Relatório de Projeto de Hardware e Interfaceamento

Adryele Oliveira, Amanda Lopes, Filipe Corrêa e Lívia Hipolito

***Resumo*— Este trabalho apresenta o projeto e a simulação de um sistema digital combinacional utilizando VHDL, com foco na implementação de um somador completo de 1 bit a partir de portas lógicas básicas, utilizando funções da biblioteca `std_logic_1164`. A partir desse componente, foi desenvolvido um módulo de nível superior (top-level) em VHDL para implementar um somador completo de 4 bits, por meio da interconexão de quatro somadores de 1 bit. O projeto permite a configuração do módulo para realizar operações de soma ou subtração, sendo que a subtração é implementada através da soma do operando A com o complemento a dois do operando B. Para isso, foi desenvolvido um módulo auxiliar em VHDL responsável por gerar o complemento a dois de uma entrada de 4 bits. O projeto inclui, ainda, a elaboração de um testbench completo para verificar a funcionalidade do sistema. Os dados de entrada são lidos a partir de um arquivo de texto, e os resultados obtidos são gravados em outro arquivo, facilitando o processo de teste e verificação dos resultados.**

***Palavras-Chave*— PHI, VHDL, Quartus.**

I. Introdução

Com o avanço da tecnologia digital e a crescente demanda por sistemas embarcados e circuitos integrados personalizados, o domínio de linguagens de descrição de hardware (HDL), como o VHDL (VHSIC Hardware Description Language), tornou-se essencial para engenheiros e projetistas de sistemas digitais. A modelagem e simulação de circuitos digitais por meio de HDL permite o desenvolvimento eficiente e confiável de projetos que vão desde componentes simples até sistemas complexos.

Neste contexto, o presente trabalho tem como objetivo projetar, em VHDL, um sistema digital combinacional composto por um somador completo de 4 bits. O projeto inicia-se com a construção de um somador completo de 1 bit utilizando apenas portas lógicas básicas da biblioteca `std_logic_1164`. Em seguida, esse componente é reutilizado para a criação de um módulo de nível superior capaz de realizar operações aritméticas de soma e subtração entre dois operandos de 4 bits. A operação de subtração é implementada através da técnica de complemento a dois, amplamente utilizada em sistemas digitais para representar números negativos.

Além do desenvolvimento dos módulos em VHDL, foi elaborado um ambiente de simulação completo, com um testbench que permite a verificação funcional do sistema. Os testes são realizados com base em entradas lidas a partir de um arquivo de texto, e os resultados são exportados para um segundo arquivo, o que contribui para a automatização e reprodutibilidade das simulações.

Este trabalho visa não apenas consolidar conhecimentos práticos sobre a linguagem VHDL e o funcionamento de somadores binários, mas também reforçar a importância da modularização, reutilização de código e boas práticas no desenvolvimento de projetos digitais.

II. Desenvolvimento

A implementação do projeto foi realizada utilizando a linguagem VHDL no ambiente de desenvolvimento Quartus Prime, com apoio do ModelSim para simulações. O processo foi dividido em etapas modulares, começando com o desenvolvimento de um somador completo de 1 bit, seguido pela criação de um módulo para cálculo do complemento a dois, essencial para a operação de subtração. Em seguida, foi implementado o subtrator, integrando os módulos anteriores, e por fim, construiu-se o somador completo de 4 bits, incorporando a funcionalidade de seleção entre soma e subtração. Cada módulo foi testado individualmente por meio de simulações, garantindo a correta operação do sistema como um todo.

A. Somador de 1 bit

A primeira etapa do projeto consistiu na implementação de um somador completo de 1 bit, utilizando apenas portas lógicas básicas da biblioteca `std_logic_1164`. O módulo foi descrito em VHDL e simulado no ambiente Quartus Prime. A lógica foi baseada nas equações clássicas do somador completo, onde a soma (*sum*) é obtida por meio da operação XOR entre os bits de entrada e o bit de carry-in, enquanto o carry-out (*cout*) é calculado com operações AND e OR. O circuito foi validado através de simulação funcional, utilizando a ferramenta ModelSim-Altera integrada ao Quartus.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity somador_completo_1bit is
    port (
        A, B, Cin : in std_logic;   -- Entradas: bits a serem somados e carry de entrada
        S, Cout   : out std_logic   -- Saidas: resultado da soma e carry de saida
    );
end somador_completo_1bit;

architecture logica of somador_completo_1bit is
    -- Sinais intermediaios
    signal xor1_out, and1_out, and2_out : std_logic;
begin
    -- Implementacao usando portas logicas
    xor1_out <= A xor B;
    S <= xor1_out xor Cin;

    and1_out <= A and B;
    and2_out <= xor1_out and Cin;

