

**Universidade Federal da Paraíba
Centro de Informática
Departamento de Sistemas de Computação**

RELATÓRIO REFERENTE AO PROJETO DE RELÓGIO DIGITAL

Victor Cavalcante Santos Lima
Gabriel Henrique Cavalcante de Sousa
Cauan Amaro de Carvalho
Isac Santos Lira

Professora:
Verônica Maria Lima Silva

RESUMO

Este relatório descreve o projeto de um relógio digital desenvolvido em SystemVerilog, permitindo a simulação e validação do seu funcionamento. O relógio digital será capaz de exibir o tempo (horas, minutos e segundos) utilizando displays de 7 segmentos. O relógio recebe um sinal de clock de 50 MHz e divide esse sinal para operar com uma frequência de 1 Hz, permitindo a contagem e exibição do tempo de forma precisa. Por fim, são apresentados os resultados obtidos e as conclusões sobre o funcionamento do relógio digital.

Palavras-chave: Relógio Digital, Clock de 50MHZ, contadores síncronos, SystemVerilog, Display de 7 Segmentos.

Sumário

1 Introdução..	5
2 Objetivos	5
3 Metodologia	5
3.1 Materiais	5
3.2 Métodos	5
3.2.1 Programação(desenvolvimento do código)	5
4 Resultados	14
Conclusão.....	17

LISTA DE FIGURAS

1. RTL Viwer do clock.....	6
2. Bloco Clockdivider_1Hz.....	6
3. Bloco bcd_7segmentos.....	7
4. Bloco SecondCounter.....	8
5. Bloco MinuteCounter.....	9
6. Bloco HourCounter.....	10
7. Bloco Topo Clock.....	11
8. Continuação do bloco Clock.....	12
9. Continuação do bloco Clock.....	13

1 INTRODUÇÃO

O desenvolvimento de sistemas digitais é essencial para aplicações embarcadas, automação e eletrônica de consumo. Neste contexto, a linguagem de descrição de hardware (HDL) SystemVerilog permite modelar e simular circuitos digitais com alta precisão. Este relatório apresenta o projeto de um relógio digital cujo funcionamento se baseia na divisão de um sinal de clock de 50 MHz para gerar pulsos de 1 Hz e no uso de contadores síncronos para controlar a progressão dos segundos, minutos e horas, utilizando módulos distintos para divisão de frequência e contagem de tempo, com saídas codificadas em BCD (Binary-Coded Decimal) adequadas para exibição em displays de sete segmentos.

2 OBJETIVOS

2.1 OBJETIVOS GERAIS

Desenvolver e implementar em SystemVerilog um Relógio Digital, aplicando os conceitos estudados na disciplina de Circuitos Lógicos II, capaz de realizar a contagem correta de horas, minutos e segundos de forma sincronizada e compatível com displays de 7 segmentos.

2.2 OBJETIVOS ESPECÍFICOS

1. Implementar contadores independentes para segundos, minutos e horas, respeitando os limites de 59 segundos, 59 minutos e 23 horas.
2. Garantir a comunicação entre os contadores por meio de sinais de incremento (carry).
3. Codificar os valores de tempo em formato BCD (Binary-Coded Decimal), facilitando a integração com módulos de exibição.
4. Validar o funcionamento correto do sistema por meio de simulação e verificação dos resultados esperados.

3 METODOLOGIA

3.1 Materiais

Para o desenvolvimento do projeto foi necessário um computador que possuísse o software adequado mais especificamente o software para programação de dispositivo lógico Altera Quartus II na versão 13.0sp1, para o desenvolvimento do código, na linguagem de programação para hardware SystemVerilog.

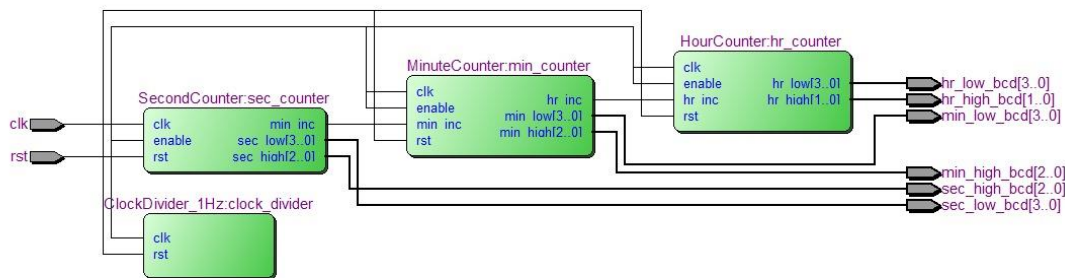
3.2 Métodos

A metodologia utilizada no desenvolvimento do projeto consistiu na programação do código no Quartus.

3.2.1 Programação (Desenvolvimento do Código)

O código em SystemVerilog foi desenvolvido de acordo com as necessidades do projeto para formação do Relógio Digital, tanto para teste quanto para a apresentação em sala de aula. O código foi desenvolvido no Quartus II, que forneceu suporte completo para escrita, compilação e simulação do Relógio. A construção do relógio digital foi realizada de forma modular. A metodologia adotada consistiu em dividir o projeto em partes independentes, com funções específicas e bem definidas. Na figura 1 pode-se ver o RTL Viwer do relógio.

Figura 1 – RTL VIWER



Fonte: Elaborado pelo autor (2025)

O primeiro passo foi desenvolver um divisor de clock, responsável por reduzir um sinal de 50 MHz para 1 Hz, ou seja, um pulso por segundo. Isso foi feito utilizando um contador de 26 bits que gera um pulso lógico sempre que atinge o valor máximo pré-definido.

