Circuito Divisor em Verilog

Thiago Cordeiro de Melo[1]

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

tcdm.eng23@uea.edu.br

Abstract. *This report presents the design and simulation of an 8-bit digital divider implemented in Verilog Hardware Description Language (HDL). The module operates synchronously and performs binary division using a classical shift-subtract algorithm. The design includes division-by-zero detection, an operation completion flag, and outputs both the quotient and the remainder. To validate the design, a testbench was developed comprising five test cases, covering divisions with and without remainders, division by one, division by zero, and zero dividend. Simulation results verified the correct behavior of the divider, including proper signaling through the* `fim` *(done) and* `zero_div` *flags. The project is suitable for future integration into embedded processors and arithmetic logic units (ALUs).*

Resumo. *Este relatório apresenta o desenvolvimento e a simulação de um divisor digital de 8 bits utilizando a linguagem de descrição de hardware Verilog. O módulo foi projetado para operar de forma síncrona, realizando divisões sucessivas por meio de deslocamentos e subtrações, seguindo o método clássico de divisão binária. O sistema possui detecção de divisão por zero, sinalização de término de operação e fornece como saída o quociente e o resto da divisão. Para validação do projeto, foi elaborado um testbench contendo cinco casos de teste, abrangendo divisões com e sem resto, divisão por 1, por zero e com dividendo nulo. Os testes confirmaram o correto funcionamento do módulo, incluindo o acionamento adequado das flags* `fim` *e* `zero_div`*. O projeto pode ser expandido para aplicações em processadores embarcados e unidades aritméticas digitais.*

1. Introdução

Operações aritméticas básicas como adição, subtração, multiplicação e divisão são fundamentais para o funcionamento de sistemas digitais, especialmente em unidades de processamento aritmético (ALUs) e processadores embarcados. Dentre essas operações, a divisão se destaca por ser computacionalmente mais complexa, exigindo algoritmos e estruturas de controle mais elaboradas.

Neste contexto, este projeto tem como objetivo o desenvolvimento de um módulo divisor de 8 bits utilizando a linguagem de descrição de hardware Verilog. O divisor implementa uma arquitetura sequencial baseada no algoritmo de subtração sucessiva com deslocamentos, também conhecido como método clássico de divisão binária. Este método é amplamente utilizado em circuitos digitais por sua simplicidade e adequação a implementações síncronas.

Além de fornecer o quociente e o resto da operação, o módulo também é capaz de sinalizar quando a divisão foi concluída por meio da flag `fim`, e detectar tentativas de divisão por zero, ativando a flag `zero_div`, o que garante maior robustez e controle em aplicações práticas.

A implementação foi acompanhada de um conjunto de testes automatizados via *testbench*, cobrindo diferentes cenários e validando o comportamento do sistema em condições normais e de exceção. A simulação dos testes confirmou a funcionalidade correta do divisor, tornando-o apto para ser integrado a sistemas maiores, como processadores customizados ou controladores digitais.

