

**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS –
CEFET/MG**

**DIOGO EMANUEL ANTUNES SANTOS
GUILHERME AUGUSTO DE OLIVEIRA
LUIZ EDUARDO LEROY SOUZA**

Circuito Sequencial — Registrador de deslocamento
Relatório 8

**BELO HORIZONTE
2021**

Sumário

Objetivos	2
Desenvolvimento	2
2.1 Tabela verdade	2
2.2 Expressões lógicas	2
2.3 Descrição em Verilog HDL	3
2.4 Circuito lógico	5
2.5 Simulação do circuito no ModelSim	7
Simulação do circuito bloqueante	8
Simulação do circuito não bloqueante	9
Análise dos resultados:	10
Referências	11

1. Objetivos

O objetivo do relatório é criar um circuito sequencial utilizando o modelo de descrição de hardware comportamental, que tem como funcionalidade realizar o deslocamento dos bits de um número de quatro posições.

2. Desenvolvimento

O flip-flop é uma ferramenta extremamente versátil e pode ser utilizada em diversos casos que envolvem a lógica digital. Um dos usos mais comuns associados a eles é o armazenamento de dados. Para suprir esses casos normalmente é feita a composição de grupos de flip-flops denominados registradores, que são os alvos do presente relatório.

De acordo com Tocci, “A operação mais comum realizada sobre os dados armazenados em FFs ou registradores é a transferência de dados, que envolve a transferência de dados de um FF ou registrador para outro”. Tal operação geralmente é implementada utilizando um registrador de deslocamento, que funciona de forma serial e é construído por meio de um grupo de flip-flops capaz de deslocar números binários armazenados continuamente para os FFs (flip-flops) seguintes, a cada passagem do fluxo de clock.

Assim, por meio da linguagem Verilog, serão demonstrados os principais passos realizados para sintetizar dois registradores de deslocamento, com sentido da esquerda para a direita. O primeiro deles utiliza um modo bloqueador de deslocar os bits, enquanto o segundo é basicamente uma refatoração simples do primeiro registrador, com o intuito de utilizar a lógica sequencial de forma correta e deslocar os bits de forma não bloqueante.

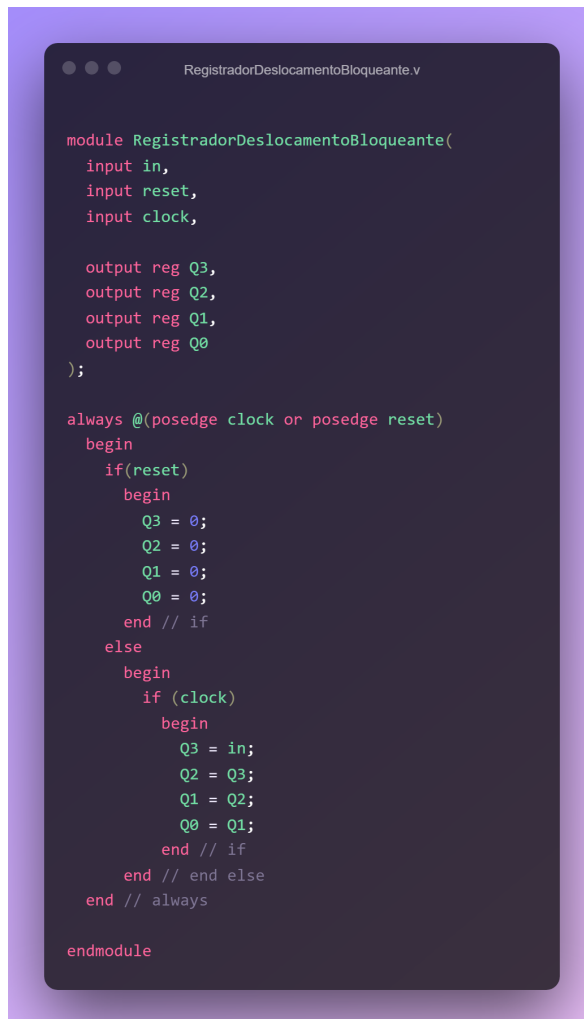
2.1 Tabela verdade

Para esse projeto não foram desenvolvidas tabelas verdades, uma vez que o método utilizado para criar o código em verilog e consequentemente sintetizar o circuito utilizando o Quartus foi o modelo comportamental. Tal modelo utiliza blocos de código sequenciais, e sintetiza o projeto por meio da descrição de como o circuito deve se comportar mediante determinadas situações.

