UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

CENTRO DE TECNOLOGIA

ESCOLA POLITÉCNICA

Relatório: Unidade Lógica e Aritmética (ULA)

Sistemas Digitais

2025.1

Alunos:

Dimitri (122138668),

Felipe Vasconcellos (119154566),

Gustavo Moura (119156071).

Rio de Janeiro

1. Introdução

A Unidade Lógica e Aritmética (ALU - Arithmetic Logic Unit) é responsável pela execução de operações lógicas e aritméticas em processadores e outros dispositivos computacionais. Neste contexto, o presente projeto consiste na implementação de uma ALU de 4 bits com capacidade de realizar 8 operações lógicas ou aritméticas. Este trabalho inclui uma descrição detalhada do projeto com ênfase nos seguintes tópicos:

- **Diagrama de estados e diagrama de blocos:** ilustrando a estrutura e o fluxo de operações da ULA;
- **Descrição formal de cada bloco funcional:** explicando sua contribuição para o funcionamento do circuito;
- **Código VHDL comentado:** implementando a ULA de forma modular e eficiente;
- **Simulações pertinentes:** validando o comportamento do circuito em diferentes cenários;
- **Problema do bouncing:** um desafio comum em sistemas digitais que pode causar leituras incorretas de sinais e, portanto, serão discutidas as medidas adotadas para mitigar este efeito, garantindo a confiabilidade e a entrada dos dados.

2. Fluxo de funcionamento da Unidade Lógica e Aritmética (ALU)

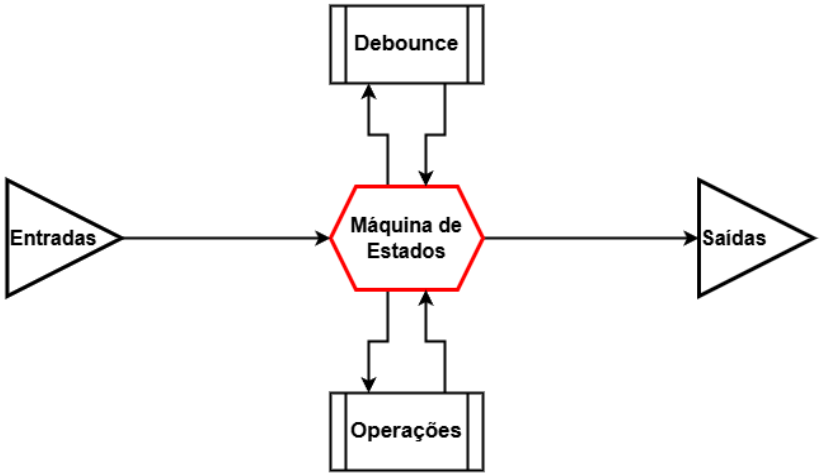

Imagem 1: Diagrama de blocos da Unidade Lógica e Aritmética

- # Operações

O módulo criado para ALU realiza operações com dois operandos de 4 bits (*num_1* e *num_2*), controladas por um código de operação de 3 bits (op). A saída da ALU é um vetor de 8 bits (*result*), composto por 4 flags e o resultado de 4 bits da operação.

A ALU opera de acordo com o valor da entrada op, com as seguintes funções:

- **"000" → Soma**: Soma os operandos como números com sinal. Calcula as flags *carry_out*, *zero*, *negative* e *overflow*.

- **"001" → Subtração**: Subtrai *num_2* de *num_1*, com o mesmo tratamento de flags da soma.

- **"010" → AND**: Operação lógica bit a bit.

- **"011" → OR**: Operação lógica bit a bit.

- **"100" → XOR**: Operação lógica exclusiva.

- **"101" → Comparação**: Retorna 1 se iguais, 2 se *num_1* < *num_2*, se *num_1* > *num_2*.

- **"110" → Shift lógico à esquerda**: Realiza deslocamento de num_1 para a esquerda, com a quantidade indicada por *num_2*.

- **"111" → Shift lógico à direita**: Realiza deslocamento de *num_1* para a direita, com a quantidade indicada por *num_2*.

A saída *result* é formada pela concatenação dos sinais: [overflow], [zero], [negative], [carry_out] e [resultado de 4 bits]. Cabe destacar que todos os sinais são atualizados a cada borda de subida do clock, garantindo sincronia com o sistema.

