UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

CONTADOR DIGITAL BCD

Entrega 3

SEL0628 – Sistemas Digitais

Prof. Dr. Maximiliam Luppe

Daniel Dias Silva Filho - 13677114

Daniel Umeda Kuhn - 13676541

Francyélio - 13676537

João Marcelo Ferreira Battaglini - 13835472

Manoel Thomaz - 13676392

Lucas Sales Duarte - 11734490

São Carlos – SP

10/07/2024

Daniel Dias Silva Filho - 13677114
Daniel Umeda Kuhn - 13676541
Francyélio - 13676537
João Marcelo Ferreira Battaglini - 13835472
Manoel Thomaz - 13676392
Lucas Sales Duarte - 11734490

3° Entrega

3° parte do trabalho de recuperação da disciplina SEL 0628
Professor: Maximiliam Luppe

São Carlos – SP

10/07/2024

Sumário

1. Introdução

Este relatório apresenta a implementação de um contador digital BCD que conta de 000 a 999, utilizando a linguagem de descrição de hardware Verilog. A escolha de Verilog permite uma descrição precisa e modular do circuito, facilitando a síntese em FPGA (Field-Programmable Gate Array) ou ASIC (Application-Specific Integrated Circuit).

A implementação proposta é dividida em componentes funcionais, incluindo contadores de unidades, dezenas e centenas, flip-flops para armazenamento dos estados intermediários e módulos para a conversão dos valores binários para a representação em displays de 7 segmentos. Cada componente é descrito em detalhes, destacando seu papel no funcionamento do contador como um todo.

O uso de contadores BCD é particularmente vantajoso em aplicações onde a saída precisa ser exibida em formato decimal, como em sistemas de medição, temporizadores e instrumentos de teste. A implementação descrita neste relatório garante precisão e eficiência, utilizando técnicas de contagem modular e sincronização de clock para assegurar que os valores sejam corretamente incrementados e exibidos.

Além da descrição detalhada dos componentes e do funcionamento do contador, este relatório também inclui uma análise do circuito RTL (Register Transfer Level), gerado a partir do código Verilog. A visualização do circuito RTL oferece uma compreensão clara de como os componentes interagem no nível de hardware, proporcionando uma visão completa do design e suas funcionalidades.

Este trabalho não só demonstra a implementação prática de um contador BCD, mas também serve como um exemplo educacional de como projetos digitais podem ser desenvolvidos e testados usando ferramentas modernas de design e simulação. A documentação completa e a análise do circuito reforçam o entendimento teórico e prático, contribuindo para a formação de profissionais capacitados na área de eletrônica digital e design de hardware.

2. Código

2.1. Cabeçalho do código

```
module bcd_counter_3digit (
    input wire clk,          // Sinal de clock
    input wire rst_s,        // Sinal de reset síncrono
    input wire ld,           // Sinal de load (não usado neste código)
    input wire [3:0] enb,    // Vetor de sinais de enable para os contadores
```

output wire [6:0] bcd_sgm_0, // Saída para display de 7 segmentos do dígito menos significativo

output wire [6:0] bcd_sgm_1, // Saída para display de 7 segmentos do dígito do meio

output wire [6:0] bcd_sgm_2 // Saída para display de 7 segmentos do dígito mais significativo

);

// Declaração dos fios intermediários

wire [3:0] q0, q1, q2; // Sinais dos contadores

wire [3:0] d0, d1, d2; // Sinais dos flip-flops

// Instanciação dos contadores e flip-flops

cnt_9 U0 (.clk(clk), .rst_s(rst_s), .enb(enb[0]), .q(q0)); // Contador de 0 a 9 para o dígito menos significativo

cnt_9 U1 (.clk(clk), .rst_s(rst_s), .enb(enb[1]), .q(q1)); // Contador de 0 a 9 para o dígito do meio

cnt_9 U2 (.clk(clk), .rst_s(rst_s), .enb(enb[2]), .q(q2)); // Contador de 0 a 9 para o dígito mais significativo

dff D0 (.clk(clk), .d(q0), .q(d0)); // Flip-flop para armazenar o valor do contador menos significativo

dff D1 (.clk(clk), .d(q1), .q(d1)); // Flip-flop para armazenar o valor do contador do meio

dff D2 (.clk(clk), .d(q2), .q(d2)); // Flip-flop para armazenar o valor do contador mais significativo

