RELATÓRIO PROBLEMA II - MI DE CIRCUITOS DIGITAIS

Nícolas do Carmo Mota Andrade[1]
Walef Souza da Silva[2]

RESUMO

Relatório apresentado como requisito parcial para obtenção da nota da disciplina MI de Circuitos Digitais, vinculada ao colegiado de Engenharia de Computação da Universidade Estadual de Feira de Santana (UEFS), sob a orientação do professor Anfranserai Morais Dias. O presente documento descreve o desenvolvimento de uma Unidade Lógica Aritmética integrada a um sistema sequencial baseado na Notação Reversa Polonesa (RPN) em linguagem de descrição de hardware Verilog para aplicação em placa de circuito lógico programável (FPGA).

Palavras-chave: ULA, Circuitos Digitais, FPGA, RPN, Verilog, Memória.

1. INTRODUÇÃO

A evolução da eletrônica digital, impulsionada desde a invenção do transistor, estabeleceu os Circuitos Digitais como a base fundamental dos computadores modernos. Nesse contexto, a Unidade Lógica e Aritmética (ULA) destaca-se como um elemento central em processadores e microcontroladores, responsável pela execução das operações lógicas e aritméticas. A prototipação de sistemas baseados em ULAs, partindo de arquiteturas simples para um sistema mais complexo envolvendo 8 bits, permitindo a exploração não apenas de operações básicas, mas também funcionalidades avançadas e o uso de memória.

O desafio na construção de calculadoras digitais não reside apenas na execução de operações matemáticas, mas sim na implementação de um ecossistema que permita gerenciar múltiplas entradas e a seleção de dados. Este projeto aborda essa questão através da implementação de uma calculadora baseada na Notação Polonesa Reversa (RPN). A RPN, também conhecida como notação pós-fixada, é uma sintaxe formal que permite a especificação de expressões matemáticas de forma inequívoca, notavelmente eliminando a necessidade de parênteses. Sob essa perspectiva, a lógica RPN está intrinsecamente relacionada a uma "pilha operacional" (*operational stack*), em que os últimos valores a entrar são os primeiros a sair, onde os operando são armazenados temporariamente por registradores.

A implementação de uma calculadora RPN em um circuito lógico programável (FPGA) provoca uma mudança de paradigma de desenvolvimento, migrando de projetos puramente combinacionais para sistemas sequenciais. O sistema exige não apenas a ULA para executar os cálculos, mas também a implementação de pilhas, contadores, registradores e módulos auxiliares, essenciais para gerenciar as funcionalidade do RPN e integrar os diferentes níveis exigências requeridas para a inserção e processamento de operadores a medida que são inseridos.

O presente relatório tem como objetivo apresentar o projeto, a implementação em Verilog e a validação de uma ULA de 8 bits capaz de funcionar como o núcleo de uma calculadora RPN. O trabalho foi desenvolvido no âmbito da disciplina TEC 498 MI - Projeto de Circuitos Digitais , utilizando a placa DE10-Lite. O documento detalha a fundamentação teórica da RPN, a arquitetura modular do sistema, as simulações funcionais e os testes práticos realizados para validar as operações aritméticas, lógicas e a correta exibição dos resultados em múltiplas bases numéricas.

2. VISÃO MACRO DO PROJETO

Em primeiro plano, o projeto consiste em uma calculadora digital completa de 8 bits, implementada na plataforma FPGA DE10-Lite, com a lógica de entrada e operação baseada na Notação Polonesa Reversa (RPN) com a finalidade de realizar operações lógicas e aritméticas. Em que, diferente de um circuito combinacional, o sistema é arquitetado como um projeto sequencial síncrono, onde uma unidade de controle gerencia o fluxo de dados, a entrada de operandos e o armazenamento de resultados em registradores.

A interação do usuário com a calculadora é realizada através das chaves, botões, leds e displays da placa, em que a entrada de dados é realizada de forma manual pela base binária. O sistema utiliza as seguintes entradas e saídas físicas:

- **Entradas:**
    - Botões: **KEY[0]** para confirmação e avanço das etapas e **KEY[1]** para resetar o sistema.
    - Chaves (Seleção de Dados): **SW[7:0]** para a entrada dos operandos de 8 bits (A e B), **SW[3]** para carry-in ou borrow-in, **SW[2:0]** para selecionar a operação.
    - Chaves (Seleção de Base): **SW[9:8]** para definir a base de exibição do resultado (Decimal, Hexadecimal ou Octal).
- **Saídas (Displays):**

    - Display de Resultado: **HEX[2:0]** exibem o resultado da operação na base selecionada.
    - Display de Resto: **HEX[5:3]** exibem o resto para quando a operação for uma divisão.

- **Saídas (LEDs):**

    - LEDs de Flags: **LEDR[4:0]** indicam o estado das flags do sistema, sendo representados na respectiva ordem: carry-out ou borrow-out, overflow, zero, erro e resto da divisão.
    - LEDs de Passos: **LEDR[8:5]** indicam visualmente em qual etapa do processo RPN o usuário se encontra.

O funcionamento do sistema é gerenciado por um contador de etapas que define a fase da entrada RPN. O usuário posiciona os dados nas chaves e utiliza o botão ENTER para avançar o estado da calculadora. O fluxo de operação, que pode ser visualizado no diagrama de blocos do projeto, segue a seguinte ordem:

1. Passo 0 (Contador 00): Carrega o operando A, direcionando o valor inserido nas chaves para o Registrador A.

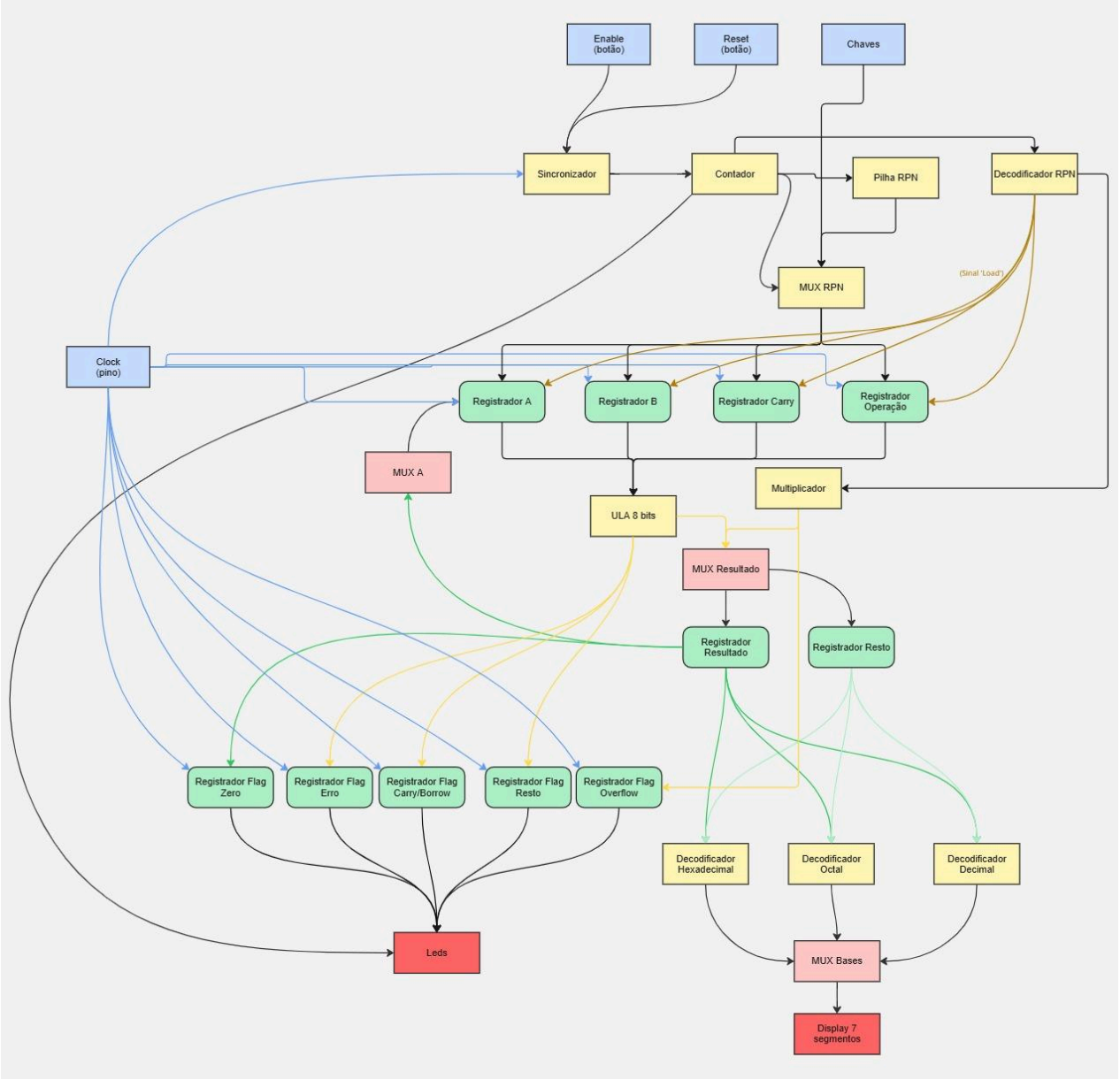
2. Passo 1 (Contador 01): Carrega o operando B, direcionando o valor inserido nas chaves para o Registrador B.

3. Passo 2 (Contador 10): Carrega a operação e o carry-in, direcionando o valor inserido nas chaves para seus respectivos registradores.

4. Passo 3 (Contador 11): Processa os dados dos registradores e carrega o resultado da ULA (ou do Multiplicador) para o registrador de resultado, que é exibido nos displays. O sistema então retorna automaticamente ao passo 0, permitindo que o resultado anterior seja usado como o novo operando A, mantendo a lógica da pilha RPN.

O diagrama a seguir apresenta a visão estrutural do funcionamento do circuito, detalhando a interconexão dos módulos. Em seguida, é ilustrada a integração física desses componentes com a placa DE10-Lite.

Figura 1 - Diagrama do Projeto



Fonte: compilação do autor

Figura 2 - Mapeamento físico da placa

Fonte: compilação do autor

3. FUNDAMENTAÇÃO TEÓRICA

Em projetos de sistemas digitais modernos, desde circuitos simples até complexos sistemas em FPGAs, fundamentam-se na interação de dois tipos principais de circuitos: combinacionais e sequenciais. Conforme descrito por Tocci, a distinção fundamental é a presença ou ausência de memória. O projeto anterior, focado em uma ULA de 4 bits, era puramente combinacional. Este projeto evolui para um sistema de 8 bits que introduz a lógica sequencial síncrona, que, segundo Chu, é a base de "praticamente todos os sistemas digitais úteis", pois permite o armazenamento de dados e a criação de máquinas de estado.

Os circuitos combinacionais formam a lógica de processamento do sistema, onde as saídas dependem exclusivamente das entradas atuais. Nesse sentido, eles são empregados na Unidade Lógica e Aritmética (ULA), que agrupa as operações de soma, subtração, multiplicação, divisão e lógicas (AND, OR, XOR, NOT); nos Multiplexadores (MUX), usados como seletores de dados, que a partir de sinais de controle, determina qual entrada será conectada à saída; e nos Decodificadores, que convertem os resultados binários para o formato visual dos displays de 7 segmentos.

A implementação da lógica RPN, no entanto, exige uma arquitetura com memória, fundamentada em circuitos sequenciais síncronos. O bloco de construção mais básico é o Flip-Flop Tipo D, um dispositivo que armazena um único bit. Sua principal característica, segundo Tocci, é capturar o valor da entrada (D) *apenas* na borda de subida do *clock*, mantendo esse valor até o próximo ciclo. Um conjunto de Flip-Flops D operando em paralelo forma um Registrador Paralelo-Paralelo, que segundo Chu, é o principal componente para reter dados em um sistema. No contexto desta calculadora, os registradores são usados para implementar a "pilha operacional", armazenando os operandos (A e B) e o último resultado.

O gerenciamento dessa pilha e do fluxo de operações é ditado pela Notação Polonesa Reversa (RPN). A RPN é uma sintaxe pós-fixada que elimina a ambiguidade de expressões matemáticas e a necessidade de parênteses. Sua implementação em hardware exige uma estrutura de uma pilha, em que o último a entrar é primeiro a sair para os operandos, que é naturalmente implementada pelos registradores. A sequência de "empilhar" dados e "executar" operações é governada por uma Máquina de Estados Finito (FSM). Essa FSM é implementada usando um Contador Síncrono, que conforme Tocci, garante que todas as transições de estado ocorram simultaneamente com o *clock*, e um decodificador de controle que dita as ações em cada etapa.

