UNIVERSIDADE FEDERAL DE RORAIMA

CENTRO DE CIÊNCIA E TECNOLOGIA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DCC301– ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES– 2024

PROF. DR. HEBERT OLIVEIRA ROCHA

KAUÃ VITOR CAVALCANTE ANSELMO

LEONAM DE SOUSA SILVA

LABORATÓRIO DE CIRCUITOS

BOA VISTA, RR

2024

1. INTRODUÇÃO

Este relatório técnico apresenta a construção de 17 componentes previamente selecionados, onde cada componente foi construído usando as portas lógicas, constantes, distribuidores e pinos de entrada e saída, toda essa implementação foi realizada no software Logisim versão 2.7.1, simulador de circuitos digitais amplamente utilizado em projetos acadêmicos e profissionais. O objetivo principal desse trabalho é a aplicação do conhecimento de circuito e documentação dos processos de descrição de cada componente, lógica aplicada e testes realizados para garantir sua funcionalidade.

1. COMPONENTES

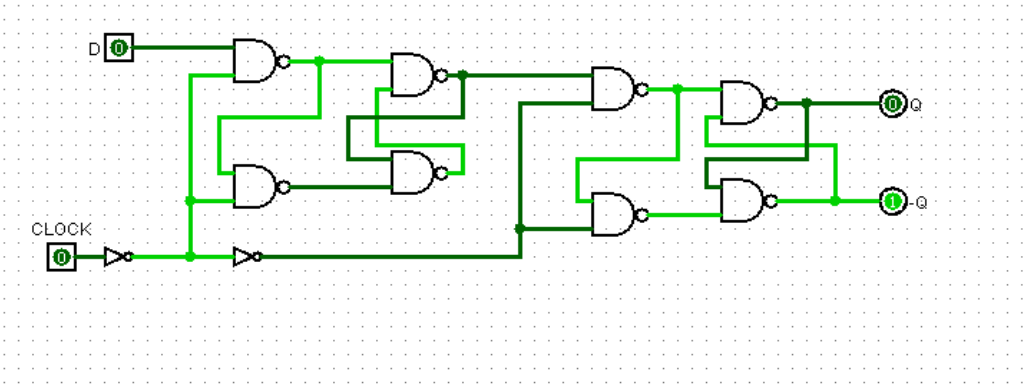
Registrador Flip-Flop do tipo D e do tipo JK

Flip-flops são circuitos digitais usados para armazenar 1 bit de informação. Eles são componentes fundamentais na construção de memórias e dispositivos sequenciais, como contadores e registradores.

Registrador Flip-Flop do tipo D

O Flip-flop D é um circuito utilizado em geral para armazenar um único bit, como se fosse um registrador de 1 bit, ele é normalmente utilizado em circuitos sequenciais e dispositivos de armazenamento como contadores, registradores e memórias. Este circuito funciona da seguinte forma, ele depende do clock, quando ocorre a borda de subida, o valor que tiver na entrada D, vai para a saída Q.

Figura 1 - Flip-Flop D



O circuito acima representa um Flip-Flop do tipo D, que é composto pelas portas lógicas NAND e NOT, possui duas entradas uma de clock(CLK) e uma de dado(D), e possui duas saídas Q e -Q, onde Q representado o estado atual armazenado no Flip-Flop e -Q o inverso.

Tabela 1 – Tabela verdade do Flip-Flop tipo D

D	CLK	Q	~Q
0	1	0	1
1	1	1	0

Nos testes feitos no Logisim o resultado foi essa tabela, onde há uma mudança de estado quando D recebe um dado e CLK recebe uma entrada alta, fazendo assim Q receber o estado atual do Flip-Flop, assim cumprindo o papel do circuito.

