Linguagem e Descrição de Hardware

Greatest Common Divisor - (GCD)

Giovanna Fantacini[1], Higor Grigorio[2], Leonardo Reneres[3], Luis[4], Raul Prado Dantas[5]

SUMÁRIO

1 Introdução **3**

2 Fundamentação Teórica **3**
2.1 Conceitos Básicos de GCD . . . 3
2.2 Algoritmo de Euclides . . . 3
2.3 Aplicações do GCD em Computação . . . 3

3 Desenho de Arquitetura **3**
3.1 Partição em Módulos . . . 3
3.2 Diagramas de Blocos . . . 3

4 Desenvolvimento e Implementação **3**
4.1 Descrição do Algoritmo em C . . . 4
4.2 Modelagem em Verilog . . . 4
4.2.1 Módulo Datapath . . . 4
4.2.2 Módulo Controle . . . 6

5 Máquina de Estados Finitos (FSM) **8**
5.1 Descrição dos Estados . . . 8
5.2 Implementação da FSM . . . 8

6 Simulações e Verificação **8**
6.1 Simulação do módulo GCD . . . 8
6.2 Ambiente de Simulação . . . 8
6.3 Resultados da Simulação . . . 8

7 Otimizações e Melhorias Futuras **8**
7.1 Otimização do Datapath . . . 8
7.1.1 Implemetação do Pipelining . . . 8
7.1.2 Benefícios do Pipelining . . . 9
7.2 Expansão do Design para Sistemas Maiores . . . 10

8 Discussão sobre Integração com Sistemas Maiores e Otimização do Datapath **10**

9 Conclusões **10**

RESUMO: Este artigo apresenta a implementação do algoritmo para cálculo do Greatest Common Divisor (GCD) em hardware, utilizando uma abordagem de Control e Datapath. Serão abordadas as principais etapas de desenvolvimento, incluindo a descrição do algoritmo em linguagem C, a modelagem dos módulos de controle e datapath em Verilog, e a simulação e validação do sistema.

PALAVRAS-CHAVE: GCD; Greatest Common Divisor; Hardware Description Language; Control; Datapath.

Greatest Common Divisor - (GCD)

ABSTRACT: This paper presents the implementation of the Greatest Common Divisor (GCD) algorithm in hardware, using a Control and Datapath approach. The main development steps will be addressed, including the description of the algorithm in C language, the modeling of control and datapath modules in Verilog, and the system simulation and validation.

KEYWORDS: GCD; Greatest Common Divisor; Hardware Description Language; Control; Datapath.

1 INTRODUÇÃO

O Greatest Common Divisor (GCD) é um problema clássico da teoria dos números, com aplicações em diversas áreas da computação. Este trabalho baseia-se nas notas de aula da disciplina CSE 141L: Introduction to Computer Architecture Lab, ministrada na Universidade da Califórnia. O objetivo é desenvolver uma implementação eficiente do GCD utilizando linguagem de descrição de hardware.

O cálculo do GCD é frequentemente utilizado em algoritmos de criptografia, compressão de dados e outros campos onde operações matemáticas eficientes são essenciais. Este projeto visa criar um design em hardware que maximize a eficiência dessas operações.

As referências devem estar citadas no trabalho conforme a sua forma de citação, como por exemplo (ALVES; RODRIGUES, 2004), (GALVANI, 2008) e (INSTRUMENTS, 2019) ou em Pandorfi, et al, (2007). Na seção Referências devem ser listadas em ordem alfabética (PANDORFI et al.,).

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Conceitos Básicos de GCD

2.2 Algoritmo de Euclides

2.3 Aplicações do GCD em Computação

3 DESENHO DE ARQUITETURA

3.1 Partição em Módulos

3.2 Diagramas de Blocos

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

Os materiais e métodos utilizados no desenvolvimento da pesquisa incluem a descrição do algoritmo em C, a modelagem dos módulos de controle e datapath em Verilog, e a simulação utilizando o ambiente

Intel QuestaSim.

