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 LOPES DA SILVA COSTA

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:

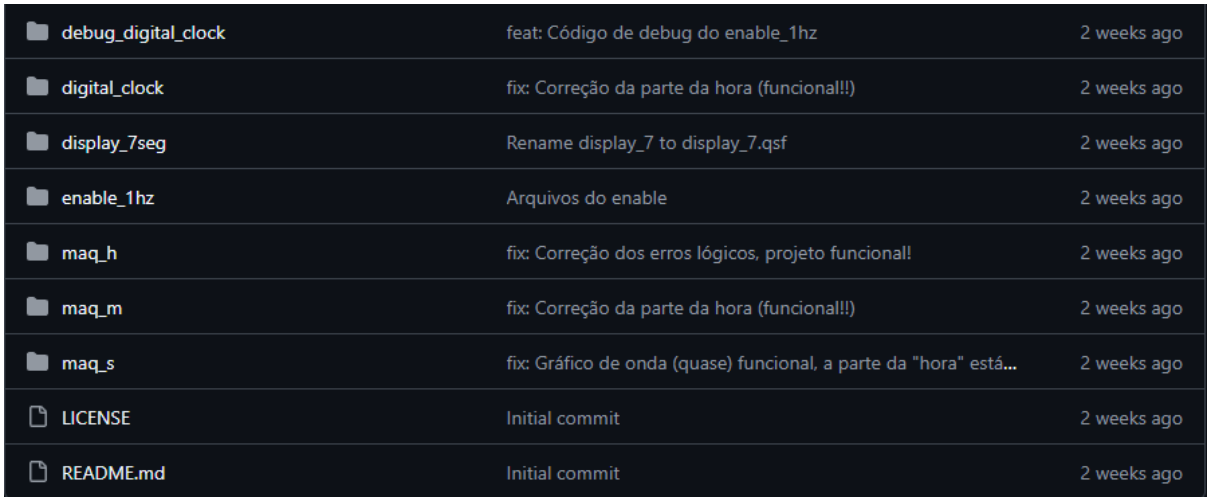
Como já afirmado antes, o projeto foi criado através da linguagem de descrição de hardware SystemVerilog e implementado utilizando o software Quartus 2 para a escrita e compilação do código assim como fornecida nas especificações do projeto.

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 o sinal é enviado ao módulo maq_s, o qual analisa o número atual, e então faz o incremento dos segundos conforme a borda positiva do clock, caso o segundo esteja em 58 segundos, a máquina já "prepara" para o envio o sinal para a atualização no módulo dos minutos que é enviado aos 59 segundos, e quando chega a 60 segundos, se reinicia para 0, repetindo o processo.

maq_s.sv

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

always_ff @(posedge maqs_clock or negedge maqs_reset) begin
    if(!maqs_reset)
    begin
      maqs_Lsd <= 4'b0000;
      maqs_Msd <= 3'b000;
      maqs_addminuto <= 1'b0;
    end
    else if(maqs_enable)
    begin
```

```
        if(maqs_Lsd >= 4'b1001)
        begin
            maqs_Lsd <= 4'b0000;

            if(maqs_Msd >= 3'b101)
            begin
                maqs_Msd <= 3'b000;
                maqs_addminuto <= 1'b0;
            end
            else
            begin
                maqs_Msd <= maqs_Msd + 1'b1;
                maqs_addminuto <= 1'b0;
            end
        end
        else if(maqs_Msd >= 3'b101)
        begin
            if(maqs_Lsd >= 4'b1000 && maqs_Lsd < 4'b1001)
            begin
                maqs_addminuto <= 1'b1;
            end
            else
            begin
                maqs_addminuto <= 0;
            end
            maqs_Lsd <= maqs_Lsd + 1;
        end
        else
            begin
                maqs_Lsd <= maqs_Lsd + 1;
                maqs_addminuto <= 0;
            end
    end
end

endmodule
```

Analogamente, o módulo dos minutos recebe como sinal de entrada a saída do módulo dos segundos para atualizar seu estado atual, e tambem quando atinge 58 minutos, “prepara” o sinal de saída para o envio aos 59 minutos para o módulo das horas maq_h e quando chega a 60 minutos, se reinicia para 0, repetindo o preocesso.

maq_m.sv

