Projeto RTL - Interface de Barramento

Téo Mendes Araújo - 2024015454
Giovana Nunes Fioravante - 2024103850
Turma PN5

12 de Dezembro de 2025

Sumário

1 Introdução

O presente projeto[1] fundamenta-se na criação de uma interface de Barramento utilizando a linguagem de descrição de hardware VHDL. O objetivo principal é aprimorar a capacidade de manipulação e tráfego de dados entre módulos digitais, consolidando os conhecimentos na metodologia RTL (*Register Transfer Level*).

2 Definição e Arquitetura de Barramento

Para alcançar os objetivos propostos, desenvolveu-se uma interface de barramento baseada na troca de informações entre o processador e seus módulos periféricos. Teoricamente, o barramento é um conjunto de condutores paralelos subdivididos em três vias distintas:

- **Barramento de Endereço (ADDR):** Define a localização do módulo a ser acessado.
- **Barramento de Dados (DATA):** Via de transporte bidirecional de informação.
- **Barramento de Controle (READW):** Responsável pela temporização e especificação da operação (sinal lógico para *READ* ou *WRITE*).

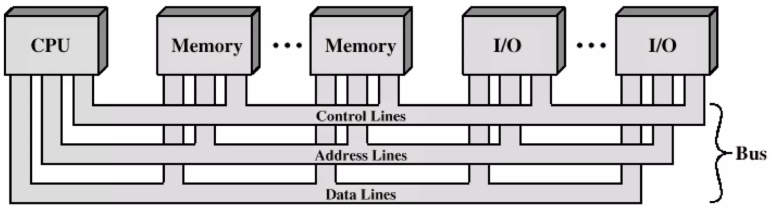


Figura 1: Esquemático da Interface de Barramento

O projeto da interface, conforme ilustrado na metodologia RTL, resolve o problema de controle de acesso ao meio compartilhado. A implementação requer uma lógica de controle que gerencia a transição entre os estados operacionais, garantindo que o módulo periférico só participe da comunicação quando especificamente endereçado.

2.1 Protocolo de Comunicação

O processador decide qual periférico (ex: memória) deseja acessar posicionando o endereço único deste no **Barramento de Endereços**. Todos os periféricos leem esse endereço, mas apenas aquele cujo identificador interno corresponde ao transmitido é ativado. Em seguida, o processador envia um comando pelo **Barramento de Controle** (ex: $readW = 0$ para leitura ou $readW = 1$ para escrita), definindo a direção do fluxo:

- **Leitura:** O periférico ativado insere seu dado no Barramento de Dados. O processador copia a informação e o periférico desconecta-se (estado de alta impedância) para liberar a via.
- **Escrita:** O processador insere o novo dado diretamente no Barramento de Dados e o periférico ativado armazena a informação em seu registrador interno.

Em suma, a interface estrutura-se na relação mestre-escravo entre o processador, suas memórias e outros dispositivos de I/O.

[1]Palavras-chave: Barramento, Processador, Dados, Endereço, Von Neumann

3 Metodologia RTL

3.1 Diagrama de Blocos

O sistema consiste em um processador com 8 bits de memória interna, uma Unidade Lógica e Aritmética (ALU) e uma unidade de controle.

