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

RYAN PIMENTEL DE OLIVEIRA

CLEILLYSON OSMAR SOUZA DINIZ DE ALMEIDA

LABORATÓRIO DE CIRCUITOS

BOA VISTA, RR

2024

RYAN PIMENTEL DE OLIVEIRA

CLEILLYSON OSMAR SOUZA DINIZ DE ALMEIDA

LABORATÓRIO DE CIRCUITOS

Trabalho da disciplina de Arquitetura e Organização de Computadores do ano de 2024 apresentado à Universidade Federal de Roraima do curso de Bacharelado em ciência da computação.

Docente: Prof. Dr. Hebert O. Rocha

BOA VISTA, RR

2024

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FF	*Flip-Flop*
RAM	*Random Access Memory*
ROM	*Read-Only Memory*
ULA	Unidade Lógica e Aritmética

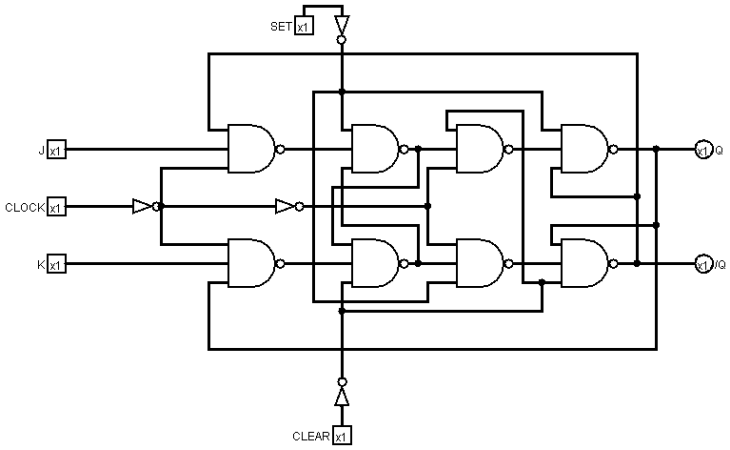
1 Componentes

Nesta seção será apresentado todos os componentes que foram construídos junto de suas respectivas descrições, funcionalidades, também serão apresentados os circuitos feitos no *Logisim* e alguns testes.

1.1 Registrador *Flip-Flop* do tipo D e do tipo JK

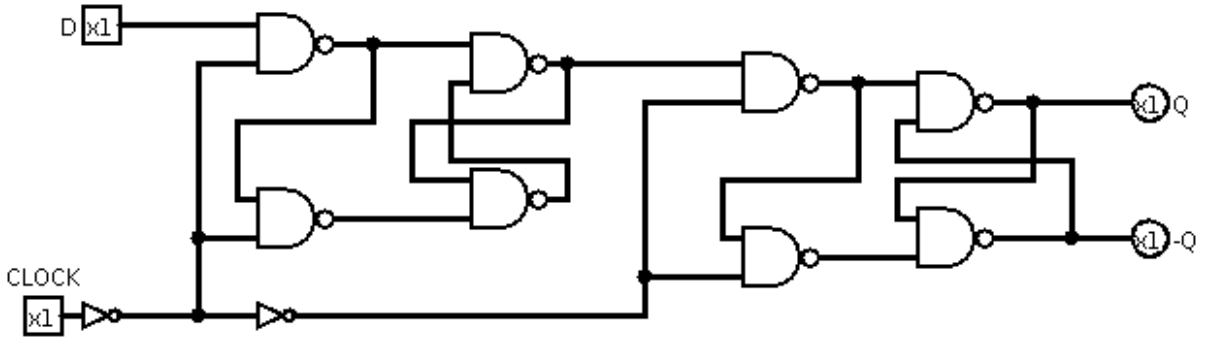
O FF JK junto do FF D é usado para armazenar um valor logico, foi construído primeiro o FF JK com 5 entradas sendo elas: J, K, *Clock*, *Set* e *Clear*. O *Set* e *Clear* Foram adicionados para melhorar a confiabilidade do componente.

Figura 1 - Flip-Flop JK



O FF D foi feito usando dois circuitos SR combinados em uma configuração de mestre e escravo, o primeiro circuito é o “mestre” e o segundo o “escravo” e apresenta duas entradas a D e o *clock* e duas saídas Q e Q/

Figura 2 - Flip-Flop D



1.2 Multiplexador

O multiplexador apresenta 4 entradas de dados, 2 de controle e uma saída, as entradas de controle(s1s0) controlam qual vai ser a saída que será exibida.

Figura 3 - Multiplexador de 4 opções de entrada

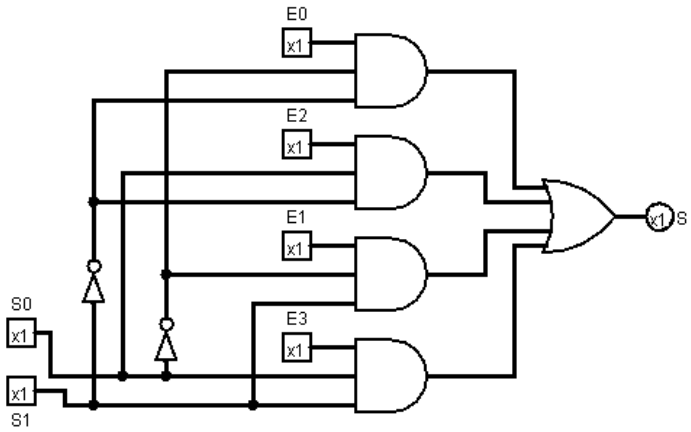


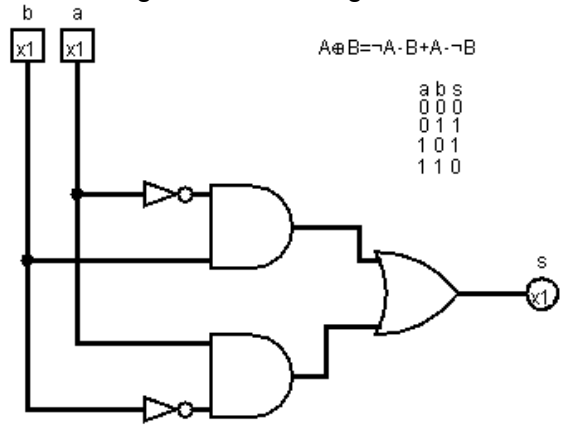
Tabela 1- Tabela verdade multiplexador

Controle		Entrada ativa
S0	**S1**	**S**
0	**0**	**E0**
0	**1**	**E1**
1	**0**	**E2**
1	**1**	**E3**