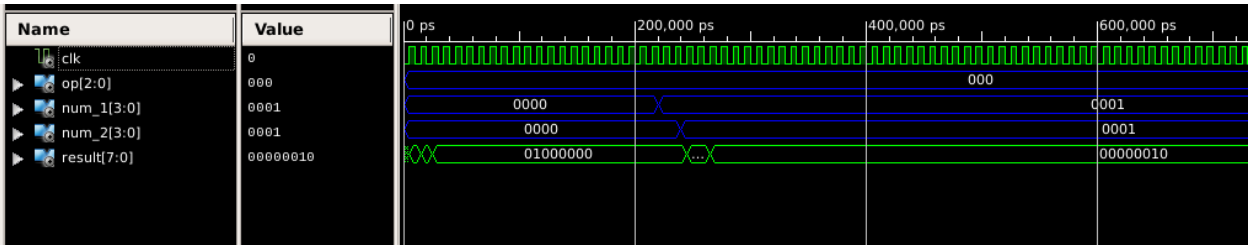

Imagem 2: Simulação da Unidade Lógica e Aritmética

- ## State Machine (bloco top level)

Este bloco controla a captura de entradas e a exibição de saídas. Há apenas um processo sensível ao clock, ao botão set (após ser tratado) e aos switches da placa. Este processo implementa a mudança dos estados ao apertar o botão set.

Há uma exibição prévia nos leds para a indicação do estado. Por exemplo: no primeiro estado, os leds estarão em 00000001, no segundo estado estarão em 00000010 e no terceiro estado estarão em 00000011, conforme mostrado no diagrama de blocos.

Estado Atual	Próximo Estado (após botão)	Saída (leds_out)
s0	**s1**	"00000001"
s1	s2	"00000010"
s2	s3	"00000011"
s3	s4	alu_result
s4	s0	"00000000"

Imagem 3: Tabela de transição de estados

A máquina de estados opera em cinco estados sequenciais (*s0* a *s4*), avançando a cada pressionamento do botão *set_btn*, que passa por um circuito de debounce. O fluxo é controlado por um registrador de estado (*current_state*) sincronizado com o clock.

- **Estado s4**: A máquina inicializa e aguarda o código da operação a partir dos 3 primeiros bits de switches. Os LEDs exibem "*00000000*".
- **Estado s0**: Armazena o código da operação e aguarda o primeiro operando de 4 bits a partir de switches. Os LEDs mostram "*00000001*".
- **Estado s1**: Armazena o primeiro operando e aguarda segundo operando de 4 bits a partir de switches. LEDs mostram "*00000010*".
- **Estado s2**: Armazena o segundo operando e aguarda o próximo estado. LEDs mostram "*00000011*".
- **Estado s3**: Executa a operação da ULA com os operandos fornecidos. O resultado é exibido diretamente nos LEDs.

Esse ciclo se repete indefinidamente, permitindo a execução contínua de operações com novos dados a cada ciclo completo.

