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 - EXPERIÊNCIA 3

Prof. Edson Midorikawa

Prof. Paulo Cugnasca

Prof. Reginaldo Arakaki

SÃO PAULO - SP

2021

SUMÁRIO

1 INTRODUÇÃO 3
1.1 Visão Geral: Transmissão serial 3
1.2 Funcionalidade 3
1.3 Não Funcionalidade 3
2 SOLUÇÃO TÉCNICA 4
2.1 Descrição Geral 4
3 ATIVIDADES 5
3.1 Atividade 1 5
3.2 Atividade 2 8
3.3 Atividade 3 12
3.4 Atividade 4 13
4 RESULTADOS 20
5 DESAFIO 21
5.1 Descrição Geral 21
5.2 Montagem 21
5.3 Testagem 22
6 APÊNDICE. 24
6.1 Circuito Base 24
6.1.1 rx_serial_8N2 24
6.1.2 rx_serial_8N2_fd 26
6.1.3 rx_serial_8N2_tick_uc 28
6.2 Desafio 30
6.2.1 uart_8N2 30
6.2.2 Testbench 32
7 REFERÊNCIAS BIBLIOGRÁFICAS 38

1 INTRODUÇÃO

1.1 Visão Geral: Transmissão serial

Durante este experimento, os alunos terão que aprender sobre transmissão serial, mais especificamente um receptor de um sinal serial. Será desenvolvido um componente que recebe um valor em ao longo do tempo e transmite o valor recebido (de maneira vetorial).

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 recepção serial assíncrona pode ser descrito da seguinte forma:
PREENCHER

1.3 Não Funcionalidade

Durante este experimento, não será implementada
PREENCHER

2 SOLUÇÃO TÉCNICA

2.1 Descrição Geral

Neste experimento da disciplina de Laboratório Digital II, será desenvolvido o projeto de um circuito receptor de comunicações seriais. A especificações fornecidas pelo docente da disciplina e as instruções da apostila serão utilizadas como guia de projeto.

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.

Aplicar o circuito desenvolvido na placa FPGA, testando-o em conjuntura com o suporte à tecnologia *IoT* (Internet of Things) fornecida pelo Laboratório Digital, utilizando o aplicativo **MQTT Dash**. Além disso, utilizará-se os sinais de depuração na placa FPGA para testar o circuito ao vivo.

3 ATIVIDADES

3.1 Atividade 1

a) Explique o funcionamento básico do circuito de recepção serial assíncrona em termos dos principais componentes digitais que fazem parte do fluxo de dados (p.ex. deslocador, contador, etc). Considere nesta descrição um circuito básico com a recepção de um bit por período de clock.

O circuito de recepção serial assíncrona, ao perceber o início do sinal, começa a registrar os sinais recebidos em um deslocador. A cada ciclo de clock, o circuito amostra mais um sinal, e o armazena o bit amostrado no deslocador.

O contador interno ao fluxo de dados é atualizado a cada bit recebido, quando chega ao limite dos bits é liberado um sinal de fim.

Sinalizado o fim, o sinal *registra* ligado a um registrador permite que ele armazene o dado passado pelo deslocador naquele instante.

Esses bits que foram amostrados, já que foram armazenados em um registrador, fornecem essa informação na saída do circuito, permitido assim ler o sinal enviado (em forma de vetor).

---

b) Explique a máquina de estados básica da unidade de controle do circuito de recepção. Considere aqui também um circuito básico para recepção de um bit por período de clock.

