Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Residência Profissional em FPGA – Turma DELTA - 2025.2

Atividade – Referente ao capítulo 01 da unidade 05

Tema do Capítulo – Linguagem VHDL: Registradores

Prazo dia 21/09/2025 as 23:59

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.

Enunciado: 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.

Instruções:

Ação	Função
1	Implementar o registrador paralelo de 8 bits em um simmulador, suando flipo-flop tipo D, com os seguintes requisitos: • 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
2	Implementar em VHDL um registrador paralelo de 8 bits utilizando flip-flops tipo D do item 1.
3	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 testebench.

Solução

- GitHub: Segunda_Fase_FPGA/Unidade_05/Cap_01
- Link do Vídeo: https://youtu.be/dbzM9yc8OsM
- Link do Código Completo: Unid_05_Cap_01.zip

1) Implementar o registrador paralelo de 8 bits. Simular e usando Flip-Flops tipo D

É fundamental entender a estrutura do circuito em um nível lógico. Um registrador paralelo de 8 bits é composto por 8 flip-flops tipo D, todos compartilhando o mesmo sinal de clock no pulso ascendente.

Componentes: 8 Flip-Flops (FF) do tipo D.

- Cada flip-flop possui uma entrada D (Dado), uma entrada CLK (Clock) e uma saída Q (Saída principal) e Q' (Saída invertida).

  Irei usar a nomenclatura em vetores para facilitar D_IN [7:0] 8 bits de entrada, 0 a 7. E outro vetor Q_OUT [7:0] 8 bits de saída, 0 a 7.

Entradas do Registrador:

- D_IN [7:0]: Um barramento de dados de 8 bits que serve como a entrada paralela. Cada linha deste barramento (D_IN[0], D_IN[1], ..., D_IN[7]) se conecta à entrada D do flip-flop correspondente.
- CLK: Um único sinal de clock que é conectado à entrada CLK de todos os 8 flip-flops simultaneamente. Isso garante que todos os bits sejam atualizados ao mesmo tempo.

Saídas do Registrador:

- Q_OUT [7:0]: Um barramento de dados de 8 bits que representa a saída paralela. Cada linha deste barramento (Q_OUT[0], Q_OUT[1], ..., Q_OUT[7]) é conectada à saída Q do flip-flop correspondente.

Na imagem a seguir mostra um exemplo parar 3 bits, iremos construir em VHDL para 8 bits. Esses dados podem vir de circuitos combinacionais diversos ou de outros registradores. Tudo se movimenta de acordo com a frequência do Clock, no pulso ascendente.

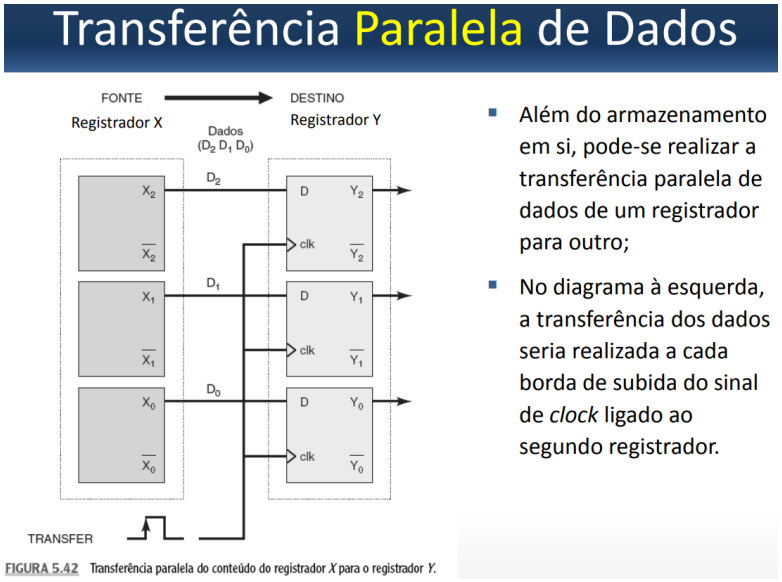


FIGURA 5.42 Transferência paralela do conteúdo do registrador *X* para o registrador *Y*.

Esta operação síncrona é a base para o armazenamento de dados em sistemas digitais.

