UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO

AVALIAÇÃO 2- CIRCUITOS LÓGICOS II

PROJETO RELÓGIO DIGITAL

João Pessoa

2025

PROJETO RELÓGIO DIGITAL

EQUIPE:

GABRIEL LORENZO XAVIER

IAGO VITOR LOPES DAS CHAGAS

JOÃO PEDRO PEREIRA MARANHÃO

LUIS EDUARDO PEREIRA NUNES DA COSTA

THIAGO SERGIO LIMA DE OLIVEIRA

SUMÁRIO:

1. INTRODUÇÃO:

O presente relatório tem como objetivo apresentar o desenvolvimento de um **relógio digital** implementado em *SystemVerilog.* O projeto foi concebido a fim de explorar conceitos fundamentais de **projeto digital**, **linguagens de descrição de hardware (HDL)** e **síntese lógica em FPGA**, consolidando os conhecimentos adquiridos na disciplina.

O relógio digital tem como funções principais a contagem de horas, minutos e segundos, bem como a correta atualização desses valores em tempo real. Para isso, foi necessário estruturar módulos que representassem desde divisores de frequência e contadores até a lógica responsável pela formatação e exibição dos dados.

A utilização do *Quartus* possibilitou a simulação, verificação e síntese do circuito, garantindo que o projeto atendesse aos requisitos de funcionamento e pudesse ser implementado em um dispositivo FPGA. Além disso, o desenvolvimento permitiu exercitar práticas de **modularização do código**, **testbench para validação** e **otimização de recursos lógicos**, aproximando o trabalho das demandas encontradas em aplicações reais de sistemas digitais.

2.OBJETIVO:

O objetivo do projeto é implementar um relógio digital especifica de que possa contar em tempo real e de forma precisa segundos, minutos e horas, através da manipulação de sinais periódicos de um clock em circuitos contadores, utilizando da linguagem SystemVerilog no software Quartus 2, simulando em forma de onda os sinais do relógio, e depois implementando o funcionamento dele na placa FPGA, demostrando nos displays de 7 segmentos o funcionamento do relógio.

3.METODOLOGIAS:

3.1 MÉTODOS:

O processo de criação do relógio digital foi dividido em duas etapas, a programação e o teste, as quais serão aqui descritas:

3.2 PROJETO DO RELÓGIO:

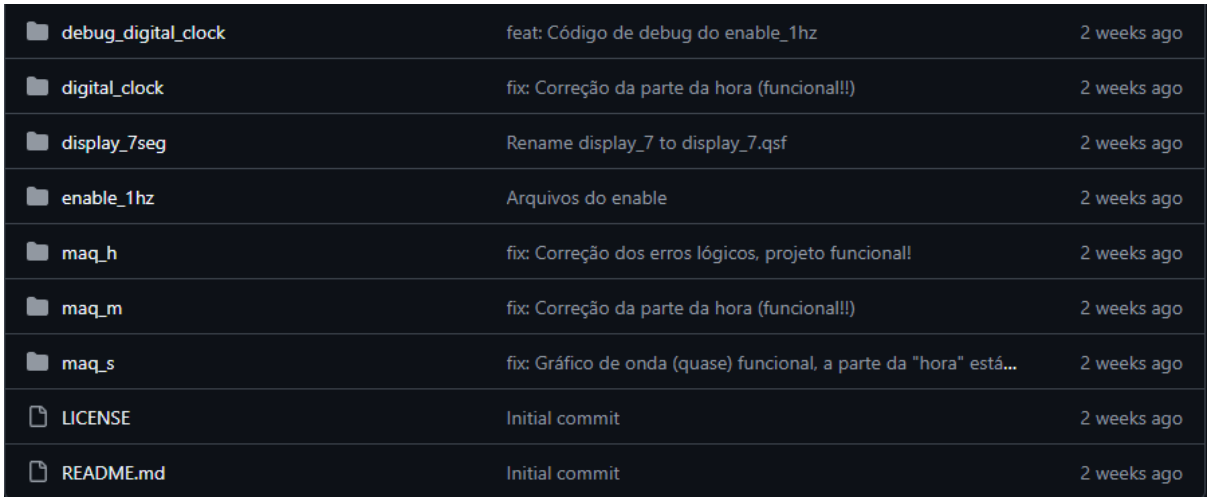
Como já afirmado antes, o projeto foi criado através da linguagem de descrição de hardware SystemVerilog e implementado utilizando o software Quartus 2 para a escrita e compilação do código assim como fornecida nas especificações do projeto.

