arquitetura de computadores, empregada para armazenar múltiplos valores em registradores separados, mas endereçados seletivamente por sinais de controle. Este banco de registradores recebe um conjunto de linhas de seleção, uma linha de controle (R) para habilitar a escrita, e um barramento de entrada de dados (A[7:0]) que será escrito no registrador selecionado. Ao final, cada registrador fornece uma saída de 8 bits, compondo o conjunto de saídas do banco.

Na base do circuito, temos um demultiplexador 3-para-8. Ele recebe 3 linhas de seleção (A, B, C) e 1 sinal de controle de escrita (R)

Este demultiplexador direciona o sinal R para exatamente uma de suas 8 saídas, de acordo com a combinação binária nas entradas A, B, C. Assim, se A, B, C representam o endereço do registrador a ser escrito, apenas a saída correspondente àquele endereço será ativada (igual ao valor de R). As demais saídas permanecerão em zero. Ele age como um seletor de destino. Ao receber o sinal R e o endereço de seleção (A, B, C), o demultiplexador garante que somente um dos 8 registradores superiores "veja" um sinal ativo de carga. Em outras palavras, ele habilita a escrita apenas no registrador endereçado.

Acima do demultiplexador, temos 8 registradores (cada um de 8 bits). Cada registrador possui uma entrada de 8 bits, uma linha de habilitação de escrita (fornecida por uma das saídas do demultiplexador), uma saída de 8 bits que retém o valor armazenado no registrador. Cada registrador funciona de forma síncrona a um sinal de clock. Quando o sinal do demultiplexador corresponde a 1, no momento da borda de subida do clock, o valor presente na entrada A é carregado e armazenado. Se a linha de habilitação não estiver ativa, o registrador mantém seu valor anterior. Os registradores armazenam valores de 8 bits independentemente. Ao todo, com 8 registradores, o banco oferece armazenamento para 8 palavras de 8 bits.

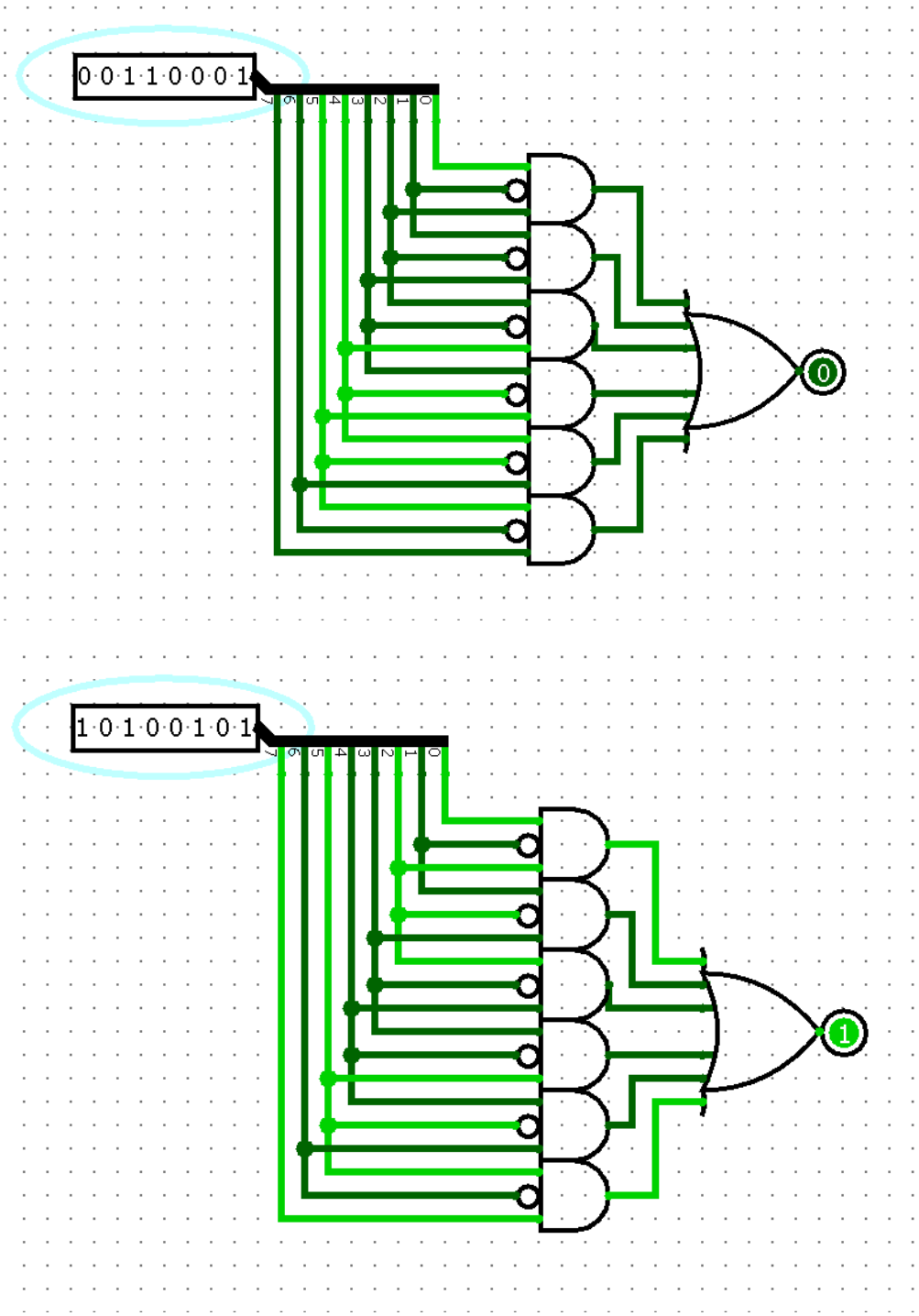
COMPONENTE 08 – Somador de 8 bits:



O somador de 8 bits é um componente responsável por realizar a soma de dois números inteiros binários de 8 bits, resultando em um valor também de 8 bits, além de um possível bit de carry-out (transbordamento). Sua construção baseia-se no encadeamento de 8 somadores completos (full adders), onde cada somador completo opera sobre um bit dos operandos e o bit de carry proveniente do estágio anterior.

Cada Full Adder recebe dois bits correspondentes — um do operando A e outro do operando B — além de um sinal de carry in (exceto o primeiro, que recebe sempre 0), e produz um bit de soma e um bit de carry out. Esse carry out é então propagado para o próximo Full Adder na cadeia. O carry out é gerado sempre que houver mais de um bit '1' entre as entradas a, b, cin.

COMPONENTE 09 – Construa um detector de sequência binária para identificar a sequência “101” em um fluxo de entrada.



O objetivo deste circuito é detectar a ocorrência da sequência binária "101" em um fluxo de 8 bits fornecido como entrada. A saída será ativada (em nível alto) se, em qualquer conjunto consecutivo de três bits dentro desses 8 bits, a sequência "101" for encontrada. Caso contrário, a saída permanece em nível baixo.

Entrada: Uma entrada de 8 bits, onde cada bit I[i] (0 ≤ i ≤ 7) é um sinal binário individual.

O arranjo formará 6 subconjuntos sobrepostos de 3 bits cada:

- Grupo 1: I[0], I[1], I[2]
- Grupo 2: I[1], I[2], I[3]
- Grupo 3: I[2], I[3], I[4]
- Grupo 4: I[3], I[4], I[5]
- Grupo 5: I[4], I[5], I[6]
- Grupo 6: I[5], I[6], I[7]

Para detectar a sequência "101" em um triplo de bits (x, y, z), é preciso que:

x = 1

y = 0

z = 1

Temos 6 portas AND, cada uma com 3 entradas. Em cada porta AND o pino do meio (ou pino 2) está negado, isso significa que para a sequência "101", a lógica da AND será: AND(x,~y,z)

Onde x e z devem ser iguais a 1, e y deve ser igual a 0 para que a saída da AND seja 1.

Testes Pontuais:

- Caso sem a sequência "101":
  Exemplo: I = 00000000

Nenhuma AND encontrará "101" ⇒ S = 0

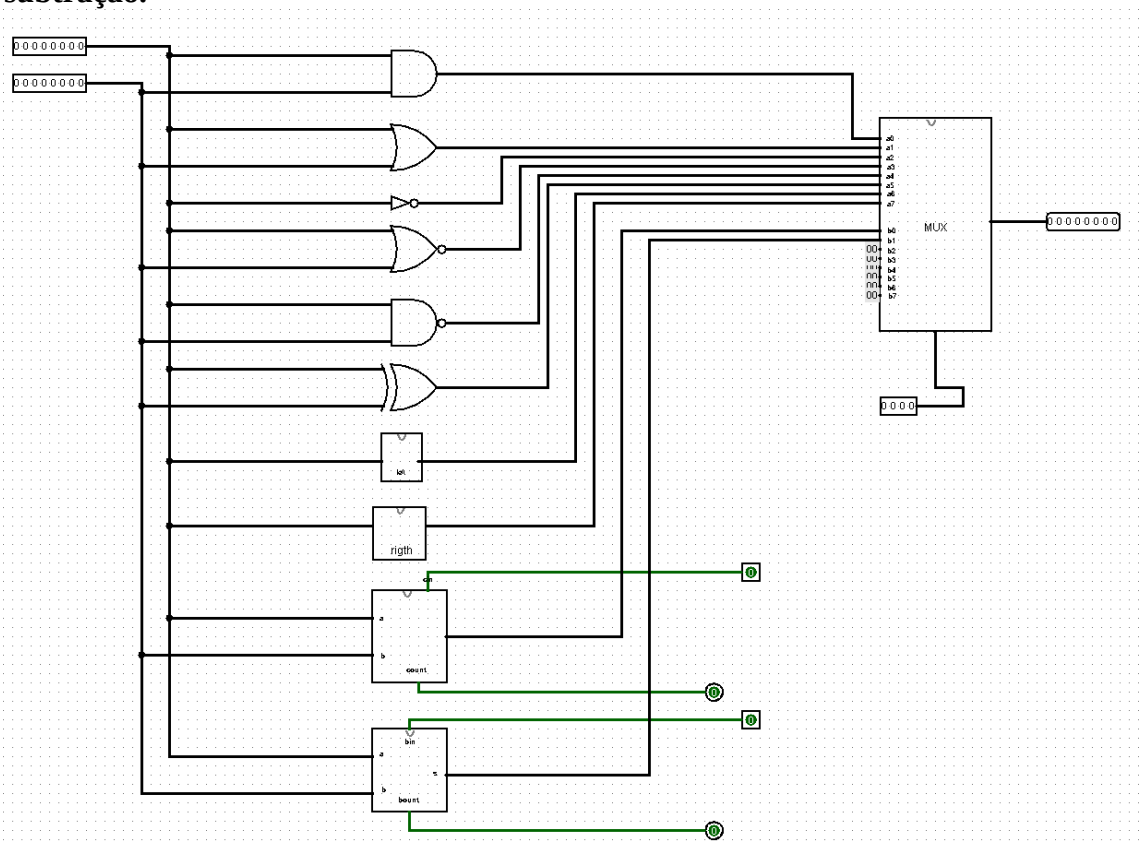
- Caso com a sequência "101" bem no início:
  Exemplo: I = 10100000

