UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO - POLI USP

PCS3645 - Laboratório Digital II

Gabriel Yugo Nascimento Kishida 11257647
Gustavo Azevedo Corrêa 11257693

Turma 1 - Bancada A4

RELATÓRIO - EXPERIMENTO 2 - TRANSMISSÃO SERIAL

Prof. Edson Midorikawa

Prof. Paulo Cugnasca

Prof. Reginaldo Arakaki

SÃO PAULO - SP

2021

SUMÁRIO

1 INTRODUÇÃO

1.1 Visão Geral: Transmissão Serial

Durante este experimento, os alunos terão que aprender sobre transmissão serial, mais especificamente um emissor de um código serial. Será desenvolvido um componente que recebe um valor em ASCII e que comunica este valor serialmente.

Uma transmissão serial é uma forma de comunicar dados em uma linha de só 1 bit. Isto é, utiliza-se a variável tempo para passar mais informações do que a linha permite. Neste caso, para comunicar um dado de 7 bits em uma linha de só 1 bit, transformamos esse vetor de dados em uma onda quadrada, para assim transportar essa informação ao longo do tempo.

1.2 Funcionalidade

O **circuito base** de transmissão serial assíncrono pode ser descrito da seguinte forma:

Figura 1: Definição de bloco do circuito base

O circuito fornecido transmite um caractere ASCII de 7 bits especificado na entrada *dados_ascii* com o acionamento do sinal *partida*. Ao final da transmissão, o sinal *pronto* é acionado. A saída *saida_serial* deve ser usada para ser ligada ao terminal *serial*. A configuração da comunicação serial adotada no projeto é denominada _7E2, ou seja, 7 bits de dados, paridade par e 2 bits de *stop*, com uma taxa de comunicação de 115200 *bauds*.

1.3 Não Funcionalidade

Durante este experimento, não será implementada a recepção e leitura dos sinais fornecidos serialmente. Também não seria implementada a correção de dados que se perderem durante a transmissão (falha na transmissão, como corrigir erros de paridade), já que não foi utilizado algum protocolo como o *Código de Hamming*.

2 SOLUÇÃO TÉCNICA

2.1 Descrição Geral

Neste experimento da disciplina de Laboratório Digital II, nos familiarizaremos com o desenvolvimento de um transmissor serial assíncrono de sinais em ASCII (deslocando o vetor, e transmitindo bit por bit).

No entanto, para um desenvolvimento completo e organizado, o projeto deve ser acompanhado dos seguintes cuidados:

Aferir o circuito fornecido pelo docente em *.vhd*, estudando o funcionamento do circuito e possíveis melhorias a serem aplicada na lógica do sistema. Isto também inclui a documentação do funcionamento no relatório, de forma extensa e compreensiva para auxiliar os estudantes e os leitores a melhor entenderem o desenvolvimento do projeto.

Verificar se o circuito desenvolvido foi aplicado de maneira correta, analisando suas entradas e saídas no arquivo *.vhd*, procurando erros no código e verificando se o mesmo compila no *Quartus Prime*.

Simular o circuito desenvolvido para diversas entradas, estudando as saídas obtidas no *Modelsim* e analisando se os resultados obtidos são condizentes com o que era esperado.

3 ESTRATÉGIA DE MONTAGEM E AFERIÇÃO DE QUALIDADE

3.1 Funcionamento do circuito

Dado o acionamento do reset, o circuito entra no estado inicial e permanece nesse até a ativação do sinal *partida*, nisso ele zera os contadores e vai para o estado de espera. No estado de espera, o contador soma 1 unidade a cada clock até que se atinja 434 clocks, quando se atinge esse valor é ativado o sinal de tick e passa-se para o estado de transmissão onde o deslocador seleciona o próximo bit a ser passado pela comunicação serial. Terminado a transmissão dos dados o circuito volta para o estado inicial.

3.2 Atividade 1

3.2.1 Montagem

Como o circuito base já está montado, esta etapa se resume ao estudo do **circuito base**. Para tanto identificou-se:

Componentes do Fluxo de Dados:

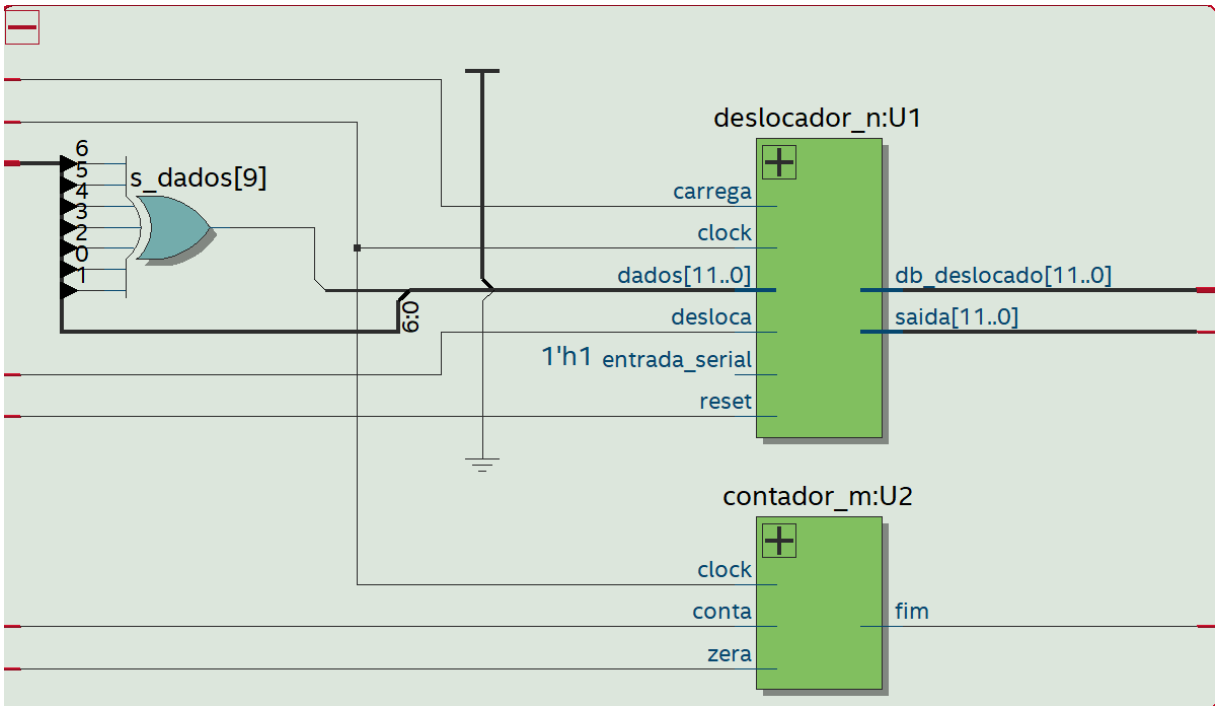

Figura 2: Diagrama de estados

Como pode-se observar na ferramenta RTL do Quartus o fluxo de dados é composto por:

- A monstagem do vetor de dados a ser passado (o que inclui uma porta XOR de 7 bits para identificar a paridade do dado)
- O deslocador responsável por atualizar o bit que será passado na transmissão serial
- Contador responsável por determinar o fim do vetor a ser transmitido

Como os bits de entrada são armazenados no componente interno do fluxo de dados:

Os bits de entrada são armazenados em um registrador interno ao deslocador.

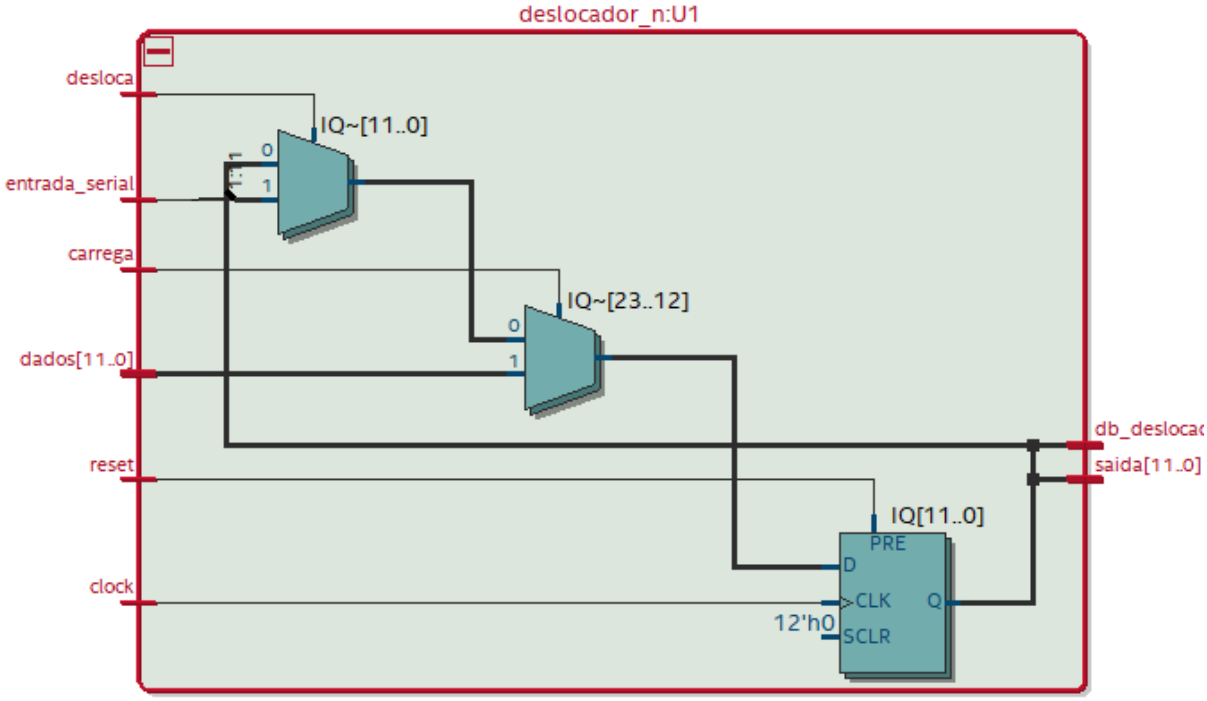

Figura 3: Esquema interno do deslocador

Analisar a máquina de estados da unidade de controle (entidade *tx_serial_uc*) e estude os estados e as transições entre eles.

