RELATÓRIO PBL II: ULA de 8 Bits em FPGA

Henrique Zeu, Luan Rodrigues Araújo
Universidade Estadual de Feira de Santana – UEFS
Engenharia de Computação
luanlra002@gmail.com, henrique.zeuu@gmail.com

2025

Resumo

O presente relatório descreve o desenvolvimento de uma Unidade Lógica e Aritmética (ULA) de 8 bits implementada em FPGA, utilizando a linguagem de descrição de hardware Verilog no ambiente Quartus II. O projeto teve como objetivo projetar e sintetizar uma calculadora digital baseada na notação polonesa reversa (RPN), capaz de realizar operações aritméticas e lógicas, armazenar resultados em registradores e exibir valores em diferentes bases numéricas (decimal, octal e hexadecimal). O sistema foi desenvolvido em arquitetura modular, combinando circuitos combinacionais e sequenciais, com a inclusão de flags de estado, como overflow, carry out, zero e erro. O protótipo foi sintetizado na placa DE10-Lite, permitindo a interação através de chaves e botões físicos, e a visualização de resultados em displays de sete segmentos e LEDs indicadores. Os resultados obtidos demonstram o funcionamento correto da ULA e a viabilidade de sua aplicação como ferramenta didática para o estudo de sistemas digitais e arquitetura de processadores.

Palavras-chaves: Verilog, ULA, FPGA, Circuitos Digitais, RPN, Quartus II, Displays de Sete Segmentos.

Introdução

Ao longo do desenvolvimento da computação digital, as Unidades Lógicas e Aritméticas (ULAs) consolidaram-se como elementos fundamentais na arquitetura de processadores e microcontroladores. Elas são responsáveis pela execução das operações aritméticas e lógicas essenciais ao funcionamento de qualquer sistema computacional, sendo o núcleo de processamento em dispositivos digitais. O avanço da eletrônica digital e o uso de linguagens de descrição de hardware, como o Verilog, possibilitaram o estudo, a simulação e a prototipagem dessas unidades em plataformas reconfiguráveis, como as FPGAs (Field Programmable Gate Arrays).

Neste contexto, o presente projeto tem como objetivo o desenvolvimento de uma ULA de 8 bits implementada em FPGA, capaz de realizar operações aritméticas, lógicas e

especiais, seguindo o modelo de calculadora digital baseada em notação polonesa reversa (RPN – Reverse Polish Notation). O sistema deve permitir a entrada de operandos, execução de operações diversas e exibição dos resultados em diferentes bases numéricas (decimal, octal e hexadecimal), além de armazenar automaticamente o último resultado em um registrador de memória reutilizável.

A construção do protótipo busca aplicar os conceitos teóricos de circuitos combinacionais e sequenciais, abordando também tópicos como flags de estado, conversão de bases e modularização de sistemas digitais. O projeto foi desenvolvido utilizando a ferramenta Quartus II e o kit de desenvolvimento DE10-Lite, adotando o Verilog Estrutural como linguagem de descrição de hardware.

Dessa forma, este relatório está estruturado em introdução, metodologia, desenvolvimento, resultados, e conclusão, apresentando o processo de construção, simulação e validação da ULA de 8 bits, evidenciando sua importância didática para a compreensão da arquitetura de sistemas digitais e da operação de processadores em nível de hardware.

1 Metodologia

1.1 Referencial Teórico

Nesta seção, são abordados os diferentes tipos de circuitos utilizados no desenvolvimento, suas características, conceitos-chave de projeto e a linguagem de descrição de hardware aplicada. O protótipo foi desenvolvido em sistemas digitais e utiliza tanto circuitos combinacionais quanto sequenciais. Todas as informações desta seção foram obtidas do livro Sistemas Digitais: Princípios e Aplicações, de Ronald J. Tocci, Neal S. Widmer e Gregory L. Moss (TOCCI, 2011).

1.1.1 Circuitos Sequenciais

Os circuitos sequenciais são elementos de sistemas lógicos digitais que, diferentemente dos circuitos combinacionais, possuem memória. Isso significa que suas saídas dependem não apenas das entradas atuais, mas também do histórico de estados anteriores. Esses circuitos são controlados por um sinal de clock, responsável por sincronizar as mudanças de estado em intervalos regulares.

Graças a essa característica, os circuitos sequenciais são ideais para aplicações que exigem armazenamento de dados ou operações realizadas em sequência, como contadores, registradores e máquinas de estados finitos.

Enquanto os circuitos combinacionais respondem de forma imediata às variações nas entradas, os circuitos sequenciais atualizam seus estados internos de forma controlada e sincronizada pelo clock, possibilitando a execução de operações mais complexas e organizadas no tempo.

