Implementação do Divisor de 16 bits

Orleoncio Maciel de Oliveira Filho
Curso de ciências da computação
IFCE - Maracanaú
Maracanaú, Ceará
orleoncio.maciel.oliveira07@aluno.ifce.edu.br

Yan Cavalcante Câmara
Curso de ciências da computação
IFCE - Maracanaú
Maracanaú, Ceará
yan.cavalcante09@aluno.ifce.edu.br

Juliano Magalhães Jurity
Curso de ciências da computação
IFCE - Maracanaú
Maracanaú, Ceará
juliano.magalhaes06@aluno.ifce.edu.br

***Resumo*—Relatório do projeto de implementação de um circuito divisor de 16 bits com maquina de estado em SystemVerilog, além de testes sendo elespor meio de testbench e esquema de ondas, o documento descreve cada passo para a criação de tal circuito dês das pesquisas feitas para entender mais sobre o mesmo até a finalização de sua implementação.**

I. Introdução

O objetivo deste projeto estava em elaborar, testar e comprovar a funcionalidade de um circuito divisor de 16 bits que utiliza a tecnica FMDS (Finite State Machine with Data Path). Essa elaboração se dava por meio do código em SystemVerilog e tambem do diagrama ASM.

A divisão da nossa equipe foi feita de modo no qual nós buscamos resolver o problema o mais rápido possível. No início, todos pesquisaram sobre como seria possivel implementar tal circuito, tambem pesquisamos como seria posivel implementar uma maquina de estados para controlar a divisão feita e como seria a melhor forma de gerar um diagrama ASM. Toda essa pesquisa foi feita para que ambos os 3 membros da equipe estivessem a par do que o trabalho trata.

Após isso foi dado inicio a etapa de elaboração do projeto onde iniciamos pela elaboração do diagrama ASM assim foi posivel colocar as ideias sobre como implementar esse circuito na mesa e analisar com mais cautela o passo a passo a ser tomado.

Mais à frente neste documento, iremos falar sobre a metodologia passo a passo utilizada para chegar nos resultados encontrados. Também iremos apresentar tais resultados e se eles foram ou não compatíveis com o esperado, e por fim, teremos uma conclusão com as considerações finais do projeto.

II. Metodologia

A. Pesquisa

Para iniciar a elaboração do projeto do circuito divisor, foi-se necessário realizar várias pesquisas, tanto para se entender como poderiamos aplicar a formula da divisão feita no papel em circuitos, quanto para saber como elaborar o diagrama ASM e transformalo posteriormente em um código em SystemVerilog.

Nas pesquisas realizadas, foi visto que uma das formas de implementar um circuito divisor seria pelo metodo das subtrações sucecivas, este metodo consiste em subtrair o valor do divisor do dividendo ate que o dividendo seja menor que o divisor, desta maneira quando isso acontecer o valor que sobrar do dividendo sera o resto e a quantidade de vezes que a subtração ocorrer sera o quociente.

B. Elaboração do diagrama ASMD

Após a etapa de pesquisa, a equipe já estava ciente sobre o que se tratava um diagrama ASM, e de como demostrar o Data Path então, foi iniciada a etapa de elaboração do mesmo.

Para a realização do diagrama ASM foi-se ultilizado o software AutoCad de desenho técnico, esse software foi escolhido simplesmente por gosto pessoal de quem elaborou o diagrama, ja o DataPath foi ultilizado o logiSim ja que se tratava de uma logica de datos bem simples.

C. Elaboração do Código

O próximo passo após concluir a etapa de elaboração do diagrama ASM foi a criação do código. Como sujerido na documentação o código foi escrito baseado no diagrama ASM elaborado anteriormente.

Como a logica ultilizada no diagrama ASM para a realização da divisão foi a logica das subtrações sucessivas esta mesma logica foi ultilizada na elaboração do código em SystemVerilog.

D. Testes do circuito

Tendo prontos o diagrama ASMD e o código em SystemVerilog foi possível iniciar os testes do divisor de 16 bits.

Para a realização dos testes foi elaborado um código de testbench que receberia uma serie de valores diferentes em sua entrada e retornaria os valores de saida por meio de um esquema de ondas no software gtkwave.

Como ambas as tentradas do codigo (dividendo e divisor) são de 16 bits eram muita as posibilidades de entradas que poderiam ser adicionadas no testbench, por isso com ajuda de uma inteligencia artificial foram escolhidos uma serie de valores especificos onde fosse posivel comprovar a eficacia do circuito com um menor numero de posibilidades.

