RELATÓRIO PROBLEMA I - 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 em linguagem de descrição de hardware para aplicação em placa de circuito lógico programável (FPGA).

Palavras-chave: ULA, Circuitos Digitais, FPGA.

1. INTRODUÇÃO

Desde a invenção do transistor, a eletrônica passou por diversos avanços, sendo a tecnologia computacional uma área extremamente promissora e de intensos processos de inovação até os dias vigentes, tendo os Circuitos Digitais como base de todo o desenvolvimento dos computadores modernos. Nessa perspectiva, a Unidade Lógica/Aritmética é uma das principais partes de um computador, sendo responsável por realizar decisões lógicas e cálculos aritméticos que serão enviados a uma unidade de memória ou para a saída do circuito (TOCCI, 2011, p.18).

Dessa forma, este relatório tem o objetivo de apresentar o desenvolvimento de uma Unidade Lógica e Aritmética atendendo aos requisitos do Problema I da disciplina de Circuitos Digitais (TEC498 - 2025.2), dando enfoque à metodologia e fundamentação teórica utilizadas, além do detalhamento do resultado obtido como resolução final. A ULA solicitada deve obter como entrada dois operandos inteiros de 4 bits e um valor (1 ou 0) de Carry-in, em seguida realizar operações lógicas (And, Or e Xor), aritméticas (soma, subtração, multiplicação e divisão), e por fim, exibir os resultados das operações no display de 7 segmentos e nos LEDs do dispositivo fornecido. Vale considerar que o circuito deveria ser puramente combinacional, ou seja, sem a utilização de memória para quaisquer operações.

Toda a implementação foi realizada no dispositivo de *hardware* programável DE10-Lite, utilizando-se da linguagem de descrição de *hardware* Verilog por meio do *software* Quartus Prime Lite.

2. VISÃO MACRO DO PROJETO

Em primeira análise, a Unidade Lógica e Aritmética (ULA) desenvolvida possui como entradas principais duas palavras binárias de 4 bits, representadas pelas variáveis A e B, além de um bit de Carry-In (Cin) proveniente de operações anteriores, e uma palavra seletora de 3 bits (Sel), responsável por definir qual operação será executada.

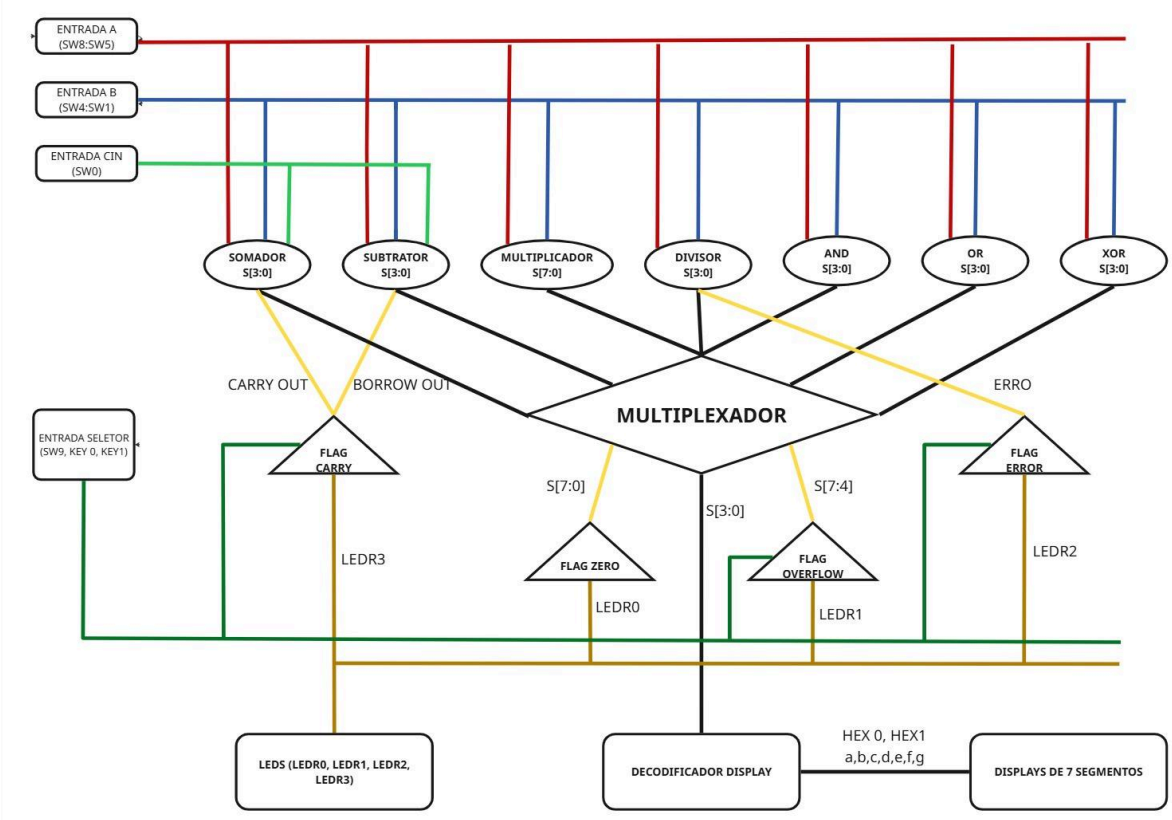
As entradas A e B são direcionadas simultaneamente para os módulos responsáveis pelas operações aritméticas (soma, subtração, multiplicação e divisão) e para as operações lógicas básicas (AND, OR e XOR). Tendo isso em vista, devido a impossibilidade de representar diretamente as sete operações em uma única saída, foi projetado um multiplexador de 8 entradas para 1 saída, que seleciona o resultado final de acordo com o valor do seletor Sel.

O resultado selecionado é então compartilhado com os módulos auxiliares de análise e exibição:

- **Flag de Zero (Z):** verifica se a palavra de saída é igual a zero.
- **Flag de Overflow (O):** identifica estouros em multiplicações.
- **Flag de Carry/Borrow (C):** sinaliza se houve transporte (carry out) ou empréstimo (borrow out) na operação.
- **Flag de Erro (E):** ativada em casos de divisão por zero.
- **Decodificador para Display de 7 segmentos:** responsável por converter os 4 bits menos significativos do resultado em sinais adequados para exibição decimal.

O diagrama a seguir apresenta a visão estrutural do funcionamento circuito, destacando a organização modular da ULA.

Figura 1 - Diagrama de Alto Nível do projeto.