1.1.2 Latches e FlipsFlops

Os latches e flip-flops são componentes de memória digital encarregados de guardar um único bit de informação, porém diferem na maneira como executam essa função. Ambos são essenciais em circuitos sequenciais, mas os latches funcionam de forma transparente, ao passo que os flip-flops reagem à borda do sinal de clock. Esses componentes possibilitam

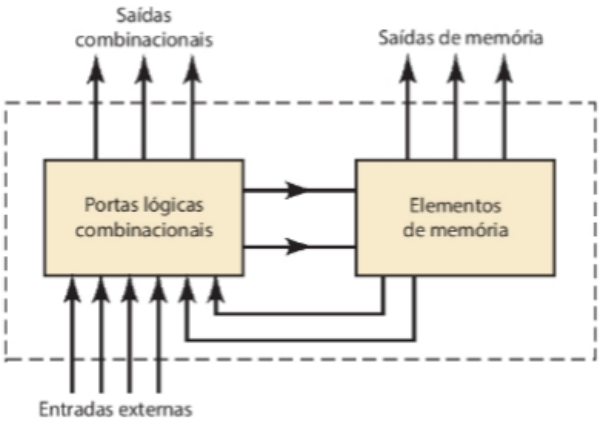

Figura 1 – Diagrama geral de um sistema digital. Fonte: (TOCCI, 2011).

que sistemas digitais armazenem e processem dados de maneira sincronizada com o clock, sendo frequentemente empregados em contadores, registradores e circuitos similares.

Os latches mantêm a saída em um determinado estado enquanto o sinal de controle (ou habilitação) estiver ativo. Existem diferentes tipos, como o latch S-R, que utiliza duas entradas para definir ou redefinir a saída, e o latch D, que armazena o valor da entrada de dados enquanto o controle permanece habilitado.

Já os flip-flops alteram o estado de saída apenas no instante da borda do clock — seja ela de subida ou de descida. Entre os principais modelos estão o flip-flop D, que transfere o valor da entrada D no momento da transição do clock, e o flip-flop J-K, capaz de realizar operações mais complexas, como a comutação entre estados. Além de serem sincronizados por clock, os flip-flops se destacam por evitar estados indefinidos, tornando-os mais adequados para sistemas sequenciais síncronos.

O projeto foi elaborado com flip-flops do tipo D (Data Flip-Flop) e flip-flops do tipo T (Toggle Flip-Flop), componentes comumente utilizados em sistemas digitais devido à sua simplicidade e eficácia no armazenamento de informações.

O flip-flop D tem uma única entrada de dados (D) e uma entrada de clock (CLK). Sua função principal é capturar o valor presente na entrada D na borda ativa do clock e manter esse valor constante na saída Q até a próxima transição do sinal de clock. Essa propriedade o torna perfeito para aplicações que necessitam de armazenamento e sincronização de dados, pois o aparelho preserva um estado estável durante todo o período entre as bordas. Ademais, o flip-flop D foi selecionado por não exibir estados indeterminados, ao contrário de outros modelos, como os flip-flops S-R e J-K, o que o faz mais confiável e simples de implementar.

O projeto também utiliza o flip-flop tipo T, que pode mudar o estado da saída Q a cada pulso de clock, desde que sua entrada T esteja em nível lógico alto. Essa característica é particularmente vantajosa em circuitos contadores e divisores de frequência, uma vez que possibilita a criação de sinais alternados de maneira simples e estável. A combinação de flip-flops D e T no sistema permitiu um gerenciamento exato e sincronizado dos estados lógicos, assegurando uma maior confiabilidade no desempenho do circuito.

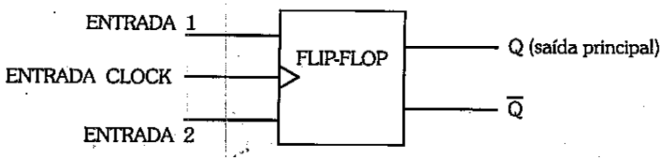

Figura 2 – Símbolo geral para um flip-flop. Fonte: (CAPUANO; IDOETA, 2019)

.

1.1.3 Contadores

Os contadores são circuitos formados por flip-flops interligados em sequência, responsáveis por registrar e avançar estados lógicos a cada pulso de clock. De acordo com a aplicação pretendida, esses aparelhos podem ser ajustados para funcionar em várias bases numéricas, como binária, decimal (BCD) ou outras.

Um exemplo típico é o contador de módulo 8, que emprega três flip-flops para representar os valores de 0 a 7 em código binário, retornando automaticamente ao estado inicial após a conclusão do ciclo. O estado dos flip-flops é modificado a cada pulso de clock, e o bit mais significativo (MSB) é representado pelo último flip-flop da sequência.

Na eletrônica digital, os contadores são essenciais, pois exercem um papel crucial no gerenciamento de operações temporizadas, na sequência de eventos e na medição de frequência. Ademais, são frequentemente utilizados em temporizadores, divisores de frequência, geradores de sinais e em vários sistemas embarcados que necessitam de um controle preciso e sincronizado de eventos.