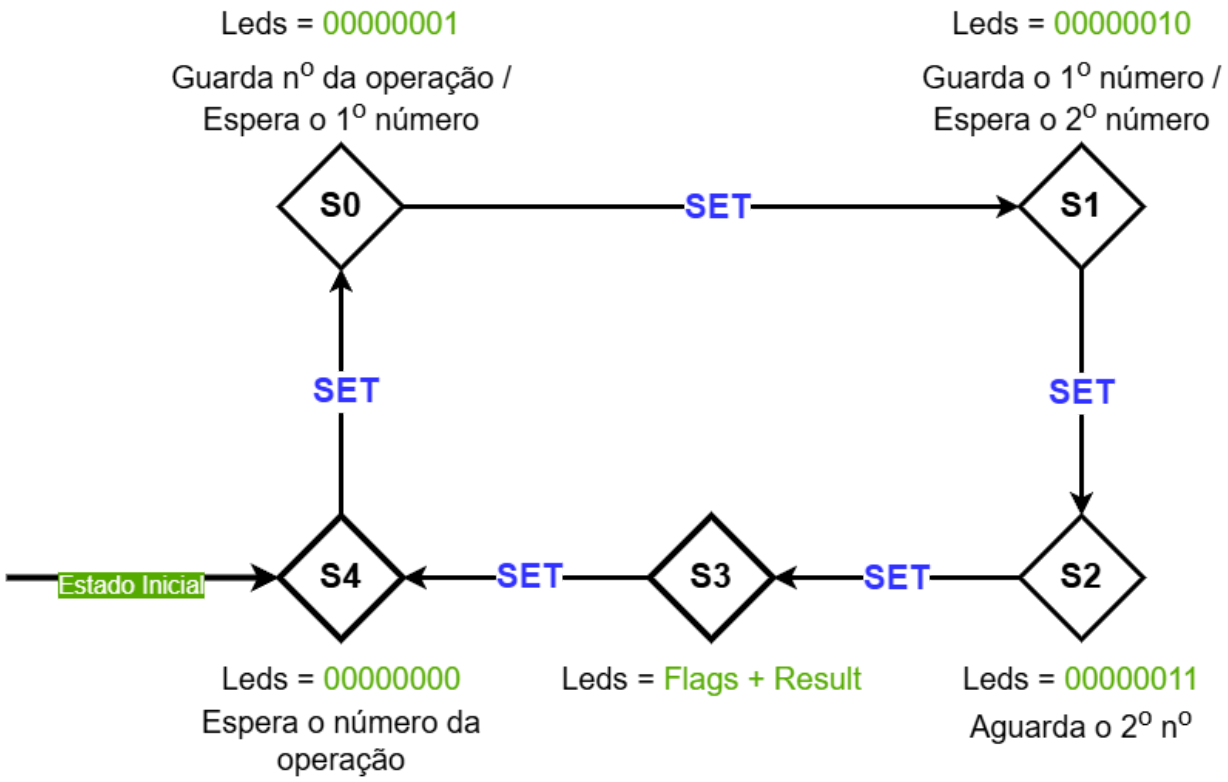

Imagem 4: Diagrama de blocos da Máquina de Estados

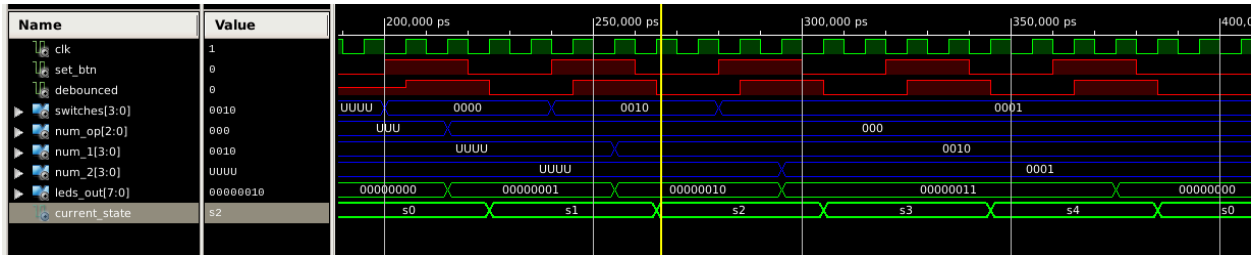

Imagem 5: Simulação da Máquina de Estados

- ## Debounce do botão *set*

Quando um botão físico é pressionado (em um contexto eletrônico), o contato mecânico que ocorre no botão não se encerra de forma instantânea e limpa. Na prática, ele gera uma série de pequenos contatos e desconexões muito rápidas antes de estabilizar tanto no estado pressionado quanto liberado, este fenômeno é conhecido como “bounces”.

Cabe destacar que esses "bounces" acontecem em poucos milissegundos, mas para um microcontrolador ou circuito digital, que pode ler o estado do botão a cada microssegundo, isso pode ser interpretado como múltiplos pressionamentos rápidos, podendo prejudicar o desenvolvimento de qualquer projeto neste cenário.

O objetivo deste módulo é filtrar **esses** ruídos, garantindo que apenas uma transição limpa seja registrada por clique.

Como funciona:

- O sinal de entrada *btn_in* representa o botão pressionado.
- O sinal *db* é a saída limpa (debounced).
- Quando o botão muda de estado (pressionado ou solto), o sistema inicia uma contagem usando o clock (*clk*).
- Essa contagem dura até atingir o valor da constante *max* (50 milhões, que equivale a aproximadamente 1 segundo em um clock de 50 MHz).
- Durante esse tempo, novas mudanças no botão são ignoradas.
- Depois desse tempo, o sistema pode registrar uma nova mudança no botão.

Detalhes técnicos:

- A variável *count* funciona como um temporizador.
- *btn_prev* guarda o valor anterior do botão para detectar mudanças.
- A atualização da saída *db* só acontece quando uma mudança real é detectada, e não há contagem em andamento.

3. Conclusão

Após a implementação do código do projeto em VHDL, conseguimos verificar as operações funcionando de forma correta na simulação dentro do software ISE. No entanto, ao tentarmos executar diretamente na placa *Xilinx Spartan 3AN-Starter Kit* verificamos que a mudança de estados não ocorria como esperado, os leds da placa indicavam que o programa não saia do primeiro estado. Tentamos verificar se havia algo de errado com a implementação do bouncing, tentamos reescrevê-lo e o problema persistia até que descobrimos que o problema era que estávamos usando a pinagem *V12* para o clock quando, na verdade,

era para usarmos o E12. Além disso, também descobrimos que estávamos atribuindo o tipo *signal* em vez de *variable* para nomes que precisavam guardar informações que são variáveis e, portanto, o *signal* não executava o programa da forma correta. Após essas identificações, o nosso projeto funcionou conforme o esperado.