Universidade Católica de Santos
Centro de Ciências Exatas, Arquitetura e Engenharia

XII Semana de Ciência, Tecnologia e Inovação
Instrutor: Felipe Ferreira / Thiago Ferauche

Arquivo de apoio
Última atualização: October 17, 2022

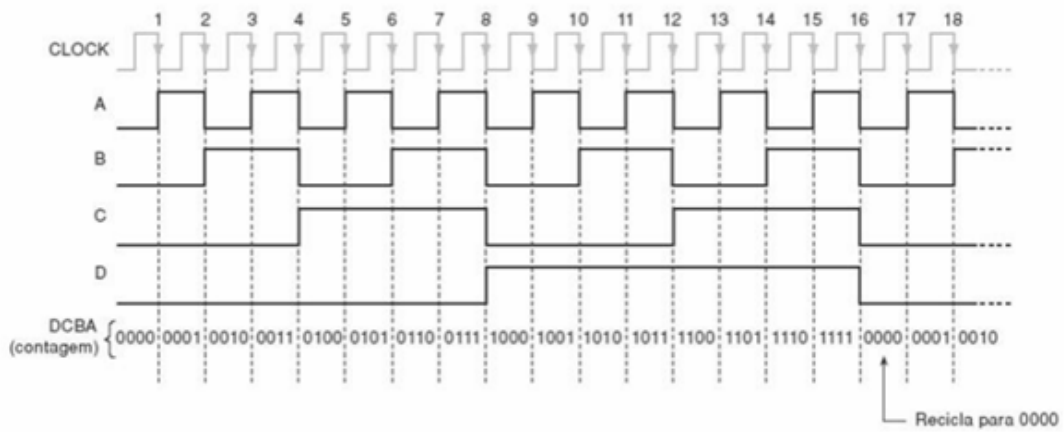
1 Objetivos

1.1 Objetivos Gerais

- Familiarize-se com o ambiente de desenvolvimento Lattice iCE40;
- Compreender o processo de projeto de circuitos usando HDLs e implementação em FPGA;
- Simule e verifique resultados com placas FPGA reais;

1.2 Objetivos específicos

- Implementar um divisor de frequência (algo como a imagem abaixo);
- Implementar um decodificador para display de 7 segmentos;



Frequency divider.

2 Desenvolvimento de laboratório

2.1 Análise do circuito

Neste laboratório, vamos implementar um divisor de frequência usando dois contadores. Este circuito recebe um relógio ou tipo similar de entrada e emite o referido sinal dividido por um determinado valor como saída.

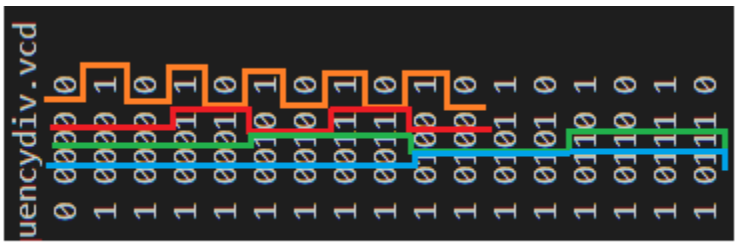
2.2 Codificação e simulação HDL

1. Crie uma pasta chamada "my_design";
2. Implemente o seguinte código em Verilog, chamado "frequency_divider.v";

```
module frequency_divider (

    // Inputs
    input clk,
    input rst_btn,

    // Outputs
    output reg[3:0] out_clk
);
    wire rst;
    reg[31:0] count;
    localparam [31:0] max_count = (12000000) - 1;

    // Reset is the inverse of the reset button
    assign rst = ~rst_btn;

    // Clock divider
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            out_clk <= 4'b0;
            count <= 32'b0;
        end
        else if (count == max_count) begin
            count <= 32'b0;
            out_clk <= out_clk + 1'b1;
        end
        else begin
            count <= count + 1;
        end
    end

endmodule
```