2.2 Expressões lógicas

Assim como a questão da tabela verdade, não foram utilizadas expressões lógicas para descrever o circuito, tendo em vista que não existem tabelas para extrair tais expressões.

2.3 Descrição em Verilog HDL



```
module RegistradorDeslocamentoBloqueante(
    input in,
    input reset,
    input clock,

    output reg Q3,
    output reg Q2,
    output reg Q1,
    output reg Q0
);

always @(posedge clock or posedge reset)
begin
    if(reset)
    begin
        Q3 = 0;
        Q2 = 0;
        Q1 = 0;
        Q0 = 0;
    end // if
    else
    begin
        if (clock)
        begin
            Q3 = in;
            Q2 = Q3;
            Q1 = Q2;
            Q0 = Q1;
        end // if
    end // end else
end // always

endmodule
```

Imagem 01: Representação em Verilog HDL do módulo registrador bloqueante

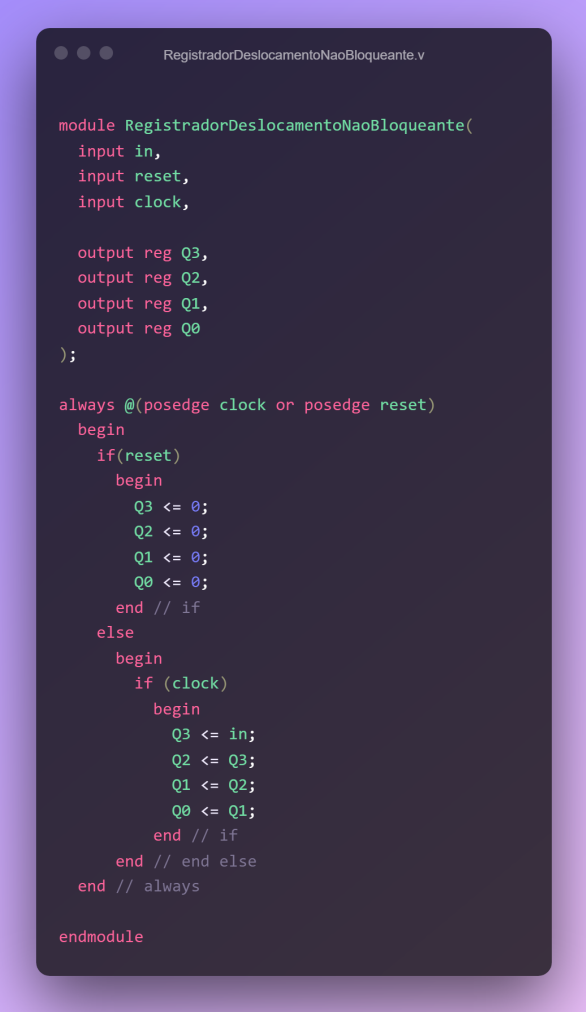
Na imagem anterior é possível compreender o funcionamento do circuito a ser sintetizado. Conforme é possível observar, o mesmo contém três entradas:

- *in*: é o bit que entra no circuito a partir de um momento posterior aos resets e será deslocado pelos quatro flip-flops;
- *reset*: é a entrada responsável por reiniciar o circuito e zerar as quatro saídas;
- *clock*: é a entrada utilizada para representar a passagem de tempo no circuito e indica os intervalos de funcionamento/armazenamento de dados.

E conta também com quatro saídas, que são as saídas de cada flip-flop, representadas de Q3 a Q0.