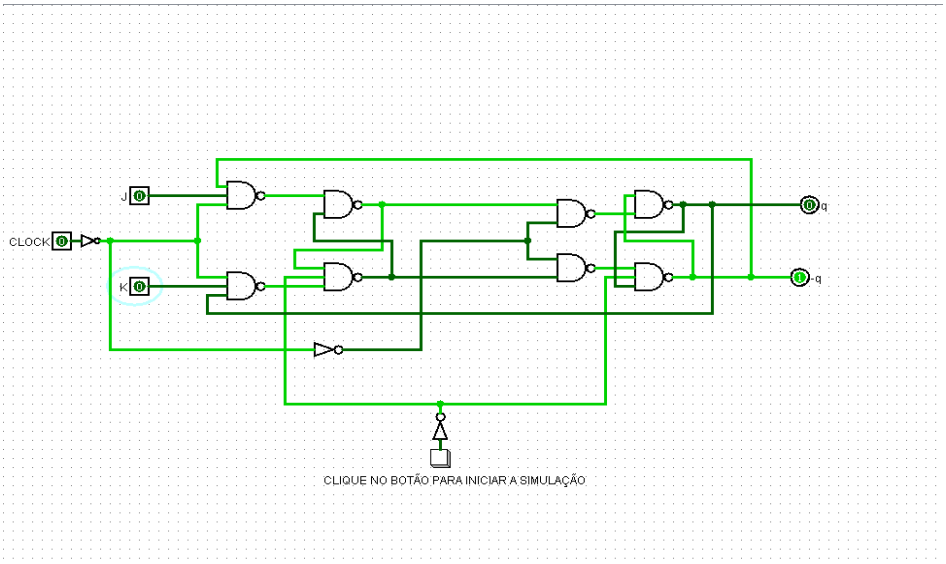


Figura 2 - Flip-Flop JK

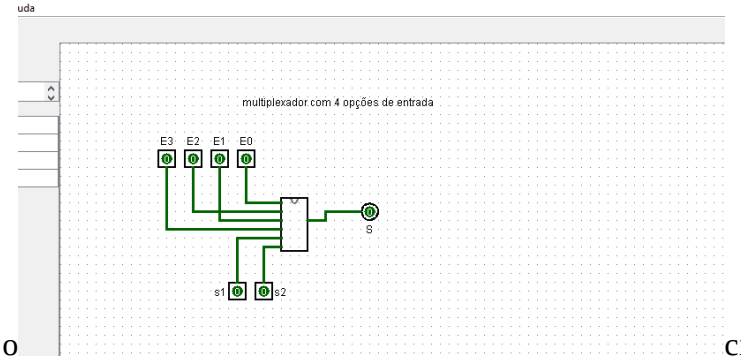
O circuito acima representa um Flip-Flop do tipo JK, composto por portas lógicas NAND e NOT contendo três pinos de entada o J, K e CLOCK para a variação de pulso , duas saídas de informação além de um botão para iniciar o circuito.

2. Multiplexador de quatro opções de entrada.

Um multiplexador com portas lógicas é um circuito combinacional que seleciona uma entrada e a conecta a uma saída única. Ele é composto por portas lógicas, principalmente portas AND. Os multiplexadores podem ser usados em aplicações digitais ou analógicas. Em aplicações digitais, são construídos a partir de portas lógicas padrão.

As principais aplicações dos multiplexadores são: Roteamento de dados, Geração de funções lógicas, Seleção de informações digitais para serem transmitidas para outro circuito, Serialização de informações de vários bits.

Figura 3 – multiplexador



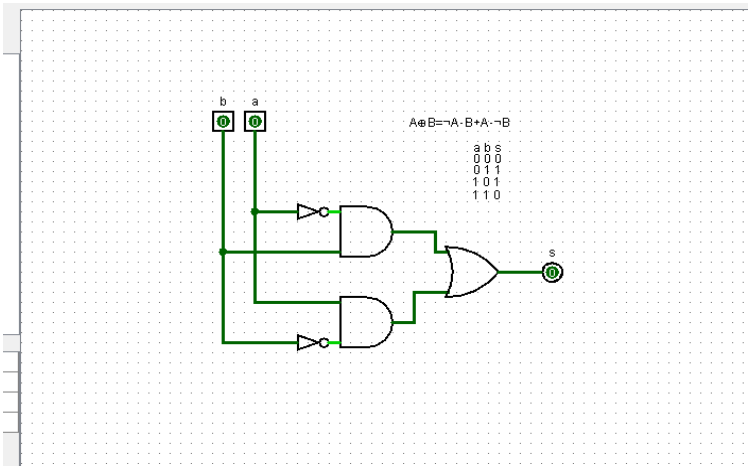
Para o circuito a cima foi utilizado 6 pinos de entrada sendo 4 para entrada de informação e 2 para controle de pulso, o registrador garanti a saída correta do dado.

3. Porta lógica XOR usando os componentes: AND, NOT, e OR

O OU exclusivo, também conhecido como disjunção exclusiva, tem como principal função verificar a igualdade entre os valores de entrada, e por isso é aplicado na criação de testadores. A saída será verdadeira se, e somente se, as duas entradas forem diferentes.

Dessa forma, o resultado será falso sempre que os valores de A e B forem iguais

Figura 4 – porta XOR



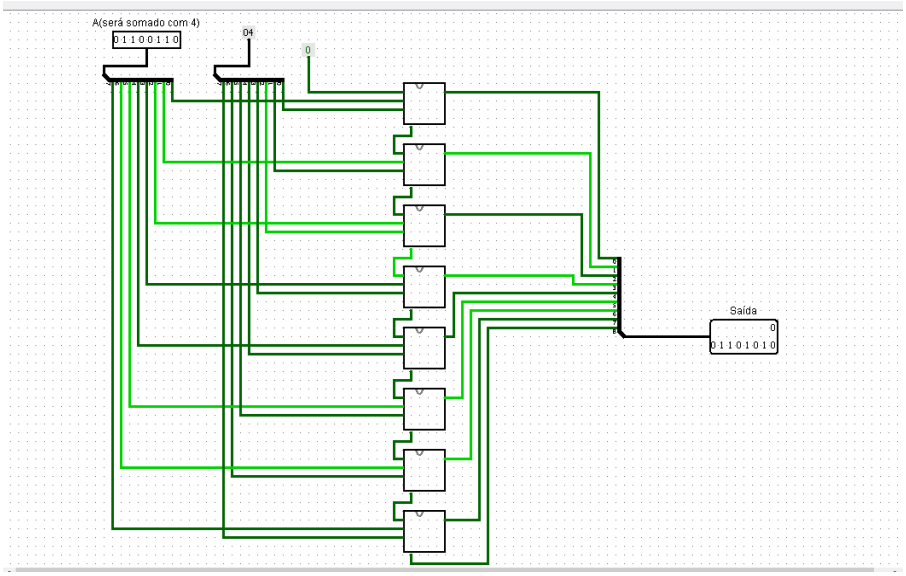
No circuito a cima foi utilizado dois pinos de entrada, duas portas NOT para inversão do sinal, duas AND para receber sinais iguais e uma Or para mandar o sinal para o pino de saida.

4. Somador de 8 bits que recebe um valor inteiro e soma com o valor 4

Um somador é um circuito lógico digital em eletrônica, que realiza a adição de dois ou mais números binários.

É usado em circuitos lógicos de computadores, processadores (unidades de ALU), e muitas outras aplicações.
Os somadores são basicamente classificados em dois tipos: Half Adder (Meio Somador) e Full Adder (Somador completo).

