INSTITUTO FEDERAL
São Paulo

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

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

O conceito de Greatest Common Divisor (GCD), ou Máximo Divisor Comum (MDC) em português, é fundamental na teoria dos números e tem ampla aplicação em diversas áreas da matemática e da ciência da computação. O GCD de dois números inteiros é o maior número inteiro que divide ambos sem deixar resto. Esse conceito é importante não apenas na simplificação de frações, onde o GCD é utilizado para reduzir frações ao seu menor termo, mas também em áreas mais avançadas, como criptografia, teoria de algoritmos e circuitos digitais.

Na criptografia, por exemplo, o GCD desempenha um papel crítico em algoritmos de chave pública, como o RSA, onde o conceito de coprimosidade (números cujo GCD é 1) é utilizado para garantir a segurança do sistema. Em algoritmos, o GCD é frequentemente utilizado para otimizar cálculos que envolvem divisões e reduções, sendo a base de algoritmos clássicos como o Algoritmo de Euclides, que é amplamente utilizado por sua eficiência na computação do GCD.

No contexto de circuitos digitais, especificamente em módulos de controle e datapath, como os discutidos nos materiais analisados, o GCD pode ser implementado de forma eficiente através de circuitos que utilizam registradores, multiplexadores e unidades aritméticas. Esses componentes são coordenados por um módulo de controle que decide a sequência de operações, como subtrações e trocas de valores, até que o GCD seja obtido. A implementação eficiente desses circuitos é crucial para o desempenho de sistemas que exigem cálculos frequentes de GCD, especialmente em aplicações embarcadas e de tempo real.

Em resumo, o GCD não é apenas um conceito básico da matemática, mas também um elemento essencial em várias aplicações práticas, desde a simplificação de frações até sistemas complexos de criptografia e arquiteturas de hardware digital

2.2 Algoritmo de Euclides

O Algoritmo de Euclides é um dos métodos mais antigos e eficientes para calcular o Greatest Common Divisor (GCD), ou Máximo Divisor Comum (MDC), de dois números inteiros. Esse algoritmo, atribuído ao matemático grego Euclides, tem uma fundamentação simples e robusta, baseada na observação de que o GCD de dois números não muda se o maior dos dois números é substituído pela diferença entre eles. Essa propriedade permite a construção de um algoritmo iterativo que reduz progressivamente o problema até que o divisor comum seja encontrado.

O Algoritmo de Euclides funciona através de uma série de divisões sucessivas. Dado dois números inteiros A e B, onde $A \geq B$, o algoritmo segue os seguintes passos:

1. **Divisão**: Divida A por B e obtenha o quociente q e o resto r tal que $A = Bq + r$.
2. **Substituição**: Substitua A por B e B por r.
3. **Repetição**: Repita os passos acima até que r seja igual a zero. Quando isso acontecer, o GCD é o valor de B naquele ponto.

Matematicamente, isso pode ser representado como:

$$\text{GCD}(A, B) = \text{GCD}(B, A \mod B)$$

Quando B se torna zero, A contém o GCD dos dois números iniciais.
Considere a aplicação do algoritmo para encontrar o GCD de 48 e 18:

- $48 \div 18 = 2$ com resto 12 $(48 = 18 \times 2 + 12)$.
- Substitua $A = 18$ e $B = 12$.
- $18 \div 12 = 1$ com resto 6 $(18 = 12 \times 1 + 6)$.
- Substitua $A = 12$ e $B = 6$.
- $12 \div 6 = 2$ com resto 0 $(12 = 6 \times 2 + 0)$.

Como o resto é zero, o GCD é 6.

O Algoritmo de Euclides é conhecido por sua eficiência. A cada iteração, o tamanho dos números envolvidos é reduzido, e em média, o número de iterações necessárias é proporcional ao logaritmo do

menor número entre os dois. A complexidade computacional do algoritmo é $O(\log(\min(A, B)))$, o que o torna extremamente eficiente mesmo para números muito grandes.

A eficiência do Algoritmo de Euclides o torna preferido em muitos contextos de aplicação, desde cálculos aritméticos básicos até implementações em hardware e criptografia, onde cálculos rápidos e precisos do GCD são frequentemente necessários.