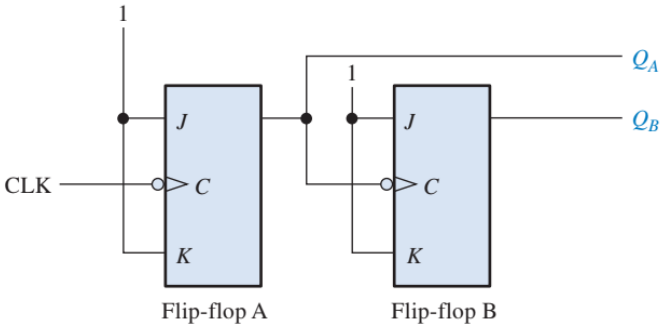

Figura 3 – Flip-flops usados para gerar uma seqüência de contagem binária. Fonte: (FLOYD, 2007)

.

1.1.4 Debounce

O debounce é o processo de eliminação do efeito de trepidação de contato (contact bounce) causado por dispositivos mecânicos, como chaves e botões, ao serem acionados. Em sistemas digitais, quando uma chave é pressionada, seus contatos podem oscilar rapidamente entre os estados aberto e fechado, gerando múltiplos pulsos indesejados que o sistema pode interpretar como vários acionamentos.

Para evitar esse problema, o debounce utiliza circuitos eletrônicos, como flip-flops ou buffers, ou ainda técnicas em software que garantem a detecção de apenas uma transição

estável. Dessa forma, o debounce assegura que os sistemas digitais respondam de maneira correta e confiável aos comandos de entrada.

1.2 Multiplexador

O multiplexador, ou MUX, é um circuito lógico combinacional utilizado para selecionar um entre vários sinais de entrada e enviá-lo para uma única saída, de acordo com o valor de sinais de controle chamados de entradas de seleção.

Em um multiplexador com 2^n entradas, existem n linhas de seleção, que determinam qual entrada será conectada à saída. Por exemplo, um MUX 4:1 possui quatro entradas de dados, duas entradas de seleção e uma saída.

Os multiplexadores são amplamente empregados em sistemas digitais para rotear dados, otimizar o uso de barramentos, selecionar fontes de sinais e implementar funções lógicas. Também são elementos fundamentais em processadores, sistemas de comunicação e circuitos de controle, por permitirem a comutação eficiente de informações dentro do hardware.

1.2.1 Conversão de Base

A conversão de bases numéricas é um princípio fundamental na eletrônica e na computação, uma vez que diferentes formatos numéricos são empregados de acordo com a aplicação. As bases mais frequentes são a decimal (base 10), octal (base 8) e hexadecimal (base 16).

A base decimal é a mais conhecida, formada pelos números de 0 a 9. A base octal usa os dígitos de 0 a 7, ao passo que a base hexadecimal utiliza os dígitos de 0 a 9 e as letras de A a F, correspondendo aos valores de 10 a 15.

Essas bases estão diretamente relacionadas à representação binária:

- Cada dígito octal equivale a **3 bits**.
- Cada dígito hexadecimal equivale a **4 bits**.

Por esse motivo, a conversão entre binário e outras bases é realizada agrupando os bits em conjuntos de três ou quatro. Por outro lado, a conversão para decimal emprega o valor posicional de cada dígito, multiplicando-o pela base elevada à potência que corresponde à sua posição.

Essas conversões são frequentemente utilizadas em projetos eletrônicos, programação, endereçamento de memória e interpretação de dados, facilitando a leitura e o processamento de informações numéricas de maneira mais simples e eficiente.

1.3 Especificações da Linguagem Verilog

O Verilog é uma linguagem de descrição de hardware (HDL) muito utilizada no desenvolvimento de circuitos digitais. Ela possibilita que engenheiros e projetistas representem sistemas de forma textual, o que pode ser transformado em implementações físicas de forma eficiente. Uma das maiores vantagens do Verilog é sua portabilidade: projetos criados nessa linguagem podem ser implementados em diversas tecnologias de

hardware e em várias plataformas de ferramentas CAD, sem precisar fazer mudanças significativas no código.

A flexibilidade na descrição de circuitos é outro aspecto relevante do Verilog, que pode ser realizada de duas maneiras: estrutural ou comportamental. A abordagem estrutural explica como os componentes estão conectados, definindo a lógica interna do circuito. Por outro lado, a descrição comportamental se concentra no funcionamento do circuito, possibilitando que ele seja caracterizado com base em seu comportamento previsto, sem a necessidade de especificar o layout físico.

A modelagem de circuitos sequenciais, como flip-flops e registradores, que necessitam de sinais de clock para mudar seu estado, é especialmente eficaz com a abordagem comportamental. Nesse cenário, os blocos always do Verilog são frequentemente empregados para estabelecer atualizações de estado em função das alterações nos sinais, como bordas de clock.

O gerenciamento do clock é essencial para as descrições comportamentais, assegurando a sincronização adequada dos circuitos. Por exemplo, ao usar a expressão `@(posedge Clock)`, o Verilog realiza o bloco somente na borda de subida do clock. Isso é fundamental em flip-flops do tipo D, em que o estado muda precisamente nesse momento. Ademais, o uso de atribuições não-bloqueantes `<=` em blocos sincronizados ao clock possibilita a atualização simultânea de todas as saídas ao término do ciclo, prevenindo questões de dependência de ordem entre os sinais.