Para o funcionamento apropriado do relógio e sincronização com o tempo real, o projeto é dividido em diferentes módulos, o módulo enable 1Hz, segundos, minutos e horas, o qual cada um vai se interagir entre si e contar com o pulso positivo do clock as suas respectivas unidades de tempo, diminuindo a frequência de pulsação a cada máquina para passar o sinal para a próxima máquina.

Ademais após processado o sinal, há um módulo decodificador para a conversão dos bits em tempo real para o display de 7 segmentos, que mostrará na placa FPGA o tempo real contado de forma precisa pelo programa.

3.3 PROGRAMAÇÃO:

Para a programação do projeto dividimos cada módulo em uma pasta específica seguindo a seguinte estrutura de diretórios:



Inicialmente o módulo enable_1Hz, entra em ação, com a tarefa de converter, usando flip-flops, o sinal de entrada do clock de 50MHz em um sinal de saída de 1Hz, que será utilizado como sinal de entrada para contar os segundos no módulo maq_s.

enable_1hz.sv

```
module enable_1hz(
    input enable_clock,
    input enable_reset,
```

```
    output logic enable_pulseout
);

logic [25:0] contador;

always_ff @(posedge enable_clock)
    if(!enable_reset)
        contador <= 26'd0;
    else
        if(contador == 26'd49999999)
            contador <=26'd0;
        else
            contador <= contador + 26'd1;

always_comb
    enable_pulseout <= (contador == 26'd49999999);

endmodule
```

Após a ação do módulo enable_1Hz, o sinal é enviado ao módulo maq_s, responsável pela contagem dos segundos utilizando dois registradores: maqs_Lsd, que representa a unidade dos segundos e varia de 0 a 9, e maqs_Msd, que representa a dezena dos segundos e varia de 0 a 5. A cada pulso válido de clock, quando o enable está ativo, a unidade dos segundos é incrementada; quando maqs_Lsd atinge 9, ele é zerado e maqs_Msd é incrementado. Quando maqs_Msd está em 5 e maqs_Lsd em 9, ou seja, no instante de 59 segundos, o módulo gera o sinal maqs_addminuto = 1 para indicar que os minutos devem ser incrementados, e ambos os contadores são resetados para zero, reiniciando o ciclo de contagem. Dessa forma, o módulo maq_s assegura a contagem correta de 00 a 59 segundos e propaga o sinal de incremento para o módulo de minutos no momento exato da virada do minuto.

```
module maq_s(
    input  logic maqs_clock,
    input  logic maqs_reset,
    input  logic maqs_enable,
    output logic [3:0] maqs_Lsd,
    output logic [2:0] maqs_Msd,
    output logic maqs_addminuto
);

    // Estado atual
    logic [3:0] Lsd_reg, Lsd_next;
    logic [2:0] Msd_reg, Msd_next;

    // Lógica sequencial
    always_ff @(posedge maqs_clock or negedge maqs_reset) begin
        if (!maqs_reset) begin
```

```
            Lsd_reg <= 4'd0;
            Msd_reg <= 3'd0;
        end
        else if (maqs_enable) begin
            Lsd_reg <= Lsd_next;
            Msd_reg <= Msd_next;
        end
    end

    // Lógica combinacional (próximo estado)
    always_comb begin
        Lsd_next = Lsd_reg;
        Msd_next = Msd_reg;

        if (Msd_reg == 3'd5 && Lsd_reg == 4'd9) begin
            Lsd_next = 4'd0;
            Msd_next = 3'd0;
        end
        else if (Lsd_reg == 4'd9) begin
            Lsd_next = 4'd0;
            Msd_next = Msd_reg + 1'b1;
        end
        else begin
            Lsd_next = Lsd_reg + 1'b1;
        end
    end

    // Saídas
    assign maqs_Lsd = Lsd_reg;
    assign maqs_Msd = Msd_reg;
    assign maqs_addminuto = (Msd_reg == 3'd5 && Lsd_reg == 4'd9);

endmodule
```

