Implementação do Divisor de 16 bits

Orleoncio Maciel de Oliveira Filho
Curso de ciências da computação
IFCE - 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* — Este relatório apresenta a implementação de um circuito divisor de 16 bits, utilizando uma máquina de estados finitos com caminho de dados (FSMD) em SystemVerilog. Inclui testes realizados através de testbench e a visualização das ondas no software GTKWave. O documento detalha desde as pesquisas iniciais até a finalização do projeto.**

I. Introdução

O objetivo desse projeto estava em elaborar, testar e comprovar a funcionalidade de um circuito divisor de 16 bits que utiliza a técnica FMDS (Finite State Machine with Data Path). Essa elaboração se dava por meio do código em SystemVerilog e também 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 possível implementar tal circuito, também pesquisamos como seria possível implementar uma máquina 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 início a etapa de elaboração do projeto onde iniciamos pela elaboração do diagrama ASM. Assim, foi possível 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 necessário realizar várias pesquisas, tanto para se entender como poderíamos aplicar a fórmula da divisão feita no papel em circuitos, quanto para saber como elaborar o diagrama ASM e transformá-lo, posteriormente, em um código em SystemVerilog.

Nas pesquisas realizadas, foi visto que uma das formas de implementar um circuito divisor seria pelo método das subtrações sucessivas, esse método consiste em subtrair o valor do divisor do dividendo até que o dividendo seja menor que o divisor, desta maneira, quando isso acontecer o valor que sobrar do dividendo será o resto, e a quantidade de vezes que a subtração ocorrer será 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 utilizado o software AutoCad de desenho técnico, esse software foi escolhido simplesmente por gosto pessoal de quem elaborou o diagrama, já para elaborar o DataPath foi utilizado o logiSim, pois se tratava de uma lógica de dados bem simples.

C. Elaboração do Código

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

Como a lógica utilizada no diagrama ASM para a realização da divisão foi a lógica das subtrações sucessivas, essa mesma lógica foi utilizada 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 série de valores diferentes em sua entrada e retornaria os valores de saída, por meio de um esquema de ondas no software gtkwave.

Como ambas as entradas do código (dividendo e divisor) são de 16 bits eram muitas as possibilidades de entradas que poderiam ser adicionadas no testbench, por isso, com a ajuda de uma inteligência artificial foram escolhidos uma série de valores específicos onde fosse possível comprovar a eficácia do circuito com um menor número de possibilidades.

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 está o resultado da elaboração do diagrama ASM e na figura 1.2 está 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 lógica das divisões múltiplas.

É possível reparar que no primeiro estado da nossa máquina que foi chamado de idle, a primeira operação é fazer com que a variável 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 início do idle, essas condicionais 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, será realizada uma operação com os registradores, onde o registrador acumulador vai receber o dividendo e o registrador quociente recebe 0.

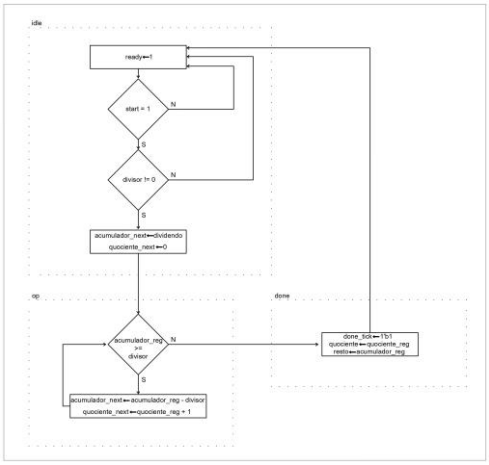

Figura 1.1 – Diagrama ASM

Após isso, no segundo estado do nosso diagrama o estado op existe outra condicional que analisa se o valor de saída 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 condicional retornar um valor falso é iniciado o próximo estado chamado de done.

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

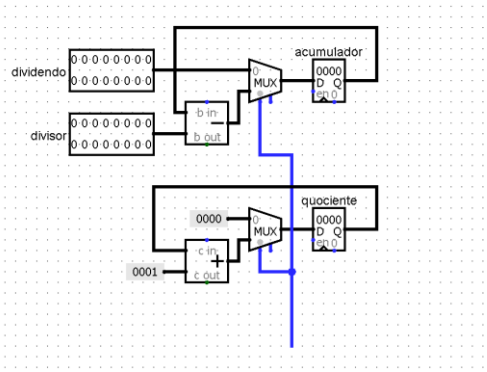

Figura 1.2 – Data Path

Como visto na imagem 1.2, está o diagrama referente ao Data Path do código, o Data Path nada mais é que a logica por trás das operações aritméticas da nossa máquina de estados.

