Divisor 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 **divisor de 8 bits** utilizando a linguagem de descrição de hardware **Verilog**. O divisor foi projetado para operar de forma síncrona, recebendo dois números de 8 bits, um dividendo (A) e um divisor (B), e fornecendo como saída o quociente e o resto da operação. A arquitetura se baseia no algoritmo de divisão por subtrações e deslocamentos sucessivos, comumente empregado em sistemas digitais. O objetivo principal é demonstrar a funcionalidade e a corretude do módulo de divisão através de simulações e testes, garantindo sua robustez para futuras integrações em projetos de maior complexidade.

2. Funcionamento do divisor

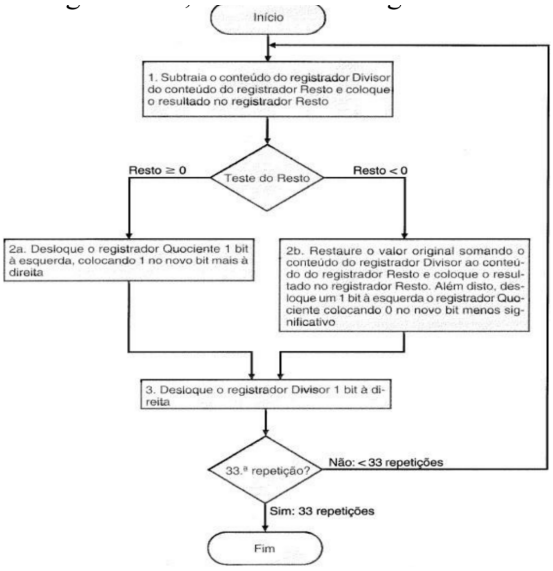

Figura 1. Diagrama de implementação do divisor.

O módulo ‘div_8b‘ implementa um divisor não restaurador, otimizado para divisão de números de 8 bits baseado em partes no diagrama da figura 1. A operação é controlada por um relógio (‘clk‘), um sinal de reset (‘rst‘) e um sinal de início (‘inicio‘).

O algoritmo funciona da seguinte forma:

1. **Inicialização**: Ao detectar um reset (‘rst‘), todos os registradores (resto, divisor, quociente, count, fim, busy) são zerados para garantir um estado inicial conhecido.
2. **Início da Divisão**: Quando o sinal ‘inicio‘ é ativado e o módulo não está ocupado (‘!busy‘), o processo de divisão é iniciado. O dividendo (A) é carregado na parte

Iteração	Passo	Quociente	Divisor	Resto
0	Valores iniciais	0000	0010 0000	0000 0111
1	1: Resto = Rem – Div	0000	0010 0000	**1110 0111**
	2b: Resto < 0 ⇒ +Div, sll Q, Q0 = 0	**0000**	0010 0000	**0000 0111**
	3: Deslocamento do Divisor 1 bit à direita	0000	**0001 0000**	0000 0111
2	1: Resto = Resto – Div	0000	0001 0000	**1111 0111**
	2b: Resto < 0 ⇒ +Div, sll Q, Q0 = 0	**0000**	0001 0000	**0000 0111**
	3: Deslocamento do Divisor 1 bit à direita	0000	**0000 1000**	0000 0111
3	1: Resto = Resto – Div	0000	0000 1000	**1111 1111**
	2b: Resto < 0 ⇒ +Div, sll Q, Q0 = 0	**0000**	0000 1000	**0000 0111**
	3: Deslocamento do Divisor 1 bit à direita	0000	**0000 0100**	0000 0111
4	1: Resto = Resto – Div	0000	0000 0100	**0000 0011**
	2a: Resto ≥ 0 ⇒ sll Q, Q0 = 1	**0001**	0000 0100	0000 0011
	3: Deslocamento do Divisor 1 bit à direita	0001	**0000 0010**	0000 0011
5	1: Resto = Resto – Div	0001	0000 0010	**0000 0001**
	2a: Resto ≥ 0 ⇒ sll Q, Q0 = 1	**0011**	0000 0010	0000 0001
	3: Deslocamento do Divisor 1 bit à direita	0011	**0000 0001**	0000 0001

Figura 2. Exemplo do processo de divisão.

menos significativa do registrador ‘resto‘ de 16 bits, e o divisor (B) é posicionado na parte mais significativa do registrador ‘divisor‘ de 16 bits (equivalente a $B \ll 8$). Um contador ‘count‘ é inicializado com o valor 8, indicando 8 iterações para a divisão de 8 bits. O sinal ‘busy‘ é ativado.