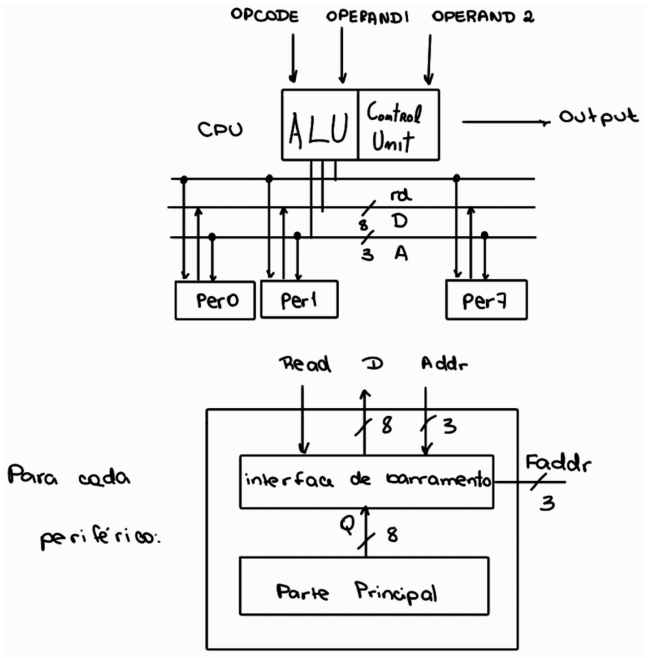
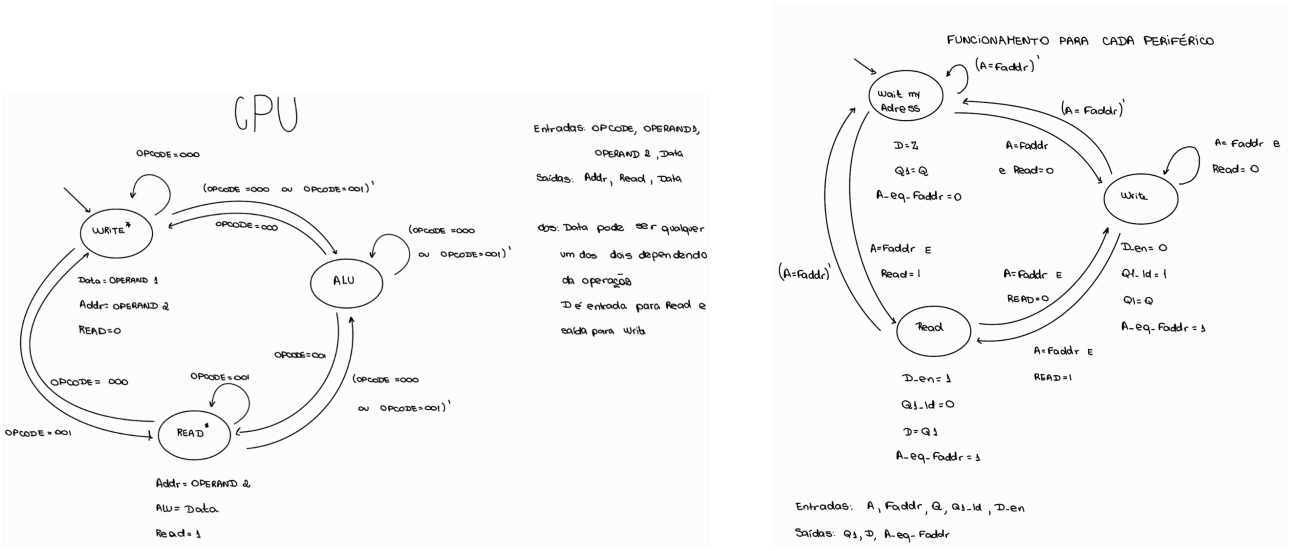


Figura 2: Diagrama de Blocos de Alto Nível

3.2 Diagrama de Fases

Abaixo, apresentam-se os diagramas temporais e de estados das fases de operação da CPU e dos periféricos.



(a) Fases da CPU

(b) Estados dos Periféricos

Figura 3: Diagramas de Fases do Sistema

3.3 Controladora

Representa a unidade de controle dentro do caminho de dados, executando funções que modulam o comportamento do processador de acordo OPCODEs

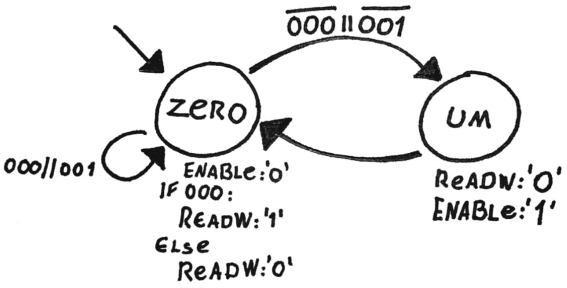


Figura 4: FSM

3.4 Caminho de Dados (Datapath)

O processador utiliza uma **pseudo-arquitetura de Von Neumann**.

