A logo with blue and green squares

Description automatically generatedUniversidade de Brasília – UNB

Curso: Engenharia de Redes de Comunicação  
Disciplina: Laboratório de Sistemas Digitais   
Turma: 08

Relatório da Disciplina Laboratório de Sistemas Digitais  
Tema: Experimento 06 – Circuitos Sequenciais

Aluno: Pedro Henrique Dias Avelar  
Matrícula: 241037112  
Professor: Eduardo Paiva

# Lista de Referências

[Figura 1:Resultado da simulação do flip-flop JK 6](#_Toc188103883)

[Figura 2: PRESET e CLEAR = 0; JK = 00 e 01 6](#_Toc188103884)

[Figura 3: PRESET e CLEAR = 0; JK = 10 e 11 6](#_Toc188103885)

[Figura 4: CLEAR = 1 e PRESET = 0 para todas as combinações de JK 7](#_Toc188103886)

[Figura 5:PRESET = 1 e CLEAR = 0 para todas as combinações de JK 7](#_Toc188103887)

[Figura 6:PRESET e CLEAR = 1 para todas as combinações de JK 7](#_Toc188103888)

[Figura 7: PRESET e CLEAR = 0 para todas as combinações de JK, com Q iniciando ativada 7](#_Toc188103889)

[Figura 8:Resultado da simulação do registrador de deslocamento 12](#_Toc188103890)

[Figura 9:RST = 0, LOAD = 1 para todas as combinações de DIR, L e R 12](#_Toc188103891)

[Figura 10: RST = 1 para todas as combinações de LOAD, DIR, L e R 13](#_Toc188103892)

[Figura 11:RST = 0, LOAD = 0, DIR = 0 para todas as combinações de L e R 14](#_Toc188103893)

[Figura 12:RST = 0, LOAD = 0, DIR = 1 para todas as combinações de L e R 14](#_Toc188103894)

[Figura 13:Deslocamentos para a esquerda quando L = 1 em azul e L = 0 em vermelho 14](#_Toc188103895)

[Figura 14:Deslocamentos para a direita quando R = 1 em azul e R = 0 em vermelho 14](#_Toc188103896)

[Tabela 1: Tabela-verdade do flip-flop JK 3](#_Toc188103900)

[Tabela 2:Tabela-verdade do registrador de deslocamento 8](#_Toc188103901)

[Tabela 3:Simplificação da Tabela 2 por meio de indução das entradas L e R 10](#_Toc188103902)

[Código 1: Modelagem do flip-flop JK 4](#_Toc188103910)

[Código 2: Testbench para o flip-flop JK 5](#_Toc188103911)

[Código 3:Modelagem do Registrador de Deslocamento Bidirecional 9](#_Toc188103912)

[Código 4: Testbench para o registrador de deslocamento 11](#_Toc188103913)

[Código 5:Teste da função LOAD 12](#_Toc188103914)

[Código 6:Teste da função RESET 13](#_Toc188103915)

[Código 7:Teste das funções de deslocamento para a esquerda e para a direita 14](#_Toc188103916)

# Introdução

O presente experimento tem os seguintes objetivos:

* Usar a estrutura “process” da linguagem VHDL
* Implementar e simular circuitos sequenciais

# Atividade 01

Usando a estrutura “process” da linguagem VHDL, implemente e simule no ModelSim um flip-flop JK gatilhado pela borda de subida com funcionamento descrito pela tabela 1. Q é a saída do flip-flop e as outras variáveis na tabela são entradas, CLK é o clock, os valores marcados como X não afetam o valor da saída.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PR | CLR | CLK | J | K | Q |
| 1 | X | X | X | X | 1 |
| 0 | 1 | X | X | X | 0 |
| 0 | 0 | Transição de 0 para 1 | 0 | 0 | Mantém |
| 0 | 0 | Transição de 0 para 1 | 0 | 1 | 0 |
| 0 | 0 | Transição de 0 para 1 | 1 | 0 | 1 |
| 0 | 0 | Transição de 0 para 1 | 1 | 1 | Inverte |
| 0 | 0 | Outros | X | X | Mantém |

Tabela 1: Tabela-verdade do flip-flop JK

Um flip-flop é um circuito que atua como uma unidade de memória, capaz de armazenar um bit. Seu funcionamento é similar ao de um latch, sendo a principal diferença que o flip-flop muda de estado apenas na borda de transição de um clock, enquanto o latch pode mudar de estado a qualquer momento em que estiver ativo.

O flip-flop JK possui 3 funções. Quando a entrada PR (Preset) está ativa, ela tem precedência sobre as demais entradas, e ativa a saída Q. Quando PR está inativa e a entrada CLR (Clear) está ativa, a saída Q é então desativada. Quando ambas as entradas PR e CLR estão inativas, então o flip-flop irá operar a cada borda de subida de clock, de acordo com as entradas J e K.