1.3 Xor

Essa porta logica apesar de simples é de extrema importância para simplificar vários circuitos, é o “ou exclusivo” onde a saída é 1 quando apenas uma das entradas é 1, mas não as duas ao mesmo tempo.

Figura 4 - Porta logica XOR



1.4 Somador de 8 *bits* e soma mais 4

O somador tem 2 partes fundamentais: o somador geral e o somador completo, o somador completo tem 3 entradas: A, B são os números originais e *carry in* que é o *carry out* do *bit* anterior, as três entradas são somadas. O componente tem 2 saídas: S que é o resultado da soma que fica e o *carry out* que vai ser somado com o próximo *bit*.

Figura 5 - Somador de 1 *bit*

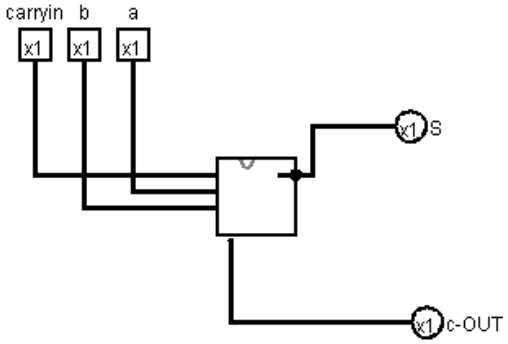
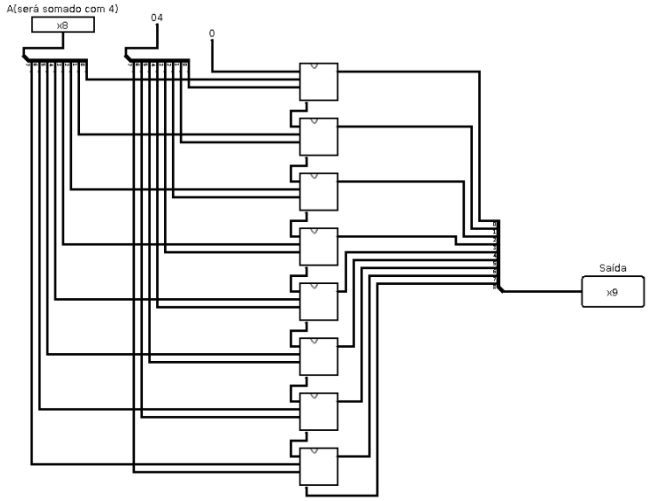


Tabela 2 - Tabela verdade somador 1 *bit*

A	B	carryin	S	c-OUT
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

A segunda parte o somador de 8 *bits* usa 8 somadores de *bit* a *bit*, esse componente tem uma entrada de 8 *bits*, pois o número somado será o 4, e uma saída de 9 *bits* em que o bit mais significativo é o *overflow*.

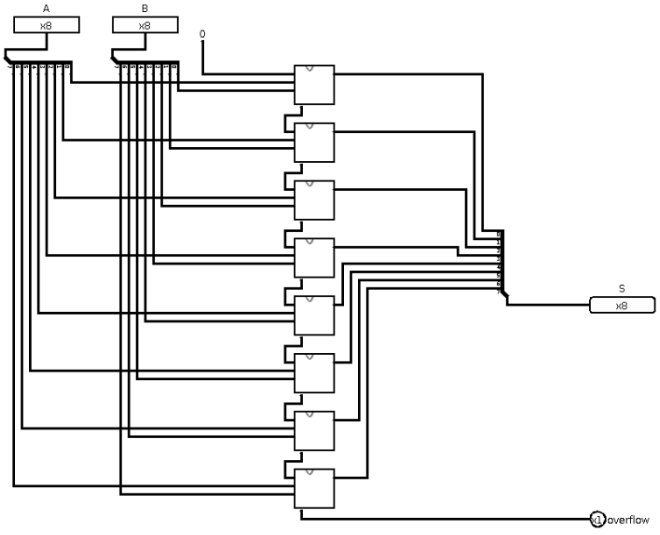
Figura 6 - Somador de 8 *bits* mais 4



1.5 Somador de 8 *bits*

O somador de 8 *bits* é idêntico ao somador de 8 *bits* anterior com exceção de que ele tem duas entradas de 8 *bits* ao invés de apenas uma e apresenta duas saídas uma de 8 *bits* e outra de 1 *bit* para *overflow*.