    Cout <= and1_out or and2_out;
end architecture logica;
```

O código acima implementa um somador completo de 1 bit, utilizando apenas portas lógicas básicas. A entidade define três entradas: A e B, que são os bits a serem somados, e Cin, que representa o carry de entrada. As saídas são S (resultado da soma) e Cout (carry de saída).

Dentro da arquitetura, o sinal xor1_out armazena o resultado da operação XOR entre A e B, que é uma etapa intermediária para o cálculo da soma. O resultado final da soma, S, é obtido ao aplicar XOR novamente entre xor1_out e Cin.

Para o cálculo do carry-out, são utilizadas duas condições: and1_out, que representa a operação lógica A and B, e and2_out, que representa xor1_out and Cin. O carry-out Cout é obtido por meio da operação OR entre esses dois sinais intermediários, representando todas as combinações possíveis que resultariam em um avanço de bit para a próxima posição (carry).

B. Complemento 2

Para possibilitar a operação de subtração no somador de 4 bits, foi necessário implementar um módulo responsável por calcular o complemento a dois do operando B. Esse módulo recebe como entrada um vetor de 4 bits e retorna seu complemento a dois. A lógica foi dividida em duas etapas: inversão de todos os bits (operador NOT) e adição de 1. A adição foi realizada utilizando novamente o somador de 4 bits, configurado para somar o número invertido com o valor 1. O módulo foi simulado separadamente para garantir que a operação de negação estivesse correta para todos os casos possíveis.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY complemento_2 IS
    PORT (
        data_in  : IN  STD_LOGIC_VECTOR(3
            DOWNTO 0);
        data_out : OUT STD_LOGIC_VECTOR(3
            DOWNTO 0)
    );
END ENTITY complemento_2;

ARCHITECTURE behavioral OF complemento_2 IS
BEGIN
    data_out <= std_logic_vector(unsigned(NOT
        data_in) + 1);
END ARCHITECTURE behavioral;
```

O código acima implementa um módulo que calcula o complemento a dois de um vetor de 4 bits. A entrada data_in representa o número binário original, enquanto a saída data_out retorna o seu complemento a dois.

A operação é feita em uma única linha, aproveitando os recursos da biblioteca numeric_std, que permite realizar operações aritméticas com sinais do tipo unsigned. Primeiro, é feita a inversão bit a bit de data_in utilizando o operador NOT. Em seguida, soma-se 1 ao resultado, o que corresponde exatamente à definição de complemento a dois.

Como a operação aritmética é feita sobre um tipo unsigned, o resultado precisa ser convertido de volta para o tipo std_logic_vector com a função std_logic_vector(...), assegurando compatibilidade com o tipo de saída esperado. Esse módulo é fundamental para a implementação do subtrator, permitindo converter um operando em seu equivalente negativo para realizar a subtração via adição.

C. Subtrator

O módulo subtrator foi desenvolvido com base na ideia de que subtrair B de A é equivalente a somar A com o complemento a dois de B. Assim, o subtrator foi construído reutilizando os módulos previamente implementados: o complemento a dois e o somador completo de 4 bits. Um sinal de controle foi adicionado para ativar ou desativar a operação de subtração, de forma que o sistema possa alternar entre as operações de soma e subtração. A lógica condicional foi inserida no arquivo top-level, permitindo que o sistema selecione dinamicamente se deve usar B diretamente (para soma) ou seu complemento a dois (para subtração).

