Unidade Lógico Aritmética (ULA)

Adriana Raffaella dos Santos Fonseca

[1]Escola Superior de Tecnologia – Universidade do Estado do Amazonas (UEA)
Manaus – AM – Brasil

ardsf.eng23@uea.edu.br

1. Introdução

Uma Unidade Lógico Aritmética (ULA) é um bloco digital fundamental presente em processadores (CPUs) e outros circuitos digitais. Sua principal função é executar operações aritméticas (como soma e subtração) e lógicas (como AND, OR e NOT) com dados binários. A ULA é um componente essencial para o funcionamento de qualquer sistema computacional, pois é responsável por grande parte do processamento de dados [Pedro Souza 2020c]. Este documento descreve a implementação de uma ULA de 8 bits em Verilog, detalhando seu funcionamento, a descrição do código e os testes realizados.

2. Funcionamento da ULA

A ULA implementada neste projeto é uma ULA de 8 bits capaz de realizar quatro operações distintas: soma, AND, OR e NOT [Pedro Souza 2020e]. Ela recebe dois operandos de 8 bits, A e B, e dois bits de seleção, x e y, que determinam qual operação será executada. Além da saída do resultado da operação, a ULA também gera um conjunto de flags que indicam características importantes do resultado, como zero, negativo, carry e overflow.

As operações são selecionadas da seguinte forma:

x	y	Operação
0	0	Soma
0	1	AND
1	0	OR
1	1	NOT

Para a operação de soma, os operandos são tratados como números com sinal (complemento de dois), permitindo a correta detecção de overflow e carry.

Os flags de saída são definidos da seguinte maneira:

- flag[0] (Negativo): Indica se o bit mais significativo (MSB) do resultado é 1, significando um número negativo em complemento de dois.
- flag[1] (Zero): Indica se o resultado da operação é zero.
- flag[2] (Overflow): Ativado apenas na operação de soma. Indica se ocorreu um overflow durante a soma, ou seja, se o resultado excedeu a capacidade de representação de 8 bits com sinal. Isso acontece quando a soma de dois números positivos resulta em um negativo, ou a soma de dois negativos resulta em um positivo.
- flag[3] (Carry): Ativado apenas na operação de soma. Indica se houve um carry para fora do bit mais significativo durante a soma.

3. Descrição em verilog

A ULA é implementada em dois módulos Verilog: ULA_8b e mux_4x1.

3.1. Módulo ULA_8b

Este é o módulo principal da ULA.

```
module ULA_8b(
input [7:0] A, B,
input x, y,
output [3:0] flag,
output [7:0] saida
);

    // Uso do signed
    wire signed [7:0] As = A, Bs = B;
    wire signed [8:0] soma_ext;  // bit extra para carry
    wire [7:0] w0, w1, w2, w3, s;

    assign soma_ext = As + Bs;
    assign w0 = soma_ext[7:0];

    assign w1 = A & B;
    assign w2 = A | B;
    assign w3 = ~A;

    mux_4x1 muxout(
        .i0(w0), .i1(w1), .i2(w2), .i3(w3), .s1(x), .s0(y), .f(s)
    );

    assign saida = s;

    // Flags
    wire is_soma = ~x & ~y;
    assign flag[1] = (s == 8'b0);                    // Zero
    assign flag[0] = s[7];                           // Negativo (bit de
       sinal)
    assign flag[3] = is_soma ? soma_ext[8] : 1'b0; // Carry
    assign flag[2] = is_soma ? (As[7] == Bs[7] && s[7] != As[7]) : 1'b0
       ; // Overflow

endmodule
```

- Entradas: A e B (8 bits para os operandos), x e y (bits de seleção da operação).
- Saídas: flag (4 bits para os flags), saida (8 bits para o resultado da operação).
- Operações:
  - A soma é realizada com As e Bs que são as versões com sinal de A e B, e o resultado soma_ext é estendido para 9 bits para capturar o carry. O resultado da soma (w0) é os 8 bits menos significativos de soma_ext.
  - As operações lógicas AND (w1), OR (w2) e NOT (w3) são realizadas diretamente nos operandos A e B.
- Multiplexador: Um multiplexador mux_4x1 seleciona qual das operações (w0, w1, w2, w3) será a saída final s, com base nos bits de seleção x e y.