Analogamente, o módulo dos minutos, maq_m, recebe como entrada o pulso de incremento enviado pelo módulo dos segundos, maqs_addminuto, para atualizar seu estado atual. Ele utiliza dois registradores: maqm_Lsd, que representa a unidade dos minutos e varia de 0 a 9, e maqm_Msd, que representa a dezena dos minutos e varia de 0 a 5. A cada pulso válido de incremento, quando o enable está ativo, a unidade dos minutos é incrementada; quando maqm_Lsd atinge 9, ele é zerado e maqm_Msd é incrementado. Quando maqm_Msd está em 5 e maqm_Lsd em 9, ou seja, no instante de 59 minutos, o módulo ativa o sinal maqm_incrementahora para indicar que as horas devem ser incrementadas, e ambos os contadores são resetados para zero, reiniciando o ciclo de contagem. Dessa forma, o módulo

maq_m assegura a contagem correta de 00 a 59 minutos e propaga o sinal de incremento para o módulo das horas no momento exato da virada da hora.

maq_m.sv

```
module maq_m(
    input  logic maqm_clock,
    input  logic maqm_reset,
    input  logic maqm_enable,
    input  logic maqm_incremento,     // pulso vindo dos segundos
    output logic [3:0] maqm_Lsd,
    output logic [2:0] maqm_Msd,
    output logic maqm_incrementahora  // pulso para horas
);

    logic [3:0] Lsd_reg, Lsd_next;
    logic [2:0] Msd_reg, Msd_next;

    // Estado atual
    always_ff @(posedge maqm_clock or negedge maqm_reset) begin
        if (!maqm_reset) begin
            Lsd_reg <= 4'd9;
            Msd_reg <= 3'd5;
        end
        else if (maqm_enable && maqm_incremento) begin
            Lsd_reg <= Lsd_next;
            Msd_reg <= Msd_next;
        end
    end

    // Lógica combinacional
    always_comb begin
        Lsd_next = Lsd_reg;
        Msd_next = Msd_reg;

        if (Msd_reg == 3'd5 && Lsd_reg == 4'd9) begin
            Lsd_next = 4'd0;
            Msd_next = 3'd0;
        end
        else if (Lsd_reg == 4'd9) begin
            Lsd_next = 4'd0;
            Msd_next = Msd_reg + 1'b1;
        end
        else begin
            Lsd_next = Lsd_reg + 1'b1;
        end
    end
```

```
    // Saídas
    assign maqm_Lsd = Lsd_reg;
    assign maqm_Msd = Msd_reg;
    assign maqm_incrementahora = (Msd_reg == 3'd5 && Lsd_reg == 4'd9);

endmodule
```