4.1 Descrição do Algoritmo em C

O algoritmo para cálculo do GCD pode ser implementado de diversas maneiras. Abaixo, apresentamos uma implementação em linguagem C:

```
int GCD(int inA, int inB) {
    int swap;
    int done = 0;
    int A = inA;
    int B = inB;
    while (!done) {
        if (A < B) {
            swap = A;
            A = B;
            B = swap;
        } else if (B != 0) {
            A = A - B;
        } else {
            done = 1;
        }
    }
    return A;
}
```

4.2 Modelagem em Verilog

A implementação do GCD em hardware envolve a criação de módulos para o controle e o datapath. O datapath lida com a movimentação e transformação dos dados, enquanto o módulo de controle gerencia as operações de controle.

4.2.1 Módulo Datapath

```
module GCDdatapath#( parameter W=16 )
(
    input clk,
    input [W-1:0] operand_A,
    input [W-1:0] operand_B,
    output [W-1:0] result_data,
    input A_ld,
    input B_ld,
    input [1:0] A_sel,
    input B_sel,
    output B_zero,
    output A_lt_B,
    output [W-1:0] A_chk, B_chk, sub_chk, A_mux_chk, B_mux_chk
```

```
);

    wire [W-1:0] A;
    wire [W-1:0] B;
    wire [W-1:0] sub_out;
    wire [W-1:0] A_mux_out;
    wire [W-1:0] B_mux_out;

    Mux3#(W) A_mux
    (.in0 (operand_A),
    .in1 (sub_out),
    .in2 (B),
    .sel (A_sel),
    .out (A_mux_out) );

    register#(W) A_reg
    (.clk (clk),
    .d (A_mux_out),
    .en (A_ld),
    .q (A) );

    Mux2#(W) B_mux
    (.in0 (A),
    .in1 (operand_B),
    .sel (B_sel),
    .out (B_mux_out) );

    register#(W) B_reg
    (.clk (clk),
    .d (B_mux_out),
    .en (B_ld),
    .q (B) );

    assign B_zero = (B==0);
    assign A_lt_B = (A < B);
    assign sub_out = A - B;
    assign result_data = A;
    // send checking signals only for debugging purposes
    assign A_chk = A;
    assign B_chk = B;
    assign sub_chk = sub_out;
    assign A_mux_chk = A_mux_out;
    assign B_mux_chk = B_mux_out;
endmodule
```

4.2.2 Módulo Controle

```
module GCDcontrol(
    input input_available,
    output reg idle,
    input clk, reset,
    output reg A_ld, B_sel, B_ld,
    output reg [1:0] A_sel,
    input B_zero, A_lt_B,
    output reg result_rdy,
    input result_taken,
    output [1:0] State
);

    // States naming
    localparam WAIT = 2'd0;
    localparam CALC = 2'd1;
    localparam DONE = 2'd2;

    // Constants naming for A_mux selector
    localparam A_SEL_IN = 2'b00;
    localparam A_SEL_SUB = 2'b01;
    localparam A_SEL_B = 2'b10;
    localparam A_SEL_X = 2'b11;
    // Constants naming for B_mux selector
    localparam B_SEL_A = 1'b0;
    localparam B_SEL_IN = 1'b1;
    localparam B_SEL_X = 1'bx;

    reg [1:0] CurrentState, NextState;

    always @(posedge clk or posedge reset)
    begin
        if (reset)
            CurrentState <= WAIT;
        else
            CurrentState <= NextState;
    end

    always @(CurrentState)
    begin
        // default is to stay in the same state
        NextState <= CurrentState;
        case ( CurrentState )
            WAIT :
```

```
            if ( input_available )
                NextState <= CALC;
        CALC :
            if ( B_zero )
                NextState <= DONE;
        DONE :
            if ( result_taken )
                NextState <= WAIT;
    endcase
end

always @( * )
begin
    // Default control signals
    A_sel <= A_SEL_X;
    A_ld <= 1'b0;
    B_sel <= B_SEL_X;
    B_ld <= 1'b0;
    idle <= 1'b0;
    result_rdy = 1'b0;

    case ( CurrentState )
        WAIT :
            begin
                idle <= 1'b1;
                if(input_available)begin
                    A_sel <= A_SEL_IN;
                    B_sel <= B_SEL_IN;
                    A_ld <= 1'b1;
                    B_ld <= 1'b1;
                end
            end
        CALC :
            if ( A_lt_B )begin
                A_sel <= A_SEL_B;
                B_sel <= B_SEL_A;
                A_ld <= 1'b1;
                B_ld <= 1'b1;
            end
            else if ( !B_zero )begin
                A_sel <= A_SEL_SUB;
                A_ld <= 1'b1;
            end
        DONE :
```

```
                result_rdy <= 1'b1;
        endcase
    end

    assign State = CurrentState;
endmodule
```