AND1 = (1, ¬0, 1) = 1 * 1 * 1 = 1

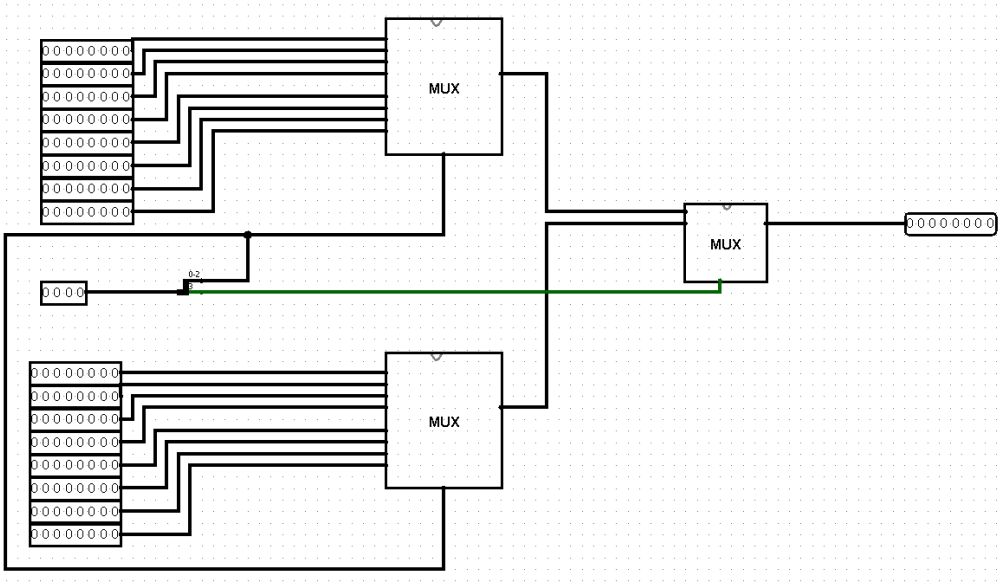
Assim, S = 1

- Caso com a sequência "101" no meio:
  Exemplo: I = 01010100
  Aqui, verificar cada tripla:
  - Grupo 1: 010 → não é "101"
  - Grupo 2: 101 → detecta
    Logo S = 1
- Caso com múltiplas ocorrências de "101":
  Exemplo: I = 10110100
  - Grupo 1: I[0..2] = 101 → detecta
  - Grupo 2: I[1..3] = 011 → não
  - Grupo 3: I[2..4] = 110 → não
  - Grupo 4: I[3..5] = 101 → detecta novamente
    Mesmo que fosse detectado apenas uma vez, S já seria 1, mas aqui vemos que o circuito pode detectar mais de uma ocorrência simultânea.

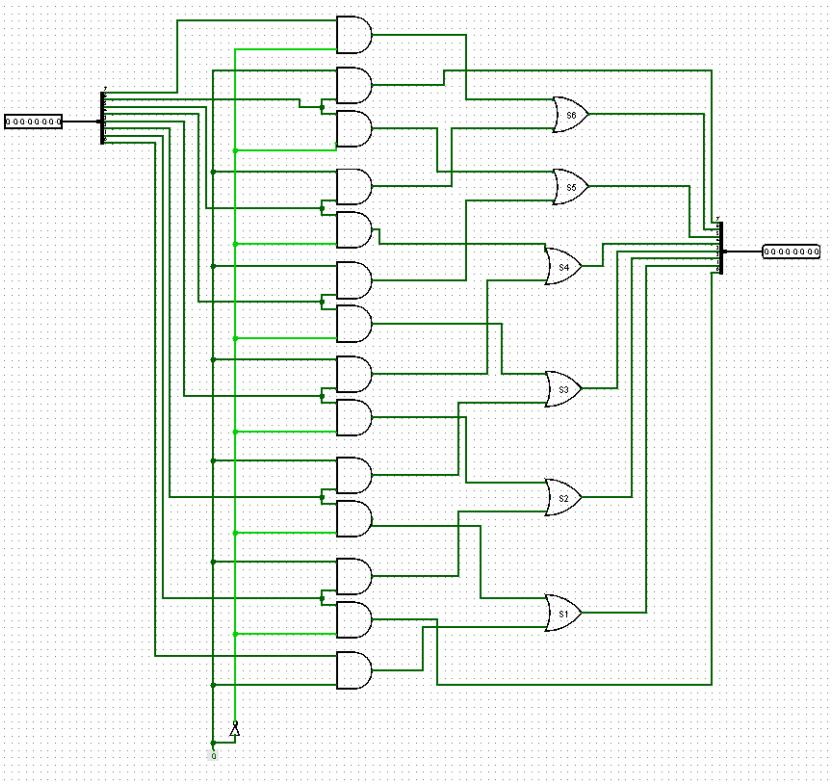
[COMPONENTE 10]. ULA de 8 bits com as seguintes operações: AND, OR, NOT, NOR, NAND, XOR, SHIFT de 2 bits à esquerda, SHIFT de bits à direita, soma e subtração.



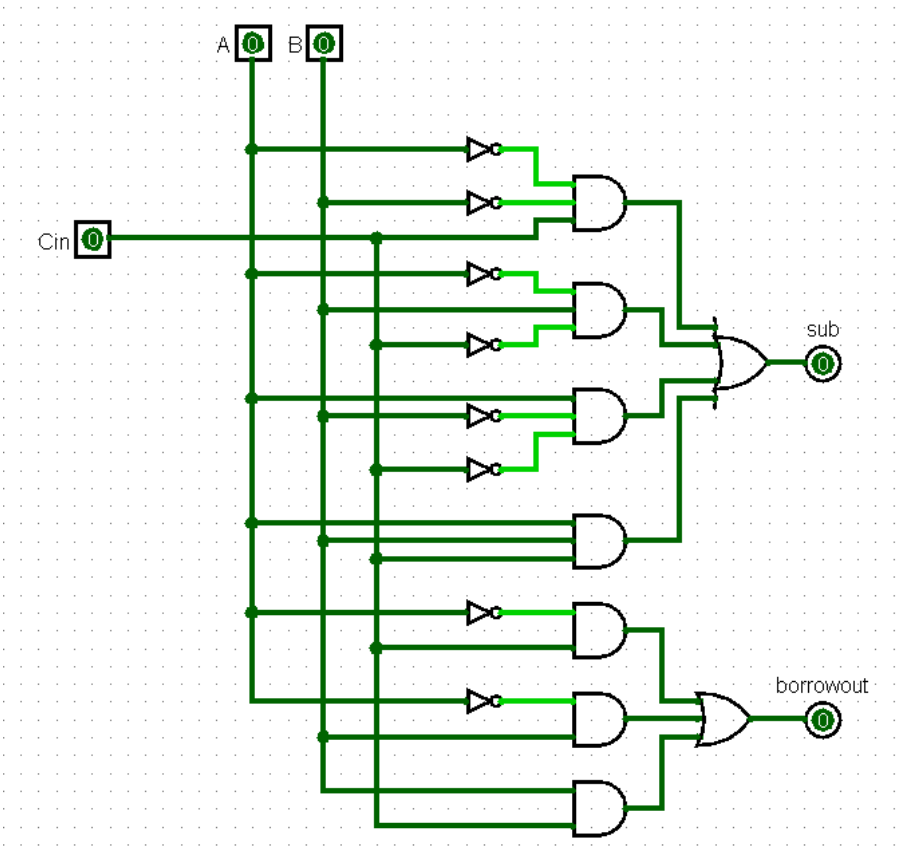
[componentes]