O último módulo, maq_h, é responsável pela contagem das horas e recebe como entrada o pulso de incremento proveniente do módulo dos minutos quando estes atingem 59. Ele utiliza dois registradores: maqh_Lsd, que representa a unidade das horas e varia de 0 a 9, e maqh_Msd, que representa a dezena das horas e varia de 0 a 2. A cada pulso válido de incremento, quando o enable está ativo, a unidade das horas é incrementada; ao atingir 9 unidades, ela é zerada e a dezena é incrementada, respeitando a limitação de 23 horas. Quando o relógio alcança 23 horas (2 dezenas e 3 unidades), ambos os contadores são zerados, reiniciando o ciclo de 24 horas. Como é o último estágio do relógio, o módulo maq_h não propaga sinal para outro bloco, apenas envia seus valores para os displays, garantindo a contagem contínua e correta no intervalo de 00 a 23 horas.

```
module maq_h(
    input  logic maqh_clock,
    input  logic maqh_reset,
    input  logic maqh_enable,
    input  logic maqh_incremento_min,
    input  logic maqh_incremento_seg,
    output logic [3:0] maqh_Lsd,
    output logic [1:0] maqh_Msd
);

    logic [3:0] Lsd_reg, Lsd_next;
    logic [1:0] Msd_reg, Msd_next;

    // Pulso de incremento da hora -> quando estamos em 59 min e 59 seg
    logic maqh_incremento;
    assign maqh_incremento = (maqh_incremento_min && maqh_incremento_seg);

    // Estado atual
    always_ff @(posedge maqh_clock or negedge maqh_reset) begin
        if (!maqh_reset) begin
            Lsd_reg <= 4'd0;
            Msd_reg <= 2'd0;
```

```
        end
        else if (maqh_enable && maqh_incremento) begin
            Lsd_reg <= Lsd_next;
            Msd_reg <= Msd_next;
        end
    end

    // Lógica combinacional
    always_comb begin
        Lsd_next = Lsd_reg;
        Msd_next = Msd_reg;

        // Caso: 23 → 00
        if (Msd_reg == 2'd2 && Lsd_reg == 4'd3) begin
            Lsd_next = 4'd0;
            Msd_next = 2'd0;
        end
        // Caso: passou de 9 unidades (quando dezena < 2)
        else if (Lsd_reg == 4'd9 && Msd_reg < 2) begin
            Lsd_next = 4'd0;
            Msd_next = Msd_reg + 1'b1;
        end
        // Caso: passou de 3 unidades (quando dezena == 2)
        else if (Msd_reg == 2'd2 && Lsd_reg == 4'd3) begin
            Lsd_next = 4'd0;
            Msd_next = 2'd0;
        end
        // Incremento normal
        else begin
            Lsd_next = Lsd_reg + 1'b1;
        end
    end

    // Saídas
    assign maqh_Lsd = Lsd_reg;
    assign maqh_Msd = Msd_reg;

endmodule
```

Como somos humanos e não entedemos pulsos elétricos, se faz necesário um módulo para ser feita a representação numérica visual do horáriop atual que está sendo contado pelo

relógio, assim usamos o módulo display_7seg.sv para captar o estado atual das máquinas de segundo, minuto e hora e representar em números.

```
module display_7(
    input [3:0] bcd_bcd_in,
    output logic [6:0]bcd_display_out
);

always_comb
    case (bcd_bcd_in)
        4'd0: bcd_display_out <= 7'b0000001;
        4'd1: bcd_display_out <= 7'b1001111;
        4'd2: bcd_display_out <= 7'b0010010;
        4'd3: bcd_display_out <= 7'b0000110;
        4'd4: bcd_display_out <= 7'b1001100;
        4'd5: bcd_display_out <= 7'b0100100;
        4'd6: bcd_display_out <= 7'b0100000;
        4'd7: bcd_display_out <= 7'b0001111;
        4'd8: bcd_display_out <= 7'b0000000;
        4'd9: bcd_display_out <= 7'b0000100;
        default: bcd_display_out <= 7'b1111111;
    endcase

endmodule
```

O módulo topo.sv é o bloco responsável por integrar todos os módulos que compõem o projeto do relógio digital. Ele atua como a camada central de conexão, recebendo o sinal de clock e o reset da FPGA, distribuindo-os para os módulos internos e organizando a comunicação entre os contadores de segundos, minutos e horas. Além disso, realiza a interface com os decodificadores de display de 7 segmentos, garantindo que os valores contados sejam exibidos corretamente em tempo real.