2.3 Aplicações do GCD em Computação

O *Greatest Common Divisor* (GCD), ou Máximo Divisor Comum (MDC), é uma operação fundamental com diversas aplicações práticas em computação. Sua relevância se estende desde algoritmos básicos até sistemas complexos de criptografia, compressão de dados e implementação em hardware.

Uma das aplicações mais notáveis do GCD está na criptografia, especialmente em sistemas de criptografia assimétrica, como o algoritmo RSA. O RSA depende da dificuldade de fatorar grandes números primos, e a função GCD é usada para determinar se dois números são *coprimos*, ou seja, se seu GCD é 1. A coprimosidade é um conceito crucial no processo de geração de chaves no RSA, onde o GCD garante que a chave pública e o módulo sejam adequados para a operação de criptografia e descriptografia.

Outra aplicação importante do GCD é na compressão de dados. Técnicas de compressão como o *Run-Length Encoding* (RLE) podem utilizar o GCD para determinar padrões de repetição e reduzir a redundância nos dados. Além disso, algoritmos de compressão baseados em análise de fatores de números, como o *factoring-based compression*, usam o GCD para simplificar expressões numéricas e otimizar o armazenamento.

Em **sistemas de controle**, especialmente em hardware, o GCD é utilizado para simplificar os cálculos que envolvem múltiplos ciclos de operação ou para sincronizar sinais em circuitos digitais. No contexto de *Control and Datapath*, a operação de GCD pode ser implementada em circuitos que exigem operações de redução, como divisões sucessivas, facilitando a minimização de recursos computacionais.

A operação de GCD é relevante em hardware porque ela pode ser implementada de forma eficiente utilizando componentes básicos como registradores, multiplexadores e subtratores, que são comuns em *datapaths* de circuitos digitais. A eficiência do GCD, especialmente quando implementado com o *Algoritmo de Euclides*, permite que ele seja utilizado em sistemas de tempo real e em dispositivos embarcados, onde a velocidade de processamento e o uso mínimo de recursos são críticos. Além disso, o GCD é frequentemente empregado em **algoritmos de síntese lógica** para otimizar o design de circuitos e na verificação formal de *hardware*.

Em resumo, o GCD é uma operação essencial em diversas áreas da computação, proporcionando uma base sólida para algoritmos criptográficos, técnicas de compressão de dados e a implementação eficiente de sistemas de controle em *hardware* digital.

3 DESENHO DE ARQUITETURA

Esta seção descreve a arquitetura do sistema, incluindo a partição em módulos de controle e datapath, e os diagramas de blocos que ilustram a interação entre os componentes.

3.1 Partição em Módulos

A arquitetura do sistema GCD foi organizada em dois módulos principais: Control e Datapath. Esta divisão permite uma clara separação de responsabilidades, permite uma maior modularidade,

facilita a implementação, manutenção e expansão do sistema.

3.1.1 Módulo de Controle

O módulo de Controle é responsável por gerenciar o fluxo de dados entre os diferentes componentes do sistema e controlar as transições de estado. Ele coordena as operações do Datapath com base nas condições de controle, assegurando que as etapas do cálculo do GCD sejam executadas na ordem correta. Este módulo inclui uma Máquina de Estados Finitos (FSM), a qual será discutida posteriormente, que determina a sequência de operações, incluindo a inicialização, execução do cálculo e término.