O código descrito em Verilog HDL utiliza um bloco *always* que tem como dependência as entradas *clock* e *reset*. Ambos ativam o bloco quando passam por uma borda de subida. Caso o *reset* esteja ligado (bit com valor 1), as quatro saídas são zeradas. Caso contrário, se o *clock* estiver igual a 1, os bits são deslocados para o flip-flop à sua direita.

Vê-se pela construção da última parte que a atribuição ocorre com o operador bloqueador (=). Dessa maneira, a instrução $Q2 = Q3$ só será acionada após a instrução $Q3 = in$. Resumidamente, isso fará com que todos os flip-flops assumam saída igual ao bit de entrada in .

A screenshot of a code editor window titled "RegistradorDeslocamentoNaoBloqueante.v" showing Verilog HDL code. The code defines a module with inputs in, reset, and clock, and outputs reg Q3, Q2, Q1, and Q0. It uses a non-blocking assignment operator (<=) within an always block triggered by the clock or reset signal. The logic implements a right-to-left shift register where each output is updated with the value of the next output in the chain during the next clock edge.

```
module RegistradorDeslocamentoNaoBloqueante(
    input in,
    input reset,
    input clock,

    output reg Q3,
    output reg Q2,
    output reg Q1,
    output reg Q0
);

always @(posedge clock or posedge reset)
begin
    if(reset)
    begin
        Q3 <= 0;
        Q2 <= 0;
        Q1 <= 0;
        Q0 <= 0;
    end // if
    else
    begin
        if (clock)
        begin
            Q3 <= in;
            Q2 <= Q3;
            Q1 <= Q2;
            Q0 <= Q1;
        end // if
    end // end else
end // always

endmodule
```

Imagem 02: Representação em Verilog HDL do módulo registrador não bloqueante

A imagem acima representa a versão do registrador de deslocamento que utiliza a atribuição não bloqueada (<=). Esta é a única mudança em relação ao código anterior. Basicamente, tal alteração permite que todos os deslocamentos ocorram na mesma passagem de *clock* e sejam executados simultaneamente, assim os bits serão deslocados da esquerda para a direita corretamente (direção Q3 -> Q0).

- Q3 recebe a entrada in ;
- Q2 recebe o valor de Q3;
- Q1 recebe o valor de Q2;
- Q0 recebe o valor de Q1.