O processo de design de cada um desses blocos, sejam combinacionais ou sequenciais, seguiu um fluxo metodológico clássico. O ponto de partida é a Tabela-Verdade, a especificação formal que descreve como a saída de um circuito lógico depende dos níveis lógicos presentes nas entradas. A partir dela, deriva-se uma Expressão Lógica (Booleana) que, para fins de otimização, é simplificada. O Mapa de Karnaugh foi empregado como o método gráfico de sistematização para obter a expressão mínima, garantindo um circuito final mais eficiente e confiável.

Com as equações mínimas, o Desenho do Circuito (diagrama lógico) é elaborado, servindo como a planta baixa de interconexão das portas. Em sequência, a implementação é realizada em Verilog Estrutural. Esta abordagem, de acordo com Chu, é ideal para construir um sistema grande a partir de componentes mais simples, pois é essencialmente uma descrição textual de um projeto hierárquico e modular que reflete fielmente o diagrama de blocos.

4. DESCRIÇÃO DA SOLUÇÃO E IMPLEMENTAÇÃO DOS MÓDULOS

Para atender aos requisitos do problema, que exigem a implementação de uma calculadora de 8 bits baseada na lógica RPN, a arquitetura proposta evolui de um sistema puramente combinacional para um projeto sequencial síncrono. Esta abordagem é fundamental, pois a RPN exige o gerenciamento de estado e memória para armazenar operandos inseridos em momentos distintos e para reter o resultado da operação anterior. A solução adotada, portanto, baseia-se na arquitetura clássica

de processadores, dividindo o sistema em duas unidades principais: a Unidade de Controle e o Caminho de Dados.

A Unidade de Controle é implementada como uma Máquina de Estados Finitos (FSM) e é responsável por governar todo o fluxo de operação da calculadora. Ela é composta pelos módulos de sincronização de botões (botaosincronizado), que filtram os sinais assíncronos de RESET e ENTER; pelo contador de etapas (ContadorRPN), que armazena o estado atual do ciclo RPN (00, 01, 10 ou 11); e pelo decodificador de controle (DecodificadorRPN), que atua como a lógica central da FSM, gerando todos os sinais de habilitação (LoadA, LoadB, LoadOp, LoadResultado) e seleção que comandam o percurso dos dados.

O caminho de dados multiplexados agrupa todos os módulos de armazenamento e processamento que manipulam os dados de 8 bits. A Unidade de Armazenamento é composta por um banco de Registradores (registrador8x8, registrador1x1) que implementam a "pilha operacional", retendo os operandos RegA e RegB, o RegOp e o RegResultado . A lógica da pilha é gerenciada pelo módulo pilhaRPN, que decide se o RegA deve ser carregado com um novo dado das chaves SW ou com o valor do RegResultado anterior.

A Unidade de Processamento recebe os dados dos registradores RegA e RegB. Ela é composta por dois tipos de módulos de cálculo: a ula8bits combinacional, que executa operações instantâneas (soma, subtração, divisão, lógicas); e o multiplicador8x8 sequencial, que requer múltiplos ciclos de clock para operar. O decodificadorRPN gerencia essa diferença: para a ULA, o resultado é imediato; para o multiplicador, o controle ativa o StartMult, pausa o contadorRPN e aguarda o sinal ProntoMult antes de prosseguir. Um Multiplexador de Resultado seleciona a saída correta (SUla ou ResultadoMuli) para ser enviada ao RegResultado.

Por fim, a Unidade de Interface traduz os dados armazenados para o usuário. O displayRPN é um bloco conversor que recebe o RegResultado e o RegRestoDiv, converte o binário de 8 bits para a base selecionada Decimal (via Double Dabble), Octal ou Hexadecimal e aciona os seis Displays de 7 Segmentos. Paralelamente, a lógica de Flags (flagzero, flagcout, etc.) e os LEDs de passos leem os sinais do sistema para fornecer um apoio visual imediato sobre o resultado e o estado da máquina.

4.1 UNIDADE DE CONTROLE

4.1.1 Módulo de Sincronização de Botões

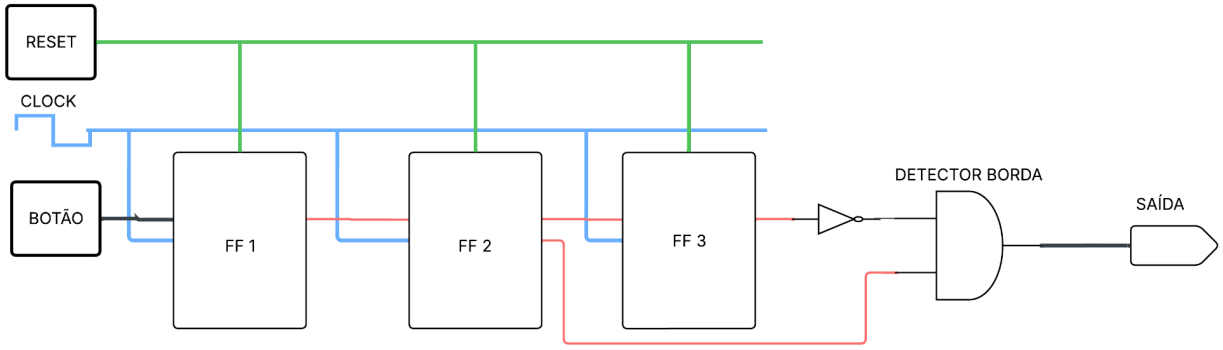
O módulo **botaosincronizado** tem como finalidade adaptar e estabilizar sinais provenientes de botões mecânicos utilizados na placa DE10-Lite. Em sistemas digitais síncronos, esses sinais estão sujeitos a dois efeitos indesejáveis:

metaestabilidade, decorrente da amostragem assíncrona em relação ao clock do sistema, e *bounce* (ou repique), causado pelas oscilações mecânicas durante o acionamento do botão. Esse módulo é primariamente um sincronizador com detector de borda. Ele não é um *debouncer* no sentido clássico (com temporizador), mas a combinação da sincronização e detecção de borda frequentemente é suficiente para lidar com o bounce de botões em FPGAs rodando a clocks rápidos, especialmente se é preciso de apenas um pulso único no momento em que o botão é pressionado.

O circuito começa invertendo o sinal do botão, já que na DE10-Lite os botões são ativos em nível baixo (ou seja, o sinal vai para 0 quando o botão é pressionado). Essa inversão transforma o sinal em ativo-alto, o que facilita o tratamento lógico dentro do sistema. Depois disso, o sinal passa por dois flip-flops do tipo D ligados em sequência. O primeiro flip-flop tem a função de eliminar a metaestabilidade, que pode ocorrer quando o botão é pressionado entre bordas de clock. O segundo flip-flop garante que o sinal resultante esteja totalmente sincronizado com o clock, evitando instabilidades. Em seguida, há um terceiro flip-flop que guarda o valor do ciclo anterior. Com isso, o circuito consegue comparar o estado atual do botão com o estado passado e identificar uma transição de 0 para 1. Quando essa transição é detectada, a saída gera um pulso de um ciclo de clock, indicando que o botão foi pressionado.

Na prática, isso significa que cada toque no botão gera apenas um pulso limpo e estável, sem repiques e totalmente alinhado com o clock do sistema, permitindo que outros módulos digitais reconheçam o comando corretamente.

Figura 3 - Diagrama do circuito sincronizador



Fonte: compilação do autor

4.1.2 Contador de Etapas

O módulo **contadorRPN** implementa um contador binário síncrono de 2 bits, utilizado para controlar as quatro etapas principais de operação da calculadora

RPN. Esse contador define em qual fase o sistema está — por exemplo, leitura de operandos, execução da operação ou exibição do resultado — e avança de forma ordenada a cada pulso de clock.

O circuito é formado por dois flip-flops tipo D conectados de forma a criar uma lógica de *toggle*. A entrada de cada flip-flop é controlada por uma operação XOR, o que faz com que o valor de saída mude (ou "toggle") no momento certo para gerar a sequência binária correta. Dessa forma, o contador percorre os estados **00**, **01**, **10** e **11**, repetindo o ciclo continuamente enquanto o sistema estiver habilitado.

4.1.3 Decodificador de Controle

O módulo **decodificadorRPN** é o responsável por coordenar o funcionamento geral da calculadora, gerando os sinais de controle que determinam quando cada registrador deve ser carregado e quando o contador de etapas deve avançar. Ele analisa o passo atual do sistema (a partir dos dois bits menos significativos do contador) e o tipo de operação selecionada (RegOp[2:0]), ajustando o comportamento conforme a necessidade.

Nas operações combinacionais, como soma, subtração e lógicas, o decodificador libera os sinais de carga (LoadA, LoadB, LoadOp, LoadCarry e Resultado) conforme o botão *Enter* é pressionado, permitindo que o contador avance normalmente entre os quatro passos de execução. Quando a operação detectada é uma multiplicação (RegOp = 010), o módulo ativa o sinal StartMult para iniciar o multiplicador e entra em um modo de espera (aguardandoMult). Nessa condição, o sinal Enable do contador é desativado, impedindo que o sistema avance até que o multiplicador indique o término do cálculo (ProntoMult = 1). Assim que o resultado está pronto, o decodificador libera novamente o avanço do sistema e carrega o valor final.

Esse mecanismo garante que operações rápidas e operações sequenciais funcionem de forma sincronizada, evitando conflitos de tempo e garantindo a execução correta de todas as instruções.

4.2 UNIDADE DE ARMAZENAMENTO

4.2.1 Registradores

Os registradores são elementos de memória sequencial que armazenam temporariamente os dados durante o processamento da calculadora RPN. O sistema utiliza flip-flops tipo D organizados em bancos paralelos para criar registradores de diferentes larguras (1 bit e 8 bits), todos sincronizados pelo clock do sistema e controlados por sinais de *enable* individuais (*LoadA, LoadB, LoadOp, LoadCarry, LoadResultado*).

São utilizados Registradores de 8 bits para armazenar valores relacionados aos operadores A e B, ao resultado da operação e ao resto da divisão. O módulo **registrador1x1** é usado em 3 instâncias para armazenar o código da operação e de forma única para o Carry-in e as flags Cout/Bout, Zero, Overflow, Resto e Erro.

4.2.2 Lógica da Pilha

O módulo pilhaRPN implementa o comportamento característico da notação polonesa reversa: a capacidade de reutilizar automaticamente o resultado de uma operação como operando da operação seguinte, sem necessidade de reinserção manual. Este módulo atua como um controlador inteligente que decide, a cada nova operação, se o operando A deve vir das chaves de entrada ou do resultado armazenado anteriormente.

A implementação utiliza um flip-flop tipo D como elemento de memória para criar uma "flag de estado" que persiste através de múltiplas operações, lembrando que existe um resultado disponível para ser reutilizado. Esta flag só é resetada quando o usuário pressiona explicitamente o botão de reset manual, permitindo cadeias infinitas de operações.

O circuito da **pilha RPN** é dividido em três blocos principais, que trabalham em conjunto para decidir quando o próximo operando deve vir das chaves de entrada ou do resultado anterior armazenado.

- **Detector de Passo 00:**

Este bloco identifica quando o sistema está no passo de carregamento do operando A (Etapa[1:0] = 00). Somente nesse instante o circuito precisa decidir qual será a fonte do próximo valor a ser carregado — o que torna essa etapa o ponto de controle da pilha.

- **Flip-Flop de Memória:**

Armazena a informação sobre se o sistema já executou pelo menos uma operação. A entrada D do flip-flop recebe o resultado de uma porta OR que combina sua própria saída (realimentação) e o sinal LoadResultado, que indica o fim de uma operação. Essa configuração cria um comportamento de “trava”: assim que uma operação é concluída, o flip-flop mantém a flag em nível alto até que ocorra um reset manual.

- **Lógica de Decisão:**

A decisão sobre usar o resultado anterior depende de três condições: o sistema precisa estar no passo 00, já deve existir um resultado válido (flag = 1) e

não pode estar em estado de reset. Quando todas essas condições são verdadeiras, o circuito habilita o sinal MuxSelA, que controla os oito multiplexadores 2:1 responsáveis por selecionar, bit a bit, entre o valor das chaves de entrada ou o resultado anterior armazenado.