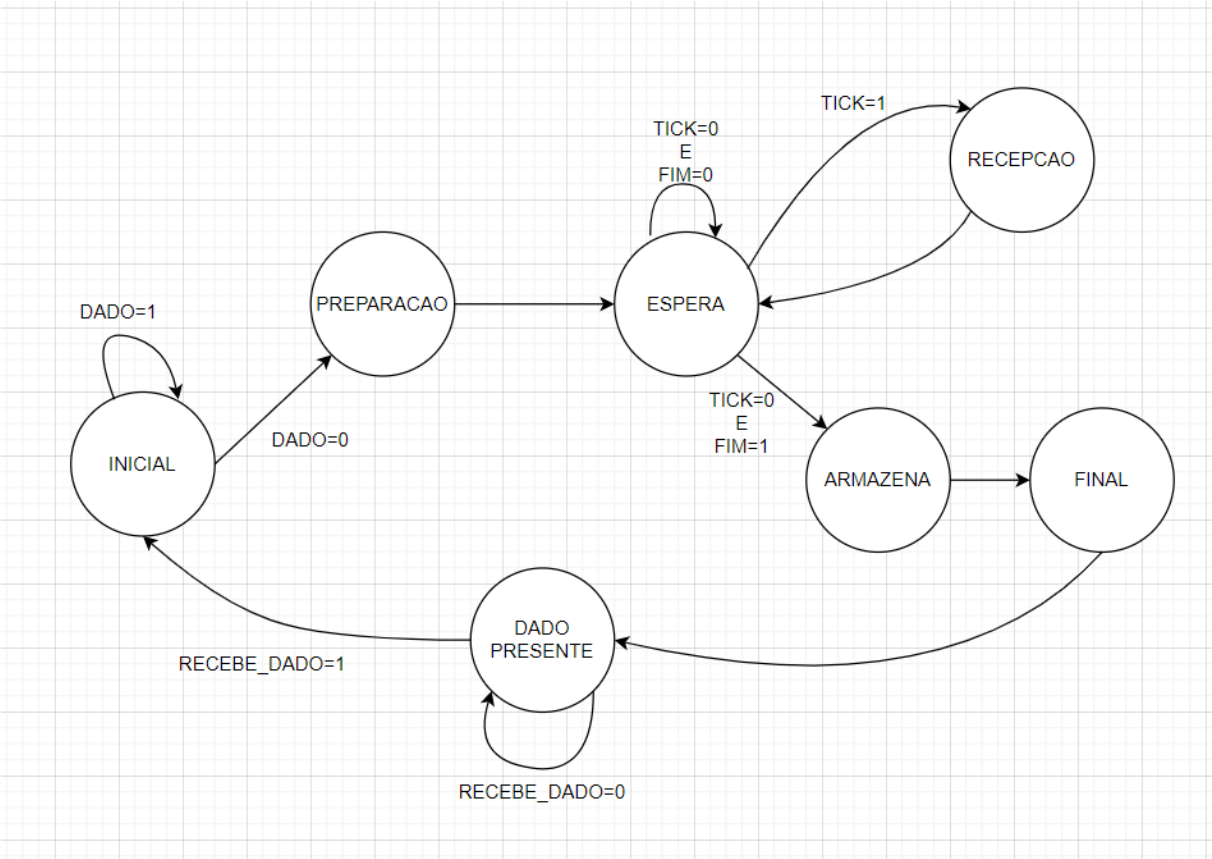

Figura 1: Máquina de estados finita simples

- **inicial**: Estado inicial, espera-se o dado de início;
- **preparação**: Reseta todas as variáveis do sistema;
- **espera**: Espera-se o tick para amostragem do dado ou o sinal de fim para o encerramento do ciclo;
- **recepção**: Ativa-se o deslocamento para salvar o bit atual da transmissão serial e incrementa-se o contador de parada da amostragem;
- **armazena**: O valor que estava armazenado no deslocador é passado para o registrador;
- **final**: O sinal de pronto é ativado indicando o fim da transmissão;

- **dado presente**: O sinal de $tem_dadoativadoindicandoqueodadoestdisponvelnasadaeaguarda-seosinal$

c) Explique o funcionamento da aplicação da técnica de superamostragem para o circuito de recepção serial assíncrona. Que alterações são necessárias na unidade de controle do item b) com a adoção da técnica de superamostragem?

Para utilizar a técnica de superamostragem, primeiramente necessitaria-se de um período de clock bem menor do que a duração da transmissão de um bit. Assim, vários ciclos de clock se passam dentro da transmissão de um bit.

Dessa forma, a superamostragem consiste em amostrar o sinal de maneira mais precisa – isto é, o mais próximo do "meio"da transmissão de um único bit o possível. Isso evita erros de amostragem (amostragem durante a transição entre bits consecutivos).

Além do aumento da frequência do clock do circuito, seria necessária uma sincronização da amostragem com o sinal recebido. Para tanto, ao receber o sinal de start da transmissão, o circuito deve esperar metade da transmissão de 1 bit, e amostrar o sinal de start. Em seguida, deve esperar a duração da transmissão de 1 bit e amostrar novamente – coletando o dado no "meio"da transmissão do próximo bit. Em seguida, essa amostragem (uma a cada a duração de 1 bit) se repete até o fim do sinal (quando chegar nos bits de stop).