2.4 Circuito lógico

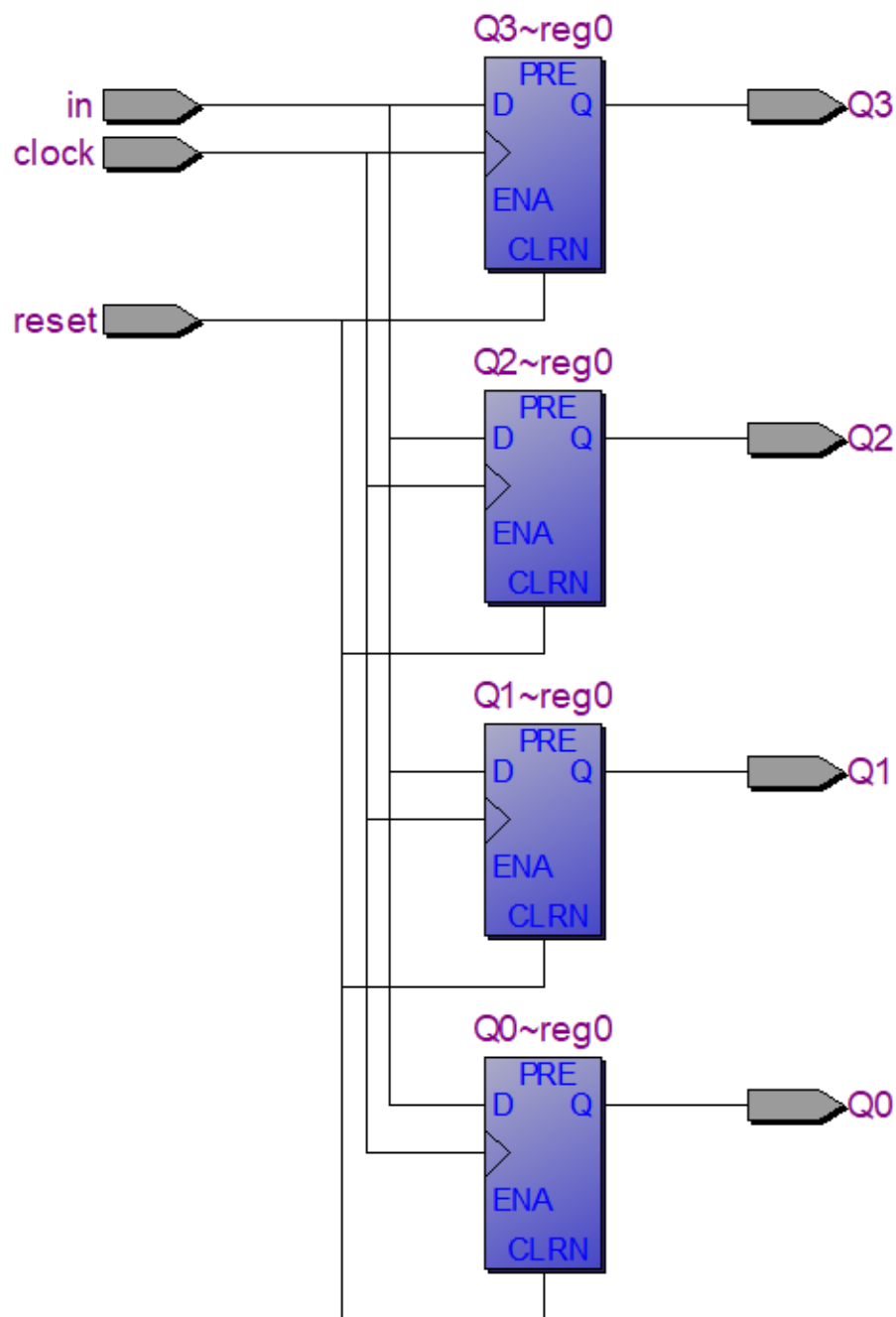


Imagem 03: Circuito da primeira parte

Descrição do circuito lógico:

No circuito lógico acima temos como entrada reset que é responsável por reiniciar o funcionamento dos flip-flops quando em nível lógico ALTO, temos também a entrada clock

que é a mesma para todos os quatro flip-flops assim como a entrada *in* que corresponde ao bit de input da entrada D de cada flip-flop devido ao operador bloqueante que utilizamos na construção do projetos. Além disso temos 4 saídas do tipo reg que mantêm seu nível lógico de uma transição de subida até a próxima.

Dessa forma, todos os flip-flops D utilizados para o deslocamento recebem a mesma saída correspondente ao nível lógico do bit *in* quando ocorre uma transição positiva no clock já que esses flip-flops são ativados por borda de subida.

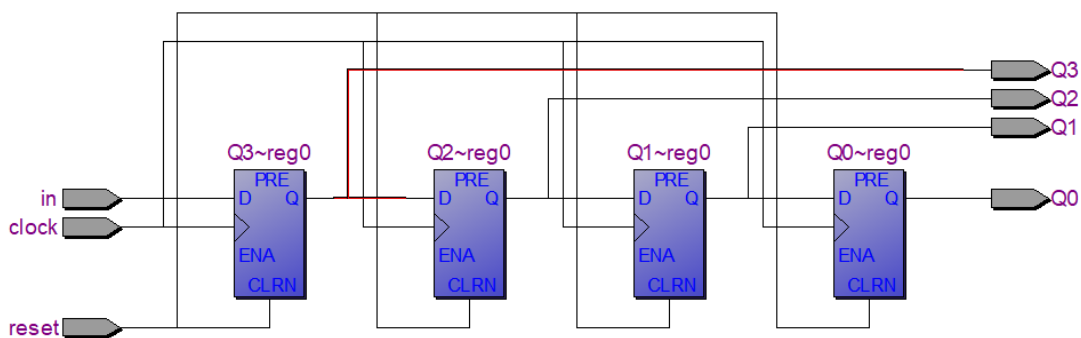


Imagem 04: Circuito da segunda parte

Descrição do circuito lógico:

Com relação ao circuito acima, nele temos os mesmos inputs e outputs do circuito anterior, como a entrada *in*, *clock* e *reset*. Porém, o funcionamento é completamente diferente, pois é nesse circuito que o deslocamento do nível lógico ALTO presente na entrada *in* ocorre devidamente. Além disso os 4 bits do tipo reg para saída de cada flip flop.