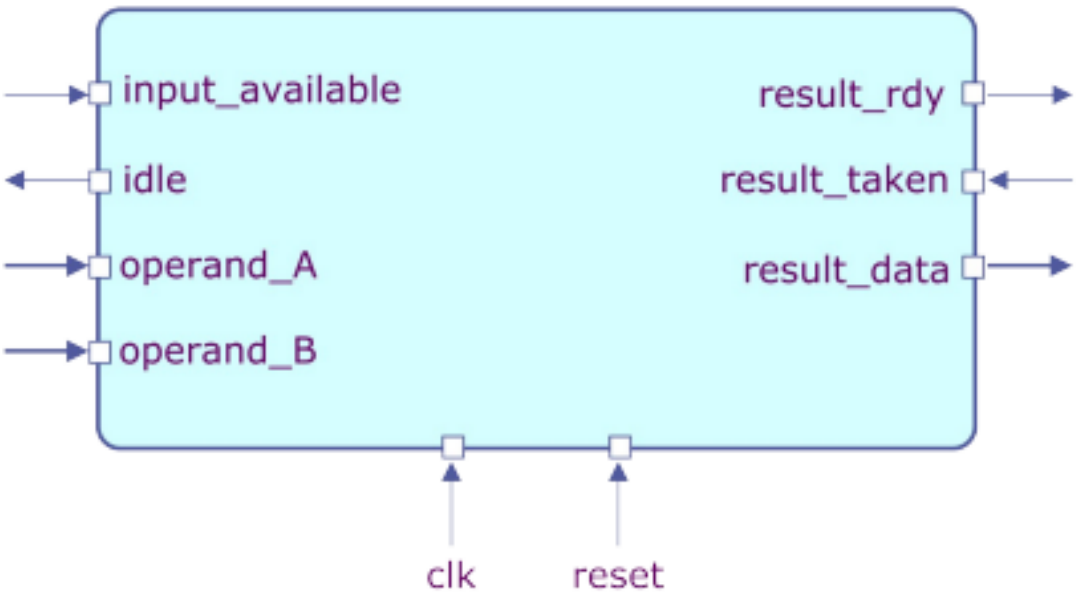
3.1.2 Módulo Datapath

O Datapath realiza as operações aritméticas necessárias para o cálculo do GCD, como subtrações e comparações entre os valores de entrada. Este módulo contém registradores, multiplexadores, e a unidade de subtração, que juntos formam o núcleo da operação matemática. A eficiência do Datapath é crítica para o desempenho global do sistema, uma vez que ele executa as operações fundamentais repetidamente até que o resultado seja obtido.

3.2 Diagramas de Blocos

O diagrama de bloco na Figura 1 ilustra a arquitetura de nível superior do sistema GCD, apresentando todas as entradas e saídas em um único bloco. Este diagrama de blocos demonstra como os módulos de Controle e Datapath estão interconectados e como o fluxo de dados é gerenciado entre eles.