Para podermos simular este comportamento no ModelSim faz-se necessário o uso da estrutura “process”. Em uma linguagem de programação convencional, as linhas de código escritas são executadas em sequência. A linguagem VHDL foi criada para para poder documentar o funcionamento de circuitos digitais, e para estes circuitos comumente há processos concomitantes. A estrutura “process” serve para podermos modelar circuitos com esta característica. Embora o código VHDL seja escrito com suas declarações em sequência, o código escrito dentro de uma estrutura “process” terá seu processamento realizado simultaneamente no momento da execução do código.

Na linha 21 do código 1 iniciamos a estrutura “process” com os parâmetros PRESET, CLR e CLK. Estes parâmetros irão compor a lista de sensitividade do processo. Isto significa que, a qualquer mudança em um desses parâmetros, todas as declarações dentro do processo serão executadas simultaneamente. Com o uso das estruturas de condição IF e ELSIF nas linhas 23, 24 e 25, é possível então garantir que o flip-flop tenha o comportamento descrito pela tabela verdade – isto é, a precedência da operação PRESET, seguida pela operação CLEAR e por fim então o comportamento de acordo com as entradas J e K para o caso de PRESET e CLEAR estarem desativadas.

Código 1: Modelagem do flip-flop JK

01 -- Experimento 06 - Questão 01

02 -- Aluno: Pedro Henrique Dias Avelar 241037112

03 -- Turma 08

04 -- Data: 08/01/2025

05

06 -- Flip-Flop JK

07

08 **LIBRARY** IEEE**;**

09 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

10

11 **ENTITY** FLIP\_FLOP\_JK **IS**

12 **PORT** **(**PRESET**,** CLR**,** CLK**,** J**,** K**:** **IN** STD\_LOGIC**;**

13 Q**:** **OUT** STD\_LOGIC**);**

14 **END** FLIP\_FLOP\_JK**;**

15

16 **ARCHITECTURE** ARC\_FLIP\_FLOP\_JK **OF** FLIP\_FLOP\_JK **IS**

17 **SIGNAL** GDA\_Q **:** STD\_LOGIC **:=** '0'**;**

18 **SIGNAL** JK **:** STD\_LOGIC\_VECTOR**(**1 **DOWNTO** 0**);**

19 **BEGIN**

20 JK **<=** J **&** K**;**

21 **PROCESS(**PRESET**,**CLR**,**CLK**)**

22 **BEGIN**

23 **IF** PRESET **=** '1' **THEN** GDA\_Q **<=** '1'**;**

24 **ELSIF** CLR **=** '1' **THEN** GDA\_Q **<=** '0'**;**

25 **ELSIF** **RISING\_EDGE(**CLK**)** **THEN**

26 **CASE** JK **IS**

27 **WHEN** "00" **=>** **NULL;**

28 **WHEN** "01" **=>** GDA\_Q **<=** '0'**;**

29 **WHEN** "10" **=>** GDA\_Q **<=** '1'**;**

30 **WHEN** "11" **=>** GDA\_Q **<=** **NOT(**GDA\_Q**);**

31 **WHEN** **OTHERS** **=>** **NULL;**

32 **END** **CASE;**

33 **END** **IF;**

34 **END** **PROCESS;**

35 Q **<=** GDA\_Q**;**

36 **END** ARC\_FLIP\_FLOP\_JK**;**

Para o caso em que PRESET e CLEAR estejam desativadas e J e K estejam ativadas, o comportamento esperado é a inversão do sinal de Q. No entanto, na linha 13 do código 1, Q foi declarada como saída da entidade FLIP\_FLOP\_JK. Por conta disso, seu valor não pode ser lido dentro do “process”; uma saída pode apenas ter um novo valor atribuído. Para contornar essa limitação, fez-se necessário então criar um sinal de guarda, GDA\_Q, na linha 17. Com isso então, utilizamos esse sinal GDA\_Q dentro do “process” e, ao final, repassamos então o sinal para a saída Q. Por uma limitação da linguagem VHDL é necessário inicializar o valor deste sinal para podermos simular o circuito.

Para podermos simular o comportamento da entidade FLIP\_FLOP\_JK, foi montado o seguinte testbench:

Usando duas estruturas de repetição FOR aninhadas, podemos iterar as quatro combinações de valores das entradas PRESET e CLEAR. Usando uma variável do tipo STD\_LOGIC nos loops FOR, podemos usar essas variáveis diretamente para atribuir valores para PRESET e CLEAR, sem precisar de nenhum tipo de conversão (linhas 36 e 38 do código 2). Para cada iteração dentro dos FOR aninhados, passamos então pelas 4 combinações possíveis das entradas J e K. Com o clock configurado com um período de 10ns (5ns para cada inversão de clock) e a simulação aguardando um período de 20ns para cada mudança de valor das entradas J e K, então cada valor de J e K passará por duas bordas de subida, facilitando a visualização das transições.