Figura 5 – somador de 8 bits



o circuito a cima apresenta um somador de 8 bits que recebe um valor inteiro e o soma com mais 4.

Nele foi utilizado um pino de entrada que recebe de 0 a 7 bits, duas contantes que são a quantidade de bit a ser acresentado no caso mais 4 e outra para o bit 1 feito para processar o ‘vai um’ na soma. Três barramentos estão sendo utilizados, um para receber o valor de entrada outro para receber a constante 4 e um terceiro barramento para os bits de saída.

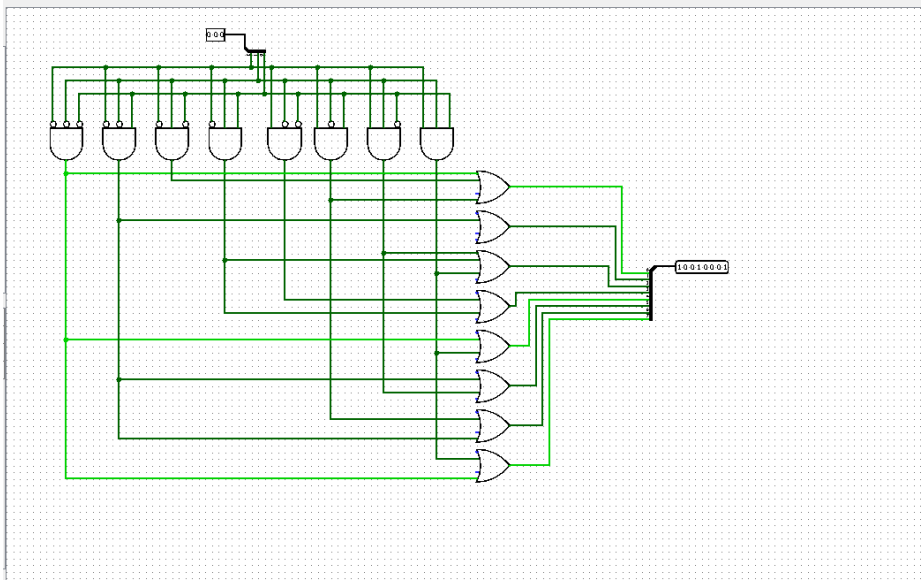
Cada bit será somado calculado por meio somador presento no circuito ele recebera a constate e cada bit a ser somado elam do ‘vai um’ quando necessário, assim gerando a soma do numero recebido somado com mais 4.

5. Memória ROM de 8 bits

A memória ROM, sigla no inglês para “memória somente de leitura”, é um tipo de memória que, como o próprio nome sugere, permite apenas a leitura de dados e não a escrita. Isso porque suas informações são gravadas pelo fabricante uma única vez e não podem ser alteradas ou apagadas, somente acessadas, sendo classificadas como memória não volátil.

Ao contrário da memória RAM, que perde dados quando a energia é removida, a memória ROM consegue armazenar firmwares ou pequenos softwares que funcionam apenas em um hardware específico.

Figura 6 – memória ROM



. Entradas e Decodificador (3 para 8)

- Entradas: Conecte os 3 bits de entrada (A2,A1,A0) a um barramento.
- Decodificação com NAND:
    - Cada porta NAND representa uma linha de decodificação.
    - Cada uma das 8 portas NAND recebe uma combinação de 2A ,A1, A0 e suas inversões (A2 Barrado, A1Barrado, A0 Barrado.
    - Para gerar as inversões, conecte portas NOT às entradas.

Cada porta NAND ativa (saída baixa) somente para seu endereço correspondente.

. Conexão com as Portas XOR

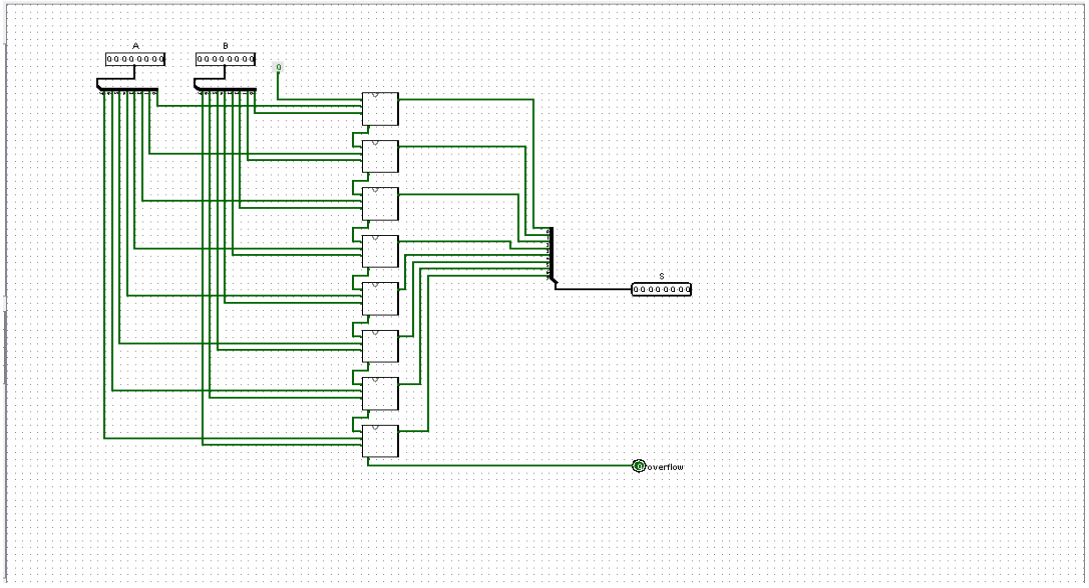
- Cada uma das 8 linhas de saída do decodificador (portas NAND) será conectada às portas XOR.
- Cada linha do decodificador controla uma porta XOR correspondente.
- As portas XOR permitem configurar a saída de 8 bits de dados para cada endereço:
    - Conecte as outras entradas das portas XOR a valores lógicos (0 ou 1) para programar os bits de saída.