- Cálculo dos Flags:
  - is_soma: Uma wire auxiliar que é verdadeira (1'b1) se a operação selecionada for soma (x = 0, y = 0).
  - Zero (flag[1]): É setado se s (a saída final) for igual a 0.
  - Negativo (flag[0]): É setado se o bit mais significativo de s (s[7]) for 1.
  - Carry (flag[3]): É setado como o nono bit de soma_ext (soma_ext[8]), mas somente se a operação atual for soma. Caso contrário, é 0.
  - Overflow (flag[2]): É setado se a operação for soma e ocorrer um overflow. A lógica de overflow para soma de números com sinal é: se os sinais dos operandos As e Bs forem iguais, mas o sinal do resultado s for diferente, então ocorreu overflow.

3.2. Módulo mux_4x1

No contexto da Unidade Lógico Aritmética (ULA) apresentada, o módulo mux_4x1 atua como um multiplexador de 4 para 1 [Pedro Souza 2020a], sendo responsável por selecionar uma das quatro possíveis saídas de operações (soma, AND, OR, NOT) e direcioná-la para a saída final da ULA.

```
1   module mux_4x1(i0, i1, i2, i3, s1, s0, f);
2   input [7:0] i0, i1, i2, i3;
3   input s1, s0;
4   output [7:0] f;
5
6       assign f[7] = ~s1&(~s0)&i0[7] | ~s1&(s0)&i1[7] | s1&(~s0)&i2[7]
            | s1&(s0)&i3[7];
7       assign f[6] = ~s1&(~s0)&i0[6] | ~s1&(s0)&i1[6] | s1&(~s0)&i2[6]
            | s1&(s0)&i3[6];
8       assign f[5] = ~s1&(~s0)&i0[5] | ~s1&(s0)&i1[5] | s1&(~s0)&i2[5]
            | s1&(s0)&i3[5];
9       assign f[4] = ~s1&(~s0)&i0[4] | ~s1&(s0)&i1[4] | s1&(~s0)&i2[4]
            | s1&(s0)&i3[4];
10      assign f[3] = ~s1&(~s0)&i0[3] | ~s1&(s0)&i1[3] | s1&(~s0)&i2[3]
            | s1&(s0)&i3[3];
11      assign f[2] = ~s1&(~s0)&i0[2] | ~s1&(s0)&i1[2] | s1&(~s0)&i2[2]
            | s1&(s0)&i3[2];
12      assign f[1] = ~s1&(~s0)&i0[1] | ~s1&(s0)&i1[1] | s1&(~s0)&i2[1]
            | s1&(s0)&i3[1];
13      assign f[0] = ~s1&(~s0)&i0[0] | ~s1&(s0)&i1[0] | s1&(~s0)&i2[0]
            | s1&(s0)&i3[0];
14  endmodule
```

A implementação em Verilog do mux_4x1 mostra que a seleção é feita bit a bit. Por exemplo, para o bit 7 da saída (f[7]), ele é o resultado de uma lógica OR de quatro termos. Cada termo representa a combinação dos bits de seleção com um bit correspondente de uma das entradas de dados. Isso significa que, se os bits de seleção forem 00, apenas o termo envolvendo i0[7] será ativado, e assim por diante para cada um dos 8 bits da saída [Pedro Souza 2020b].

- Entradas: i0, i1, i2, i3 (entradas de 8 bits), s1, s0 (bits de seleção).
- Saída: f (saída de 8 bits).

- Funcionalidade: Este multiplexador 4x1 seleciona uma das quatro entradas de 8 bits (i0 a i3) e a direciona para a saída f, com base na combinação dos bits de seleção s1 e s0. Cada bit da saída f é selecionado individualmente, garantindo a seleção correta para toda a palavra de 8 bits.

4. Teste da ULA

Para verificar o correto funcionamento da ULA, foi criado um testbench (tb_ULA_8b) em Verilog. Este testbench instancia a ULA e aplica uma série de entradas de teste, verificando as saídas e os flags para cada caso.