Neste Data Path, nós 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 máquina de estados retorne o estado idle o valor 0 será adicionado ao controlador do nosso mux isso fará com que o registrador acumulador receba apenas i valor do dividendo e o registrador quociente receba apenas 0.

No estado seguinte (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 está presente no diagrama ASM.

B. Código SystemVerilog

Na figura 2 está os resultados da elaboração do nosso código de divisão de números de 16 bits. Para a elaboração deste código usamos o diagrama presente na figura 1 para usar como base, gerando assim o código, além também, do código Fibonacci apresentado em sala.

A funcionalidade é muito simples já que o método usado para a implementação da divisão foi o da subtração sucessiva, basicamente, assim como é possível ver no código da figura 1, o código se inicia na declaração de variáveis seguindo como base o código Fibonacci dado em aula, após isso os estados idle, op e done são criados assim como os registradores acumulador e quociente.

Após isso, a lógica se torna simples, pois o que o código 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 próximo ciclo, ele receberá um decremento 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 será 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 já 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*

Também foi realizado um testbench, onde era 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 utiliza 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, analisar todas as possíveis divisões entre dois números de 16 bits seria inviável, pois seriam muitas possibilidades, por isso, com a ajuda de uma IA, conseguimos diminuir a quantidade de possibilidades para facilitar a visualização, mas que mesmo assim fosse possível comprovar a eficácia do código, essas possibilidades estão presentes na figura 3.1 que representa o código do testbench.

```
`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'd50; 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'd6;
        #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'd150; 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
```

Figura 3.1 – Código do Testbench

O resultado presente na figura 3.2, que é referente ao padrão de ondas, 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 números da entrada. No padrão de ondas é visto o mesmo comportamento analisado no código onde o quociente é incrementado de 1 em 1, até chegar no valor correto e o resto é decrementado com o valor do divisor até chegar a zero ou se tornar menor que ele. Por esse motivo, algumas divisões demoram mais ciclos de clock para serem realizadas que outras.

Vale ressaltar também que, ao analisar o padrão de ondas, foi visto que ao trocar o valor do divisor e do dividendo, o circuito deveria realizar um breve reset, pois caso o contrário, a divisão entre os novos valores teria o resultado alterado pelo resultado dos valores anteriores.

IV. Conclusão

Concluindo o relatório, devo dizer que, apesar de trabalhoso foi bem enriquecedor elaborar esse trabalho final, por conter mais conteúdo, foi necessária uma melhor organização da equipe, para que fosse possível atender todas as demandas no prazo correto.

V. Referências

Segue abaixo algumas referencias utilizadas pela equipe para a elaboração de todo o projeto.

- Slides de máquinas de estados presente no classroom (*link)*
- Slideplayer – máquina de estados finitos com dados (FSMD) (*link)*
- Pdf – projeto de um divisor binário – EPUSP – PCS 2011 (*link)*

VI. Conteúdos extras

Segue abaixo um link para alguns conteúdos extras do nosso projeto, como por exemplo, arquivos .sv e .vcd.

https://github.com/orleoncio/circuito-divisor.git

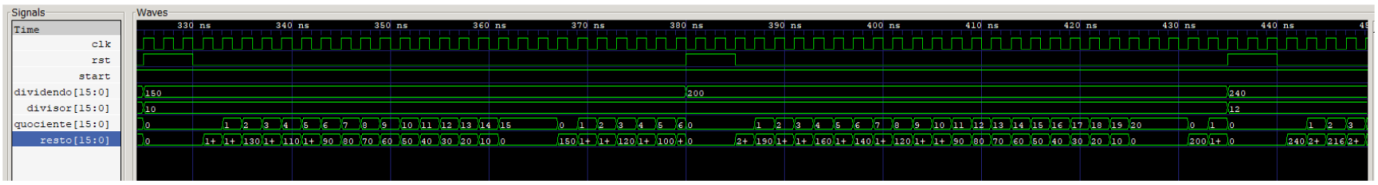

Figura 3.2 – Padrão de Ondas