. Barramento de Saída

- As saídas das 8 portas XOR (representando os 8 bits de dados) são combinadas em um barramento de 8 bits.
- O barramento fornece o valor de saída da ROM.

8. Somador de 8 bits

Figura 9 – somador de 8 bits



O somador de 8 bits descrito é um circuito combinacional que soma dois números de 8 bits e inclui:

- Entradas:
    - Dois barramentos de 8 bits (A e B).
    - Um bit de entrada de Carry In (Vai 1), que permite somar com um valor de transporte vindo de uma operação anterior.
- Componentes principais:
    - 8 somadores completos (Full Adders): Cada somador realiza a soma de um bit correspondente dos barramentos A e B, considerando o bit de transporte (Carry) da posição anterior.
    - Overflow: Detecta se houve um transbordo ao somar os dois números.
- Saídas:
    - Um barramento de 8 bits para o resultado.
    - Um bit de Overflow, indicando se o resultado excedeu a capacidade de representação de 8 bits.

Componentes do Circuito

1. Entradas:
    - Barramento A (8 bits): Primeiro número a ser somado.
    - Barramento B (8 bits): Segundo número a ser somado.
    - Carry In (Cin, 1 bit): Entrada para transporte.
2. Full Adders:
    - Cada somador realiza: $S=A \oplus B \oplus Cin$

- Cout = (A . B) + (Cin . A⊕B))
- S: Soma do bit atual.
- Cout: Carry para o próximo estágio.

3. Overflow:
    - Detecta se ocorreu um transbordo:
    - Overflow = $Cn-1 \oplus Cn$
    - Onde $Cn-1$é o transporte do penúltimo bit e Cn é o transporte do último bit.
4. Saídas:
    - Barramento de 8 bits para o resultado.
    - Bit de Overflow.

Funcionamento do Circuito

1. Soma Bit a Bit

- O somador de 8 bits utiliza 8 somadores completos conectados em cascata:
    - O Carry Out de cada somador é conectado ao Carry In do próximo somador.
    - O primeiro somador recebe o Carry In externo (Cin).

2. Transporte Entre Bits

- Cada somador propaga um bit de transporte ao próximo estágio.

3. Detectando Overflow

- O overflow ocorre se o transporte gerado no último bit (Cn) não é igual ao transporte no penúltimo bit ($Cn-1$).

4. Saída Final

- O barramento de saída coleta os bits de soma (S0 a S7).
- O bit de overflow é emitido separadamente para indicar um possível erro de transbordo.

Estrutura Lógica

1. Entradas

- A= [A7,A6,…,A0]: Primeiro número.
- B = [B7,B6,…,B0]: Segundo número.
- Cin: Transporte inicial.

2. Somadores Completos

- O somador calcula:
    - Soma (Si) usando os bits correspondentes de Ai, Bi e Cin.
    - Transporte (Cout) para o próximo somador.

3. Overflow

- Cout do último somador é comparado com $Cn-1$.

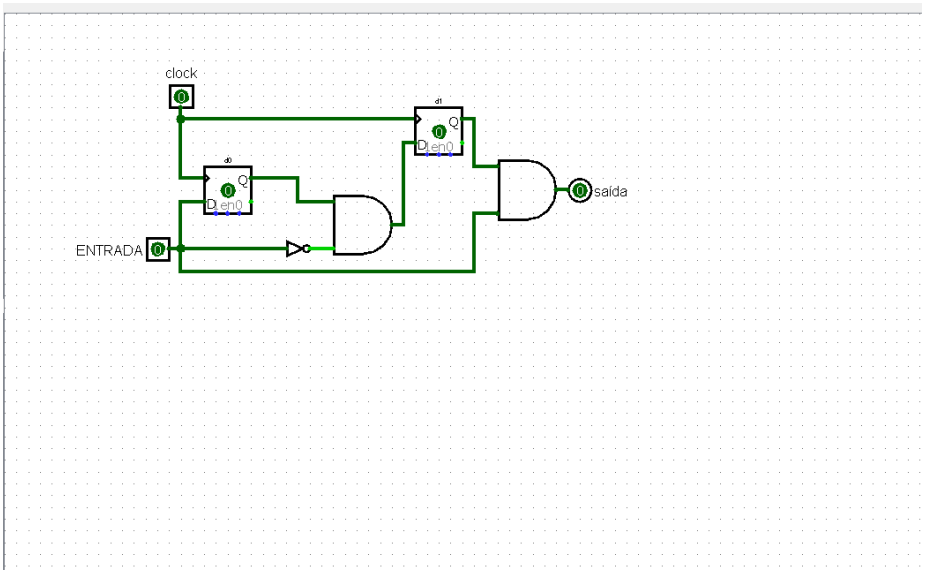
4. Saída

- S=[S7,S6,…,S0]: Resultado da soma.
- Overflow: Cn−1⊕Cn.

9 . Construa um detector de sequência binária para identificar a sequência "101" em um fluxo de entrada.

Um detector de sequência binária é um circuito digital projetado para identificar uma sequência específica de bits em um fluxo de entrada binário contínuo. Ele verifica os padrões de entrada e sinaliza quando a sequência desejada é detectada.

Figura 10 – detector de sequência binaria



1. Entradas

- O pino de entrada fornece o fluxo binário.
- O clock sincroniza o avanço dos estados.