E. Elaboração da documentação

Logo tendo o código pronto e testado foi dado início a elaboração da documentação sendo está o relatório (documento em questão).

III. Resultados

A. Diagrama ASMD

Presente na figura 1.1 esta o resultado da elaboração do diagrama ASM e na figura 1.2 esta o resultado do Data Path do circuito de divisão de 16 bits. Como dito anteriormente a lógica usada para a elaboração do diagrama ASM foi a logica das divisões multiplas.

É possivel reparar que no primero estado da nossa maquina que foi chamado de idle a primeira operação é fazer com que a variavel ready receba 1 para informar que a execução do código vai começar. Após isso existem duas condicionais que caso sejam falsas o circuito retorna para o inicio do idle, essas consicionais analisam se o nosso start é igual a 1 (caso não seja o circuito não deve iniciar) e se nosso divisor é diferente de 0 (pois não existe divisão por 0), caso ambas as condicionais sejam verdadeiras sera realizada uma operação com os registradores onde o registrador acumulador vai receber o dividendo e o registrador

quociente recebe 0.

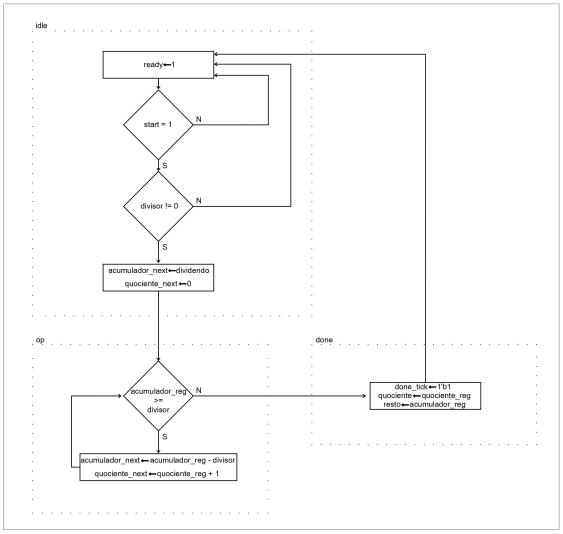

Figura 1 – Diagrama ASM

Apos isso no segundo estado do nosso diagrama o estado op existe outra condicional que analisa se o valor de saida do registrador acumulador é maior ou igual ao valor do divisor, caso ele seja o registrador acumulador recebe ele mesmo menos o divisor e o registrador quociente recebe ele mesmo mais 1 ja se essa condicionar retornar um valor falso é iniciado o proximo estado chamado de done.

No terceiro e ultimo estado chamdo done, não existem mais condicionais ele simplesmente atribui os valores dos registradores nas saidas, logo a saida done_tick vai receber 1, a saida quociente vai receber o valor do registrador quociente e o resto vai receber o valor do registrador acumulador.

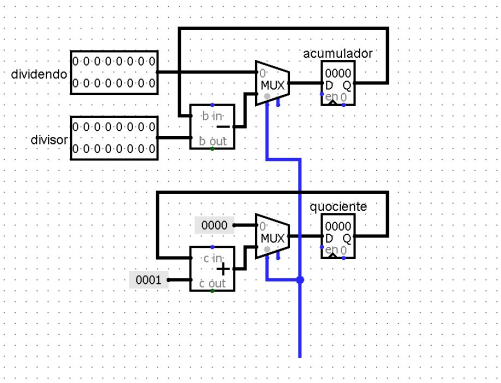

Figura 2 – Data Path

Como visto na imagem 1.2 esta o diagrama referente ao Data Path do código, o Data Path nada mais é que a logica por tras das operações aritimeticas da nossa maquina de estados.

Neste Data Path nos temos as duas entradas de 16 bits referentes ao dividendo e o divisor da nossa divisão e a funcionalidade dele é muito simples, caso a nossa maquina de estados retorne o estado idle o valor 0 sera adicionado ao controlador do nosso mux isso fara com que o registrador acumulador receba apenas i valor do dividendo e o registrador quociente receba apenas 0.

No estado seginte (op) o controlador do mux vai receber o valor 1 e por conta disso o registrador acumulador vai receber o valor dele mesmo menos o valor do divisor e o registrador quociente vai receber o valor dele mesmo mais 1 assim como esta presente no diagrama ASM.