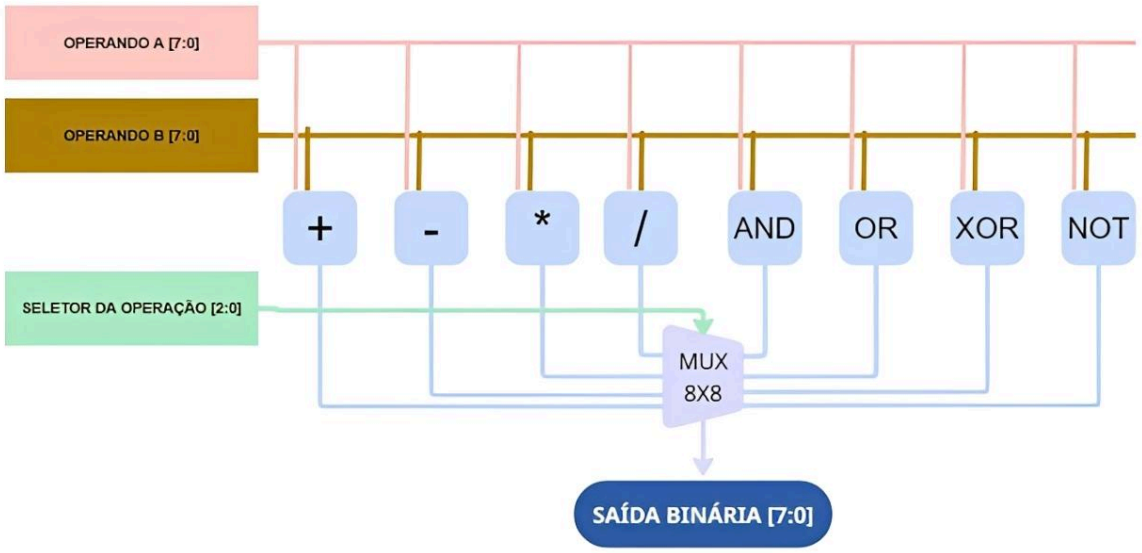
Com esse conjunto de blocos, a pilha RPN garante que os resultados de operações anteriores possam ser reutilizados automaticamente em novos cálculos, sem a necessidade de reinserir manualmente os operandos. Isso torna o fluxo de operações contínuo e reflete o comportamento típico de uma calculadora RPN real.

4.3 UNIDADE DE PROCESSAMENTO

4.3.1 ULA Combinacional

O módulo **ula8bits** implementa a Unidade Lógica e Aritmética combinacional, responsável por executar operações aritméticas (soma, subtração, divisão) e lógicas (AND, OR, XOR, NOT) sobre operandos de 8 bits. A ULA opera de forma puramente combinacional (como deve ser por definição), processando todas as operações em paralelo e utilizando um multiplexador 8x8 para selecionar o resultado desejado. Este design permite que o resultado esteja disponível instantaneamente (sujeito apenas aos atrasos de propagação das portas lógicas), sem necessidade de ciclos de clock adicionais. Vale ressaltar que a operação de multiplicação é constituída por um circuito sequencial, portanto, realizada fora dessa ULA.

Figura 4 - Representação da ULA



Fonte: compilação do autor

4.3.1.1 Somador 8x8

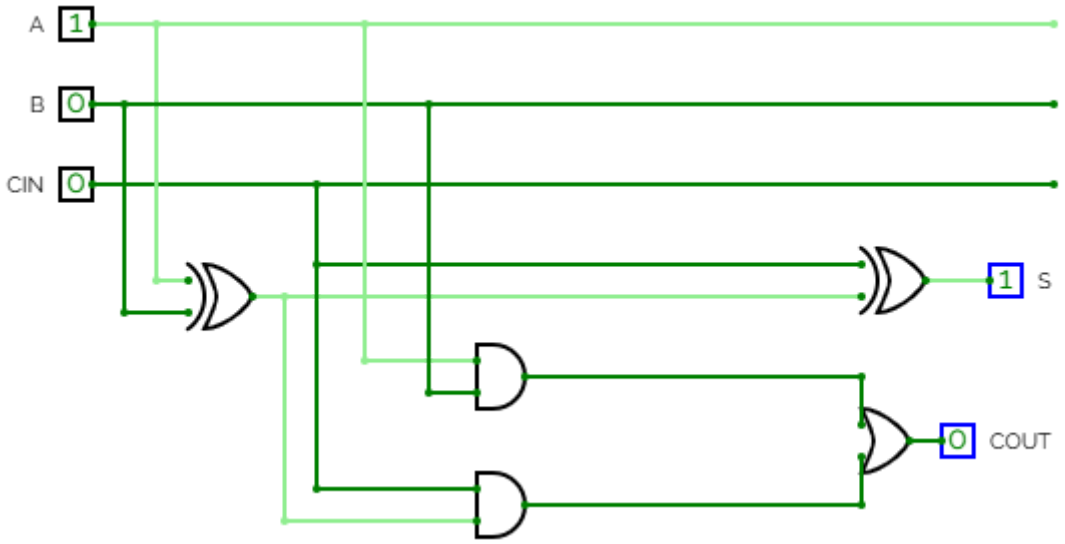
O módulo **somador8x8** realiza a soma binária de dois operandos de 8 bits usando uma cadeia de somadores completos no formato *ripple-carry*. O carry se propaga do bit menos significativo ao mais significativo, o que torna o circuito simples, porém com tempo de resposta proporcional ao número de bits. O carry final (**Cout**) indica estouro em operações sem sinal.

Tabela 1 - Tabela Verdade Somador Completo

ENTRADA A	ENTRADA B	CARRY IN	SAÍDA	CARRY 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

Fonte: compilação do autor

Figura 5 - Circuito do Somador Completo



Fonte: compilação do autor

4.3.1.2 Subtrator 8x8

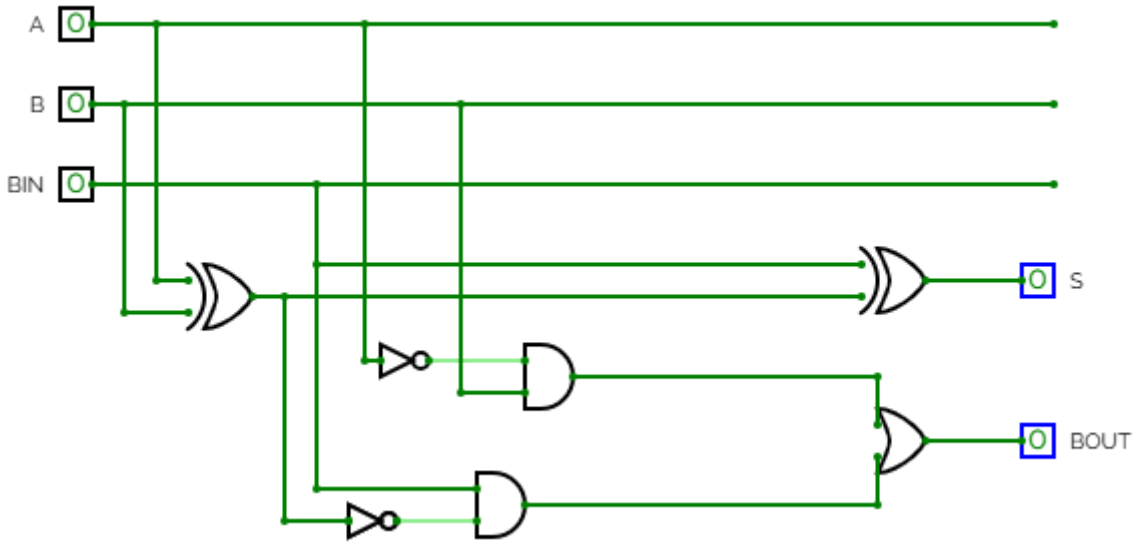
O **subtrator8x8** segue a mesma estrutura do somador, mas realiza a subtração bit a bit (**A - B**) por meio de subtratores completos em cascata. O *borrow* se propaga entre os estágios de forma semelhante ao carry, e o sinal final (**Bout**) indica quando ocorre *underflow* (A < B).

Tabela 2 - Tabela Verdade do Subtrator Completo.

ENTRADA A	ENTRADA B	BORROW IN	SAÍDA	BORROW OUT
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

Fonte: compilação do autor

Figura 6 - Circuito do Subtrator Completo



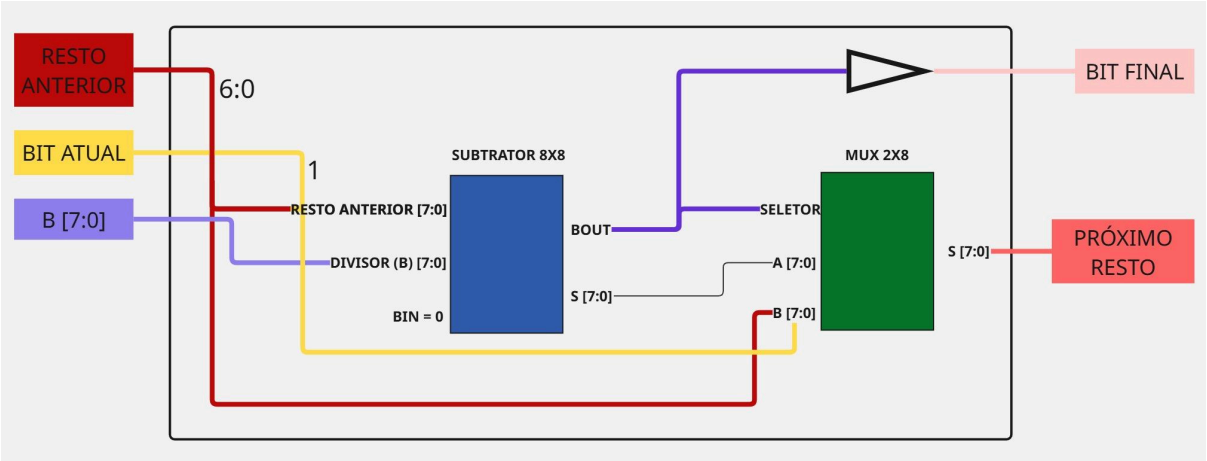
Fonte: compilação do autor

4.3.1.3 Divisor 8x8

O módulo **divisor8x8** executa a divisão inteira usando o algoritmo de divisão com restauração. Ele processa o dividendo da esquerda para a direita, tentando subtrair o divisor em cada estágio. Quando a subtração é possível, o bit do quociente é 1 e o resto é atualizado; caso contrário, o resto anterior é mantido e o bit do quociente recebe 0. Isso tudo é realizado com circuitos multiplexadores e subtratores.

O divisor8x8 é implementado hierarquicamente, sendo composto por uma cascata de 8 instâncias do módulo **estagiodivisao8x8**, conforme ilustrado na figura 7. Este diagrama representa visualmente um único passo do algoritmo de restauração.

Figura 7 - Circuito dos Estágios da Divisão



Fonte: compilação do autor

Em cada estágio, o circuito recebe o resto anterior e tenta subtrair o divisor usando o subtrator8x8. A lógica de decisão é controlada diretamente pelo sinal Bout (Borrow Out) do subtrator:

- Se Bout = 0 (subtração válida): Significa que o bit do quociente é 1 e o muxdivisor8x8 seleciona (S=0) o resultado da subtração como novo resto.

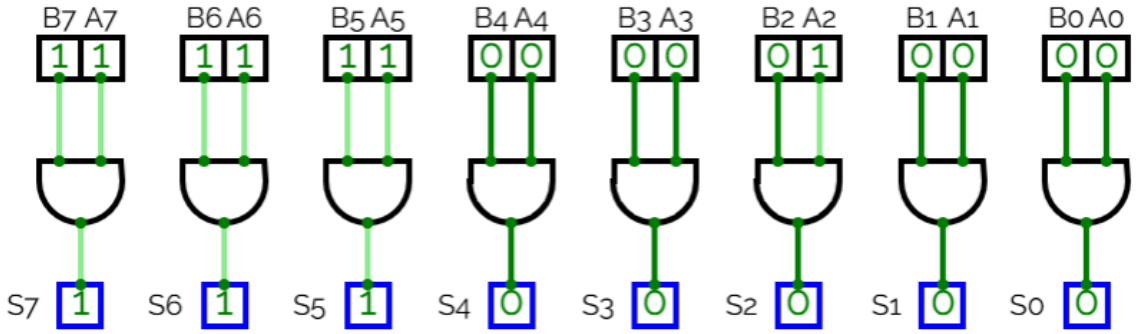
- Se Bout = 1 (subtração inválida): Significa que o bit do quociente é 0 e o muxdivisor8x8 seleciona (S=1) o resto anterior original, "restaurando" o valor, que se torna o novo resto.