Fonte: Compilação do autor

Tendo isso em vista, no mapeamento físico da FPGA DE10-Lite, foram adotadas as seguintes conexões:

- **Entradas (CHAVES/BOTÕES):**
    - Operando A: **SW[8] a SW[5]**
    - Operando B: **SW[4] a SW[1]**
    - *Carry-In*: **SW[0]**
    - Seletor Sel: **SW[9], KEY[0], KEY[1]**
- **Sinalização das Flags (LEDR):**
    - *Flag Zero*: **LEDR[0]**
    - *Flag Overflow*: **LEDR[1]**
    - *Flag Erro*: **LEDR[2]**
    - *Flag Carry/Borrow*: **LEDR[3]**
- **Saídas nos Display de 7 Segmentos (HEX):**
    - Unidades: **HEX[00] a HEX[06]**
    - Dezenas: **HEX[10] a HEX[16]**

3. METODOLOGIA DE DESENVOLVIMENTO E IMPLEMENTAÇÃO

O desenvolvimento da Unidade Lógica Aritmética (ULA) de 4 bits seguiu uma metodologia sistemática, baseada no fluxo clássico de projeto de sistemas digitais. O processo foi dividido em etapas sequenciais que permitiram a organização do trabalho, a verificação parcial de cada componente e, ao final, a integração completa do sistema. Em primeira análise, foram estabelecidas as operações a serem suportadas pela ULA, bem como suas entradas e saídas. A partir dessa definição, construiu-se a tabela verdade geral, relacionando cada código do seletor de operações à função correspondente.

Na sequência, cada operação foi detalhada individualmente. Para cada módulo, foram elaboradas as seguintes etapas:

- **CRIAÇÃO DA TABELA VERDADE**

De acordo com Tocci, uma tabela-verdade é uma técnica para descrever como a saída de um circuito lógico depende dos níveis lógicos presentes nas entradas do circuito. No contexto deste projeto, a tabela verdade foi utilizada como a especificação formal para cada módulo combinacional. Sendo uma representação visual de todas as combinações possíveis de entrada e saída para uma função Booleana, ela serviu como o ponto de partida inequívoco para o design.

A tabela verdade fornece uma análise completa da função lógica, detalhando todos os valores que a função pode assumir com base em suas entradas. Para componentes com um número limitado de entradas, como o somador de 1 bit ou multiplexador 8x1 a tabela verdade completa foi elaborada manualmente. Para módulos mais complexos, como o circuito da flag de overflow, foram utilizadas tabelas verdade condensadas que descrevem a função lógica de forma mais compacta, mas ainda completa.

- **ÁLGEBRA BOOLEANA E MAPAS DE KARNAUGH**

Uma vez definida a função lógica na tabela verdade, o passo seguinte consistiu na simplificação da função, de modo a reduzir a quantidade de portas lógicas necessárias. A Álgebra Booleana é a ferramenta matemática clássica para esse fim, permitindo manipular expressões até obter uma forma reduzida.

Para funções com até 8 variáveis de entrada, foram empregados Mapas de Karnaugh (Mapa-K), recurso gráfico que facilita a identificação de termos redundantes e a derivação de equações minimizadas na forma de soma de produtos (SOP) ou produto das somas (POS).

- **DIAGRAMA E IMPLEMENTAÇÃO DO CIRCUITO VERILOG**

Com as equações booleanas mínimas em mãos, a etapa seguinte consistiu no desenho do diagrama do circuito lógico para cada módulo, utilizando as portas lógicas fundamentais (AND, OR, NOT, XOR). Este diagrama serviu como a planta baixa para a fase final de codificação. A implementação foi realizada em Verilog puramente estrutural, onde cada porta lógica do circuito foi traduzido diretamente em uma instância de módulo ou declaração de wire no código.

As portas lógicas são os dispositivos que implementam fisicamente uma função Booleana, e o Verilog estrutural permite descrever textualmente essa interconexão de componentes, garantindo que o hardware sintetizado seja um reflexo fiel do circuito projetado. Esta abordagem foi aplicada a todos os módulos, desde os mais simples até os mais elaborados,

A seguir, será detalhado o processo de criação para cada um dos módulos funcionais da ULA.

3.1 MÓDULOS ARITMÉTICOS

3.1.1 SOMADORES

A adição em base binária segue os mesmos princípios da adição no sistema decimal, porém adaptada à restrição de que cada dígito só pode assumir os valores 0 ou 1. Dessa forma, ao somar 1 com 1, não é possível representar o resultado apenas em um dígito; surge, então, a necessidade do transporte, conhecido como *carry*. Nesse contexto, a operação de soma pode ser resumida em quatro casos elementares:

Tabela 1 - Tabela Verdade do Meio Somador

BIT DE A	OPERADOR	BIT DE B	CARRY SOMADO	RESULTADO
0	+	0	0	0
1	+	0	0	1
1	+	1	0	0 (com carry de 1)
1	+	1	1	1 (com carry de 1)

Tabela 2 - Tabela Verdade Somador Completo

ENTRADA A	ENTRADA B	CARRY IN	SAÍDA	CARRY OUT
0	0	0	0	0
0	0	1	1	0

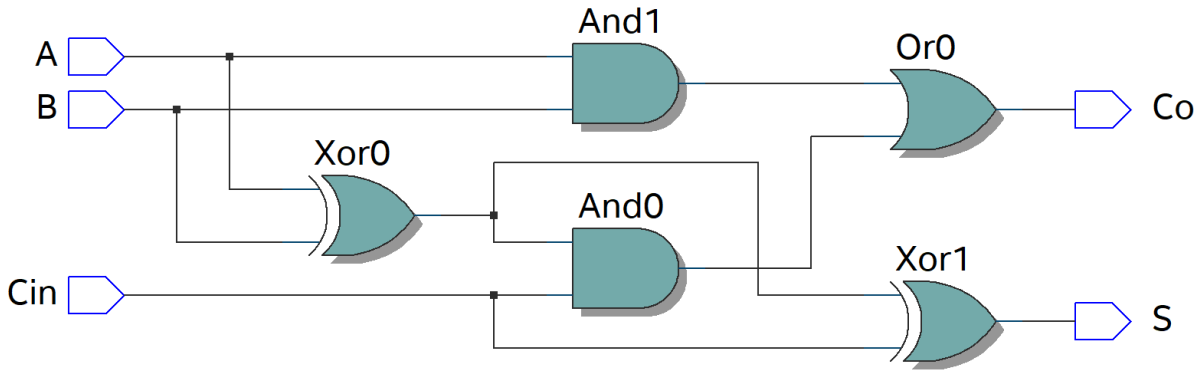
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- **Equações Booleanas**

  S = Cin ⊕ A ⊕ B
  Cout = AB + Cin(A⊕B)

- **Diagrama do Circuito**

Figura 2 - Circuito somador completo



Fonte: Compilação do autor.

Esse princípio fundamenta a construção de dispositivos chamados somadores completos, que constituem a base para operações aritméticas em sistemas digitais.

3.1.2 SUBTRATORES

A subtração binária também mantém forte analogia com o processo decimal, diferenciando-se apenas no tratamento da necessidade de “tomar emprestado” de uma posição mais significativa, operação conhecida como *borrow*. Assim, os casos básicos possíveis são:

Tabela 3 - Tabela Verdade do Meio Subtrator.