Figura 2 – Bloco Clockdivider_1Hz

```

1 module ClockDivider_1Hz (
2     input clk,
3     input rst,
4     output logic pulse_out
5 );
6
7     logic [25:0] counter;
8
9     always_ff @(posedge clk) begin
10         if (!rst) begin
11             counter <= 26'd0; // Reseta o contador se o reset estiver ativo
12         end
13         else if (counter == 26'd49999999) begin
14             counter <= 26'd0; // Reinicia o contador ao atingir o valor máximo
15         end
16         else begin
17             counter <= counter + 26'd1; // Incrementa o contador
18         end
19     end
20
21     always_comb begin
22         pulse_out = (counter == 26'd49999999); // Gera um pulso quando o contador atinge o valor máximo
23     end
24
25 endmodule

```

Fonte: Elaborado pelo autor (2025)

De forma simples, o bloco é sensível à borda de subida do clock e sempre que o contador atinge o valor de 49.999.999 ciclos, ele é zerado, com isso o pulse_out é ativo, o equivalente a uma vez por segundo

Já o bloco bcd_7seg converte os números para um formato adequado para o display de 7 segmentos, como pode ser visto na figura:

Figura 3 – Bloco cbd_7segmentos

```

1 module cbd_7segmentos (
2     input [3:0] bcd_input,
3     output logic [6:0] seg_output
4 );
5
6 always_comb begin
7     case (bcd_input)
8         4'd0: seg_output = 7'b1111110; // 0
9         4'd1: seg_output = 7'b0110000; // 1
10        4'd2: seg_output = 7'b1101101; // 2
11        4'd3: seg_output = 7'b1111001; // 3
12        4'd4: seg_output = 7'b0110011; // 4
13        4'd5: seg_output = 7'b1011011; // 5
14        4'd6: seg_output = 7'b1011111; // 6
15        4'd7: seg_output = 7'b1110000; // 7
16        4'd8: seg_output = 7'b1111111; // 8
17        4'd9: seg_output = 7'b1111011; // 9
18        default: seg_output = 7'b0000000; // Apaga o display para valores inválidos
19    endcase
20 end
21
22 endmodule
  
```

Fonte: Elaborado pelo autor (2025)

Dependendo da entrada recebida, o código codifica para a forma de como o display de 7 segmentos atua. Sendo cada bit um segmento dentre os 7, garantindo que saia o número esperado.

Em seguida, implementou-se o módulo SecondCounter, responsável por contar os segundos de 00 a 59. O módulo foi programado para emitir um sinal min_inc ao atingir 59 segundos, sinalizando o incremento do contador de minutos. O valor é representado em BCD por dois vetores: sec_low (unidades) e sec_high (dezenas).

Figura 4 – Bloco SecondCounter

```

1 module SecondCounter(
2     input rst,
3     input clk,
4     input enable,
5     output logic [3:0] sec_low,
6     output logic [2:0] sec_high,
7     output logic min_inc
8 );
9
10 always_ff @(posedge clk or negedge rst) begin
11     if (!rst) begin
12         sec_low    <= 4'd0; // Reseta o dígito das unidades dos segundos
13         sec_high   <= 3'd0; // Reseta o dígito das dezenas dos segundos
14         min_inc    <= 1'd0; // Reseta o sinal de carry para minutos
15     end
16     else if (enable) begin
17         if (sec_high == 3'd5 && sec_low == 4'd8) begin
18             // Prepara o carry para minutos no próximo incremento
19             sec_low    <= sec_low + 1'd1;
20             min_inc    <= 1'd1; // Gera um carry para incrementar os minutos
21         end
22         else if (sec_high == 3'd5 && sec_low == 4'd9) begin
23             // Reinicia o contador quando atinge 59 segundos
24             sec_low    <= 4'd0;
25             sec_high   <= 3'd0;
26             min_inc    <= 1'd0; // Zera o carry após o reinício
27         end
28         else if (sec_low == 4'd9) begin
29             // Incrementa as dezenas dos segundos quando as unidades atingem 9
30             sec_low    <= 4'd0;
31             sec_high   <= sec_high + 1'd1;
32             min_inc    <= 1'd0; // Não há carry para minutos
33         end
34         else begin
35             // Incrementa as unidades dos segundos
36             sec_low    <= sec_low + 1'd1;
37             min_inc    <= 1'd0; // Não há carry para minutos
38         end
39     end
40 end
41
42 endmodule