Este novo resto é então deslocado logicamente e alimentado na entrada do resto anterior do próximo estágio em cascata, repetindo o processo para cada bit do quociente.

4.3.1.4 Operações Lógicas

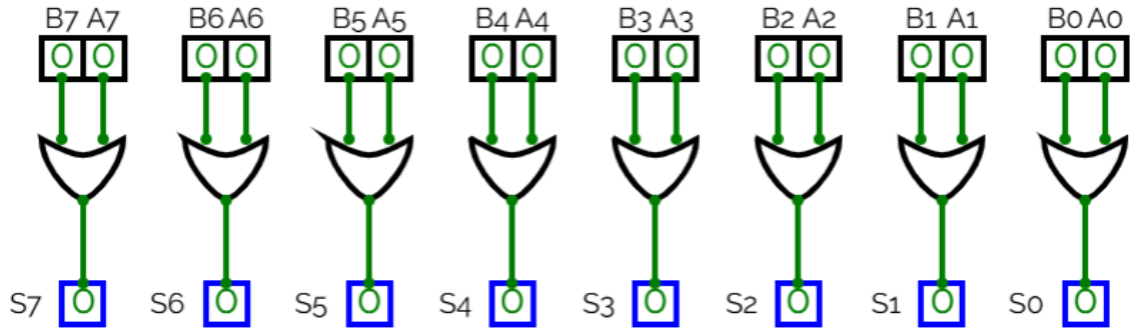
AND8bits, OR8bits, XOR8bits: Implementam operações bit a bit através de 8 portas lógicas em paralelo. Cada bit de saída depende apenas dos bits correspondentes das entradas (S[i] = A[i] operação B[i]), sem propagação de sinais entre bits.

Figura 8 - Diagrama do circuito And8bits



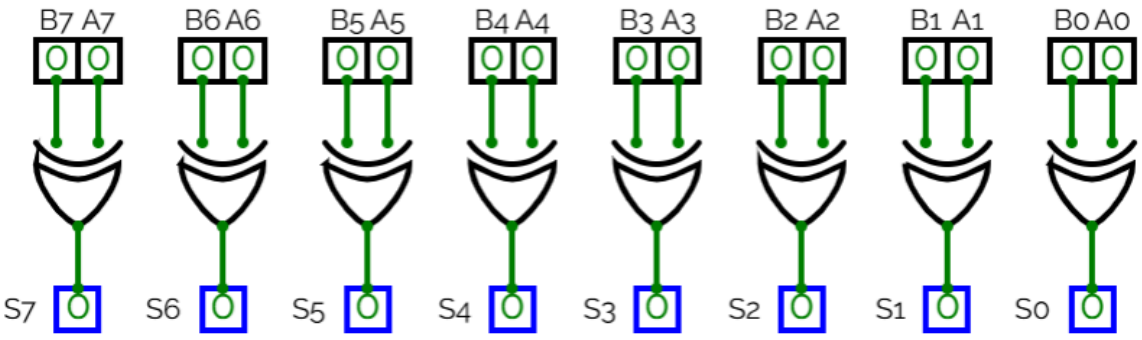
Fonte: compilação do autor

Figura 9 - Diagrama do circuito Or8bits



Fonte: compilação do autor

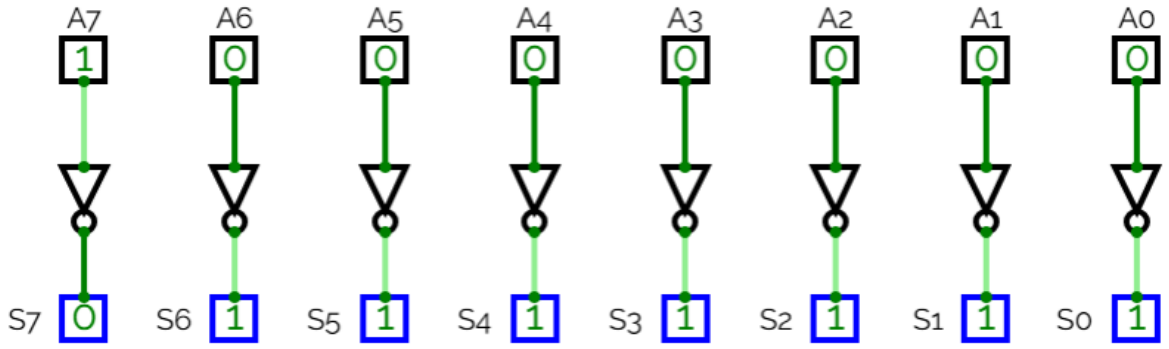
Figura 10 - Diagrama do circuito Xor8bits



Fonte: compilação do autor

NOT8bits: Opera exclusivamente sobre o operando A, invertendo cada bit (S[i] = ~A[i]). O operando B é ignorado.

Figura 11 - Diagrama do circuito Not8bits



Fonte: compilação do autor

4.3.1.5 Flags

A ULA gera sinais intermediários que alimentam módulos de flags especializados:

- **Flag Zero:** Porta NOR de 8 entradas detecta resultado 00000000, ativando quando todos os bits são zero.
- **Flag Cout/Bout:** Lógica condicional ativa Cout para soma (Sel=000) ou Bout para subtração (Sel=001), indicando carry/borrow de saída.
- **Flag Erro:** Ativa apenas para divisão (Sel=011) quando divisor B = 0, prevenindo resultados inválidos.

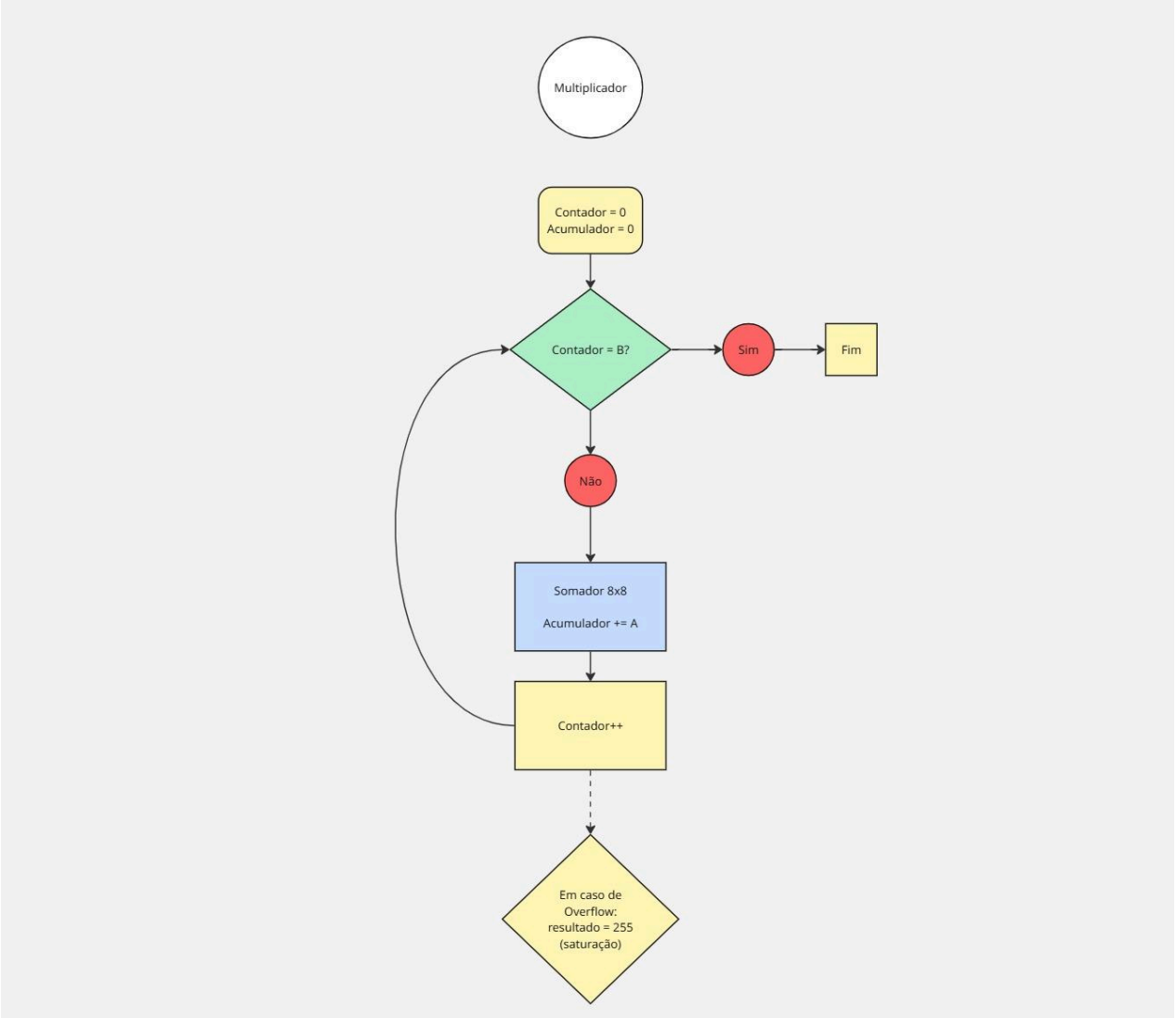
- **Flag Resto:** Porta OR de 8 entradas detecta resto não-zero na divisão, indicando resultado não-exato.

- **Flag Overflow:** Gerada pelo módulo multiplicador sequencial (não pela ULA), indica quando o produto excede o limite de 8 bits (resultado > 255), sinalizando truncamento.

As flags são registradas no passo 11 junto com o resultado e exibidas nos LEDs da placa.

4.3.2 Multiplicador

Figura 12 - Diagrama ilustrativo do circuito Multiplicador



Fonte: compilação do autor

O módulo **multiplicador8x8** realiza a multiplicação sequencial de dois operandos de 8 bits, produzindo um resultado acumulado ao longo de vários ciclos

de clock. O circuito é estruturado de forma modular, combinando blocos dedicados que controlam o avanço das operações, o armazenamento parcial dos resultados e a proteção contra overflow. O funcionamento parte do sinal START, que inicia a multiplicação e ativa o estado interno de operação (Operando_Q). A partir daí, um contador de 8 bits começa a incrementar, representando o número de somas já realizadas. O processo segue até que o valor do contador atinja o segundo operando (B), momento em que o módulo comparador8x8 detecta igualdade e sinaliza que o cálculo foi concluído (Pronto = 1).

Durante cada ciclo, o acumulador armazena o resultado parcial da soma entre seu próprio valor e o operando A, realizada pelo somador8x8 interno. Esse somador trabalha com arquitetura ripple-carry, garantindo simplicidade e operação estável a cada incremento. O acumulador é composto por um registrador de 16 bits, sendo os 8 bits mais significativos responsáveis por detectar eventuais estouros (overflow).

O controle de saturação é feito por meio da flag de saturação, que monitora os bits mais significativos do acumulador. Caso esses bits atinjam 1 em qualquer ponto da operação, a flag é ativada e permanece travada até o final do processo. No estágio final, o saturador8bits utiliza essa flag para limitar o resultado máximo em 255 (11111111), evitando que o valor de saída ultrapasse o intervalo representável por 8 bits.

Com isso, o multiplicador executa o produto A × B de forma iterativa, utilizando apenas operações de soma e controle sequencial. A arquitetura favorece simplicidade e uso eficiente de recursos lógicos, garantindo resultado correto com detecção automática de overflow.

4.4 UNIDADE DE INTERFACE E ROTEAMENTO

4.4.1 Multiplexadores de Dados

Nos sistemas digitais, muitas vezes é necessário escolher entre diversos sinais de entrada aquele que será encaminhado para a saída. Esse processo de seleção é essencial em arquiteturas de processamento modernas, já que permite otimizar o uso dos recursos de hardware e organizar o fluxo de informações de acordo com a operação desejada. O multiplexador (MUX) é o circuito lógico combinacional responsável por essa função. Conforme definido por Tocci, ele atua como um "seletor de dados", cuja função é "selecionar uma de várias linhas de entrada de dados e a conectá-la a uma única linha de saída".