Como dito anteriormente, a entrada *reset* reinicia o funcionamento do circuito quando a mesma possui nível lógico ALTO, a entrada *in* funciona apenas como input do primeiro flip-flop, o clock é o mesmo para todos e, a principal diferença no circuito ocorre devido a utilização do operador não bloqueante, já que ao utilizar esse operador, a cada vez que ocorre uma transição de subida no clock, as saídas recebem o nível lógico presente na saída anterior,

por exemplo, após a primeira transição de subida, o nível lógico presente em Q3 passa para Q2, o de Q2 para Q1 e assim por diante.

2.5 Simulação do circuito no ModelSim

Com o objetivo de gerar as ondas no ModelSim Altera e assim simular os dois circuitos construídos foi utilizado um arquivo *testbench* escrito em Verilog para testar todas as possibilidades, e assim evitar a necessidade de gerar as ondas manualmente no software de simulação. O código do *testbench* é basicamente o mesmo para ambos os tipos de projeto e foi disponibilizado no moodle da disciplina pela professora.

Segue o código utilizado:

```
testbench_registrador_naoBloqueante.v

module TB_regis;

    reg TB_reset, TB_clock, TB_in;
    wire TB_Q3,TB_Q2,TB_Q1,TB_Q0;

    // parameter stop_time = 1000;
    // initial # stop_time ;

    RegistradorDeslocamentoNaoBloqueante dut(TB_in, TB_reset, TB_clock, TB_Q3, TB_Q2, TB_Q1,
    TB_Q0);

    initial
        begin
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b1;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b1;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b1;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b1;      TB_in = 1'b0;      #20
            TB_reset = 1'b0;      TB_clock = 1'b0;      TB_in = 1'b0;      #20;
        end // initial

endmodule
```

Imagem 5: Testbench para os circuitos

É válido destacar que por ser um arquivo utilizado para gerar as ondas, o mesmo não possui variáveis de entrada/saída a nível do módulo, porque todas as mudanças são definidas no interior do módulo.

O código é simples e inicialmente realiza uma chamada ao módulo do registrador não bloqueador ou registrador bloqueador, dependendo do projeto. Isso é realizado do lado de fora do bloco *initial*, pois o mesmo não suporta estrutura hierárquica. Dentro do bloco *initial* são definidas algumas entradas e o *clock* do circuito é modificado ao longo do tempo. O comando `#20` indica ao circuito que é preciso esperar 20 unidades de tempo até a realização da próxima instrução.