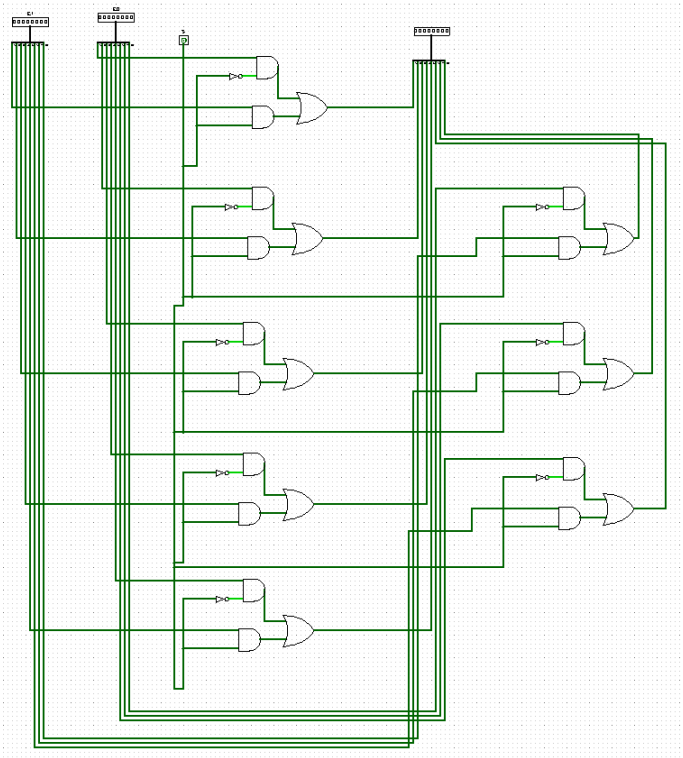
Mux 16-1



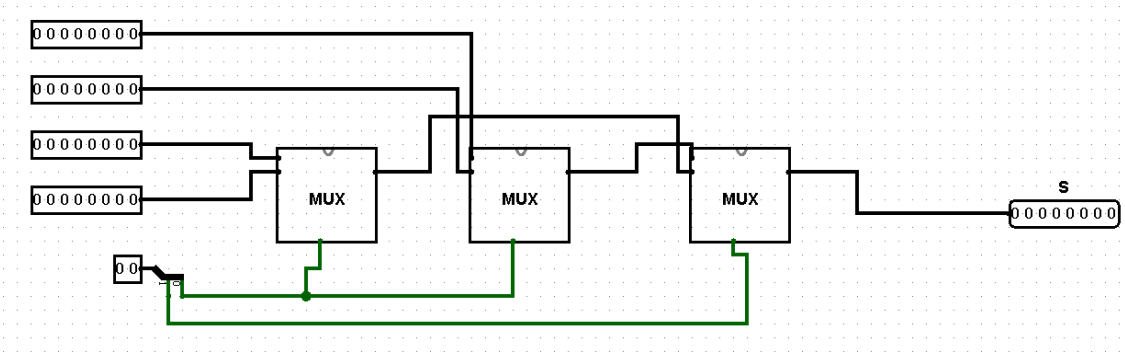
Deslocamento de bits



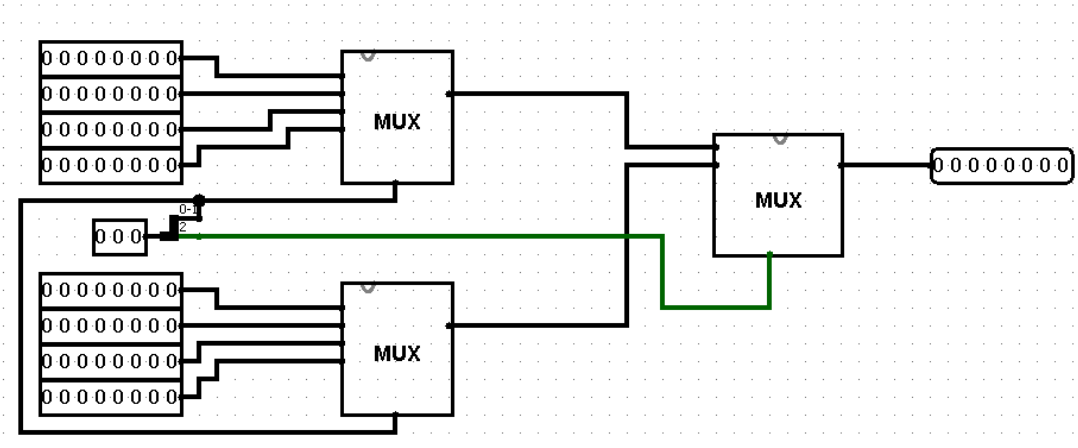
Subtração

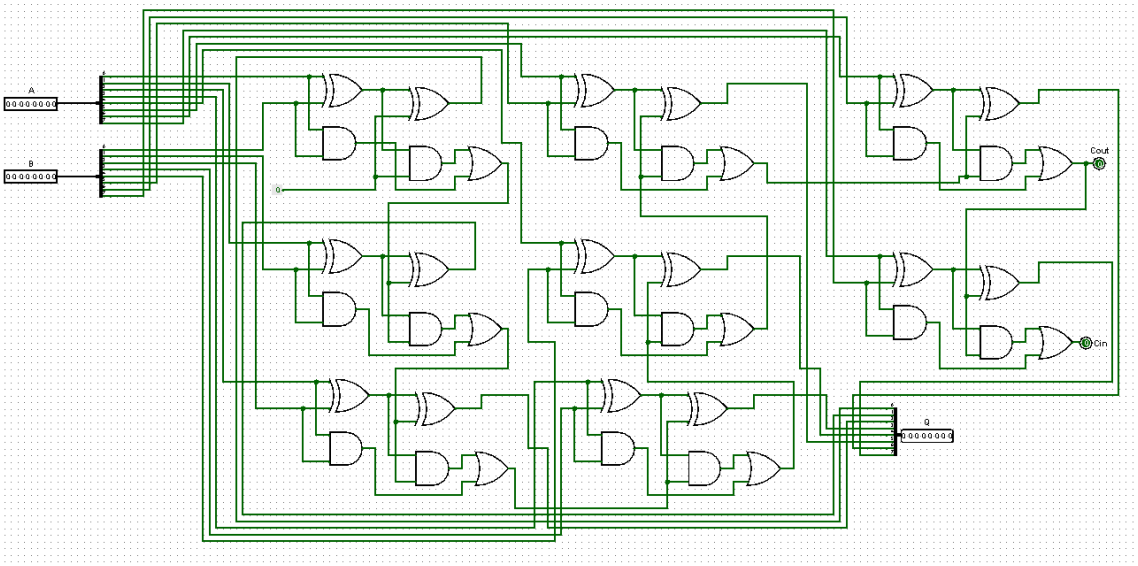


Mux 2x1 8 bits

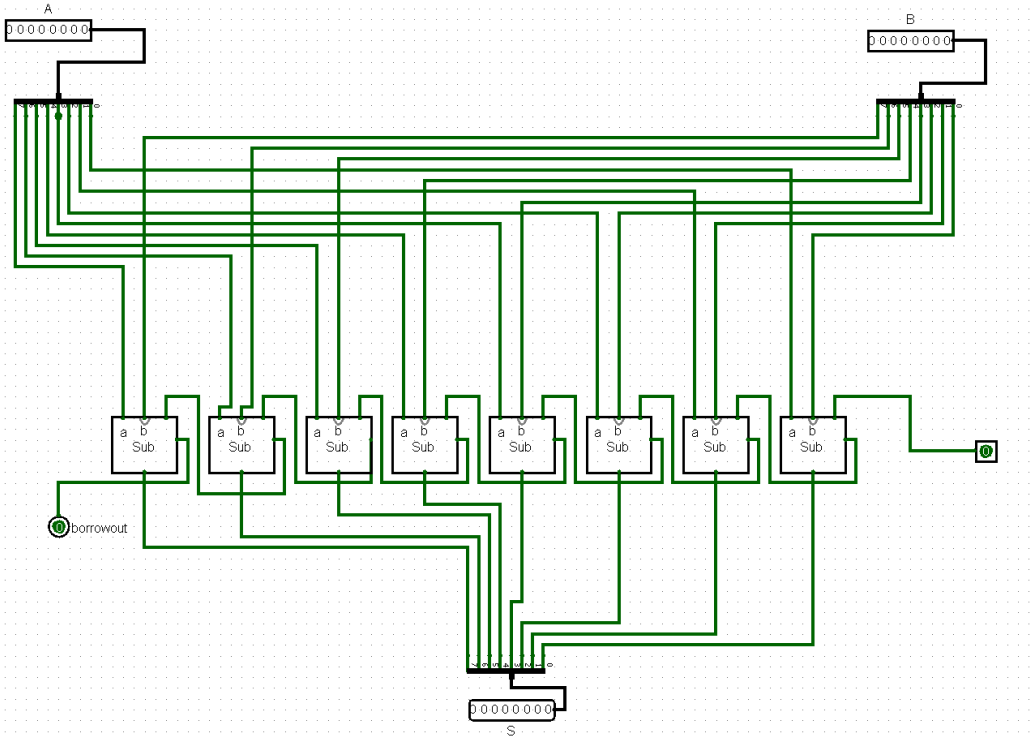


Mux 4x1 8 bits

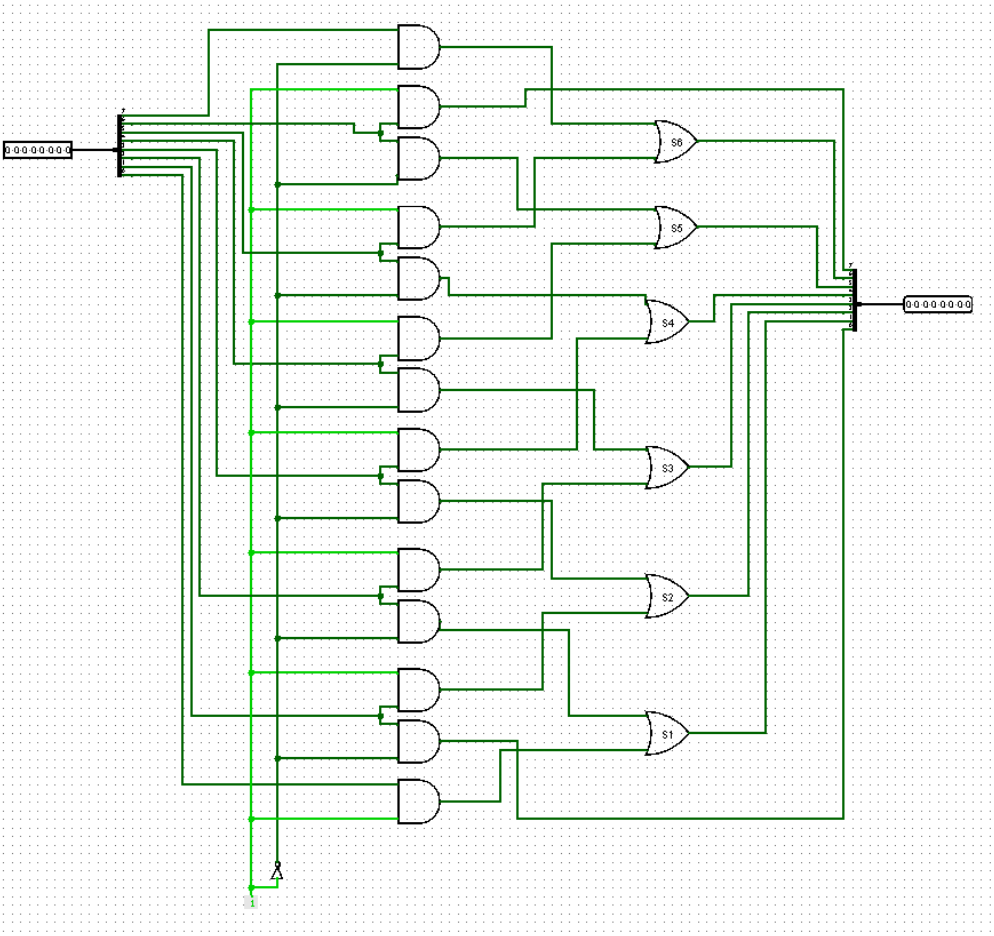




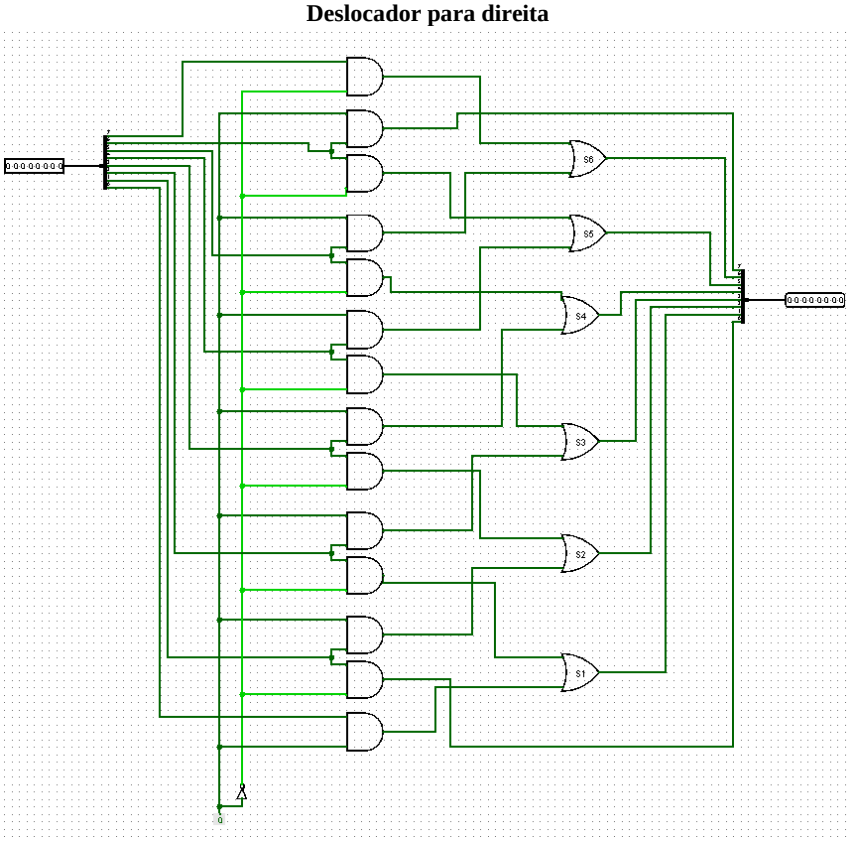
Somador 8 bits



Subtração 8 bits



Deslocador para direita



Deslocador para esquerda

A unidade Lógica e Aritimética (ULA) é um componente central de um processador simples, responsável por executar diversas operações lógicas e aritméticas sobre operandos de 8 bits. A ULA implementada contém um conjunto extenso de instruções, incluindo: AND, OR, NOT, NOR, NAND, XOR, deslocamento à esquerda, deslocamento à direita, soma e subtração.

A ULA recebe como entrada dois operandos de 8 bits, denominados A e B, e com base em sinais de controle adicionais (linhas de seleção, bits de carry-in, borrow-in e demais flags), a ULA produz uma saída de 8 bits que representa o resultado da operação selecionada. Adicionalmente, a ULA fornece sinais de saída do tipo carry-out (para operações de soma) e borrow-out (para operações de subtração).

A seleção da operação é realizada por um multiplexador 16x1 de 8 bits na saída final. Este MUX recebe, internamente, diferentes blocos funcionais, cada qual executa uma das operações desejadas. A partir de uma entrada de controle de 4 bits, o MUX de 16 entradas seleciona uma entre 16 possibilidades, garantindo a flexibilidade da ULA.

O MUX de 16x1 de 8 bits é formado internamente por:

- Dois MUX 8x1 de 8 bits cada um: cada MUX 8x1 recebe 8 entradas de dados, totalizando 8 x 8 bits.
- Uma linha de seleção de 4 bits controla tanto o MUX superior quanto o inferior. As três linhas de menor peso pode ser utilizadas para selecionar uma entre 8 entradas em cada MUX 8x1, enquanto o bit mais significativo seleciona qual dos dois MUX 8x1 é encaminhado ao estágio final.
- A saída dos dois MUX 8x1 é então roteada para um MUX 2x1 de 8 bits, que efetivamente escolhe entre o conjunto superior e o inferior, produzindo assim a saída de 8 bits final da ULA.

O Módulo de Deslocamento para a Direita recebe uma entrada de 8 bits e produz uma saída de 8 bits resultante do deslocamento para a direita. Através de uma porta NOT associada a uma constante 1, o comportamento do bit de entrada mais significativo será ajustado. Todo este processo de deslocamento de bit, podem ser vistos como uma série de estágios de filtragem e roteamento dos sinais.