Em termos práticos, ele pode ser comparado a uma chave eletrônica que, a partir de sinais de controle, determina qual entrada será conectada à saída. Dessa

forma, mesmo que existam vários sinais disponíveis, somente um é transmitido por vez. Do ponto de vista teórico, um multiplexador com *n* linhas de seleção consegue controlar até 2 elevado a *n* entradas distintas, uma relação que define a capacidade do componente. No presente projeto, foram utilizados três tipos principais de multiplexadores (8x1, 3x1 e 2x1), que, quando necessário, foram instanciados em paralelo para manipular os barramentos de dados de 8 bits, conforme detalhado a seguir.

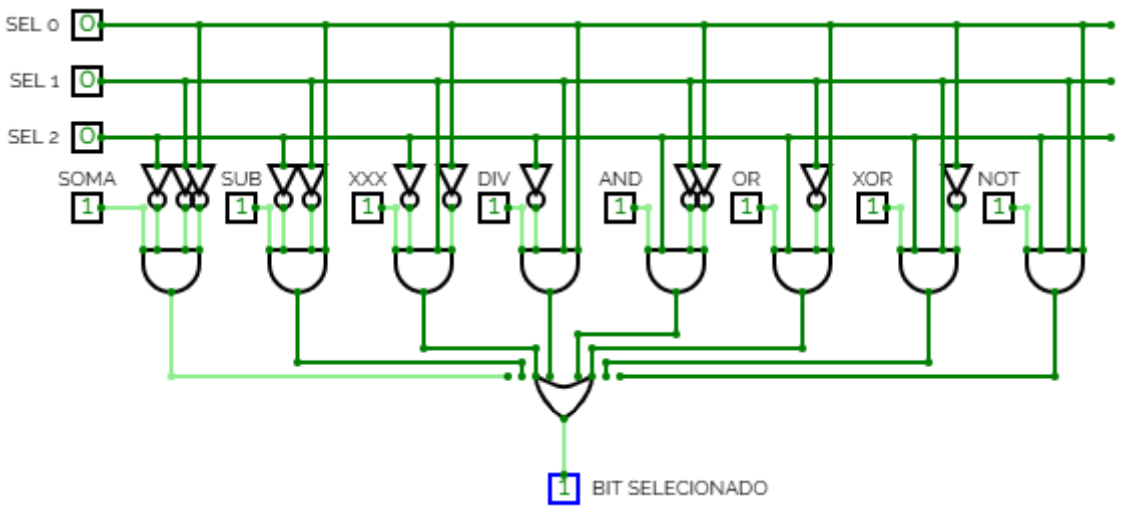
O Multiplexador 8x1, que possui 3 linhas de seleção, foi empregado no núcleo da ULA combinacional. Sua função é receber os resultados das oito operações aritméticas e lógicas disponíveis (soma, subtração, AND, etc.) e, com base nos 3 bits do Registrador de Operação, selecionar qual desses resultados será a saída final do cálculo.

Tabela 3 - Tabela Verdade Multiplexador 8x1

SELETOR [2]	**SELETOR [1]**	**SELETOR [0]**	**SAÍDA DA OPERAÇÃO**
0	0	0	Soma
0	0	1	Subtração
0	1	0	Multiplicação
0	1	1	Divisão
1	0	0	And
1	0	1	Or
1	1	0	Xor
1	1	1	Not

Fonte: compilação do autor

Figura 13 - Multiplexador de oito entradas para uma saída



Fonte: compilação do autor

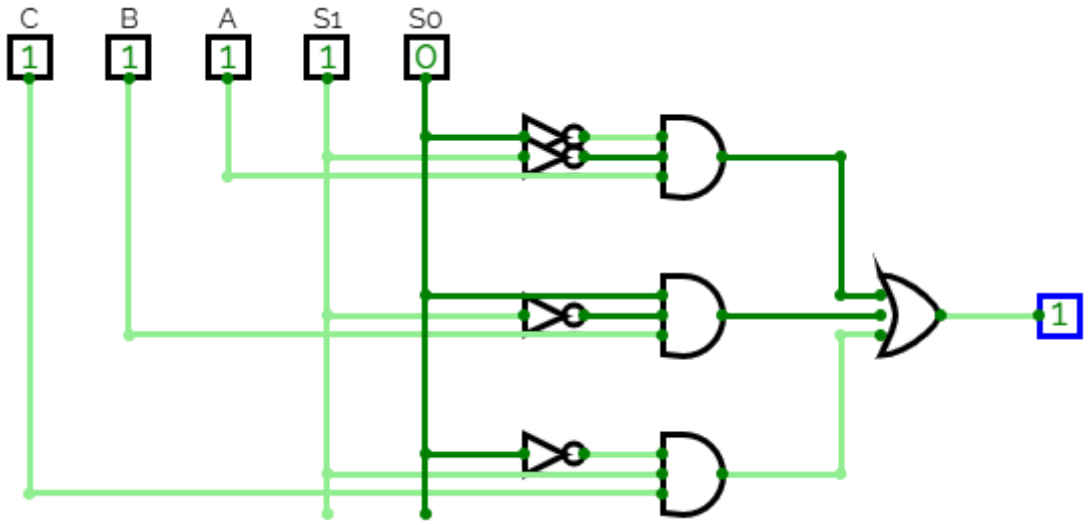
O Multiplexador 3x1 é utilizado na interface de saída do sistema. Ele é responsável por implementar a seleção da base de exibição do resultado. Este MUX recebe o valor do Registrador de Resultado já convertido para três formatos distintos (Decimal, Octal e Hexadecimal) e, com base nas chaves de seleção de base (SW[9:8]), seleciona qual desses formatos será enviado aos displays de 7 segmentos.

Tabela 4 - Tabela Verdade Multiplexador 3x1

SW[9]	**SW[8]**	**SAÍDA DA BASE**
0	0	Decimal
0	1	Octagonal
1	0	Hexadecimal
1	1	X

Fonte: compilação do autor

Figura 14 - Multiplexador de três entradas para uma saída



Fonte: compilação do autor

Por fim, o Multiplexador 2x1, que possui uma única linha de seleção, é o componente mais utilizado na arquitetura, sendo a peça-chave na implementação do caminho de dados e da lógica de controle sequencial. Ele é empregado em diversas funções críticas, tais como:

- Gerenciamento da Pilha RPN: decide se o registrador A receberá um novo dado vindo das chaves (SW) ou se manterá o resultado da operação anterior (vindo do RegResultado), implementando o elo da pilha.

- Seleção do Resultado Final: escolhe qual resultado será carregado no RegResultado, selecionando entre a saída da ULA combinacional (para operações instantâneas) ou a saída do multiplicador sequencial (para a operação de múltiplos ciclos).

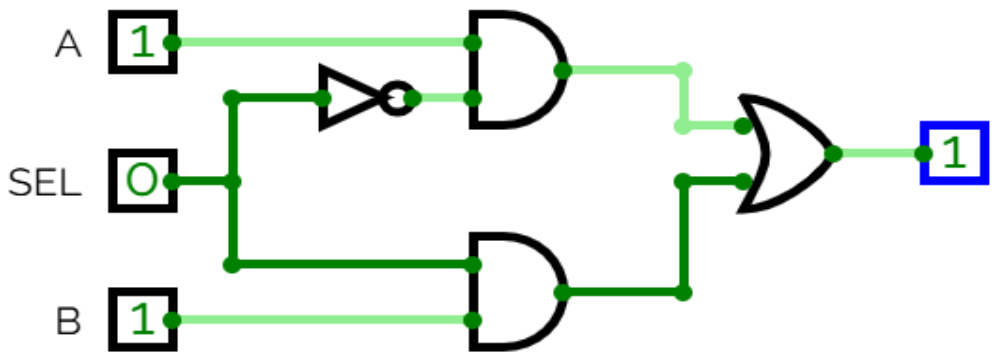
- Controle de Exibição: gerencia a exibição do resto da divisão, selecionando entre o valor do registrador do resto da divisão ou um valor nulo, para "limpar" o display quando a operação não for uma divisão.

Tabela 5 - Tabela Verdade Multiplexador 2x1

SELETOR	**BIT DE SAÍDA**
0	A
1	B

Fonte: compilação do autor

Figura 15 - Multiplexador de duas entradas para uma saída



Fonte: compilação do autor

4.4.2 Módulos de Flags

- **Carry ou Borrow Out**

A flag de Carry/Borrow (Cout) é um indicador de um único bit que sinaliza que uma operação aritmética excede a capacidade de representação do barramento de dados, especificamente em aritmética com números sem sinal. Sua função difere ligeiramente entre a soma e a subtração. Na soma, o “Carry Out” é o bit de transporte gerado pela soma dos bits mais significativos dos operandos, sinalizando que o resultado verdadeiro é maior do que o representado nos displays. Já na subtração, o “Borrow Out” é o empréstimo que não pode ser pago pelo bit mais significativo em casos que o subtraendo é menor que o minuendo.

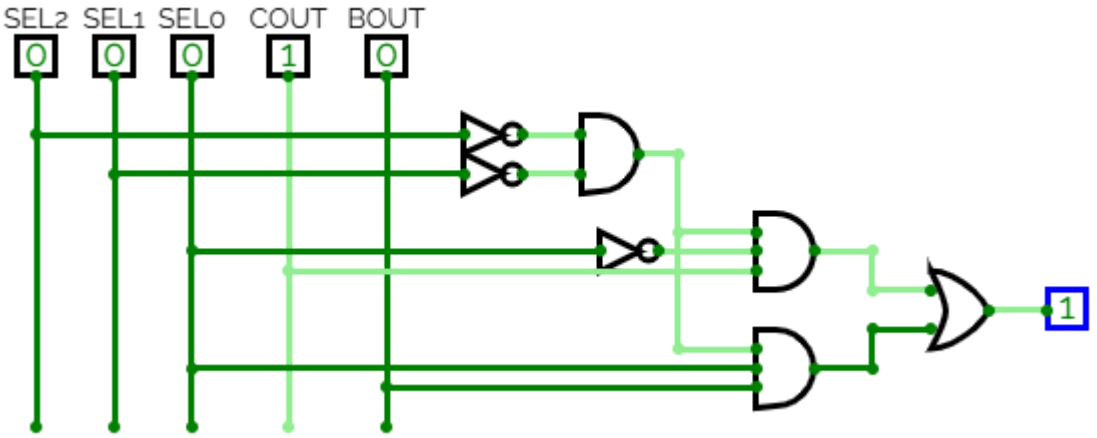
Tabela 6 - Tabela Verdade da flag de Cout

SELETOR 2	SELETOR 1	SELETOR 0	COUT	BOUT	FLAG ON
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0

0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Fonte: compilação do autor

Figura 16 - Circuito da Flag de Carry/Borrow Out



Fonte: compilação do autor

- **Flag de Erro**

A Flag de Erro (E) é um indicador de um único bit projetado para sinalizar uma operação matematicamente inválida, ou seja, uma operação para a qual é impossível determinar um resultado numérico válido. A função desta flag é especificamente a detecção da condição de divisão por zero, sendo ativada quando a ULA recebe um comando para executar a operação de divisão e o circuito de controle detecta que o divisor é igual a zero. Essa verificação é essencial, pois a divisão por zero é uma operação indefinida na aritmética, não possuindo um quociente ou resto válidos que possam ser representados.

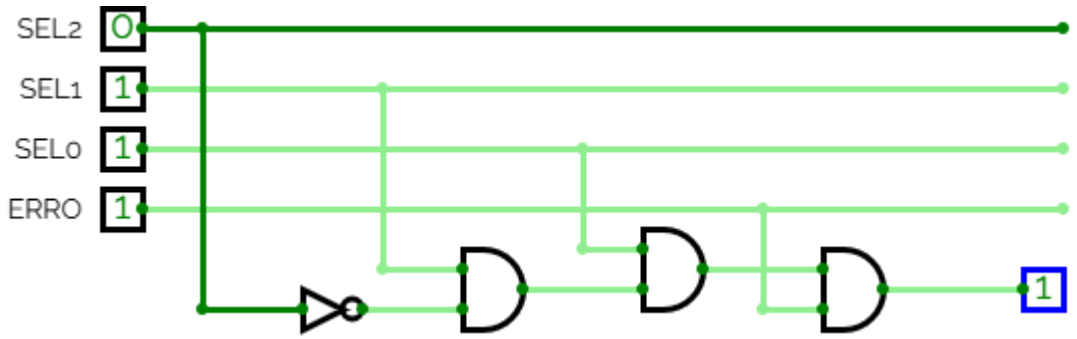
Tabela 7 - Tabela Verdade da Flag de Erro

SELETOR [2]	**SELETOR[1]**	**SELETOR[0]**	**ERRO DIVISÃO**	**FLAG ON**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0

1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Fonte: compilação do autor

Figura 17 - Circuito da Flag de Erro na Divisão



Fonte: compilação do autor

- **Flag de Resultado Zero**

A Flag de Zero (Z) é um indicador de um único bit cuja função principal é sinalizar que o resultado de uma operação aritmética ou lógica executada pela ULA é exatamente zero. Conforme a convenção padrão, a flag assume o nível lógico 1 (ativa) apenas quando esta condição específica é verdadeira e permanece em 0 em todos os outros casos.