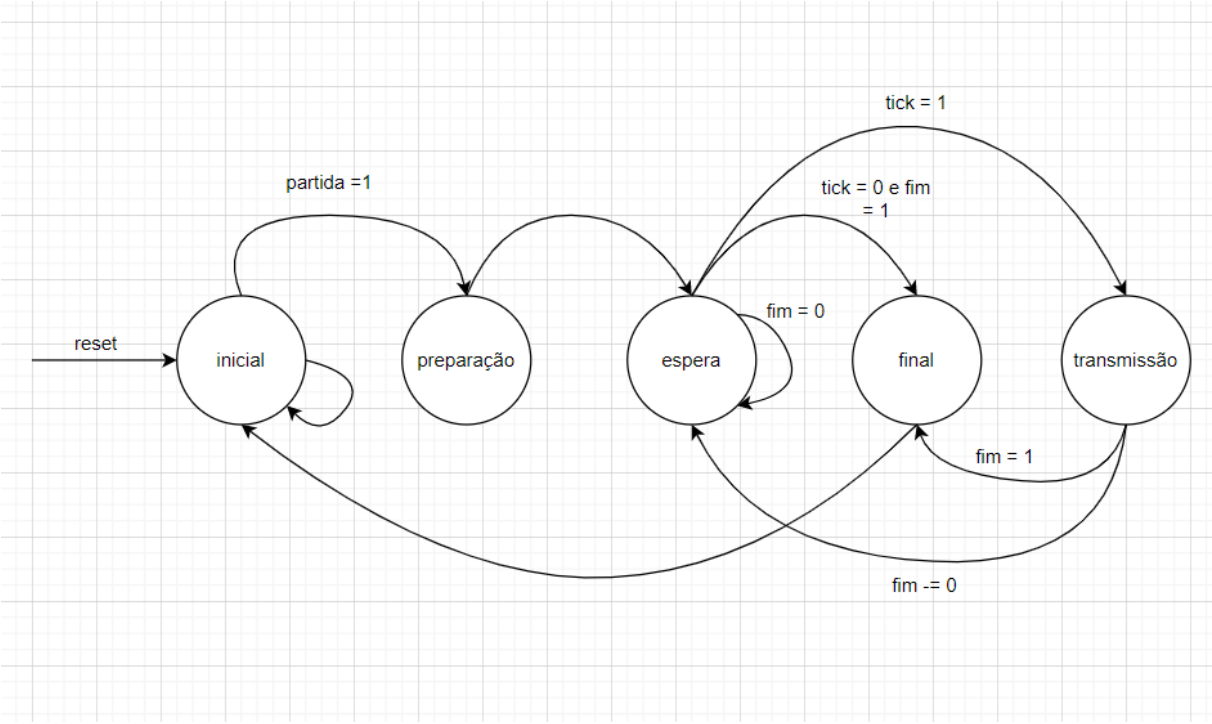

Figura 4: Diagrama de estados

A máquina de estados da unidade de controle é composta por 5 estados:

- **Inicial**: O estado em que não se iniciou o processo de trasmissão dos dados e aguarda-se o sinal partida para tal.
- **Preparação**: O estado em que se zera os contadores para inicio do ciclo de transmissão
- **Espera**: O estado em que se espera o tick para transmissão do próximo bit ou o sinal de fim para encerrar o ciclo de transmissão.
- **Transmissão**: Estado em que efetivamente ocorre a trasmissão do bit fornecido pelo deslocador na saída serial para então ou retornar para o estado de espera ou ir para o de fim caso tenha acabado o dado a ser transmitido.
- **Final**: Estado final atingido ao final da transmissão dos dados e que retorna para o estado inicial para o aguardo de outra transmissão.

Verificar como o sinal interno de tick é gerado e como este sinal é usado pela unidade de controle para determinar os instantes em que o sinal serial é modificado.

O sinal de tick tem que ser gerado de forma a atingir os 115 200 bauds de taxa de transmissão, para isso faz-se a proporção entre essa os bauds e o clock de funcionamento da unidade de controle afim de chegar no número de ciclos a cada tick.

Nesse caso a porporção é de 434 clocks por tick, e, portanto, o contador fora da unidade de controle tem que contar 434 clocks até a ativação do tick.

Uma vez recebido o sinal do tick, a unidade de controle passa para o estado de transmissão e passa os sinais necessários para tal, fazendo o deslocador passar o próximo bitm mudando o bit de saída etc.

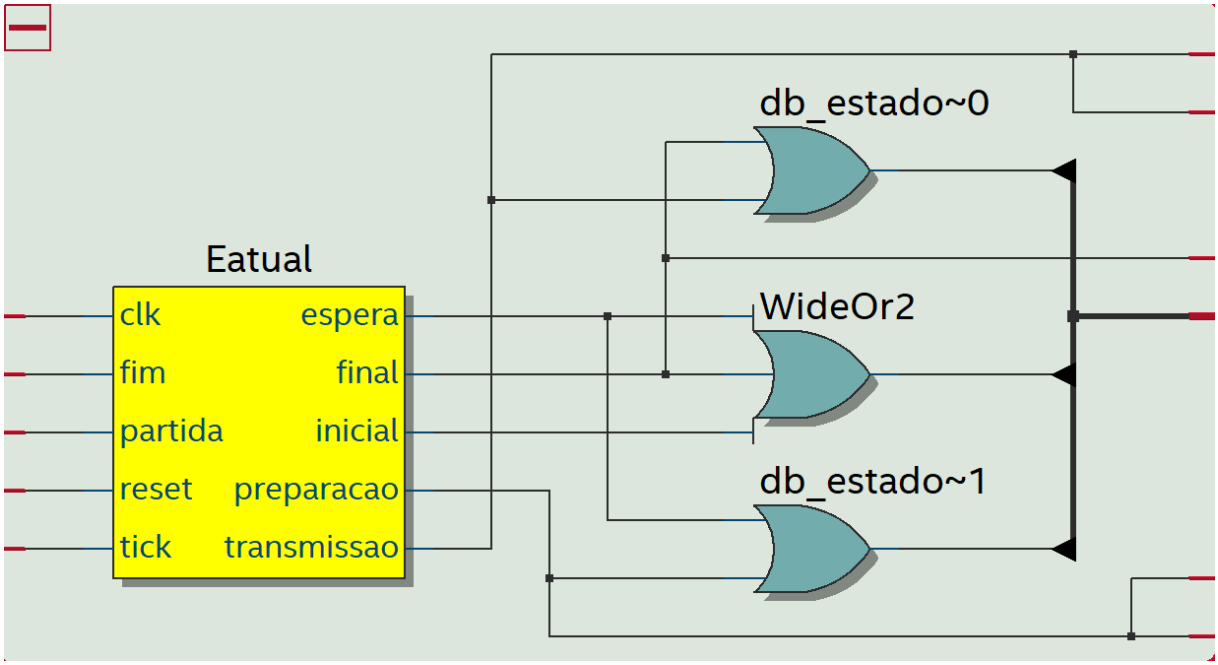

Figura 5: Unidade de controle

Além de realizar estes estudos, para facilitar a testagem e o aprendizado sobre o circuito, adicionamos os seguintes sinais de depuração:

- *db_estado*: sinal que representa qual o estado atual da máquina de estados na unidade de controle;
- *db_deslocado*: sinal que representa qual a informação armazenada dentro do registrador do deslocador;
- *db_tick*: sinal que representa quando o contador chegou ao fim de sua contagem

3.2.2 Testagem

Para um projeto adequado, existe a necessidade de uma testagem adequada. Desta forma, desenvolvemos os seguintes casos de teste:

1. **Testar** se a comunicação serial para o sinal *"0110101"* (sinal com paridade **par**) foi feita de maneira correta;

2. **Testar** se a comunicação serial para o sinal *"0101001"* (sinal com paridade **ímpar**) foi feita de maneira correta;

Não é conveniente testar todas as possibilidades de sinais enviados, pois seriam $2^7 = 128$ casos de testagem. Para sinais de maior quantidade de bit, essa quantidade de casos cresceria exponencialmente. Portanto, testam-se somente os casos importantes.

Teste - Paridade Par

Simulando o funcionamento do circuito no software Quartus, é gerado a carta de tempos com as saídas do circuito pelo ModelSim e a partir disso verificam-se os sinais seriais gerados.

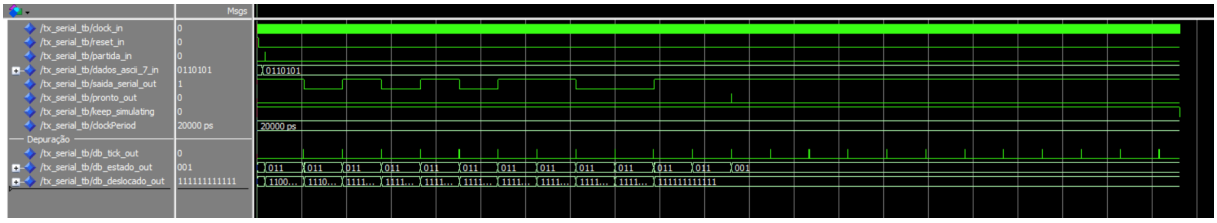

Figura 6: Testagem para paridade par

Da simulação do circuito base no ModelSim, percebeu-se que o funcionamento estava correto: o sinal serial correspondia ao que era esperado:

- 1 (Repouso)
- 0 (Start bit)
- 1010110 (Dados do bit menos significativo para o mais)
- 0 (Bit de paridade)
- 11 (Bits de parada)

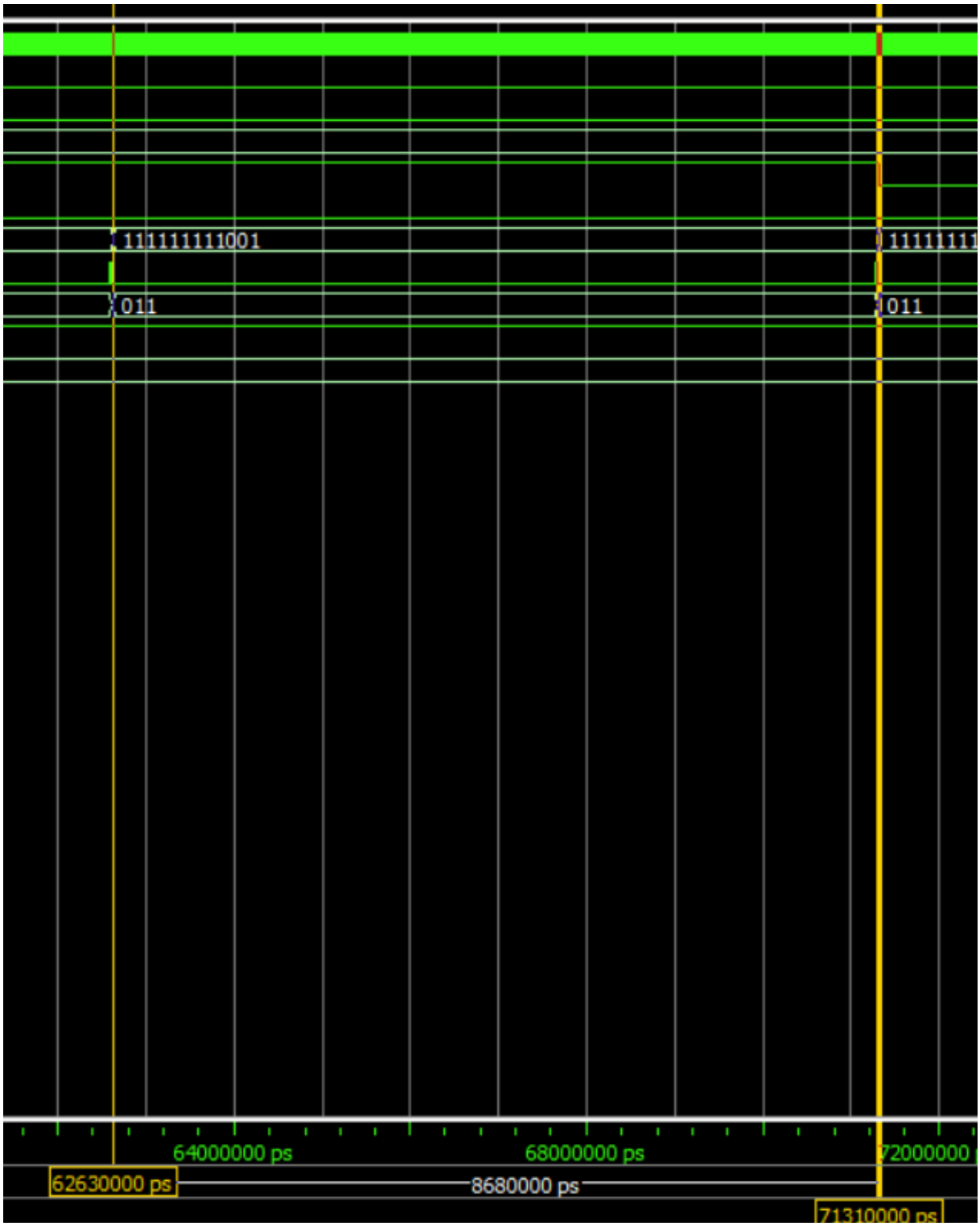

Figura 7: Medida do tempo para bits do caso 1

O sinal de tick era ativado antes de cada transmissão de bit e a largura dos bits é de aproximadamente 8,68us de acordo com o esperado.