Código 2: Testbench para o flip-flop JK

01 -- Experimento 06 - Questão 01 - TESTBENCH

02 -- Aluno: Pedro Henrique Dias Avelar 241037112

03 -- Turma 08

04 -- Data: 08/01/2025

05

06 -- Testbench - Tempo de simulação: 400ns

07

08 **LIBRARY** IEEE**;**

09 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

10 **USE** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

11 **USE** IEEE**.**NUMERIC\_STD**.ALL;**

12

13 **ENTITY** TESTBENCH\_Q01 **IS**

14 **END** TESTBENCH\_Q01**;**

15

16 **ARCHITECTURE** ARC\_TESTBENCH\_Q01 **OF** TESTBENCH\_Q01 **IS**

17

18 **COMPONENT** FLIP\_FLOP\_JK **IS**

19 **PORT** **(**PRESET**,** CLR**,** CLK**,** J**,** K**:** **IN** STD\_LOGIC**;**

20 Q**:** **OUT** STD\_LOGIC**);**

21 **END** **COMPONENT;**

22

23 **SIGNAL** JK\_TB**:** STD\_LOGIC\_VECTOR**(**1 **DOWNTO** 0**);**

24 **SIGNAL** Q\_TB**:** STD\_LOGIC**;**

25 **SIGNAL** PRESET\_TB**:** STD\_LOGIC**;**

26 **SIGNAL** CLEAR\_TB **:** STD\_LOGIC**;**

27 **SIGNAL** CLOCK\_TB **:** STD\_LOGIC **:=** '0'**;**

28

29 **BEGIN**

30 DUT**:** FLIP\_FLOP\_JK **PORT** **MAP** **(**PRESET\_TB**,** CLEAR\_TB**,** CLOCK\_TB**,** JK\_TB**(**1**),** JK\_TB**(**0**),** Q\_TB**);**

31 CLOCK\_TB **<=** **NOT(**CLOCK\_TB**)** **AFTER** 5 NS**;**

32 **PROCESS**

33 **BEGIN**

34 **REPORT** "INICIANDO TESTE..."**SEVERITY** NOTE**;**

35 **FOR** I **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

36 PRESET\_TB **<=** I**;**

37 **FOR** J **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

38 CLEAR\_TB **<=** J**;**

39 JK\_TB **<=** "00"**;**

40 **WAIT** **FOR** 20 NS**;**

41 JK\_TB **<=** "01"**;**

42 **WAIT** **FOR** 20 NS**;**

43 JK\_TB **<=** "10"**;**

44 **WAIT** **FOR** 20 NS**;**

45 JK\_TB **<=** "11"**;**

46 **WAIT** **FOR** 20 NS**;**

47 **END** **LOOP;**

48 **END** **LOOP;**

49 PRESET\_TB **<=** '0'**;**

50 CLEAR\_TB **<=** '0'**;**

51 JK\_TB **<=** "00"**;**

52 **WAIT** **FOR** 20 NS**;**

53 JK\_TB **<=** "01"**;**

54 **WAIT** **FOR** 20 NS**;**

55 JK\_TB **<=** "10"**;**

56 **WAIT** **FOR** 20 NS**;**

57 JK\_TB **<=** "11"**;**

58 **WAIT** **FOR** 20 NS**;**

59

60 **REPORT** "TESTE FINALIZADO!" **SEVERITY** NOTE**;**

61 **WAIT;**

62 **END** **PROCESS;**

63 1END ARC\_TESTBENCH\_Q01**;**

A simulação foi realizada com um período de 400ns. Após a iteração por todos as combinações de PRESET e CLEAR, foi realizada mais uma iteração dos valores J e K, mas desta vez com PRESET e CLEAR desativadas e Q iniciando ativada. A simulação gerou o seguinte resultado:

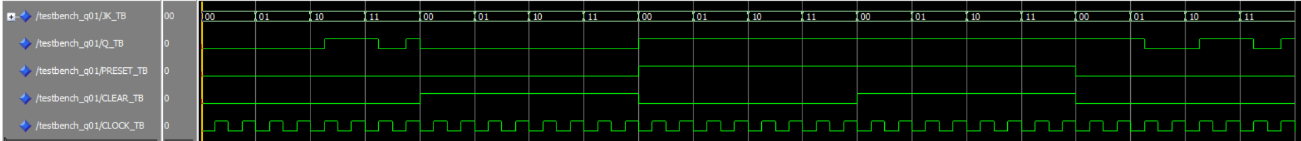


Figura 1:Resultado da simulação do flip-flop JK

O primeiro cenário da simulação são as entradas PRESET e CLEAR desativadas, de modo que a saída Q dependerá então dos valores de J e K.



Figura 2: PRESET e CLEAR = 0; JK = 00 e 01

Podemos ver que não houve alteração da saída Q para os cenários em que JK = 00 (Valor de Q se mantém) e JK = 01 (Q = 0)

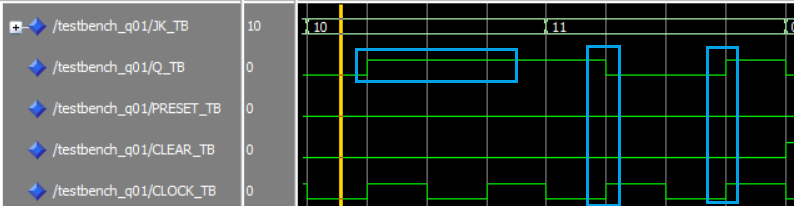


Figura 3: PRESET e CLEAR = 0; JK = 10 e 11

Na primeira borda de subida de borda do clock com JK = 10, a saída Q foi ativada, e se manteve na segunda borda subida de clock. Já com JK = 11, a saída Q teve seu sinal invertido em cada uma das duas bordas de subida de clock.