```

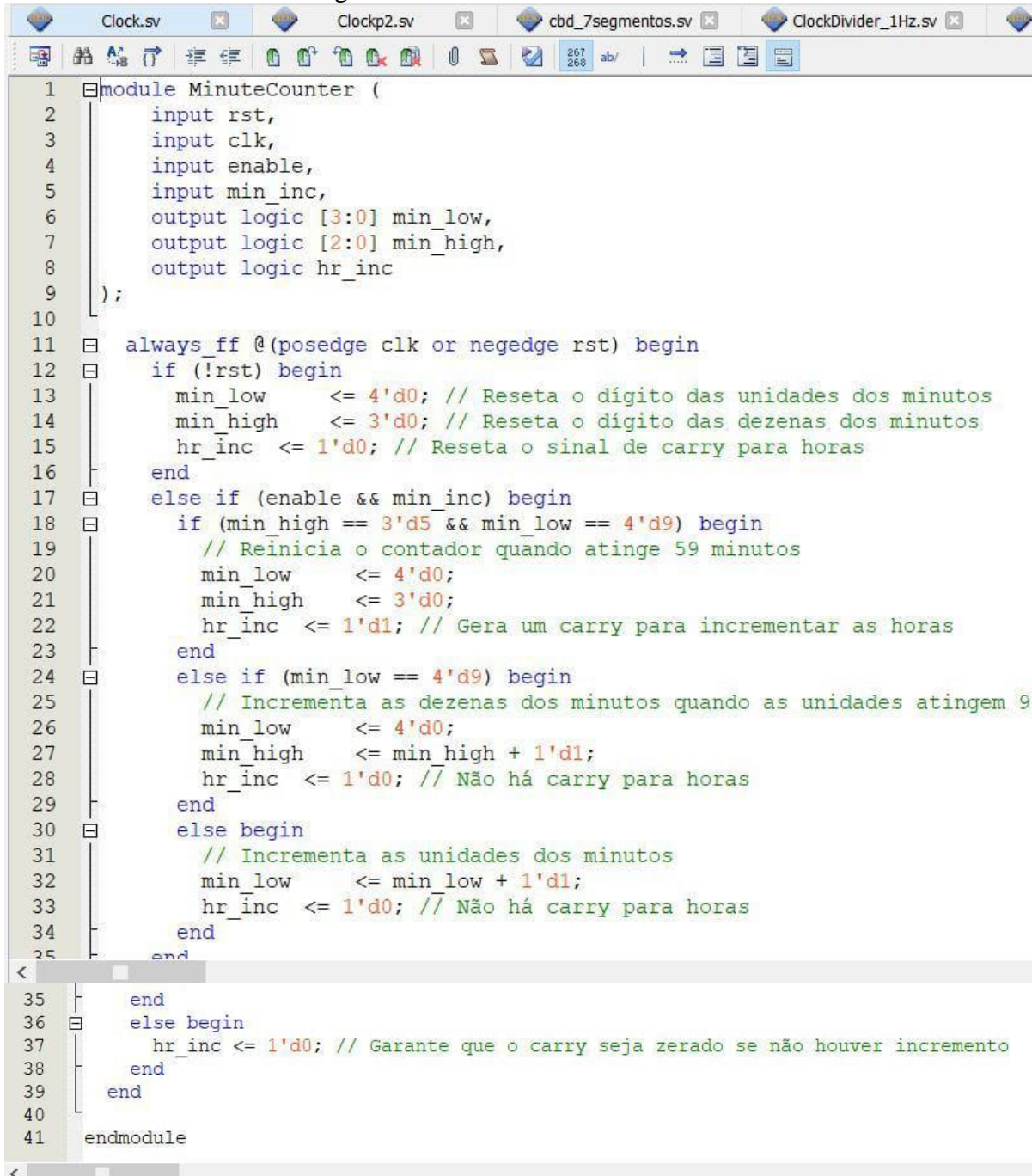
Fonte: Elaborado pelo autor (2025)

Na linha inicial, ocorre a definição do módulo do bloco e as respectivas entradas e saídas. A partir de então, entra-se no loop de funcionamento do sistema, em que é ativo para cada ciclo positivo do relógio. Nele, é feita a manipulação dos segundos caso o enable esteja ativo, se não, ele é reiniciado. Durante o funcionamento normal (isto é, quando o sinal enable está em nível alto e o sistema não está em reset), o contador é incrementado a cada borda de subida do sinal de clock. A contagem se dá em dois dígitos: sec_low é incrementado de 0 a 9, e ao atingir o valor máximo (9), é reiniciado para 0, enquanto sec_high é incrementado em uma unidade. Quando sec_high atinge o valor 5 e sec_low atinge 9 (ou seja, 59 segundos), o contador retorna para 0 (reinicializa) e o sinal min_inc é ativado por um ciclo de clock,

indicando que um minuto completo foi contado.

Após a contagem dos segundos, foi desenvolvido o módulo MinuteCounter, que realiza a contagem de 00 a 59 minutos. Ele é habilitado por meio do sinal min_inc, vindo do módulo de segundos. Quando o valor 59 é atingido, o módulo gera o sinal hr_inc para o incremento da hora.

Figura 5 – Bloco MinuteCounter



```
1 module MinuteCounter (
2     input rst,
3     input clk,
4     input enable,
5     input min_inc,
6     output logic [3:0] min_low,
7     output logic [2:0] min_high,
8     output logic hr_inc
9 );
10
11 always_ff @(posedge clk or negedge rst) begin
12     if (!rst) begin
13         min_low    <= 4'd0; // Reseta o dígito das unidades dos minutos
14         min_high   <= 3'd0; // Reseta o dígito das dezenas dos minutos
15         hr_inc    <= 1'd0; // Reseta o sinal de carry para horas
16     end
17     else if (enable && min_inc) begin
18         if (min_high == 3'd5 && min_low == 4'd9) begin
19             // Reinicia o contador quando atinge 59 minutos
20             min_low    <= 4'd0;
21             min_high   <= 3'd0;
22             hr_inc    <= 1'd1; // Gera um carry para incrementar as horas
23         end
24         else if (min_low == 4'd9) begin
25             // Incrementa as dezenas dos minutos quando as unidades atingem 9
26             min_low    <= 4'd0;
27             min_high   <= min_high + 1'd1;
28             hr_inc    <= 1'd0; // Não há carry para horas
29         end
30         else begin
31             // Incrementa as unidades dos minutos
32             min_low    <= min_low + 1'd1;
33             hr_inc    <= 1'd0; // Não há carry para horas
34         end
35     end
36     else begin
37         hr_inc <= 1'd0; // Garante que o carry seja zerado se não houver incremento
38     end
39 end
40
41 endmodule
```