Teste - Paridade Ímpar

Mais uma vez, simulando o funcionamento do circuito no software Quartus, obtemos:

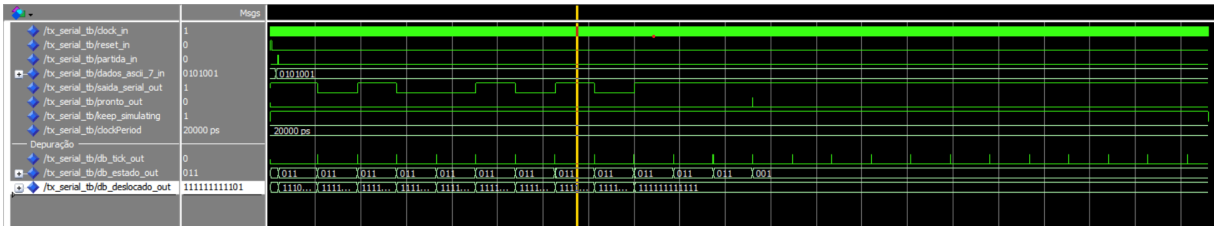

Figura 8: Testagem para paridade ímpar

A maior diferença entre estes dois testes é que (além dos bits de dados serem diferentes), o **bit de paridade** neste caso de teste dever ser **igual a "1"**.

Da simulação do circuito base no ModelSim, percebeu-se que o funcionamento estava correto: o sinal serial correspondia ao que era esperado:

- 1 (Repouso)
- 0 (Start bit)
- 1001010 (Dados do bit menos significativo para o mais)
- 1 (Bit de paridade)
- 11 (Bits de parada)

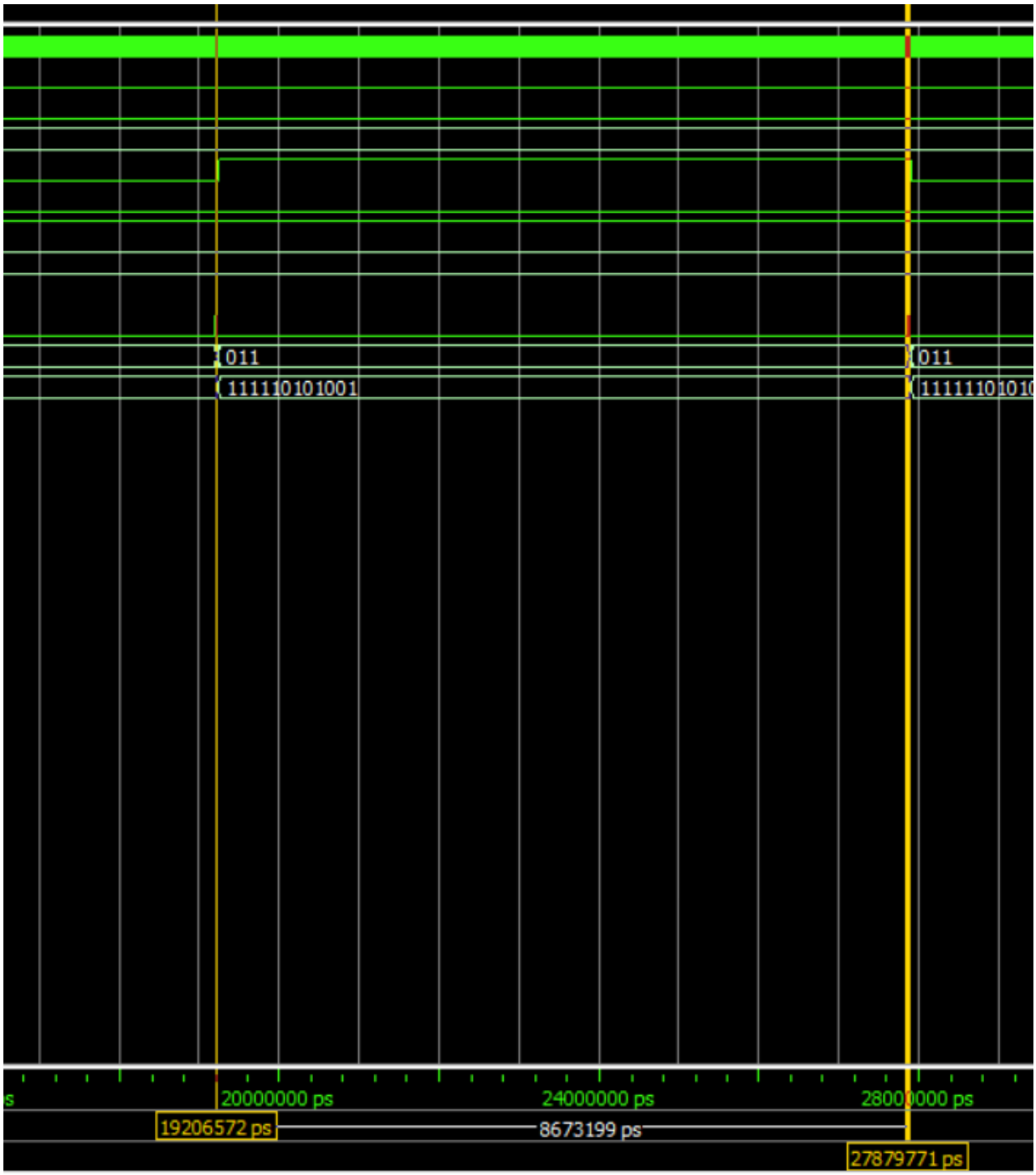

Figura 9: Medida do tempo para bits do caso 2

O sinal de tick era ativado antes de cada transmissão de bit e a largura dos bits é de aproximadamente 8,68us de acordo com o esperado.

3.3 Atividade 2

Esta atividade envolve a modificação do projeto base para transmissão serial assíncrona desenvolvido na Atividade 1 para usar a configuração de transmissão 8N2 (8 bits

de dados, sem paridade, 2 stop bits) e uma taxa de comunicação de 9600 bauds.

3.3.1 Montagem

Para realizar as alterações propostas, as seguintes mudanças foram encessárias:

Para ajustar a configuração de comunicação 8N2

- Alterar o dado de entrada para ter 8 bits;
- Remover o bit de paridade.

Para alterar a taxa de comunicação serial para 9600 bauds

- Alterar o módulo do contador de 434 para 5208;
- Alterar a quantidade de bits do contador para 13 (para ser possível chegar no 5208);

Essa mudança no módulo do contador é efetiva pois, como nós queremos atingir **9600 bauds** (isto é, 9600 bits por segundo), necessitamos que o nosso contador apresente 9600 ticks por segundo. Como temos um *clock* de 50MHz (ou seja, 50 milhões de períodos de clock por segundo), conseguimos determinar a quantidade M de *clocks* que temos em $\frac{1}{9600}s$ a partir da equação:

$$M = \frac{50000000}{9600} = 5208,3333... \tag{1}$$

Arredondando este valor então para 5208.

OBS: neste experimento, não se alterou a máquina de estados, visto que a transmissão se deu da mesma forma. Além disso, também se mantiveram os sinais de depuração da primeira atividade, para auxiliar na análise do novo circuito em questão.

Componentes do novo circuito:

A seguir, está a visão (obtida por meio da ferramenta RTL Viewer) do novo circuito desenvolvido, assim como a visão de dentro do fluxo de dados:

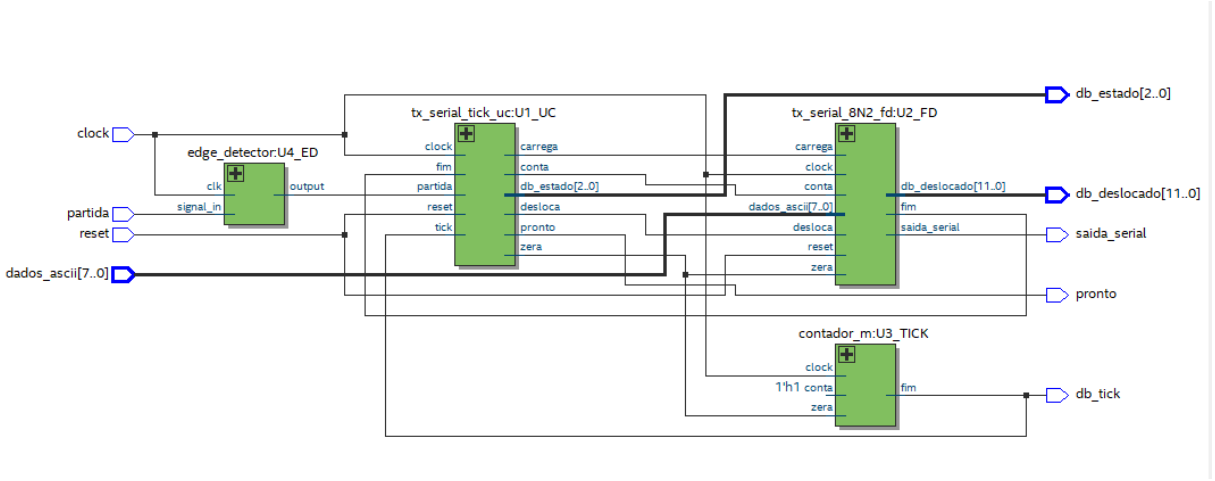

Figura 10: RTL Viewer do circuito da atividade 2

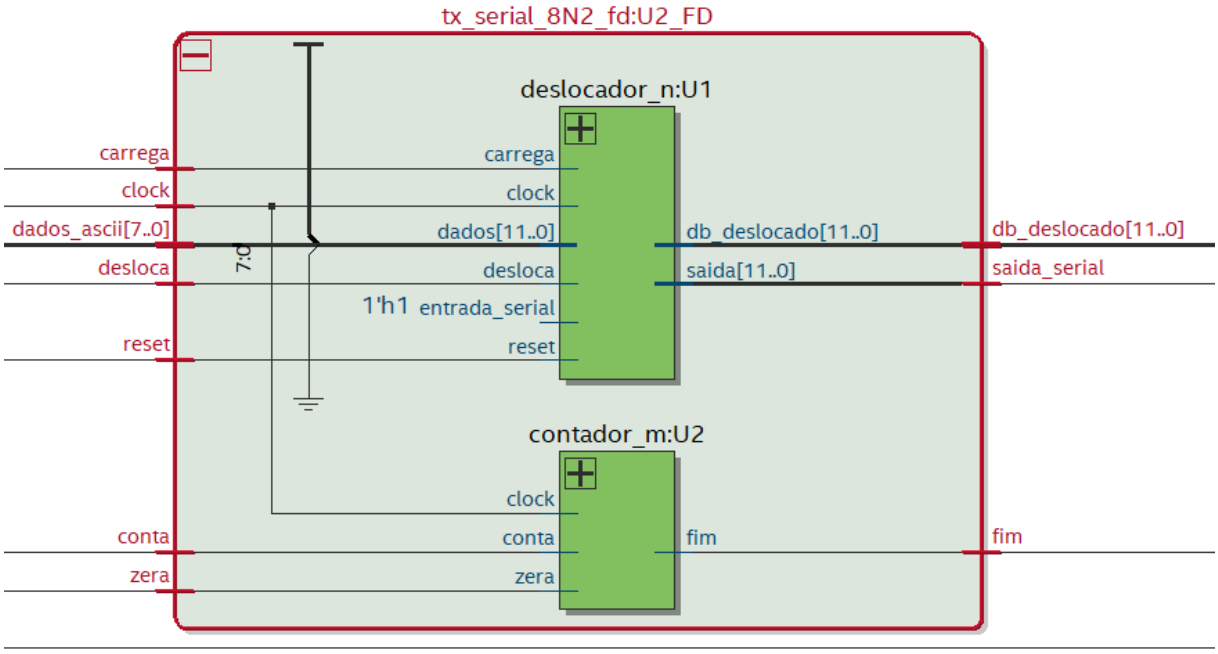

Figura 11: RTL Viewer do fluxo de dados do circuito da atividade 2

Muito do que já estava na atividade 1 se manteve.