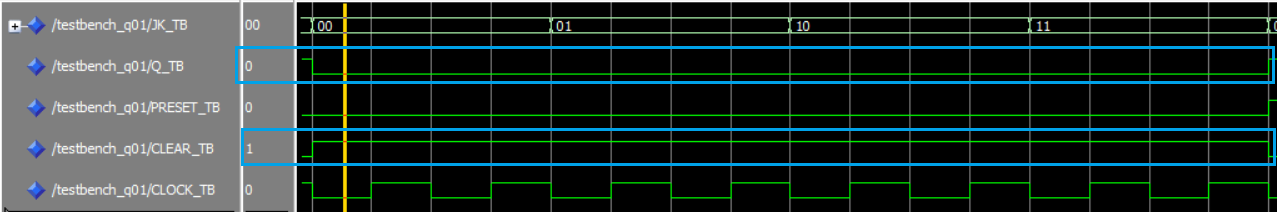


Figura 4: CLEAR = 1 e PRESET = 0 para todas as combinações de JK

O segundo cenário é a entrada CLEAR ativada e a entrada PRESET desativada. Neste cenário, a saída Q deve ser desativada independentemente dos valores das entradas J e K.



Figura 5:PRESET = 1 e CLEAR = 0 para todas as combinações de JK



Figura 6:PRESET e CLEAR = 1 para todas as combinações de JK

O terceiro cenário é a entrada PRESET ativada. Neste cenário a entrada Q deve ser ativada independente do valor das demais entradas, inclusive quando CLEAR também estiver ativa.

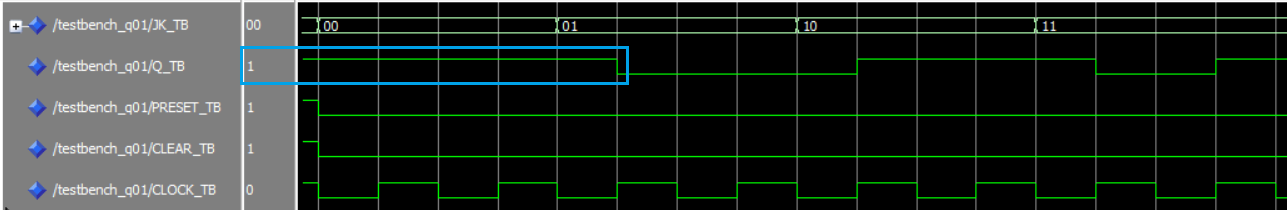


Figura 7: PRESET e CLEAR = 0 para todas as combinações de JK, com Q iniciando ativada

Por fim, repetimos o primeiro cenário com PRESET e CLEAR desativada, porém com a entrada Q iniciando ativada. Podemos ver que ao diferentemente da figura 2, neste cenário a saída Q permanece ativa nas subidas de borda do clock com JK = 0. Apenas na borda de subida de clock com JK = 01 a saída Q é então desativada, tendo então o restante do comportamento igual ao demonstrado nas figuras 2 e 3.

Por meio das figuras 2 a 7 podemos ver então que a entidade FLIP\_FLOP\_JK do código 1 conseguiu atender a tabela-verdade apresentada na tabela 1.

# Atividade 02

Usando a estrutura “process”, implemente em VHDL e simule no ModelSim um registrador de deslocamento bidirecional de 4 bits com funcionamento descrito pela tabela 2. Q é a saída e as outras variáveis na tabela são entradas, CLK é o clock e o X indica que o valor daquela entrada não afeta a saída. Respeite a ordem de significância dos bits dos vetores: o bit mais a direita é o menos significativo.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CLK | RST | LOAD | D | DIR | L | R | Q |
| Transição de 0 para 1 | 1 | X | XXXX | X | X | X | 0000 |
| Transição de 0 para 1 | 0 | 1 | D3D2D1D0 | X | X | X | D3D2D1D0 |
| Transição de 0 para 1 | 0 | 0 | XXXX | 0 | 0 | X | Q2Q1Q00 |
| Transição de 0 para 1 | 0 | 0 | XXXX | 0 | 1 | X | Q2Q1Q01 |
| Transição de 0 para 1 | 0 | 0 | XXXX | 1 | X | 0 | 0Q3Q2Q1 |
| Transição de 0 para 1 | 0 | 0 | XXXX | 1 | X | 1 | 1 Q3Q2Q1 |
| Outros | X | X | XXXX | X | X | X | mantém |

Tabela 2:Tabela-verdade do registrador de deslocamento

O registrador de deslocamento consiste em um circuito registrador, que por sua vez consiste em um conjunto de flip-flops, capaz de deslocar o valor armazenado por seus flip-flops para a esquerda ou para a direita, de acordo com suas entradas. Conforme a tabela 2, podemos ver que o registrador de deslocamento solicitado possui 3 funções. A função RESET, acionada quando a entrada RST está ativa na borda de subida do clock, reseta a saída Q, desativando todos os seus bits. A função LOAD, acionada quando a entrada RST está desativada e a entrada LOAD está ativa, carrega a saída Q com a entrada D; esta função é comumente chamada de carga paralela ou “parallel load”. E por fim, com as entradas RST e LOAD desativadas, o circuito então irá fazer a operação de deslocamento, de acordo com as entradas DIR e L/R.