BIT DE A	OPERADOR	MENOS BIT DE B	RESULTADO
0	-	0	0
1	-	1	0
1	-	0	1
0	-	1	1 (com borrow de 1)

Tabela 4 - Tabela Verdade do Subtrator Completo.

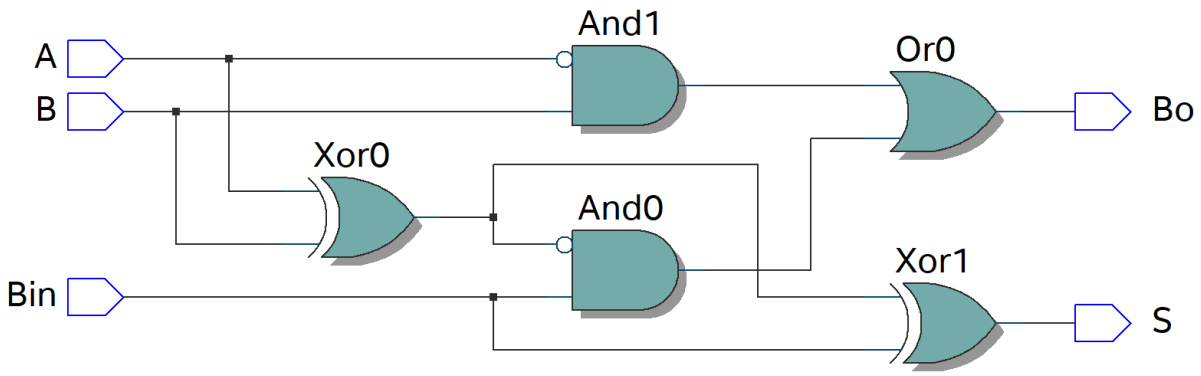
ENTRADA A	ENTRADA B	BORROW IN	SAIDA	BORROW OUT
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- **Equações Booleanas**

  S = Bin ⊕ A ⊕ B
  Cout = ¬(AB) + Bin ¬A⊕B)

- **Diagrama do Circuito**

Figura 3 - Circuito subtrator completo

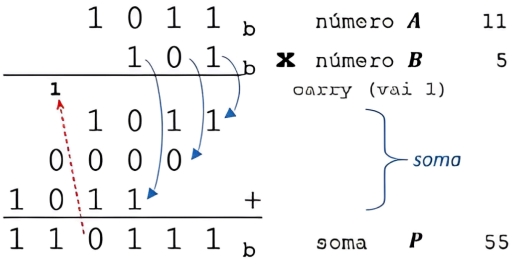


Fonte: compilação do autor.

3.1.3 MULTIPLICADORES

A multiplicação binária é, de certo modo, ainda mais simples do que a decimal, pois os fatores podem assumir apenas 0 ou 1. Isso significa que, quando o bit do multiplicador é 1, o resultado parcial corresponde ao próprio multiplicando; quando é 0, o produto parcial é nulo. Ao final, é necessário realizar a soma dos produtos parciais, deslocados conforme a posição de cada bit do multiplicador. O processo abaixo fundamenta o conceito de multiplicadores em hardware digital:

Figura 4. Multiplicação entre dois vetores binários



Fonte: (João Ranhel)

Nesse sentido, Tocci sintetiza que a maioria das máquinas digitais pode somar apenas dois números binários de cada vez e que por conta disso os produtos parciais obtidos durante a multiplicação não podem ser somados ao mesmo tempo. Em vez disso, são somados dois de cada vez. Ou seja, o primeiro é somado ao segundo, o resultado é somado ao terceiro, e assim por diante.

Tabela 5 - Tabela Verdade Meio Somador

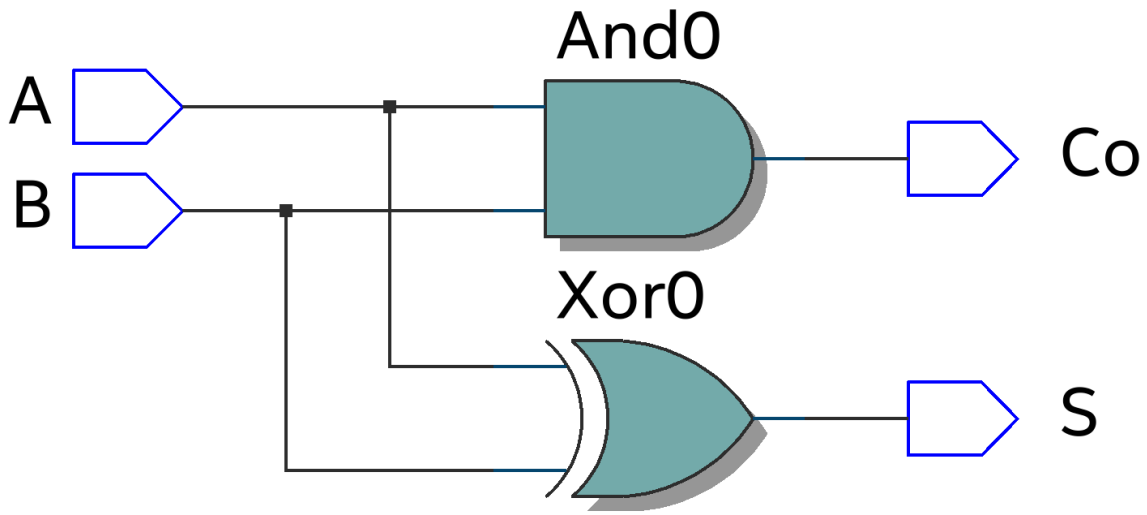
ENTRADA A	ENTRADA B	SAÍDA	CARRY OUT
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **Equações Booleanas**

  S = A ⊕ B
  Cout = AB

- **Diagrama do Circuito**

Figura 4 - Circuito Meio Somador.



Fonte: compilação do autor.

3.1.4 DIVISORES

A divisão em base binária corresponde, em essência, à técnica da divisão longa decimal. O processo consiste em verificar quantas vezes o divisor pode ser subtraído do dividendo até que o valor residual seja inferior ao divisor. O resultado final é composto pelo quociente, que representa quantas vezes o divisor enquadrar-se no dividendo, e pelo resto, valor que permanece ao final da operação.