3.3.2 Testagem

Para a testagem, conforme indicado no roteiro do experimento, utilizarão-se os mesmos casos de testagem. A única modificação no *testbench* será o tempo em que ele se mantém ativo, já que o comprimento da onda (devido à alteração dos no número de **bauds**)

Teste - Paridade Par

Novamente, as simulações foram obtidas no ModelSim, gerando o seguinte formato de onda para o primeiro caso de teste (cujo dado de entrada é *"00110101"*):

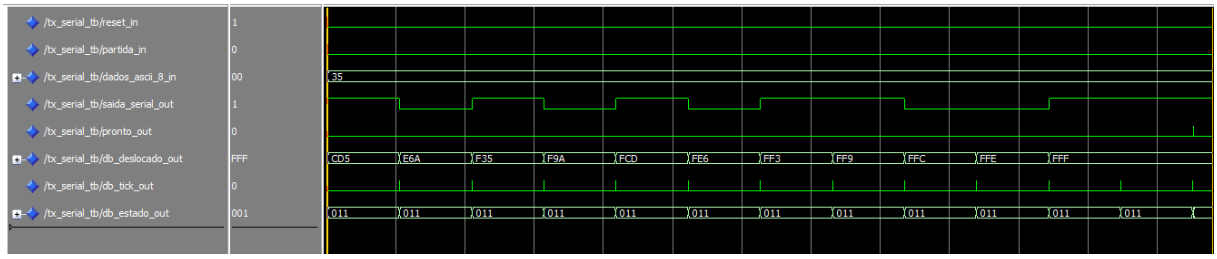

Figura 12: Testagem para paridade par

Da simulação deste circuito modificado no ModelSim, percebeu-se que o funcionamento estava de acordo com o esperado. O sinal serial apresenta:

- 1 tick em alta (Repouso)
- 1 tick em baixa (Start bit)
- O dado inserido "1010110"(do bit menos significativo para o mais)
- 2 ticks em alta (Bits de parada)

É notável que, nesse protocolo de comunicação, não existe o bit de paridade, diferentemente do 7E2.

Por fim, também avaliou-se se o tempo decorrido entre cada *tick* correspondia com o esperado (para **9600 bauds**, deveria ser $\frac{1}{9600}s = 0,10416ms$).

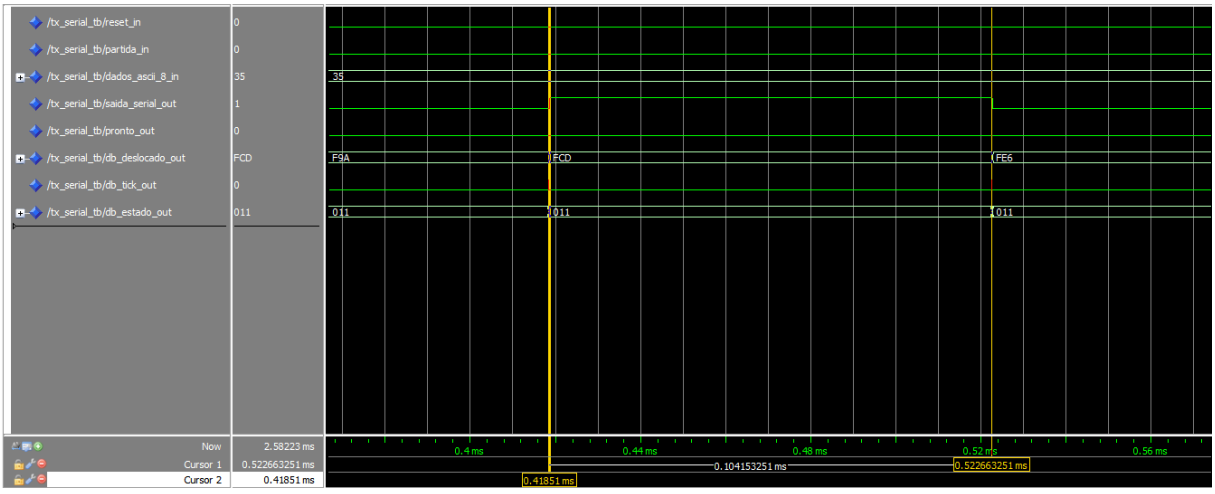

Figura 13: Tempo decorrido entre cada tick para o primeiro cenário de teste

Como este valor está dentro do esperado, o circuito passou neste primeiro caso de teste.

Teste - Paridade Ímpar

Por fim, o seguinte formato de onda para o segundo caso de teste (cujo dado de entrada é *"00101001"*):

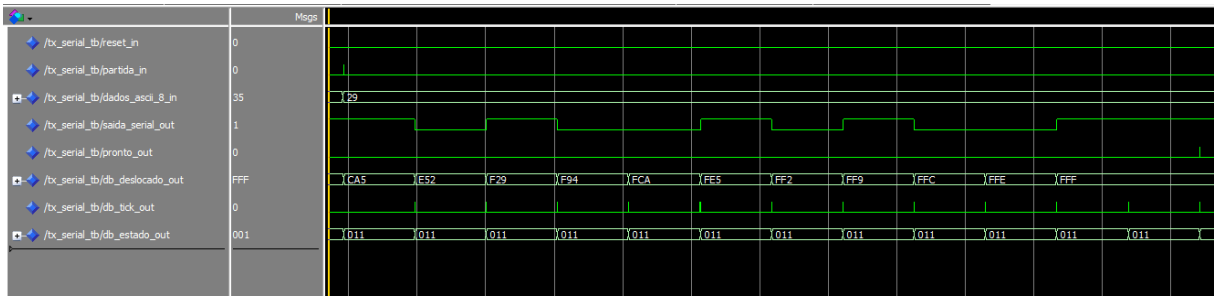

Figura 14: Testagem para paridade ímpar

- 1 tick em alta (Repouso)
- 1 tick em baixa (Start bit)
- O dado inserido "10010100"(do bit menos significativo para o mais)
- 2 ticks em alta (Bits de parada)

Mais uma vez, avaliando o tempo decorrido entre cada *tick* (que deve ser $0,10416ms$):

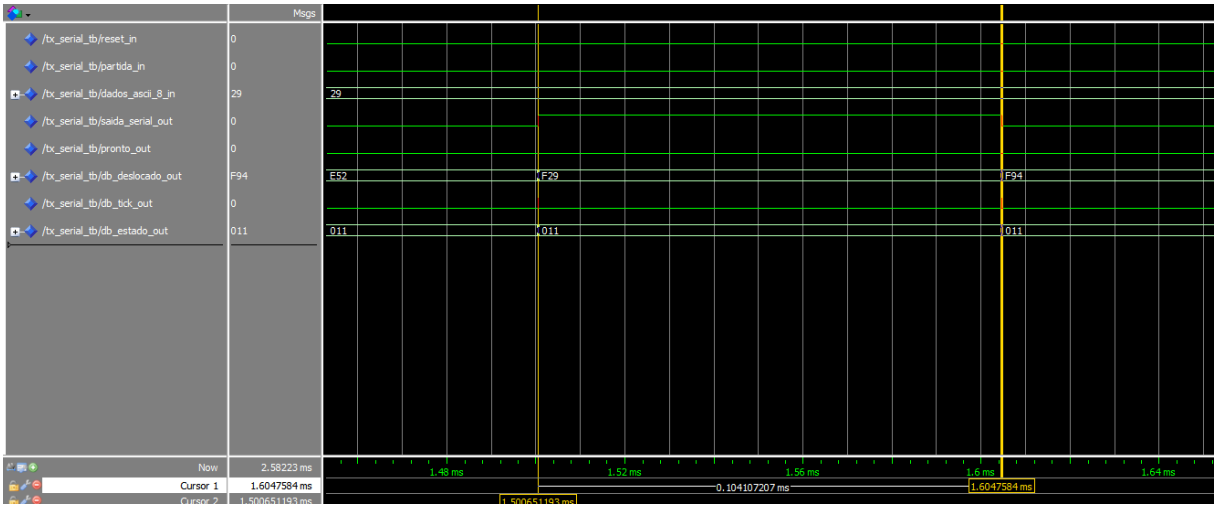

Figura 15: Tempo decorrido entre cada tick para o segundo cenário de teste

Como este valor está dentro do esperado, o circuito passou neste segundo caso de teste.

3.3.3 Pinagem

Como a testagem foi bem sucedida, prossegue-se então para a pinagem do circuito, preparando-o para passá-lo à placa FPGA.

Percebeu-se uma falta de LEDs para o sinal de depuração *db_deslocado*, e portanto, adicionaram-se três componentes *hexa7seg*, que transformam um **sinal de 4 bits hexadecimal** em um sinal apresentável em um **display de 7 segmentos**. Dividiu-se os 12 bits do sinal *db_deslocado* em três grupos de 4 bits, gerando o novo circuito:

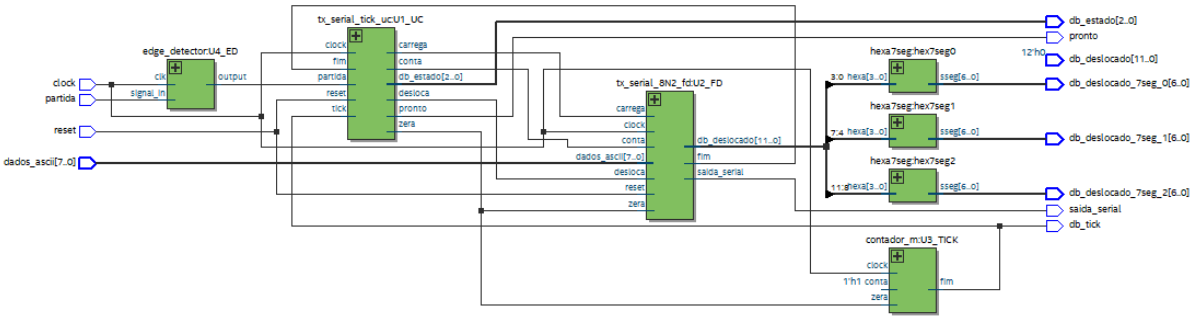

Figura 16: Novo circuito, adaptado para os display de 7 segmentos

Desta forma, a pinagem será possibilitada, e não faltarão LEDs.