Figura 7 - Somador de 8 *bits*



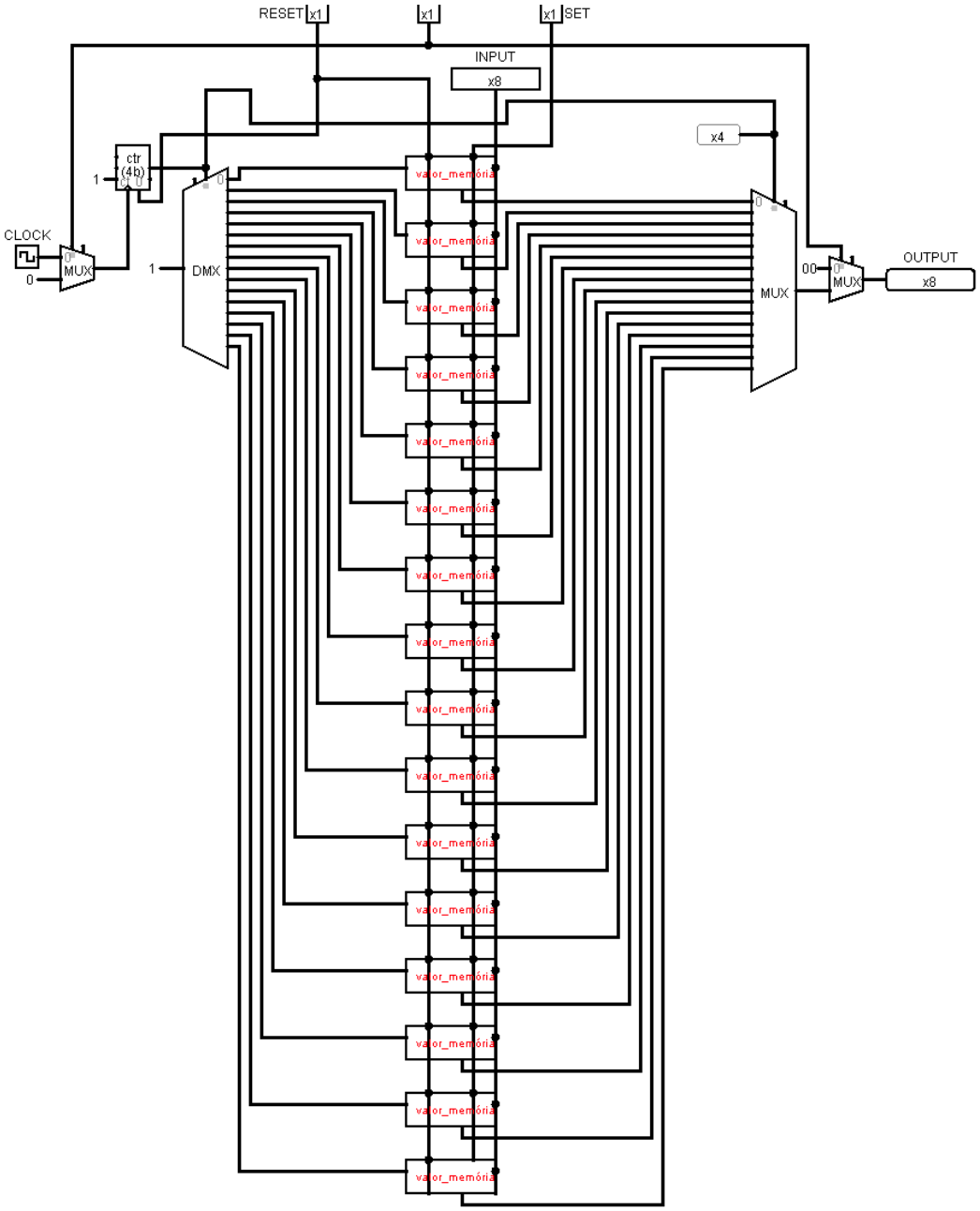
1.6 Memória *ROM*

A Memória ROM, diferentemente da RAM, armazena valores de forma sequencial, sem exigir a entrada de um endereço específico para salvar os dados. Por

isso, ela funciona como um "registro" das instruções realizadas, onde o valor de saída sempre corresponde ao último valor armazenado.

Nesta ROM, o endereço é controlado por um contador que é atualizado por um *clock*, passando por um multiplexador. Quando a chave de leitura/escrita está em 1, o *clock* não atualiza o contador, permitindo que o valor no endereço atual seja lido na saída *output*. Para escrever um valor, é necessário inseri-lo na entrada *input* e ajustar a chave de leitura/escrita para 0, esta ROM possui 16 espaços de armazenamento, cada um com 8 bits.

Figura 8 - Memória ROM



1.7 Memória *RAM* de 8 *bits*

A Memória RAM é um tipo de memória que armazena dados de forma não sequencial, sendo o endereço de memória responsável por definir o local exato de armazenamento. A RAM projetada conta com 8 espaços de armazenamento, cada um capaz de guardar um valor de 8 bits. O seletor é conectado a um demultiplexador como entrada de seleção (*select*), enquanto a entrada do demultiplexador é o *clock* principal. Uma das oito saídas do demultiplexador atua como o *clock* para o componente "palavra", que é responsável por armazenar 8 bits de dados através de oito FF do tipo D. O componente “palavra” possui uma entrada de 8 *bits*, duas de apenas um *bit* o *clock e* o *clear*. Esse componente possui apenas uma saída de 8 *bits*.

Figura 9 - Componente "palavra"

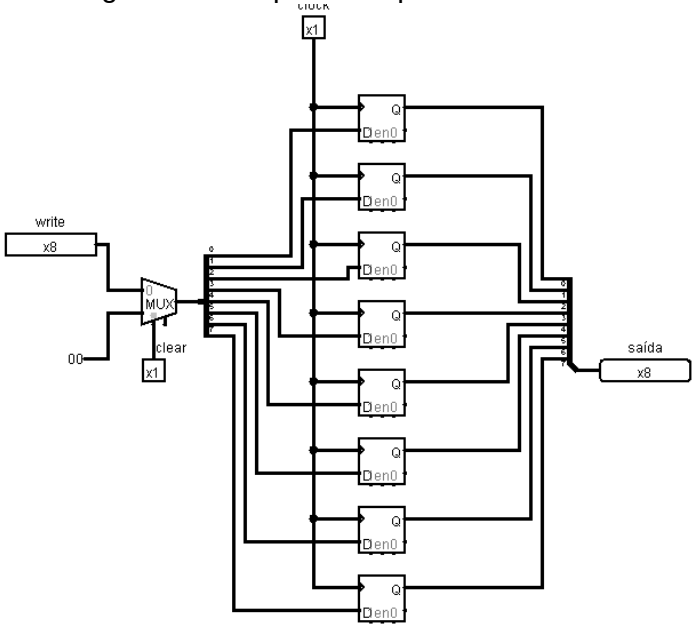
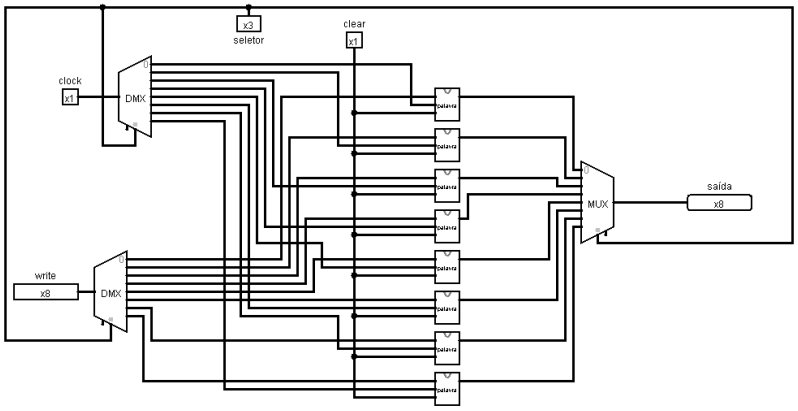