Para a contagem ocorrer de forma satisfatória é utilizado um contador de arquitetura especial com uma saída para quando se atingisse o meio da contagem e outro para quando se atingisse o final dela.

---

d) Mostre a relação de frequências entre os sinais de clock e tick. Explique os instantes da ocorrência dos sinais de tick em relação aos bits de dados do sinal serial de entrada do circuito de recepção. DICA: considere por exemplo um clock de 50MHz e uma taxa de comunicação de 9600 bauds

Primeiramente, necessita-se do primeiro tick de amostragem no meio da transmissão do primeiro bit. Considerando um clock de 50MHz e uma taxa de comunicação de 9600 bauds, o número M de ciclos de clock que equivale à metade da transmissão de um bit é:

$$M = \frac{50000000}{9600.2} = 2604,16667 \tag{1}$$

Arredondando este valor, temos 2604 ciclos de clock. Assim, após perceber o o sinal de início de transmissão, o circuito deve aguardar 2604 ciclos de clock para realizar a primeira amostragem.

Em seguida, o circuito deve esperar o dobro do tempo para amostrar da segunda vez para pegar o meio da amostragem do bit seguinte. Já que $M = 2604$ ciclos de clock equivalem a metade do tempo de transmissão de um bit, o dobro – $2M = 5208$ ciclos de clock – equivale ao tempo de transmissão de um bit. Desta forma, o intervalo de toda amostragem após a primeira é de 5208 ciclos de clock.

---

e) Ao comparar o projeto do circuito da experiência com o projeto do circuito de transmissão serial, o que a adoção da superamostragem muda no desenvolvimento do fluxo de dados e da unidade de controle?

A adoção da superamostragem implica nas seguintes modificações:

No fluxo de dados: - Será necessário o uso de um contador especial com sinais de saída que indicam o meio e o fim da contagem.

Na unidade de controle: - A unidade de controle funciona com um número elevado de trocas entre os estados de espera e recepção.

3.2 Atividade 2

Esta atividade envolve o desenvolvimento do projeto lógico do circuito de recepção serial assíncrona, usando a técnica de superamostragem. A configuração da comunicação serial deve ser 8N2 com uma taxa de comunicação de 9600 bauds.

Especificação do Projeto: O circuito de recepção de dados seriais recebe a sequência de bits proveniente de um terminal serial pela entrada dado_serial. Ao detectar o envio de um dado serial, o circuito deve mostrar seu código ASCII (dado_recebido) e acionar um led para indicar a recepção do dado (tem_dado).

Este sinal tem_dado deve permanecer ativado até que o circuito receba o acionamento do sinal de entrada recebe_dado, indicando que o dado recebido foi registrado (p.ex. pelo módulo principal do sistema digital). O circuito não deve descartar este dado recebido até que o sinal recebe_dado seja acionado (manter registrado o dado, mesmo com a vinda de outro dado serial pela linha de comunicação).

Ao receber a confirmação de recebimento do dado pelo acionamento de recebe_dado,

o circuito deve desativar a saída tem_dado. Somente a partir deste momento, o circuito poderá receber novos dados seriais.

Estes sinais fazem parte de um circuito de interface entre o circuito de recepção serial e o sistema digital que faz uso da comunicação serial. Isto está ilustrado nas formas de onda da figura 1.

A documentação do circuito deve detalhar os elementos do fluxo de dados e seu controle pela unidade de controle. A interface do circuito é mostrada na Figura 2. Sinais adicionais de depuração podem ser especificados e devem ser documentados no Planejamento. A configuração da comunicação serial a ser adotada no projeto é denominada **8N2**, ou seja, 8 bits de dados, sem bit de paridade e 2 stop bits.

A entidade principal do circuito de recepção serial deve seguir a especificação dada.

```
entity rx_serial_8N2 is
    port (
        clock: in std_logic;
        reset: in std_logic;
        dado_serial: in std_logic;
        recebe_dado: in std_logic;
        pronto_rx: out std_logic;
        tem_dado: out std_logic;
        dado_recebido: out std_logic_vector (7 downto 0);
        db_estado: out std_logic_vector (6 downto 0)
    );
end entity;
```

Note que foi especificada uma saída de depuração para a apresentação do estado da unidade de controle em um display de 7 segmentos da placa FPGA.

f) Incluir na documentação do projeto os diagramas de funcionamento usados no projeto (diagrama de estados da unidade de controle, diagrama de blocos do fluxo de dados, etc).