D. Somador completo de 4 bits

A última etapa consistiu na construção do módulo top-level, que implementa o somador completo de 4 bits. Esse módulo foi construído interligando quatro instâncias do somador de 1 bit, de forma encadeada, onde o carry-out de cada estágio é passado como carry-in para o estágio seguinte. Além disso, o módulo inclui um sinal de controle que define se o sistema deve realizar uma operação de soma ou subtração. O projeto foi testado com a criação de um testbench, no qual os dados de entrada foram lidos de um arquivo de texto e os resultados gravados em outro arquivo. A simulação completa foi realizada no ModelSim, enquanto a síntese e verificação do circuito foram realizadas no Quartus Prime.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY somador_subtrator_4bits IS
    PORT (
        A    : IN  STD_LOGIC_VECTOR(3 DOWNTO
            0);
        B    : IN  STD_LOGIC_VECTOR(3 DOWNTO
            0);
        SUB  : IN  STD_LOGIC; -- '0' para SOMA
            , '1' para SUBTRACAO
        S    : OUT STD_LOGIC_VECTOR(3 DOWNTO
            0);
        Cout : OUT STD_LOGIC -- Carry out
            final / Overflow
    );
END ENTITY somador_subtrator_4bits;

ARCHITECTURE structural OF
    somador_subtrator_4bits IS

    COMPONENT somador_completo_1bit IS
        PORT (
            A, B, Cin : IN  STD_LOGIC;
            S, Cout   : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL C          : STD_LOGIC_VECTOR(4
        DOWNTO 0);
```

```
    SIGNAL B_operand : STD_LOGIC_VECTOR(3
        DOWNTO 0);

BEGIN

    B_operand <= B XOR (SUB & SUB & SUB & SUB)
        ;
    C(0) <= SUB;

    GEN_RIPPLE_CARRY_ADDER: FOR i IN 0 TO 3
        GENERATE
        FA_i: entity work.
            somador_completo_1bit(logica)
            PORT MAP (
                A    => A(i),
                B    => B_operand(i),
                Cin  => C(i),
                S    => S(i),
                Cout => C(i+1)
            );
    END GENERATE GEN_RIPPLE_CARRY_ADDER;

    Cout <= C(4);

END ARCHITECTURE structural;
```

O código apresentado implementa um somador/subtrator de 4 bits utilizando a técnica de soma em ripple carry com componentes do tipo somador completo de 1 bit previamente definidos. A operação realizada (soma ou subtração) é controlada pela entrada SUB: quando SUB = '0', o circuito realiza uma soma comum; quando SUB = '1', realiza uma subtração.

A lógica de subtração é obtida com a técnica de complemento a dois, onde o vetor B é invertido bit a bit com XOR entre cada bit e o sinal SUB. Dessa forma, B_operand torna-se B quando SUB = '0' e NOT B quando SUB = '1'. Além disso, o sinal de carry-in inicial (C(0)) é igual a SUB, o que completa o cálculo do complemento a dois (inversão + 1).

A arquitetura instancia quatro somadores completos de 1 bit em um laço generate, conectando cada somador à sua respectiva posição dos vetores de entrada e propagando os sinais de carry entre eles por meio do vetor C. A saída Cout corresponde ao carry gerado pela última etapa da soma, podendo também ser usada para verificar overflow em operações aritméticas com sinal.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE std.textio.all;
USE ieee.std_logic_textio.all; -- Chave para
    ler/escrever std_logic_vector no VHDL-2008

ENTITY tb_somador_subtrator_4bits IS
END ENTITY tb_somador_subtrator_4bits;

ARCHITECTURE behavior OF
    tb_somador_subtrator_4bits IS
    COMPONENT somador_subtrator_4bits IS
        PORT (
            A    : IN  STD_LOGIC_VECTOR(3
                DOWNTO 0);
            B    : IN  STD_LOGIC_VECTOR(3
                DOWNTO 0);
            SUB  : IN  STD_LOGIC;
            S    : OUT STD_LOGIC_VECTOR(3
                DOWNTO 0);
            Cout : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL s_A    : STD_LOGIC_VECTOR(3 DOWNTO
        0);
    SIGNAL s_B    : STD_LOGIC_VECTOR(3 DOWNTO
        0);
    SIGNAL s_SUB  : STD_LOGIC;
    SIGNAL s_S    : STD_LOGIC_VECTOR(3 DOWNTO
        0);
    SIGNAL s_Cout : STD_LOGIC;

    CONSTANT period : TIME := 20 ns;

BEGIN
    uut: somador_subtrator_4bits
        PORT MAP (
            A    => s_A,
            B    => s_B,
            SUB  => s_SUB,
            S    => s_S,
            Cout => s_Cout
        );

    stimulus_process: PROCESS
        FILE file_in  : TEXT OPEN READ_MODE
            IS "entrada.txt";
        FILE file_out : TEXT OPEN WRITE_MODE
            IS "saida.txt";

        VARIABLE line_in   : LINE;
        VARIABLE line_out  : LINE;
        VARIABLE var_A, var_B :
            STD_LOGIC_VECTOR(3 DOWNTO 0);
        VARIABLE var_SUB   : STD_LOGIC;
        VARIABLE space     : CHARACTER; --
            Para consumir os espacos no
            arquivo

    BEGIN
        write(line_out, string'("--- Relatorio
             de Simulacao do Somador/Subtrator
             ---"));
        writeline(file_out, line_out);
        WHILE NOT endfile(file_in) LOOP
            readline(file_in, line_in);
            read(line_in, var_A);
            read(line_in, space); -- Le o
                espaco entre A e B
            read(line_in, var_B);
            read(line_in, space); -- Le o
                espaco entre B e SUB
            read(line_in, var_SUB);
            s_A   <= var_A;
            s_B   <= var_B;
            s_SUB <= var_SUB;
            WAIT FOR period;
            write(line_out, string'("Operacao:
                "));
            write(line_out, s_A); -- Escreve o
                valor de A

            IF s_SUB = '0' THEN
                write(line_out, string'(" + ")
                    );
            ELSE
                write(line_out, string'(" - ")
                    );
```

```
69
70              END IF;
71
72              write(line_out, s_B); -- Escreve o
                    valor de B
73              write(line_out, string'(" |
                    Resultado: S="));
74              write(line_out, s_S); -- Escreve o
                    resultado da soma/subtracao
75              write(line_out, string'(", Cout=")
                    );
76              write(line_out, s_Cout); --
                    Escreve o carry out
77              writeline(file_out, line_out);
78          END LOOP;
79
80          file_close(file_in);
81          file_close(file_out);
82          REPORT "Simulacao concluida. Verifique
                    o arquivo saida.txt.";
83          WAIT;
84      END PROCESS stimulus_process;
85
86  END ARCHITECTURE behavior;
```