Figura 1: Diagrama de Blocos do Sistema GCD

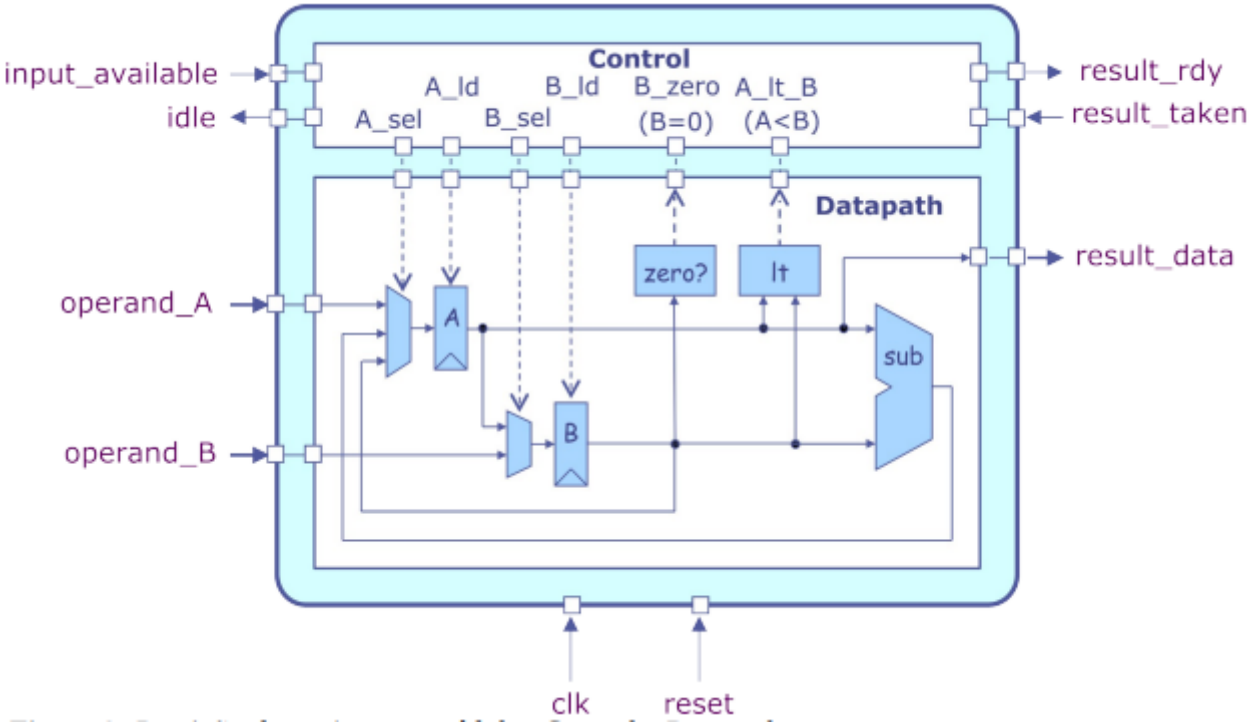


Fonte: (TAYLOR, 2019)

O diagrama de blocos na Figura 2 ilustra como os componentes principais do sistema GCD se interconectam.

O Datapath inclui os registradores, multiplexadores e a unidade de subtração, enquanto o Control FSM gerencia as operações com base nos sinais de controle.

Figura 2: Diagrama de Blocos do Sistema GCD



Fonte: (TAYLOR, 2019)

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

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

4.1 Descrição do Algoritmo em C

O código a seguir é uma implementação da função GCD (Greatest Common Divisor - Maior Divisor Comum). Essa função calcula o maior divisor comum entre dois números inteiros.

A função GCD recebe dois parâmetros: inA e inB, que representam os dois números inteiros para os quais queremos calcular o maior divisor comum.

O código utiliza um algoritmo chamado "algoritmo de Euclides"para calcular o maior divisor comum. O algoritmo de Euclides é baseado na observação de que o maior divisor comum entre dois números não muda se o menor número for subtraído do maior número repetidamente até que um dos números seja igual a zero.

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;
}
```

Aqui está uma explicação passo a passo do funcionamento do código:

1. O código declara algumas variáveis locais:

   `swap`

   ,

   `done`

,

`A`

e

`B`

.

`swap`

é usada para trocar os valores de

`A`

e

`B`

quando necessário.

`done`

é uma variável de controle que indica se o cálculo do maior divisor comum está concluído.

`A`

e

`B`

são inicializadas com os valores dos parâmetros

`inA`

e

`inB`

, respectivamente.

2. O código entra em um loop

   `while`

   que continua até que a variável

   `done`

seja igual a 1.

3. O loop é usado para realizar as iterações necessárias para calcular o maior divisor comum.
4. Dentro do loop, há uma estrutura

   `if-else`

   que verifica três condições:

   (a) Se

   `A`

   for menor que

   `B`

   , os valores de

   `A`

   e

   `B`

   são trocados usando a variável

   `swap`

   . Isso garante que

   `A`

   seja sempre o maior número.

   (b) Se

   `B`

   for diferente de zero,

   `A`

   é atualizado para

   `A - B`

   . Isso realiza a subtração repetida até que um dos números seja igual a zero.

   (c) Se nenhuma das condições anteriores for verdadeira, significa que o cálculo do maior divisor comum está concluído e a variável

   `done`

   é definida como 1 para sair do loop.

5. Após o loop, o código retorna o valor de

   `A`

, que representa o maior divisor comum entre os números

`inA`

e

`inB`

.

Por exemplo, se chamarmos a função GCD(24, 36), o algoritmo de Euclides será aplicado da seguinte maneira:

1. Na primeira iteração,

   `A`

   é atualizado para 36 e

   `B`

   é atualizado para 24.
2. Na segunda iteração,

   `A`

   é atualizado para 12 e

   `B`

   é atualizado para 24.
3. Na terceira iteração,

   `A`

   é atualizado para 12 e

   `B`

   é atualizado para 12.
4. Na quarta iteração,

   `A`

   é atualizado para 0 e

   `B`

   é atualizado para 12.
5. Como

   `B`

   é igual a zero, o cálculo do maior divisor comum está concluído e o valor retornado é 12.

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

O código a seguir é um exemplo de um módulo em Verilog que implementa o datapath para um algoritmo de cálculo do Máximo Divisor Comum (GCD - Greatest Common Divisor).