2. Flip-Flops (Armazenam o estado)

- Q1 (FF1): Indica se estamos no estado S1.
- Q2 (FF2): Indica se estamos no estado S2.

3. Lógica de Transição

- Porta NOT:
    - Inverta a entrada para usar o valor "0" em transições específicas.
- Porta NAND 1:
    - Gera o próximo estado de Q1 (D1):

D1 = Entrada· Q2 Barrado

- Define S1 quando "1" é detectado e o circuito não está em S2.

- Porta NAND 2:
    - Gera o próximo estado de Q2 (D2): D2=Q1· Entrada Barrada
        - Define S2 quando "0" é detectado após S1.

4. Saída

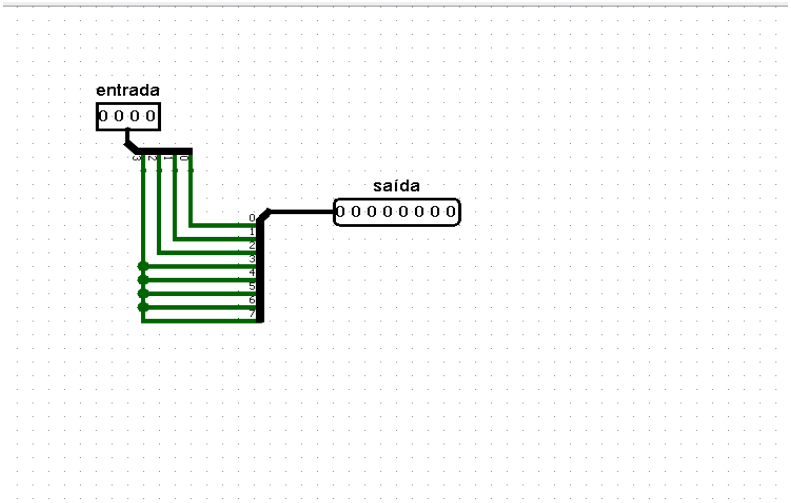
- A saída é ativada quando a sequência "101" é detectada: Saída = Q2· Entrada
- Isso ocorre quando o circuito está em S2 e recebe um "1".

11. Extensor de sinal de 4 bits para 8 bits.

Um extensor de sinal, também conhecido como repetidor de sinal, é um dispositivo que amplia o sinal de uma rede sem fio, permitindo que a conexão seja acessada em locais mais distantes do roteador.

O repetidor recebe o sinal do roteador e o retransmite, ampliando a área de cobertura da rede. Ele é uma opção popular para ambientes domésticos, pois é fácil de instalar e usa

Figura 12 – extensor de sinal



Com uma entrada de 4 bits em barramento o circuito de liga em um barramento de saída de 8 bits implementando a seguinte lógica:

Se o número de 4 bits for positivo (quando o bit mais significativo, ou MSB, for 0), a extensão de sinal é feita adicionando zeros à esquerda.

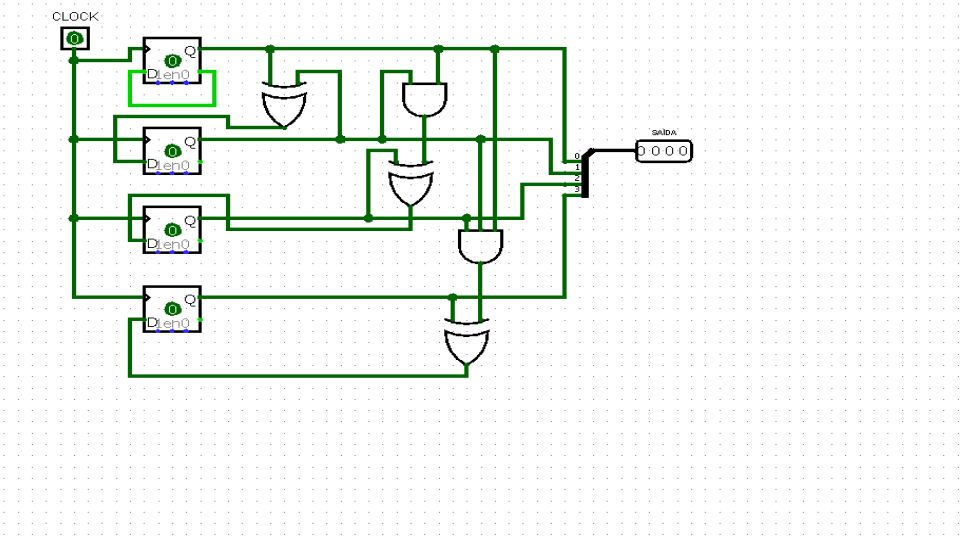
- Se o número de 4 bits for negativo (quando o MSB for 1), a extensão de sinal é feita adicionando uns à esquerda.

13. Contador Síncrono.

Os contadores digitais são dispositivos eletrônicos que realizam contagens com base em um sinal. Eles são utilizados para: Contagens, Geração de palavras, Divisão de frequências, Medição de frequência e tempo.

Os contadores podem ser divididos em síncronos e assíncronos. Nos contadores assíncronos, os FFs não recebem o mesmo sinal de clock.

Figura 13 – contador síncrono



Entradas:

- Clock: Controla a sincronização de todos os flip-flops (eles mudam de estado a cada borda de clock).
- Entrada de Reset ou Clear: (Opcional) Pode ser usado para zerar o contador, configurando todos os bits de saída para 0.

2. Flip-Flops Tipo D:

- Cada flip-flop armazena um bit do contador.
- O estado de cada flip-flop será atualizado a cada borda de clock.