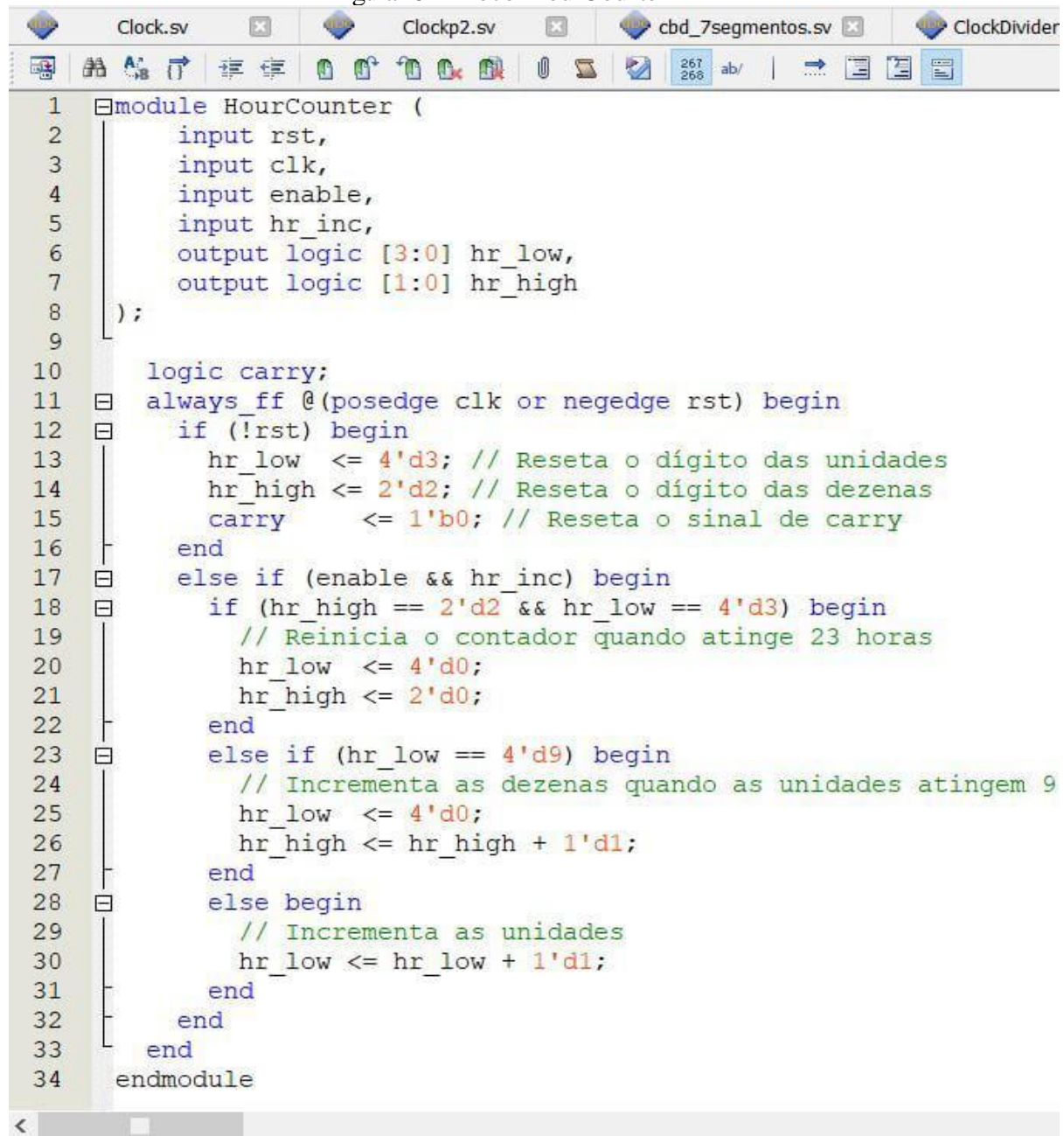
Fonte: Elaborado pelo autor (2025)

Assim como nos segundos na linha inicial, ocorre a definição do módulo do bloco e as respectivas entradas e saídas. A partir de então, entra-se no loop de funcionamento do sistema, em que é ativo para cada ciclo positivo do relógio. Nele, é feita a manipulação dos minutos, caso o enable esteja ativo, se não, ele é reiniciado. A lógica principal está descrita

em um bloco `always_ff`, sensível à borda de subida do sinal de clock e à borda de descida do sinal de reset. Quando o sistema está em estado de reset (`rst = 0`), os sinais `min_low`, `min_high` e `hr_inc` são imediatamente atribuídos ao valor zero. Isso garante a reinicialização do contador, independentemente do estado anterior.

Por fim, o último bloco de operação é o módulo `HourCounter` que realiza a contagem das horas no formato de 24 horas, indo de 00 a 23. A estrutura de contagem é semelhante aos módulos anteriores, utilizando BCD para separar unidades (`hr_low`) e dezenas (`hr_high`). O contador reinicia automaticamente ao passar das 23 horas.