2. Funcionamento do Divisor

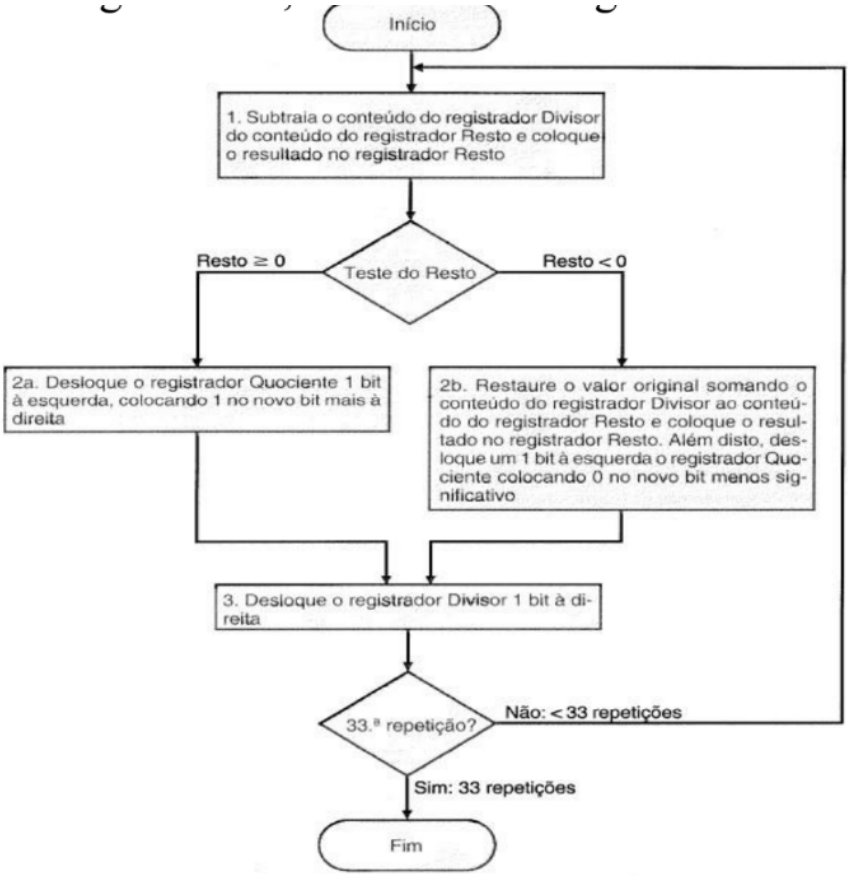

Figura 1. Diagrama do fluxo do Divisor

O algoritmo opera da seguinte forma:

- **Passo 1**: Subtrai-se o divisor do resto atual
- **Avaliação do Resto**:
  - Se `Resto < 0` (bit de sinal ativo):
    - Executa o caminho 2a (operação de restauração)
    - Desloca o quociente inserindo 0
  - Caso contrário:
    - Desloca o quociente inserindo 1
- **Passo 3**: Deslocamento do divisor para direita
- **Controle de Iteração**: Repete por 33 ciclos (para números de 32 bits + sinal)

3. Implementação em Verilog

O módulo `Div_8b` implementa um divisor de 8 bits.

Listing 1. Módulo Divisor 8 Bits

```
module Div_8b (
    input clk,
    input reset,
    input start,
    input [7:0] div1,
    input [7:0] div2,
    output reg [7:0] quo,
    output reg [7:0] resto,
    output reg fim,
    output reg zero_div
);
    reg [3:0] count;
    reg [15:0] rem_reg;
    reg [7:0] divisor;
    reg [2:0] state;
    localparam IDLE  = 3'b000,
               CHECK = 3'b001,
               LOAD  = 3'b010,
               DIV   = 3'b011,
               DONE  = 3'b100;
    always @(posedge clk) begin
        if (reset) begin
            quo <= 0;
            resto <= 0;
            fim <= 0;
            zero_div <= 0;
            state <= IDLE;
        end else begin
            case (state)
                IDLE: begin
                    fim <= 0;
                    if (start) state <= CHECK;
                end

                CHECK: begin
                    if (div2 == 0) begin
                        zero_div <= 1;
                        state <= DONE;
                    end else begin
                        zero_div <= 0;
                        state <= LOAD;
                    end
```

```
43                  end
44
45                  LOAD: begin
46                      rem_reg <= {8'b0, div1};
47                      divisor <= div2;
48                      quo <= 0;
49                      count <= 8;
50                      state <= DIV;
51                  end
52
53                  DIV: begin
54                      rem_reg = rem_reg << 1;
55                      rem_reg[0] = quo[7];
56
57                      rem_reg[15:8] = rem_reg[15:8] - divisor;
58
59                      if (rem_reg[15]) begin
60                          rem_reg[15:8] = rem_reg[15:8] + divisor;
61                          quo = quo << 1;
62                      end else begin
63                          quo = (quo << 1) | 1'b1;
64                      end
65
66                      count = count - 1;
67                      if (count == 0) state <= DONE;
68                  end
69
70                  DONE: begin
71                      resto <= rem_reg[15:8];
72                      fim <= 1;
73                      state <= IDLE;
74                  end
75              endcase
76          end
77      end
78  endmodule
```

- **Entradas:**
  - `clk`: Sinal de clock. Responsável por sincronizar as transições de estado e execuções do algoritmo de divisão.
  - `reset`: Sinal de reset. Reinicializa todos os registradores internos e flags para o estado inicial (IDLE).
  - `start`: Sinal de controle. Inicia o processo de divisão quando ativado.
  - `div1 [7:0]`: Dividendo da operação de divisão, com 8 bits.
  - `div2 [7:0]`: Divisor da operação de divisão, com 8 bits.
- **Saídas:**
  - `quo [7:0]`: Quociente da divisão. Representa o resultado da parte inteira da divisão.