A nova pinagem ficou:

Sinal	Pino	Pino FPGA	MQTT	Analog Discovery
clock	CLK_50	M9	-	-
reset	GPIO_0_D0	N16	E0	-
partida	GPIO_0_D1	B16	E1	-
dados_ascii[0...7]	GPIO_0_D2, D3, D4, D5, D6, D7, D8, D9	M16, C16, D17, K20, K21, K22, M20, M21	E2, E3, ... E9	-
pronto	LEDR9	L1	-	-
saida_serial	GPIO_1_D27	F15	-	Scope - CH1+
db_deslocado	-	-	-	-
db_deslocado_7seg_0	HEX0	U21, V21, W22, W21, Y22, Y21, AA22	-	-
db_deslocado_7seg_1	HEX1	AA20, AB20, AA19, AA18, AB18, AA17, U22	-	-
db_deslocado_7seg_2	HEX2	Y19, AB17, AA10, Y14, V14, AB22, AB21	-	-
db_estado	LEDR0, 1, 2	AA2, AA1, W2	-	-
db_tick	LEDR3	Y3	-	-

Figura 17: Pinagem para o circuito da atividade 2

3.3.4 Aplicação

Para a aplicação, criamos mais duas saídas para a transmissão serial

Osciloscópio - Scope

Ao passar o circuito para o Anydesk, foi realizada a pinagem do dispositivo, e preparou-se o Waveforms para o modo de osciloscópio - Scope.

Após a configuração, foram enviados sinais via MQTT Dash para realizar a verificação do formato de onda enviado para o Analog Discovery. Para isso, foi enviado o sinal em ASCII equivalente ao caractere *"e"* (*"1100101"em binário*) e ao caractere *"m"* (*"1101101"em binário*):

Testagem para o caractere *"e"*

A entrada no MQTT Dash para enviar o caractere *"e"* foi:

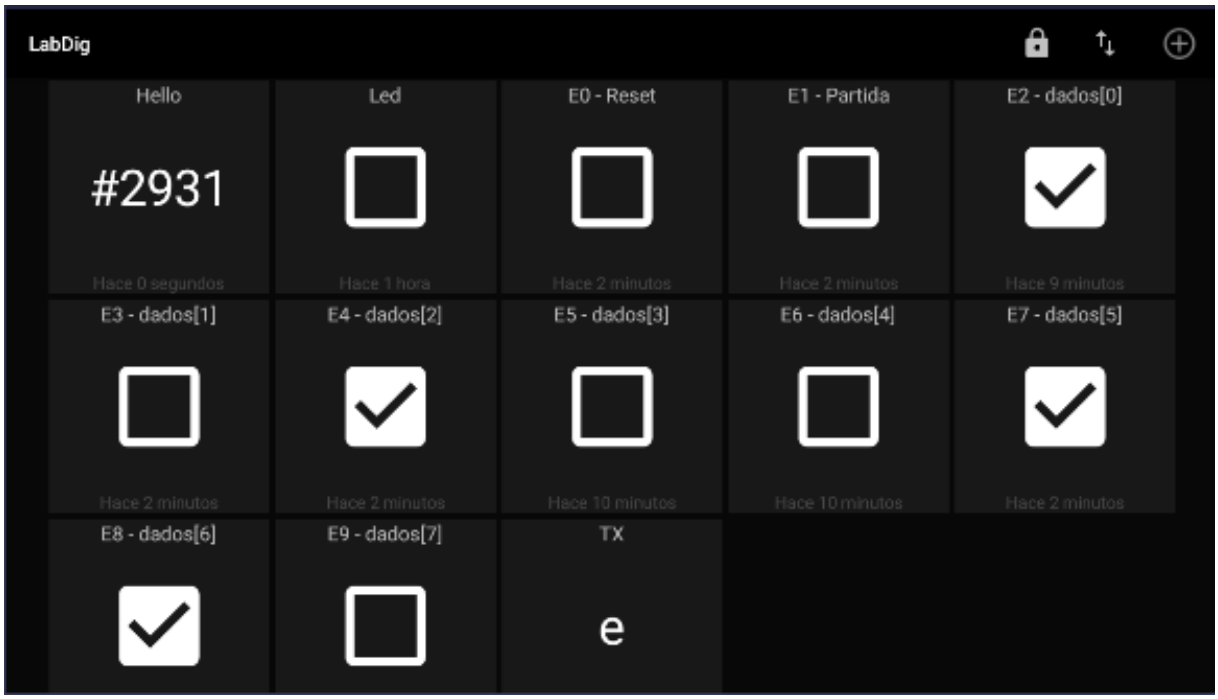

Figura 18: Entrada MQTT para o caractere ”e”

A onda recebida no osciloscópio foi:

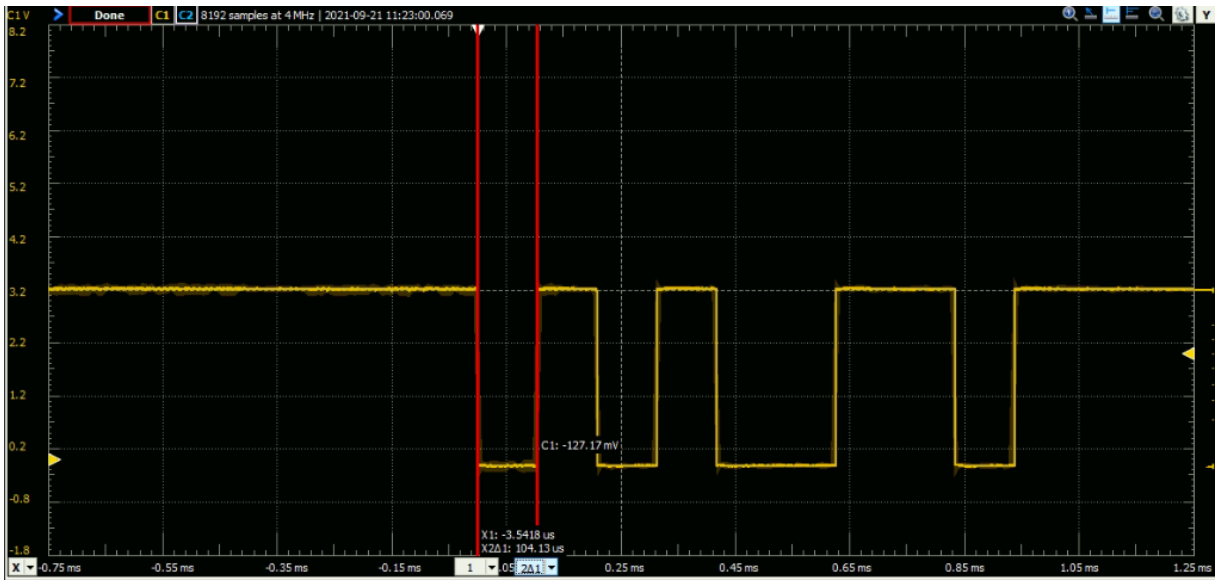

Figura 19: Onda serial para o caractere ”e”

Primeiramente, é possível notar que a duração de 1 bit do sinal é adequada para **9600 bauds**, isto é, é de *104us*.

Além disso, é notável que o sinal está correto de acordo com as expectativas:

- 1 bit em alto (repouso)

- 1 bit em baixo (start bit)
- Dado ASCII (1 bit em alta, 1 bit em baixo, 1 bit em alta, 2 bits em baixo, 1 bit em alto)
- 1 bit em baixo (paridade)
- 2 bits em alto (Stop bit)

Testagem para o caractere *"m"*

A entrada no MQTT Dash para enviar o caractere *"m"* foi:

Figura 20: Entrada MQTT para o caractere "m"

A onda recebida no osciloscópio foi:

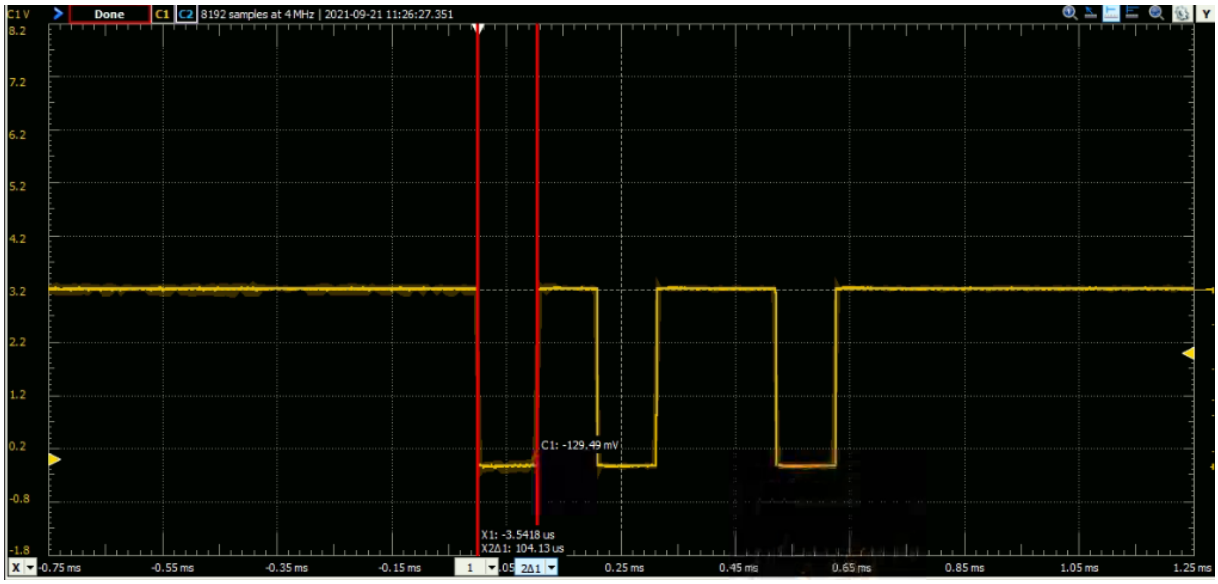

Figura 21: Onda serial para o caractere ”m”

Novamente, a duração de 1 bit do sinal é adequada para **9600 bauds**, isto é, tendo uma duração de *104us*.

Mais uma vez, é notável que o sinal está correto de acordo com as expectativas:

- 1 bit em alto (repouso)
- 1 bit em baixo (start bit)
- Dado ASCII (1 bit em alta, 1 bit em baixo, 2 bits em alta, 1 bit em baixo, 2 bits em alto)
- 1 bit em alta (paridade)
- 2 bits em alto (Stop bit)

Protocolo de recepção - Protocol

Terminada a validação dos formatos de onda pelo osciloscópio usou-se a ferramenta Protocol para verificar se os sinais de saída correspondiam com os seus valores ASCII intendidos. Foram usados os caracteres **a** e **c** para essa checagem.