B. Código SystemVerilog

Na figura 2 esta os resultados da elaboração do nosso código de divisão de numeros de 16 bits. Para a elaboração deste codigo usamos a diagrama presente na figura 1 para usar como base gerando assim o código, além tambem do código fibonacci apresentado em sala.

A funcionalidade é muito simple ja que o metodo usado para a implementação da divisão foi o da subtração sucesiva, basicamente assim como é possivel ver no código da figura 1 o código se inicia na declaração de variaveis seguindo como base o código fibonacci dado em aula, apois isso os estados idle, op e done são criados assim como os registradores acumulador e quociente.

Apos isso a lógica se torna simples pois o que o codigo faz é atribuir o valor do dividendo no registrador acumulador assim a cada ciclo de clock o valor do registrador é comparado com o valor do dividendo se o valor do registrador for maior ou igual ao valor do divisor no proximo ciclo ele recebera um decrementro do valor do divisor e o quociente vai receber um incremento de 1 em seu registrador.

O código continua realizando esse processo repetidamente até o registrador acumulador ter o valor menor que o do divisor, quando esse momento chegar o valor do registrador acumulador sera o resto da divisão e o valor do registrador quociente vai ser o quociente da divisão. Como cada subtração é feita a cada ciclo de clock algumas divisões vão levar mais tempo que outras por exemplo 10 dividido por 5 leva apenas 2 ciclos de clock ja que o quociente é 2 já 1000 dividido por 2 vai levar 500 ciclos de clock para chegar ao valor final.

Figura 2 – Código systemverilog

C. Testbench e Padrão de Ondas

Tambem foi-se realizado um testbench onde era-se analisado o comportamento das 4 entradas do código sendo elas o reset o start e as duas entradas de 16 bits (dividendo e divisor). O testbench ultiliza o iverilog para gerar um arquivo vcd que pode ser aberto pelo gtkwave e mostrar os padrões de onda do circuito baseado nos valores adicionados nas entradas.

Como dito anteriormente analizar todas as possiveis divisões entre dois numeros de 16 bits seria inviavel pois seriam muitas posibilidades por isso com a ajuda de uma IA conseguimos diminuir a quantidade de posibilidades para facilitar a visualização mas que mesmo assim fosse posivel comprovar a eficacia do código.

O resultado presente na figura 3 está de acordo com o que se era esperado do código já que o padrão de ondas reflete perfeitamente o resultado da divisão entre os numeros da entrada. No padrão de ondas é visto o mesmo comportamento analizado no código onde o quociente é incrementado de 1 em 1 ate chegar no valor correto e o resto é decrementado com o valor do divisor ate chegar a zero ou se tornar menor que ele e por esse motivo algumas divisões demoram mais ciclos de clock para serem realizadas que outras.

Vale resaltar tambem que ao analizar o padrão de onda foi visto que ao trocar o valor do divisor e do dividendo o circuito deveria realizar um breve reset pois caso o contratio a divisão entre os novos valores teria o resultado alterado pelo resultado dos valores anteriores.

D. Circuitos – LogiSim e DigitalJS

Para finalizar tudo o que foi produzido na elaboração do projeto, foram elaborados circuitos no logisim assim como no digitalJs sendo este último produzido a partir do código em systemverilog da figura 1, o circuito produzido no digitalJs está presente na figura 3 já o circuito produzido no logisim está presente na figura 4.