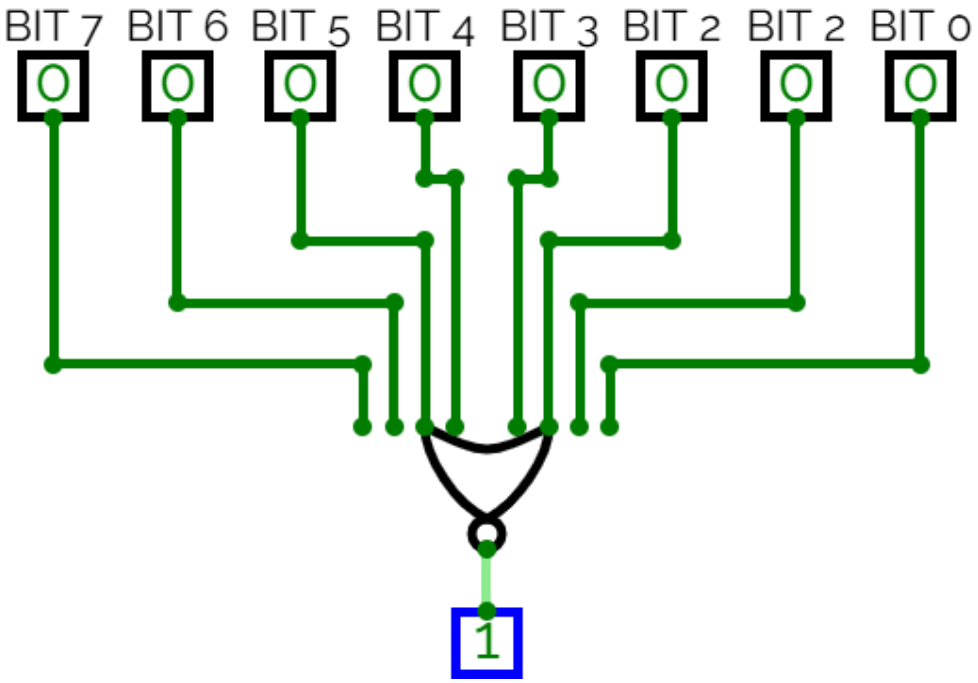
Do ponto de vista lógico, para que um valor de 8 bits seja zero, é necessário que todos os bits do barramento de resultado, de S[7] a S[0], sejam iguais a 0. A implementação de hardware mais direta e eficiente para detectar esta condição é através de uma porta NOR de 8 entradas. Por definição, uma porta NOR produzirá uma saída em nível lógico 1 somente se todas as suas entradas estiverem em nível lógico 0, o que corresponde perfeitamente à definição da Flag de Zero.

Tabela 8 - Tabela Verdade da Flag de Resultado Zero

S[7]	S[6]	S[5]	S[4]	S[3]	S[2]	S[2]	S[1]	S[0]	CASO ZERO
0	0	0	0	0	0	0	0	0	1

Fonte: compilação do autor

Figura 18 - Circuito da Flag de Resultado Zero



Fonte: compilação do autor

- **Flag de Resto da Divisão**

A Flag de Resto é um indicador de um único bit que sinaliza que a operação de divisão resultou em um resto não-nulo. A ativação desta flag é condicional e depende de duas lógicas simultâneas, conforme ilustrado no circuito.

1. **Verificação do Resto:** Primeiramente, o sistema verifica se o resultado do resto é diferente de zero. Isso é implementado por uma porta **OR** de 8 entradas, que recebe todos os bits do barramento Resto. Se qualquer um desses 8 bits for 1, a saída da porta OR será 1, indicando que existe um resto.
2. **Verificação da Operação:** Em segundo lugar, o sistema garante que essa flag só seja ativada se a operação selecionada for, de fato, a "Divisão". A saída da porta OR ("Resto ≠ 0") é combinada em uma porta **AND** com os bits de seleção de operação.

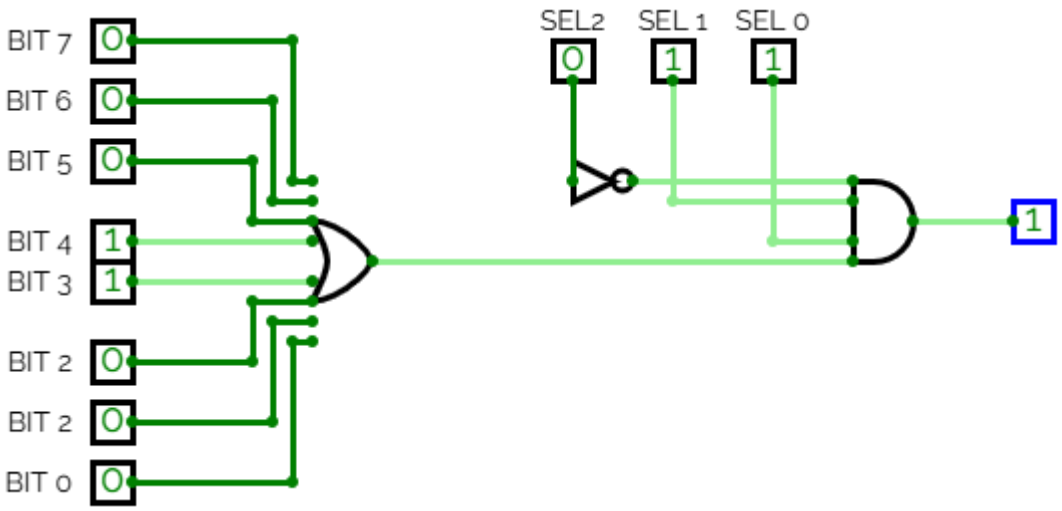
A flag de resto só assumirá o nível lógico 1 se ambas as condições forem verdadeiras: a operação selecionada for "Divisão" e o resultado dessa divisão gerar um resto diferente de zero.

Tabela 9 - Tabela Verdade da Flag de Resto

SELETOR[2]	SELETOR[1]	SELETOR[0]	RESTO?	FLAG ATIVA
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Fonte: compilação do autor

Figura 19 - Circuito da Flag de Resto



Fonte: compilação do autor

4.4.3 Decodificadores

Em sistemas digitais, o processamento de dados frequentemente requer a tradução ou conversão de informações de um código para outro. Os decodificadores são os circuitos lógicos combinacionais fundamentais responsáveis por essa tarefa. Conforme definido por Tocci, um decodificador converte uma entrada codificada em uma saída codificada, onde os códigos de entrada e saída são diferentes. Embora existam muitos tipos, a forma mais comum é o decodificador *n* para 2 elevado a *n*, que aceita uma entrada binária de *N* bits e a utiliza para ativar exatamente uma das saídas.

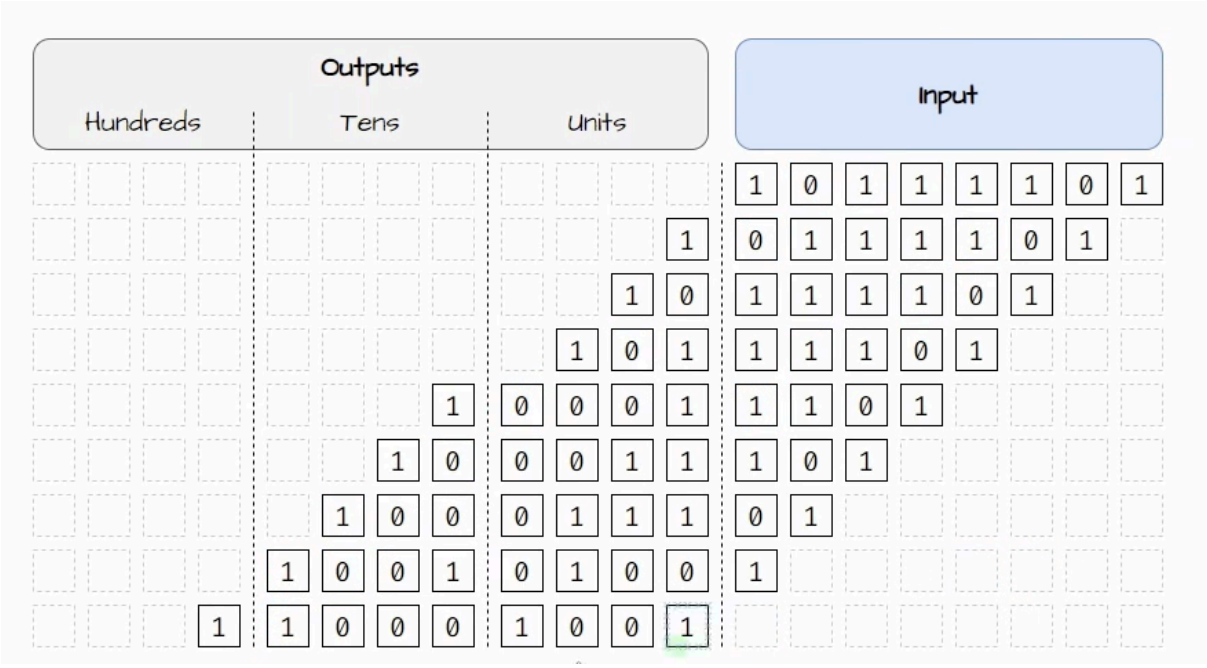
O princípio de operação de um decodificador é o reconhecimento de um código binário específico. Por exemplo, em um decodificador 3-para-8 (N=3), se a entrada for o código 101, apenas a quinta saída será ativada, enquanto todas as outras permanecem inativas. Esta lógica é a base para a conversão de binário para sistemas numéricos como octal e hexadecimal, bem como para a seleção de endereços em memórias e periféricos. No contexto do projeto, diferentes tipos de decodificadores e conversores são necessários para exibir o resultado de 8 bits em múltiplas bases.

- **Binário para Decimal**

A conversão de Binário para Decimal é a mais complexa, pois não pode ser implementada por um decodificador de 2 elevado a *n* bits padrão. Um resultado de 8 bits (0-255) precisa ser exibido em três dígitos decimais (centena, dezena, unidade). Para isso, o número binário puro deve ser primeiro convertido para o formato BCD (Binary-Coded Decimal). Para isso, utilizamos o algoritmo "shift-and-add-3", também conhecido como Double Dabble, o algoritmo opera deslocando o número binário de entrada, bit a bit, para dentro de registradores BCD. Antes de cada deslocamento, o

circuito verifica cada dígito BCD (4 bits), se o valor for 5 ou maior, 3 é somado a ele. Esse ajuste corrige o valor para o formato BCD antes que o próximo deslocamento ocorra, garantindo uma conversão decimal precisa.

Figura 20 - Fluxo do Algoritmo de Deslocamento Double Dabble



Fonte: Logic Alligator (Youtube)

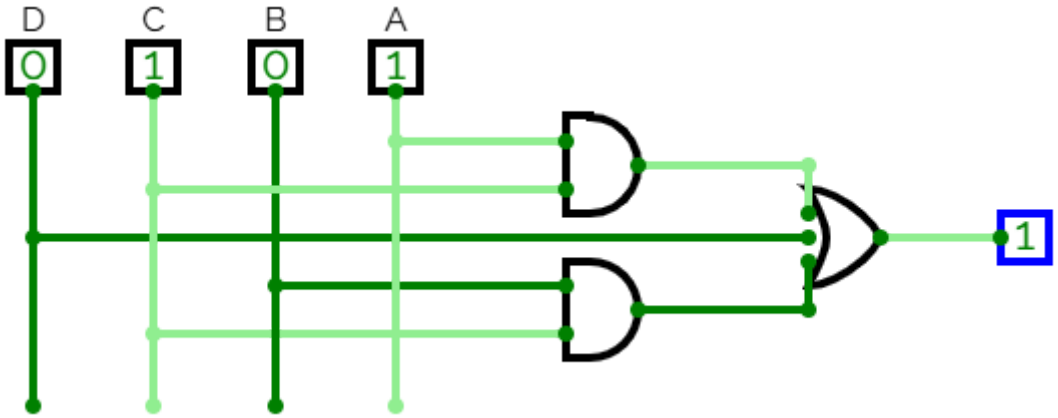
Tabela 10 - Tabela Verdade do Comparador maior ou igual a 5

Entrada D	**Entrada C**	**Entrada B**	**Entrada A**	**S**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X

1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Fonte: compilação do autor

Figura 21 - Circuito do Comparador > 5



Fonte: compilação do autor

- **Binário para Octal**

A conversão de Binário para Octal é uma aplicação direta de um decodificador. O sistema octal (base 8) naturalmente agrupa números binários em trios de bits (3 bits), onde cada trio (de 000 a 111) corresponde a um dígito octal (de 0 a 7). Nesse sentido, um decodificador 3-para-8 implementa precisamente divide o barramento de 8 bits em grupos de 3 bits, e cada grupo alimenta um decodificador 3-para-8 para gerar os dígitos octais correspondentes.