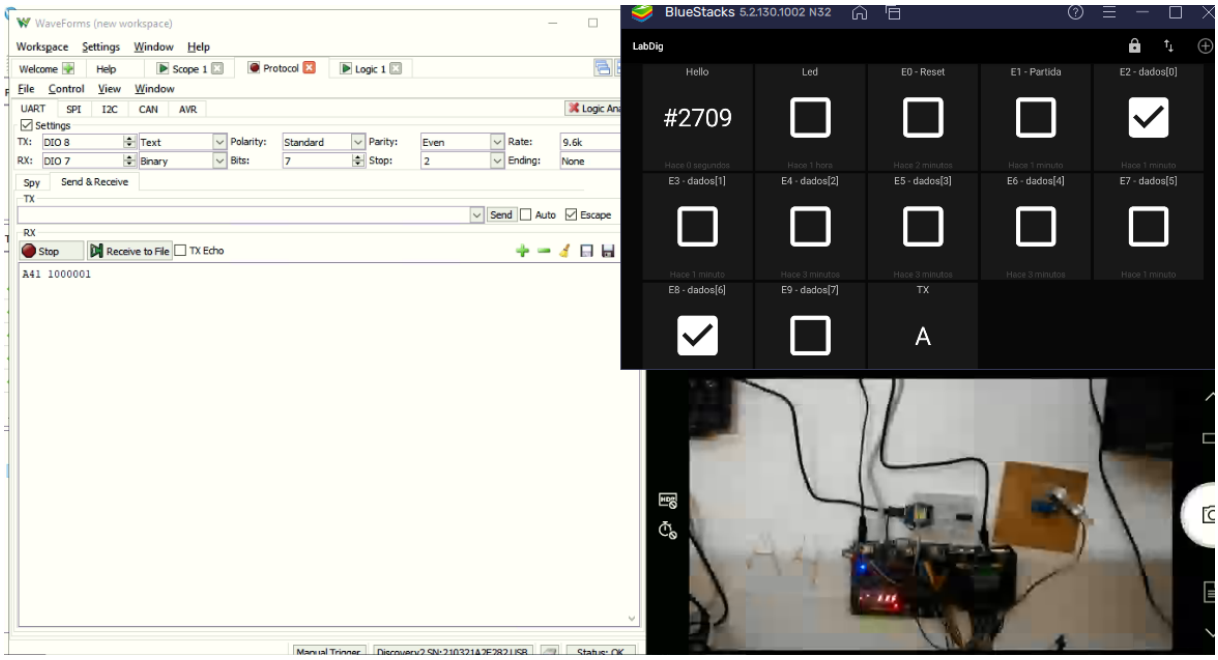

Figura 22: Testagem do circuito para o recebimento do caractere A

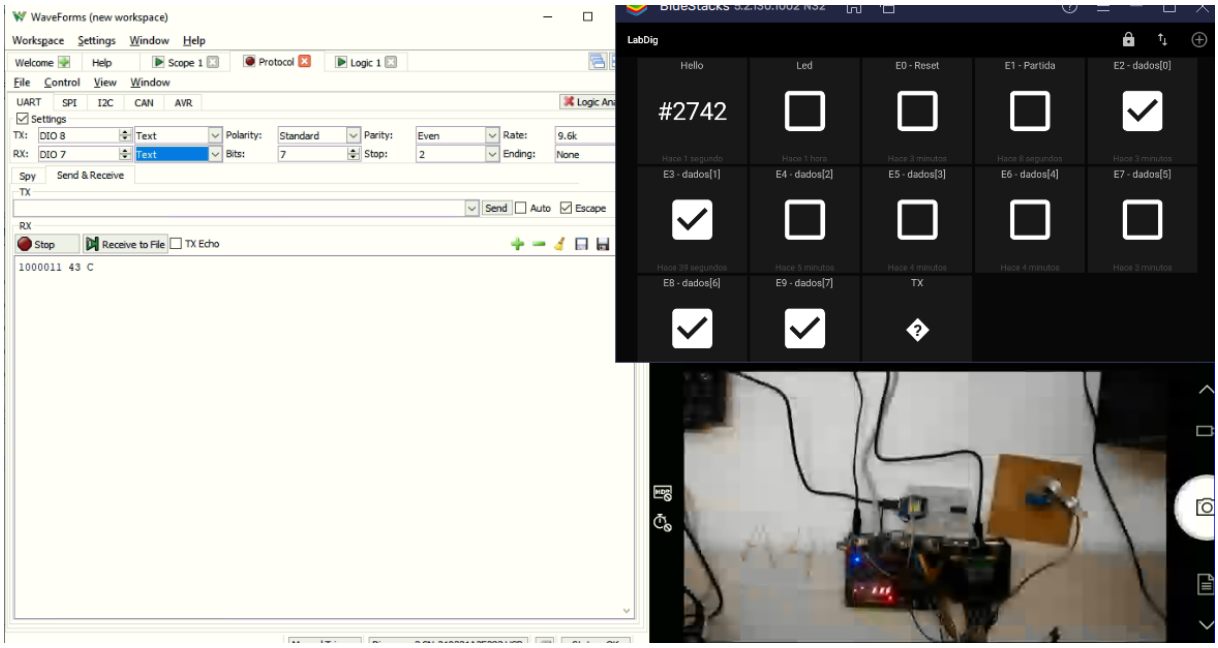

Figura 23: Testagem do circuito para o recebimento do caractere C

Como pode se verificar pelas imagens 22 e 23, o circuito transmite para a saída de forma correta. Isso pode ser verificado também pela saída **tx** do MQTT Dash, onde os caracteres também se mostram corretos.

4 RESULTADOS

No começo do experimento houveram algumas dificuldades técnicas, as quais foram corrigidas com a ajuda do monitor, porém isso causou um atraso considerável no desenvolvimento do experimento.

Apesar disso o decorrimento do laboratório foi como o esperado. O formato da onda na saída da transmissão serial teve o formato correto e os planos de teste tiveram resultados positivos, com o circuito passando em todos. O funcionamento da geração da saída serial pode ser verificado pelo osciloscópio nas imagens 19 e 21.

A transmissão dos caracteres também ocorreu de forma satisfatória, isso podendo ser verificado nas imagens 22 e 23.

Durante a realização do experimento os sinais de depuração estabelecidos no planejamento foram de grande utilidade para testagem fora da placa, porém, devido a frequência alta de funcionamento do circuito, esses sinais se tornaram menos uteis para o processo de debugging.

Considerando que o intuito do experimento era familiarizar os alunos com os protocolos de comunicação serial e sua montagem e testagem em hardware, pode-se considerar que foi um sucesso.

4.1 Avaliação

4.1.1 Avaliação da Aula - Nota: 8

Por parte do grupo, sentiu-se que a aula foi razoavelmente corrida - principalmente por duas causas:

1. Imprevistos técnicos de conexão;
2. Velocidade do processamento do Anydesk estava muito devagar (estava travando);
3. Delay para a apresentação das atividades;

O grupo reconhece que nenhuma dessas é intencional por parte do docente, mas foi um fator que frustrou o desenvolvimento do projeto, e atrasou o suficiente para não ser possível a realização do *Desafio*.

No entanto, o projeto desenvolvido foi muito bem estruturado, e as ferramentas disponibilizadas para o desenvolvimento e testagem foram ótimas, e por isso avalia-se bem o experimento 2, com uma **nota 8**.

4.1.2 Autoavaliação - Nota 8

O grupo sente que fez um preparo bem minuncioso para a aula: o planejamento foi feito de maneira adequada com casos de teste bem pensados. Por isso, no momento da aplicação, não foi necessária a modificação da lógica do circuito (as únicas mudanças realizadas foram nos *outputs*, para redirecionar a saída para o osciloscópio, ou para o Protocol, ou para o MQTT).

A nota só não é máxima devido ao fato de que, devido os imprevistos técnicos, o grupo se atrasou e não conseguiu realizar o desafio. Seria possível, com uma maior organização e atenção, preparar para a apresentação de maneira mais rápida, e assim sobrar um maior tempo para a realização do Desafio.

Além disso, os sinais de depuração não foram muito pensados para a aplicação na placa FPGA (*o sinal db_deslocado*, por exemplo, só é útil quando se analisa no ModelSim, já que na placa FPGA suas alterações são muito rápidas e imperceptíveis ao olho nu.) Como aprendizado, será levado em conta nos próximos experimentos estes dois tipos de sinais de depuração: para o uso no ModelSim (ou na análise de ondas) e para o uso na placa FPGA.

5 APÊNDICE

5.1 Atividade 1 - tx_serial_7E2

5.1.1 tx_serial_7E2$_fd$

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity tx_serial_7E2_fd is
7      port (
8          clock, reset:                   in  std_logic;
9          zera, conta, carrega, desloca: in  std_logic;
10         dados_ascii:                    in  std_logic_vector (6 downto 0);
11         saida_serial, fim :             out std_logic;
12       db_deslocado:           out std_logic_vector (11 downto 0)
13     );
14 end entity;
15
16 architecture tx_serial_7E2_fd_arch of tx_serial_7E2_fd is
17
18     component deslocador_n
19     generic (
20         constant N: integer
21     );
22     port (
23         clock, reset: in std_logic;
24         carrega, desloca, entrada_serial: in std_logic;
25         dados: in std_logic_vector (N-1 downto 0);
26         saida: out  std_logic_vector (N-1 downto 0);
27       db_deslocado: out std_logic_vector (N-1 downto 0)
28     );
29     end component;
30
31     component contador_m
32     generic (
33         constant M: integer;
34         constant N: integer
35     );
36     port (
37         clock, zera, conta: in std_logic;
38         Q: out std_logic_vector (N-1 downto 0);
39         fim: out std_logic
40     );
41     end component;
42
43     signal s_dados, s_saida, s_db_deslocado: std_logic_vector (11 downto 0);
44
45 begin
46
47     s_dados(0) <= '1';  -- repouso
48     s_dados(1) <= '0';  -- start bit
49     s_dados(8 downto 2) <= dados_ascii;
50     -- bit de paridade da transmissao par
51     s_dados(9) <= dados_ascii(0) xor dados_ascii(1) xor dados_ascii(2) xor dados_ascii(3)
```

```
52                  xor dados_ascii(4) xor dados_ascii(5) xor dados_ascii(6);
53      s_dados(11 downto 10) <= "11";  -- stop bits
54
55      U1: deslocador_n generic map (N => 12)  port map (clock, reset, carrega, desloca, '1', s_dados,
         s_saida, s_db_deslocado);
56
57      U2: contador_m generic map (M => 13, N => 4) port map (clock, zera, conta, open, fim);
58
59     db_deslocado <= s_db_deslocado;
60
61      saida_serial <= s_saida(0);
62
63  end architecture;
```

5.1.2 tx_serial_7E2

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5
6   entity tx_serial_7E2 is
7       port (
8           clock, reset, partida: in  std_logic;
9           dados_ascii:           in  std_logic_vector (6 downto 0);
10          saida_serial, pronto : out std_logic;
11        db_deslocado         : out std_logic_vector (11 downto 0);
12        db_tick: out std_logic;
13        db_estado: out std_logic_vector (2 downto 0)
14      );
15  end entity;
16
17  architecture tx_serial_7E2_arch of tx_serial_7E2 is
18
19      component tx_serial_tick_uc port (
20              clock, reset, partida, tick, fim:      in  std_logic;
21              zera, conta, carrega, desloca, pronto: out std_logic;
22          db_estado:                      out std_logic_vector (2 downto 0)
23      );
24      end component;
25
26      component tx_serial_7E2_fd port (
27          clock, reset: in std_logic;
28          zera, conta, carrega, desloca: in std_logic;
29          dados_ascii: in std_logic_vector (6 downto 0);
30          saida_serial, fim : out std_logic;
31        db_deslocado: out std_logic_vector (11 downto 0)
32      );
33      end component;
34
35      component contador_m
36      generic (
37          constant M: integer;
38          constant N: integer
39      );
40      port (
41          clock, zera, conta: in std_logic;
42          Q: out std_logic_vector (N-1 downto 0);
43          fim: out std_logic
44      );
45      end component;
46
47      component edge_detector is port (
```

```
48              clk         : in    std_logic;
49              signal_in   : in    std_logic;
50              output      : out   std_logic
51      );
52      end component;
53
54      signal s_reset, s_partida, s_partida_ed: std_logic;
55      signal s_zera, s_conta, s_carrega, s_desloca, s_tick, s_fim: std_logic;
56     signal s_db_deslocado: std_logic_vector (11 downto 0);
57     signal s_db_estado: std_logic_vector (2 downto 0);
58
59  begin
60
61      -- sinais reset e partida mapeados na GPIO (ativos em alto)
62      s_reset   <= reset;
63      s_partida <= partida;
64
65      -- unidade de controle
66      U1_UC: tx_serial_tick_uc port map (clock, s_reset, s_partida_ed, s_tick, s_fim,
67                                         s_zera, s_conta, s_carrega, s_desloca, pronto, s_db_estado);
68
69      -- fluxo de dados
70      U2_FD: tx_serial_7E2_fd port map (clock, s_reset, s_zera, s_conta, s_carrega, s_desloca,
71                                        dados_ascii, saida_serial, s_fim, s_db_deslocado);
72
73      -- gerador de tick
74      -- fator de divisao 50MHz para 115.200 bauds (434=50M/115200), 9 bits
75      U3_TICK: contador_m generic map (M => 434, N => 9) port map (clock, s_zera, '1', open, s_tick);
76
77      -- detetor de borda para tratar pulsos largos
78      U4_ED: edge_detector port map (clock, s_partida, s_partida_ed);
79
80     db_deslocado <= s_db_deslocado;
81     db_tick <= s_tick;
82     db_estado <= s_db_estado;
83
84
85  end architecture;
```