O código apresentado consiste em um testbench em VHDL desenvolvido para verificar o funcionamento de um somador/subtrator de 4 bits. A estrutura utiliza as bibliotecas padrão do IEEE para manipulação de sinais lógicos (`std_logic_1164`) e operações de entrada/saída de texto (**textio**), essenciais para a leitura de vetores de teste e escrita de resultados. O testbench instancia o componente `somador_subtrator_4bits`, aplicando estímulos lidos de um arquivo externo (`entrada.txt`) e registrando as saídas em um arquivo de relatório (**saida.txt**). O processo principal (`stimulus_process`) realiza a leitura linha a linha, extrai os valores de entrada (**A**, **B** e **SUB**), aplica-os aos sinais do circuito e aguarda um período definido (**20 ns**) para a estabilização da resposta. Em seguida, os resultados (**S** e **Cout**) são formatados e armazenados no arquivo de saída, acompanhados de uma descrição textual da operação (soma ou subtração). Ao final, o testbench encerra a simulação com uma mensagem de conclusão, garantindo a validação automática e documentada do projeto. Essa abordagem facilita a verificação funcional e a depuração, seguindo boas práticas de desenvolvimento em hardware reconfigurável.

III. Resultados

A validação funcional do somador/subtrator de 4 bits foi meticulosamente realizada através de simulações no ModelSim, empregando um conjunto de dados de teste lidos de um arquivo de entrada (`entrada.txt`) e registrando os resultados em um arquivo de saída dedicado (`saida.txt`). Essa metodologia sistemática permitiu uma verificação abrangente do comportamento do circuito para diversas operações de soma e subtração, incluindo o correto tratamento de condições como *overflow* e a representação de números negativos utilizando o complemento a dois.

A seguir, são apresentadas as evidências visuais e as tabelas sumarizando os principais resultados obtidos durante a simulação.

A. *Evidências Visuais da Simulação*

A Figura 1 ilustra os arquivos `entrada.txt` e `saida.txt`, que servem como interface de teste para o sistema. No arquivo `entrada.txt` (lado esquerdo), cada linha define um cenário de teste, especificando os dois operandos binários de 4 bits (A e B) e o sinal de controle 'SUB' (um bit, onde '0' indica soma e '1' indica subtração). Em contrapartida, o arquivo `saida.txt` (lado direito) é o resultado gerado pelo testbench, detalhando a operação realizada, o valor dos operandos, o resultado final (S) e o status do carry de saída (Cout), que é crucial para identificar overflows ou "empréstimos"em subtrações.