```
module maq_m(
    input maqm_clock,
    input maqm_reset,
    input maqm_enable,
    input maqm_incremento,
    output logic [3:0] maqm_Lsd,
    output logic [2:0] maqm_Msd,
    output logic maqm_incrementahora
);

logic temp_maqm_inc_hora;

always_ff @(posedge maqm_clock or negedge maqm_reset)
begin
    if(!maqm_reset)
    begin
        maqm_Lsd <= 4'b1000;
        maqm_Msd <= 3'b101;
        temp_maqm_inc_hora <= 1'b0;
    end
    else if(maqm_enable)
    begin
        if(maqm_incremento)
        begin
            if(maqm_Lsd >= 4'b1001)
            begin
                maqm_Lsd <= 4'b0000;

                if(maqm_Msd >= 3'b101)
                begin
                    maqm_Msd <= 3'b000;
                    temp_maqm_inc_hora <= 1'b0;
                end
                else
                begin
                    maqm_Msd <= maqm_Msd + 1'b1;
                    temp_maqm_inc_hora <= 1'b0;
                end
            end
            else if(maqm_Msd >= 3'b101)
            begin
                if(maqm_Lsd >= 4'b1000 && maqm_Lsd <= 4'b1001)
                begin
                    temp_maqm_inc_hora <= 1'b1;
                end
                else
                begin
```

```
                    temp_maqm_inc_hora <= 0;
                end
                maqm_Lsd <= maqm_Lsd + 1;
            end
            else
            begin
                maqm_Lsd <= maqm_Lsd + 1;
                temp_maqm_inc_hora <= 0;
            end
        end
        end
        else begin
            if(temp_maqm_inc_hora) begin
                maqm_incrementahora = temp_maqm_inc_hora & maqm_incremento;
            end
            else begin
                maqm_incrementahora = 1'b0;
            end
    end
end

endmodule
```

E então o último módulo, o maq_h, atualizará a hora quando o relógio chega os minutos chegam a 60, e incrementa a cada hora, como é o ultimo módulo, não passa nenhum sinal para outro bloco, apenas nos displays, e quando chega a 23 horas se “prepara” para reiniciar, e então é resetado para 0 quando chega ás 24 horas.

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

always_ff @(posedge maqh_clock or negedge maqh_reset) begin
    if(!maqh_reset)
    begin
        maqh_Lsd <= 4'b0000;
        maqh_Msd <= 2'b00;
    end
    else if(maqh_enable && maqh_incremento)
```

```
        begin
        if ((maqh_Msd >= 2'b10) && (maqh_Lsd >= 4'b0011)) begin
            maqh_Lsd <= 4'b0000;
            maqh_Msd <= 2'b00;
        end
        else if (maqh_Lsd >= 4'b1001) begin
            maqh_Lsd <= 0;
            maqh_Msd <= maqh_Msd + 1;
        end
        else begin
            maqh_Lsd <= maqh_Lsd + 1;
        end
    end
end

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 maquinas 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'b1111110;
        4'd1: bcd_display_out <= 7'b0110000;
        4'd2: bcd_display_out <= 7'b1101101;
        4'd3: bcd_display_out <= 7'b1111001;
        4'd4: bcd_display_out <= 7'b0110011;
        4'd5: bcd_display_out <= 7'b1011011;
        4'd6: bcd_display_out <= 7'b1011111;
        4'd7: bcd_display_out <= 7'b1110000;
        4'd8: bcd_display_out <= 7'b1111111;
        4'd9: bcd_display_out <= 7'b1111011;
        default: bcd_display_out <= 7'b0000000;
    endcase
endmodule
```

O módulo **topo.sv** é o bloco responsável por interligar todos os módulos do projeto do relógio digital. Ele atua como a camada de integração, recebendo o sinal de clock e o reset da FPGA e organizando a comunicação entre os contadores de segundos, minutos e horas, além de realizar a interface com os módulos de exibição em display de 7 segmentos.