4.1. Módulo tb_ULA_8b

```
`timescale 1ns/100ps

module tb_ULA_8b;

    reg [7:0] A_tb, B_tb;
    reg x_tb, y_tb;
    wire [3:0] flag_tb;
    wire [7:0] saida_tb;

    ULA_8b uut (
        .A(A_tb), .B(B_tb), .x(x_tb), .y(y_tb), .flag(flag_tb), .saida(
            saida_tb)
    );

    initial begin
        // Teste 1: 1 + 3 = 4 Flag = 0000
        #10
        A_tb = 8'b00000001;
        B_tb = 8'b00000011;
        x_tb = 0; y_tb = 0;
        #1;
        $display("Teste 1: 1 + 3 = %d | Flag = %b", saida_tb, flag_tb);
        if (saida_tb !== 8'd4)
                $display("Erro: Esperado 4");
        else
                $display("Sucesso!");

        // Teste 2: 1 - 1 = 0 (1 + -1) Flag = 0010
        #10
        A_tb = 8'b00000001;
        B_tb = 8'b11111111; // -1
        x_tb = 0; y_tb = 0;
        #1;
        $display("Teste 2: 1 + (-1) = %d | Flag = %b", saida_tb,
            flag_tb);
        if (saida_tb !== 8'd0)
                $display("Erro: Esperado 0");
        else
                $display("Sucesso!");

        // Teste 3: 127 + 127 = -2 (Overflow esperado) Flag = 0101
        #10
```

```
41          A_tb = 8'b01111111;
42          B_tb = 8'b01111111;
43          x_tb = 0; y_tb = 0;
44          #1;
45          $display("Teste 3: 127 + 127 = %d | Flag = %b", $signed(
                saida_tb), flag_tb);
46          if (saida_tb !== 8'b11111110)
47                  $display("Erro: Esperado -2 (overflow)");
48          else
49                  $display("Sucesso!");
50
51          // Teste 4: -3 + 1 = -2 Flag = 1001
52          #10
53          A_tb = 8'b11111101; // -3
54          B_tb = 8'b00000001; // 1
55          x_tb = 0; y_tb = 0;
56          #1;
57          $display("Teste 4: -3 + 1 = %d | Flag = %b", $signed(saida_tb),
                flag_tb);
58          if (saida_tb !== 8'b11111110)
59                  $display("Erro: Esperado -2");
60          else
61                  $display("Sucesso!");
62
63          // Teste 5: A & B = 00000001 Flag = 0000
64          #10
65          A_tb = 8'b00000001;
66          B_tb = 8'b00000011;
67          x_tb = 0; y_tb = 1;
68          #1;
69          $display("Teste 5: A & B = %b | Flag = %b", saida_tb, flag_tb);
70          if (saida_tb !== 8'b00000001)
71                  $display("Erro: Esperado 00000001");
72          else
73                  $display("Sucesso!");
74
75          // Teste 6: A | B = 00000011 Flag = 0000
76          #10
77          A_tb = 8'b00000001;
78          B_tb = 8'b00000010;
79          x_tb = 1; y_tb = 0;
80          #1;
81          $display("Teste 6: A | B = %b | Flag = %b", saida_tb, flag_tb);
82          if (saida_tb !== 8'b00000011)
83                  $display("Erro: Esperado 00000011");
84          else
85                  $display("Sucesso!");
86
87          // Teste 7: ~A = 11111110 Flag = 0001
88          #10
89          A_tb = 8'b00000001;
90          x_tb = 1; y_tb = 1;
91          #1;
92          $display("Teste 7: ~A = %b | Flag = %b", saida_tb, flag_tb);
93          if (saida_tb !== 8'b11111110)
94                  $display("Erro: Esperado 11111110");
```

```
95          else
96                  $display("Sucesso!");
97      end
98  endmodule
```

O testbench executa os seguintes testes:

- Teste 1 (Soma Positiva): 1 + 3 = 4. Esperado: saida = 4, flag = 0000 (nenhum flag ativado).
- Teste 2 (Soma com Negativo - Zero): 1 + (-1) = 0. Esperado: saida = 0, flag = 0010 (apenas flag de zero ativado).
- Teste 3 (Soma com Overflow): 127 + 127 = -2. Esperado: saida = -2 (8'b11111110), flag = 0101 (flags de negativo e overflow ativados).
- Teste 4 (Soma com Negativo): -3 + 1 = -2. Esperado: saida = -2 (8'b11111110), flag = 1001 (flags de negativo e carry ativados).
- Teste 5 (AND): A = 00000001, B = 00000011. A & B = 00000001. Esperado: saida = 00000001, flag = 0000.
- Teste 6 (OR): A = 00000001, B = 00000010. A — B = 00000011. Esperado: saida = 00000011, flag = 0000.
- Teste 7 (NOT): A = 00000001. A = 11111110. Esperado: saida = 11111110, flag = 0001 (flag de negativo ativado).