Comprovação de Funcionamento

O princípio de funcionamento é o seguinte:

1) O valor a ser armazenado é colocado no barramento de entrada D_IN.
2) Enquanto o sinal de CLK está em nível baixo ou alto (estável), o valor na saída Q_OUT permanece inalterado, mantendo o dado que foi armazenado no pulso de clock anterior.
3) No exato momento em que o CLK transita de nível baixo para nível alto (**borda de subida**), cada flip-flop "captura" o valor presente em sua entrada D e o transfere para sua saída Q.
4) Como todos os flip-flops recebem o mesmo sinal de clock, todos os 8 bits de D_IN são "fotografados" e armazenados simultaneamente, aparecendo na saída Q_OUT.

Exemplo de Simulação:

- Tempo t0: D_IN = 10101010, Q_OUT = 00000000 (estado inicial).
- Tempo t1 (ocorre uma borda de subida no CLK): O valor de D_IN (10101010) é carregado no registrador. A saída Q_OUT se torna 10101010.
- Tempo t2: O valor de D_IN muda para 11110000. A saída Q_OUT permanece 10101010, pois ainda não houve uma nova borda de subida no clock.
- Tempo t3 (ocorre a próxima borda de subida no CLK): O novo valor de D_IN (11110000) é carregado. A saída Q_OUT agora se torna 11110000.

2) Código em VHDL

Esse Registrador é do tipo PIPO (Parallel In / Parallel Out).

A justificativa é a seguinte:

- **Parallel In (Entrada Paralela):** O seu registrador possui a porta D : IN STD_LOGIC_VECTOR(7 DOWNTO 0). Isso significa que ele recebe todos os 8 bits de dados de uma só vez, em paralelo.
- **Parallel Out (Saída Paralela):** A porta de saída Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) disponibiliza todos os 8 bits armazenados simultaneamente.

```vhdl
-- Atividade: Unidade 05, Capítulo 01
-- Autor:     Manoel Felipe Costa Furtado
-- Data:      21/09/2025
-- Arquivo:   unid_05_cap_01.vhd
-- Versão:    1.0
-- Descrição: Implementa um registrador paralelo de 8 bits utilizando o conceito
--            de flip-flops tipo D. A operação é síncrona, baseada na borda
--            de subida do sinal de clock.
--------------------------------------------------------------------------------

-- Biblioteca padrão IEEE.
-- A cláusula USE importa o pacote std_logic_1164, que define os tipos
-- STD_LOGIC e STD_LOGIC_VECTOR, essenciais para a modelagem de hardware.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- ENTIDADE (ENTITY)
-- Define a interface pública do nosso componente, ou seja, suas portas de
-- entrada e saída. Funciona como um "contrato" que descreve como outros
-- circuitos podem se conectar a este.
ENTITY unid_05_cap_01 IS
    PORT (
        -- Porta de entrada do clock. É o sinal que dita o ritmo das operações.
        CLK : IN  STD_LOGIC;
        -- Barramento de entrada de dados com 8 bits.
        D   : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
        -- Barramento de saída de dados com 8 bits, onde o valor armazenado é lido.
        Q   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END ENTITY unid_05_cap_01;

-- ARQUITETURA (ARCHITECTURE)
-- Descreve o comportamento interno e a estrutura lógica do componente
-- definido na entidade 'unid_05_cap_01'.
ARCHITECTURE behavioral OF unid_05_cap_01 IS
```

```vhdl
BEGIN
    -- PROCESSO (PROCESS) SÍNCRONO
    -- Um processo é um bloco de código sequencial que é sensível a mudanças
    -- nos sinais de sua lista de sensibilidade (neste caso, apenas 'CLK').
    -- Por ser sensível apenas ao clock, ele modela um comportamento síncrono.
    process(CLK)
    BEGIN
        -- Detecção de borda de subida do clock.
        -- A função 'rising_edge()' retorna verdadeiro apenas no instante em
        -- que o sinal CLK transita de '0' para '1'.
        -- Esta é a condição que dispara o armazenamento de dados.
        IF rising_edge(CLK) THEN
            -- Atribuição síncrona.
            -- O valor presente no barramento de entrada 'D' é amostrado e
            -- atribuído ao barramento de saída 'Q'. Esta operação simula
            -- o comportamento de 8 flip-flops tipo D operando em paralelo.
            Q <= D;
        END IF;
    END PROCESS;

END ARCHITECTURE behavioral;
```