Módulo de Deslocamento para a esquerda similar ao deslocamento para a direita, o deslocamento à esquerda utiliza uma porta NOT com constante 0 para definir o bit mais à direita após o deslocamento, 14 portas AND e 6 portas OR, dispostas de modo a mover cada bit para a posição adjacente de maior peso. O funcionamento é análogo ao módulo de deslocamento à direita, porém invertendo a direção do deslocamento: agora a saída [7] recebe entrada [6], saída [6] recebe entrada [5], etc., inserindo zeros no bit menos significativo.

O somador de 8 bits é composto por 8 full-adders (FA) em cascata:

- Entradas: A[7:0], B[7:0].
- Cada full-adder soma três bits: A[i], B[i] e o carry-in proveniente do estágio anterior.
- O primeiro full-adder recebe um carry-in (cin) de 0 (exceto se definido externamente), e o carry-out resultante torna-se o carry-in do próximo full-adder.
- Ao final, temos uma saída de 8 bits representando A + B (ou A + B + cin se o cin for 1), e um carry-out final que indica overflow ou que a soma extrapolou a capacidade de 8 bits.

Subtração de 8 Bits é implementada por 8 subtratores de 1 bit (cada um lidando com A[i] - B[i]):

- Entradas: A[8], B[8].
- Cada subtrator de 1 bit recebe A[i], B[i] e um borrow-in do estágio anterior.
- O primeiro subtrator recebe um borrow-in inicial (normalmente 0), e o borrow-out final pode indicar se houve um “empréstimo” que extrapolou a capacidade do módulo.
- A saída de 8 bits é o resultado de A - B, considerando também o borrow-in inicial.

Operações Lógicas Básicas (AND, OR, NOT, NOR, NAND, XOR) Cada operação lógica básica é implementada por um conjunto de portas lógicas:

- AND: Combina A[i] e B[i] bit a bit.
- OR: Combina A[i] e B[i] bit a bit.
- NOT: Aplica a negação bit a bit a um dos operandos (geralmente A).
- NOR, NAND, XOR: Analogamente implementadas, cada saída[i] é o resultado da operação correspondente sobre A[i] e B[i].

Cada bloco lógico fornece uma saída de 8 bits, resultante da aplicação da operação a cada par de bits de entrada.

Organizando todos os componentes, a ULA coloca lado a lado todas as operações que criamos acima através de circuitos. Para simplificar, vamos mapear as entradas de seleção do MUX 16x1 de 8 bits que realiza as operações:

SELEÇÃO (4 BITS)	OPERAÇÃO
0000	AND
0001	OR
0010	NOT
0011	NOR
0100	NAND
0101	XOR
0110	SHIFT LEFT
0111	SHIFT RIGHT
1000	SUM (A+B)
1001	SUB (A-B)
1010 a 1111	RESERVADAS (CONSTANTES 0)

TESTES:

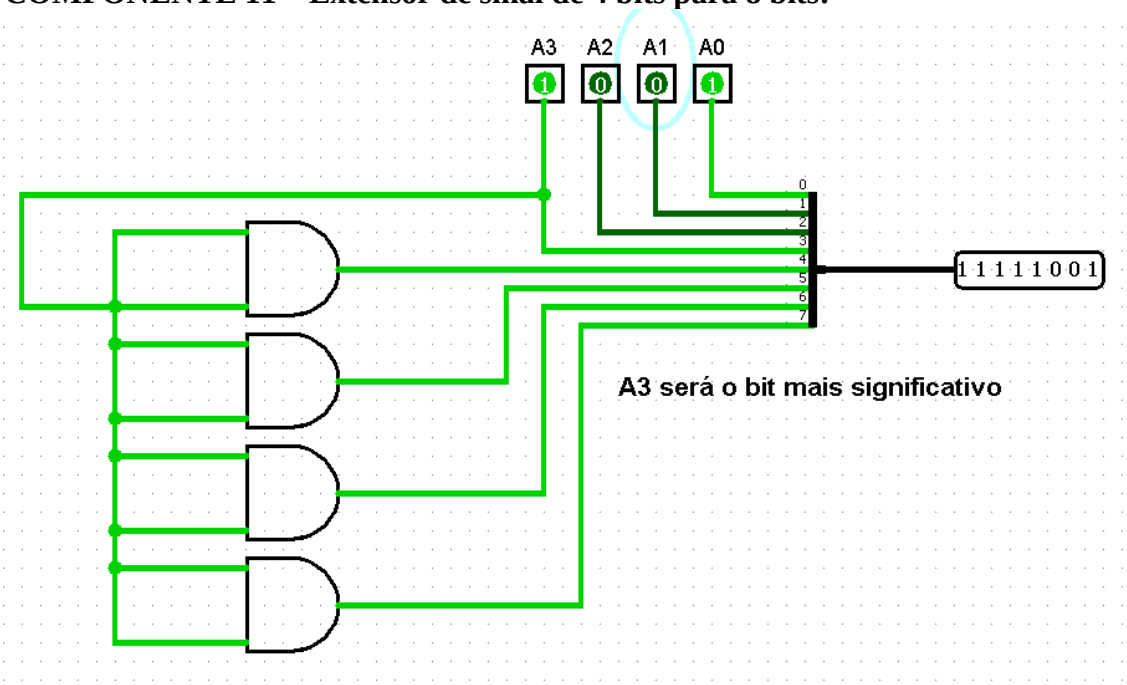
Exemplo de teste AND

Entrada: A = 0xF0 (11110000), B = 0x0F (00001111)
Saída esperada: 0x00 (00000000)

SHIFT LEFT:
Entrada: A = 0x0C (00001100)
Saída esperada para um deslocamento de 2 bits à esquerda: 0x30 (00110000)

Exemplo de teste SOMA:
Entrada: A = 0x12 (00010010), B = 0x34 (00110100)
Saída esperada: 0x46 (01000110), carry-out = 0.

COMPONENTE 11 – Extensor de sinal de 4 bits para 8 bits:



O circuito de extensão de sinal implementado acima tem como função principal "estender" um número de 4 bits para um formato de 8 bits, preservando o bit de sinal (o bit mais significativo, a3) nos bits adicionais. Para números binários com sinal em complemento de dois, o bit mais significativo (MSB) indica o sinal do número:

Caso a3 = 0, o número é não-negativo, e a extensão deve preencher os bits superiores com zeros.

Caso a3 = 1, o número é negativo, e a extensão deve preencher os bits adicionais à esquerda com uns, de modo a manter a representação correta.

Assim, ao receber um número de 4 bits (a3 a2 a1 a0), a extensão de sinal de 4 para 8 bits produz um número de 8 bits (a7 a6 a5 a4 a3 a2 a1 a0), onde os novos 4 bits mais significativos (a7 a6 a5 a4) devem ser cópias do bit de sinal a3. Dessa forma, se a3=1, os quatro bits superiores serão 1; se a3=0, os quatro bits superiores serão 0.

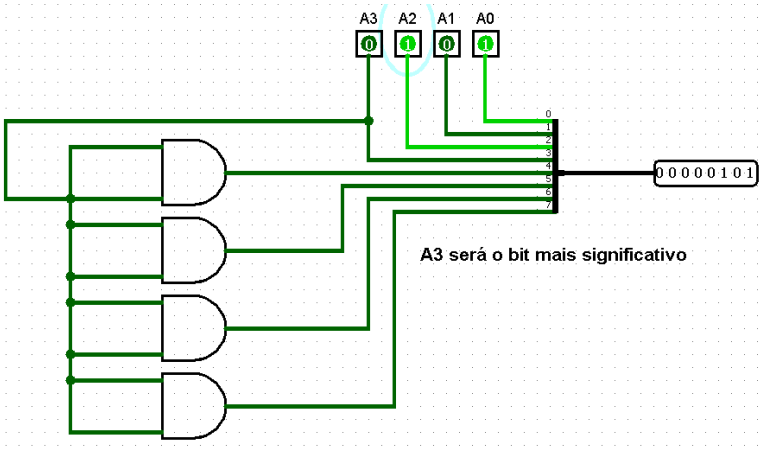
Teste Básico (a3=0):

Se a3=0, espera-se que os 4 bits mais significativos sejam todos 0.

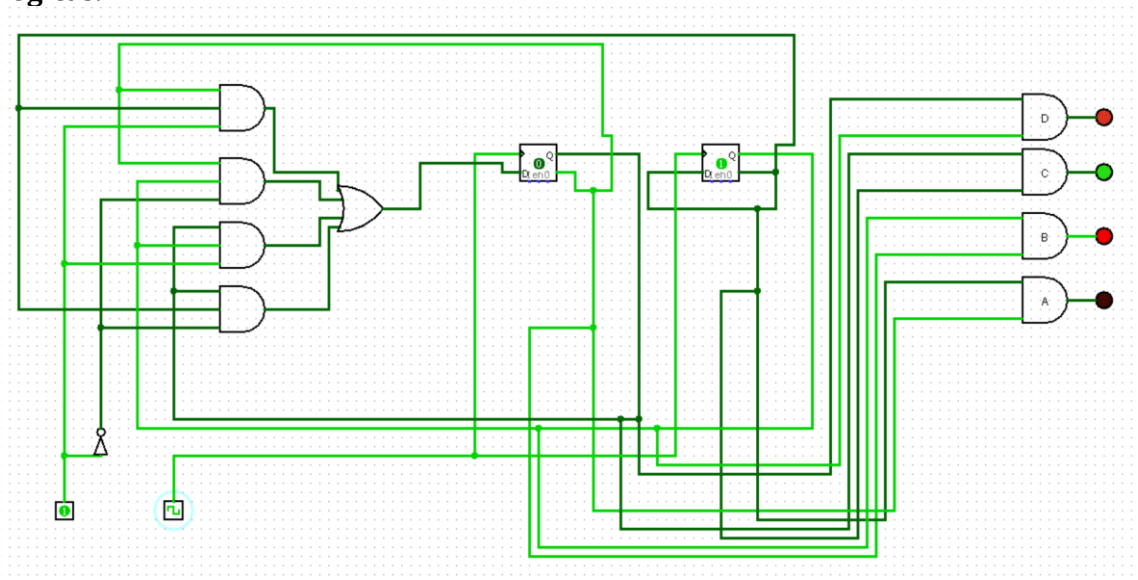
Entrada: a3 a2 a1 a0 = 0 1 0 1 (por exemplo, 0x05 em hexadecimal)

Saída esperada: a7 a6 a5 a4 a3 a2 a1 a0 = 0 0 0 0 0 1 0 1 = 0x05

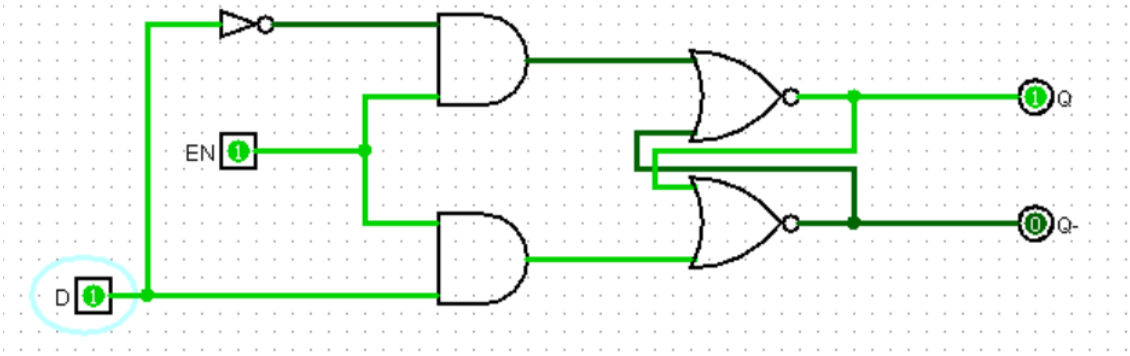
Aqui a saída não difere do valor original, apenas foi estendida com zeros.



COMPONENTE 12 – Implemente uma máquina de estados utilizando portas lógicas.



[componentes]



Registrador Flip-Flop do tipo D

A máquina de estados implementada no circuito acima, foi projetado para exemplificar o funcionamento de uma máquina de estados simples, com entradas e saídas representadas por LEDs, utilizando flip-flops tipo D para armazenar o estado e a lógica combinacional para controlar as transições de estado.

O circuito possui uma entrada de 1 bit, que alimenta dois caminhos:

- Um caminho direto, que segue para a primeira etapa da lógica combinacional.
- Um caminho que passa por uma porta NOT, invertendo o valor do bit de entrada. Este valor invertido será utilizado em outra parte da lógica para determinar as transições de estado.

As portas AND produzem sinais lógicos que serão usados para determinar o próximo estado da máquina. As portas AND são conectadas de forma a gerar uma combinação de sinais para alimentar uma porta OR.