Tabela 6 - Tabela Verdade Mux Divisor

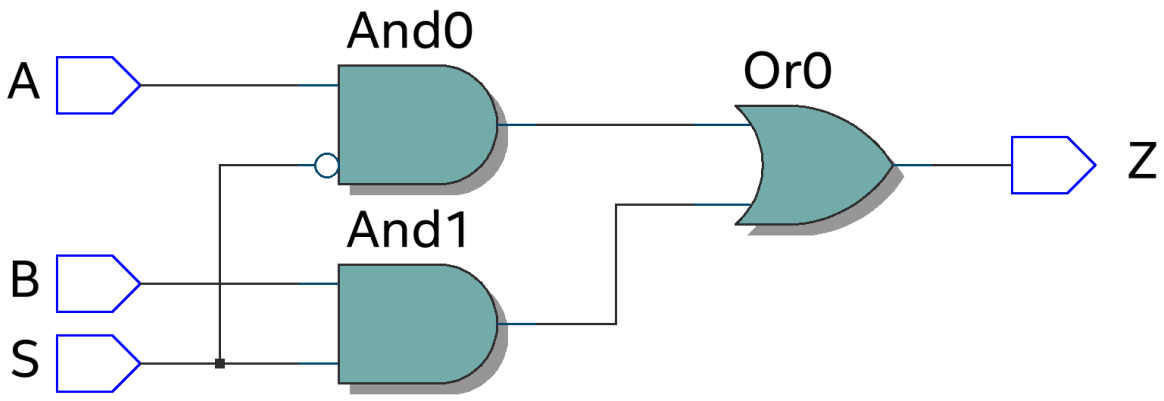
ENTRADA A	ENTRADA B	BORROW	SAÍDA
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- **Equação Booleana**

  S = Borrow ¬A ⊕ B) + B(A⊕Borrow)

- **Diagrama do Circuito**

Figura 5 - Circuito Multiplexador.



Fonte: compilação do autor.

3.2 MÓDULOS DE CONTROLE E LÓGICA

3.2.1 MULTIPLEXADORES

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 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. 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^n entradas distintas. Por exemplo:

- Um MUX 2:1 possui 2 entradas de dados, 1 linha de seleção e 1 saída.
- Um MUX 4:1 possui 4 entradas de dados, 2 linhas de seleção e 1 saída.
- Um MUX 8:1 possui 8 entradas de dados, 3 linhas de seleção e 1 saída.

Tabela 7- Tabela Verdade Multiplexador 8x1

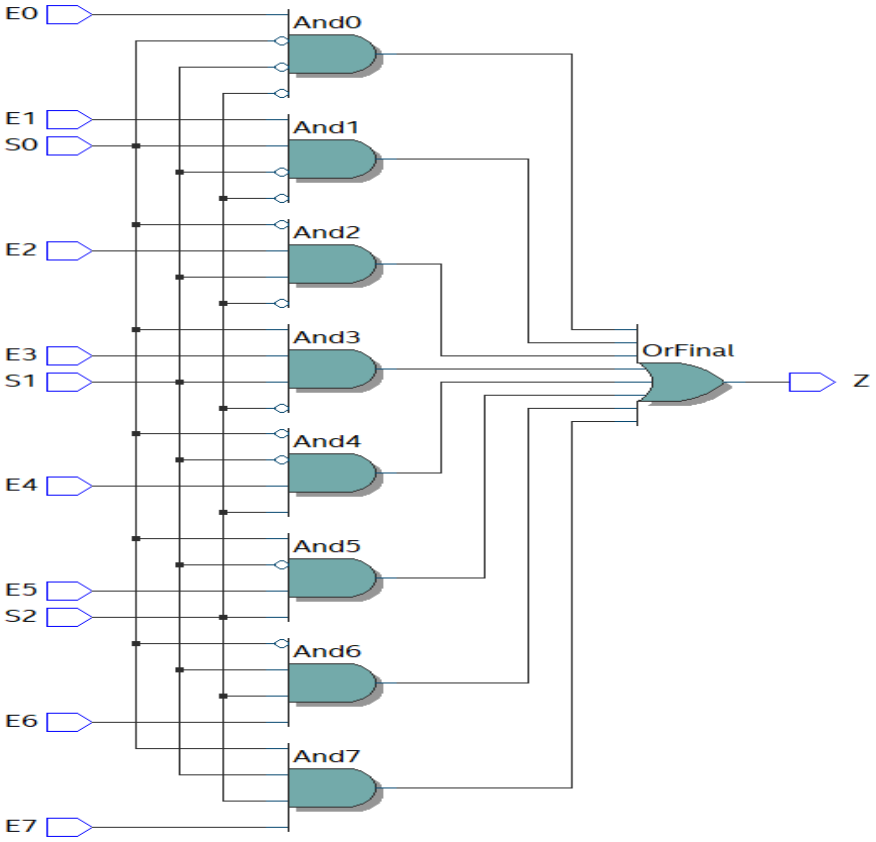
SELETOR 2 (S2)	**SELETOR 1 (S1)**	**SELETOR 0 (S0)**	**SAÍDA OPERAÇÃO (Z)**
0	0	0	Soma (Z0)
0	0	1	Subtração (Z1)
0	1	0	Multiplicação (Z2)
0	1	1	Divisão (Z3)
1	0	0	And (Z4)
1	0	1	Or (Z5)
1	1	0	Xor (Z6)
1	1	1	Sem operação (Z7)

- **Equações Booleanas**

Z = (¬S2 ¬S1 ¬S0 Z0) + (¬S2 ¬S1 S0 Z1) + (¬S2 S1¬S0 Z2) + (¬S2 S1 S0 Z3) + (S2 ¬S1 ¬S0 Z4) + (S2 ¬S1 S0 Z5) + (S2 S1 ¬S0 Z6) + (S2 S1 S0 Z7)

- **Diagrama dos Circuitos**

Figura 6 - Circuito Multiplexador 8x1



Fonte: compilação do autor.

3.2.2 DECODIFICADORES

Conceito:

> Um decodificador é um circuito lógico que recebe um conjunto de entradas que representa um número binário e ativa apenas a saída que corresponde ao número recebido. Em outras palavras, um circuito decodificador analisa as entradas, determina o número binário presente e ativa a saída correspondente ao número na entrada; todas as outras saídas permanecem desativadas (TOCCI, 2011, p. 502).

A lógica do circuito decodificador é o fundamento para o tipo de MUX utilizado no projeto, pois é possível observar que este segue a mesma ideia apresentada no conceito: receber diversas entradas numeradas (que a seguir serão explicitadas como os resultados de cada operação realizada), distinguir e ativar as saídas produzidas de forma correspondente.

Para além disso, o decodificador é essencial para a utilização do display de 7 segmentos. A fim de representar um valor qualquer nesse mecanismo, deve-se produzir uma tabela-verdade para cada segmento do display, considerando seu estado (aceso ou apagado) e nível lógico na figura. Por exemplo, é necessário se

questionar: “para gerar tal número no display, o segmento X estará em nível lógico alto ou baixo?”. A implementação em conjunto do circuito de cada segmento resultará na produção da imagem desejada. Assim, o *decoder* para display de 7 segmentos é responsável por receber as entradas do circuito e “traduzi-las” de acordo com a numeração definida em programação.