3. **Processo Iterativo**: Enquanto o módulo está ocupado (‘busy‘) e o contador ‘count‘ é maior que zero, o algoritmo realiza as seguintes etapas a cada ciclo de clock:
    - O registrador ‘resto‘ é deslocado para a esquerda em 1 bit (‘resto = resto $\ll$ 1‘).
    - É feita uma comparação entre a parte mais significativa do ‘resto‘ (os 8 bits mais à esquerda, ‘resto[15:8]‘) e o divisor (‘divisor $\gg$ 8‘, que é o valor original de B).
    - Se ‘resto[15:8]‘ for maior ou igual ao divisor, o divisor é subtraído de ‘resto[15:8]‘, e o bit menos significativo de ‘resto‘ (‘resto[0]‘) é definido como 1 (indicando que o bit do quociente é 1).
    - Caso contrário, ‘resto[0]‘ é definido como 0 (indicando que o bit do quociente é 0).
    - O contador ‘count‘ é decrementado.
4. **Finalização**: Quando ‘count‘ atinge 1 (indicando a última iteração), o resultado é finalizado. O quociente é armazenado nos 8 bits menos significativos de ‘resto‘ (‘resto[7:0]‘), e o resto final é obtido dos 8 bits mais significativos de ‘resto‘ (‘resto[15:8]‘). Os sinais ‘fim‘ e ‘busy‘ são ativados e desativados, respectivamente, para indicar a conclusão da operação.

Este método garante que a divisão seja realizada em um número fixo de ciclos de clock (8 ciclos para um divisor de 8 bits) após o sinal de ‘inicio‘.

3. Descrição em verilog

A seguir, é apresentada a descrição do módulo Verilog para o divisor de 8 bits (‘div_8b‘).

```
module div_8b (
        input clk, rst, inicio,
        input [7:0] A, B, //Entradas dividendo e divisor
        output reg [7:0] quociente,
        output reg fim,
        output reg [15:0] resto_out
);
```

```
8
9           reg [15:0] divisor;
10          reg [15:0] resto;
11          reg [3:0] count;
12          reg busy; //indica que a divis~ao esta em andamento
13
14          always @(posedge clk or posedge rst) begin
15                  if (rst) begin
16                          resto <= 0;
17                          divisor <= 0;
18                          quociente <= 0;
19                          resto_out <= 0;
20                          count <= 0;
21                          fim <= 0;
22                          busy <= 0;
23                  end else if (inicio && !busy) begin
24                          resto <= {8'd0, A};
25                          divisor <= {B, 8'd0};
26                          count <= 8;
27                          fim <= 0;
28                          busy <= 1;
29
30                  end else if (busy && count > 0) begin
31                          resto = resto << 1;
32
33                          if (resto[15:8] >= (divisor >> 8)) begin
34                                  resto[15:8] = resto[15:8] - (divisor >>
                                        8);
35                                  resto[0] = 1'b1;
36                          end else begin
37                                  resto[0] = 1'b0;
38                          end
39
40                          count <= count - 1;
41
42                          if (count == 1) begin
43                                  quociente <= resto[7:0];
44                                  resto_out <= resto[15:8]; // agora sim,
                                        valor final e atualizado
45                                  fim <= 1;
46                                  busy <= 0;
47                          end
48                  end
49          end
50
51  endmodule
```

4. Testes

Para verificar a funcionalidade e corretude do módulo 'div_8b', foi desenvolvido um *testbench* em Verilog. O *testbench* simula cenários de entrada e verifica as saídas do divisor, comparando-as com os resultados esperados. O clock foi configurado para 100 MHz (período de 10 ns).