Para DIR desativada, o deslocamento será feito para a esquerda – isto é, cada um dos bits de Q será deslocado uma posição a esquerda, e o bit menos significativo será carregado com o valor da entrada L. Para dor ativada, o deslocamento será então feito para a direita, isto é, cada um dos bits de Q será deslocado uma posição a direita e o bit mais significativo será carregado com o valor de R. Para o circuito proposto, só deve haver mudança nos momentos de subida de borda do clock; para qualquer outra situação, a saída Q deve permanecer a mesma.

Assim, podemos então modelar o registrador de deslocamento proposto com o seguinte código:

O registrador de deslocamento bidirecional poderia ser modelado utilizando a entidade FLIP\_FLOP\_JK do código 1, de modo a se aproximar mais da estrutura real do registrador de deslocamento; no entanto, modelar fidedignamente o comportamento do registrador tem grande utilidade para a geração de um “golden module” como o modelado no experimento 5 – isto é, um módulo com acurácia perfeita e que assim possa ser usado para testar a acurácia de outras modelagens do mesmo circuito.

Código 3:Modelagem do Registrador de Deslocamento Bidirecional

01 -- Experimento 06 - Questão 02

02 -- Aluno: Pedro Henrique Dias Avelar 241037112

03 -- Turma 08

04 -- Data: 12/01/2025

05

06 -- Registrador de Deslocamento

07

08 **LIBRARY** IEEE**;**

09 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

10

11 **ENTITY** REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL **IS**

12 **PORT** **(**CLK**,** RST**,** LOAD**,** DIR**,** L**,** R**:** **IN** STD\_LOGIC**;**

13 D**:** **IN** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**);**

14 Q**:** **OUT** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**));**

15 **END** REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL**;**

16

17 **ARCHITECTURE** ARC\_REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL **OF** REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL **IS**

18

19 **SIGNAL** GDA\_Q **:** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**)** **:=** "0000"**;**

20

21 **BEGIN**

22 **PROCESS(**CLK**)**

23 **BEGIN**

24 **IF** **RISING\_EDGE(**CLK**)** **THEN**

25 **IF** RST **=** '1' **THEN** GDA\_Q **<=** "0000"**;**

26 **ELSIF** LOAD **=** '1' **THEN** GDA\_Q **<=** D**;**

27 **ELSE** **CASE** DIR **IS**

28 **WHEN** '0' **=>** GDA\_Q **<=** GDA\_Q**(**2**)** **&** GDA\_Q**(**1**)** **&** GDA\_Q**(**0**)** **&** L**;**

29 **WHEN** '1' **=>** GDA\_Q **<=** R **&** GDA\_Q**(**3**)** **&** GDA\_Q**(**2**)** **&** GDA\_Q**(**1**);**

30 **WHEN** **OTHERS** **=>** **NULL;**

31 **END** **CASE;**

32 **END** **IF;**

33 **END** **IF;**

34 **END** **PROCESS;**

35 Q **<=** GDA\_Q**;**

36 **END** ARC\_REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL**;**

De forma similar a modelagem feita no código 1, novamente usamos a estrutura “process” para atender a funcionalidade da tabela 2. Neste caso, como o circuito executa suas funções apenas nas bordas de subida do clock, apenas a entrada CLK precisa estar presente na lista de sensibilidade do processo. Similar também ao que foi feito no código 1, na linha 19 declaramos um sinal contendo um vetor de 4 bits GDA\_Q, para poder “guardar” o valor de Q. Com uma sequência de IFs e ELSIFs iniciada na linha 27 definimos então o comportamento do circuito para atender a tabela 2. As declarações dentro do CASE (linhas 29-33) consistem em uma simples indução das entradas L e R na tabela 2. Podemos simplificar a tabela 2, fazendo essas induções, para:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CLK | RST | LOAD | D | DIR | L | R | Q |
| Transição de 0 para 1 | 1 | X | XXXX | X | X | X | 0000 |
| Transição de 0 para 1 | 0 | 1 | D3D2D1D0 | X | X | X | D3D2D1D0 |
| Transição de 0 para 1 | 0 | 0 | XXXX | 0 | 0/1 | X | Q2Q1Q0L |
| Transição de 0 para 1 | 0 | 0 | XXXX | 1 | X | 0/1 | R Q3Q2Q1 |
| Outros | X | X | XXXX | X | X | X | mantém |

Tabela 3:Simplificação da Tabela 2 por meio de indução das entradas L e R

Ou seja, quando RST, LOAD e DIR estiverem desativadas, Q terá seus bits deslocados a esquerda e o bit menos significativo será preenchido com o valor de L; já quando RST e LOAD estiverem desativadas e DIR ativada, Q terá seus bits deslocados a direita e o bit mais significativo será preenchido com o valor de R.