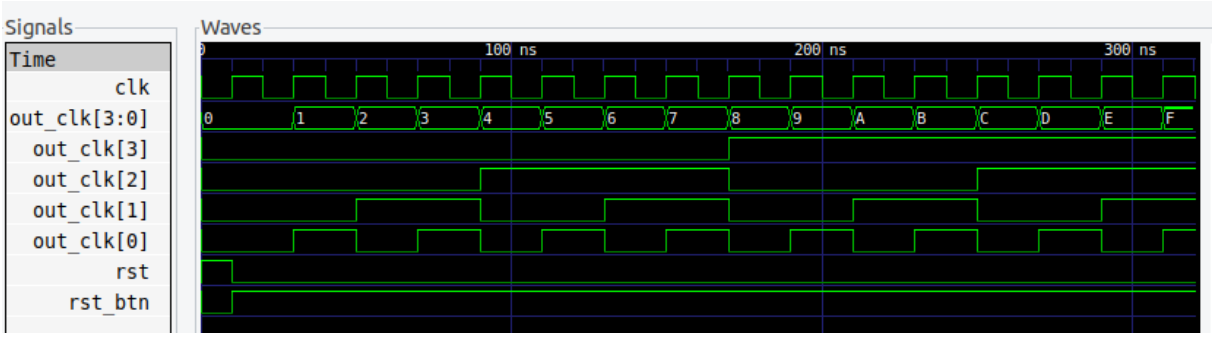
3. Na mesma pasta, digite os comandos fornecidos abaixo usando o terminal Linux. Em seguida, abra o arquivo "frequency_divider_tb.vcd" com gtkwave e compare seus resultados com a próxima imagem:

```
$ iverilog frequency_divider_tb.v
$ vvp a.out
```



Frequency divider vvp.

- **Saber mais:** Verilog fornece algumas funções e tarefas do sistema especificamente para gerar entradas e saídas para ajudar na verificação. **Monitor** é um exemplo de tarefa do sistema. As tarefas do sistema não são sintetizadas, são ignoradas pelas ferramentas de síntese.
- A instrução monitor não é tão poderosa quanto as ferramentas gráficas de forma de onda, mas é útil em muitos casos e fornece ideia do comportamento do circuito.
- A palavra-chave 'always' recebe dois argumentos (conhecidos como 'lista de sensibilidade'). Olhando para o bloco Always, estamos ativando-o em cada borda de subida do clock ou sinais de reset.

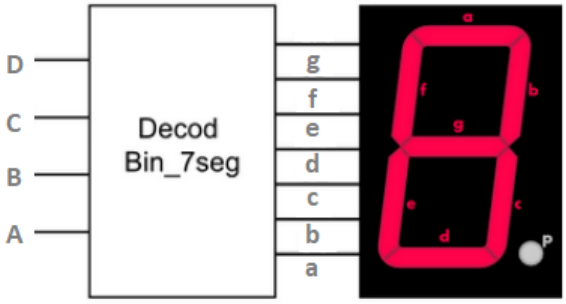


Frequency divider simulation.

3 Desafio

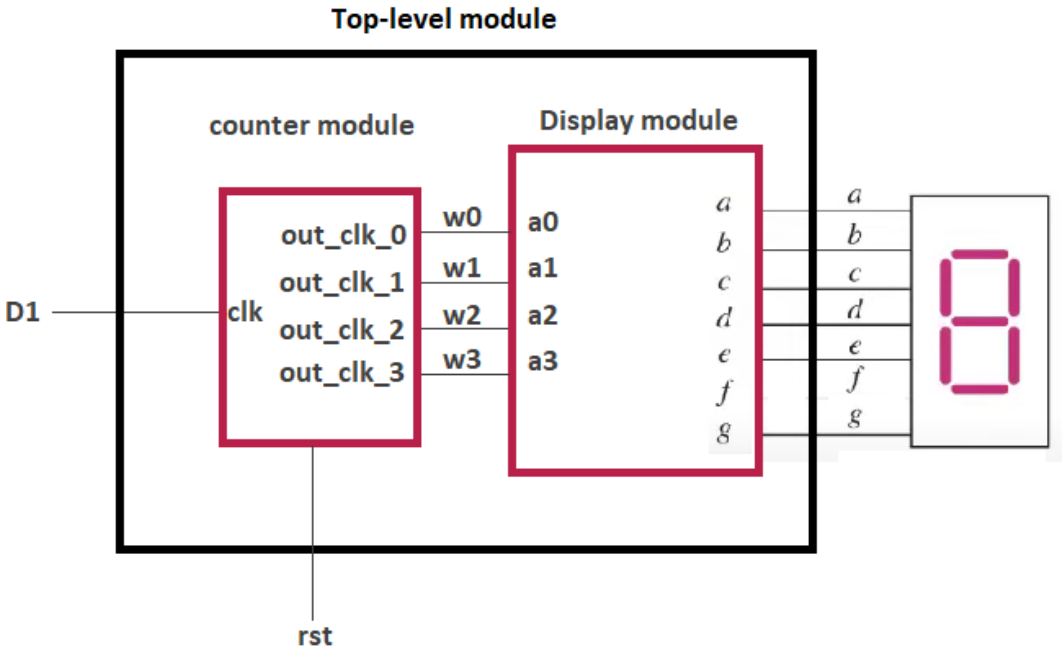
1. O design abaixo descrito em verilog, é um circuito que converte um número BCD de entrada para uma representação em um display de 7 segmentos de ânodo comum (imagem a seguir). Observe que o HDL está descrito apenas para representar até o valor 10 decimais. Dadas as explicações, tente completar a descrição para ser possivel representar os valores em hexadecimal até 15.

```
module decod_bcd (

    //Input
    input wire[3:0] count,

    //Output
    output reg[6:0] seg_count
);

    always @ * begin
        case (count)
        4'b0000 : seg_count = 7'b0000001;
        4'b0001 : seg_count = 7'b1001111;
        4'b0010 : seg_count = 7'b0010010;
        4'b0011 : seg_count = 7'b0000110;
        4'b0100 : seg_count = 7'b1001100;
        4'b0101 : seg_count = 7'b0100100;
        4'b0110 : seg_count = 7'b0100000;
        4'b0111 : seg_count = 7'b0001111;
        4'b1000 : seg_count = 7'b0000000;
        4'b1001 : seg_count = 7'b0000100;
        4'b1010 : seg_count = 7'b;
        4'b1011 : seg_count = 7'b;
        4'b1100 : seg_count = 7'b;
        4'b1101 : seg_count = 7'b;
        4'b1110 : seg_count = 7'b;
        4'b1111 : seg_count = 7'b;
        endcase
    end
endmodule
```