A formatação da instrução é determinada pela entrada *opcode*:

- `000`: Dado → Endereço
- `001`: Endereço → *Don't care*
- `Demais`: Endereço → Endereço

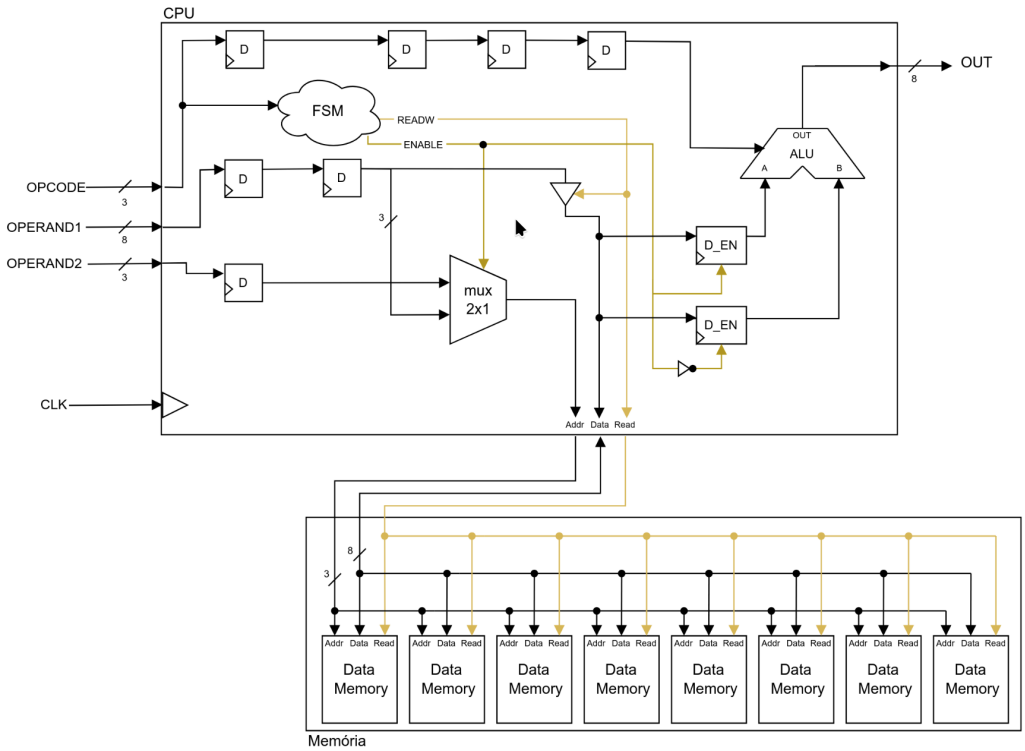


Figura 5: FSM

Por conta da característica de DATA ser INOUT, por conta da lógica do barramento, se fez necessário a implementação de um tri-state, o qual modula a impedancia do trannsistor para habilitar troca de dados

3.5 Unidade Lógica e Aritmética (ALU)

A ALU é implementada via multiplexador e executa instruções baseadas no *OPCODE*.

Opcode	Mnemônico	Descrição
000	WRITE	Escrita na memória
001	READ	Leitura da memória
010	NOT	Inversão lógica (Complemento de 1)
011	AND	AND Bitwise
100	ADD	Adição
101	SUB	Subtração
110	SLL	Shift Left Logical
111	SRL	Shift Right Logical

Tabela 1: Tabela de Operações da ALU

Apesar de não haver instruções explícitas para *OR bitwise*, multiplicação ou divisão, é possível executá-las via software através da repetição de operações básicas, similar à arquitetura de processadores RISC simplificados (como alguns núcleos ARM Cortex-M0 que não possuem hardware dedicado para divisão).

4 Netlist e Esquemáticos RTL

Esta seção apresenta a síntese dos componentes principais gerada pela ferramenta de CAD.