Figura 10 - Memória RAM



1.8 Banco de registradores de 8 *bits*

Um banco de registradores é um componente digital formado por um conjunto de registradores organizados para acesso organizado. Ele permite a realização de operações de leitura, para recuperar dados previamente armazenados, e de escrita, para atualizar ou modificar as informações internas.

O nosso banco de registradores possui 4 entradas:

- *data_in* (8 *bits*): Representa o valor a ser escrito no registrador selecionado.
- *write_select* (4 *bits*): Controla qual dos 16 registradores receberá o valor de *data_in*. Ele está conectado a um demultiplexador que direciona o sinal de escrita para o registrador correspondente.
- *rx_select* (4 *bits*): Determina qual registrador será lido para a saída *rx*.
- *ry_select* (4 *bits*): Determina qual registrador será lido para a saída *ry*.

E apresenta duas saídas:

- *rx* (8 *bits*): Fornece o valor armazenado no registrador selecionado por *rx_select*.
- *ry* (8 *bits*): Fornece o valor armazenado no registrador selecionado por *ry_select*.

Sua arquitetura interna possui 16 registradores de 8 *bits* cada um com entradas para dados, a entrada *write_select* é conectada a um demultiplexador que direciona o sinal de escrita para o registrador apropriado. O valor em *data_in* é armazenado no registrador selecionado, por fim dois multiplexadores para Leitura um é controlado pelo *rx_select* que conecta o registrador a saída *rx* e o *ry_select* que faz o mesmo, mas para a saída *ry*.

Existem duas operações principais: a escrita e a leitura. Na operação de escrita, o valor presente em *data_in* é armazenado no registrador indicado pelo sinal *write_select*. Apenas o registrador especificado é atualizado com o novo valor,

enquanto os demais registradores mantêm seus conteúdos inalterados. E na de leitura, os registradores a serem lidos para *rx* e *ry* são determinados por *rx_select* e *ry_select*, respectivamente. Cada seletor escolhe um registrador cuja saída será enviada ao barramento correspondente.

Figura 11 - Registradores

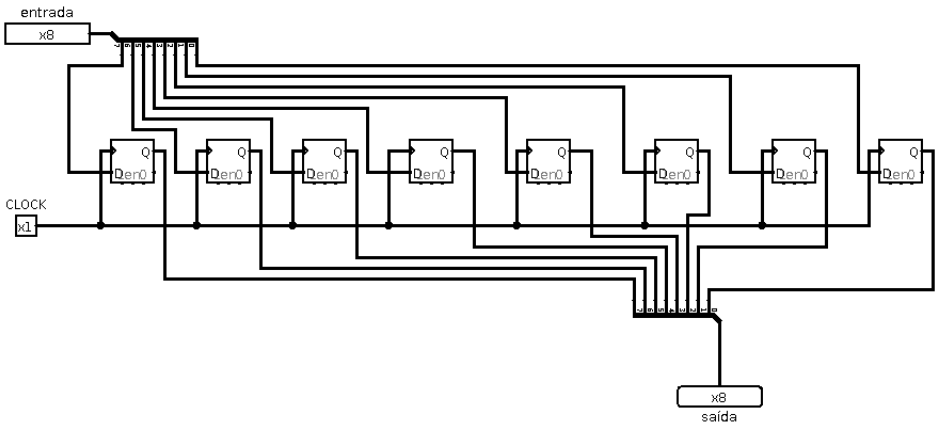
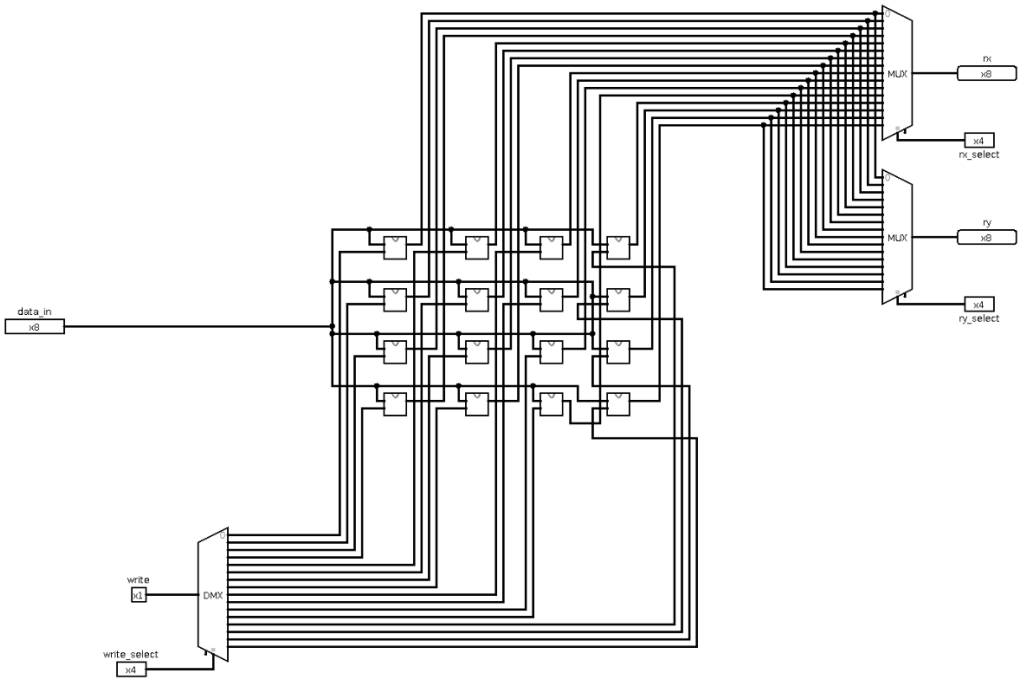