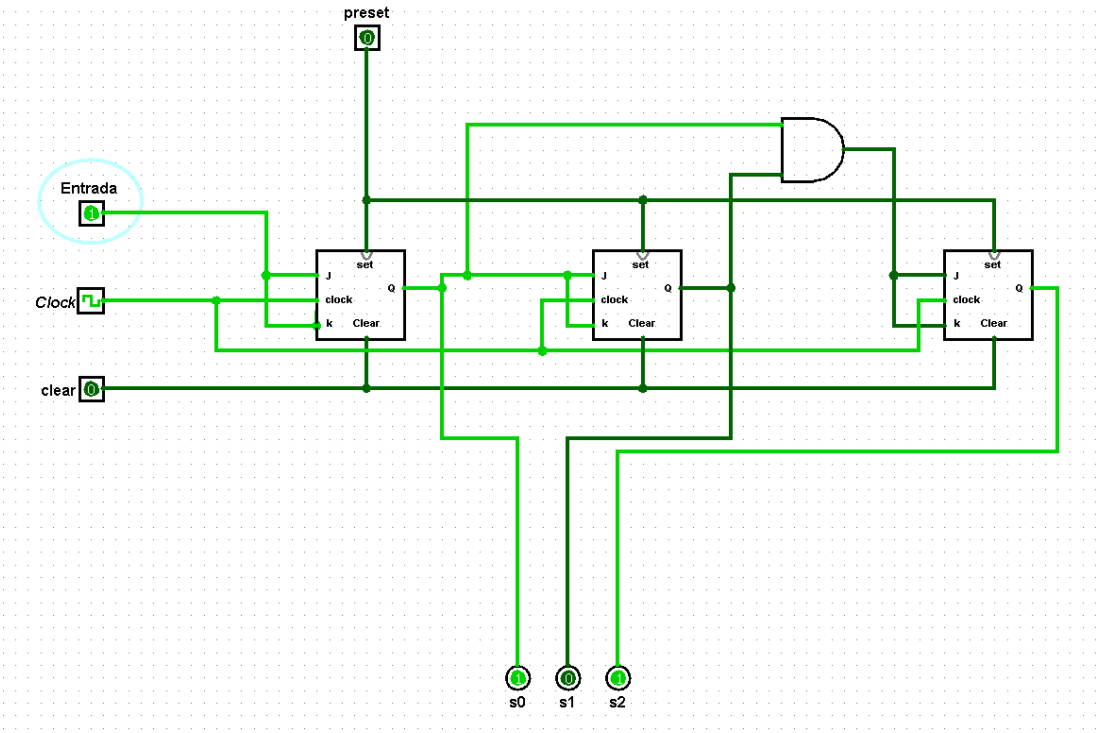
Os resultados das 4 portas AND são combinados em uma porta OR. O sinal de saída da porta OR é um dos elementos chave para controlar a transição entre os estados, dependendo do valor da entrada e do estado atual armazenado nos flip-flops.

O comportamento do circuito pode ser descrito como uma máquina de estados síncrona, em que a entrada de 1 bit e os estados armazenados nos flip-flops determinam

as saídas. A cada ciclo de clock, o circuito avalia a combinação de sinais lógicos e determina a próxima transição de estado.

Os LEDs acendem ou apagam de acordo com o estado atual da máquina, permitindo a visualização das transições de estado. A lógica combinacional, composta pelas portas AND, OR e NOT, gera os sinais de controle necessários para alimentar os flip-flops e determinar as saídas.

COMPONENTE 13 – Contador Síncrono:



Utilizado para contar pulsos de clock e representar o progresso do tempo ou a progressão de estados internos do sistema, o Contador Síncrono de 3 bits foi implementado com Flip-Flops JK, além de sinais de controle. Esse contador produzirá três saídas (s0, s1, s2) que variam de acordo com o número de pulsos de clock, percorrendo a sequência binária (exemplo: 000, 001, 010, 011, 100, 101, 110, 111 e 000).

A sequência de contagem típica (assumindo J=K=1 no primeiro FF e entrada CLR inicialmente acionada para zerar tudo) é:

Estado inicial (após CLR): s2 s1 s0 = 0 0 0

Primeiro pulso de clock: s0 toggla para 1, s1 e s2 permanecem 0 → estado: 0 0 1

Segundo pulso de clock: s0 toggla para 0 (pois J=K=1 no primeiro FF), como s0=1 na borda anterior, o segundo FF toggla s1 para 1 → estado: 0 1 0

Terceiro pulso de clock: s0 toggla para 1 novamente, com s0=0 anteriormente, o segundo FF não toggla (pois J=K=s0=0 naquele instante de amostragem), então s1 permanece 1. Agora temos s0=1 e s1=1, ativando a saída AND=1. Mas o toggling do terceiro FF só ocorrerá no próximo pulso. Estado atual: 0 1 1

Para validar o contador, podem-se empregar testes de unidade e análise de tabela de estados.

Testes Básicos por Ciclos de Clock:

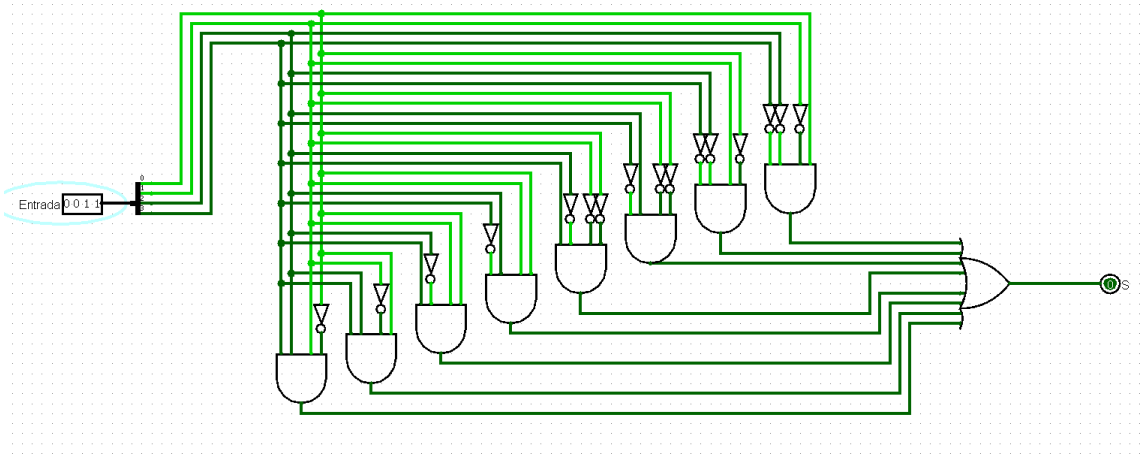
Inicialmente, acione CLR para garantir s2 s1 s0 = 000.

Forneça um pulso de clock e verifique se o contador foi para 001.

Continue aplicando pulsos de clock e confirme a sequência:

000 → 001 → 010 → 011 → 100 → 101 → 110 → 111 → (próximo pulso) 000 …

COMPONENTE 14 – Combine portas AND, OR e NOT para criar a lógica de um detector de paridade ímpar (entrada com número ímpar de 1s)



O objetivo do circuito é detectar se o número de bits '1' em uma entrada de 4 bits é ímpar. Caso a entrada apresente 1, 3 ou todos os 1 (no caso de quatro bits, paridades possíveis são 0, 1, 2, 3 ou 4 bits a 1), a saída deverá indicar 1 (verdadeiro) quando houver uma contagem ímpar de bits igual a 1.

A função de paridade ímpar é 1 se o número de bits '1' na entrada for ímpar. Portanto, consideramos as seguintes combinações como saída=1:

Com exatamente 1 bit igual a 1 (e os outros 3 bits = 0):

0001 (A3=0, A2=0, A1=0, A0=1)

0010 (A3=0, A2=0, A1=1, A0=0)

0100 (A3=0, A2=1, A1=0, A0=0)

1000 (A3=1, A2=0, A1=0, A0=0)

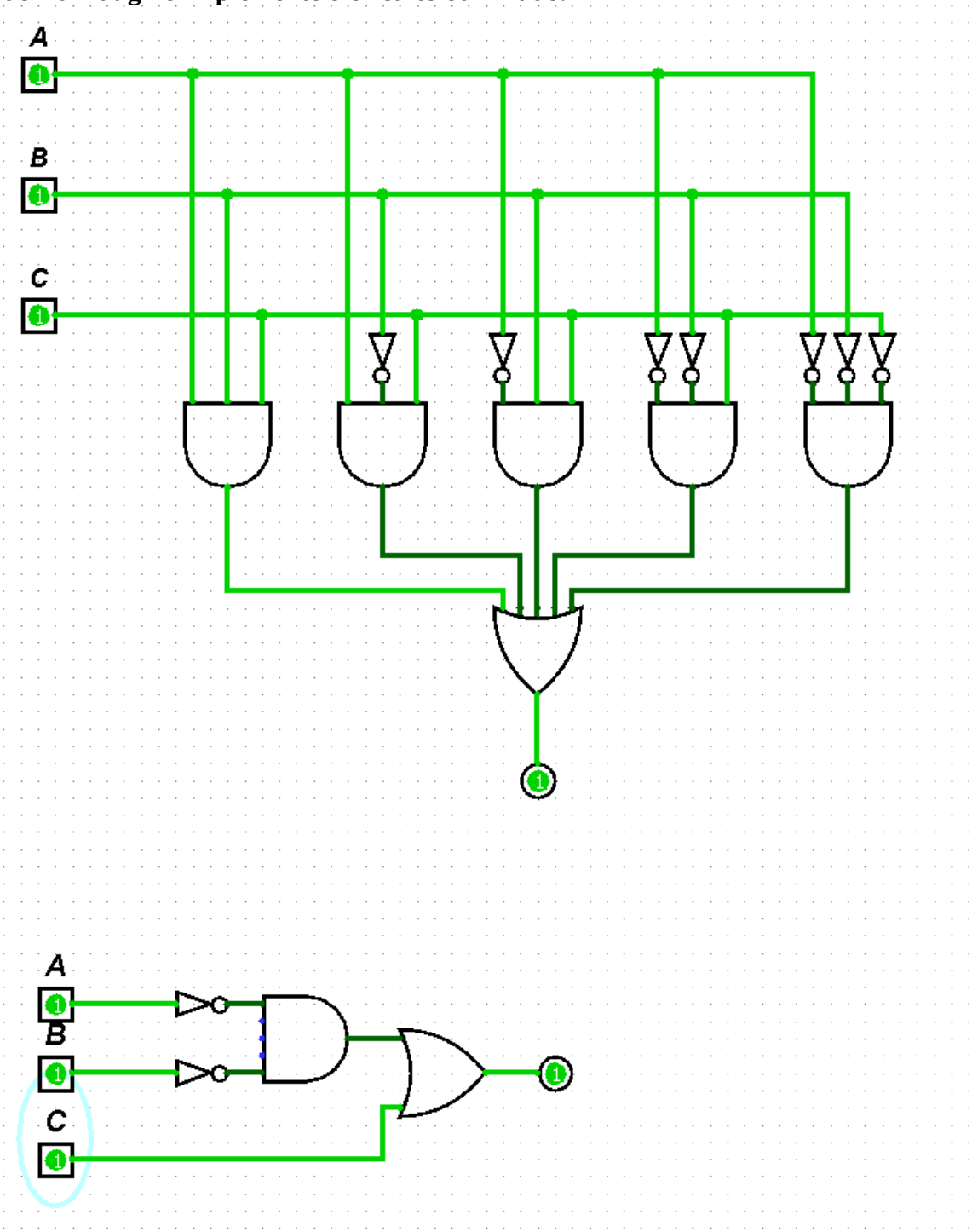
Com exatamente 3 bits iguais a 1 (e 1 bit = 0): 5) 0111 (A3=0, A2=1, A1=1, A0=1) 6) 1011 (A3=1, A2=0, A1=1, A0=1) 7) 1101 (A3=1, A2=1, A1=0, A0=1) 8) 1110 (A3=1, A2=1, A1=1, A0=0)

Empregamos a abordagem de soma de mintermos (forma canônica) para construir o circuito. Cada uma das 8 combinações acima será implementada como um mintermo, ou seja, um produto (AND) das variáveis ou suas negações, resultando em 1 somente para aquela combinação específica.