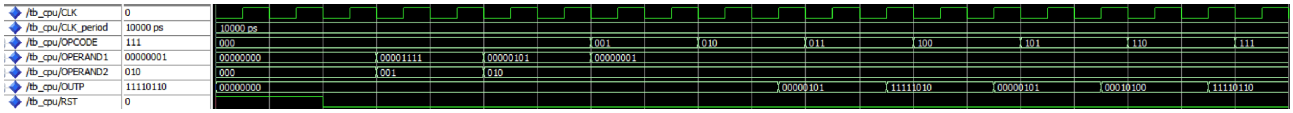


Figura 6: Wave signal

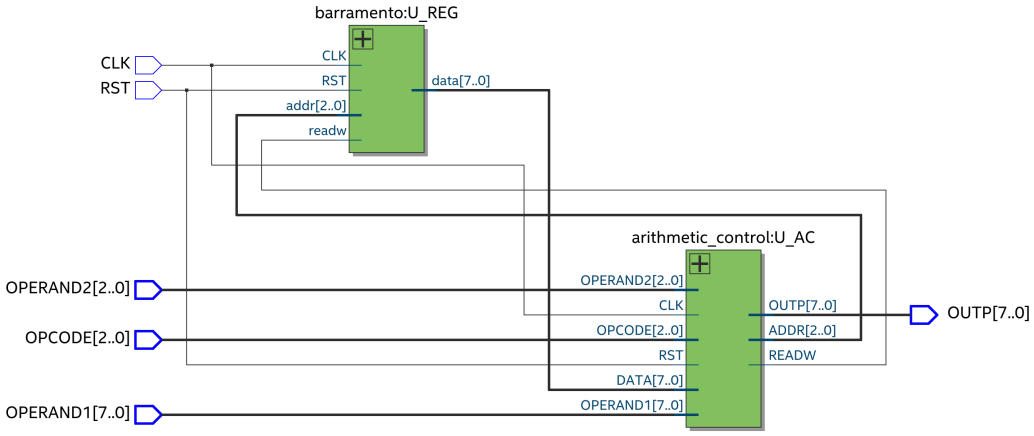


Figura 7: Netlists

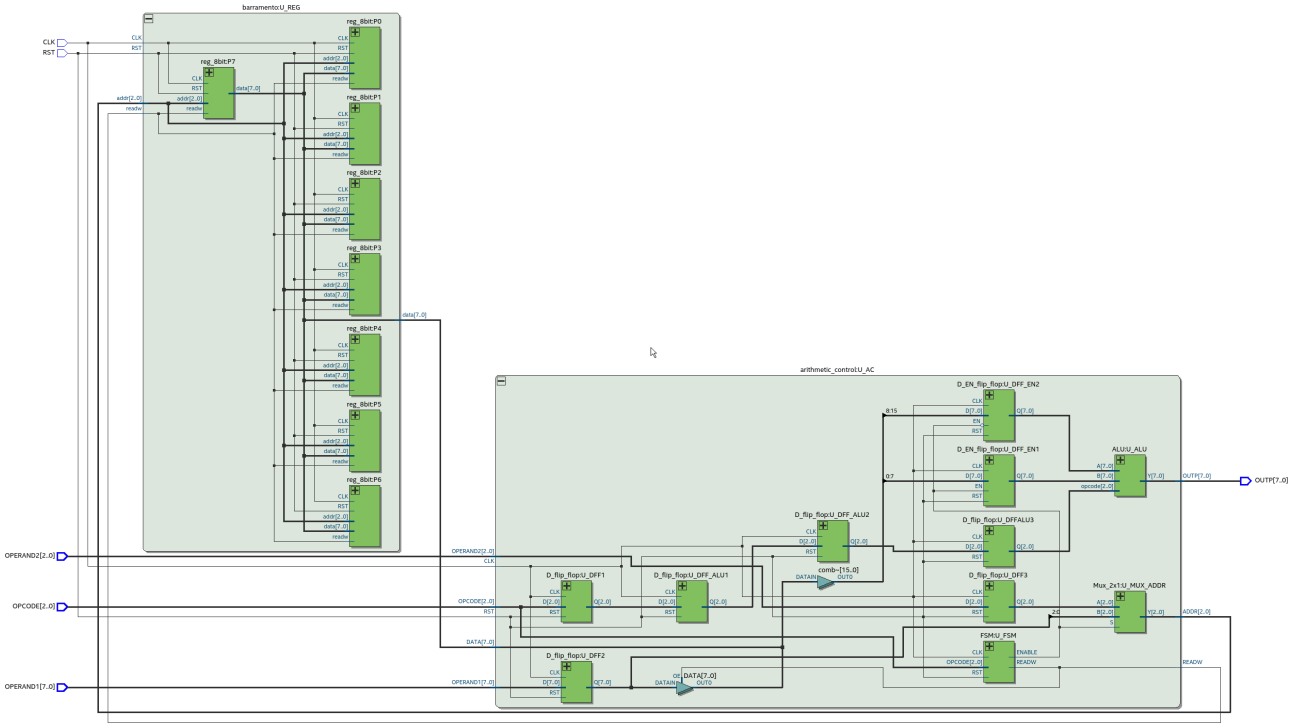


Figura 8: Visão expandida da Netlist

5 Funcionamento e Pipeline

O sistema utiliza um modelo de **pipeline de dois estágios**, o qual caracteriza-se paralelismo de instruções. A representação numérica adotada é o **Complemento de 2**.

5.1 Estágios do Pipeline

1. **Busca/Decodificação:** No primeiro ciclo, o *Opcode* é analisado. Instruções de escrita (`000`) ou leitura roteiam os operandos para a ALU.

2. **Execução:**

   - **Latência:** Devido à estrutura do pipeline, o resultado da operação estará disponível na saída após 4 ciclos de *clock*. Isso permite que, enquanto uma instrução é calculada, a próxima seja instruída.
   - **Conflitos:** Por se tratar de um pipeline de 2 ciclos, em ciclos impares as operações de escrita podem sofrer conflitos de dados (sobrescrita) devido à seleção do MUX de operandos. Recomenda-se a inserção de um ciclo de espera (*stall*).

Nota sobre FSM: Por se tratar de uma implementação paralelizada multiciclo a FSM é essencial para escolher como o OPCODE é decodificado, desde habilitando escrita à fazer operações aritiméticass

6 Conclusão