Figura 12 - Banco de registradores

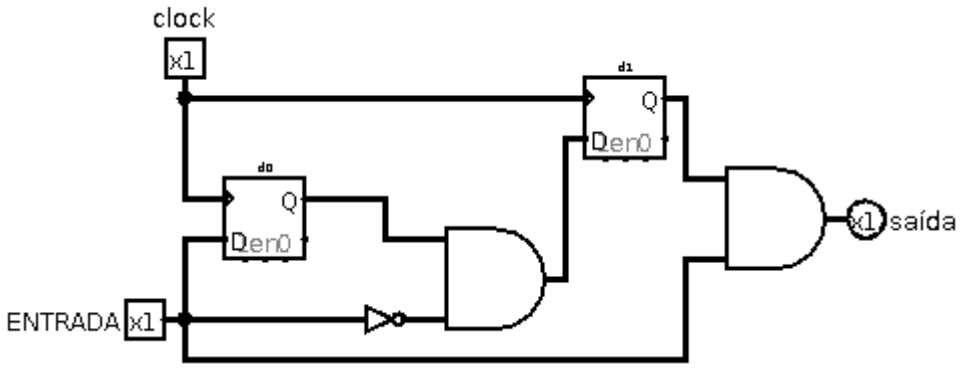


1.9 Detector de sequência binaria

Para criar o componente de sequência binária, que reconheça a sequência "101", foi criado primeiro uma máquina de estados finitos que o aceite, esta máquina possui 3 estados: A(00), B(01) e C(10), os FF tipo d “d1” e “d0” representam juntos

em qual estado a máquina está(com “d1” sendo o bit mais significativo). Para criar a tabela verdade, as entradas foram definidas como: d1, d0 e x(entrada), e as saídas são: d1’(próximo estado de d1), d0’(próximo estado de d0) e S(saída), a partir da tabela verdade se chega à essas expressões do detector representado na figura 13, que ao reconhecer a entrada “101”, sua saída se torna 1.

Figura 13 - Detector de sequência binaria



1.10 Ula de 8 *bits*

A ULA é um componente digital responsável por realizar operações lógicas e aritméticas básicas. Esta ULA possui as seguintes características e funcionalidades.

1.10.1 Entradas

- A (8 *bits*): Operando de entrada principal.
- B (8 *bits*): Operando de entrada secundário.
- SEL (4 *bits*): Sinal de seleção que determina a operação a ser executada.

1.10.2 Saídas

- *RESULT* (8 *bits*): Resultado da operação selecionada. 2.
- *overflow_sub* (1 *bit*): *overflow* do sub .
- *overflow_sum* (1 *bit*): *overflow* de soma.

1.10.2 Operações Disponíveis (Controladas pelo SEL)

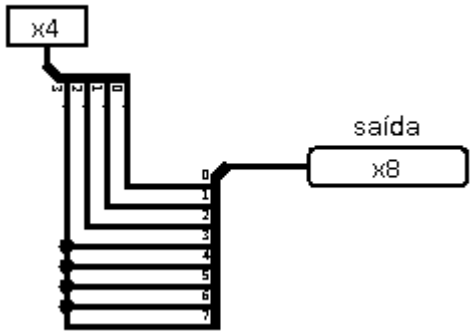
1. *AND*: Operação lógica *AND* entre A e B. *RESULT* = A&B
2. *OR*: Operação lógica *OR* entre A e B. *RESULT* = A| B
3. *NOT* (A): Operação lógica *NOT* aplicada apenas ao operando A. *RESULT* = ~A
4. *NOR*: Operação lógica *NOR* entre A e B. *RESULT* = ~ (A | B)
5. *NAND*: Operação lógica *NAND* entre A e B. *RESULT* = ~ (A & B)
6. *XOR*: Operação lógica *XOR* entre A e B. *RESULT* = A⊕ B
7. SUBTRAÇÃO: Subtração entre os operandos A e B. *RESULT* = A-B
8. *SHIFT LEFT*: Deslocamento lógico do operando A para a esquerda em 2 *bits*. *RESULT* = A<<2

9. *SHIFT RIGHT*: Deslocamento lógico do operando A para a direita em 2 *bits*. *RESULT* = A>>2
10. SOMADOR: Somados operandos A e B, com detecção de "*carry out*". *RESULT* = A+B *overflow_sum* = (Resultado > 255)

1.11 Extensor de Sinal 4 *bits* para 8 *bits*

Um extensor de sinal de 4 *bits* para 8 *bits* copia os 4 *bits* de entrada para os 4 *bits* menos significativos da saída. Para extensão de sinal, o *bit* mais significativo da entrada (*bit* 3) é replicado nos 4 *bits* mais significativos da saída. Isso mantém o valor correto em complemento de dois.