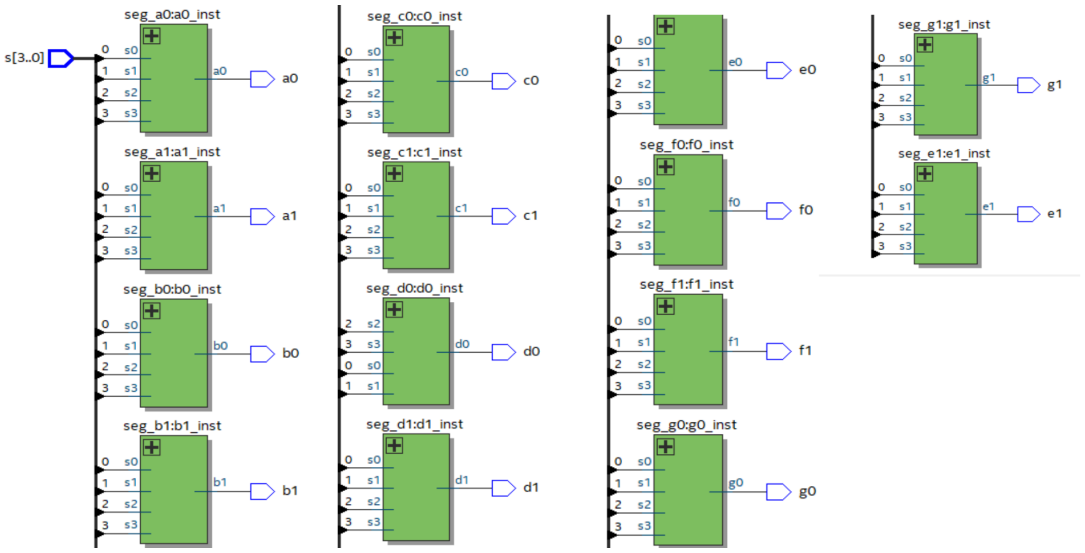
- **Equações Booleanas**

  Exemplo do segmento a0: s3's2's1's0 + s3's2s1's0' + s3s2's1s0 + s3s2s1s0'

  O segmento a0 corresponde ao segmento “a” da casa das unidades. Dessa forma, ele só obtém nível lógico alto (apagado) em valores iguais a: 1, 4, 11 e 14.

- **Diagrama do Circuito**

Figura 8 - Circuito do decodificador.



Fonte: compilação do autor.

3.2.3 FLAG ZERO

A flag de zero (Z) tem como função indicar que o resultado de uma operação da ULA é 0 (zero), tendo, portanto, nível lógico 1 nesse tipo de situação (TOCCI, 2011, p. 303).

Tabela 8 - Tabela Verdade da Flag Zero

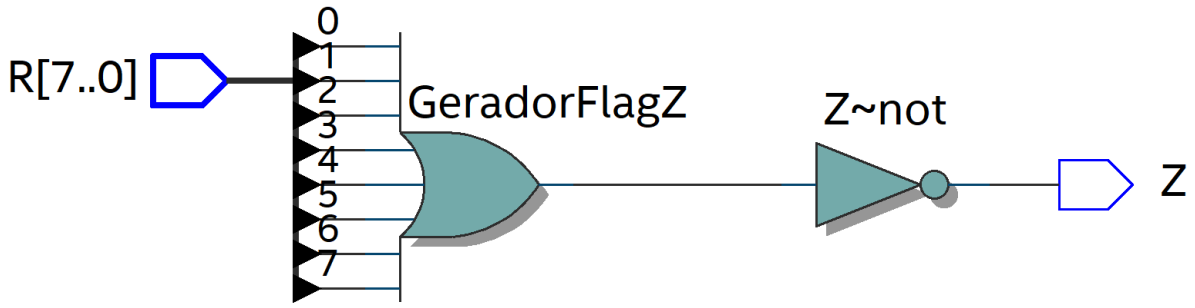
S[7]	S[6]	S[5]	S[4]	S[3]	S[2]	S[2]	S[1]	S[0]	CASO ZERO
0	0	0	0	0	0	0	0	0	1

- **Equações Booleanas**

  Z = NOR(S[7], S[6], S[5], S[4], S[3], S[2], S[1], S[0])

- **Diagrama do Circuito**

Figura 9 - Circuito da Flag Zero



Fonte: compilação do autor.

3.2.4 FLAG OVERFLOW

Flag de Overflow (O) é fundamentada para ter nível lógico alto em casos de operações que ultrapassem o limite de memória da ULA, sendo assim, mostrando que não é possível alcançar um resultado preciso com a máquina construída. Por exemplo, em uma ULA de limite de 4 bits, uma operação cujo resultado ultrapasse esse valor deve acionar a flag de Overflow.

Tabela 9 - Tabela Verdade da Flag Overflow.

Numero	S7	S6	S5	S4	S3	S2	S1	S0	OVERFLOW?
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1	1	0
4	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	1	0	0
7	0	0	0	0	0	1	1	1	0
8	0	0	0	0	1	0	0	0	0
9	0	0	0	0	1	0	0	1	0
10	0	0	0	0	1	0	1	0	0
11	0	0	0	0	1	0	1	1	0
12	0	0	0	0	1	1	0	0	0
13	0	0	0	0	1	1	0	1	0
14	0	0	0	0	1	1	1	0	0
15	0	0	0	0	1	1	1	1	0

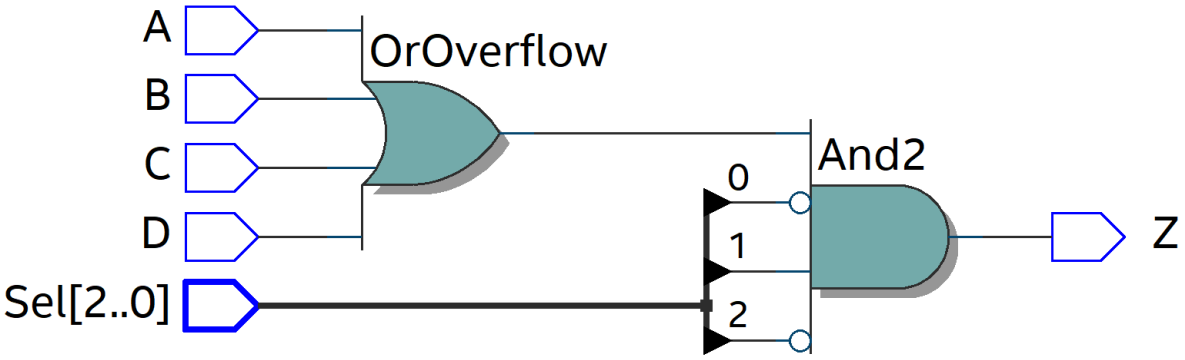
Numero	SEL 2	SEL 1	SEL 0	OVERFLOW	FLAG ON
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

- **Equações Booleanas**

  Overflow = S[7] + S[6] + S[5] + S[4]
  Flag on = ¬SEL[2] · SEL[1] · ¬SEL[0] · Overflow

- **Diagrama do Circuito**

Figura 10 - Circuito da Flag Overflow



Fonte: Compilação do autor

3.2.5 FLAG ERRO

Flag de Erro (E): Houve o consenso em sessão tutorial de que a flag de Erro deveria ser usada em casos de divisão por zero, haja vista que é impossível determinar um valor válido para tal operação.

