Multiplicador 8bits

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

Este documento apresenta o projeto e a implementação de um multiplicador de 8 bits em Verilog. O multiplicador é um componente fundamental em sistemas digitais, sendo essencial para operações aritméticas em processadores e outras unidades lógicas. Este projeto visa demonstrar o funcionamento de um multiplicador sequencial baseado no algoritmo de ”shift-and-add”, amplamente utilizado devido à sua simplicidade e eficiência. Serão detalhadas as seções de funcionamento, a descrição em Verilog dos módulos envolvidos (multiplicador e multiplexadores auxiliares) e a bancada de teste para validação do seu comportamento.

2. Funcionamento do Multiplicador

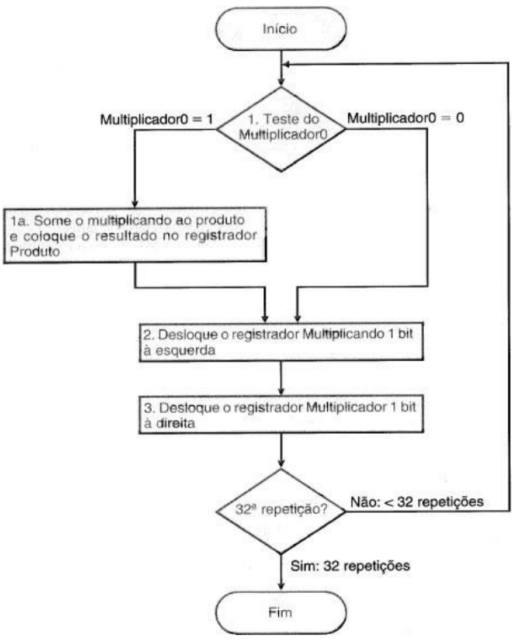

Figura 1. Diagrama do funcionamento do multiplicador.

O multiplicador de 8 bits implementado neste projeto utiliza o algoritmo de ”shift-and-add”(deslocamento e soma). Este algoritmo funciona da seguinte forma:

1. Inicialmente, o **produto** é zerado, o **multiplicando** é carregado em um registrador de 16 bits (B), e o **multiplicador** é carregado em um registrador de 8 bits (A). Um contador (count) é inicializado com o número de bits do multiplicador (8, neste caso).

2. Em cada ciclo de clock, o bit menos significativo do registrador **A** (multiplicador) é verificado.
    - Se o bit for '1', o valor do registrador **B** (multiplicando deslocado) é adicionado ao **produto**.
    - Se o bit for '0', nenhuma soma é realizada.
3. Em seguida, o registrador **B** é deslocado para a esquerda em uma posição, e o registrador **A** é deslocado para a direita em uma posição.
4. O contador é decrementado.
5. Este processo se repete por 8 ciclos (ou até o contador chegar a zero).
6. Ao final dos 8 ciclos, o registrador **produto** conterá o resultado da multiplicação. Um sinal de **fim** é ativado para indicar a conclusão da operação.

Este método simula a multiplicação manual, onde a soma de parcelas parciais (multiplicando vezes cada bit do multiplicador, com os devidos deslocamentos) resulta no produto final.

3. Descrição em verilog

A implementação do multiplicador é dividida em três módulos Verilog: `multi_8b`, `mux8x1_8b` e `mux8x1_16b`.

3.1. Módulo `multi_8b`

Este é o módulo principal do multiplicador.

```
1   module multi_8b(
2     input clk,
3     input rst,
4     input inicio,
5     input [15:0] multiplicando,
6     input [7:0] multiplicador,
7     output reg [15:0] produto,
8     output reg fim
9   );
10
11    reg [7:0] A;
12    reg [15:0] B;
13    reg [3:0] count;
14    wire [15:0] w_soma, w_shiftB;
15    wire [7:0] w_shiftA;
16
17    always @(posedge clk or posedge rst) begin
18      if (rst) begin
19        produto <= 0;
20        A <= 0;
21        B <= 0;
22        count <= 0;
23        fim <= 0;
24      end else if (inicio) begin
25        A <= multiplicador;
26        B <= multiplicando;
27        count <= 8;
28        produto <= 0;
29        fim <= 0;
```

```
        end else if (count > 0) begin
          if (A[0] == 1) begin
            produto <= w_soma;
          end
          B <= w_shiftB;
          A <= w_shiftA;
          count <= count - 1;
          if (count == 1)
            fim <= 1;
        end
      end

      mux8x1_16b mux_soma (.a(produto), .b(B), .x(1'b0), .y(1'b0), .z
          (1'b0), .s(w_soma));
      mux8x1_16b mux_shiftB (.a(B), .b(16'b1), .x(1'b0), .y(1'b1), .z
          (1'b0), .s(w_shiftB));
      mux8x1_8b mux_shiftA (.a(A), .b(8'b1), .x(1'b0), .y(1'b1), .z(1'
          b1), .s(w_shiftA));

    endmodule
```