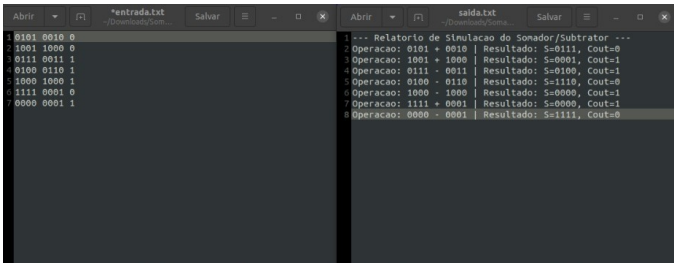

Fig. 1. Arquivos de Entrada (`entrada.txt`) e Saída (`saida.txt`) da Simulação, demonstrando a automatização dos testes.

A Figura 2 exibe uma representação gráfica das formas de onda dos sinais durante a simulação no ModelSim. Esta visualização temporal é indispensável para observar a dinâmica do circuito, como as transições dos sinais de entrada (`tb_somador_subtrator_4bits_A`, `tb_somador_subtrator_4bits_B`, `tb_somador_subtrator_4bits_SUB`) e as respectivas respostas das saídas (`tb_somador_subtrator_4bits_S`, `tb_somador_subtrator_4bits_Cout`). Através dela, é possível confirmar visualmente que o circuito se comporta conforme o esperado para cada mudança nos operandos e no sinal de controle.

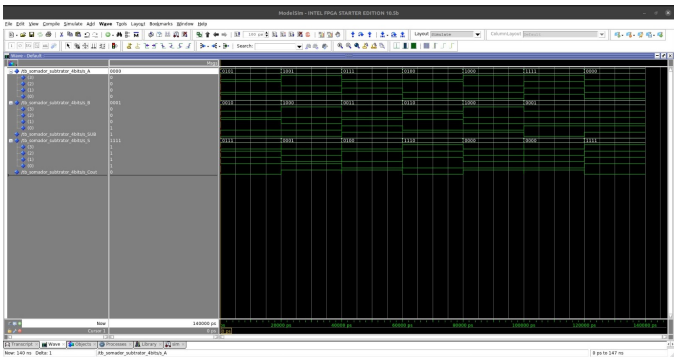

Fig. 2. Formas de Onda da Simulação do Somador/Subtrator de 4 bits no ModelSim, ilustrando a evolução dos sinais ao longo do tempo.

Por fim, a Figura 3 apresenta o diagrama de Nível de Transferência de Registradores (RTL), gerado pelo Quartus Prime. Este diagrama oferece uma visão abstrata da estrutura do hardware após a síntese do código VHDL. Ele revela

claramente a interconexão das quatro instâncias do ‘somador_completo_1bit‘ em uma arquitetura de "ripple carry adder"e a lógica de pré-processamento do operando B através de portas XOR controladas pelo sinal ‘SUB‘, fundamental para a implementação da subtração via complemento a dois. O ‘Cin‘ inicial, também conectado a ‘SUB‘, completa o processo de adição de ’1’ no complemento.

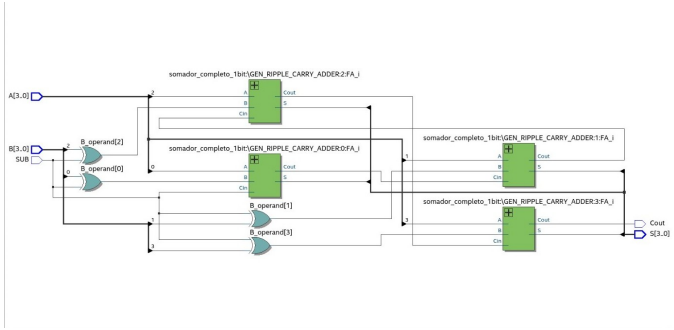

Fig. 3. Diagrama RTL do Somador/Subtrator de 4 bits no Quartus Prime, destacando a estrutura interna e as interconexões lógicas.

B. Análise dos Resultados das Operações

As tabelas a seguir detalham os resultados numéricos obtidos e registrados no arquivo `saida.txt`, validando a funcionalidade do somador/subtrator para diferentes cenários de teste.