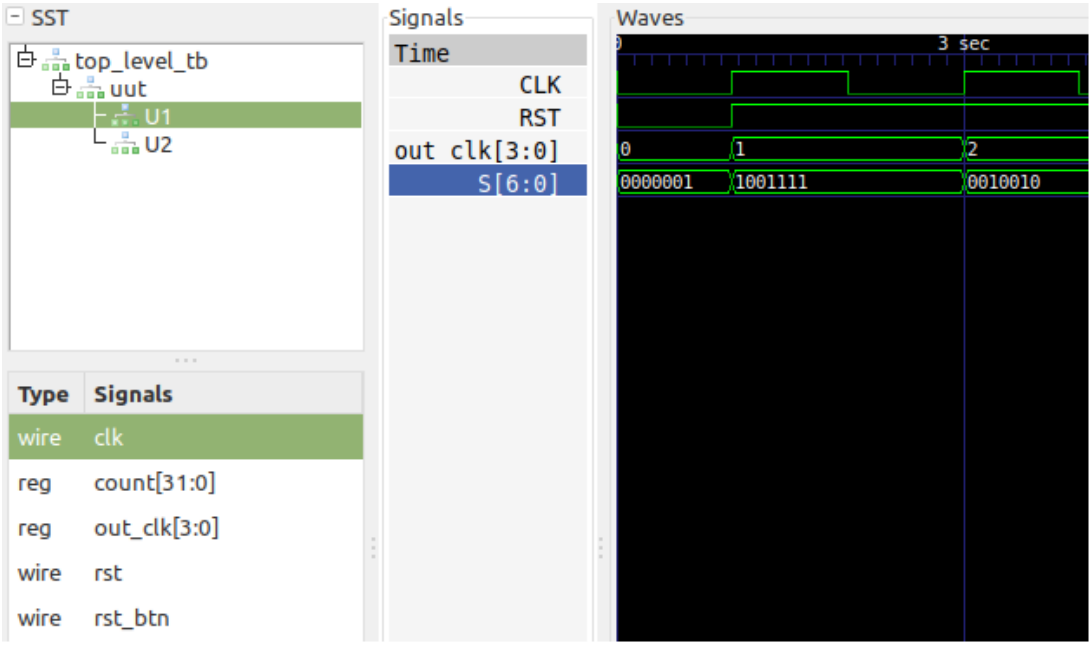


Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

Decoder for 7-segment display.



Top level design structure.



Simulation top level design.

3.1 Implementação FPGA

1. Crie o arquivo de restrição de pinos "top_level.pcf" dentro da pasta. (disponibilizado):

```
set_io s[6] E2
set_io s[5] B1
set_io s[4] C5
set_io s[3] C6
set_io s[2] E3
set_io s[1] C2
set_io s[0] B4
set_io clk_d1 D1
set_io -pullup yes rst A1
set_io -nowarn PMOD1 D1
set_io -nowarn PMOD1 A1
set_io -nowarn PMOD2 E2
set_io -nowarn PMOD1 B1
set_io -nowarn PMOD2 C5
set_io -nowarn PMOD2 C6
set_io -nowarn PMOD1 E3
set_io -nowarn PMOD2 C2
set_io -nowarn PMOD2 B4
```

   Observe que o sinal de clock é atribuído ao pino D1, ele aproveita o oscilador da placa conectada diretamente ao D1 cuja frequência é 12MHz.

2. Use o makefile fornecido para realizar: síntese, PnR, empacotamento e gravação no iCESugar-nano. Na pasta do projeto, digite os comandos fornecidos abaixo no terminal linux:

```
$ make build
$ make run
```