Simulação do circuito bloqueante

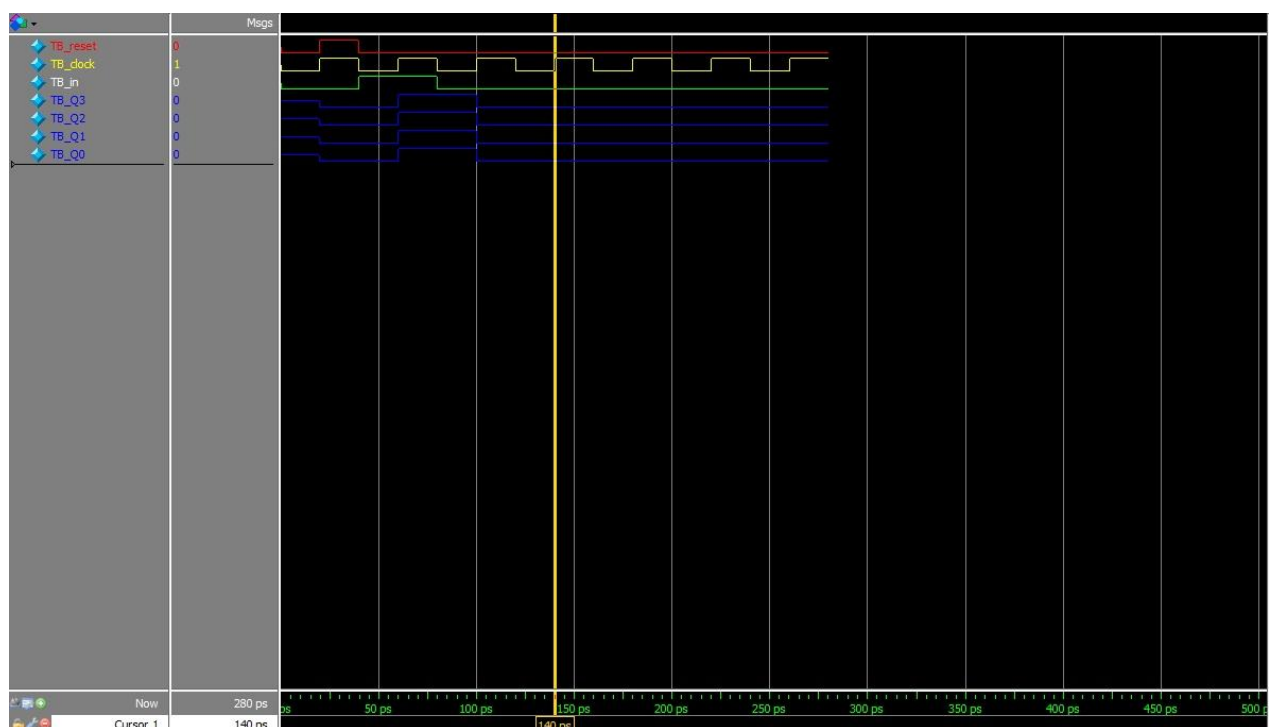


Imagem 6: Painel de ondas do Modelsim Altera

A imagem acima representa as ondas geradas pelo *testbench*. Como é possível observar, as variáveis de saída inicialmente estão desativadas. Após a primeira borda de subida do sinal *reset* as variáveis recebem bit zero. Depois de certo intervalo de tempo a variável de entrada *in* fica com valor 1. Na próxima borda de subida do *clock* o valor da variável de entrada *in* (1) é repassado para todos os registradores, formando uma saída *IIII*, uma vez que está sendo utilizado o padrão bloqueador de atribuição. Na próxima borda de subida do *clock*, todos os registradores recebem 0, tendo em vista o valor de *in*.