A implementação de um processador sem o conhecimento prévio da matéria de AOC(Arquitetura de Computadores) demonstrou-se como um excelente exercício de criatividade e pesquisa, além de ter ajudado na fixação da linguagem VHDL e técnicas RTL. O projeto cumpriu o objetivo de solidificar o conhecimento prático em Sistemas Digitais.

7 Referências Bibliográficas

Referências

[1] GEEKSFORGEEKS. **Computer Organization | Von Neumann architecture**. Disponível em: https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-von-neumann-architecture/. Acesso em: 10 out. 2025.

[2] SARANYA, S. **Bus Structures**. LinkedIn Pulse, 2018. Disponível em: https://www.linkedin.com/pulse/bus-structres-saranya-s/. Acesso em: 10 out. 2025.

[3] GEEKSFORGEEKS. **Difference between Von Neumann and Harvard Architecture**. Disponível em: https://www.geeksforgeeks.org/computer-organization-architecture/difference-between-von-neumann-and-harvard-architecture/. Acesso em: 10 out. 2025.

[4] TAPPERO, Fabrizio; MEALY, Bryan. **Free Range VHDL**. Disponível em: https://github.com/fabriziotappero/Free-Range-VHDL-book. Acesso em: 10 out. 2025.

[5] GEEKSFORGEEKS. **Computer Organization | Instruction Formats**. Disponível em: https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-instruction-formats-zero-one-two-three-address-instruction/. Acesso em: 10 out. 2025.

8 Apêndices

Todo o código fonte, arquivos de simulação e *testbenches* relativos a este trabalho encontram-se disponíveis no repositório:

https://github.com/TeoMAraujo/Laboratorio-de-Sistemas-Digitais/tree/main/TP