Tabela 10 - Tabela Verdade da Flag Erro

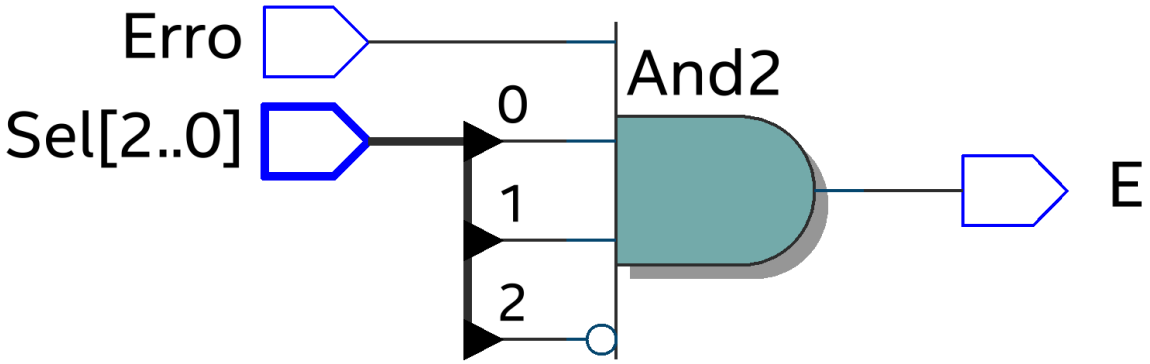
Numero	SEL 2	SEL 1	SEL 0		ERRO		FLAG
0	0	0	0		0		0
1	0	0	0		1		0
2	0	0	1		0		0
3	0	0	1		1		0
4	0	1	0		0		0
5	0	1	0		1		0
6	0	1	1		0		0
7	0	1	1		1		1
8	1	0	0		0		0
9	1	0	0		1		0
10	1	0	1		0		0
11	1	0	1		1		0
12	1	1	0		0		0
13	1	1	0		1		0
14	1	1	1		0		0
15	1	1	1		1		0

- **Equação Booleana**

  Flag on = ¬SEL[2] · SEL[1] · SEL[0] · Erro

- **Diagrama do Circuito**

Figura 11 - Circuito da Flag Erro



Fonte: compilação do autor

3.2.6 FLAG CARRY

Flag de Carry Out (C): Indica estouro em números sem sinal. Nesse caso, opera em nível lógica alto quando houver algum carry que não pôde ser computado no resultado final.

Tabela 11 - Tabela Verdade da Flag Carry

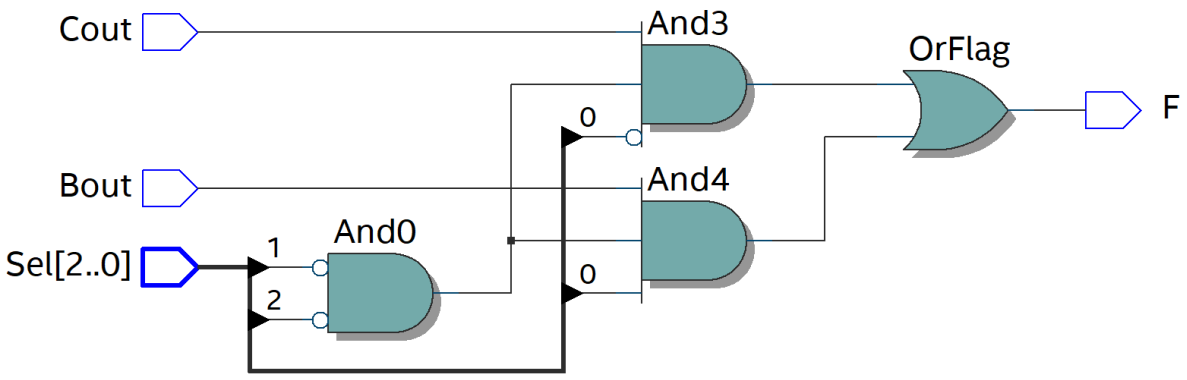
Indice	SEL 2	SEL1	SEL 0	COUT	BOUT		FLAG ON
0	0	0	0	0	0		0
1	0	0	0	0	1		0
2	0	0	0	1	0		1
3	0	0	0	1	1		1
4	0	0	1	0	0		0
5	0	0	1	0	1		1
6	0	0	1	1	0		0
7	0	0	1	1	1		1
8	0	1	0	0	0		0
9	0	1	0	0	1		0
10	0	1	0	1	0		0
11	0	1	0	1	1		0
12	0	1	1	0	0		0
13	0	1	1	0	1		0
14	0	1	1	1	0		0
15	0	1	1	1	1		0
16	1	0	0	0	0		0
17	1	0	0	0	1		0
18	1	0	0	1	0		0
19	1	0	0	1	1		0
20	1	0	1	0	0		0
21	1	0	1	0	1		0
22	1	0	1	1	0		0
23	1	0	1	1	1		0
24	1	1	0	0	0		0
25	1	1	0	0	1		0
26	1	1	0	1	0		0
27	1	1	0	1	1		0
28	1	1	1	0	0		0
29	1	1	1	0	1		0
30	1	1	1	1	0		0
31	1	1	1	1	1		0

- **Equações Booleanas :**

  Flag on = ¬SEL[2] ¬SEL[1] ¬SEL[0] Cout + ¬SEL[2] ¬SEL[1] ¬SEL[0] Bout

- **Diagrama do Circuito**

Figura 12 - Circuito da Flag Carry



Fonte: compilação do autor

4. DESCRIÇÃO DA SOLUÇÃO PROPOSTA

A Unidade Lógica e Aritmética foi implementada como um sistema puramente combinacional, onde cada função é encapsulada em um módulo Verilog estrutural. A seguir, detalha-se a interconexão desses módulos, a arquitetura interna dos componentes mais complexos e o mapeamento dos periféricos de entrada e saída da placa DE10-Lite.

4.1 Conexão dos Módulos e Fluxo de Dados

A arquitetura do sistema, conforme ilustrado no diagrama de blocos da seção anterior, baseia-se em um fluxo de dados paralelo. As entradas principais (operandos A e B (SW[8:5] e SW[4:1]) e o Carry-In (SW[0])) são distribuídas simultaneamente para todos os módulos operacionais. Cada um destes módulos (somador4x4, subtrator4x4, etc.) processa os dados de forma independente e contínua.

O resultado de cada operação é então direcionado para o multiplexador8x8. Este módulo de controle, governado pelas chaves de seleção (SW[9], KEY[0] e KEY[1]), seleciona um único resultado de 8 bits e o envia para a saída principal S. Esta saída, por sua vez, alimenta a lógica das flags e o sistema de display.