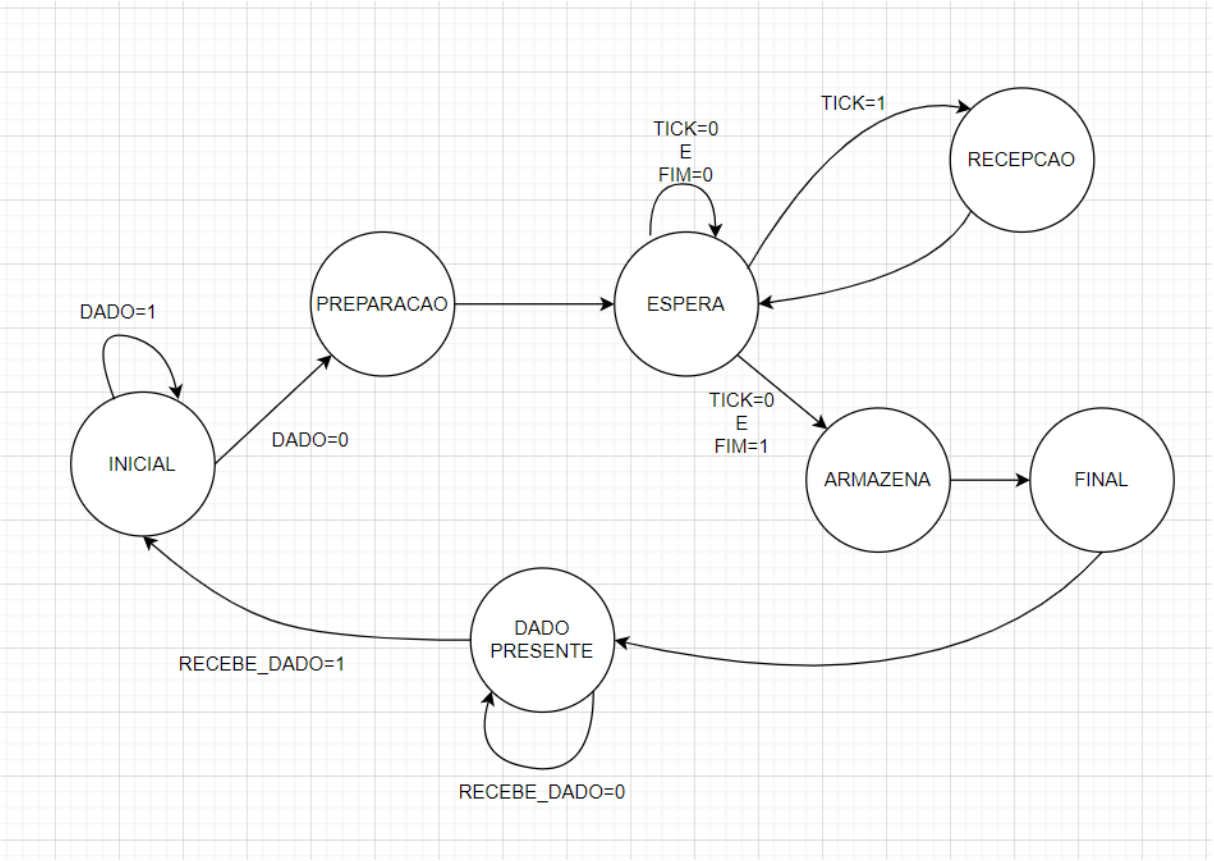

Figura 2: Máquina de estados da unidade de controle

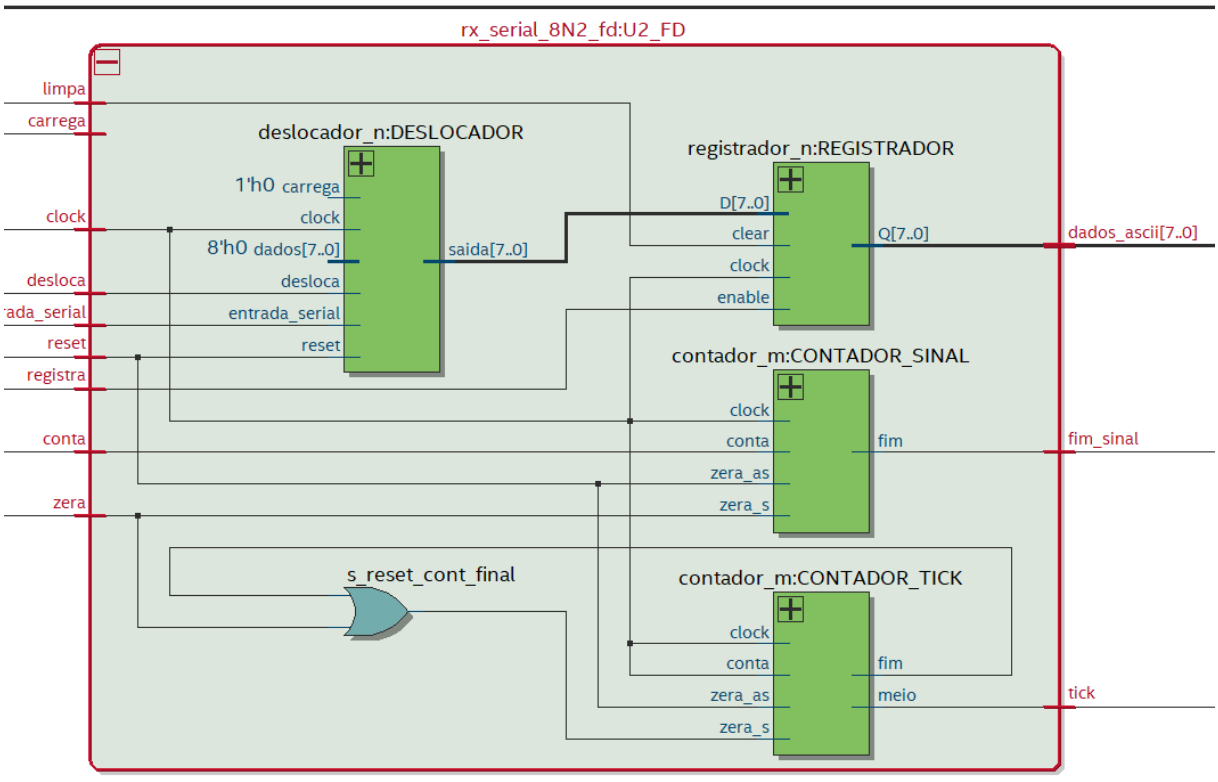

Figura 3: Diagrama de blocos do fluxo de dados