Testes:

- Testar entradas com 1 bit igual a 1 (por exemplo, 0001): a saída deve ser 1.
- Testar entradas com 2 bits iguais a 1 (por exemplo, 0011): a saída deve ser 0.
- Testar entradas com 3 bits iguais a 1 (por exemplo, 0111): a saída deve ser 1.

COMPONENTE 15 – Resolva um problema de otimização lógica utilizando mapas de Karnaugh e Implemente o circuito otimizado.



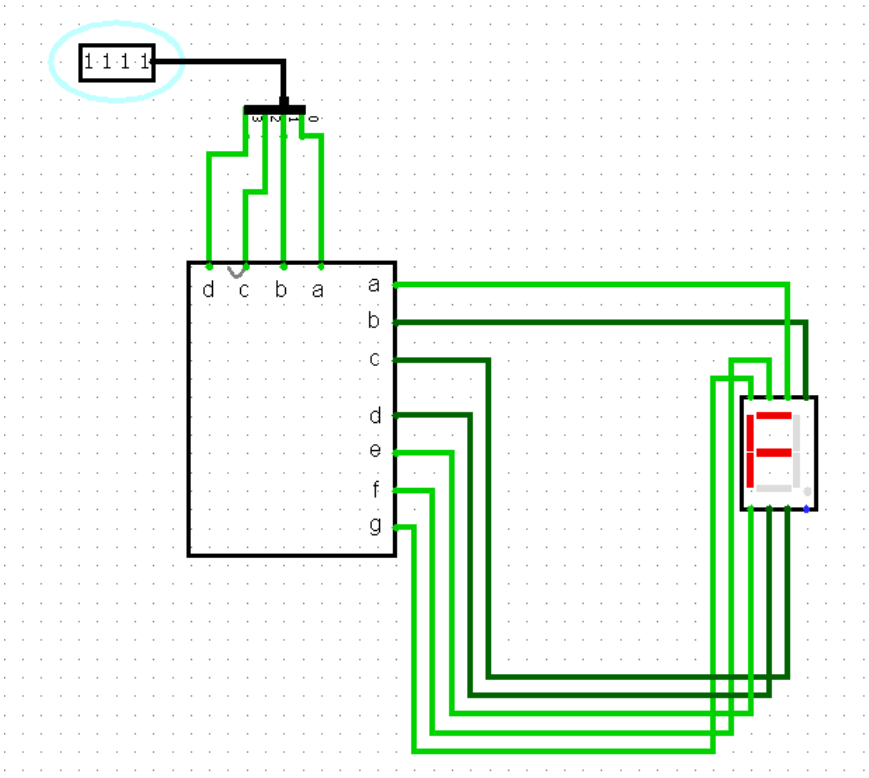
A imagem mostra um circuito lógico que foi otimizado a partir de mapas de Karnaugh, partindo de três entradas: A, B e C. Após a análise e simplificação, o objetivo era encontrar a forma mais reduzida possível da função lógica resultante. Eis um resumo detalhado do circuito e das funções das portas lógicas:

1. **Entradas (A, B, C)**:
   São as três variáveis lógicas de entrada. Cada uma pode assumir valor 0 (falso) ou 1 (verdadeiro).

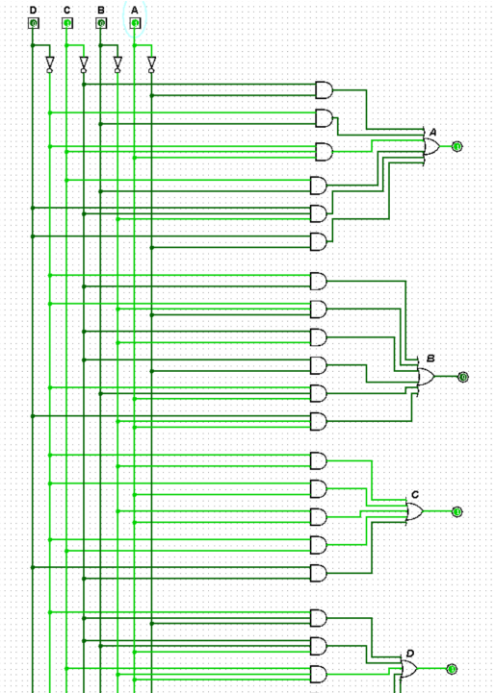
2. **Conjunto de Portas AND com combinações de B e C invertidas ou não**:
   O circuito apresentado na imagem possui quatro portas AND (E) paralelas, cada uma recebendo o sinal A, e uma combinação específica de B e C (com possíveis inversões). As portas AND geram mintermos — ou seja, termos que representam combinações específicas de A, B e C que levam a saída a 1.
   As quatro portas AND mostradas parecem ser (do lado esquerdo para o direito):
   - Primeira porta AND: recebe A, B, C sem inversões, produzindo o mintermo **A·B·C**.
   - Segunda porta AND: recebe A, B e C com C invertido (¬C), produzindo **A·B·¬C**.
   - Terceira porta AND: recebe A, B com B invertido (¬B) e C normal, produzindo **A·¬B·C**.
   - Quarta porta AND: recebe A com B e C invertidos (¬B e ¬C), produzindo **A·¬B·¬C**.
3. Cada uma destas portas AND verifica se há uma combinação específica de (A,B,C) igual a (1,1,1), (1,1,0), (1,0,1) ou (1,0,0) respectivamente.
4. **Porta OR Final**:
   A saída de todas as portas AND é combinada em uma porta OR (OU), que produz a soma lógica desses mintermos. Na prática, a saída final é o OR de todos os mintermos encontrados:
   (A·B·C)+(A·B·¬C)+(A·¬B·C)+(A·¬B·¬C)(A·B·C) + (A·B·¬C) + (A·¬B·C) + (A·¬B·¬C)(A·B·C)+(A·B·¬C)+(A·¬B·C)+(A·¬B·¬C)
5. **Simplificação via Karnaugh**:
   A partir do mapa de Karnaugh, todos esses quatro mintermos podem ser combinados, pois juntos eles cobrem todas as possibilidades de B e C quando A=1. Isso significa que, independentemente do valor de B e C, se A=1, a saída será 1; se A=0, a saída será 0. Logo, a expressão simplifica-se a apenas:
   F=AF = AF=A
   Ou seja, o resultado otimizado da função lógica é simplesmente igual à entrada A.

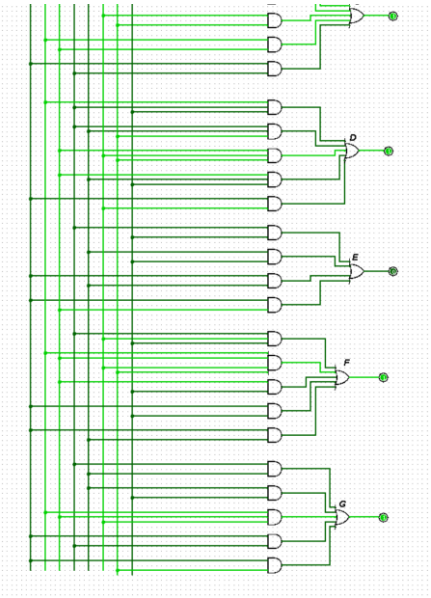
Em resumo: O circuito mostrado, embora apresente quatro portas AND combinando diversas inversões em B e C, mais uma porta OR ao final, pode ser totalmente reduzido a um único fio da entrada A. A presença dos mintermos AB*C, AB¬C, A¬BC e A¬B¬C apenas confirmam que, quando A=1, não importa o valor de B ou C, a saída será 1. Portanto, a função implementada após otimização é a mais simples possível: F = A.

COMPONENTE 16 - Decodificador de 7 Segmentos: Projete um circuito que converta um número binário de 4 bits para os sinais necessários para acionar um display de 7 segmentos (formato hexadecimal).



[componentes]





Decodificador

O decodificador de 7 segmentos é um circuito digital que é capaz de converter um número binário de 4 bits em sinais apropriados para acionar um display de 7 segmentos, exibindo números no formato hexadecimal. O circuito utiliza um decodificador para gerar os sinais necessários para acionar cada um dos 7 segmentos do display.

A entrada do circuito é composta por 4 bits (A, B, C, D), representando um número binário de 4 bits que varia de 0000 a 1111 (0 a 15 em decimal, ou 0 a F em hexadecimal). Cada um desses bits é uma entrada para o decodificador que será responsável por gerar os sinais correspondentes aos segmentos do display.

O decodificador de 7 segmentos possui 4 entradas (A, B, C, D) e gera 7 saídas (a, b, c, d, e, f, g), que controlam os segmentos do display. Para acionar os segmentos corretamente, o circuito usa portas lógicas, como AND, OR e NOT, para processar as entradas binárias e gerar as saídas.