1.3.1 Variáveis “REG”

No Verilog, variáveis do tipo reg são usadas para guardar valores que podem ser mantidos por um certo período, permitindo que os dados fiquem acessíveis até que sejam atualizados de forma explícita. Embora o nome indique um registrador, o tipo reg vai além do simples armazenamento físico de bits, ele pode ser usado para representar tanto valores intermediários de cálculos quanto dados que precisam ser mantidos temporariamente.

As "reg"podem ter larguras variadas, estabelecidas no momento da declaração, permitindo o uso de diferentes tamanhos de palavras binárias sem risco de perda de informação. Essa versatilidade possibilita o uso do reg em várias aplicações, tanto para armazenar resultados de operações lógicas quanto para conservar dados que serão empregados em fases futuras do projeto.

Elas são essenciais na modelagem de circuitos digitais por sua habilidade de reter valores e pela possibilidade de determinar o número de bits, proporcionando controle e flexibilidade na manipulação de dados binários.

1.3.2 Bloco Always

O bloco always em Verilog é fundamental para a definição de circuitos combinacionais e sequenciais. Ele permite que o projetista defina a lógica de um circuito que precisa ser reavaliada ou atualizada sempre que certos eventos acontecem, como alterações nos sinais de entrada ou mudanças de borda em um sinal de clock.

Em circuitos combinacionais, o bloco always é acionado sempre que há mudanças em qualquer sinal que compõe a expressão lógica, assegurando que a saída seja atualizada de acordo com as mudanças nas entradas.

Nos circuitos sequenciais, o bloco always é essencial para definir o comportamento sincronizado com o clock. Nessas situações, ele é programado para reagir a transições particulares do sinal de clock, geralmente empregando always @(posedge Clock) para identificar a borda de subida ou always @(negedge Clock) para a borda de descida. Esse tipo de configuração é comumente utilizado na construção de flip-flops e registradores, que mudam seu estado somente nos momentos determinados pelo ciclo do clock.

2 Desenvolvimento

O protótipo foi implementado na placa de desenvolvimento MAX 10, disponível no Laboratório de Eletrônica Digital e Sistemas (LEDS) da Universidade Estadual de Feira de Santana. O projeto foi desenvolvido no software Quartus (Intel, 2024) utilizando a linguagem de descrição de hardware Verilog, adotando o estilo estrutural. Segundo (TOCCI, 2011), o Verilog Estrutural é amplamente utilizado por permitir que um circuito seja descrito por meio da instanciação e interconexão de componentes como portas lógicas, fios e saídas.

O sistema desenvolvido possibilita ao usuário inserir dois números de 8 bits, alternar entre diferentes estados por meio de um botão, selecionar a operação desejada utilizando duas chaves e definir o tipo de conversão de base com outras duas. O resultado da operação e o valor obtido são exibidos em um display de 7 segmentos.

É importante ressaltar que as chaves e botões são usados novamente em várias fases do funcionamento do sistema, o que é viável devido à implementação de uma máquina de estados finita (FSM - Finite State Machine). Essa estratégia possibilita que os mesmos componentes de entrada desempenhem papéis diferentes de acordo com o estado atual do sistema, maximizando a utilização dos recursos disponíveis na placa e tornando o projeto mais eficaz em termos de hardware.

As subsubseções a seguir detalham os principais blocos lógicos que compõem o projeto. Para cada bloco, serão apresentadas as ferramentas e os métodos utilizados em seu desenvolvimento, como as tabelas-verdade, as simplificações de circuitos e as explicações de sua funcionalidade específica. Enquanto estas seções focam nos componentes individuais, uma visualização completa da arquitetura do projeto, que ilustra a interconexão entre todos os blocos, é apresentada na Figura 4.

2.0.1 Adição

O módulo Addition8bit implementa a soma aritmética dos operandos de 8 bits `primeiro_operando` e `segundo_operando`. Este módulo é construído hierarquicamente: ele é composto por duas instâncias de um somador de 4 bits, que por sua vez são compostos por quatro Somadores Completos (Full Adders) de 1 bit. Os módulos de 4 bits são ligados em cascata: o bit de "vai-um"(Cout) do somador dos 4 bits inferiores (bits 3-0) é conectado diretamente ao bit de "vem-um"(Cin) do somador dos 4 bits superiores (bits 7-4). O módulo Addition8bit gera a soma final de 8 bits (sum), que é enviada ao multiplexador de resultado, e o bit de "vai-um"final (`sum_cout`), que é usado pelo módulo FLAGS8bit para detecção de overflow.