5.1.3 tx_serial_7E2$_tb$

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity tx_serial_tb is
6   end entity;
7
8   architecture tb of tx_serial_tb is
9
10    -- Componente a ser testado (Device Under Test -- DUT)
11    component tx_serial_7E2
12      port (
13          clock, reset, partida: in  std_logic;
14          dados_ascii:           in  std_logic_vector (6 downto 0);
15          saida_serial, pronto : out std_logic;
16        db_deslocado:      out std_logic_vector (11 downto 0);
17        db_tick:           out std_logic;
18        db_estado:       out std_logic_vector (2 downto 0)
19      );
20    end component;
21
22    -- Declara  o de sinais para conectar o componente a ser testado (DUT)
```

```
23  --    valores iniciais para fins de simulacao (ModelSim)
24  signal clock_in: std_logic := '0';
25  signal reset_in: std_logic := '0';
26  signal partida_in: std_logic := '0';
27  signal dados_ascii_7_in: std_logic_vector (6 downto 0) := "0000000";
28  -- signal dados_ascii_8_in: std_logic_vector (7 downto 0) := "00000000";
29  signal saida_serial_out: std_logic := '1';
30  signal pronto_out: std_logic := '0';
31  signal db_deslocado_out: std_logic_vector (11 downto 0) := "000000000000";
32  signal db_tick_out: std_logic;
33  signal db_estado_out: std_logic_vector (2 downto 0);
34
35  -- Configura  es do clock
36  signal keep_simulating: std_logic := '0'; -- delimita o tempo de gera  o do clock
37  constant clockPeriod : time := 20 ns;     -- clock de 50MHz
38
39 begin
40  -- Gerador de clock: executa enquanto 'keep_simulating = 1', com o per odo
41  -- especificado. Quando keep_simulating=0, clock    interrompido, bem como a
42  -- simula  o de eventos
43  clock_in <= (not clock_in) and keep_simulating after clockPeriod/2;
44
45  -- Conecta DUT (Device Under Test)
46  dut: tx_serial_7E2
47       port map
48       (
49           clock=>          clock_in,
50           reset=>          reset_in,
51           partida=>        partida_in,
52           dados_ascii=>    dados_ascii_7_in,
53           saida_serial=>   saida_serial_out,
54           pronto=>         pronto_out,
55         db_deslocado=>   db_deslocado_out,
56         db_tick=>      db_tick_out,
57         db_estado=>      db_estado_out
58       );
59
60  -- geracao dos sinais de entrada (estimulos)
61  stimulus: process is
62  begin
63
64    assert false report "Inicio da simulacao" severity note;
65    keep_simulating <= '1';
66
67    ---- inicio da simulacao: reset ----------------
68    partida_in <= '0';
69    reset_in <= '1';
70    wait for 20*clockPeriod;  -- pulso com 20 periodos de clock
71    reset_in <= '0';
72    wait until falling_edge(clock_in);
73    wait for 50*clockPeriod;
74
75    ---- dado de entrada da simulacao (caso de teste #1)
76    dados_ascii_7_in <= "0110101"; -- x35 = '5'
77    wait for 20*clockPeriod;
78
79    ---- acionamento da partida (inicio da transmissao)
80    partida_in <= '1';
81    wait until rising_edge(clock_in);
82    wait for 5*clockPeriod; -- pulso partida com 5 periodos de clock
83    partida_in <= '0';
84
85    ---- espera final da transmissao (pulso pronto em 1)
86  wait until pronto_out='1';
87
88  ---- final do caso de teste 1
```

```
89
90      -- intervalo entre casos de teste
91      wait for 5000*clockPeriod;
92
93      ----
94      ---- colocar aqui outros casos de teste
95      ----
96
97
98      ---- final dos casos de teste da simulacao
99      assert false report "Fim da simulacao" severity note;
100     keep_simulating <= '0';
101
102     wait; -- fim da simula  o: aguarda indefinidamente
103   end process;
104
105
106 end architecture;
```

5.2 Atividade 2 - tx_serial_8N2

5.2.1 hexa7seg

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity hexa7seg is
5      port (
6          hexa : in  std_logic_vector(3 downto 0);
7          sseg : out std_logic_vector(6 downto 0)
8      );
9  end hexa7seg;
10
11 architecture comportamental of hexa7seg is
12 begin
13
14   sseg <= "1000000" when hexa="0000" else
15           "1111001" when hexa="0001" else
16           "0100100" when hexa="0010" else
17           "0110000" when hexa="0011" else
18           "0011001" when hexa="0100" else
19           "0010010" when hexa="0101" else
20           "0000010" when hexa="0110" else
21           "1111000" when hexa="0111" else
22           "0000000" when hexa="1000" else
23           "0000000" when hexa="1000" else
24           "0010000" when hexa="1001" else
25           "0001000" when hexa="1010" else
26           "0000011" when hexa="1011" else
27           "1000110" when hexa="1100" else
28           "0100001" when hexa="1101" else
29           "0000110" when hexa="1110" else
30           "0001110" when hexa="1111" else
31           "1111111";
32
33 end comportamental;
```

5.2.2 tx_serial_8N2$_{fd}$

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity tx_serial_8N2_fd is
6      port (
7          clock, reset:                  in  std_logic;
8          zera, conta, carrega, desloca: in  std_logic;
9          dados_ascii:                   in  std_logic_vector (7 downto 0);
10         saida_serial, fim :            out std_logic;
11       db_deslocado:           out std_logic_vector (11 downto 0)
12     );
13 end entity;
14
15 architecture tx_serial_8N2_fd_arch of tx_serial_8N2_fd is
16
17     component deslocador_n
18     generic (
19         constant N: integer
20     );
21     port (
22         clock, reset: in std_logic;
23         carrega, desloca, entrada_serial: in std_logic;
24         dados: in std_logic_vector (N-1 downto 0);
25         saida: out  std_logic_vector (N-1 downto 0);
26       db_deslocado: out std_logic_vector (N-1 downto 0)
27     );
28     end component;
29
30     component contador_m
31     generic (
32         constant M: integer;
33         constant N: integer
34     );
35     port (
36         clock, zera, conta: in std_logic;
37         Q: out std_logic_vector (N-1 downto 0);
38         fim: out std_logic
39     );
40     end component;
41
42     signal s_dados, s_saida, s_db_deslocado: std_logic_vector (11 downto 0);
43
44 begin
45
46     s_dados(0) <= '1';  -- repouso
47     s_dados(1) <= '0';  -- start bit
48     s_dados(9 downto 2) <= dados_ascii;
49     s_dados(11 downto 10) <= "11";  -- stop bits
50
51     U1: deslocador_n generic map (N => 12)  port map (clock, reset, carrega, desloca, '1', s_dados,
       s_saida, s_db_deslocado);
52
53     U2: contador_m generic map (M => 13, N => 4) port map (clock, zera, conta, open, fim);
54
55    db_deslocado <= s_db_deslocado;
56
57     saida_serial <= s_saida(0);
58
59 end architecture;
```

5.2.3 tx_serial_8N2

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6  entity tx_serial_8N2 is
7      port (
8          clock, reset, partida : in  std_logic;
9          dados_ascii       : in  std_logic_vector (7 downto 0);
10         saida_serial, pronto  : out std_logic;
11       db_deslocado        : out std_logic_vector (11 downto 0);
12       db_deslocado_7seg_0 : out std_logic_vector (6 downto 0);
13       db_deslocado_7seg_1 : out std_logic_vector (6 downto 0);
14       db_deslocado_7seg_2 : out std_logic_vector (6 downto 0);
15       db_tick         : out std_logic;
16       db_estado         : out std_logic_vector (2 downto 0)
17     );
18 end entity;
19
20 architecture tx_serial_8N2_arch of tx_serial_8N2 is
21
22     component tx_serial_tick_uc port (
23             clock, reset, partida, tick, fim:      in  std_logic;
24             zera, conta, carrega, desloca, pronto: out std_logic;
25         db_estado:                     out std_logic_vector (2 downto 0)
26     );
27     end component;
28
29     component tx_serial_8N2_fd port (
30         clock, reset: in std_logic;
31         zera, conta, carrega, desloca: in std_logic;
32         dados_ascii: in std_logic_vector (7 downto 0);
33         saida_serial, fim : out std_logic;
34       db_deslocado: out std_logic_vector (11 downto 0)
35     );
36     end component;
37
38     component contador_m
39     generic (
40         constant M: integer;
41         constant N: integer
42     );
43     port (
44         clock, zera, conta: in std_logic;
45         Q: out std_logic_vector (N-1 downto 0);
46         fim: out std_logic
47     );
48     end component;
49
50     component edge_detector is port (
51             clk          : in   std_logic;
52             signal_in    : in   std_logic;
53             output       : out  std_logic
54     );
55     end component;
56
57    component hexa7seg is
58     port (
59         hexa : in  std_logic_vector(3 downto 0);
60         sseg : out std_logic_vector(6 downto 0)
61     );
62    end component;
63
```

```
64     signal s_reset, s_partida, s_partida_ed: std_logic;
65     signal s_zera, s_conta, s_carrega, s_desloca, s_tick, s_fim: std_logic;
66    signal s_db_deslocado: std_logic_vector (11 downto 0);
67    signal s_db_estado: std_logic_vector (2 downto 0);
68
69 begin
70
71     -- sinais reset e partida mapeados na GPIO (ativos em alto)
72     s_reset   <= reset;
73     s_partida <= partida;
74
75     -- unidade de controle
76     U1_UC: tx_serial_tick_uc port map (clock, s_reset, s_partida_ed, s_tick, s_fim,
77                                        s_zera, s_conta, s_carrega, s_desloca, pronto, s_db_estado);
78
79     -- fluxo de dados
80     U2_FD: tx_serial_8N2_fd port map (clock, s_reset, s_zera, s_conta, s_carrega, s_desloca,
81                                       dados_ascii, saida_serial, s_fim, s_db_deslocado);
82
83     -- gerador de tick
84     -- fator de divisao 50MHz para 9600 bauds (5208=50M/9600), 13 bits
85     U3_TICK: contador_m generic map (M => 5208, N => 13) port map (clock, s_zera, '1', open, s_tick);
86
87     -- detetor de borda para tratar pulsos largos
88     U4_ED: edge_detector port map (clock, s_partida, s_partida_ed);
89
90    hex7seg0: hexa7seg port map (hexa => s_db_deslocado(3 downto 0), sseg => db_deslocado_7seg_0);
91    hex7seg1: hexa7seg port map (hexa => s_db_deslocado(7 downto 4), sseg => db_deslocado_7seg_1);
92    hex7seg2: hexa7seg port map (hexa => s_db_deslocado(11 downto 8), sseg => db_deslocado_7seg_2);
93
94    db_tick <= s_tick;
95    db_estado <= s_db_estado;
96
97
98 end architecture;
```