As entradas do módulo são o clock, que corresponde ao sinal de relógio do sistema gerado pela FPGA e que serve de base para toda a contagem temporal, e o reset, responsável por reinicializar todos os contadores e reiniciar o funcionamento do relógio a partir de zero. As saídas são compostas por seis vetores de 7 bits, cada um destinado ao controle de um display de 7 segmentos: s_Lsd e s_Msd representam as unidades e dezenas dos segundos, m_Lsd e m_Msd representam as unidades e dezenas dos minutos, e h_Lsd e h_Msd representam as unidades e dezenas das horas.

O funcionamento pode ser descrito em três etapas principais. Primeiro, o divisor de frequência (enable_1Hz) reduz o clock original da FPGA, de 50 MHz, para um pulso de 1 Hz, que é distribuído para os contadores. Em seguida, os contadores maq_s, maq_m e maq_h realizam a contagem de segundos, minutos e horas, propagando os sinais de incremento

conforme atingem seus limites (59 segundos, 59 minutos e 23 horas, respectivamente). Por fim, os valores de cada contador são enviados aos módulos display_7, que convertem os números binários em sinais apropriados para acender os segmentos de LEDs, possibilitando a visualização direta do horário no hardware.

Dessa forma, o módulo topo garante a integração completa do sistema, coordenando a contagem e a exibição do tempo em seis displays, representando o formato HH:MM:SS com precisão e confiabilidade.

```
module topo(
    input clock,
    input reset,
    output logic [6:0] s_Lsd,
    output logic [6:0] s_Msd,
    output logic [6:0] m_Lsd,
    output logic [6:0] m_Msd,
    output logic [6:0] h_Lsd,
    output logic [6:0] h_Msd
);

logic enable1hz;

//maq_s
logic [3:0] bcd_s_Lsd;
logic [2:0] bcd_s_Msd;
logic incrementa_minuto;

//maq_m
logic [3:0] bcd_m_Lsd;
logic [2:0] bcd_m_Msd;
logic incrementa_hora;

//maq_h
logic [3:0] bcd_h_Lsd;
logic [2:0] bcd_h_Msd;



enable_1hz meuhabilitador(
.enable_clock(clock),
.enable_reset(reset),
.enable_pulseout(enable1hz));
```