// Instanciação dos displays BCD

bcd_sgm BCD0 (.bcd(d0), .sgm(bcd_sgm_0)); // Conversor BCD para display de 7 segmentos (dígito menos significativo)

bcd_sgm BCD1 (.bcd(d1), .sgm(bcd_sgm_1)); // Conversor BCD para display de 7 segmentos (dígito do meio)

bcd_sgm BCD2 (.bcd(d2), .sgm(bcd_sgm_2)); // Conversor BCD para display de 7 segmentos (dígito mais significativo)

```
endmodule
```

2.2. Implementação dos módulos utilizados

2.2.1 Contador de 0 a 9 (cnt_9)

```
module cnt_9 (
    input wire clk,       // Sinal de clock
    input wire rst_s,     // Sinal de reset síncrono
    input wire enb,       // Sinal de enable
    output reg [3:0] q    // Saída do contador (4 bits para contar de 0 a 9)
);

always @(posedge clk or posedge rst_s) begin
    if (rst_s)              // Se o sinal de reset for alto
        q <= 4'b0000;       // Reseta o contador para 0
    else if (enb) begin     // Se o enable estiver ativo
        if (q == 4'b1001)   // Se o contador atingir 9
            q <= 4'b0000;   // Reseta o contador para 0
        else
            q <= q + 1;     // Incrementa o contador
    end
end

endmodule
```

2.2.2. Flip-Flop (dff)

```
module dff (
    input wire clk,       // Sinal de clock
    input wire [3:0] d,   // Dado de entrada (4 bits)
```

```
    output reg [3:0] q    // Saída do flip-flop (4 bits)
);

always @(posedge clk) begin
    q <= d;  // Atribui o valor de entrada 'd' à saída 'q' na borda de subida do clock
end

endmodule
```

2.2.3. Conversor para display de 7 segmentos (bcd_sgm)

```
module bcd_sgm (
    input wire [3:0] bcd,  // Entrada BCD (4 bits)
    output reg [6:0] sgm   // Saída para display de 7 segmentos (7 bits)
);

always @(*) begin
    case (bcd)
        4'b0000: sgm = 7'b0111111; // 0
        4'b0001: sgm = 7'b0000110; // 1
        4'b0010: sgm = 7'b1011011; // 2
        4'b0011: sgm = 7'b1001111; // 3
        4'b0100: sgm = 7'b1100110; // 4
        4'b0101: sgm = 7'b1101101; // 5
        4'b0110: sgm = 7'b1111101; // 6
        4'b0111: sgm = 7'b0000111; // 7
        4'b1000: sgm = 7'b1111111; // 8
        4'b1001: sgm = 7'b1101111; // 9
        default: sgm = 7'b0000000; // Caso padrão: display em branco
    endcase
```

end

endmodule

3. Circuito RTL

Ao implementar o código, foi gerado o seguinte circuito RTL (Figura 1) e a ampliação com o Technology Map Viewer (Figura 2):