- **S (soma):**

$$S = A \oplus B \oplus Cin$$

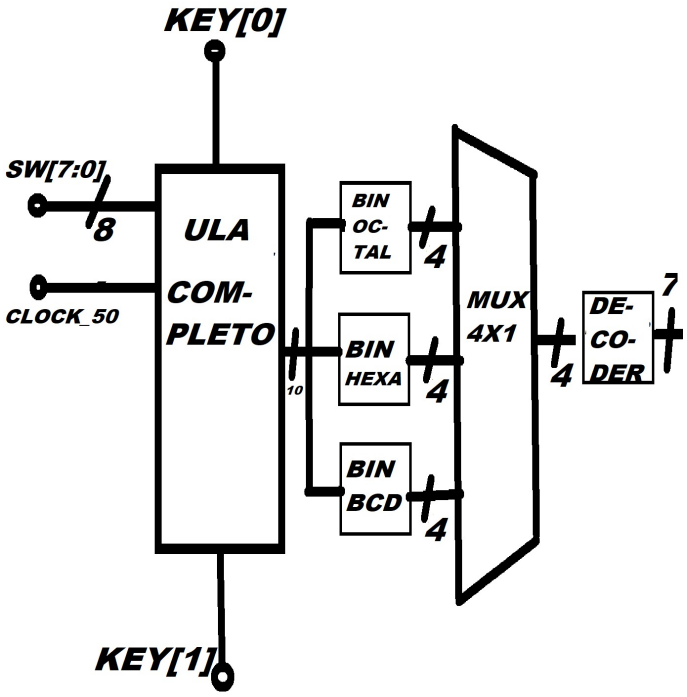

Figura 4 – Diagrama geral do projeto. Fonte: o autor

A	B	Cin	S (Soma)	Cout (Carry)
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

Tabela 1 – Tabela verdade do somador de 4 bits

- **Cout (carry out):**

$$Cout = AB + ACin + BCin$$

2.0.2 Subtração

A subtração é realizada pelo módulo `SUB8bit`, que calcula a diferença `primeiro_operando - segundo_operando`. Assim como o somador, ele é projetado de forma modular, escalando a lógica do *Subtrator Completo* (Full Subtractor) de 1 bit. Dois módulos subtratores de 4 bits são instanciados e conectados em cascata, onde o bit de "empréstimo" de saída (*Borrow-out* ou `Cout`) do primeiro bloco é conectado ao bit de "empréstimo" de entrada (*Borrow-in* ou `Cin`) do segundo. O módulo produz o resultado final de 8 bits (`sub`) e o bit de empréstimo final (`sub_cout`). Este bit `sub_cout` é então enviado ao módulo `FLAGS8bit` para determinar se ocorreu um *overflow* ou *underflow* na operação.

A Tabela 2 apresenta a tabela verdade do **subtrator completo (full subtractor)**, que realiza a subtração de três bits: A, B e Cin (ou Borrow-in).

A	B	Cin	S (Diferença)	Cout (Borrow)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabela 2 – Tabela verdade do subtrator completo (Full Subtractor)

- **S (diferença):**

$$S = A \oplus B \oplus Cin$$

- **Cout (borrow out):**

$$Cout = \overline{A}B + \overline{A}Cin + B \cdot Cin$$

2.0.3 Multiplicação

A multiplicação é implementada pelo módulo `MULT8bit` e escala o método "shift-and-add" (deslocar e somar). O processo gera oito produtos parciais de 8 bits (em vez de quatro), onde cada produto parcial `pp_i` é o resultado de um AND entre o multiplicando `A` (8 bits) e um bit do multiplicador, `B[i]`. Cada produto parcial é então deslocado i posições para a esquerda (equivalente a multiplicar por 2^i) e, em seguida, todos os oito produtos parciais deslocados são somados.

Como a multiplicação de 8x8 bits pode gerar um resultado de até 16 bits, o módulo `MULT8bit` fornece os 8 bits menos significativos como o `Result` (`mult_result`) e utiliza os 8 bits mais significativos para gerar a flag `Overflow` (`mult_overflow`), que sinaliza quando o resultado verdadeiro excede a capacidade de 8 bits.

2.0.4 Divisão

O módulo `DIV8bit` implementa a divisão inteira usando o algoritmo de subtrações sucessivas, escalando a lógica do projeto anterior. A lógica interna subtrai repetidamente o divisor (`segundo_operando`) do dividendo (`primeiro_operando`), contando o número de subtrações bem-sucedidas para determinar o quociente.

O módulo de 8 bits calcula três saídas: o `Quotient` (quociente), o `Remainder` (resto) e um bit de `Error` (erro). No módulo `ULA8bit_PRINCIPAL.v`, apenas o `Quotient` (`div_quotient`) é selecionado como o resultado final da ULA, e o sinal `Error` (`div_error`) é enviado ao `FLAGS8bit` para sinalizar uma divisão por zero.

O `Remainder` é calculado internamente, mas sua saída não é conectada a nenhuma lógica no módulo principal, sendo efetivamente descartado.