Simulação do circuito não bloqueante

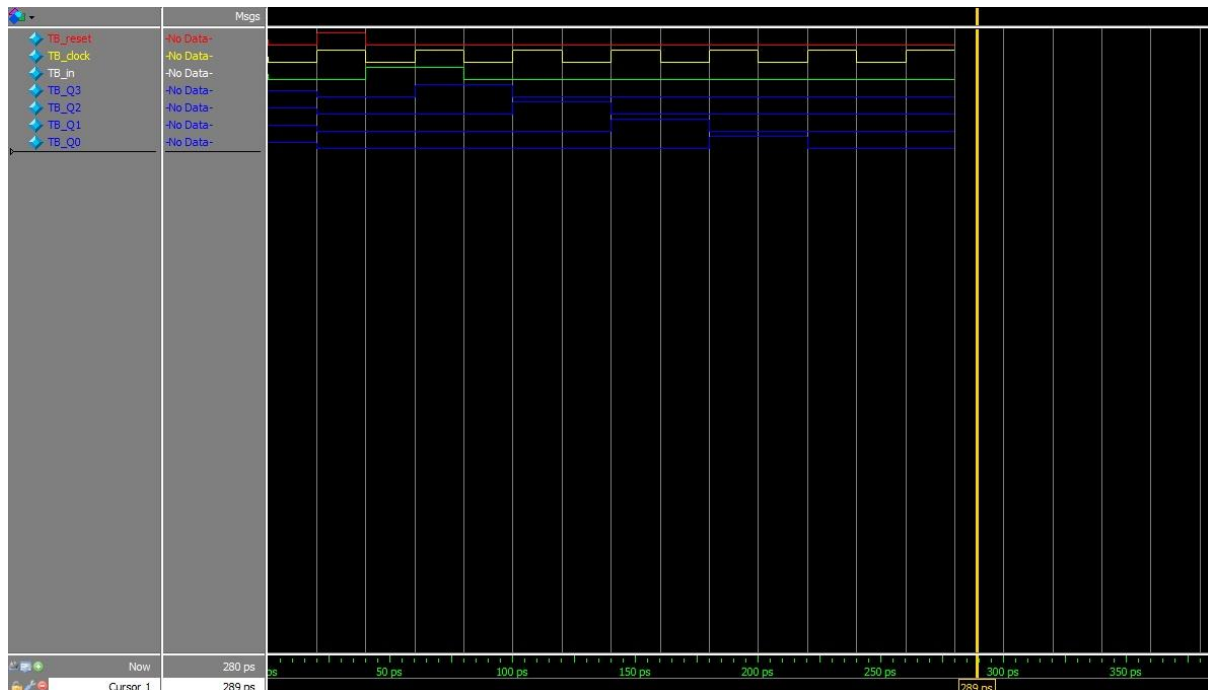


Imagem 7: Pannel de ondas do Modelsim Altera

O circuito não bloqueante é bastante diferente do anterior em termos de simulação. A parte inicial continua inalterada. Inicialmente os registradores são resetados e em determinado momento a variável de entrada *in* recebe valor 1. A partir deste momento, a cada passagem pela borda de subida do *clock*, o sinal recebido é deslocado entre os registradores, da esquerda para a direita, um a um. Diferentemente do modelo de ondas anterior, neste é possível observar que o deslocamento ocorre de forma simultânea entre todas as partes, conforme é possível notar a seguir:

- Primeira borda de subida do *clock*: Q3 = 1; Q2 = 0; Q1 = 0; Q0 = 0;
- Segunda borda de subida do *clock*: Q3 = 0; Q2 = 1; Q1 = 0; Q0 = 0;
- Terceira borda de subida do *clock*: Q3 = 0; Q2 = 0; Q1 = 1; Q0 = 0;
- Quarta borda de subida do *clock*: Q3 = 0; Q2 = 0; Q1 = 0; Q0 = 1;
- Quinta borda de subida do *clock*: Q3 = 0; Q2 = 0; Q1 = 0; Q0 = 0;

É importante notar também que os deslocamentos ocorrem desta maneira porque foi considerado que o bit de entrada *in* apenas assume o valor 0 após o início do processo. Isso permite que apenas um bit ligado (valor um) se locomova entre os registradores.

3. Análise dos resultados:

Neste relatório foi demonstrada a criação de um circuito digital sequencial em Verilog HDL de um registrador de deslocamento e verificada a diferença entre o operador de atribuição bloqueante (=) e o não bloqueante(<=). Utilizando-se de uma variável RESET para o usuário inicializar os Flip-Flops.

Primeiramente foi projetado um circuito registrador de deslocamento de 4 bits (para direita) usando atribuição bloqueante (=).

Já na segunda parte foi projetado um circuito registrador de deslocamento de 4 bits (para direita) usando atribuição NÃO bloqueante (<=).

Os dois tiveram seu funcionamento atestado em simulação no software ModelSim. Através da imagem das ondas do tópico anterior é possível analisar algumas das possibilidades de registro. No entanto, todas as combinações podem ser vistas utilizando o software Modelsim Altera e executando o arquivo “*wave.do*” fornecido.

Conclui-se que a atribuição não bloqueante (<=) utilizada em conjunto de um sinal de *clock* único entre todos os flip-flops é a melhor maneira de projetos circuitos sequenciais síncronos, porque esta maneira mantém a característica sequencial em detrimento da lógica combinacional anteriormente estudada.

Referências

TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L..Sistemas Digitais: Princípios e Aplicações. 11^a ed. São Paulo: Pearson, 2011.