3) Código em VHDL para a Simulação, testbench

```vhdl
-- Atividade: Unidade 05, Capítulo 01
-- Autor:     Manoel Felipe Costa Furtado
-- Data:      21/09/2025
-- Arquivo:   unid_05_cap_01_tb.vhd
-- Versão:    1.0
-- Descrição: Testbench para o registrador paralelo de 8 bits (unid_05_cap_01).
--            Este código gera os sinais de clock e de dados para estimular
--            o componente sob teste e permitir a verificação de seu
--            funcionamento em um simulador.
--------------------------------------------------------------------------------

-- Biblioteca padrão IEEE.
-- A cláusula USE importa o pacote std_logic_1164, que define os tipos
-- STD_LOGIC e STD_LOGIC_VECTOR, essenciais para a modelagem de hardware.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- A entidade de um testbench é tipicamente vazia, pois ele não possui
-- portas de entrada ou saída. Ele é o ambiente de simulação de mais alto nível.
ENTITY unid_05_cap_01_tb IS
END ENTITY unid_05_cap_01_tb;
```

```
-- Arquitetura de simulação para o testbench.
ARCHITECTURE simulation OF unid_05_cap_01_tb IS
    -- 1. Declaração do Componente Sob Teste (DUT - Device Under Test)
    -- A assinatura do componente deve ser idêntica à da sua entidade.
    COMPONENT unid_05_cap_01
        PORT (
            CLK : IN  STD_LOGIC;
            D   : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
            Q   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
    END COMPONENT;

    -- 2. Sinais (Signals) do Testbench
    -- Sinais internos usados para conectar ao DUT, gerar estímulos e observar saídas.
    SIGNAL s_clk : STD_LOGIC := '0'; -- Sinal para gerar o clock
    SIGNAL s_d   : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0'); -- Sinal para os dados de entrada (DUT)
    SIGNAL s_q   : STD_LOGIC_VECTOR(7 DOWNTO 0); -- Sinal para observar a saída (DUT)

    -- 3. Constantes (Constants)
    -- Usado para definir parâmetros fixos, facilitando a manutenção do código.
    CONSTANT CLK_PERIOD : TIME := 10 ns; -- Define o período do clock em 10 ns (Frequência = 100 MHz)

BEGIN
    -- 4. Instanciação do Componente Sob Teste (DUT)
    -- Aqui, criamos uma instância do nosso registrador e conectamos suas portas
    -- (CLK, D, Q) aos sinais do testbench (s_clk, s_d, s_q) usando o mapeamento de portas.
    dut_instance : unid_05_cap_01
        PORT MAP (
            CLK => s_clk,
            D   => s_d,
            Q   => s_q
        );

    -- 5. Processo de Geração de Clock
    -- Este processo executa de forma independente e contínua para gerar um sinal
    -- de clock com o período definido em CLK_PERIOD.
    clk_process : PROCESS
    BEGIN
        s_clk <= '0';
        WAIT FOR CLK_PERIOD / 2; -- Clock fica em nível baixo por 5 ns
        s_clk <= '1';
        WAIT FOR CLK_PERIOD / 2; -- Clock fica em nível alto por 5 ns
    END PROCESS clk_process;

    -- 6. Processo de Geração de Estímulos
    -- Este processo gera a sequência de valores de entrada para testar
    -- o comportamento do registrador em diferentes cenários.
    stimulus_process : PROCESS
```

```vhdl
    BEGIN
        -- Aguarda 1 ns no início para não coincidir com o tempo 0.
        WAIT FOR 1 ns;

        -- Teste 1: Carrega o valor "todos em 1".
        s_d <= "11111111";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=10ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "11111111".

        -- Teste 2: Carrega o valor "todos em 0".
        s_d <= "00000000";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=20ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "00000000".

        -- Teste 3: Carrega um padrão alternado (par).
        s_d <= "10101010";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=30ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "10101010".

        -- Teste 4: Carrega um padrão alternado (ímpar).
        s_d <= "01010101";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=40ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "01010101".

        -- Teste 5: Carrega a metade alta dos bits.
        s_d <= "11110000";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=50ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "11110000".

        -- Teste 6: Carrega um valor assimétrico para teste.
        s_d <= "11001001";
        WAIT UNTIL rising_edge(s_clk);
        -- Aguarda a próxima borda de subida do clock (em t=60ns).
        -- Após esta linha, espera-se que s_q seja atualizado para "11001001".

        -- Fim dos estímulos.
        -- A instrução 'WAIT;' suspende este processo indefinidamente. [cite: 31]
        -- Isso impede que o processo de estímulos reinicie. A simulação
        -- irá parar quando atingir o '--stop-time' definido no comando de execução. [cite: 31]
        WAIT;
    END PROCESS stimulus_process;

END ARCHITECTURE simulation;
```