Figura 6 – Bloco HourCounter



```

1  module HourCounter (
2      input rst,
3      input clk,
4      input enable,
5      input hr_inc,
6      output logic [3:0] hr_low,
7      output logic [1:0] hr_high
8  );
9
10     logic carry;
11     always_ff @(posedge clk or negedge rst) begin
12         if (!rst) begin
13             hr_low <= 4'd3; // Reseta o dígito das unidades
14             hr_high <= 2'd2; // Reseta o dígito das dezenas
15             carry <= 1'b0; // Reseta o sinal de carry
16         end
17         else if (enable && hr_inc) begin
18             if (hr_high == 2'd2 && hr_low == 4'd3) begin
19                 // Reinicia o contador quando atinge 23 horas
20                 hr_low <= 4'd0;
21                 hr_high <= 2'd0;
22             end
23             else if (hr_low == 4'd9) begin
24                 // Incrementa as dezenas quando as unidades atingem 9
25                 hr_low <= 4'd0;
26                 hr_high <= hr_high + 1'd1;
27             end
28             else begin
29                 // Incrementa as unidades
30                 hr_low <= hr_low + 1'd1;
31             end
32         end
33     end
34 endmodule

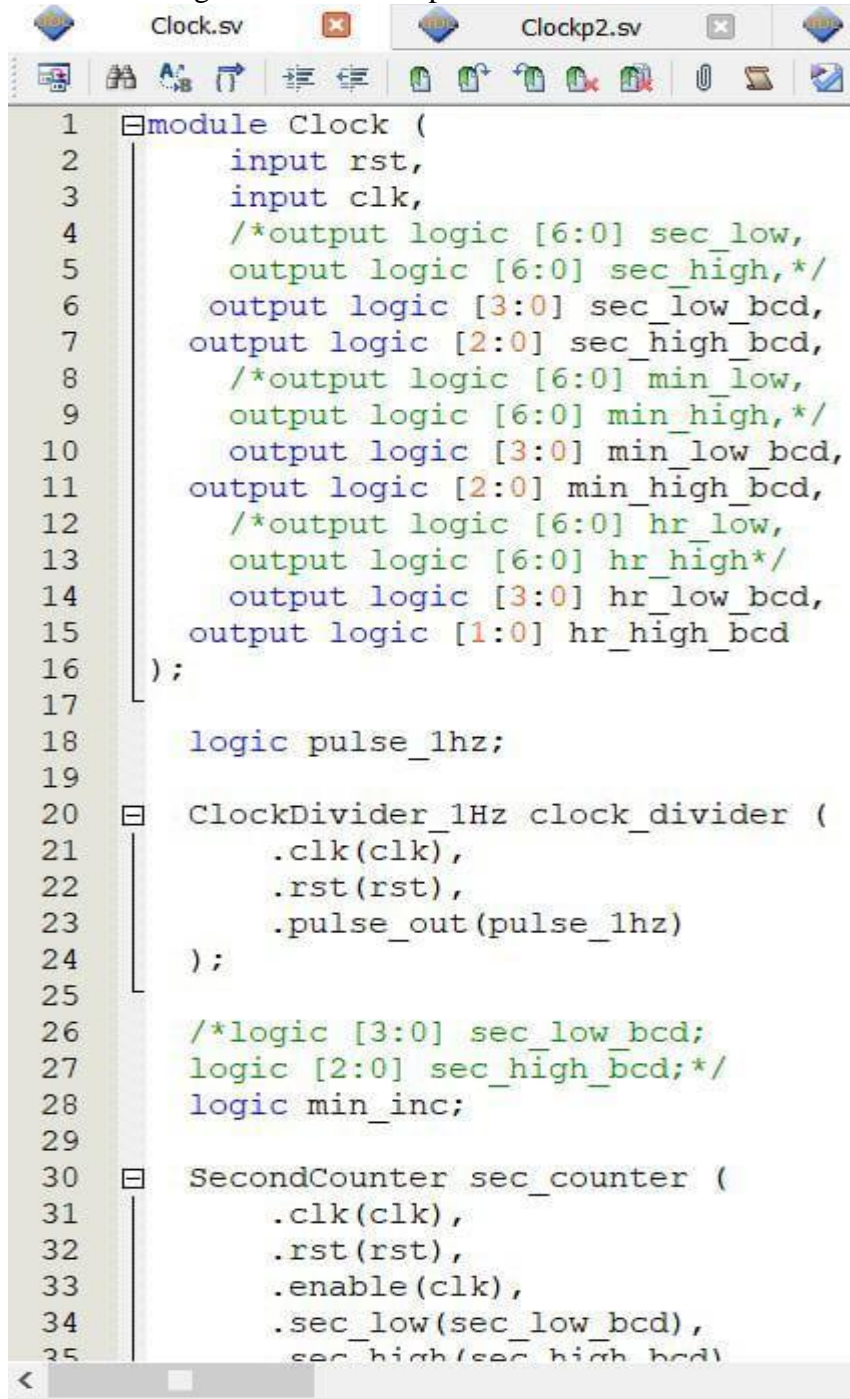
```

Fonte: Elaborado pelo autor (2025)

Novamente na linha inicial, ocorre a definição do módulo do bloco e as respectivas entradas e saídas. A partir de então, entra-se no loop de funcionamento do sistema, em que é ativo para cada ciclo positivo do relógio. Nele, é feita a manipulação das horas, caso o enable esteja ativo, se não, ele é reiniciado.

Por fim, o módulo principal Clock foi responsável por instanciar e conectar todos os módulos anteriores, garantindo que os sinais de controle fossem corretamente propagados e que os dados estivessem sincronizados. Cada valor de tempo é representado em formato BCD e pode ser facilmente convertido para exibição em displays de sete segmentos.

Figura 7 – Bloco Topo Clock



```

1 module Clock (
2     input rst,
3     input clk,
4     /*output logic [6:0] sec_low,
5     output logic [6:0] sec_high,*/
6     output logic [3:0] sec_low_bcd,
7     output logic [2:0] sec_high_bcd,
8     /*output logic [6:0] min_low,
9     output logic [6:0] min_high,*/
10    output logic [3:0] min_low_bcd,
11    output logic [2:0] min_high_bcd,
12    /*output logic [6:0] hr_low,
13    output logic [6:0] hr_high*/
14    output logic [3:0] hr_low_bcd,
15    output logic [1:0] hr_high_bcd
16 );
17
18 logic pulse_1hz;
19
20 ClockDivider_1Hz clock_divider (
21     .clk(clk),
22     .rst(rst),
23     .pulse_out(pulse_1hz)
24 );
25
26 /*logic [3:0] sec_low_bcd;
27 logic [2:0] sec_high_bcd;*/
28 logic min_inc;
29
30 SecondCounter sec_counter (
31     .clk(clk),
32     .rst(rst),
33     .enable(clk),
34     .sec_low(sec_low_bcd),
35     .sec_high(sec_high_bcd)

```

Fonte: Elaborado pelo autor (2025)

Figura 8 – Continuação do bloco Clock

```
35         .sec_high(sec_high_bcd),
36         .min_inc(min_inc)
37     );
38
39     /* cbd_7segmentos sec_low_display (
40         .bcd_input(sec_low_bcd),
41         .seg_output(sec_low)
42     );
43
44     cbd_7segmentos sec_high_display (
45         .bcd_input({1'b0, sec_high_bcd}),
46         .seg_output(sec_high)
47     );
48 */
49     //logic [3:0] min_low_bcd;
50     //logic [2:0] min_high_bcd;
51     logic hr_inc;
52
53     MinuteCounter min_counter (
54         .clk(clk),
55         .rst(rst),
56         .enable(clk),
57         .min_inc(min_inc),
58         .min_low(min_low_bcd),
59         .min_high(min_high_bcd),
60         .hr_inc(hr_inc)
61     );
```

