Residência Tecnológica em FPGA - EmbarcaTech / Turma Beta
Residente: Hirislayne Batista Ramos dos Santos
Matrícula: 20251RSE.MTC0041

Unidade 3 - Capítulo 2: Linguagem VHDL - Registradores

1. Objetivo

Desenvolver a capacidade de projetar, modelar, simular e implementar circuitos digitais por meio da linguagem VHDL (VHSIC *Hardware Description Language*), promovendo a compreensão dos conceitos de lógica digital e sua aplicação prática no desenvolvimento de sistemas embarcados e dispositivos programáveis, como FPGAs.

2. Enunciado - Registrador Paralelo-Paralelo de 8 Bits com Flip-Flops Tipo D

Desenvolver e simular um registrador paralelo de 8 bits utilizando flip-flops tipo D, onde cada flip-flop possui apenas entrada D e entrada de clock. O registrador deve permitir o armazenamento de dados paralelos e a atualização simultânea de todos os bits a cada pulso ascendente de clock.

3. Requisitos

- Implementar em VHDL um registrador paralelo de 8 bits utilizando flip-flops tipo D, onde:
  - Cada flip-flop deve ter entrada D, entrada de clock e saída Q e Q'.
  - O registrador deve possuir uma entrada de clock comum para todos os flip-flops.
  - Os dados de entrada devem ser carregados de forma paralela em cada pulso ascendente de clock.
  - Comprove o seu funcionamento e apresente o relatório da implementação
- Desenvolver um testbench que simule o comportamento do registrador, fornecendo diferentes valores de entrada e analisando as saídas. Relate a simulação e anexe os códigos .vhdl do registrador e do testbench.

4. Desenvolvimento

a. Implementação dos Códigos em VHDL:

Primeiro, foi criado o componente básico, o flip-flop tipo D:

Código *flip_flop_D.vhd*

```
-- Arquivo do flip flop tipo D para o registrador de 8 bits
```

```vhdl
-- Descrição: Flip flop tipo D com entrada de dados D, clock clk, saída Q e saída invertida Qn.
--            Os dados de entrada são carregados de forma paralela em cada pulso ascendente de clock (rising edge).

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity flip_flop_D is
    port (
        D     : in std_logic;      -- Entrada de dados de 1 bit
        clk   : in std_logic;      -- Clock
        Q     : out std_logic;     -- Saída normal de 1 bit
        Qn : out std_logic         -- Saída complementar de 1 bit
    );
end flip_flop_D;

architecture Behavioral of flip_flop_D is
    signal internal_q : std_logic := '0';  -- Sinal interno para armazenar o valor de Q, evitando inconsistências
begin
    process(clk)
    begin
        if rising_edge(clk) then
            internal_q <= D; -- Armazena o valor de D na borda de subida do clock, pois os dados não são atualizados imediatamente
        end if;
    end process;

    -- Atualiza as saídas imediatamente após o clock
    Q <= internal_q;          -- Saída do valor armazenado
    Qn <= not internal_q;     -- Complemento do valor armazenado
end Behavioral;
```

Com isso, construímos o registrador de entrada e saída paralelo usando 8 flip-flops tipo D implementado anteriormente:

Código *registrador_8bits.vhd*

```vhdl
-- Residencia Tecnologica em FPGA - Residente: Hirislayne Batista
-- Data: 17/09/2025
-- Descrição: O registrador possui uma entrada de clock comum para todos os
```

```
flip-flops e os
--            dados de entrada são carregados de forma paralela em cada pulso
ascendente de clock.

-- Arquivo do Módulo principal
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity registrador_8bits is
    port (
        clk    : in std_logic;                     -- Clock comum
        D_in   : in std_logic_vector(7 downto 0);  -- Entrada paralela de 8
bits
        Q_out  : out std_logic_vector(7 downto 0); -- Saída paralela Q de 8
bits
        Qn_out : out std_logic_vector(7 downto 0)  -- Saída paralela Q' de 8
bits (complemento)
    );
end registrador_8bits;

architecture structural of registrador_8bits is

    -- Componente do flip-flop tipo D
    component flip_flop_D is
        port (
            D     : in std_logic;
            clk   : in std_logic;
            Q     : out std_logic;
            Qn    : out std_logic
        );
    end component;
begin

    -- Instanciando 8 flip-flops tipo D
    gen_ff: for i in 0 to 7 generate
        ff_inst: flip_flop_D
            port map (
                D   => D_in(i),
                clk => clk,
                Q   => Q_out(i),
                Qn  => Qn_out(i)
```

```
            );
    end generate gen_ff;

end structural;
```