Para simular o funcionamento do registrador de deslocamento, foi preparado o seguinte testbench:

A simulação teve o seguinte resultado:

Código 4: Testbench para o registrador de deslocamento

01 -- Experimento 06 - Questão 01 - TESTBENCH

02 -- Aluno: Pedro Henrique Dias Avelar 241037112

03 -- Turma 08

04 -- Data: 12/01/2025

05

06 -- Testbench - tempo de simulação: 1160ns

07

08 **LIBRARY** IEEE**;**

09 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

10 **USE** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

11 **USE** IEEE**.**NUMERIC\_STD**.ALL;**

12

13 **ENTITY** TESTBENCH\_Q02 **IS**

14 **END** TESTBENCH\_Q02**;**

15

16 **ARCHITECTURE** ARC\_TESTBENCH\_Q02 **OF** TESTBENCH\_Q02 **IS**

17

18 **COMPONENT** REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL **IS**

19 **PORT** **(**CLK**,** RST**,** LOAD**,** DIR**,** L**,** R**:** **IN** STD\_LOGIC**;**

20 D**:** **IN** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**);**

21 Q**:** **OUT** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**));**

22 **END** **COMPONENT;**

23

24 **SIGNAL** D\_TB**:** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**);**

25 **SIGNAL** Q\_TB**:** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**);**

26 **SIGNAL** CLOCK\_TB **:** STD\_LOGIC **:=** '0'**;**

27 **SIGNAL** RLD\_TB**:** STD\_LOGIC\_VECTOR**(**2 **DOWNTO** 0**);**

28 **SIGNAL** LR\_TB**:** STD\_LOGIC\_VECTOR**(**1 **DOWNTO** 0**);**

29

30 **BEGIN**

31 DUT**:** REGISTRADOR\_DESLOCAMENTO\_BIDIRECIONAL

32 **PORT** **MAP** **(**CLOCK\_TB**,** RLD\_TB**(**2**),** RLD\_TB**(**1**),** RLD\_TB**(**0**),** LR\_TB**(**1**),** LR\_TB**(**0**),** D\_TB**,** Q\_TB**);**

33 CLOCK\_TB **<=** **NOT(**CLOCK\_TB**)** **AFTER** 5 NS**;**

34 **PROCESS**

35 **BEGIN**

36 **REPORT** "INICIANDO TESTE..."**SEVERITY** NOTE**;**

37

38 -- Teste 1: LOAD

39 D\_TB **<=** "0101"**;** RLD\_TB **<=** "010"**;** LR\_TB **<=** "00"**;**

40 **WAIT** **FOR** 20 NS**;**

41 **FOR** I **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

42 RLD\_TB**(**0**)** **<=** I**;**

43 **FOR** K **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

44 LR\_TB**(**1**)** **<=** K**;**

45 **FOR** L **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

46 LR\_TB**(**0**)** **<=** L**;**

47 **WAIT** **FOR** 20 NS**;**

48 **END** **LOOP;**

49 **END** **LOOP;**

50 **END** **LOOP;**

51

52 -- Teste 2: RESET

53 RLD\_TB **<=** "100"**;** LR\_TB **<=** "00"**;**

54 **WAIT** **FOR** 20 NS**;**

55 **FOR** I **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

56 RLD\_TB**(**1**)** **<=** I**;**

57 **FOR** J **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

58 RLD\_TB**(**0**)** **<=** J**;**

59 **FOR** K **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

60 LR\_TB**(**1**)** **<=** K**;**

61 **FOR** L **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

62 LR\_TB**(**0**)** **<=** L**;**

63 **WAIT** **FOR** 20 NS**;**

64 **END** **LOOP;**

65 **END** **LOOP;**

66 **END** **LOOP;**

67 **END** **LOOP;**

68

69 -- Teste 3a: Deslocamento para a esquerda

70 RLD\_TB **<=** "000"**;**

71 LR\_TB **<=** "10"**;** -- Deslocamento com uns

72 **WAIT** **FOR** 80 NS**;**

73 LR\_TB **<=** "00"**;** -- Deslocamento com zeros

74 **WAIT** **FOR** 80 NS**;**

75 LR\_TB **<=** "11"**;** -- Deslocamento com uns

76 **WAIT** **FOR** 80 NS**;**

77 LR\_TB **<=** "01"**;** -- Deslocamento com zeros

78 **WAIT** **FOR** 80 NS**;**

79

80 -- Teste 3b: Deslocamento para a direita

81 RLD\_TB **<=** "001"**;**

82 LR\_TB **<=** "01"**;** -- Deslocamento com uns

83 **WAIT** **FOR** 80 NS**;**

84 LR\_TB **<=** "00"**;** -- Deslocamento com zeros

85 **WAIT** **FOR** 80 NS**;**

86 LR\_TB **<=** "11"**;** -- Deslocamento com uns

87 **WAIT** **FOR** 80 NS**;**

88 LR\_TB **<=** "10"**;** -- Deslocamento com zeros

89 **WAIT** **FOR** 80 NS**;**

90

91

92 **REPORT** "TESTE FINALIZADO!" **SEVERITY** NOTE**;**

93 **WAIT;**

94 **END** **PROCESS;**

95 END ARC\_TESTBENCH\_Q02**;**

A black screen with green lines

Description automatically generated

Figura 8:Resultado da simulação do registrador de deslocamento

Para facilitar a visualização e a completitude dos testes, o testbench foi dividido em 3 partes, cada um representando uma função do registrador de deslocamento. A primeira função testada foi a função LOAD, no código 4 entre as linhas 38 e 50.