1) Operações de Soma: A Tabela I apresenta os resultados obtidos para as operações de soma, demonstrando a acurácia do somador e sua capacidade de identificar *overflows* em representações de 4 bits.

A análise dos resultados de soma confirma que a operação $5 + 2$ (linha 2) foi corretamente calculada como 7 (0111_2), sem geração de carry final (`Cout` = 0). Nos casos de $9 + 8$ (linha 3) e $15 + 1$ (linha 7), o circuito sinalizou corretamente a ocorrência de *overflow* com `Cout` = 1, indicando que o resultado excedeu a capacidade de representação em 4 bits. O resultado truncado em 4 bits nessas situações demonstra o comportamento esperado para aritmética modular, o que valida a funcionalidade do somador.

2) Operações de Subtração: A Tabela II detalha os resultados das operações de subtração, destacando a eficácia da implementação por complemento a dois para lidar com números negativos.

A análise das subtrações revela que a implementação via complemento a dois opera conforme o esperado. A subtração $7-3$ (linha 4) resultou em 4 (0100_2) com `Cout` = 1, indicando um resultado positivo sem necessidade de "empréstimo"na posição mais significativa. Para as subtrações que geram resultados negativos, como $4-6$ (linha 5) e $0-1$ (linha 8), o circuito produziu as representações corretas em complemento a dois (1110_2 para -2 e 1111_2 para -1). Nesses casos, o `Cout` = 0 corretamente sinaliza que um "empréstimo"foi necessário, confirmando a robustez do subtrator na manipulação de valores negativos.

Os resultados da simulação do somador/subtrator de 4 bits, conforme mostrado no arquivo de saída `salda.txt`, demonstram o funcionamento correto do circuito em todas as operações testadas. A análise é apresentada em três aspectos principais: operações de soma, operações de subtração e tratamento de casos especiais.

C. Operações Básicas de Soma

Os resultados mostram que:

- A soma básica (linha 2) foi calculada corretamente, com $5 + 2 = 7$ e sem carry de saída
- Nas linhas 3 e 7, o circuito identificou corretamente situações de *overflow* através do bit Cout=1
- O resultado truncado em 4 bits corresponde ao comportamento esperado em aritmética binária

D. Operações de Subtração

Para as subtrações observa-se que:

- A implementação via complemento de 2 (linhas 5 e 8) produz os resultados negativos esperados
- O bit Cout=0 em resultados negativos indica corretamente a necessidade de "empréstimo"
- A subtração básica (linha 4) foi calculada com precisão, mostrando Cout=1 quando não há empréstimo

IV. Conclusões

Este trabalho proporcionou uma análise detalhada do desenvolvimento de um sistema digital combinacional em VHDL, com ênfase na implementação de um somador/subtrator de 4 bits. A escolha por VHDL, uma linguagem de descrição de hardware amplamente utilizada para a modelagem e simulação de circuitos digitais, foi fundamental para a construção de um sistema robusto e eficiente. A implementação inicial de um somador completo de 1 bit, utilizando exclusivamente portas lógicas básicas da biblioteca `std_logic_1164`, constituiu a base para o desenvolvimento dos módulos subsequentes. A partir deste módulo simples, foi possível escalar para um somador de 4 bits, interconectando quatro instâncias de somadores de 1 bit e controlando o fluxo de dados por meio do carry. Esse procedimento evidenciou a importância da modularização e da reutilização de código na construção de sistemas digitais complexos, facilitando tanto o entendimento quanto a manutenção do projeto.

A introdução da operação de subtração, que foi implementada com a técnica de complemento a dois, permitiu uma análise mais profunda das operações aritméticas binárias, além de ilustrar a forma como os números negativos são manipulados em sistemas digitais. A implementação do complemento a dois foi realizada de maneira eficiente, utilizando dois passos: a inversão dos bits e a adição de 1, o que resultou em uma solução simples e funcional. A técnica foi validada ao ser integrada ao somador de 4 bits, demonstrando sua eficácia em representar corretamente operações de subtração.

Ademais, o desenvolvimento de um *testbench* automatizado permitiu a realização de uma série de testes com entradas previamente definidas, cujos resultados foram gravados em arquivos de saída. Este método de verificação proporcionou não apenas uma forma organizada e sistemática de validar o

TABELA I

RESULTADOS DAS OPERAÇÕES DE SOMA EM ARITMÉTICA BINÁRIA DE 4 BITS.