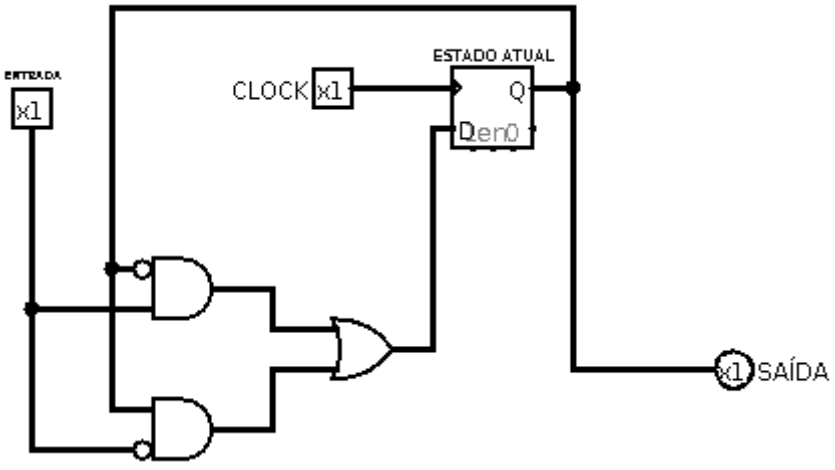
Figura 14 - Extensor de sinal



1.12 Máquina de estados: semáforo

Máquina de estados de um semáforo de pedestres, tem dois estados: verde e vermelho, a entrada é um botão que quando está em 0 não faz nada, e quando está em 1, muda o estado da máquina.

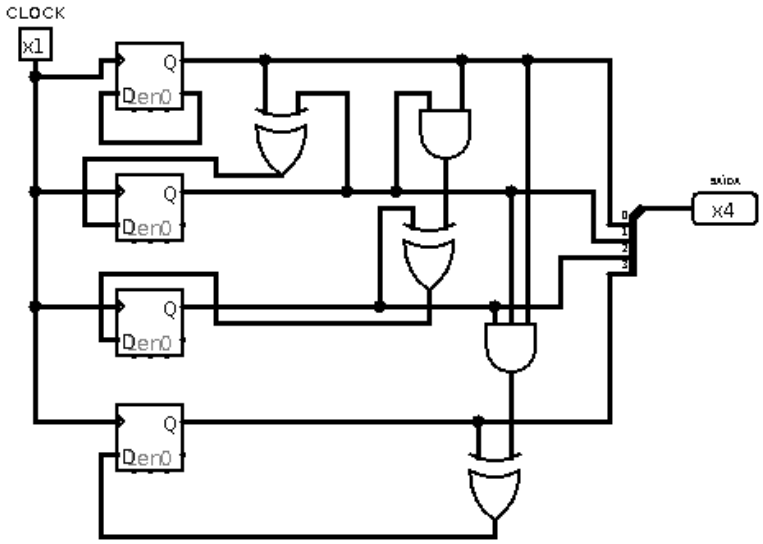
Figura 15 - Semáforo de pedestres



1.13 Contador síncrono

Um contador síncrono de 4 bits é um circuito digital que conta sequencialmente em binário, de 0 a 15, sincronizado por um único sinal de *clock*. Ele utiliza *flip-flops* para armazenar os *bits* e avança seu estado a cada pulso do *clock*, alterando os valores de acordo com uma lógica combinacional que define a sequência binária. A tabela verdade tem como entrada os bits atuais(FF3, FF2, FF1, FF0) e os bits futuros são as saídas(FF3+, FF2+, FF1+, FF0+).

Figura 16 - Contador síncrono



1.14 Paridade ímpar

Esse componente mostra quais dos sinais apresentam quantidades ímpares de 1s

Figura 17 - Paridade ímpar

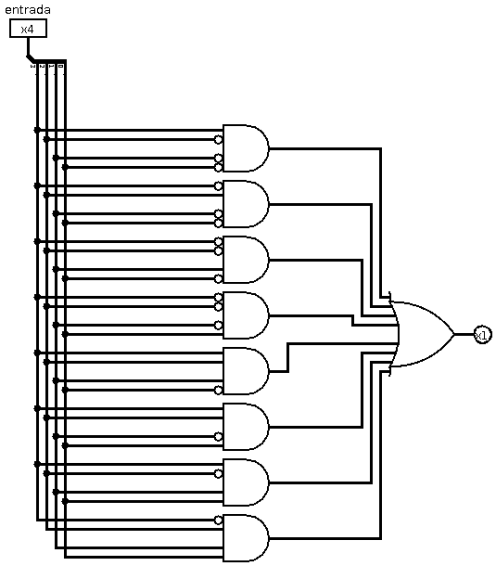
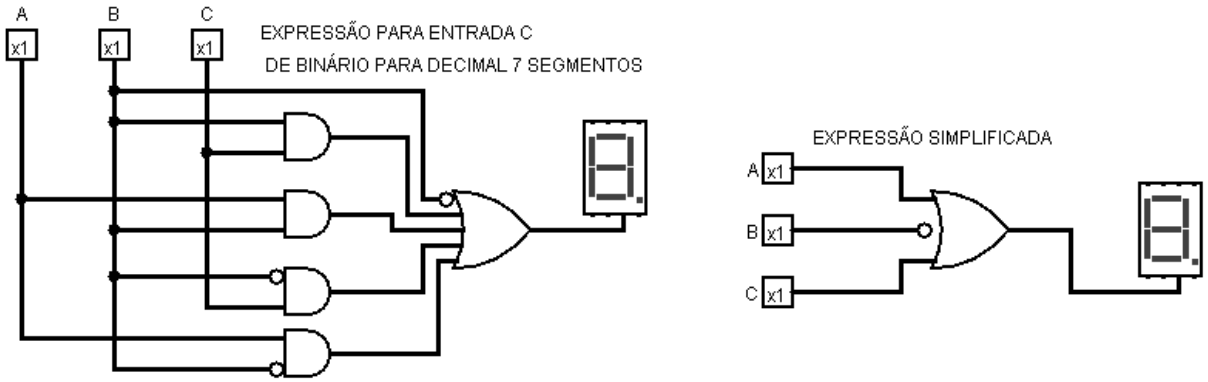