4.2 Arquitetura dos Módulos Funcionais

- **Somador e Subtrator:** Os módulos somador4x4 e subtrator4x4 foram implementados utilizando a técnica de propagação de carry/borrow ("ripple"). Cada um é composto por quatro instâncias de um módulo base de 1 bit (somadorbase ou subtratorbase), conectados em cascata.
- **Multiplicador:** O multiplicador4x4 foi projetado com base na arquitetura de um multiplicador em arranjo ("array multiplier"). Esta estrutura consiste em uma matriz de portas AND para a geração dos produtos parciais, seguida por um arranjo de somadores de 1 bit (somadorbase e meiosomador) que somam estes produtos, respeitando os deslocamentos necessários para produzir o resultado final de 8 bits, conforme o diagrama do circuito apresentado.

- **Divisor:** A operação de divisão foi implementada através do algoritmo de "Divisão com Restauração" combinacional. O módulo divisor4x4 é composto por quatro estágios (estagio_divisao) em cascata. Cada estágio utiliza um subtrator e um multiplexador para calcular um bit do quociente e o resto parcial que será propagado para o estágio seguinte.

- **Operações Lógicas (AND, OR e XOR):** As operações lógicas básicas foram implementadas de forma direta e paralela. Para cada operador, foi criado um módulo que agrupa quatro instâncias da respectiva porta lógica de 1 bit

(AND, OR ou XOR), operando bit a bit sobre as entradas A e B. Dessa forma, o módulo and4x4, por exemplo, utiliza quatro portas AND para produzir a saída de 4 bits. O mesmo raciocínio foi aplicado aos módulos or4x4 e xor4x4.

- **Decodificador para Display de 7 Segmentos:** A exibição do resultado da ULA foi realizada por meio de um decodificador combinacional dedicado aos displays de 7 segmentos presentes na placa DE10-Lite. O decodificador recebe como entrada os 4 bits menos significativos da saída da ULA (S[3:0]) e gera os sinais necessários para representar valores decimais de 0 a 15.

- **Flags de Status:** O conjunto de flags foi implementado como módulos auxiliares responsáveis por indicar condições específicas do resultado da ULA. Cada flag é gerada de forma combinacional, a partir da saída principal ou de sinais intermediários, passando pela seleção da operação em que deve estar ativa, e sendo exibida diretamente nos LEDs da placa. A flag Zero (Z) é ativada quando o barramento de saída S assume o valor 00000000, indicando que o resultado da operação foi zero. A flag Carry/Borrow (C) atua exclusivamente nas operações aritméticas de soma e subtração, sinalizando, respectivamente, o transporte de saída ou o empréstimo. A flag Overflow (O) é responsável por identificar resultados que excedem a faixa de representação definida, sendo especialmente relevante na operação de multiplicação. Por fim, a flag Erro (E) foi dedicada à detecção da divisão por zero, representando a única condição inválida do sistema.

5. ANÁLISE DOS RESULTADOS

5.1 Resultados Funcionais

Para a validação funcional, foi executada uma série de casos de teste, abrangendo todas as operações da ULA 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.

As operações de soma e subtração demonstraram a correta propagação dos sinais de *Carry* e *Borrow*, respectivamente, com a flag C sendo ativada adequadamente em casos de estouro em números sem sinal. As operações lógicas (AND, OR, XOR) produziram os resultados bit a bit corretos. O multiplicador em arranjo e o divisor com restauração também foram validados com sucesso, com as flags de *Overflow* e *Erro* sendo acionadas nas condições projetadas (multiplicação resultando em valor >15 e divisão por zero). A flag Z foi ativada corretamente em todas as operações cujo resultado era zero.

A correspondência entre os resultados simulados e os observados na placa valida o produto final do circuito lógico desenvolvido com sua implementação

estrutural de todos os módulos. Portanto, concluímos que o protótipo atendeu a todos os requisitos funcionais estabelecidos no problema.

5.2 Resultados de Síntese (Uso de Recursos)

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 da ULA no Quartus, o relatório de síntese ("Fitter" -> "Resource Usage Summary") forneceu os dados sobre a utilização de recursos do chip. Conforme a figura abaixo, o projeto completo utilizou 100 elementos lógicos totais.

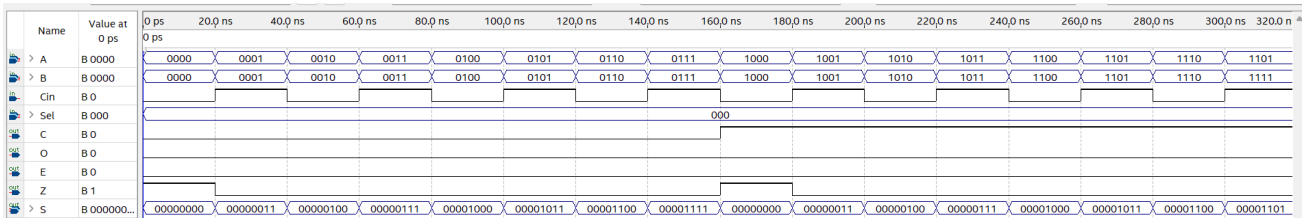
Figura 13 - Relatório de síntese

Resource	Usage
Total logic elements	100 / 49,760 (< 1 %)
-- Combinational with no register	100
-- Register only	0
-- Combinational with a register	0

Fonte: Quartus Prime Lite

5.3 Testes e Validação

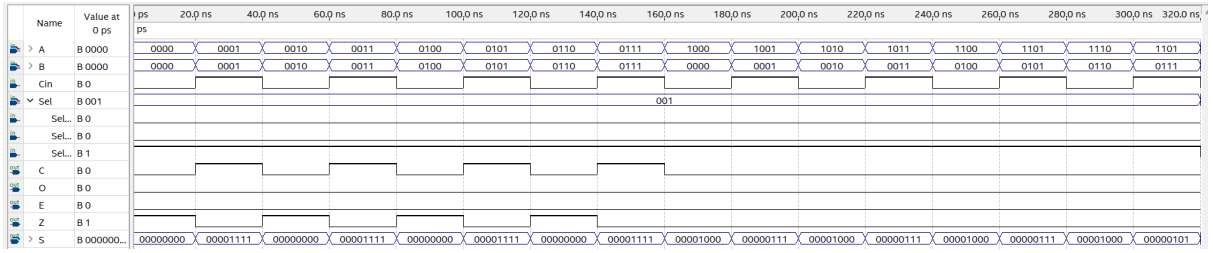
Figura 14 - Waveform para a soma



Fonte: Quartus Prime Lite

Por meio de arquivos de forma de onda é possível ter ciência do estado de funcionamento do circuito. Nesse caso, o arquivo está configurado para testar o módulo somador, que apresentou resultados satisfatórios.