A cada teste, o testbench exibe o resultado da operação e os flags correspondentes, além de uma mensagem de sucesso ou erro comparando o resultado obtido com o esperado.

```
# Teste 1: 1 + 3 =   4 | Flag = 0000
# Sucesso!
# Teste 2: 1 + (-1) =   0 | Flag = 0010
# Sucesso!
# Teste 3: 127 + 127 =   -2 | Flag = 0101
# Sucesso!
# Teste 4: -3 + 1 =   -2 | Flag = 1001
# Sucesso!
# Teste 5: A & B = 00000001 | Flag = 0000
# Sucesso!
# Teste 6: A | B = 00000011 | Flag = 0000
# Sucesso!
# Teste 7: ~A = 11111110 | Flag = 0001
# Sucesso!
```

Figura 1. Saída no terminal do ModelSim

5. Outras formas de implementar

Existem diversas abordagens para implementar uma ULA, variando em complexidade, desempenho e flexibilidade. Algumas alternativas incluem:

- Uso de estruturas case ou if-else para o multiplexador: Em vez de instanciar um módulo mux_4x1 separado com múltiplas atribuições assign (como feito no exemplo), o multiplexador pode ser implementado dentro do próprio módulo ULA_8b usando uma estrutura case ou if-else if [Pedro Souza 2020d]. Isso pode tornar o código mais conciso para um número limitado de seleções, como mostra o exemplo abaixo:

```
        always @(*) begin
            case ({x, y})
            2'b00: s = w0; // Soma
            2'b01: s = w1; // AND
            2'b10: s = w2; // OR
            2'b11: s = w3; // NOT
            default: s = 8'b0; // Valor para evitar latch
            endcase
        end
```

- Implementação de operações adicionais: A ULA pode ser estendida para incluir mais operações aritméticas (subtração, multiplicação, divisão, incremento, decremento) e lógicas (XOR, deslocamentos, rotações). Isso exigiria mais bits de seleção e mais lógica para cada nova operação.
- Design hierárquico e modular: Para ULAs muito complexas, é comum dividir a funcionalidade em módulos menores (ex: módulo para soma, módulo para lógicas, módulo para flags), e então instanciá-los e conectá-los no módulo principal da ULA. Isso melhora a organização, reusabilidade e depuração do código.

6. Conclusão

A Unidade Lógico Aritmética (ULA) é um componente vital em sistemas digitais, e sua implementação em Verilog, como demonstrado, permite a criação de um bloco funcional capaz de realizar operações aritméticas e lógicas básicas. O projeto apresentado detalha uma ULA de 8 bits com capacidade de soma, AND, OR e NOT, além de gerar flags de zero, negativo, carry e overflow, essenciais para o controle de fluxo em processadores.

Através do testbench elaborado, foi possível verificar a correção de cada operação e a ativação adequada dos flags sob diferentes condições, incluindo casos de overflow e resultados nulos. A modularidade do projeto, separando a lógica da ULA do multiplexador, contribui para a clareza e manutenibilidade do código. As alternativas de implementação discutidas, como o uso de estruturas case, parametrização e extensão de operações, evidenciam a flexibilidade do Verilog para o desenvolvimento de ULAs mais complexas e adaptáveis a diversas necessidades de design.

Referências

Pedro Souza (2020a). Multiplexador 4x1 em verilog. https://youtu.be/DD4BOkSUoFg?si=ZS6g6qV-vwoxtByt.

Pedro Souza (2020b). Multiplexadores. https://youtu.be/e70pe-Cdtfc?si=oLPpBWyT95oLqWbH.

Pedro Souza (2020c). Unidade lógico-aritmética. https://youtu.be/SemyzwoLxDo?si=usxqUa6DZtfcw0bu.

Pedro Souza (2020d). Unidade lógico-aritmética usando a abordagem comportamental. https://youtu.be/Ynymty6-5dM?si=FaJCLkKACpb70bV4.

Pedro Souza (2020e). Unidade lógico-aritmética usando abordagem por fluxo de dados. https://youtu.be/wmm-1Ut$_c$$XY?si = ttOu_o3KG55lucFx$.