Fonte: Elaborado pelo autor (2025)

Figura 9 – Continuação do bloco Clock

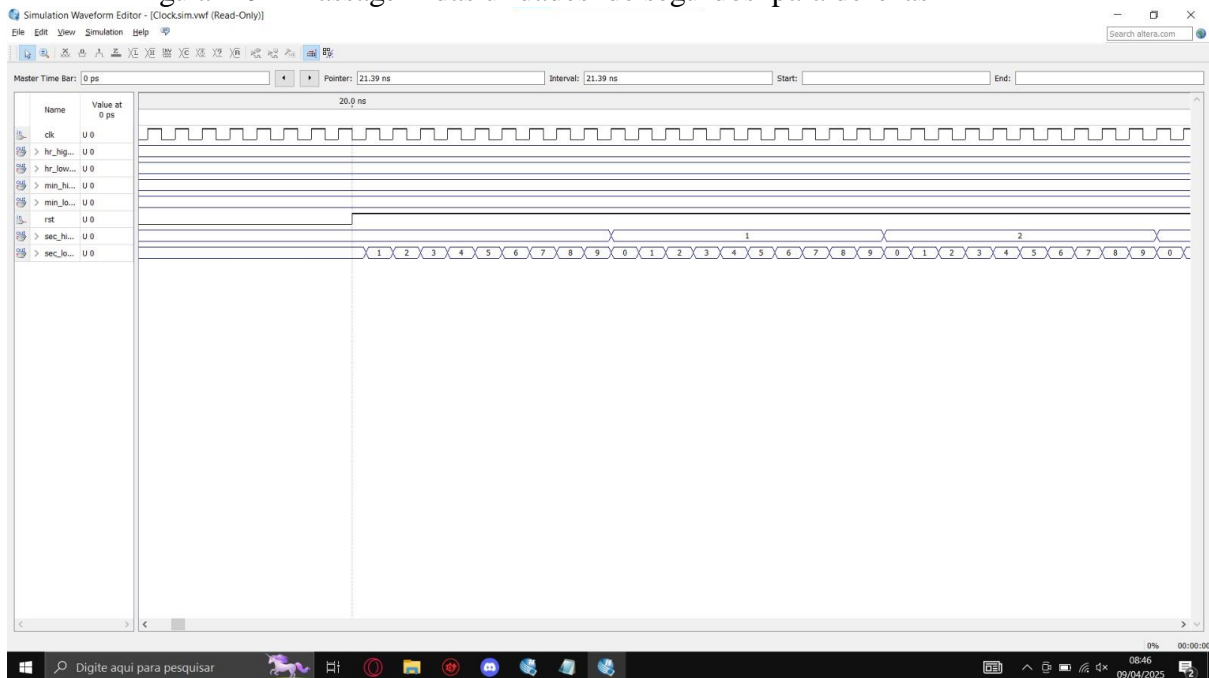
```
62 L
63     /*cbd_7segmentos min_low_display (
64         .bcd_input(min_low_bcd),
65         .seg_output(min_low)
66     );
67
68     cbd_7segmentos min_high_display (
69         .bcd_input({1'b0, min_high_bcd}),
70         .seg_output(min_high)
71     );
72 */
73 // logic [3:0] hr_low_bcd;
74 //logic [1:0] hr_high_bcd;
75
76 □ HourCounter hr_counter (
77     .clk(clk),
78     .rst(rst),
79     .enable(clk),
80     .hr_inc(hr_inc),
81     .hr_low(hr_low_bcd),
82     .hr_high(hr_high_bcd)
83 );
84
85 /* cbd_7segmentos hr_low_display (
86     .bcd_input(hr_low_bcd),
87     .seg_output(hr_low)
88 );
89
90     cbd_7segmentos hr_high_display (
91         .bcd_input({2'd0, hr_high_bcd}),
92         .seg_output(hr_high)
93     );*/
94
95 endmodule
```

Fonte: Elaborado pelo autor (2025)