- **Binário para Hexadecimal**

De forma análoga, a conversão de Binário para Hexadecimal (base 16) é implementada por decodificadores 4-para-16. O sistema hexadecimal agrupa o número binário em quartetos de bits (4 bits), onde cada quarteto (de 0000 a 1111) representa um dos 16 dígitos hexadecimais (de 0 a 9, A a F). Um decodificador 4-para-16 recebe cada quarteto de 4 bits do resultado e ativa a saída única correspondente, que por sua vez é traduzida para o formato de 7 segmentos.

No sistema geral da calculadora, esses três circuitos conversores operam em paralelo, recebendo o mesmo dado de 8 bits do Registrador de Resultado. Um multiplexador 3x1, controlado pelas chaves de seleção de base (SW[9:8]), seleciona qual dos três resultados convertidos será enviado aos Hex-para-7-Segmentos, que realizam a tradução final para acionamento dos displays.

5. ANÁLISE DOS RESULTADOS E TESTES

5.1 ANÁLISE DOS RESULTADOS

5.1.1 Resultados Funcionais

Para a validação funcional, foi executada uma série de casos de teste, abrangendo todos os cenários de operação da calculadora RPN em cenários normais e de limite. Os testes, realizados tanto em simulação no ModelSim quanto fisicamente na placa DE10-Lite, confirmaram que o protótipo se comporta exatamente como o esperado, validando a complexa interação entre a unidade de controle sequencial e o caminho de dados.

A validação confirmou o pilar central do projeto: a Máquina de Estados Finitos (FSM) que gerencia o ciclo de 4 passos da RPN . Foi verificado que os sinais de controle (LoadA, LoadB, LoadOp) são ativados em suas respectivas etapas em sincronia com o comando ENTER. Crucialmente, a lógica da pilha RPN (pilhaRPN) foi validada, demonstrando que o RegResultado da operação anterior é corretamente selecionado como o novo Operando A, permitindo operações encadeadas.

Também foi validado o gerenciamento de operações de diferentes naturezas. As operações combinacionais da ULA (soma, subtração, divisão e lógicas) apresentaram resultados instantâneos, enquanto a operação sequencial do Multiplicador demonstrou o correto travamento do contador de etapas (via AguardandoMult), que só avançou após o sinal de ProntoMult. Todas as flags (Cout, Erro, Resto, Zero e Overflow) foram acionadas precisamente nas condições projetadas.

Ao final, a interface de saída foi verificada. Os módulos conversores de base e o multiplexador de seleção de base (SW[9:8]) funcionaram como projetado, exibindo os resultados corretos nos displays de 7 segmentos para os formatos Decimal, Octal e Hexadecimal. A correspondência entre os resultados simulados e os observados na placa valida o produto final do circuito lógico desenvolvido e sua implementação estrutural. Portanto, concluímos que o protótipo atendeu a todos os requisitos funcionais estabelecidos no problema.

5.1.1 Resultados de Síntese

Um dos objetivos centrais do projeto em FPGA é não apenas criar um circuito funcional, mas também analisar como este circuito é implementado fisicamente no hardware. A etapa de síntese, realizada pelo software Quartus Prime, traduz o código Verilog em uma rede de componentes lógicos físicos disponíveis no chip da FPGA. A análise do uso desses recursos é fundamental na engenharia de sistemas digitais por vários motivos:

- **Custo e Eficiência**: Circuitos maiores e mais complexos utilizam mais recursos do chip. Em um projeto comercial, isso impacta diretamente o custo, pois FPGAs maiores são mais caras.

- **Limitações do Hardware:** Cada chip FPGA possui uma quantidade finita de recursos (elementos lógicos, memória, etc.). A análise de síntese confirma se o projeto pode ser executado na DE10-Lite.

- **Desempenho:** A complexidade do circuito e a forma como os elementos lógicos são interconectados podem afetar a velocidade máxima de operação do sistema.

Após a compilação bem-sucedida do projeto ULARPN no Quartus, o relatório de síntese ("Fitter" -> "Resource Usage Summary") forneceu os dados sobre a utilização de recursos do chip MAX 10. Conforme a figura abaixo, o projeto completo demonstrou a complexidade de um sistema sequencial, utilizando um total de 519 elementos lógicos (LEs).

Desse total, 516 foram implementados como funções combinacionais (LUTs), sendo a maioria composta por funções de 4 entradas (370) e 3 entradas (113). A natureza sequencial do projeto, responsável pelo gerenciamento de estado da RPN, pela pilha e pelos contadores, exigiu o uso de 69 registradores lógicos dedicados (Flip-Flops). Finalmente, a interface com o usuário (chaves, botões, LEDs e displays de 7 segmentos) utilizou um total de 65 pinos de I/O.

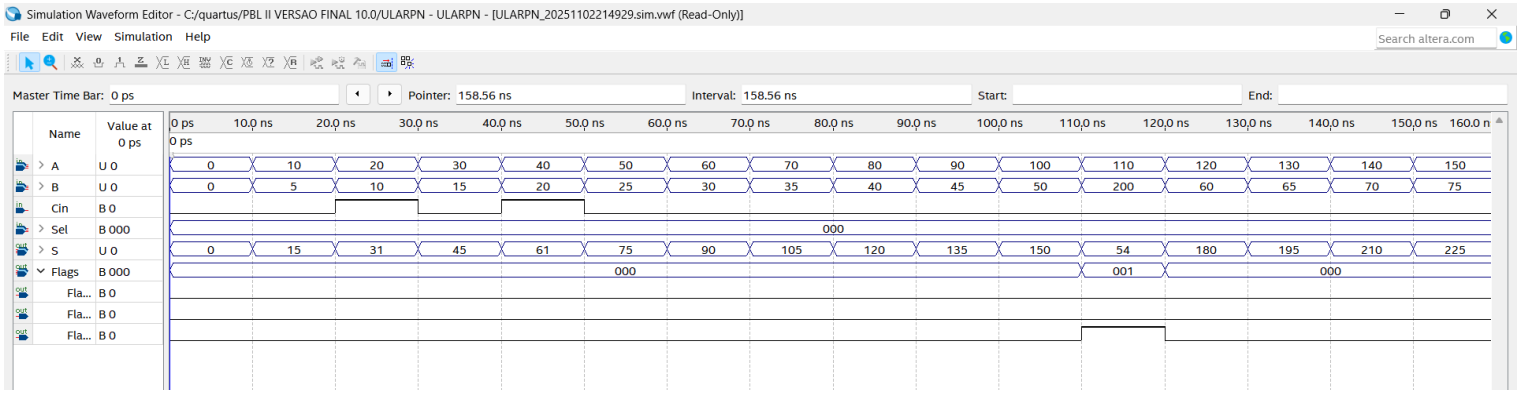
Figura 22 - Relatório de síntese

Resource	Usage
Estimated Total logic elements	519
Total combinational functions	516
Logic element usage by number of LUT inputs	
-- 4 input functions	370
-- 3 input functions	113
-- <=2 input functions	33
Logic elements by mode	
-- normal mode	516
-- arithmetic mode	0
Total registers	69
-- Dedicated logic registers	69
-- I/O registers	0
I/O pins	65

Fonte: Quartus Prime Lite

5.2 TESTES REALIZADOS (WAVEFORMS)

Figura 23 - Waveform para validar o somador



Fonte: Quartus Prime Lite

Figura 24 - Waveform para validar o subtrator

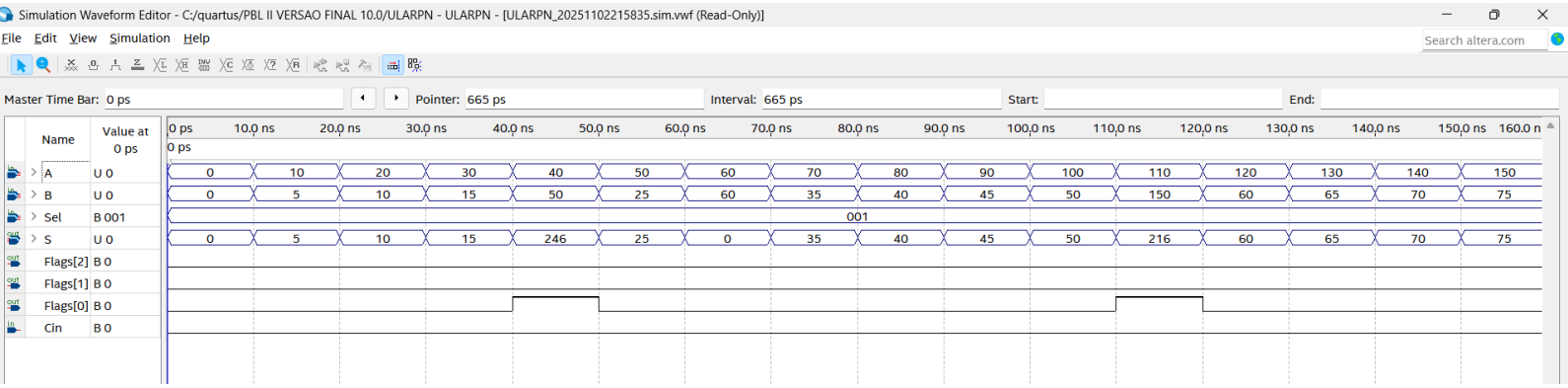
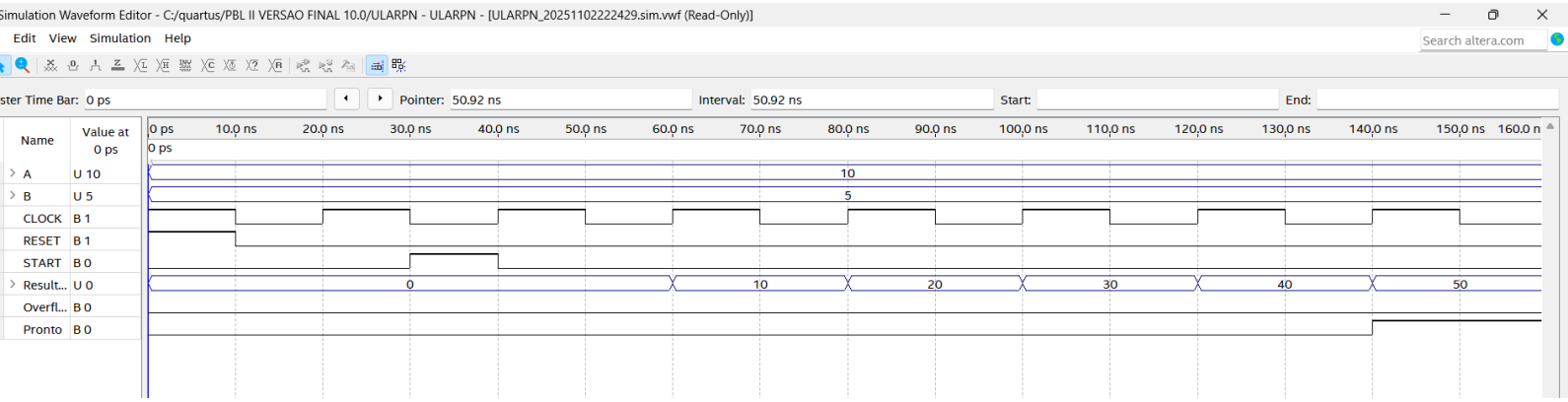
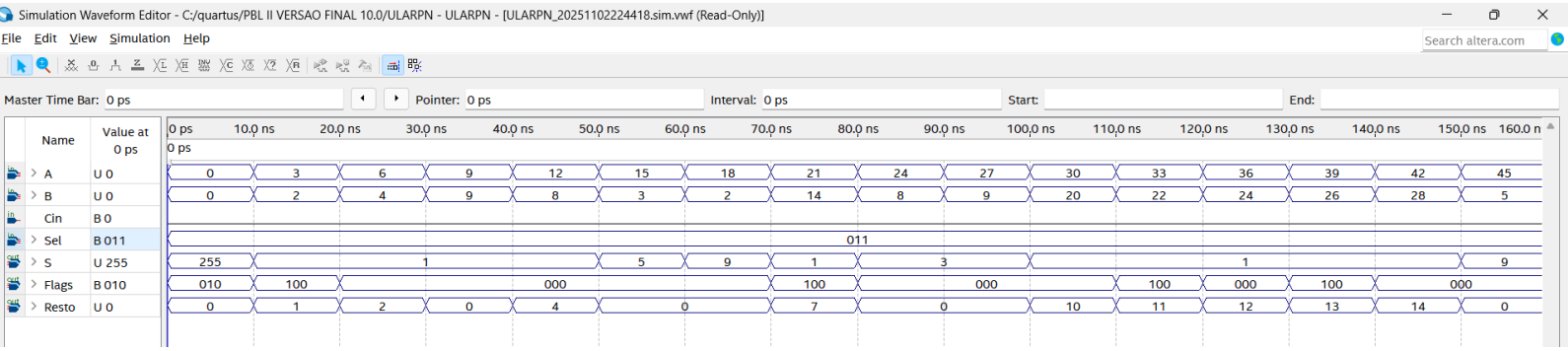