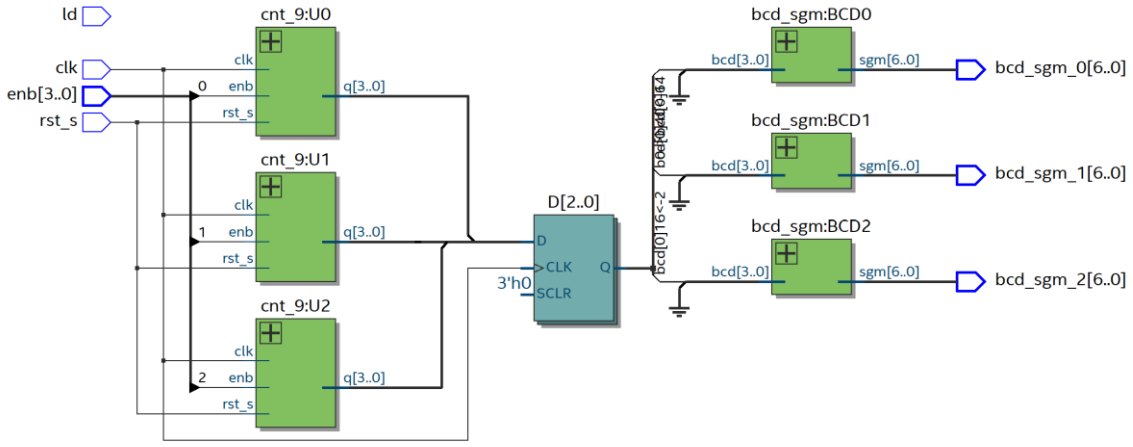

Figura 1: Circuito RTL

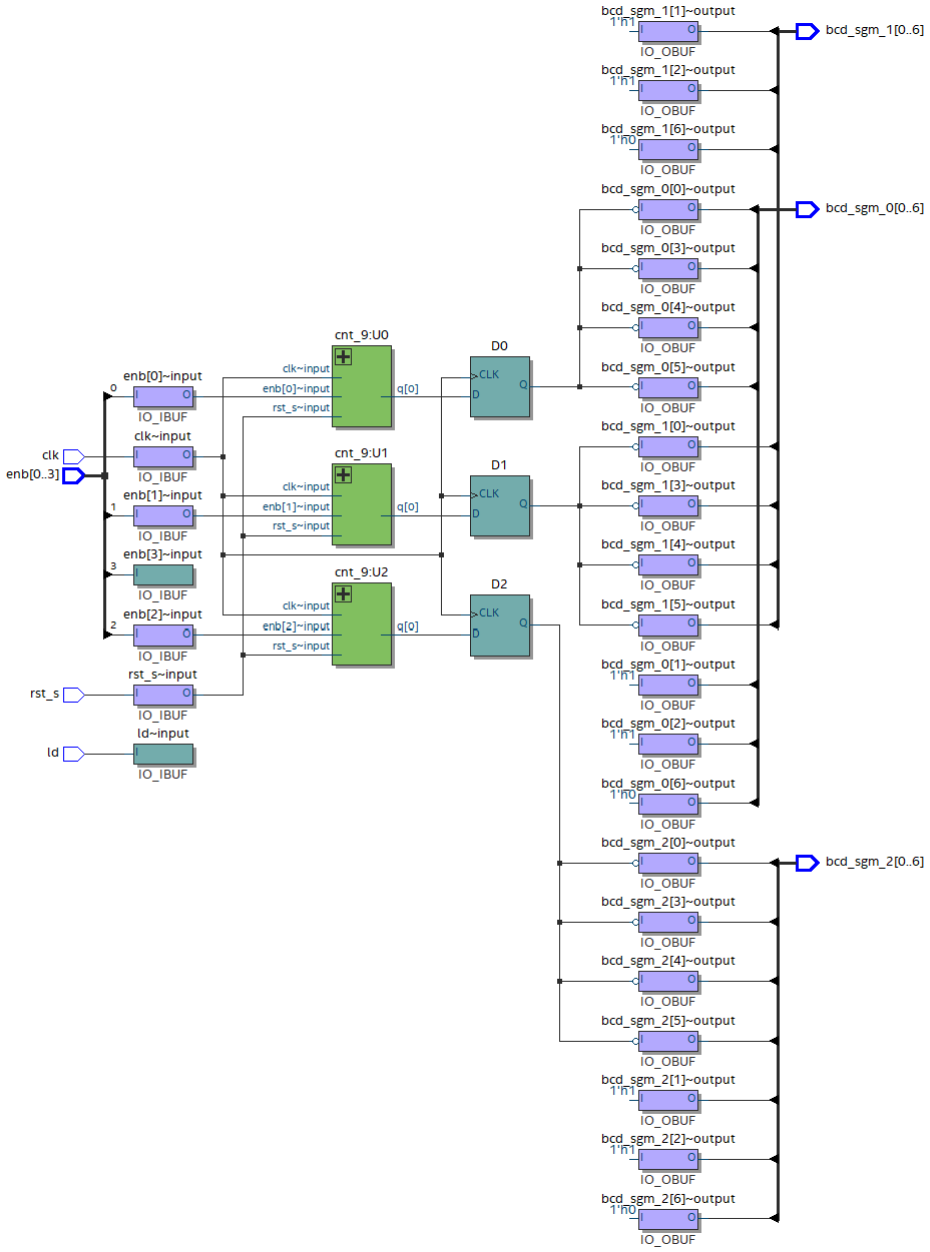

Figura 2: Ampliação do Circuito RTL com o Technology Map Viewer

4. Conclusão

Neste relatório, implementamos um contador digital BCD de 000 a 999 em Verilog. A solução foi dividida em módulos funcionais, incluindo contadores, flip-flops e conversores para displays de 7 segmentos. Cada módulo foi descrito e documentado, facilitando a compreensão do projeto.

Usamos o Quartus para compilar o projeto e visualizar o circuito RTL, confirmando a correta interconexão dos módulos. A implementação demonstrou a eficácia do Verilog na criação de designs estruturados e modulares.

Este trabalho forneceu uma base sólida para expansões e aprimoramentos no design de circuitos digitais, destacando a importância das ferramentas de síntese e simulação na validação do projeto.