As **entradas** do módulo são:

clock: sinal de relógio do sistema, proveniente da FPGA, utilizado como base para toda a contagem temporal.

reset: sinal assíncrono de reinicialização, responsável por zerar os contadores e reiniciar o funcionamento do relógio.

As **saídas** correspondem a seis vetores de 7 bits, cada um destinado ao acionamento de um display de 7 segmentos:

s_Lsd e s_Msd: representam respectivamente o dígito das unidades e das dezenas dos segundos.

m_Lsd e m_Msd: representam respectivamente o dígito das unidades e das dezenas dos minutos.

h_Lsd e h_Msd: representam respectivamente o dígito das unidades e das dezenas das horas.

O funcionamento pode ser descrito em três etapas principais:

Divisor de frequência (enable_1hz):
O sinal de clock original da FPGA (50 MHz) é dividido pelo módulo enable_1hz, gerando um pulso de 1 Hz que servirá como base de tempo para incrementar os segundos. Esse pulso (enable1hz) é distribuído para as máquinas de contagem.

Contadores de tempo (maq_s, maq_m, maq_h):

O módulo maq_s é responsável pela contagem de 0 a 59 segundos. Ao atingir o valor limite, além de reiniciar para 0, envia um pulso de incremento (incrementa_minuto) para o módulo de minutos.

O módulo maq_m recebe esse sinal e realiza a contagem dos minutos de 0 a 59. Da mesma forma, ao atingir o limite, reinicia para 0 e gera um pulso de incremento (incrementa_hora) para o módulo de horas.

O módulo maq_h é responsável pela contagem de 0 a 23 horas. Ao chegar ao limite, reinicia para 0, completando o ciclo de um dia.

Decodificação para display (display_7):
Os valores das unidades e dezenas de cada contador são representados em BCD. Para que possam ser exibidos nos displays de 7 segmentos, os sinais são enviados para módulos

display_7, que realizam a decodificação e produzem os sinais adequados para acender os LEDs de cada segmento.

Portanto, o **módulo topo integra todos os subsistemas do relógio digital**, garantindo que os pulsos do clock sejam corretamente convertidos em segundos, minutos e horas, e que esses valores sejam exibidos em tempo real nos seis displays de 7 segmentos da FPGA.

```
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(incrementa_hora),
.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(
```

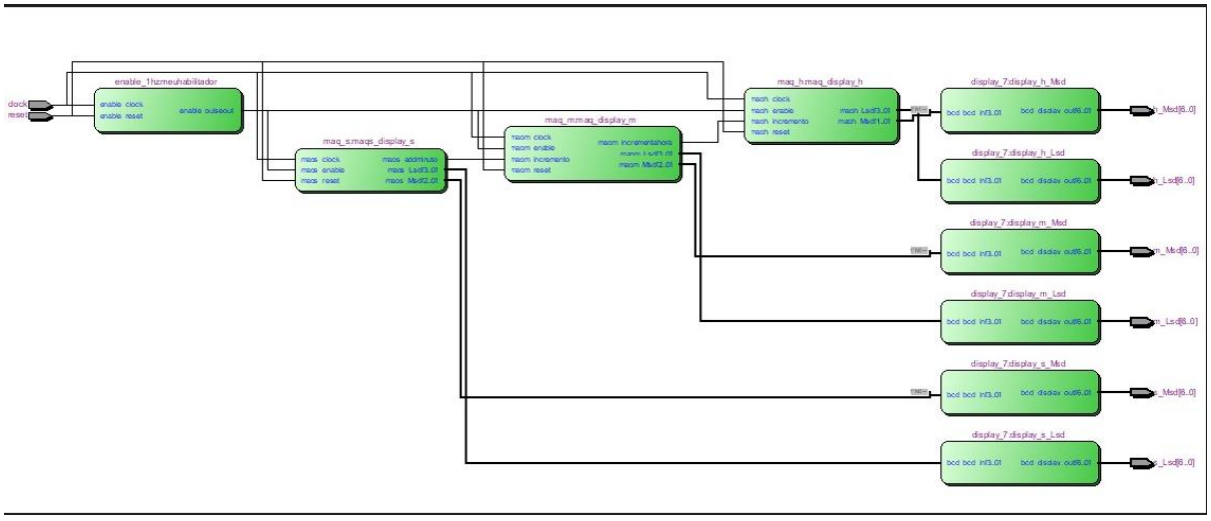
```
.bcd_bcd_in({2'd0,bcd_h_Msd}),
.bcd_display_out(h_Msd));

endmodule
```