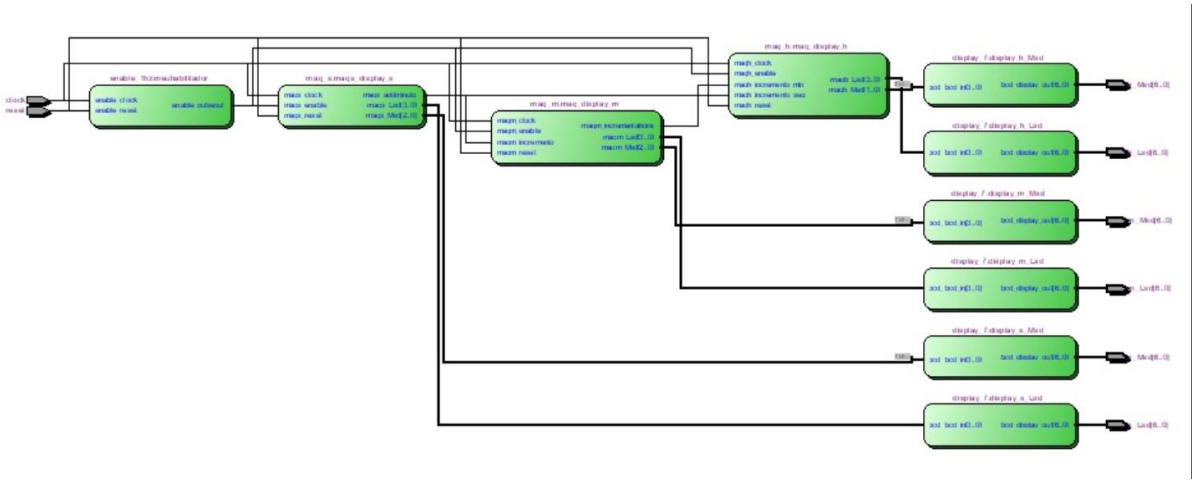
```
//maq_s + display_7
maq_s maqs_display_s(
.maqs_clock(clock),
.maqs_reset(reset),
.maqs_enable(enable1hz),
.maqs_Lsd(bcd_s_Lsd),
.maqs_Msd(bcd_s_Msd),
.maqs_addminuto(incrementa_minuto));

display_7 display_s_Lsd(
.bcd_bcd_in(bcd_s_Lsd),
.bcd_display_out(s_Lsd));

display_7 display_s_Msd(
.bcd_bcd_in({1'b0,bcd_s_Msd}),
.bcd_display_out(s_Msd));



//maq_m + display
maq_m maq_display_m(
.maqm_clock(clock),
.maqm_reset(reset),
.maqm_enable(enable1hz),
.maqm_incremento(incrementa_minuto),
.maqm_Lsd(bcd_m_Lsd),
.maqm_Msd(bcd_m_Msd),
.maqm_incrementahora(incrementa_hora));

display_7 display_m_Lsd(
.bcd_bcd_in(bcd_m_Lsd),
.bcd_display_out(m_Lsd));

display_7 display_m_Msd(
.bcd_bcd_in({1'b0,bcd_m_Msd}),
.bcd_display_out(m_Msd));

//maq_h + display
maq_h maq_display_h(
.maqh_clock(clock),
.maqh_reset(reset),
.maqh_enable(enable1hz),
.maqh_incremento_min(incrementa_hora),
.maqh_incremento_seg(incrementa_minuto),
.maqh_Lsd(bcd_h_Lsd),
.maqh_Msd(bcd_h_Msd));
```

```
display_7 display_h_Lsd(
.bcd_bcd_in(bcd_h_Lsd),
.bcd_display_out(h_Lsd));

display_7 display_h_Msd(
.bcd_bcd_in({2'd0,bcd_h_Msd}),
.bcd_display_out(h_Msd));

endmodule
```

3.4 SIMULAÇÃO:

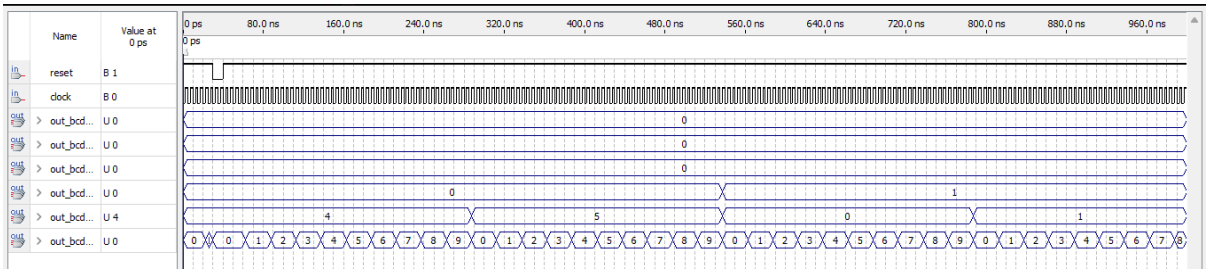
Nos testes utilizamos o software Quartus 2 para simular a topologia do circuito e as formas de onda dos sinais, com os estados atuais em cada máquina que representam as horas segundo e minutos:

Aqui temos a representação gráfica de como os circuitos do relógio funcionam:

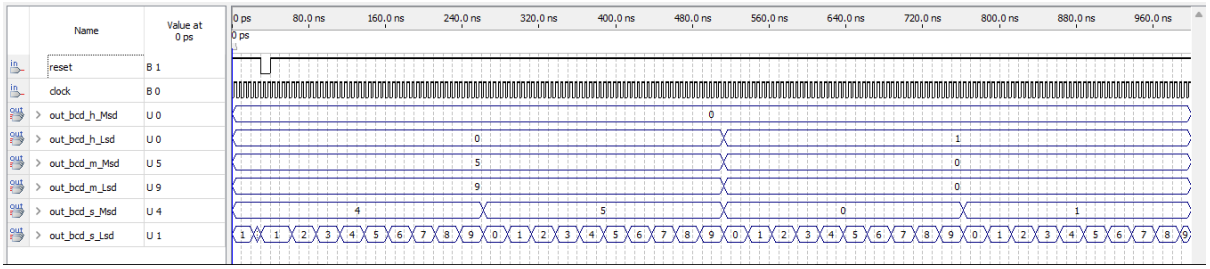


E como teste do funcionamento do codigo, simulamos graficamente as formas de onda do relógio:

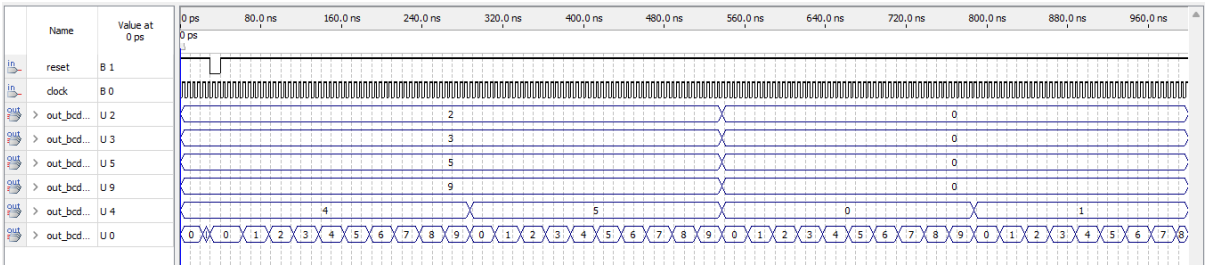
Incrementa Minuto:



Incrementa Hora:



Incrementa hora (Crítico):



4. RESULTADOS:

Dentro dos objetivos estabelecidos para o projeto, o relógio implementado demonstrou capacidade de contar corretamente segundos minutos e horas conforme o tempo certp, incluindo casos de virada de minutos, hora e dia. Todos os testes realizados foram condizentes com os resultados esperados e relógio pode ser utilizado com confiança.

5.CONCLUSÃO:

Com a implementação do relógio digital em SystemVerilog, conseguimos alcançar o objetivo principal do projeto: construir um sistema capaz de contar e exibir horas, minutos e segundos em tempo real dentro de uma FPGA. A modularização do código, separando cada parte do relógio (divisor de frequência, segundos, minutos, horas e decodificador para display), facilitou tanto o desenvolvimento quanto a validação do funcionamento.

Durante o processo foi possível reforçar conceitos importantes de circuitos digitais, como contadores síncronos, propagação de sinais de enable entre módulos e representação em BCD. Além disso, a simulação no Quartus confirmou que a lógica projetada atende corretamente aos requisitos, mostrando a evolução esperada dos valores no decorrer do tempo e garantindo que a transição entre segundos, minutos e horas ocorra sem erros.

Assim, o projeto não só cumpriu o que foi pedido, mas também serviu como prática de integração de diferentes módulos em um sistema maior, próximo do que é feito em aplicações reais de hardware digital.

Para mais informações acerca do código, aqui abaixo se encontra o link para o repositório no GitHub: https://github.com/thgsergio/Digital-Clock

-Fim do relatório-