2.0.5 Mux4to1

Um multiplexador (MUX) é um dispositivo que permite que informações digitais de diversas fontes sejam encaminhadas para uma única linha para serem transmitidas nessa linha para um destino comum. Um multiplexador básico tem várias linhas de entrada de dados e uma única linha de saída. Ele também possui entradas de seleção de dados, as quais permitem que os dados digitais de quaisquer entradas sejam comutados para a linha de saída. Os multiplexadores também são conhecidos como seletores de dados (FLOYD, 2007).

Selector[1]	**Selector[0]**	**Entrada selecionada**	**Saída Y**
0	0	A	Y = A
0	1	B	Y = B
1	0	C	Y = C
1	1	D	Y = D

Tabela 3 – Tabela verdade do multiplexador 4:1

2.0.6 Mux2to1

Selector	**Entrada selecionada**	**Y (Saída)**
0	A	Y = A
1	B	Y = B

Tabela 4 – Tabela verdade do multiplexador 2:1

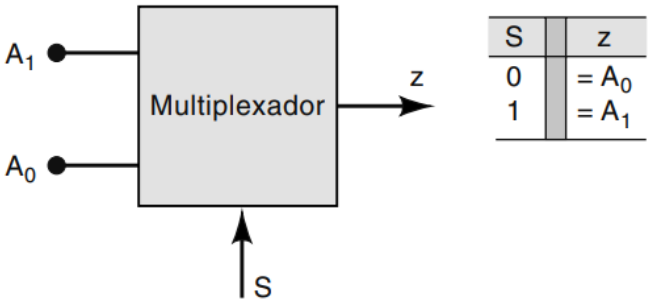

Figura 5 – Representação de um multiplexador. (TOCCI, 2011)

2.0.7 Operações Lógicas

Todas as operações lógicas são "bitwise", ou seja, atuam de maneira independente e paralela em cada um dos 8 bits dos operandos. De maneira semelhante aos módulos aritméticos, eles são organizados de forma hierárquica, utilizando dois módulos de 4 bits (por exemplo, AND4bit) para compor o módulo final de 8 bits (por exemplo, AND8bit).

- **OP = 010: OR (OU Lógico)**
  O módulo `OR8bit` realiza a operação “OU” bit a bit. Para cada posição, o bit de saída será 1 se o bit correspondente no `primeiro_operando` ou no `segundo_operando` for 1.

- **OP = 011: AND (E Lógico)**
  O módulo `AND8bit` realiza a operação "E" bit a bit. O bit de saída em cada posição só será 1 se o bit correspondente no `primeiro_operando` e no `segundo_operando` for 1.

- **OP = 100: XOR (OU Exclusivo)**
  O módulo `XOR8bit` realiza a operação "OU Exclusivo" bit a bit. O bit de saída será 1 apenas se os bits de entrada correspondentes forem diferentes (um 0 e um 1).

- **OP = 111: NOT (Negação)**
  Esta é a única operação lógica unária, implementada pelo `NOT8bit`. Ela ignora o `segundo_operando` e simplesmente inverte todos os bits do `primeiro_operando`, transformando cada 0 em 1 e cada 1 em 0.

A	B	$AORB$	$AANDB$	$AXORB$	NOT A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Tabela 5 – Tabela verdade para operações lógicas bit a bit

2.0.8 Flags de Estado

A ULA utiliza quatro flags de estado para fornecer informações contextuais cruciais sobre o resultado da operação mais recente. O design destas flags é um processo de duas etapas para garantir que apenas valores estáveis e corretos sejam exibidos.

Primeiro, um módulo puramente combinacional, `FLAGS8bit`, calcula os valores das flags (`overflow_calc`, `zero_calc`, etc.) em tempo real, com base no resultado da operação (`Result`) e em sinais auxiliares como `sum_cout`, `mult_overflow` e `div_error`.

Segundo, esses valores "calculados"não são enviados diretamente às saídas. Em vez disso, eles alimentam a entrada de quatro flip-flops Tipo D separados (`reg_overflow`, `reg_zero`, `reg_negative`, `reg_error`). Estes flip-flops são habilitados (`en`) apenas pelo sinal `enable_resultado`, que pulsa uma única vez quando a máquina de estados entra no estado final '11'. Isso garante que as flags nos LEDs sejam atualizadas apenas no exato momento em que o resultado final é salvo, evitando qualquer instabilidade ou "flicker"nos LEDs.

As quatro flags registradas são então exibidas nos LEDs `LEDR[11:8]`.

- **Flag de Overflow (saída `LEDR[8]`):** Armazenada no registrador `overflow_reg`, esta flag indica que o resultado de uma operação aritmética (soma, subtração ou multiplicação) excedeu a capacidade de representação de 8 bits. A sua lógica de cálculo é a mais complexa, pois depende do código da operação (`Op`) e dos sinais de "vai-um"ou "estouro"específicos de cada módulo aritmético (`sum_cout`, `sub_cout`, `mult_overflow`).

- **Flag de Zero (saída `LEDR[9]`):** Armazenada em `zero_reg`, esta é uma flag simples que é ativada (vai para 1) se, e somente se, o resultado final da operação (`Result`) for exatamente zero (`8'b00000000`).