Figura 25 - Waveform para validar o multiplicador



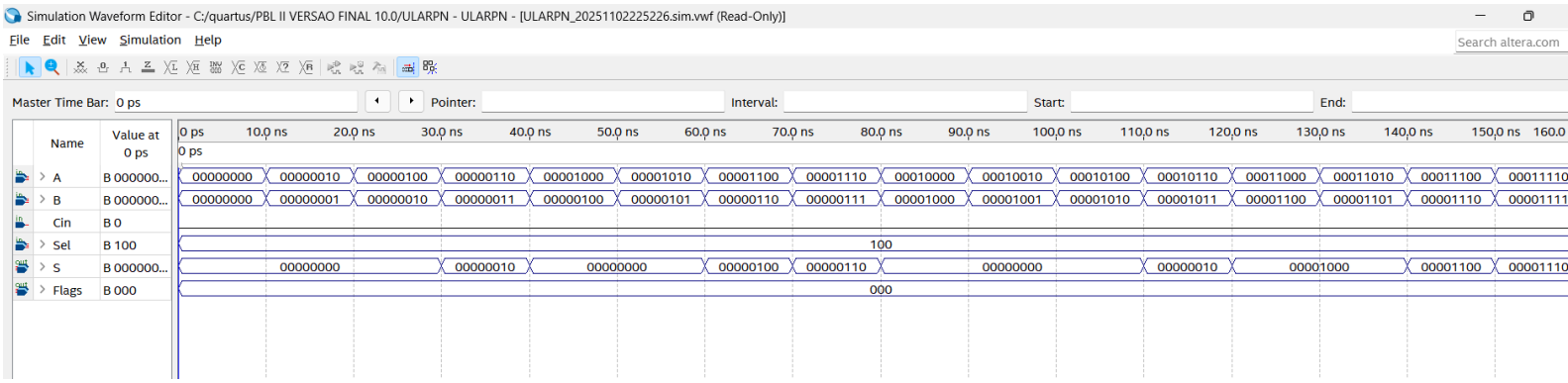
Fonte: Quartus Prime Lite

Figura 26 - Waveform para validar o divisor



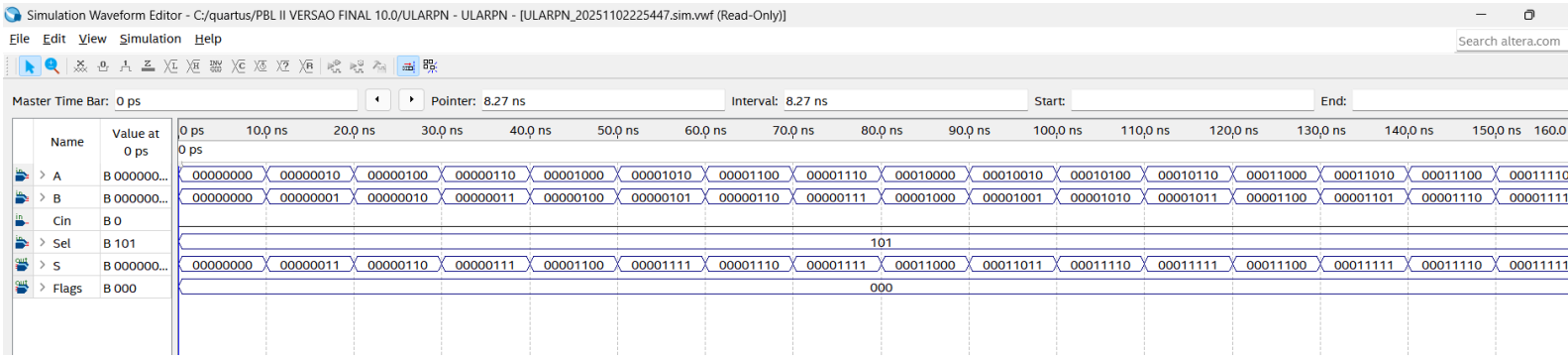
Fonte: Quartus Prime Lite

Figura 27 - Waveform para validar a and



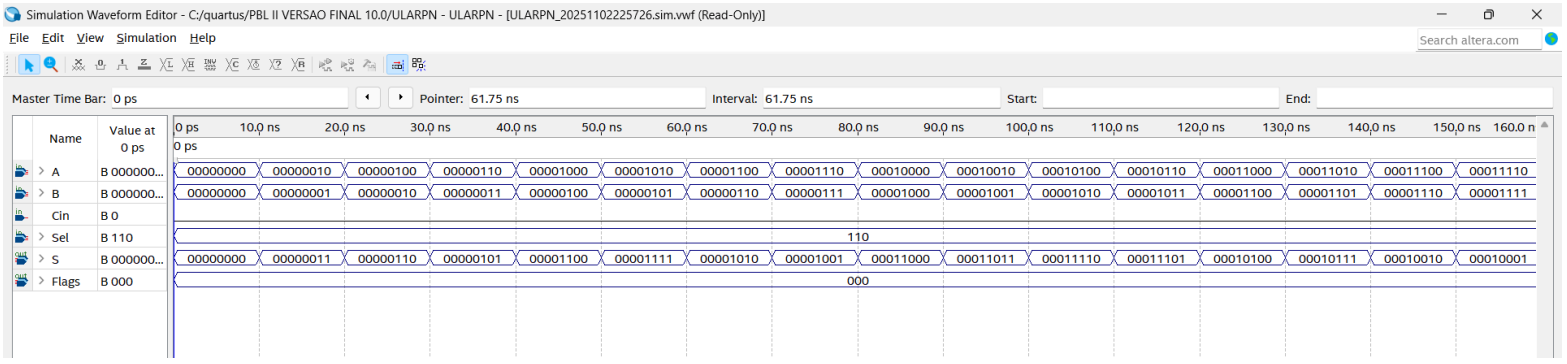
Fonte: Quartus Prime Lite

Figura 28 - Waveform para validar a or



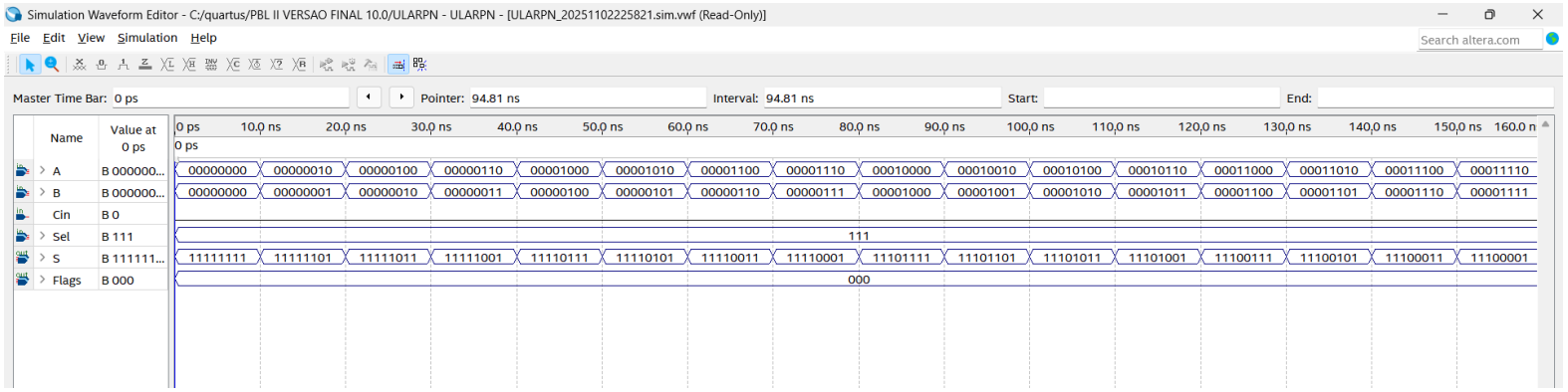
Fonte: Quartus Prime Lite

Figura 29 - Waveform para validar a xor



Fonte: Quartus Prime Lite

Figura 30 - Waveform para validar a not



Fonte: Quartus Prime Lite

Para a verificação funcional de cada módulo implementado, foram utilizados arquivos de forma de onda (waveforms). Esta metodologia permite uma análise detalhada do comportamento lógico de cada circuito, observando as saídas geradas em resposta a um conjunto controlado de vetores de teste aplicados às entradas. Os testes abrangeram todos os módulos aritméticos (como somador8x8, subtrator8x8, divisor8x8, etc.) e os módulos lógicos (como AND8bits, OR8bits, XOR8bits, etc.). Em todos os cenários simulados, os circuitos apresentaram resultados satisfatórios, com as saídas correspondendo exatamente aos valores teóricos esperados, validando assim a correta implementação de cada componente.

6. CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto permitiu a aplicação prática e aprofundada não apenas dos conceitos fundamentais de circuitos combinacionais, mas, crucialmente, dos princípios de sistemas sequenciais síncronos. O resultado foi a implementação bem-sucedida de uma calculadora de 8 bits completa, baseada na arquitetura de Notação Polonesa Reversa (RPN). A implementação em Verilog puramente estrutural validou o domínio sobre o gerenciamento de estado, a sincronização de sinais e o fluxo de dados em um projeto síncrono complexo.

A arquitetura modular adotada, centrada na Máquina de Estados Finitos que governa os registradores da pilha, demonstrou-se uma abordagem robusta e eficaz para implementar a lógica RPN de 4 passos . Esta unidade de controle gerenciou com sucesso a interação entre módulos de naturezas distintas, como a ULA

combinacional (para operações instantâneas) e o Multiplicador sequencial (que exige múltiplos ciclos de clock). A validação do projeto, confirmada por simulação e testes práticos na placa FPGA DE10-Lite, demonstrou que todas as operações, a lógica da pilha e as cinco flags de status (Cout, Overflow, Zero, Erro, Resto) operam conforme o especificado.

O protótipo final atendeu, portanto, a todos os requisitos funcionais e técnicos estabelecidos, demonstrando a capacidade de projetar um sistema digital complexo, modular e hierárquico, desde a concepção teórica até a implementação física em hardware.

7. REFERÊNCIAS BIBLIOGRÁFICAS

CHU, Pong P. *FPGA Prototyping by Verilog Examples*: *Xilinx Spartan-3 Version*. Hoboken: John Wiley & Sons, 2008.

IDOETA, Ivan Valeje; CAPUANO, Francisco Gabriel. *Elementos de eletrônica digital*. 35. ed. São Paulo: Érica.

TOCCI, Ronald J. *Sistemas digitais: princípios e aplicações*. 11. ed. São Paulo: Pearson Prentice Hall, 2011.

WHITNEY, Thomas M.; RODÉ, France; TUNG, Chung C. *The 'Powerful Pocketful': an Electronic Calculator Challenges the Slide Rule*. Hewlett-Packard Journal, v. 23, n. 10, p. 2-9, 1972.