4) Compilação

```
#--------------------------------------------------------------------------------
# Arquivo:   comandos.txt
# Descrição: Roteiro de comandos para análise, elaboração, execução e
#            visualização da simulação do registrador de 8 bits.
#--------------------------------------------------------------------------------

# ETAPA 1: ANÁLISE (COMPILAÇÃO)
# O comando 'ghdl -a' (analisar) verifica a sintaxe dos arquivos VHDL e, se
# não houver erros, compila-os para um formato intermediário na biblioteca de trabalho.
# É necessário analisar primeiro as dependências (o design) e depois os arquivos
# que dependem delas (o testbench).

ghdl -a unid_05_cap_01.vhd
ghdl -a unid_05_cap_01_tb.vhd

# ETAPA 2: ELABORAÇÃO
# O comando 'ghdl -e' (elaborar) constrói o modelo de simulação. Ele pega a
# entidade de mais alto nível (nosso testbench, 'unid_05_cap_01_tb'), e a "conecta"
# com todas as suas instâncias de componentes (nosso registrador, 'unid_05_cap_01'),
# criando um executável pronto para a simulação.

ghdl -e unid_05_cap_01_tb

# ETAPA 3: EXECUÇÃO (SIMULAÇÃO)
# O comando 'ghdl -r' (rodar) executa a simulação.

# A linha abaixo está comentada com '#' porque ela rodaria a simulação por
# tempo indefinido, gerando um arquivo .vcd gigantesco.
# ghdl -r unid_05_cap_01_tb --vcd=waveform.vcd

# Esta é a linha correta. Ela executa a simulação e inclui dois argumentos importantes:
# --vcd=waveform.vcd : instrui o simulador a salvar todas as mudanças de
#                      sinais em um arquivo chamado 'waveform.vcd'.
# --stop-time=100ns  : instrui o simulador a parar automaticamente após
#                      100 nanosegundos, evitando o loop infinito.
ghdl -r unid_05_cap_01_tb --vcd=waveform.vcd --stop-time=80ns

# ETAPA 4: VISUALIZAÇÃO
# O comando 'gtkwave' abre o programa de visualização de formas de onda,
# carregando o arquivo 'waveform.vcd' gerado na etapa anterior.
# Nele, você pode analisar graficamente o comportamento dos sinais ao longo do tempo.

gtkwave waveform.vcd
```

5) Simulação

O resultado esperado em uma visualização de formas de onda seria:

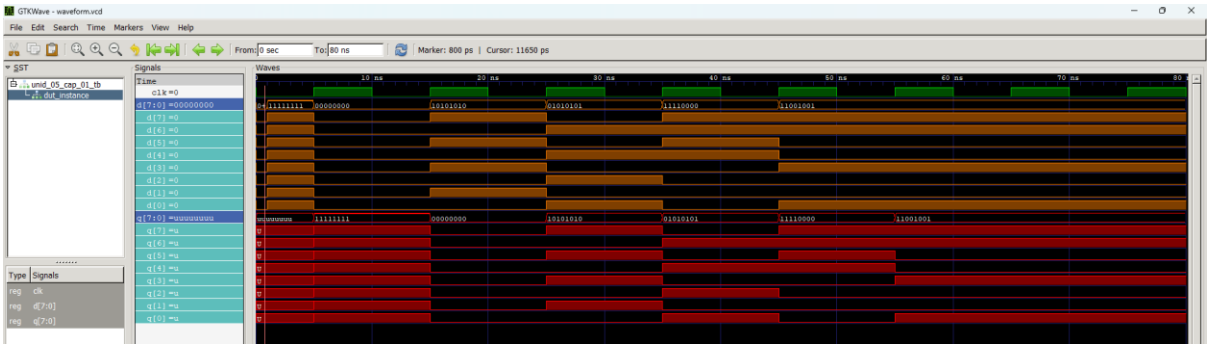
1. Geração de Clock: O sinal s_clk oscila continuamente entre '0' e '1' a cada 5 ns, criando um clock com período de 10 ns.
2. Estímulos e Respostas:

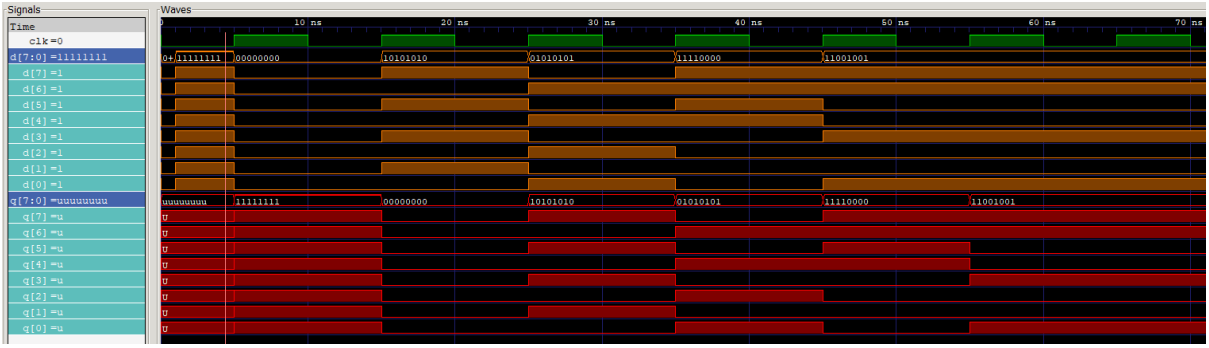
A simulação irá demonstrar o comportamento síncrono do registrador através de 6 testes distintos. A saída s_q só deve mudar de valor no exato instante da borda de subida do clock (rising_edge).

- **Início (t=0ns):** O sinal de entrada s_d é inicializado em "00000000". A saída s_q está em um estado indefinido (UUUUUUUU) ou zerado, pois nenhum pulso de clock ocorreu ainda.
- **t=1ns:** A entrada s_d muda para o primeiro valor de teste, "11111111". A saída s_q permanece inalterada.
- **Na borda de subida em t=10ns:** O registrador captura o valor de s_d. A saída s_q é atualizada para **"11111111"**.
- **Na borda de subida em t=20ns:** O segundo valor de teste ("00000000") é carregado. A saída s_q é atualizada para **"00000000"**.
- **Na borda de subida em t=30ns:** O terceiro valor de teste ("10101010") é carregado. A saída s_q é atualizada para **"10101010"**.
- **Na borda de subida em t=40ns:** O quarto valor de teste ("01010101") é carregado. A saída s_q é atualizada para **"01010101"**.
- **Na borda de subida em t=50ns:** O quinto valor de teste ("11110000") é carregado. A saída s_q é atualizada para **"11110000"**.
- **Na borda de subida em t=60ns:** O sexto e último valor de teste ("11001001") é carregado. A saída s_q é atualizada para **"11001001"**.
- **De t=60ns a t=80ns (fim da simulação):** Nenhum novo valor é aplicado a s_d. A saída s_q permanece estável em **"11001001"** pelos ciclos de clock restantes, demonstrando a capacidade de armazenamento do registrador.

Este comportamento, onde a saída s_q reflete a entrada s_d somente após uma borda de subida do clock, confirma que o registrador paralelo de 8 bits foi implementado e verificado corretamente, operando de forma síncrona como esperado.

Nas imagens a seguir podemos verificar que a simulação foi um sucesso.