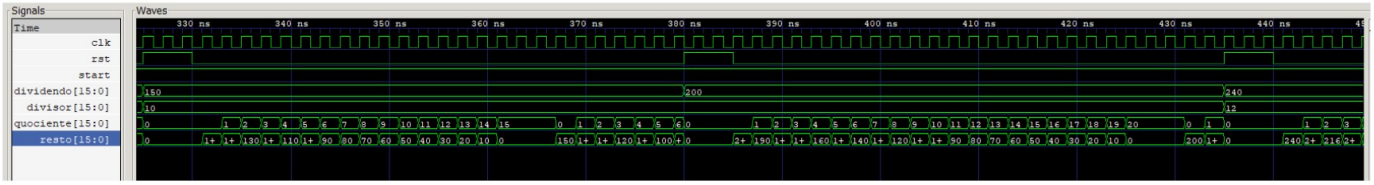

Figura 2 – Padrão de onda do TestBench

```
module divisor(
    input logic clk, rst, start,
    input logic[15:0] dividendo,
    input logic[15:0] divisor,
    output logic ready, done_tick,
    output logic[15:0] quociente,
    output logic [15:0] resto
);

    typedef enum {idle, op, done} state_type;
    state_type state_reg, state_next;

    logic [15:0] acumulador_reg, acumulador_next, quociente_reg, quociente_next;

    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            state_reg <= idle;
            acumulador_reg <= 0;
            quociente_reg <= 0;
        end else begin
            state_reg <= state_next;
            acumulador_reg <= acumulador_next;
            quociente_reg <= quociente_next;
        end
    end

    always_comb begin
        state_next = state_reg;
        ready = 1'b0;
        done_tick = 1'b0;
        acumulador_next = acumulador_reg;
        quociente_next = quociente_reg;
        resto = acumulador_reg;

        quociente = quociente_reg;
        resto = acumulador_reg;

        case (state_reg)
            idle: begin
                ready = 1'b1;
                if (start) begin
                    if (divisor != 0) begin
                        acumulador_next = dividendo;
                        quociente_next = 0;
                        state_next = op;
                    end else begin
                        state_next = idle;
                    end
                end
            end
            op: begin
                if (acumulador_reg >= divisor) begin
                    acumulador_next = acumulador_reg - divisor;
                    quociente_next = quociente_reg + 1;
                    state_next = op;
                end else begin
                    state_next = done;
                end
            end
            done: begin
                done_tick = 1'b1;
                quociente = quociente_reg;
                resto = acumulador_reg;
                state_next = idle;
            end
            default: begin
                state_next = idle;
            end
        endcase
    end
endmodule
```

```
`include "divisor.sv"
`timescale 1ns/100ps

module divisor_tb;
    logic clk, rst, start;
    logic[15:0] dividendo;
    logic[15:0] divisor;
    logic ready, done_tick;
    logic[15:0] quociente;
    logic [15:0] resto;

    divisor a1 (clk, rst, start, dividendo, divisor, ready, done_tick, quociente, resto);

    initial begin
        $dumpfile("divisor.vcd");
        $dumpvars(0, divisor_tb);
            clk = 1'b0; rst = 1'b1; start = 1'b0; dividendo = 16'd0; divisor = 16'd0;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd40; divisor = 16'd2;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd30; divisor = 16'd1;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd60; divisor = 16'd4;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd80; divisor = 16'd5;
        #5; rst = 1'b0; start = 1'b0; dividendo = 16'd30; divisor = 16'd2;
        #5; rst = 1'b0; start = 1'b0; dividendo = 16'd60; divisor = 16'd3;
        #5; rst = 1'b0; start = 1'b0; dividendo = 16'd70; divisor = 16'd4;
        #5; rst = 1'b0; start = 1'b0; dividendo = 16'd80; divisor = 16'd5;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd100; divisor = 16'd5;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd100; divisor = 16'd5;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd150; divisor = 16'd10;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd150; divisor = 16'd10;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd240; divisor = 16'd12;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd240; divisor = 16'd12;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd300; divisor = 16'd15;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd300; divisor = 16'd15;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd350; divisor = 16'd17;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd350; divisor = 16'd17;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd360; divisor = 16'd18;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd360; divisor = 16'd18;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd400; divisor = 16'd20;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd400; divisor = 16'd20;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd150; divisor = 16'd7;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd150; divisor = 16'd7;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd140; divisor = 16'd6;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd140; divisor = 16'd6;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd220; divisor = 16'd11;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd220; divisor = 16'd11;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd180; divisor = 16'd9;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd180; divisor = 16'd9;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd80; divisor = 16'd4;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd80; divisor = 16'd4;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd210; divisor = 16'd10;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd210; divisor = 16'd10;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd160; divisor = 16'd8;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd160; divisor = 16'd8;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd190; divisor = 16'd9;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd190; divisor = 16'd9;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd200; divisor = 16'd10;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd80; divisor = 16'd4;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd80; divisor = 16'd4;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd140; divisor = 16'd7;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd140; divisor = 16'd7;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd240; divisor = 16'd12;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd240; divisor = 16'd12;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd160; divisor = 16'd8;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd160; divisor = 16'd8;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd100; divisor = 16'd5;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd100; divisor = 16'd5;
        #50; rst = 1'b1; start = 1'b1; dividendo = 16'd400; divisor = 16'd20;
        #5; rst = 1'b0; start = 1'b1; dividendo = 16'd400; divisor = 16'd20;
        #50; $finish;
    end

        always #1 clk = ~clk;

endmodule
```