- **Flag Negativa (saída `LEDR[10]`):** Armazenada em `negative_reg`, esta flag indica se o resultado é um número negativo. Na representação em complemento para 2, isso é determinado simplesmente verificando o bit mais significativo (MSB) do resultado. Se `Result[7]` for 1, a flag é ativada.
- **Flag de Erro (saída `LEDR[11]`):** Armazenada em `error_reg`, esta flag sinaliza que uma operação ilegal foi tentada. No projeto atual, ela é diretamente ligada ao sinal `DivError` vindo do módulo `DIV8bit`, sendo ativada especificamente quando o usuário tenta executar uma **divisão por zero**.

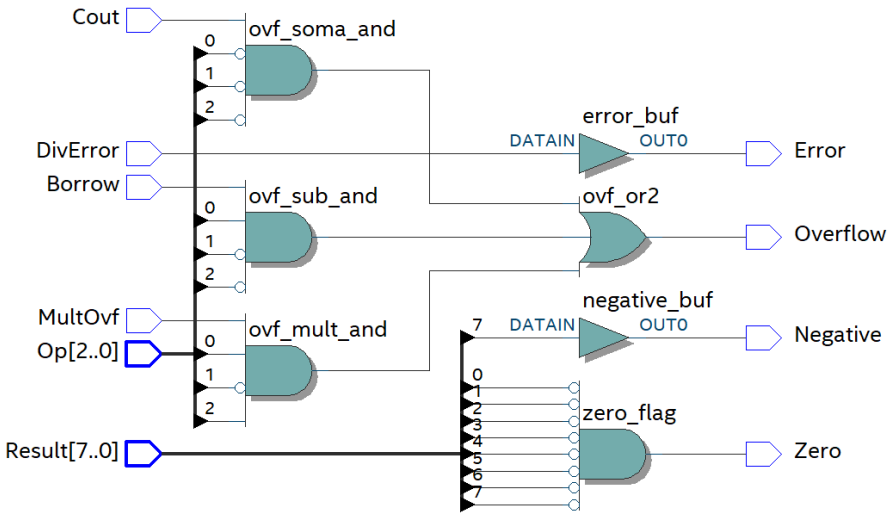

Figura 6 – Circuito geral das Flags. Fonte: o autor

3 Resultados e Discussões

Do ponto de vista técnico, o projeto foi estruturado em cinco módulos principais, sendo eles: a ULA de 8 bits, o contador de estados, o controlador do display, o registrador de resultados e o decodificador para o display de 7 segmentos.Cada um desses módulos desempenha uma função específica dentro do sistema, operando a partir de suas entradas e saídas. A interação entre eles ocorre por meio de declarações de portas lógicas e da instanciação de módulos secundários, o que garante a integração e o correto funcionamento do circuito como um todo.

3.1 ULA Principal de 8 bits

Gerencia uma máquina de estados para registrar dois operandos de 8 bits (A e B) e um código de operação (OP) de 3 bits, provenientes de chaves (SW). As entradas de botões (KEY) são tratadas com um módulo debounce e controlam o avanço dos estados e o reset.

A ULA é capaz de realizar 8 operações diferentes (Soma, Subtração, OR, AND, XOR, Multiplicação, Divisão e NOT), que são instanciadas como módulos separados (ex: Addition8bit, SUB8bit, MULT8bit, etc.). Um multiplexador (MUX8x8) seleciona o resultado correto com base no código de operação registrado.

Um módulo FLAGS8bit calcula as flags de Overflow, Zero, Negativo e Erro (como divisão por zero) com base na operação e no resultado.

O módulo permite operações encadeadas. Após a primeira operação, o resultado anterior ($resultado_anterior$) é usado como o primeiro operando para o cálculo seguinte.

3.2 Contador de Estados

Este módulo implementa a máquina de estados de 4 fases (2 bits) que controla a ULA. Ele avança para o próximo estado (ex: $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$) a cada pulso de `key_advance`. Os estados habilitam o registro sequencial de A, depois B, em seguida o código de operação (OP) e, por fim, habilitam o registro do resultado e das *flags*. Possui um *reset* assíncrono ativo-baixo.