```
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
```

A seguir, o funcionamento do código passo a passo:

O módulo GCDdatapath recebe alguns parâmetros, sendo o principal deles W, que define o tamanho dos operandos e do resultado em bits.

O módulo possui várias entradas e saídas, incluindo:

- `clk`

  : um sinal de clock para sincronizar as operações.

- `operand_A`

  e

  `operand_B`

  : os operandos de entrada para o cálculo do GCD.

- `result_data`

  : o resultado do cálculo do GCD.

- `A_ld`

  e

`B_ld`

: sinais de controle para carregar os operandos A e B.

- `A_sel`

  e

  `B_sel`

  : sinais de seleção para escolher entre os operandos A e B.

- `B_zero`

  : um sinal indicando se o operando B é igual a zero.

- `A_lt_B`

  : um sinal indicando se o operando A é menor que o operando B.

- `A_chk`

  ,

  `B_chk`

  ,

  `sub_chk`

  ,

  `A_mux_chk`

  e

  `B_mux_chk`

  : sinais de depuração para verificar os valores internos do datapath.

O módulo possui várias fiações (wires) para conectar os componentes internos:

`A, B, sub_out, A_mux_out`

e

`B_mux_out`

: fiações para transportar os valores dos operandos e resultados intermediários.

O módulo utiliza vários componentes internos para realizar as operações do datapath:

- `Mux3#(W) A_mux`

  : um multiplexador de 3 entradas que seleciona entre o operando A, o resultado da subtração e o operando B, com base no sinal de seleção

`A_sel`

. O resultado é armazenado em

`A_mux_out`

.

- `register#(W) A_reg`

  : um registrador que armazena o valor de

  `A_mux_out`

  e o atualiza no sinal de clock

  `clk`

  quando o sinal de controle

  `A_ld`

  está ativo. O valor atualizado é armazenado em

  `A`

  .

- `Mux2#(W) B_mux`

  : um multiplexador de 2 entradas que seleciona entre o valor atualizado de

  `A`

  e o operando

  `B`

  , com base no sinal de seleção

  `B_sel`

  . O resultado é armazenado em

  `B_mux_out`

  .

- `register#(W) B_reg`

  : um registrador que armazena o valor de

  `B_mux_out`

e o atualiza no sinal de clock

`clk`

quando o sinal de controle

`B_ld`

está ativo. O valor atualizado é armazenado em

`B`

.

As atribuições assign são usadas para definir os valores das saídas do módulo:

- `B_zero`

  é atribuído como verdadeiro se o valor de

  `B`

  for igual a zero.

- `A_lt_B`

  é atribuído como verdadeiro se o valor de

  `A`

  for menor que o valor de

  `B`

  .

- `sub_out`

  é atribuído como a diferença entre

  `A`

  e

  `B`

  .

- `result_data`

  é atribuído como o valor de

  `A`

.

- Os sinais de depuração

  `A_chk`

  ,

  `B_chk`

  ,

  `sub_chk`

  ,

  `A_mux_chk`

  e

  `B_mux_chk`

  são atribuídos aos respectivos valores internos do datapath.

O módulo GCDdatapath encapsula todas essas operações e fiações para formar um datapath completo para o cálculo do GCD.

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 3 mostra a simulação do módulo GCD.

6.1 Simulação do módulo GCD

Figura 3: 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

7.1 Otimização do Datapath

7.2 Expansão do Design para Sistemas Maiores

8 DISCUSSÃO SOBRE INTEGRAÇÃO COM SISTEMAS MAIORES

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.

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>.

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

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.

TAYLOR, M. B. *CSE 141L: Introduction to Computer Architecture Laboratory*. [S.l.], 2019. Disponível em: <https://cseweb.ucsd.edu/classes/sp10/cse141L/pdf/02/02-Verilog2.pdf>. Acesso em: 13 Ago. 2024.