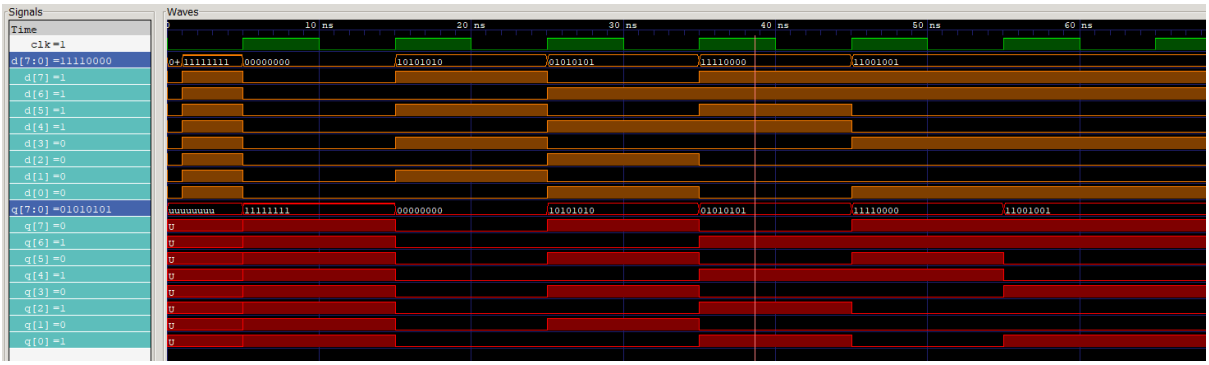

GTKWave - waveform.vcd
File Edit Search Time Markers View Help
SST
unid_05_cap_01_tb
dut_instance
Signals
Waves
Time
clk =0
d[7:0] =00000000
q[7:0] =uuuuuuuu
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =0
d[7:0] =11111111
q[7:0] =uuuuuuuu
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
11111111
00000000
10101010
01010101
11110000
11001001

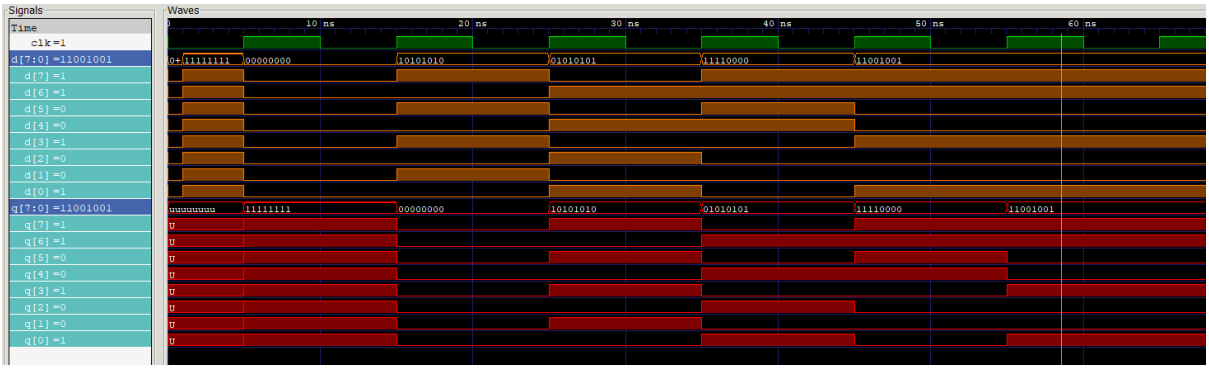

Signals
Waves
Time
clk =1
d[7:0] =00000000
q[7:0] =11111111
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =1
d[7:0] =10101010
q[7:0] =00000000
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =1
d[7:0] =01010101
d[7] =0
d[6] =1
d[5] =0
d[4] =1
d[3] =0
d[2] =1
d[1] =0
d[0] =1
q[7:0] =10101010
q[7] =1
q[6] =0
q[5] =1
q[4] =0
q[3] =1
q[2] =0
q[1] =1
q[0] =0
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =1
d[7:0] =11110000
d[7] =1
d[6] =1
d[5] =1
d[4] =1
d[3] =0
d[2] =0
d[1] =0
d[0] =0
q[7:0] =01010101
q[7] =0
q[6] =1
q[5] =0
q[4] =1
q[3] =0
q[2] =1
q[1] =0
q[0] =1
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =1
d[7:0] =11001001
d[7] =1
d[6] =1
d[5] =0
d[4] =0
d[3] =1
d[2] =0
d[1] =0
d[0] =1
q[7:0] =11110000
q[7] =1
q[6] =1
q[5] =1
q[4] =1
q[3] =0
q[2] =0
q[1] =0
q[0] =0
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001


Signals
Waves
Time
clk =1
d[7:0] =11001001
d[7] =1
d[6] =1
d[5] =0
d[4] =0
d[3] =1
d[2] =0
d[1] =0
d[0] =1
q[7:0] =11001001
q[7] =1
q[6] =1
q[5] =0
q[4] =0
q[3] =1
q[2] =0
q[1] =0
q[0] =1
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
11111111
00000000
10101010
01010101
11110000
11001001