5.2.4 tx_serial_8N2$_tb$

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity tx_serial_tb is
6  end entity;
7
8  architecture tb of tx_serial_tb is
9
10   -- Componente a ser testado (Device Under Test -- DUT)
11   component tx_serial_8N2
12     port (
13         clock, reset, partida: in  std_logic;
14         dados_ascii:           in  std_logic_vector (7 downto 0);
15         saida_serial, pronto : out std_logic;
16       db_deslocado:      out std_logic_vector (11 downto 0);
17       db_tick:           out std_logic;
18       db_estado:       out std_logic_vector (2 downto 0)
19     );
20   end component;
21
22   -- Declara  o de sinais para conectar o componente a ser testado (DUT)
23   --   valores iniciais para fins de simulacao (ModelSim)
24   signal clock_in: std_logic := '0';
25   signal reset_in: std_logic := '0';
```

```
26   signal partida_in: std_logic := '0';
27   signal dados_ascii_8_in: std_logic_vector (7 downto 0) := "00000000";
28   -- signal dados_ascii_8_in: std_logic_vector (7 downto 0) := "00000000";
29   signal saida_serial_out: std_logic := '1';
30   signal pronto_out: std_logic := '0';
31   signal db_deslocado_out: std_logic_vector (11 downto 0) := "000000000000";
32   signal db_tick_out: std_logic;
33   signal db_estado_out: std_logic_vector (2 downto 0);
34
35   -- Configura  es do clock
36   signal keep_simulating: std_logic := '0'; -- delimita o tempo de gera   o do clock
37   constant clockPeriod : time := 20 ns;     -- clock de 50MHz
38
39 begin
40   -- Gerador de clock: executa enquanto 'keep_simulating = 1', com o per odo
41   -- especificado. Quando keep_simulating=0, clock    interrompido, bem como a
42   -- simula   o de eventos
43   clock_in <= (not clock_in) and keep_simulating after clockPeriod/2;
44
45   -- Conecta DUT (Device Under Test)
46   dut: tx_serial_8N2
47        port map
48        (
49            clock=>           clock_in,
50            reset=>           reset_in,
51            partida=>         partida_in,
52            dados_ascii=>     dados_ascii_8_in,
53            saida_serial=>    saida_serial_out,
54            pronto=>          pronto_out,
55          db_deslocado=>    db_deslocado_out,
56          db_tick=>       db_tick_out,
57          db_estado=>       db_estado_out
58        );
59
60   -- geracao dos sinais de entrada (estimulos)
61   stimulus: process is
62   begin
63
64     assert false report "Inicio da simulacao" severity note;
65     keep_simulating <= '1';
66
67     ---- inicio da simulacao: reset ----------------
68     partida_in <= '0';
69     reset_in <= '1';
70     wait for 20*clockPeriod;  -- pulso com 20 periodos de clock
71     reset_in <= '0';
72     wait until falling_edge(clock_in);
73     wait for 50*clockPeriod;
74
75     ---- dado de entrada da simulacao (caso de teste #1)
76     dados_ascii_8_in <= "00110101"; -- x35 = '5'
77     wait for 20*clockPeriod;
78
79     ---- acionamento da partida (inicio da transmissao)
80     partida_in <= '1';
81     wait until rising_edge(clock_in);
82     wait for 5*clockPeriod; -- pulso partida com 5 periodos de clock
83     partida_in <= '0';
84
85     ---- espera final da transmissao (pulso pronto em 1)
86   wait until pronto_out='1';
87
88   ---- final do caso de teste 1
89
90     -- intervalo entre casos de teste
91     wait for 2000*clockPeriod;
```

```
92
93      ---- dado de entrada da simulacao (caso de teste #2)
94      dados_ascii_8_in <= "00101001"; -- x29 = ')'
95      wait for 20*clockPeriod;
96
97      ---- acionamento da partida (inicio da transmissao)
98      partida_in <= '1';
99      wait until rising_edge(clock_in);
100     wait for 5*clockPeriod; -- pulso partida com 5 periodos de clock
101     partida_in <= '0';
102
103     ---- espera final da transmissao (pulso pronto em 1)
104     wait until pronto_out='1';
105
106 ---- final do caso de teste 2
107
108     -- intervalo
109     wait for 2000*clockPeriod;
110
111     ---- final dos casos de teste da simulacao
112     assert false report "Fim da simulacao" severity note;
113     keep_simulating <= '0';
114
115     wait; -- fim da simula  o: aguarda indefinidamente
116   end process;
117
118
119 end architecture;
```

5.3 Componentes comuns

5.3.1 tx_serial_tick$_uc$

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity tx_serial_tick_uc is
5      port ( clock, reset, partida, tick, fim:      in  std_logic;
6             zera, conta, carrega, desloca, pronto: out std_logic;
7          db_estado:                    out std_logic_vector (2 downto 0)
8      );
9  end entity;
10
11 architecture tx_serial_tick_uc_arch of tx_serial_tick_uc is
12
13     type tipo_estado is (inicial, preparacao, espera, transmissao, final);
14     signal Eatual: tipo_estado;  -- estado atual
15     signal Eprox:  tipo_estado;  -- proximo estado
16
17 begin
18
19     -- memoria de estado
20     process (reset, clock)
21     begin
22         if reset = '1' then
23             Eatual <= inicial;
24         elsif clock'event and clock = '1' then
25             Eatual <= Eprox;
26         end if;
27     end process;
28
```

```
29   -- logica de proximo estado
30     process (partida, tick, fim, Eatual)
31     begin
32
33       case Eatual is
34
35         when inicial =>          if partida='1' then Eprox <= preparacao;
36                                  else                Eprox <= inicial;
37                                  end if;
38
39         when preparacao =>       Eprox <= espera;
40
41         when espera =>           if tick='1' then   Eprox <= transmissao;
42                                  elsif fim='0' then Eprox <= espera;
43                                  else               Eprox <= final;
44                                  end if;
45
46         when transmissao =>      if fim='0' then Eprox <= espera;
47                                  else            Eprox <= final;
48                                  end if;
49
50         when final =>            Eprox <= inicial;
51
52         when others =>           Eprox <= inicial;
53
54       end case;
55
56     end process;
57
58     -- logica de saida (Moore)
59     with Eatual select
60         carrega <= '1' when preparacao, '0' when others;
61
62     with Eatual select
63         zera <= '1' when preparacao, '0' when others;
64
65     with Eatual select
66         desloca <= '1' when transmissao, '0' when others;
67
68     with Eatual select
69         conta <= '1' when transmissao, '0' when others;
70
71     with Eatual select
72         pronto <= '1' when final, '0' when others;
73
74    with Eatual select
75       db_estado <= "001" when inicial,
76                    "010" when preparacao,
77                    "011" when espera,
78                    "100" when transmissao,
79                    "101" when final,
80                    "000" when others;
81
82 end tx_serial_tick_uc_arch;
```

5.3.2 contador_m

```
1 ibrary ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity contador_m is
6     generic (
```

```
7         constant M: integer := 50;  -- modulo do contador
8         constant N: integer := 6    -- numero de bits da saida
9     );
10    port (
11        clock, zera, conta: in std_logic;
12        Q: out std_logic_vector (N-1 downto 0);
13        fim: out std_logic
14    );
15 end contador_m;
16
17 architecture contador_m_arch of contador_m is
18    signal IQ: integer range 0 to M-1;
19 begin
20
21    process (clock,zera,conta,IQ)
22    begin
23        if zera='1' then IQ <= 0;
24        elsif clock'event and clock='1' then
25            if conta='1' then
26                if IQ=M-1 then IQ <= 0;
27                else IQ <= IQ + 1;
28                end if;
29            end if;
30        end if;
31
32        if IQ=M-1 then fim <= '1';
33        else fim <= '0';
34        end if;
35
36        Q <= std_logic_vector(to_unsigned(IQ, Q'length));
37
38    end process;
39 end contador_m_arch;
```

5.3.3 deslocador_n

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity deslocador_n is
5      generic (
6          constant N: integer := 11
7      );
8      port (
9          clock, reset:                      in  std_logic;
10         carrega, desloca, entrada_serial: in  std_logic;
11         dados:                             in  std_logic_vector (N-1 downto 0);
12         saida:                             out std_logic_vector (N-1 downto 0);
13       db_deslocado:              out std_logic_vector (N-1 downto 0)
14     );
15 end deslocador_n;
16
17 architecture deslocador_n_arch of deslocador_n is
18
19     signal IQ: std_logic_vector (N-1 downto 0);
20
21 begin
22
23     process (clock, reset, IQ)
24     begin
25         if reset='1' then IQ <= (others=>'1');
26         elsif (clock'event and clock='1') then
27             if carrega='1' then IQ <= dados;
```

```
28              elsif desloca='1' then IQ <= entrada_serial & IQ(N-1 downto 1);
29              else IQ <= IQ;
30              end if;
31          end if;
32          saida <= IQ;
33        db_deslocado <= IQ;
34      end process;
35
36
37  end deslocador_n_arch;
```

5.3.4 edge_detector

```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3
4  entity edge_detector is
5      port ( clk          : in    std_logic;
6             signal_in    : in    std_logic;
7             output       : out   std_logic
8      );
9  end edge_detector;
10
11 architecture Behavioral of edge_detector is
12       signal signal_d: std_logic;
13 begin
14     process(clk)
15     begin
16         if clk= '1' and clk'event then
17             signal_d <= signal_in;
18         end if;
19     end process;
20
21     output<= (not signal_d) and signal_in;
22
23 end Behavioral;
```

6 REFERÊNCIAS BIBLIOGRÁFICAS

(1) ALMEIDA, F.V. de; SATO, L.M.; MIDORIKAWA, E.T. Tutorial para criação de circuitos digitais em VHDL no Quartus Prime 16.1. Apostila de Laboratório Digital. Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da USP. Edição de 2017.

(2) ALMEIDA, F.V. de; SATO, L.M.; MIDORIKAWA, E.T. Tutorial para criação de circuitos digitais hierárquicos em VHDL no Quartus Prime 16.1. Apostila de Laboratório Digital. Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da USP. Edição de 2017.

(3) ALTERA / Intel. DE0-CV User Manual. 2015.

(4) ALTERA / Intel. Quartus Prime Introduction Using VHDL Designs. 2016.

(5) ALTERA / Intel. Quartus Prime Introduction to Simulation of VHDL Designs. 2016.

(6) D'AMORE, R. VHDL - descrição e síntese de circuitos digitais. 2a edição, LTC, 2012.

(7) WAKERLY, John F. Digital Design Principles & Practices. 4th edition, Prentice Hall, 2006.