3. Portas XOR e NAND:

- Usaremos as portas XOR e NAND para modificar as entradas de controle dos flip-flops para garantir a contagem binária adequada.
- Portas XOR: Elas podem ser usadas para criar lógicas de alternância ou transições condicionais nos flip-flops.
- Portas NAND: Elas podem ser usadas para formar combinações lógicas entre os bits armazenados nos flip-flops, gerando as condições necessárias para transições no contador.

4. Multiplexação e Controle do Contador:

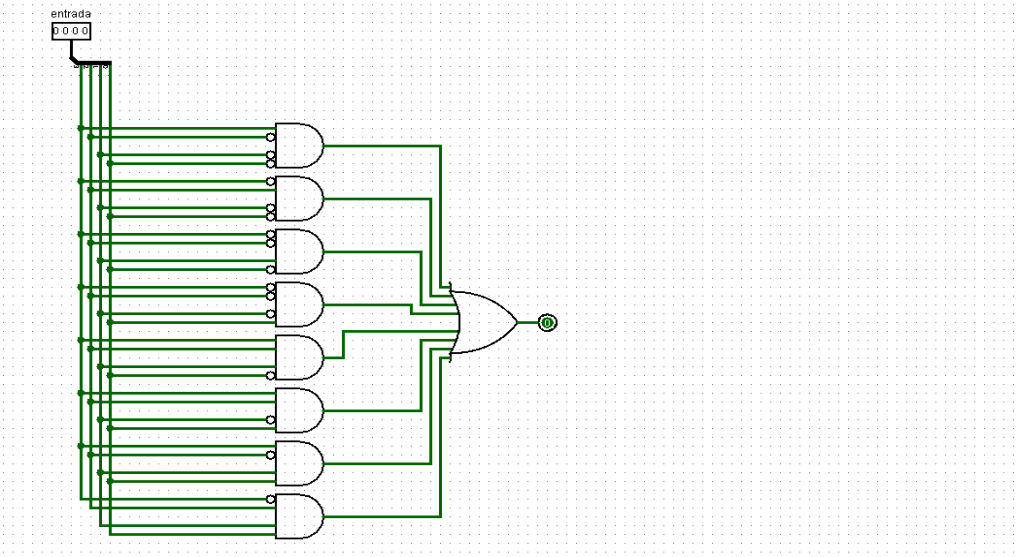
- Como estamos trabalhando com um contador síncrono, os flip-flops serão alimentados com sinais de controle definidos pelas portas XOR e NAND.
- Cada flip-flop será conectado a uma das saídas das portas XOR e/ou NAND para controlar quando ele deve alternar seu estado.

5. Saída de 4 bits:

- A saída será formada pelos 4 bits armazenados nos flip-flops Q3,Q2,Q1,Q0
- Esses bits indicam o valor atual do contador, e a cada pulso de clock o contador aumentará seu valor em 1 (em formato binário).

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

Figura 14 - detector de paridade ímpar



Um detector de paridade ímpar é um circuito que determina se o número de bits com valor 1 em uma entrada binária é ímpar. Ele pode ser implementado usando portas lógicas (AND, OR, e NOT) para processar a entrada e calcular a paridade.

A lógica básica de um detector de paridade ímpar é baseada na operação XOR, que retorna 1 se o número de entradas com valor 1 for ímpar. Como o XOR pode ser implementado usando combinações de AND, OR e NOT, aqui está como fazer isso:

1. Lógica XOR com AND, OR e NOT

Para duas entradas A e B, a operação XOR pode ser expressa como:

$A \oplus B = (\text{NOT } A \wedge B) \vee (A \wedge \text{NOT } B)$

Ou seja:

1. NOT A: Inverta A
2. NOT B: Inverta B
3. NOT A∧ B: Multiplique NOT A e B
4. NOT A∧ B: Multiplique A e NOT B
5. (NOT A∧ B)∨ (A∧ NOT B): Combine os dois termos usando OR.

Para entradas maiores que 2 bits, encadeamos múltiplas operações XOR.

2. Detector de Paridade Ímpar

Para N entradas (X1,X2,...,XN):

1. Calcule X1⊕X2⊕..⊕XN usando a implementação de XOR com AND, OR e NOT.
2. O resultado será 1 se o número de 1s na entrada for ímpar.

3. Exemplo com 3 Entradas (A,B,C)

1. Primeiro XOR (A⊕B):
    - A⊕B = (NOT A∧ B)∨ (A∧ NOT B)
2. Segundo XOR ((A⊕B)⊕C):
    - Primeiro, inverta A⊕B e C:

(NOT (A⊕B)∧ C)∨ ((A⊕B)∧ NOT C)

3. Saída:
    - O resultado é 1 se o número de 1s em A,B,C for ímpar.

4. Diagrama Lógico

Aqui está uma explicação do circuito passo a passo:

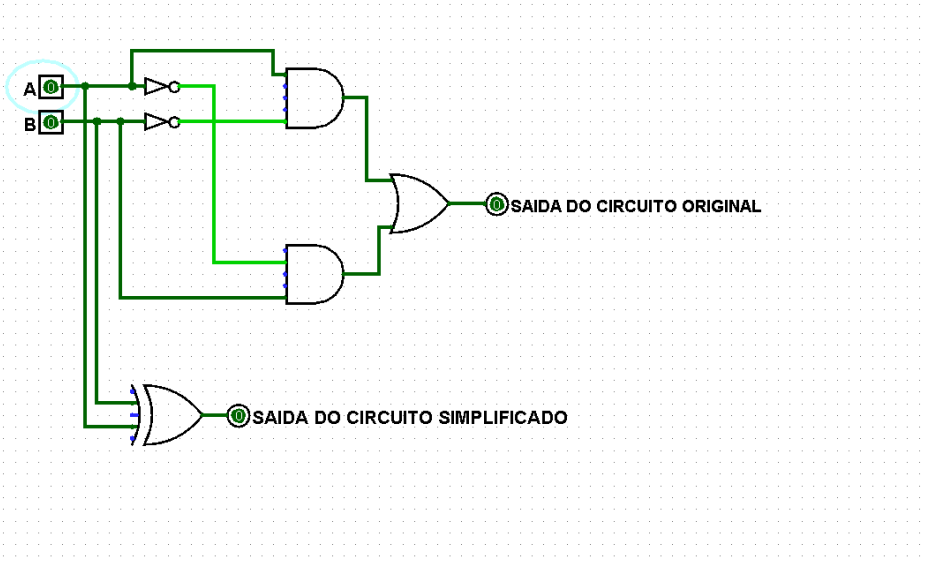
1. Use portas NOT para inverter os sinais necessários.
2. Use portas AND para calcular os termos parciais.
3. Use portas OR para combinar os resultados.

Se desejar, posso criar um diagrama ou detalhar como conectar os componentes no circuito físico.

4o

15. Resolva um problema de otimização lógica utilizando mapas de Karnaugh e implemente o circuito otimizado.

Figura 15 – circuito de Karnaugh



1. Circuito Original

Primeiro, definimos o circuito original com as portas disponíveis:

- Duas portas NOT: Para inverter as entradas A e B.
- Duas portas AND: Para combinar as entradas e suas negações.
- Uma porta OR: Para combinar os resultados das portas AND.

A função lógica do circuito original será algo como:

F original = (NOT A∧ B)∨ (A∧ NOT B)

Essa expressão é uma implementação de A⊕B (XOR), mas construída com AND, OR e NOT.

2. Mapas de Karnaugh

Um Mapa de Karnaugh simplifica expressões lógicas. Para duas variáveis A e B, temos:

A\BA \backslash BA\B	0	1
0	0	1
1	1	0

O mapa acima representa A⊕B: a saída é 1 quando A e B são diferentes.

3. Expressão Simplificada

A partir do Mapa de Karnaugh:

F simplificada = A⊕B

4. Implementação do Circuito

Circuito Original:

Usando NOT,AND,OR:

1. Inverta A e B usando portas NOT.
2. Calcule NOT A∧ B e A∧ NOT B com duas portas AND.
3. Combine os resultados das portas AND com uma porta OR.

Circuito Simplificado:

Use diretamente uma porta XOR para implementar A⊕BA \oplus BA⊕B.

5. Comparação de Circuitos

- Circuito Original:
    - Usa 2 NOT, 2 AND, 1 OR = 5 portas.
- Circuito Simplificado:
    - Usa 1 XOR = 1 porta.

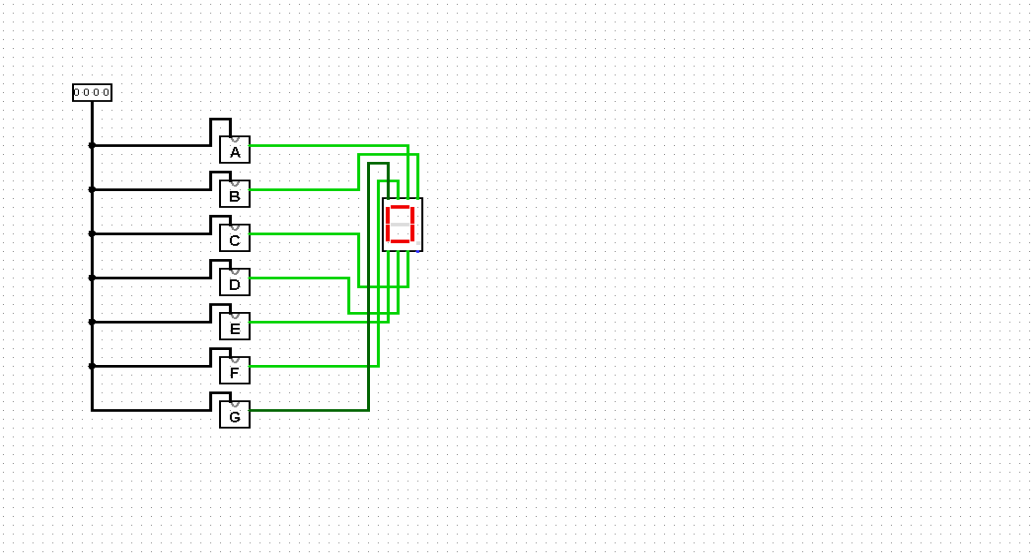
6. Diagrama Lógico

Circuito Original:

1. A passa por uma porta NOT para gerar NOT A.
2. B passa por uma porta NOT para gerar NOT B.
3. NOT A∧ B e A∧ NOT B são calculados usando duas portas AND.
4. O resultado das ANDs é combinado com uma porta OR.

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).

Figura 16 – decodificador de 7 segmentos



1. Entradas e Saídas

1. Entradas: B3,B2,B1,B0.

- Entradas binárias de 4 bits, representando números de 0 a 15 (hexadecimal 0x0 a 0xF).

2. Saídas: a,b,c,d,e,f,g.

- Cada saída controla um segmento do display, sendo 1 (ligado) ou 0 (desligado).