5 MÁQUINA DE ESTADOS FINITOS (FSM)

5.1 Descrição dos Estados

5.2 Implementação da FSM

6 SIMULAÇÕES E VERIFICAÇÃO

Para validar a implementação, foi realizada a simulação dos módulos utilizando o ambiente Intel QuestaSim. A Figura 1 mostra a simulação do módulo GCD.

6.1 Simulação do módulo GCD

Figura 1: Simulação do módulo GCD

Os resultados da simulação confirmam que o design do GCD em hardware funciona conforme esperado. O módulo datapath executa corretamente as operações de subtração e troca, enquanto o módulo de controle gerencia os estados do sistema de maneira eficiente.

6.2 Ambiente de Simulação

6.3 Resultados da Simulação

7 OTIMIZAÇÕES E MELHORIAS FUTURAS

Mesmo se tratando de um algoritmo relativamente simples, o design do GCD em hardware pode ser otimizado e expandido para atender a requisitos mais exigentes. Nesta seção, discutiremos possíveis melhorias no design, como a implementação de pipelining no datapath e a expansão do sistema para suportar operações mais complexas.

7.1 Otimização do Datapath

Uma possível otimização é a implementação de pipelining no datapath. Isso permitiria que várias operações de subtração e troca fossem executadas em paralelo, aumentando a eficiência do sistema. Além disso, o uso de registradores adicionais poderia reduzir o número de operações de leitura e escrita na memória, melhorando ainda mais o desempenho.

7.1.1 Implemetação do Pipelining

O pipelining é uma técnica fundamental para aumentar a eficiência e o desempenho de sistemas de processamento. Ao aplicar pipelining no datapath do algoritmo GCD, podemos dividir o processo de

cálculo em estágios distintos que podem ser executados em paralelo, reduzindo o tempo total necessário para a execução e aumentando o throughput (HENNESSY; PATTERSON, 2017).

1. **Divisão do Algoritmo GCD em Estágios**

   Para o algoritmo GCD, que geralmente é implementado usando o Algoritmo de Euclides, podemos dividir o processamento em vários estágios:

   - **Estágio 1:** Cálculo do Resto - Calcula o resto da divisão entre dois números.
   - **Estágio 2:** Atualização dos Valores - Atualiza os valores dos números com base no resto calculado.
   - **Estágio 3:** Verificação da Condição de Parada - Verifica se o resto é zero, o que indica que o cálculo está concluído (KNUTH, 1997).

2. **Implementação do Pipelining**

   A implementação do pipelining para o GCD pode ser feita da seguinte maneira:

   - **Pipelide de Estágios**: Cada estágio do pipeline pode ser otimizado para realizar sua tarefa específica simultaneamente com outros estágios. Por exemplo, enquanto um estágio calcula o resto, outro pode estar atualizando os valores ou verificando a condição de parada.
   - **Buffer de Pipeline**: Adicionar buffers entre os estágios para armazenar dados temporários e permitir que o processamento continue sem interrupções.
   - **Controle de Fluxo**: Implementar mecanismos de controle para gerenciar a sincronização entre os estágios e garantir que cada estágio receba os dados no momento certo (PATTERSON; HENNESSY, 2013).

7.1.2 Benefícios do Pipelining

1. **Redução do Tempo de Execução** Ao dividir o trabalho entre vários estágios e processar diferentes partes do algoritmo simultaneamente, o tempo total para concluir a operação de GCD é reduzido.

2. **Aumento do Throughput:** A capacidade de processar múltiplos cálculos de GCD em paralelo pode aumentar o throughput geral do sistema (HARRIS; HARRIS, 2010).

3. **Paralelismo de dados:**

   - **Execução Paralela:** Implementar execução paralela para processar múltiplos pares de números simultaneamente. Isso pode ser feito usando múltiplos pipelines ou unidades de processamento.