O 'tb_div_8b' realiza os seguintes testes:

1. **Teste 1:** $7 \div 2$
    - Entradas: $A = 7$, $B = 2$
    - Esperado: Quociente = 3, Resto = 1
2. **Teste 2:** $200 \div 10$
    - Entradas: $A = 200$, $B = 10$
    - Esperado: Quociente = 20, Resto = 0
3. **Teste 3:** $32 \div 7$
    - Entradas: $A = 32$, $B = 7$
    - Esperado: Quociente = 4, Resto = 4

Cada teste inclui um reset inicial para garantir que o módulo esteja em um estado conhecido antes da execução da divisão. Os resultados são exibidos no console de simulação, indicando se o teste foi bem-sucedido ou falhou.

```
`timescale 1ns / 1ps

module tb_div_8b;

    // Entradas
    reg clk;
    reg rst;
    reg inicio;
    reg [7:0] A, B;

    // Saidas
    wire [7:0] quociente;
    wire fim;
    wire [15:0] resto_out;

    // Instancia do modulo
    div_8b uut (
        .clk(clk),
        .rst(rst),
        .inicio(inicio),
        .A(A),
        .B(B),
        .quociente(quociente),
        .fim(fim),
        .resto_out(resto_out)
    );

    // Clock: 100 MHz
    always #5 clk = ~clk;

    initial begin
        $display("Inicio dos testes");
        clk = 0;
        rst = 1;
        inicio = 0;
        A = 0;
        B = 0;
        #20 rst = 0;

        // Teste 1: 7 / 2 = 3 resto 1
        A = 8'd7;
```

```
42          B = 8'd2;
43          inicio = 1;
44          #10 inicio = 0;
45
46          wait(fim);
47          #10;
48          if (quociente == 8'd3 && resto_out == 16'd1)
49              $display("Teste 1 OK: 7 / 2 = %d resto %d", quociente,
                    resto_out);
50          else
51              $display("Teste 1 FALHOU: esperado 3 / 1, obtido %d / %d",
                    quociente, resto_out);
52
53          // Teste 2: 200 / 10 = 20 resto 0
54          rst = 1; #10; rst = 0;
55          A = 8'd200;
56          B = 8'd10;
57          inicio = 1;
58          #10 inicio = 0;
59
60          wait(fim);
61          #10;
62          if (quociente == 8'd20 && resto_out== 16'd0)
63              $display("Teste 2 OK: 200 / 10 = %d resto %d", quociente,
                    resto_out);
64          else
65              $display("Teste 2 FALHOU: esperado 20 / 0, obtido %d / %d",
                     quociente, resto_out);
66
67          // Teste 3: 32 / 7 = 4 resto 4
68          rst = 1; #10; rst = 0;
69          A = 8'd32;
70          B = 8'd7;
71          inicio = 1;
72          #10 inicio = 0;
73
74          wait(fim);
75          #10;
76          if (quociente == 8'd4 && resto_out == 16'd4)
77              $display("Teste 3 OK: 32 / 7 = %d resto %d", quociente,
                    resto_out);
78          else
79              $display("Teste 3 FALHOU: esperado 4 / 4, obtido %d / %d",
                    quociente, resto_out);
80
81          $display("Fim dos testes");
82
83          $finish;
84      end
85
86  endmodule
```

5. Conclusão

Este trabalho detalhou o projeto e a implementação de um **divisor de 8 bits** em Verilog, utilizando um algoritmo de divisão por subtrações e deslocamentos. O módulo ‘div_8b‘ demonstrou capacidade de realizar divisões inteiras de forma eficiente e síncrona. Os testes realizados com o *testbench* validaram a funcionalidade do divisor para diferentes cenários de entrada, confirmando a correta obtenção do quociente e do resto. A modularidade e a clareza da descrição em Verilog permitem que este divisor seja facilmente integrado em sistemas digitais maiores, servindo como um componente fundamental em diversas aplicações. Futuros trabalhos podem incluir a otimização do consumo de recursos, a implementação de divisores com maior largura de bits ou a exploração de outros algoritmos de divisão.