  - `resto [7:0]`: Resto da divisão. Contém o valor que sobra após a operação.
  - `fim`: Flag que indica o término da operação de divisão.
  - `zero_div`: Flag que sinaliza uma tentativa de divisão por zero.
- **Flags de Status:**
  - **zero_div:**
    - Ativada quando o divisor `div2` é igual a zero.
    - Impede a realização da divisão e pula diretamente para o estado `DONE`.
    - Essencial para evitar erros aritméticos ou travamentos do sistema.
  - **fim:**
    - Ativada quando a divisão foi concluída com sucesso.
    - Permite que o sistema externo saiba quando o quociente e o resto estão prontos para leitura.
- **Registradores e Variáveis Internas:**
  - `count [3:0]`: Contador de ciclos. Controla o número de iterações (até 8) da operação de divisão.
  - `rem_reg [15:0]`: Registrador de 16 bits que armazena o conteúdo temporário do dividendo (resto parcial).
  - `divisor [7:0]`: Armazena o valor do divisor para uso durante toda a operação.
  - `state [2:0]`: Registrador de estado. Controla a máquina de estados finitos (FSM) com cinco possíveis estados:
    - `IDLE`: Estado ocioso, aguardando o sinal `start`.
    - `CHECK`: Verifica se o divisor é zero.
    - `LOAD`: Carrega os registradores internos com os operandos.
    - `DIV`: Executa o algoritmo de divisão com deslocamentos e subtrações sucessivas.
    - `DONE`: Finaliza a operação e define as saídas.

4. Testes do Divisor

Listing 2. Módulo para o test bench do divisor

```
1  `timescale 1ns/1ps
2
3  module tb_Div_8b();
4
5      // Entradas
6      reg clk;
7      reg reset;
8      reg start;
9      reg [7:0] div1;
10     reg [7:0] div2;
11
12     // Saídas
13     wire [7:0] quo;
14     wire [7:0] resto;
15     wire fim;
```

```
16      wire zero_div;
17
18      // Instanciação do módulo
19      Div_8b uut (
20          .clk(clk),
21          .reset(reset),
22          .start(start),
23          .div1(div1),
24          .div2(div2),
25          .quo(quo),
26          .resto(resto),
27          .fim(fim),
28          .zero_div(zero_div)
29      );
30
31      // Clock gerado com período de 10ns
32      always #5 clk = ~clk;
33
34      initial begin
35          // Inicialização
36          clk = 0;
37          reset = 1;
38          start = 0;
39          div1 = 0;
40          div2 = 0;
41          #20;
42
43          reset = 0;
44
45          // Teste 1: 100 / 5
46          div1 = 8'd100;
47          div2 = 8'd5;
48          start = 1;
49          #10 start = 0;
50          wait(fim);
51          #10;
52          $display("Teste 1: 100 / 5 => quo = %d, resto = %d,
                zero_div = %b", quo, resto, zero_div);
53
54          // Teste 2: 15 / 4
55          div1 = 8'd15;
56          div2 = 8'd4;
57          start = 1;
58          #10 start = 0;
59          wait(fim);
60          #10;
61          $display("Teste 2: 15 / 4 => quo = %d, resto = %d,
                zero_div = %b", quo, resto, zero_div);
62
```

```
63          // Teste 3: 200 / 1
64          div1 = 8'd200;
65          div2 = 8'd1;
66          start = 1;
67          #10 start = 0;
68          wait(fim);
69          #10;
70          $display("Teste 3: 200 / 1 => quo = %d, resto = %d,
                zero_div = %b", quo, resto, zero_div);
71
72          // Teste 4: divisão por zero
73          div1 = 8'd50;
74          div2 = 8'd0;
75          start = 1;
76          #10 start = 0;
77          wait(fim);
78          #10;
79          $display("Teste 4: 50 / 0 => quo = %d, resto = %d,
                zero_div = %b", quo, resto, zero_div);
80
81          // Teste 5: dividendo zero
82          div1 = 8'd0;
83          div2 = 8'd10;
84          start = 1;
85          #10 start = 0;
86          wait(fim);
87          #10;
88          $display("Teste 5: 0 / 10 => quo = %d, resto = %d,
                zero_div = %b", quo, resto, zero_div);
89
90          // Finaliza simulação
91          $display("Todos os testes foram executados.");
92          $stop;
93      end
94
95  endmodule
```

Explicação dos Testes no Testbench do Módulo `Div_8b`

- **Teste 1:** Divisão de $100 \div 5$
  - Entradas: $div1 = 100$, $div2 = 5$
  - Operação: $100 \div 5$
  - Resultado esperado: $quo = 20$, $resto = 0$, $zero\_div = 0$
  - Este teste verifica uma divisão exata, sem resto, entre dois números inteiros positivos.