38 -- Teste 1: LOAD

39 D\_TB **<=** "0101"**;** RLD\_TB **<=** "010"**;** LR\_TB **<=** "00"**;**

40 **WAIT** **FOR** 20 NS**;**

41 **FOR** I **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

42 RLD\_TB**(**0**)** **<=** I**;**

43 **FOR** K **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

44 LR\_TB**(**1**)** **<=** K**;**

45 **FOR** L **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

46 LR\_TB**(**0**)** **<=** L**;**

47 **WAIT** **FOR** 20 NS**;**

48 **END** **LOOP;**

49 **END** **LOOP;**

50 **END** **LOOP;**

Código 5:Teste da função LOAD

De acordo com a tabela 3, para que o circuito realize a função de leitura paralela, é necessário que a entrada RST esteja desativada e a entrada LOAD esteja ativa. As entradas DIR, L e R não importam neste cenário. Assim, fazemos a iteração do bit menos significativo do sinal RLD\_TB, o qual foi mapeado a entrada DIR, e do sinal LR\_TB, que foi mapeado as entradas L e R, de modo a cobrir todos os cenários possíveis de LOAD.

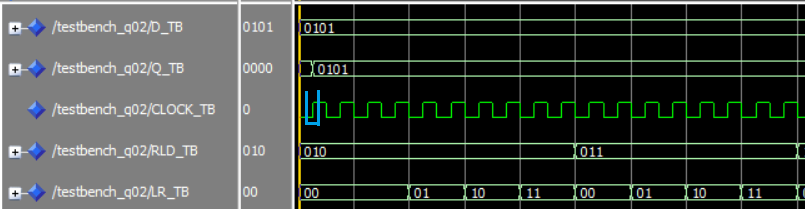


Figura 9:RST = 0, LOAD = 1 para todas as combinações de DIR, L e R

Na primeira borda de subida do clock, o valor de D é carregado na saída Q. Nenhum deslocamento foi realizado independente dos valores de DIR, L e R, respeitando a tabela 2.

A segunda função testada foi a função RESET, que zera o valor do registrador. O teste para esta função está entre as linhas 52 e 67 do código 4 .

Código 6:Teste da função RESET

52 -- Teste 2: RESET

53 RLD\_TB **<=** "100"**;** LR\_TB **<=** "00"**;**

54 **WAIT** **FOR** 20 NS**;**

55 **FOR** I **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

56 RLD\_TB**(**1**)** **<=** I**;**

57 **FOR** J **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

58 RLD\_TB**(**0**)** **<=** J**;**

59 **FOR** K **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

60 LR\_TB**(**1**)** **<=** K**;**

61 **FOR** L **IN** STD\_LOGIC **RANGE** '0' **TO** '1' **LOOP**

62 LR\_TB**(**0**)** **<=** L**;**

63 **WAIT** **FOR** 20 NS**;**

64 **END** **LOOP;**

65 **END** **LOOP;**

66 **END** **LOOP;**

67 **END** **LOOP;**

De acordo com a tabela 2, estando a entrada RST ativa, o circuito deve zerar a saída Q independente dos valores das entradas LOAD, DIR, L e R. Para isso então usamos 4 estruturas de repetição FOR aninhadas para iterar por todas as combinações possíveis destas entradas. Seria possível utilizar menos estruturas FOR com o uso de variáveis para fazer as iterações, porém como são poucas combinações possíveis, fazer a iteração usando o tipo de dado STD\_LOGIC simplifica a montagem do teste. Para um cenário com mais variáveis se tornaria inviável montar o teste assim devido ao grande número de loops FOR necessários.

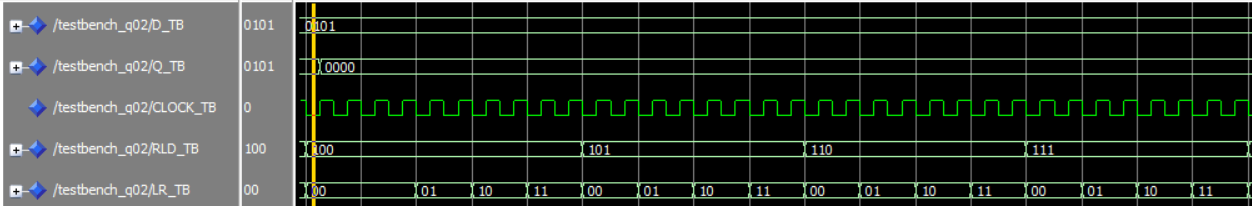


Figura 10: RST = 1 para todas as combinações de LOAD, DIR, L e R

Pela figura 10 podemos ver que na primeira borda de subida de clock após a ativação da entrada RST, o valor da saída Q é zerado e assim permanece por todas as combinações de LOAD, DIR, L e R.