E por fim, o seu testbench.

Código *tb_registrador_8bits.vhd*

```
-- Testbench para o registrador de 8 bits
-- Descrição: Este testbench gera um clock e aplica diferentes valores de
entrada ao registrador

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_registrador_8bits is
end tb_registrador_8bits;

architecture testbench of tb_registrador_8bits is
    signal clk    : std_logic := '0';
    signal D_in   : std_logic_vector(7 downto 0) := (others => '0');
    signal Q_out  : std_logic_vector(7 downto 0);
    signal Qn_out : std_logic_vector(7 downto 0);

    constant CLK_PERIOD : time := 10 ns;

begin
    -- Instância do registrador
    uut: entity work.registrador_8bits
        port map (
            clk    => clk,
            D_in   => D_in,
            Q_out  => Q_out,
            Qn_out => Qn_out
        );

    -- Geração do clock
    clk_process: process
    begin
        while now < 200 ns loop
            clk <= '0';
```

```
            wait for CLK_PERIOD/2;
            clk <= '1';
            wait for CLK_PERIOD/2;
        end loop;
        wait;
    end process;

    -- Estímulos de teste
    stimulus: process
    begin
        -- Teste 1: Dado 00000000
        D_in <= "00000000";
        wait for 20 ns;

        -- Teste 2: Dado 10101010
        D_in <= "10101010";
        wait for 20 ns;

        -- Teste 3: Dado 01010101
        D_in <= "01010101";
        wait for 20 ns;

        -- Teste 4: Dado 11110000
        D_in <= "11110000";
        wait for 20 ns;

        -- Teste 5: Dado 00001111
        D_in <= "00001111";
        wait for 20 ns;

        -- Teste 6: Dado 11111111
        D_in <= "11111111";
        wait for 20 ns;

        wait;
    end process;

end testbench;
```

b. Compilação e Simulação

Para compilar e simular o circuito implementado nos códigos acima, execute os seguintes comandos na Terminal:

- Compilar os componentes

ghdl -a flip_flop_D.vhd
ghdl -a registrador_8bits.vhd
ghdl -a tb_registrador_8bits.vhd