Tabela 3 - Tabela de paridade ímpar

ENTRADA				SAÍDA
0	0	0	1	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

1.15 Otimização lógica

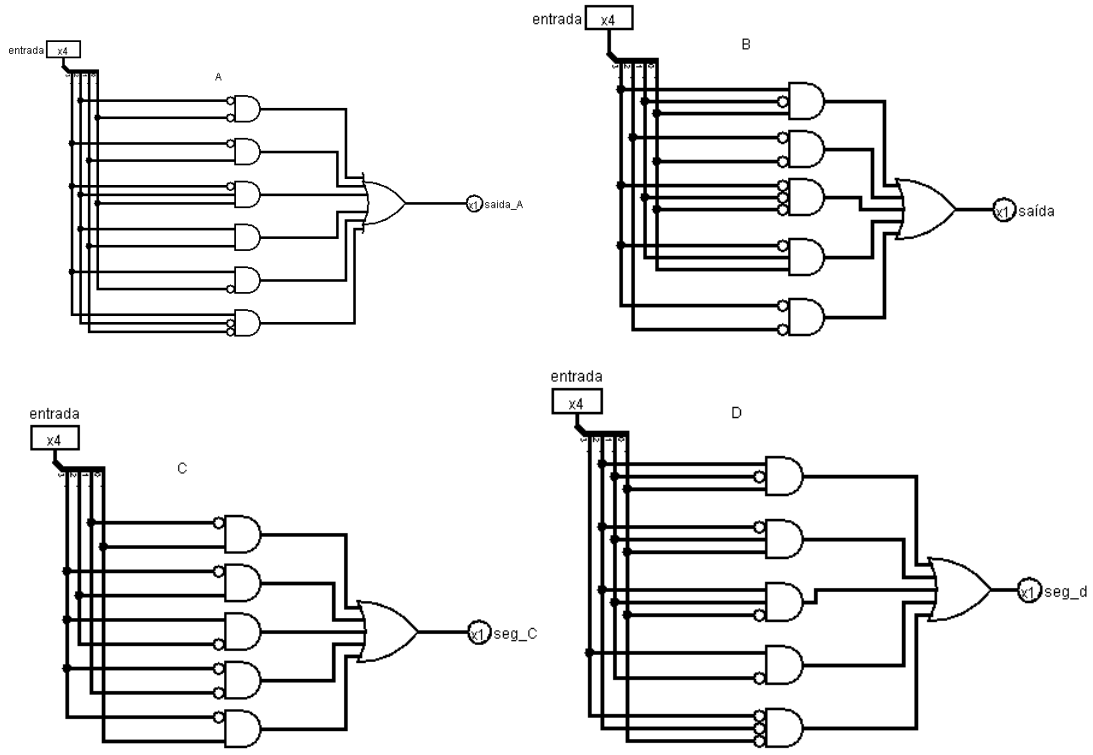
Figura 18 - Circuito simplificado



Em um decodificador de um binário de 3 bits para acender LED'S de um display de 7 segmentos em decimal, o LED "C" vai acender em todas as ocasiões, com a exceção do número 2, este circuito simplificado é o decodificador para o LED "C", de um binário de 3 bits para decimal.

1.16 Decodificador de 7 segmentos

Este decodificador converte um binário de 4 bits para os sinais necessários para acionar um display de 7 segmentos em hexadecimal, cada um dos componentes (A-G) está relacionado a um dos *leds* do display, para chegar à essas expressões foi necessário o uso de tabelas verdade, para cada led do display, onde as entradas eram os 4 bits, e a saída o sinal para o *led*.



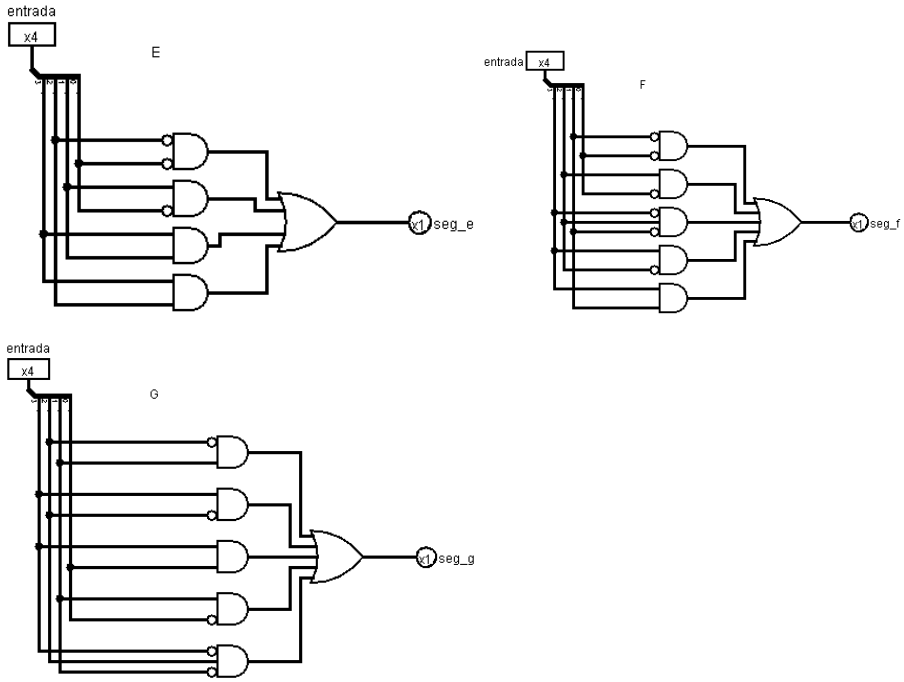
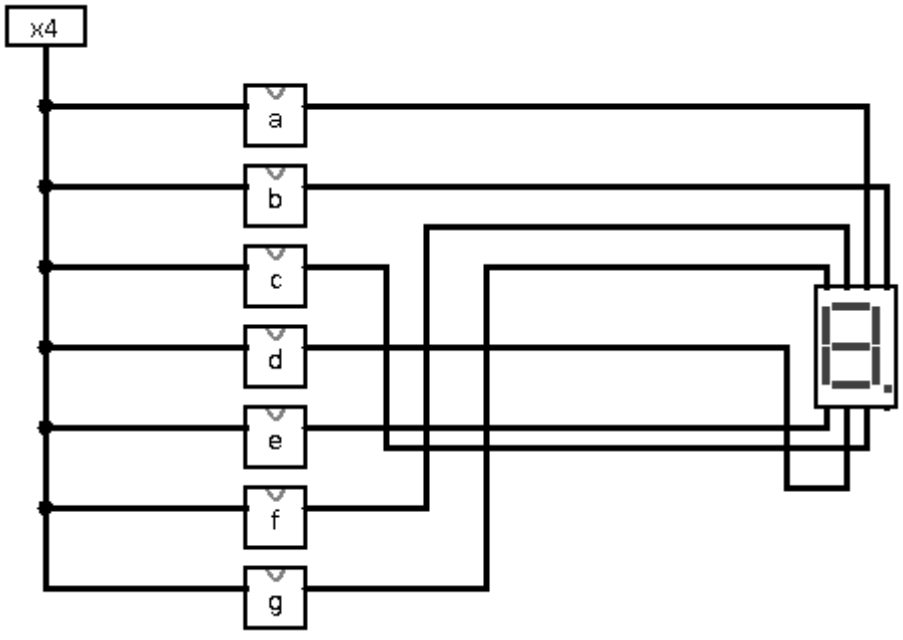