3.2. Módulo `mux8x1_8b`

Este módulo implementa um multiplexador 8x1 para operar com dados de 8 bits, realizando diferentes operações aritméticas e lógicas com base nas entradas de seleção `x`, `y`, `z` [Pedro Souza 2020].

```
module mux8x1_8b (
        input [7:0] a, b,
        input x, y, z,
        output reg [7:0] s
);

        always@*
        begin
                case ({x, y, z})
                        3'b000: s = a + b;
                        3'b001: s = a - b;
                        3'b010: s = a << b;
                        3'b011: s = a >> b;
                        3'b100: s = a & b;
                        3'b101: s = a | b;
                        3'b110: s = a ^ b;
                        3'b111: s = ~a;
                endcase
        end

endmodule
```

3.3. Módulo `mux8x1_16b`

Similar ao `mux8x1_8b`, mas projetado para operar com dados de 16 bits.

```
module mux8x1_16b (
        input [15:0] a, b,
```

```
        input x, y, z,
        output reg [15:0] s
);

        always@*
        begin
                case ({x, y, z})
                        3'b000: s = a + b;
                        3'b001: s = a - b;
                        3'b010: s = a << b;
                        3'b011: s = a >> b;
                        3'b100: s = a & b;
                        3'b101: s = a | b;
                        3'b110: s = a ^ b;
                        3'b111: s = ~a;
                endcase
        end

endmodule
```

4. Teste do Multiplicador

Para verificar o correto funcionamento do multiplicador, uma bancada de teste (`tb_multi_8b`) foi desenvolvida em Verilog. Esta bancada de teste simula diferentes cenários de entrada e verifica se a saída (`produto`) corresponde ao valor esperado.

A bancada de teste inclui:

1. Um **clock** com período de 10 ns.
2. Um **reset** inicial para garantir que o sistema comece em um estado conhecido.
3. Três casos de teste com diferentes valores para `multiplicando` e `multiplicador`:
   - **Teste 1:** 25 x 12, esperando 300.
   - **Teste 2:** 10 x 12, esperando 120.
   - **Teste 3:** 2 x 3, esperando 6.
4. Para cada teste, a bancada aguarda o sinal de `fim` para validar o resultado, exibindo mensagens de sucesso ou erro.

A seguir, o código da bancada de teste:

Listing 1. Bancada de Teste do Multiplicador

```
module tb_multi_8b;
  reg clk;
  reg rst;
  reg inicio;
  reg [15:0] multiplicando;
  reg [7:0] multiplicador;
  wire [15:0] produto;
  wire fim;

  multi_8b uut (
    .clk(clk),
    .rst(rst),
    .inicio(inicio),
```

```
14      .multiplicando(multiplicando),
15      .multiplicador(multiplicador),
16      .produto(produto),
17      .fim(fim)
18    );
19
20    // Clock de 10 ns
21    always #5 clk = ~clk;
22
23    initial begin
24      clk = 0;
25      rst = 1;
26      inicio = 0;
27      multiplicando = 0;
28      multiplicador = 0;
29      #20;
30      rst = 0;
31
32      // === Teste 1 ===
33      multiplicando = 16'd25;
34      multiplicador = 8'd12;
35      @(negedge clk); inicio = 1;
36      @(negedge clk); inicio = 0;
37
38      // Espera fim com timeout
39      repeat (100) @(posedge clk);
40      if (fim) begin
41        $display("Teste 1: 25 x 12 = %d", produto);
42        if (produto !== 16'd300)
43          $display("Erro: Esperado 300");
44        else
45          $display("Sucesso!");
46      end else begin
47        $display("Erro: timeout no Teste 1");
48      end
49
50      // Reset
51      rst = 1; @(negedge clk); rst = 0;
52
53      // === Teste 2 ===
54      multiplicando = 16'd10;
55      multiplicador = 8'd12;
56      @(negedge clk); inicio = 1;
57      @(negedge clk); inicio = 0;
58
59      repeat (100) @(posedge clk);
60      if (fim) begin
61        $display("Teste 2: 10 x 12 = %d", produto);
62        if (produto !== 16'd120)
63          $display("Erro: Esperado 120");
64        else
65          $display("Sucesso!");
66      end else begin
67        $display("Erro: timeout no Teste 2");
68      end
69
```

```
70          // Reset
71      rst = 1; @(negedge clk); rst = 0;
72
73      // === Teste 3 ===
74      multiplicando = 16'd2;
75      multiplicador = 8'd3;
76      @(negedge clk); inicio = 1;
77      @(negedge clk); inicio = 0;
78
79      repeat (100) @(posedge clk);
80      if (fim) begin
81        $display("Teste 2: 2 x 3 = %d", produto);
82        if (produto !== 16'd6)
83          $display("Erro: Esperado 6");
84        else
85          $display("Sucesso!");
86      end else begin
87        $display("Erro: timeout no Teste 3");
88      end
89
90      #20;
91      $finish;
92    end
93  endmodule
```

5. Conclusão

Este trabalho apresentou o projeto e a implementação de um multiplicador de 8 bits utilizando a linguagem Verilog. O multiplicador foi desenvolvido com base no algoritmo de "shift-and-add", que se mostrou eficaz para a operação de multiplicação sequencial. A modularização do design, com a criação de módulos de multiplexadores dedicados para 8 e 16 bits, contribuiu para a clareza e reusabilidade do código. A bancada de teste demonstrou a funcionalidade correta do multiplicador para os casos testados, validando o comportamento esperado. Este projeto serve como uma base sólida para a compreensão e implementação de circuitos aritméticos digitais mais complexos.

Referências

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