4 RESULTADOS

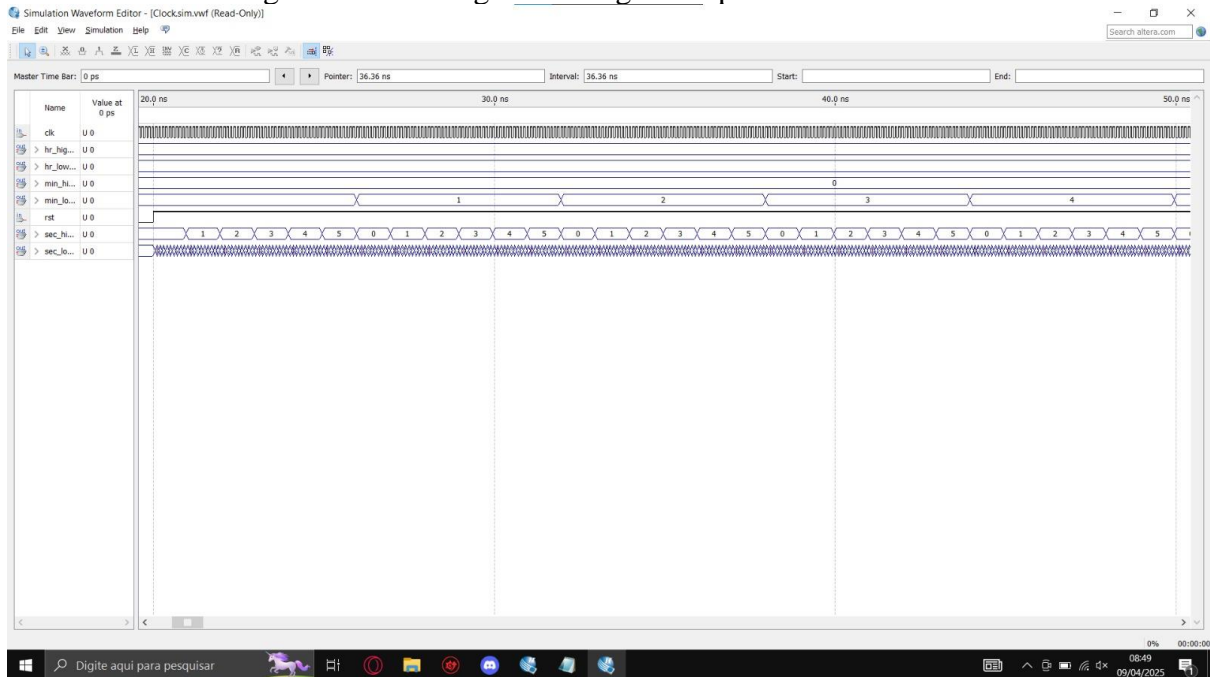
Após a implementação dos módulos em SystemVerilog e realização das simulações, foi possível verificar o funcionamento correto do relógio digital proposto. A divisão do projeto em módulos independentes divisor de frequência, contador de segundos, contador de minutos e contador de horas permitiu uma melhor organização do sistema e facilitou o processo de verificação. O divisor de frequência foi capaz de gerar um pulso com período aproximado de um segundo a partir de um sinal de entrada com frequência de 50 MHz. Esse pulso foi utilizado para habilitar os contadores de tempo, simulando o avanço real dos segundos. O contador de segundos e o de minutos apresentou funcionamento conforme esperado, realizando a contagem de 0 a 59 e gerando um sinal de incremento ao final do ciclo, sinalizando o avanço dos minutos e das horas quando necessário e o contador de horas foi projetado para operar dentro do intervalo de 0 a 23, reiniciando automaticamente ao atingir o valor de 24 horas. Todas as saídas foram representadas em formato BCD (Binary-Coded Decimal), garantindo compatibilidade com displays de sete segmentos. Nas figuras abaixo pode-se ver o funcionamento do relógio.

Figura 10 – Passagem das unidades de segundos para dezenas



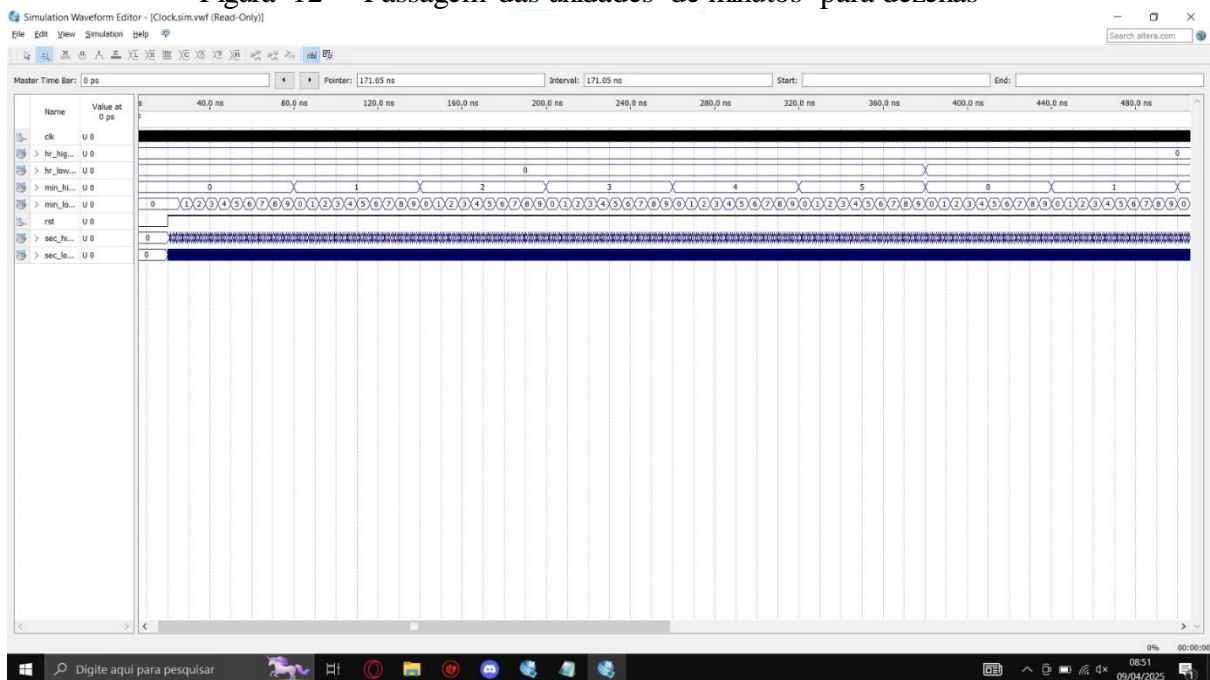
Fonte: Elaborado pelo autor (2025)