- Elaborar o testbench

ghdl -e tb_registrador_8bits

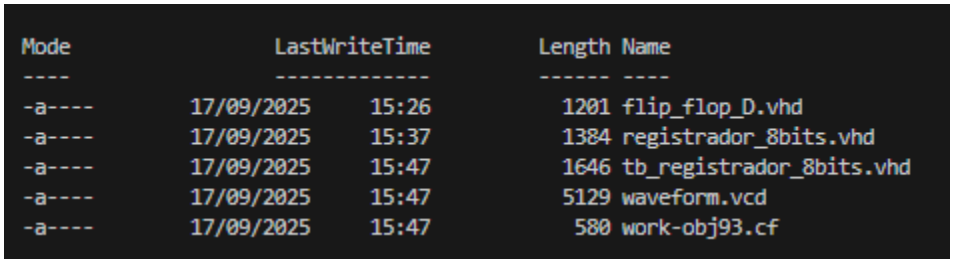
- Executar a simulação e gerar waveform

ghdl -r tb_registrador_8bits --vcd=waveform.vcd --stop-time=200ns

- Visualizar a forma de onda no GTKWave

gtkwave waveform.vcd

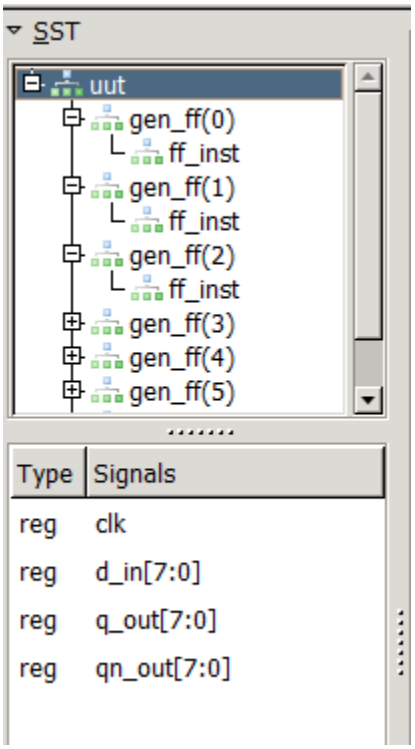
Com isso, por meio do comando *dir* é possível ver o diretório completo contendo os arquivos criados (.vhd, .vcd e .cf):



```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         17/09/2025     15:26           1201 flip_flop_D.vhd
-a----         17/09/2025     15:37           1384 registrador_8bits.vhd
-a----         17/09/2025     15:47           1646 tb_registrador_8bits.vhd
-a----         17/09/2025     15:47           5129 waveform.vcd
-a----         17/09/2025     15:47            580 work-obj93.cf
```

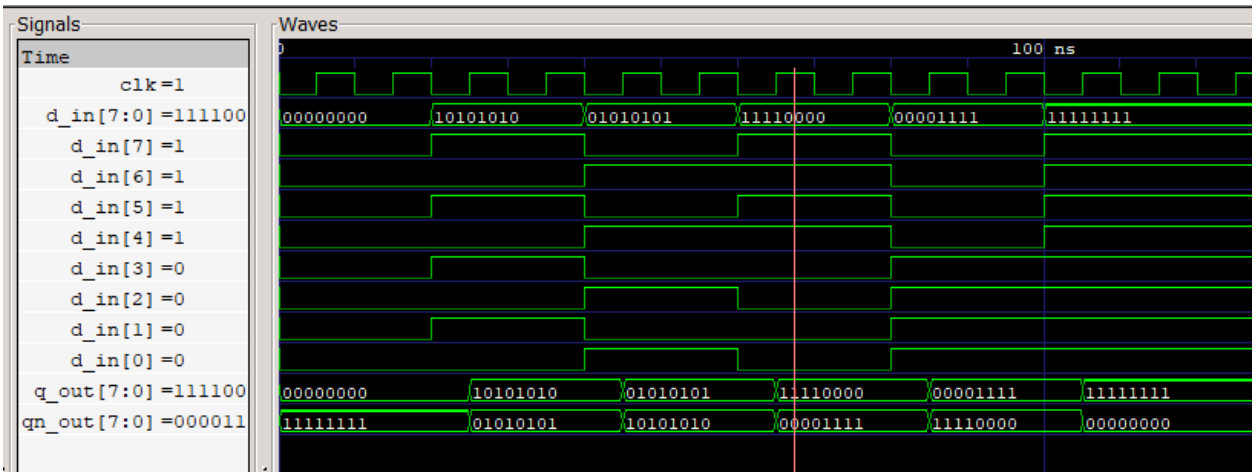
Fonte: Terminal no VSCode.

Na tela do GTKWave adicione os sinais (contidos na instância *uut* criada na aba do canto superior esquerdo) e ajuste as formas de ondas por meio da lupa "*Zoom Fit*" para melhor visualização.



Fonte: GTKWave no VSCode.

Sendo assim, podemos ver o comportamento das saídas do registrador por meio dos valores de entrada fornecidos no arquivo de testbench.



Fonte: GTKWave no VSCode.

Link para acesso à pasta do projeto: [Arquivo Tarefa U3C2 - Google Drive](#).