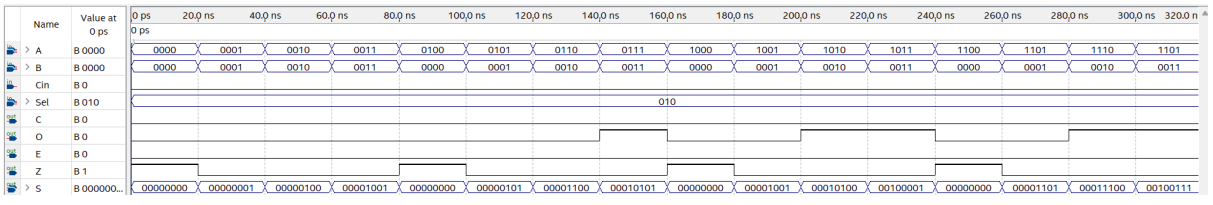
Figura 15 - Waveform para a subtração



Fonte: Quartus Prime Lite

Os testes realizados para a subtração também são satisfatórios, inclusive na identificação de valores negativos com a flag de Borrow Out (C, aproveitando o uso da flag de Carry Out), que estão fora do escopo do projeto.

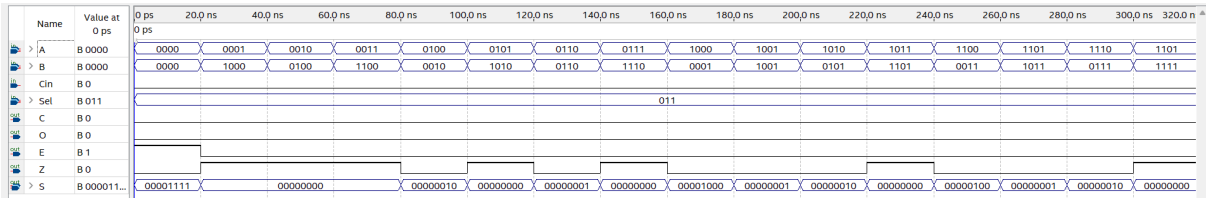
Figura 16 - Waveform para a multiplicação



Fonte: Quartus Prime Lite

O circuito multiplicador apresenta resultados consistentes e ativa as flags Zero (Z) e Overflow (O) corretamente.

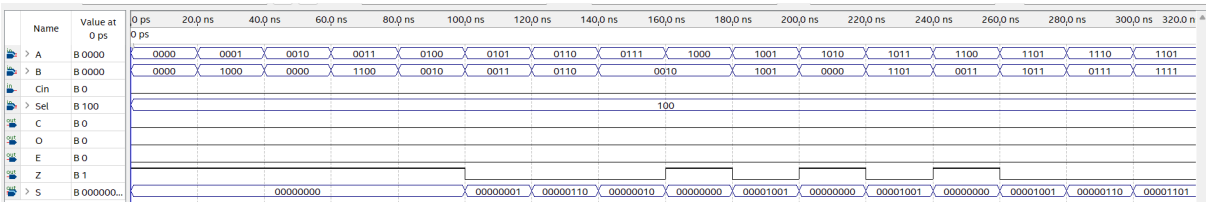
Figura 17 - Waveform para a divisão



Fonte: Quartus Prime Lite

A divisão (inteira) está correta, inclusive na ativação da flag de Erro (E) em casos de divisor igual a zero. Entretanto, os valores de resto não são exibidos, devido à ausência de pinagem das saídas correspondentes no circuito.

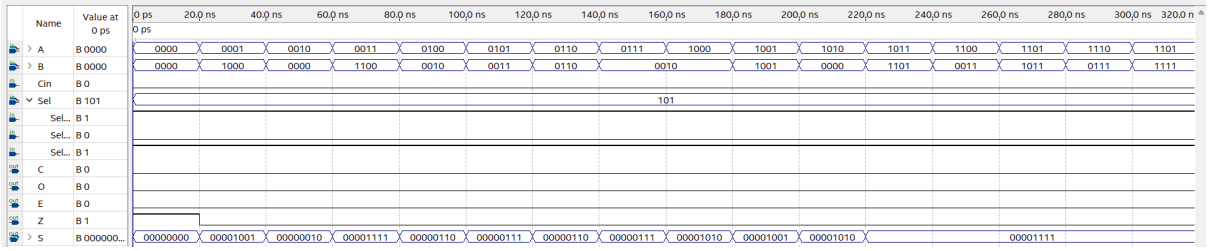
Figura 18 - Waveform para operação And



Fonte: Quartus Prime Lite

A operação And apresentou resultados conforme o esperado, tendo como valor final a soma das operações And bit a bit.

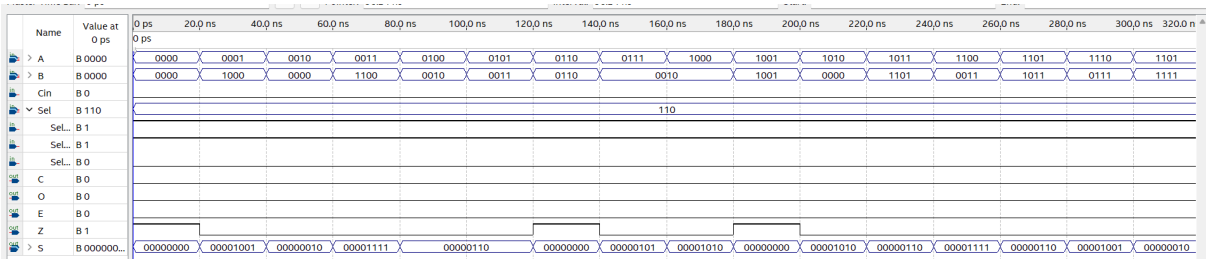
Figura 19 - Waveform da operação Or



Fonte: Quartus Prime Lite

A operação Or também operou corretamente bit a bit.

Figura 20 - Waveform da operação Xor



A operação Xor funcionou da mesma forma que as outras operações lógicas.

6. CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto permitiu a aplicação prática e aprofundada dos conceitos fundamentais de circuitos digitais, resultando em uma implementação bem-sucedida de uma Unidade Lógica e Aritmética (ULA) de 4 bits totalmente funcional. Partindo dos princípios básicos da Álgebra Booleana, cada componente do sistema foi projetado seguindo uma metodologia bem definida, desde a criação de tabelas verdade, passando pela simplificação para obtenção do circuito mínimo, até a sua implementação final em Verilog puramente estrutural.

A arquitetura modular adotada, conta com unidades de operação independentes e um multiplexador central, demonstrou-se ser a abordagem mais eficaz para gerenciar as entradas e saídas necessárias para o funcionamento adequado do protótipo. A validação do projeto, confirmada por simulação e testes práticos na placa FPGA DE10-Lite, demonstrou que todas as sete operações e as quatro flags de status operam conforme o especificado.

7. REFERÊNCIAS BIBLIOGRÁFICAS

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

RANHEL, João. *Eletrônica digital, Verilog e FPGA*. São Paulo: Clube de Autores, 2021.

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

SARMA, Subhrajit; BHUYAN, Rama. Boolean Algebra and Logic Gates. *International Journal of Mathematics Trends and Technology*, v. 65, n. 3, p. 147–151, 2019. Disponível em: <https://ijmttjournal.org/archive/ijmtt-v65i3p522>. Acesso em: 25 set. 2025.