Figura 19 - *Display* hexadecimal



1.17 Detector de números primos

O componente através de portas logicas recebe uma entrada de 4 *bits* e a saída é 1 apenas se o número for primo.

Figura 20 - Número primo

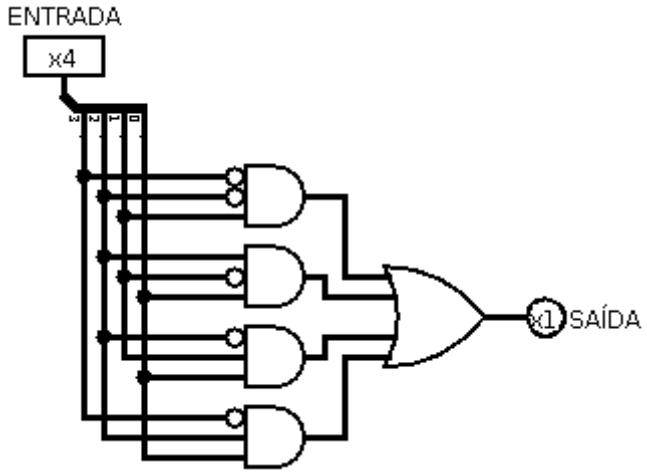


Tabela 4 - Tabela dos números primos

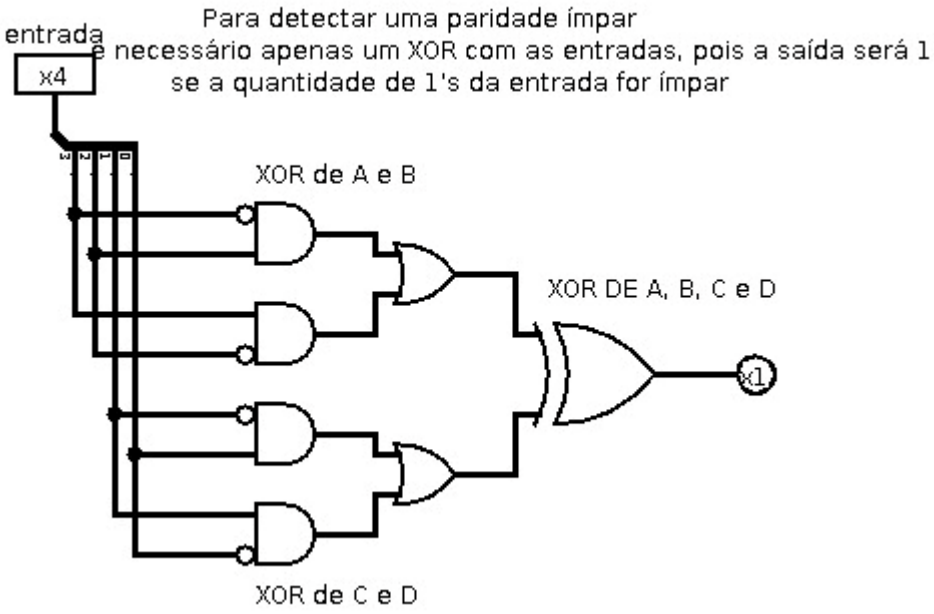
ENTRADA				SAÍDA
0	0	1	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	1	1	1
1	1	0	1	1

2 Paridade binaria pedida em sala

Foi solicitado um algoritmo para que o detector de paridade ímpar fosse escalonável, pois da forma como foi implementado, tornava praticamente impossível o aplicar em grandes quantidades de entradas. Para detectar uma paridade ímpar é necessário apenas inserir as entradas de N bits em um XOR de N bits, assim a saída

será 1 se a quantidade de 1's da entrada for ímpar, o detector implementado possui 4 bits, onde foi feito o (a⊕b) ⊕ (c⊕d), ou seja, o xor de 4 bits.

Figura 21 - Paridade binaria



REFERÊNCIAS

Maganha, G. V. (7 de 12 de 2024). *GV ensino*. Fonte: Youtube: https://youtube.com/@gvensino?si=6hcXfaUM9wlMjL1r

Souza, P. (7 de 12 de 2024). *Pedro Souza*. Fonte: Youtube: https://youtube.com/@pedrosouza-bu2dn?si=s4IaZ1mUzetzPUC1