2. Tabela Verdade

A tabela verdade relaciona as entradas binárias com os segmentos ativados para formar os dígitos no display:

Entrada (B3,B2,B1,B0)	a	b	c	d	e	f	g
0000 (0x0)	1	1	1	1	1	1	0
0001 (0x1)	0	1	1	0	0	0	0
0010 (0x2)	1	1	0	1	1	0	1
0011 (0x3)	1	1	1	1	0	0	1
0100 (0x4)	0	1	1	0	0	1	1
0101 (0x5)	1	0	1	1	0	1	1
0110 (0x6)	1	0	1	1	1	1	1
0111 (0x7)	1	1	1	0	0	0	0
1000 (0x8)	1	1	1	1	1	1	1
1001 (0x9)	1	1	1	1	0	1	1
1010 (0xA)	1	1	1	0	1	1	1
1011 (0xB)	0	0	1	1	1	1	1
1100 (0xC)	1	0	0	1	1	1	0
1101 (0xD)	0	1	1	1	1	0	1
1110 (0xE)	1	0	0	1	1	1	1
1111 (0xF)	1	0	0	0	1	1	1

3. Mapas de Karnaugh para Cada Segmento

Para cada segmento (a a g), construa Mapas de Karnaugh baseados na tabela verdade para minimizar as expressões lógicas.

4. Expressões Lógicas Simplificadas

As expressões lógicas minimizadas (após análise dos Mapas de Karnaugh) para cada segmento são:

1. Segmento a:

$a = B3^{-} \cdot B2^{-} \cdot B1 \cdot B0 + B3 \cdot B2 \cdot B1^{-} \cdot B0 + ...$

(Expanda conforme o Mapa de Karnaugh.)

2. Segmento b: Similarmente derive b com Mapas de Karnaugh.

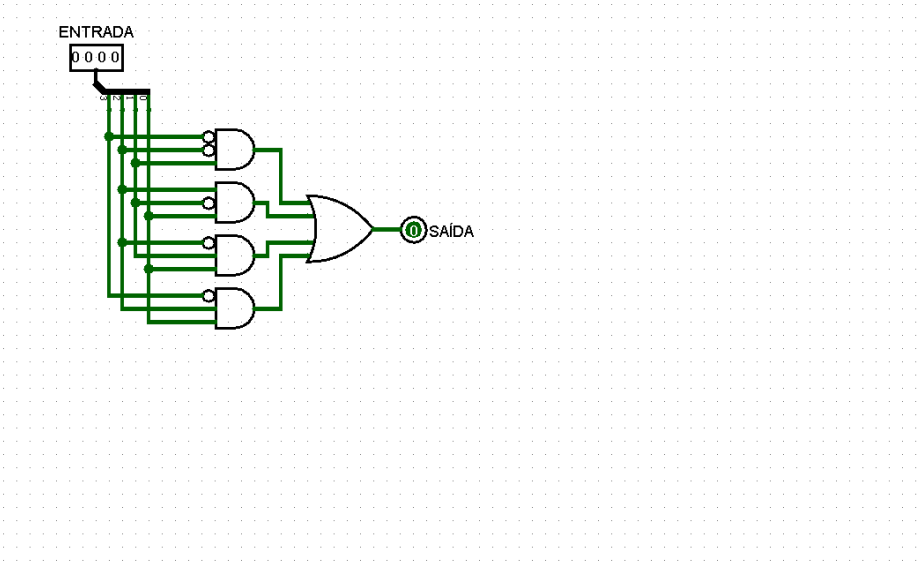
Repita para os demais segmentos (c,d,e,f,g).

5. Implementação

1. Entrada: Conecte B3,B2,B1,B0 aos 7 circuitos lógicos.
2. Circuitos: Cada segmento a,b,c,d,e,f,g será controlado por um circuito lógico minimizado conforme as expressões.
3. Saída: Os sinais a,b,c,d,e,f,g acionam diretamente o display.

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.

Figura 17 – detector de número primo



. Entradas e Saída

1. Entradas: B3,B2,B1,B0 (4 bits, números de 0 a 15).
2. Saída: P, onde:

- P=1: A entrada representa um número primo.
- P=0: A entrada não representa um número primo.

2. Números Primos no Intervalo de 0 a 15

Os números primos são aqueles divisíveis apenas por 1 e por si mesmos. No intervalo de 0 a 15, os números primos são:

Primos={2,3,5,7,11,13}

3. Tabela Verdade

A tabela verdade relaciona as entradas (B3,B2,B1,B0) com a saída P:

Entrada (B3,B2,B1,B0)	Decimal	P (Primo)
0000	0	0
0001	1	0
0010	2	1
0011	3	1
0100	4	0
0101	5	1
0110	6	0
0111	7	1
1000	8	0
1001	9	0
1010	10	0
1011	11	1
1100	12	0
1101	13	1
1110	14	0
1111	15	0

4. Mapa de Karnaugh

Representação para a saída P:

Organizamos os valores de P no Mapa de Karnaugh:

B3B2\B1B0	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	1	0	0	0
10	0	1	0	0

5. Expressão Lógica Simplificada

A partir do Mapa de Karnaugh, identificamos grupos de 1s para simplificar a função P:

$P = B3^{-} \cdot B2^{-} \cdot B1 \cdot B0 \vee B3^{-} \cdot B2^{-} \cdot B1 \cdot B0^{-} \vee B3^{-} \cdot B2 \cdot B1^{-} \cdot B0 \vee \ldots$

Simplificando os grupos:

$P = (B3^{-} \cdot B2^{-} \cdot B1) \vee (B3^{-} \cdot B2 \cdot B1^{-} \cdot B0) \vee (B3 \cdot B2^{-} \cdot B1^{-} \cdot B0)$

6. Implementação com Portas Lógicas

A expressão lógica simplificada pode ser implementada com:

- Portas AND: Para calcular os produtos lógicos ($\cdot$).
- Portas OR: Para combinar os produtos lógicos ($\vee$).
- Portas NOT: Para inverter os bits conforme necessário.