3.4 SIMULAÇÃO:

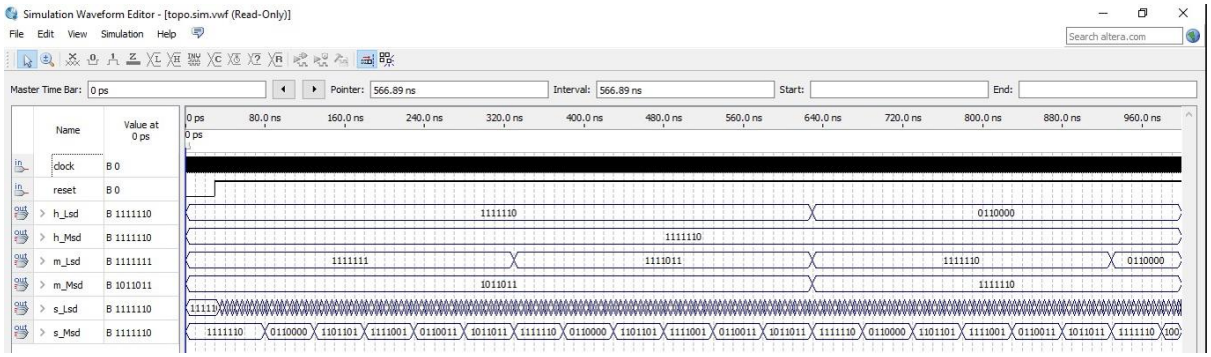
Nos teste utilizamos o software Quartus 2 para simular a topologia do circuito e as formas de onda dos sinais, com os estados atuais em cada máquina que representam as horas segundo e minutos:

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

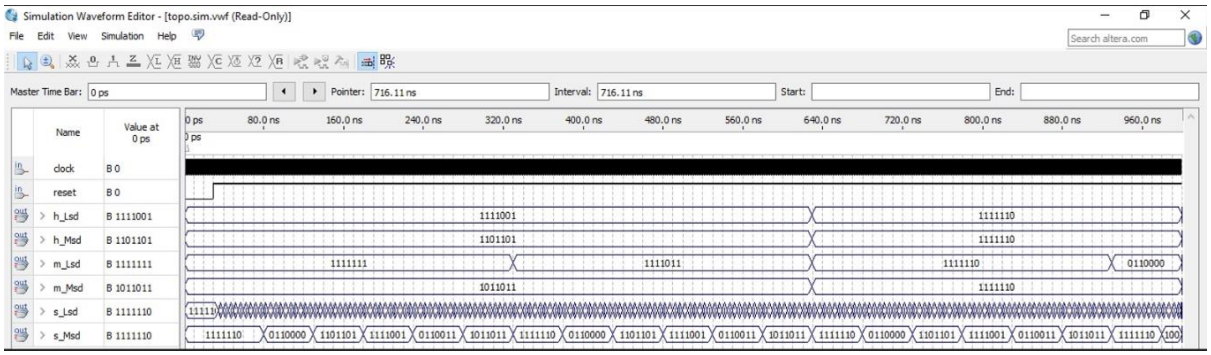


E como teste do funcionamento do codigo, simulamos graficamente as formas de onda do relógio:

Incrementa hora:



Incrementa crítico (24 horas):



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 ppode 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-