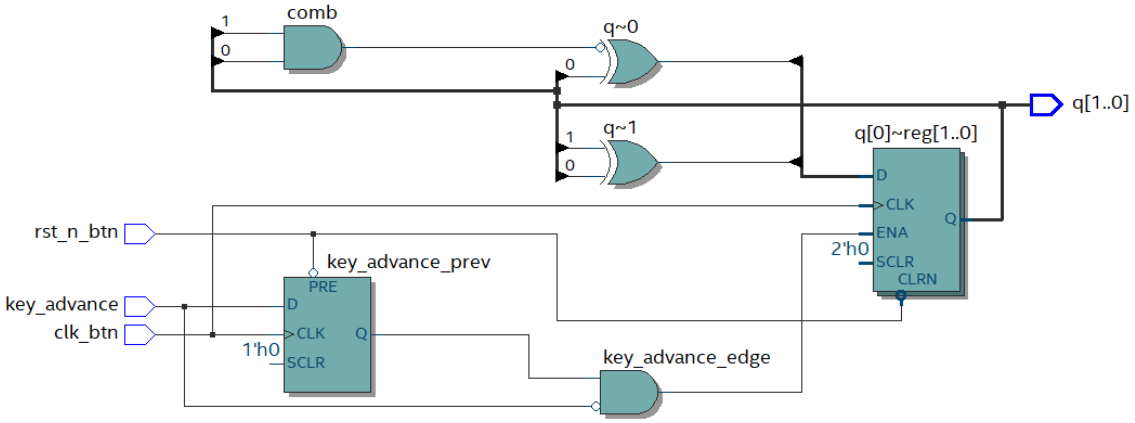

Figura 7 – Circuito geral do Contador de Estados. Fonte: o autor

3.3 Controlador do Display

Este módulo é responsável por converter o resultado binário de 8 bits da ULA para ser exibido em três displays de 7 segmentos. Ele recebe o resultado (bin) e um seletor de 2 bits (sel). O seletor determina o formato de exibição: Octal (00), Hexadecimal (01) ou Decimal (10). Ele instancia módulos de conversão específicos (binToOct, binToHex, binToBcd) para gerar os dígitos de unidade, dezena e centena em cada base. Além disso, utiliza um conjunto de multiplexadores 4-para-1 (mux4to1) para selecionar qual conjunto de sinais (Octal, Hex ou Decimal) será enviado para as saídas dos displays (segmentsUn, segmentsDez, segmentsCen).

3.4 Decodificador de 7 Segmentos

Um módulo combinacional que converte um número de 4 bits para os 7 sinais de um display. Recebe uma entrada binária de 4 bits e ativa as saídas (seg) correspondentes para formar os dígitos de 0 a 9 e A a F. A lógica de saída (usando portas OR) é projetada para displays de ânodo comum (ativos em nível baixo).

3.5 Registrador de Resultados

Armazena o resultado final da ULA. É um registrador de 9 bits que armazena o resultado da operação (8 bits) e a flag de overflow (1 bit). Ele é habilitado (en) para

carregar o novo valor apenas no estado final da máquina de estados.

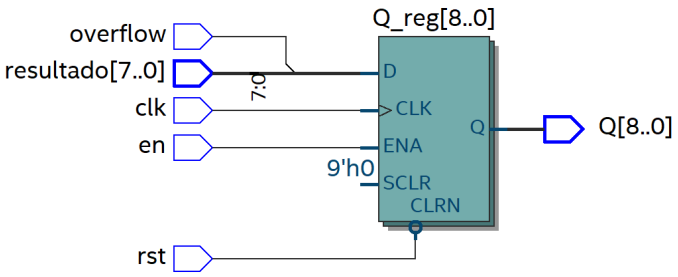

Figura 8 – Circuito do Registrador de Resultados. Fonte: O autor.

4 Conclusão

Com base no que foi apresentado e levando em conta os testes realizados, observa-se que o protótipo da Unidade Lógica Aritmética (ULA) de 8 bits foi elaborado de forma satisfatória, pois os requisitos fundamentais foram cumpridos. Os requisitos incluem: a entrada de operandos (A e B) e do código de operação (OP) por meio de chaves (SW); o controle da máquina de estados (avanço e reset) por meio de botões (KEY); a execução de 8 operações aritméticas e lógicas diferentes; e a apresentação do resultado final nos LEDs (LEDR) e em vários formatos (Octal, Hexadecimal e Decimal) em três displays de 7 segmentos.

Assim, para resolver o problema, foi criado um sistema digital composto por circuitos sequenciais e combinacionais. Para isso, foram empregados recursos como flip-flops para registrar os operandos, o estado da operação e as flags; um contador de estados para gerenciar a entrada de dados; e multiplexadores para escolher o resultado da operação e o formato de exibição corretos.

Nesse sentido, o produto pode ser considerado eficaz, mas é importante ressaltar que ele pode ser aprimorado. Ademais, a operação de divisão atualmente apresenta apenas o quociente. Uma possível melhoria futura poderia incluir uma opção para exibir também o resto, que atualmente é calculado, mas não mostrado. Isso resultaria em um desempenho e usabilidade aprimorados do protótipo.

Referências

CAPUANO, F. G.; IDOETA, I. V. *Elementos de Eletrônica Digital*. 42. ed. São Paulo: Érica, 2019. ISBN 9788536530406. Citado na página 4.

FLOYD, T. *Sistemas Digitais: Fundamentos e Aplicações*. Bookman, 2007. ISBN 9788560031931. Disponível em: <https://books.google.com.br/books?id=eLXpwAEACAAJ>. Citado 2 vezes nas páginas 4 e 10.

TOCCI, R. J. *Sistemas Digitais - Princípios e Aplicações*. 11. ed. [S.l.]: Pearson Prentice Hall, 2011. Citado 4 vezes nas páginas 2, 3, 7 e 10.