Figura 11 – Passagem dos segundos para minutos



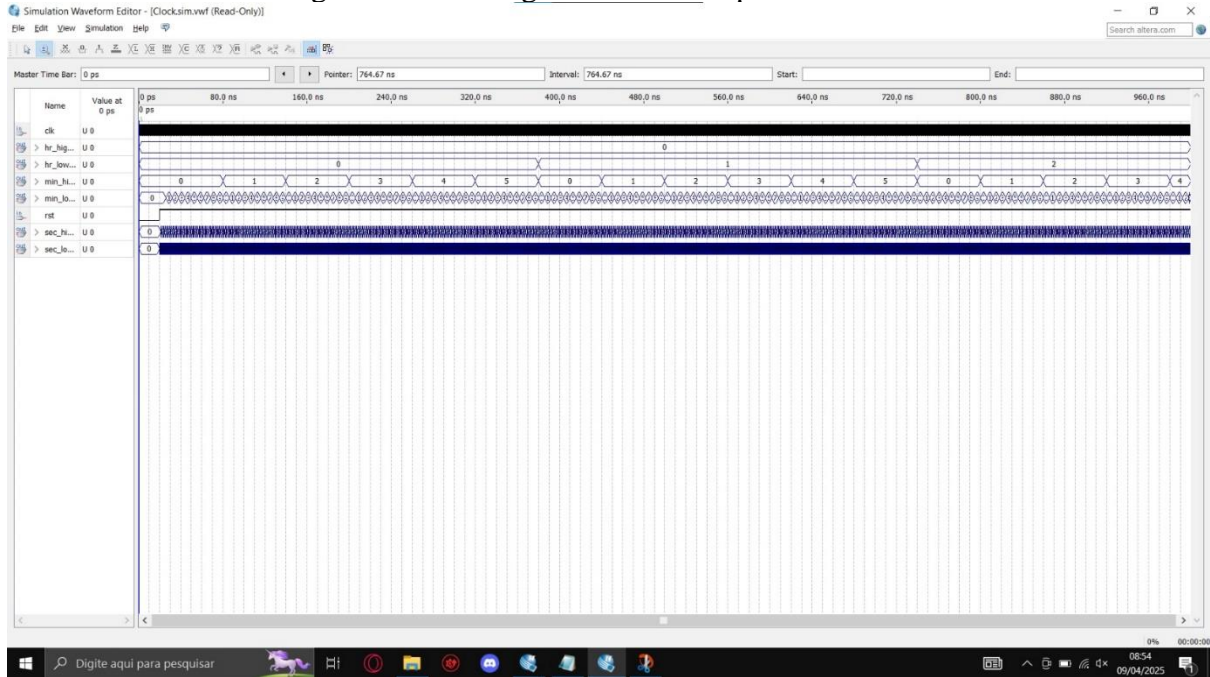
Fonte: Elaborado pelo autor (2025)

Figura 12 – Passagem das unidades de minutos para dezenas



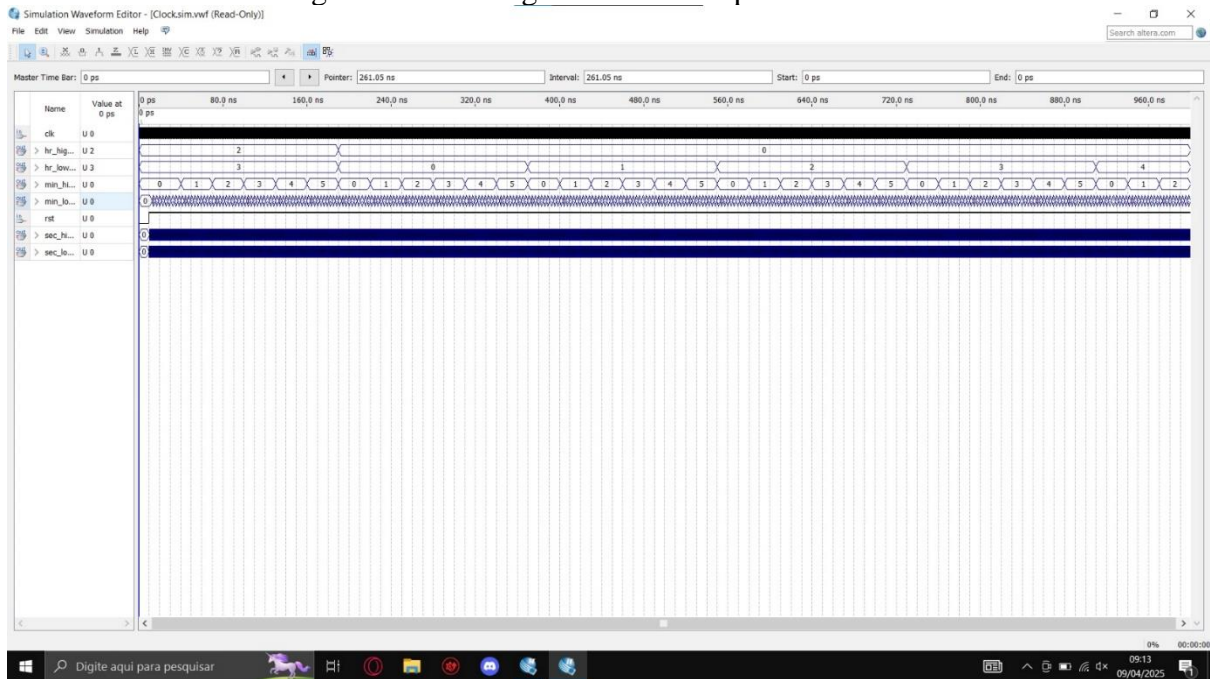
Fonte: Elaborado pelo autor (2025)

Figura 13 – Passagem de minutos para horas



Fonte: Elaborado pelo autor (2025)

Figura 14 – Passagem de 23 horas para meia noite



Fonte: Elaborado pelo autor (2025)

5 CONCLUSÃO

O projeto do relógio digital em SystemVerilog foi concluído com sucesso, demonstrando a eficácia do uso de HDL para a construção de sistemas digitais temporizados. A divisão em módulos claros e reutilizáveis facilita a manutenção e a expansão do projeto, como a inclusão de displays ou alarmes. O uso de BCD torna a interface com hardware de exibição simples e direta. O relógio digital atingiu todos os objetivos propostos, sendo capaz de exibir corretamente as horas, minutos e segundos em displays de 7 segmentos, com base em um sinal de clock de 1 Hz gerado a partir de uma frequência de entrada de 50 MHz. A estabilidade do sistema foi comprovada através de testes, nos quais o relógio manteve a precisão na contagem do tempo, sem falhas ou perda de sincronia. Em resumo, este trabalho proporcionou uma experiência prática valiosa em design de programação em SystemVerilog. O relógio digital projetado, cumpre com êxito seus objetivos e pode servir como base para sistemas que envolvem temporização ou controle baseado em tempo.