4. **Otimização de Hardware:**

   - **Uso de Recursos Específicos:** Utilizar unidades de hardware específicas para operações matemáticas, como divisores de alto desempenho, para acelerar o cálculo do resto.
   - **Redução de Latência:** Minimizar a latência entre as operações utilizando técnicas de otimização de circuito.

- **Eficiência Energética:** Implementar técnicas para reduzir o consumo de energia, como a otimização do design do circuito e o uso eficiente dos recursos de hardware.

5. **Algoritmos Alternativos:**

   - **Exploração de Algoritmos:** Explorar algoritmos alternativos para o cálculo do GCD que possam oferecer melhorias em termos de eficiência e velocidade (BRESSOUD, 2011).

Conclusão : A aplicação de pipelining e outras técnicas de otimização pode significativamente melhorar a eficiência e o desempenho do datapath para o algoritmo GCD. Ao dividir o trabalho em estágios e implementar estratégias para reduzir latência e aumentar o throughput, podemos alcançar um sistema mais rápido e eficiente.

7.2 Expansão do Design para Sistemas Maiores

Outra melhoria futura seria a expansão do design para suportar sistemas maiores ou operações mais complexas. Por exemplo, o módulo GCD poderia ser integrado a um processador ou coprocessador específico para criptografia. Isso exigiria a adaptação do módulo para se comunicar com outros componentes do sistema e lidar com algoritmos mais avançados.

8 DISCUSSÃO SOBRE INTEGRAÇÃO COM SISTEMAS MAIORES E OTIMIZAÇÃO DO DATAPATH

É importante ressaltar que essas otimizações e melhorias futuras podem trazer desafios adicionais, como a complexidade do design e a necessidade de recursos adicionais. Portanto, é necessário realizar uma análise cuidadosa dos requisitos e restrições do sistema antes de implementar essas melhorias.

Em resumo, as otimizações e melhorias futuras no design do algoritmo GCD em hardware são uma área de interesse contínua para estudantes de Engenharia de Computação. Com a combinação certa de conhecimento teórico e prático, é possível criar soluções eficientes e inovadoras que atendam às demandas cada vez maiores da computação moderna.

9 CONCLUSÕES

Neste artigo, apresentamos a implementação do algoritmo GCD em hardware, detalhando a modelagem dos módulos de controle e datapath em Verilog. A simulação mostrou que a abordagem utilizada é eficiente e atende aos requisitos do projeto. Futuras melhorias podem incluir a otimização do datapath e a integração com outros módulos de um sistema maior.

REFERÊNCIAS

ALVES, S. P.; RODRIGUES, E. Sombreamento arbóreo e orientação de instalações avícolas. *Engenharia Agrícola*, v. 24, n. 2, p. 241 – 245, 2004.

BRESSOUD, D. M. *The Algorithm Design Manual*. 2nd. ed. [S.l.]: Springer, 2011.

GALVANI, E. Estudo comparativo dos elementos do balanço hídrico climatológico para duas cidades do estado de são paulo e para paris. Confins [Online], v. 4, n. 4, p. 1–106, 2008. Disponível em: <<http://confins.revues.org/4733>.doi:10.400/confins.4733>.

HARRIS, D. M.; HARRIS, S. L. *Digital Design and Computer Architecture*. [S.l.]: Morgan Kaufmann, 2010.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. 6th. ed. [S.l.]: Morgan Kaufmann, 2017.

INSTRUMENTS, N. *Data Acquisition*. [S.l.], 2019. Disponível em: <http://www.ni.com/pt-br/shop/select/compactdaq-controller>. Acesso em: 19 Abril. 2019.

KNUTH, D. E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3rd. ed. [S.l.]: Addison-Wesley, 1997.

PANDORFI, H. et al. Estudo da lógica fuzzy na caracterização do ambiente produtivo para matrizes gestantes. *Engenharia Agrícola*, v. 27, n. 1, p. 83–92. Disponível em: <http://www.scielo.br/pdf/eagri/v27n1/01.pdf>. Acesso em: 24 Setembro. 2007.

PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design: The Hardware/Software Interface*. 5th. ed. [S.l.]: Morgan Kaufmann, 2013.