Na parte superior esquerda do circuito, temos a entrada de 4 bits (A, B, C, D) que alimenta o decodificador. O decodificador processa as entradas e gera os sinais necessários para acionar os segmentos do display. O display de 7 segmentos possui 7 entradas (a, b, c, d, e, f, g), que são controladas pelas saídas do decodificador.

O display de 7 segmentos possui 7 LEDs que podem ser acionados de forma individual para formar os números de 0 a 9 e as letras de A a F (hexadecimal). O circuito deve gerar os sinais corretos para acionar os segmentos e representar o número correspondente à entrada binária.

Tabela Verdade

A tabela verdade é fundamental para o funcionamento do decodificador, pois ela define como cada combinação de 4 bits (A, B, C, D) deve acionar os segmentos do display de 7 segmentos.

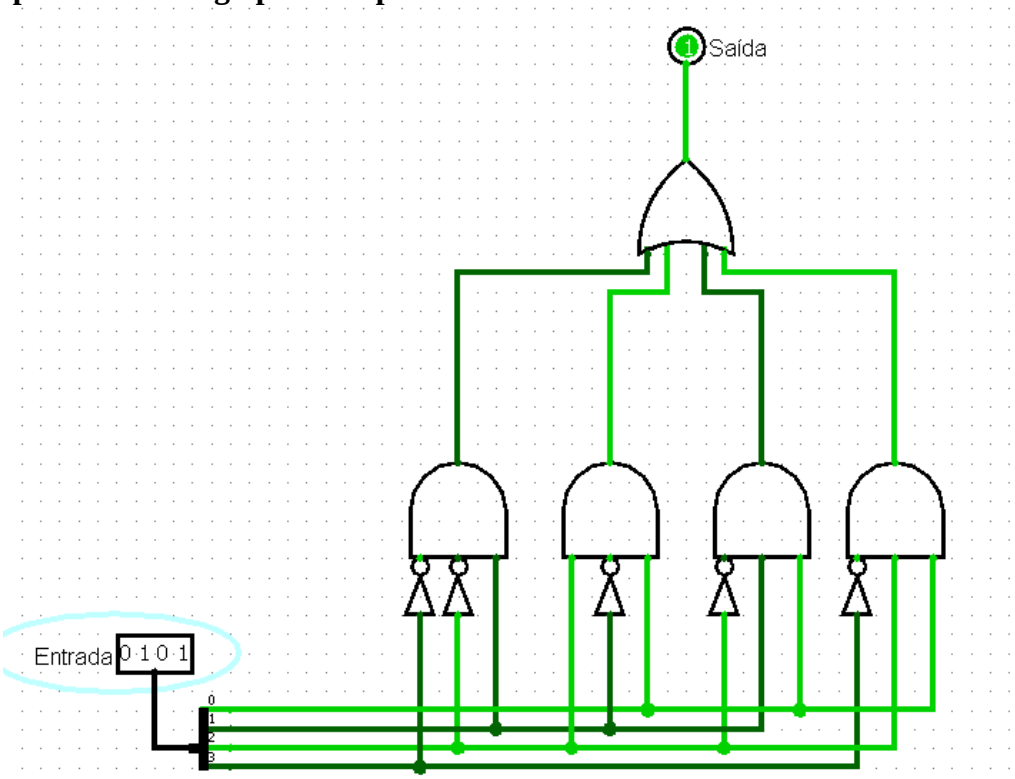
A tabela verdade é construída com base na codificação hexadecimal, onde os números de 0 a 15 (0000 a 1111 em binário) devem acionar os segmentos de maneira que mostram os números e letras correspondentes no display de 7 segmentos.

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	1	0

0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
0	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
0	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
0	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**
1	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
1	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**
1	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
1	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
1	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**

Para cada valor de entrada (de 0000 a 1111), foi verificado se a combinação de saídas (a, b, c, d, e, f, g) acionava corretamente os segmentos do display, exibindo o número hexadecimal correspondente. Os resultados foram comparados com a tabela verdade mencionada anteriormente para garantir a precisão do circuito.

COMPONENTE 17 - Detector de Número Primo: Crie um circuito que detecte se uma entrada binária de 4 bits representa um número primo. Utilize portas lógicas e mapas de Karnaugh para simplificar o circuito.



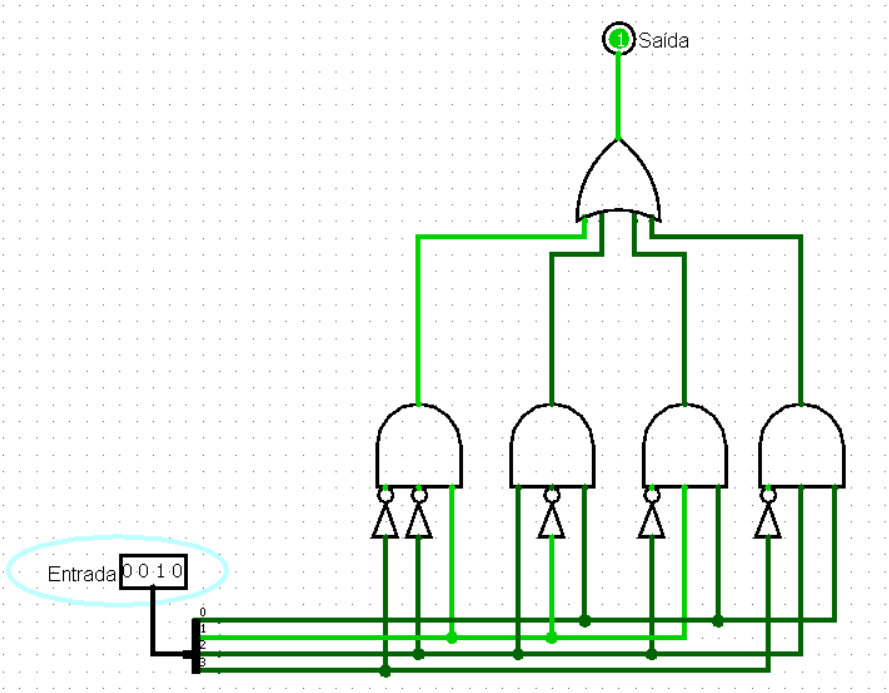
O circuito acima é capaz de detectar se o número inserido na entrada de 4 bits é primo ou não, sendo representado por 1 na saída se o número for primo e por 0 caso contrário. Utiliza um conjunto de inversores (NOT), quatro portas AND de 3 entradas cada (cada uma correspondendo a uma parte da expressão simplificada de primalidade) e uma porta OR para combinar esses termos parciais. O resultado é um circuito otimizado do ponto de vista lógico, minimizando o número de portas e garantindo o funcionamento correto conforme a especificação.

Números primos entre 0 e 15 (binário A3A2A1A0):

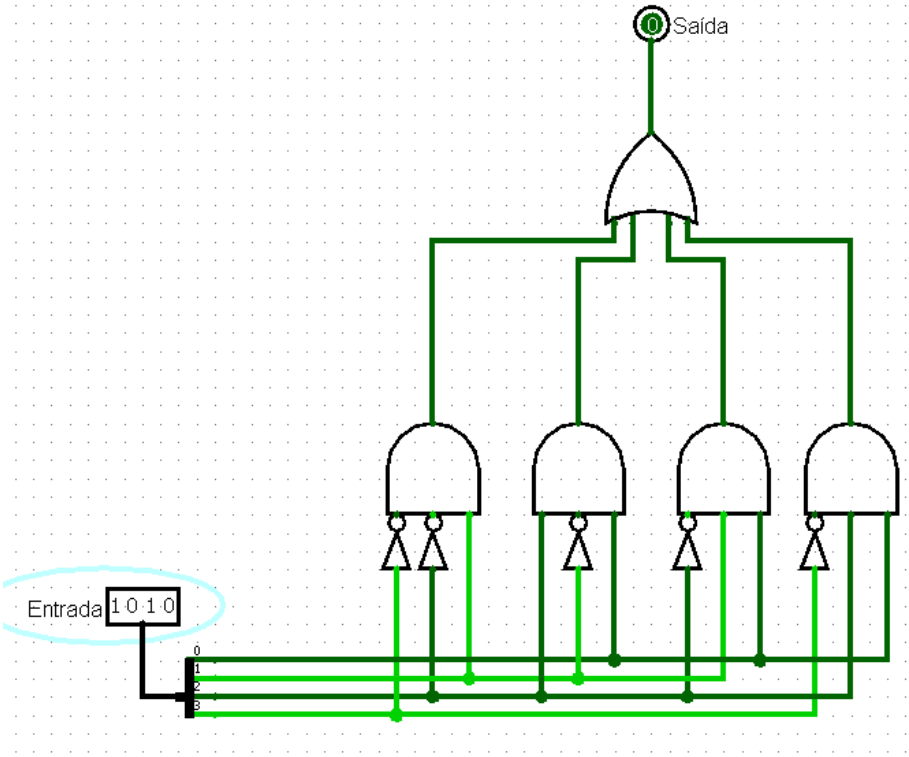
- 2 = 0010
- 3 = 0011
- 5 = 0101
- 7 = 0111
- 11 = 1011
- 13 = 1101

Essa configuração de negações e entradas diretas/negadas é fruto de uma minimização lógica da função “número-primo” para 4 bits. Com isso, em vez de simplesmente listar todos os mintermos correspondentes (o que geraria mais portas), optou-se por uma forma reduzida que agrupa os números primos em conjuntos representados por combinações mais simples de A3, A2, A1 e A0 (e suas negações).

Testes:



2 = 0010 – A saída foi 1, logo primo!



10 = 1010 – A saída foi 0, logo não é primo!

REFERÊNCIAS

BROWN, S.; VRANESIC, Z. *Fundamentals of Digital Logic with VHDL Design.* McGraw-Hill, 2009.

LOGISIM. *Logisim Official Documentation.* Disponível em: http://www.cburch.com/logisim/

Ronald Tocci - Digital systems: principles and applications 12. ed. americana. IS8N 978-85-430-2501 -3