A terceira função é a função de deslocamento, que pode ser subdividida em 2 – deslocamento para a esquerda e deslocamento para a direita. Este deslocamento pode ser com a inserção de 0s ou 1s dependendo do valor de L para deslocamento para a esquerda ou R para deslocamento para a direita. O teste para estas funções está entre as linhas 69 e 89 do código 4. Os valores de L e R não importam quando o deslocamento é feito para o sentido contrário (L para deslocamento para a direita e R para deslocamento para a esquerda). Assim, para este cenário, como as entradas RST e LOAD precisam estar obrigatoriamente desativadas, precisamos apenas iterar pelos valores de L e R para os cenários de deslocamento para a esquerda (DIR = 0) e deslocamento para a direita (DIR = 1) para atender todos os cenários possíveis da tabela 2.

Código 7:Teste das funções de deslocamento para a esquerda e para a direita

69 -- Teste 3a: Deslocamento para a esquerda

70 RLD\_TB **<=** "000"**;**

71 LR\_TB **<=** "10"**;** -- Deslocamento com uns

72 **WAIT** **FOR** 80 NS**;**

73 LR\_TB **<=** "00"**;** -- Deslocamento com zeros

74 **WAIT** **FOR** 80 NS**;**

75 LR\_TB **<=** "11"**;** -- Deslocamento com uns

76 **WAIT** **FOR** 80 NS**;**

77 LR\_TB **<=** "01"**;** -- Deslocamento com zeros

78 **WAIT** **FOR** 80 NS**;**

79

80 -- Teste 3b: Deslocamento para a direita

81 RLD\_TB **<=** "001"**;**

82 LR\_TB **<=** "01"**;** -- Deslocamento com uns

83 **WAIT** **FOR** 80 NS**;**

84 LR\_TB **<=** "00"**;** -- Deslocamento com zeros

85 **WAIT** **FOR** 80 NS**;**

86 LR\_TB **<=** "11"**;** -- Deslocamento com uns

87 **WAIT** **FOR** 80 NS**;**

88 LR\_TB **<=** "10"**;** -- Deslocamento com zeros

89 **WAIT** **FOR** 80 NS**;**

A screen shot of a screen

Description automatically generated

Figura 11:RST = 0, LOAD = 0, DIR = 0 para todas as combinações de L e R

A screen shot of a screen

Description automatically generated

Figura 12:RST = 0, LOAD = 0, DIR = 1 para todas as combinações de L e R

Podemos ver que nas figura 11 e 12 a saída Q é convertida de 0000 para 1111 e de 1111 para 0000 de acordo com os valores de L na figura 11 e R na figura 12. Expandindo o zoom fica mais visível esta alteração gradual devido ao deslocamento:

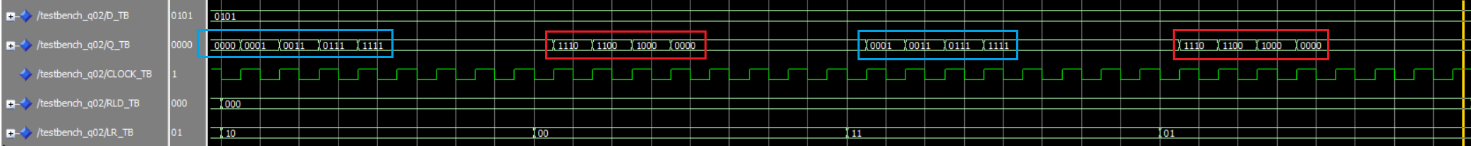


Figura 13:Deslocamentos para a esquerda quando L = 1 em azul e L = 0 em vermelho

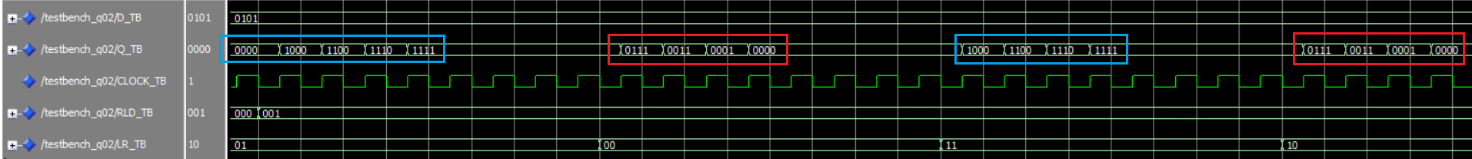


Figura 14:Deslocamentos para a direita quando R = 1 em azul e R = 0 em vermelho

Pelas figuras 13 e 14, podemos ver que os deslocamentos para esquerda e direita ocorrem de maneira idêntica independente do valor de R no deslocamento para a esquerda e do valor de L no deslocamento para a direita, cumprindo assim o comportamento descrito pela tabela 2. Assim, com esses 3 cenários foi possível atender aos requisitos da tabela 2 para o funcionamento do registrador de deslocamento.