- **Teste 2:** Divisão de $15 \div 4$
  - Entradas: $div1 = 15$, $div2 = 4$
  - Operação: $15 \div 4$

  - Resultado esperado: $quo = 3$, $resto = 3$, $zero\_div = 0$
  - Este teste avalia o comportamento do divisor em uma operação com resto diferente de zero.

- **Teste 3:** Divisão de $200 \div 1$
  - Entradas: $div1 = 200$, $div2 = 1$
  - Operação: $200 \div 1$
  - Resultado esperado: $quo = 200$, $resto = 0$, $zero\_div = 0$
  - Este teste avalia o caso onde o divisor é 1, esperando que o quociente seja o próprio dividendo.

- **Teste 4:** Divisão de $50 \div 0$
  - Entradas: $div1 = 50$, $div2 = 0$
  - Operação: divisão por zero
  - Resultado esperado: $zero\_div = 1$, quo e $resto$ podem ser ignorados
  - Este teste verifica se o módulo trata corretamente a divisão por zero, ativando o sinalizador `zero_div`.

- **Teste 5:** Divisão de $0 \div 10$
  - Entradas: $div1 = 0$, $div2 = 10$
  - Operação: $0 \div 10$
  - Resultado esperado: $quo = 0$, $resto = 0$, $zero\_div = 0$
  - Este teste verifica o comportamento do módulo ao dividir zero por um número diferente de zero.

Em todos os testes, o sinal `start` é ativado por 10 ns para iniciar a operação de divisão, e a simulação aguarda o término indicado pelo sinal `fim` antes de iniciar o próximo teste. Ao final, a simulação é finalizada com a mensagem “Todos os testes foram executados.”.

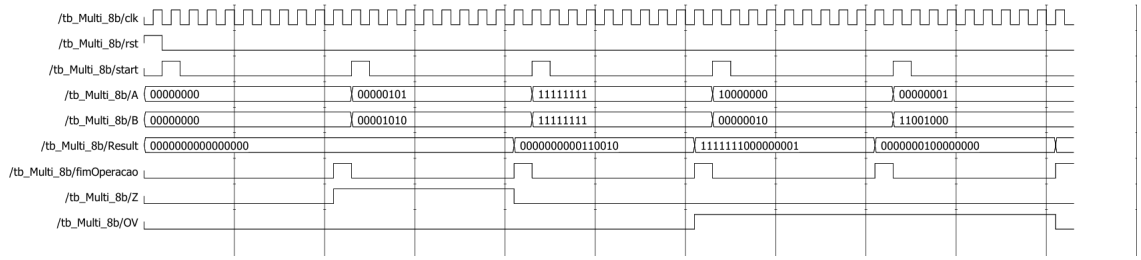

Figura 2. Verificação de cada um dos testes no ModelSim

5. Conclusão

Os testes realizados com o testbench `tb_Div_8b` demonstraram que o módulo divisor de 8 bits `Div_8b` está operando corretamente em diversas situações de interesse. Foram avaliados casos de divisão exata, divisão com resto, divisão por um, divisão com dividendo nulo e tentativa de divisão por zero.

Cada cenário foi escolhido com o objetivo de validar aspectos essenciais do funcionamento do divisor, como o cálculo correto do quociente e do resto, bem como a

detecção segura de erros como a divisão por zero. O comportamento observado nas simulações correspondeu ao esperado, indicando que o módulo implementa adequadamente a operação de divisão inteira para operandos de 8 bits.

Com base nesses resultados, conclui-se que o módulo está apto para ser utilizado em sistemas digitais que necessitem de operações aritméticas de divisão segura e precisa, respeitando as condições de controle, sinalização e finalização da operação.

Por fim, recomenda-se a expansão dos testes para incluir casos com diferentes larguras de dados, sinais (caso a arquitetura suporte números com sinal), e análise de desempenho em diferentes frequências de clock para avaliar robustez temporal e consumo de recursos em aplicações embarcadas.

Referências

[Souza] Souza, P. Unidade lógico-aritmética usando a abordagem comportamental. https://www.youtube.com/watch?v=Ynymty6-5dMt=6s.