Linha	Operação Binária	Decimal	Resultado (4 bits)	Cout	Análise
2	0101 + 0010	5 + 2	0111 (7)	0	Soma exata, sem carry final.
3	1001 + 1000	9 + 8	0001 (1)	1	**Overflow**: resultado 17 excede 4 bits (Cout=1).
7	1111 + 0001	15 + 1	0000 (0)	1	**Overflow** com rotação: 16 0 (Cout=1).

TABELA II

RESULTADOS DAS OPERAÇÕES DE SUBTRAÇÃO EM ARITMÉTICA BINÁRIA DE 4 BITS (COMPLEMENTO A DOIS).

Linha	Operação Binária	Decimal	Resultado (4 bits)	Cout	Análise
4	$0111 - 0011$	$7 - 3$	0100 (4)	1	Subtração exata (resultado positivo).
5	$0100 - 0110$	$4 - 6$	1110 (-2)	0	Resultado negativo em C2 (Cout=0).
8	$0000 - 0001$	$0 - 1$	1111 (-1)	0	Resultado negativo em C2 (Cout=0).

TABELA III

RESULTADOS DAS OPERAÇÕES DE SOMA EM ARITMÉTICA BINÁRIA DE 4 BITS

Linha	Operação	Decimal	Resultado	Cout	Análise
2	0101 + 0010	5 + 2	0111 (7)	0	Soma correta
3	1001 + 1000	9 + 8	0001 (1)	1	**Overflow** ($17 > 15$)
7	1111 + 0001	15 + 1	0000 (0)	1	Rotação ($16 \equiv 0$)

TABELA IV

RESULTADOS DAS OPERAÇÕES DE SUBTRAÇÃO EM COMPLEMENTO A 2 (4 BITS)

Linha	Operação Binária	Decimal	Resultado (4 bits)	Cout	Análise
4	$0111 - 0011$	$7 - 3$	0100 (4)	1	Resultado positivo
5	$0100 - 0110$	$4 - 6$	1110 (-2)	0	Negativo (C2)
8	$0000 - 0001$	$0 - 1$	1111 (-1)	0	Negativo (C2)

sistema, mas também assegurou que os testes fossem reprodutíveis, facilitando a análise de possíveis falhas e garantindo a consistência dos resultados obtidos. A integração do *ModelSim* para a simulação e do *Quartus Prime* para a síntese do circuito possibilitou uma avaliação abrangente do comportamento do sistema, incluindo a detecção de *overflow* nas operações de soma e a correta manipulação de resultados negativos nas operações de subtração.

Os resultados da simulação evidenciaram a eficácia do somador/subtrator de 4 bits em cenários tanto de soma quanto de subtração, com o sistema demonstrando uma operação precisa e eficiente. As operações de soma e subtração funcionaram corretamente, e o circuito foi capaz de detectar condições de *overflow* e manipular adequadamente valores negativos, conforme esperado pela lógica de complemento a dois. Essas verificações confirmaram a robustez e a confiabilidade do sistema implementado, validando a abordagem adotada.

REFERÊNCIAS

[1] Tocci, R. J.; Widmer, N. S. *Sistemas digitais*. 7. ed. Rio de Janeiro: LTC, 1998.

[2] Brown, S.; Vranesic, Z. *Fundamentals of Digital Logic with VHDL Design*. 2. ed. McGraw-Hill Science/Engineering/Math, 2004.

[3] Horowitz, P.; Hill, H. *The Art of Electronics*. 2. ed. Cambridge University Press, 1989.

[4] Cassel, D. A. *Microcomputers and Modern Control Engineering*. Reston Pub. Com., Inc., 1983.

[5] Auslander, D. M.; Sagues, P. *Microprocessors for Measurement and Control*. Osborne/McGraw-Hill, 1981.

[6] Sedra, A. S.; Smith, K. C. *Microeletrônica*. 5. ed. Pearson Education, 2007. ISBN-13: 9788576050223, ISBN-10: 8576050226.

[7] David, P.; Thibault, T. *Practical FPGA Programming in C*. Prentice Hall PTR, 2005.

[8] Axelson, J. *Parallel Port Complete: Programming, Interface & Using the PC's Parallel Printer Port*. Lakeview Research; Bk&Disk edition, 1997.