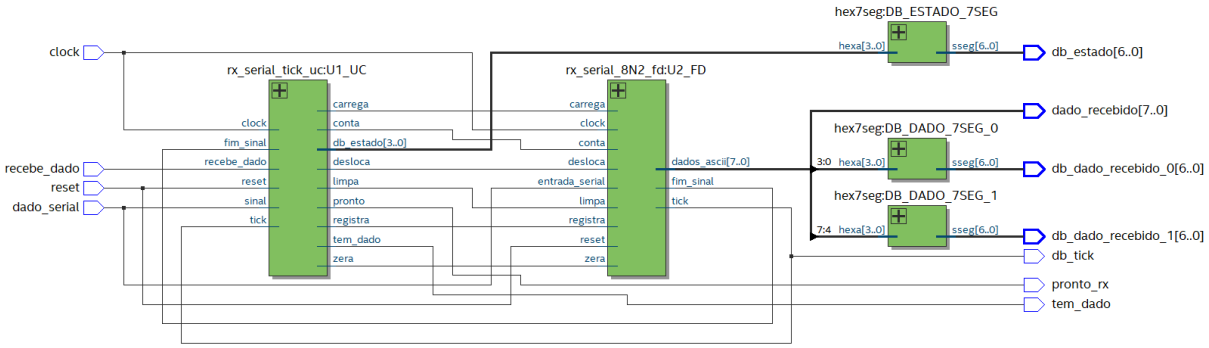

Figura 4: Diagrama de blocos geral do circuito

---

g) O Planejamento deve conter a descrição do projeto em VHDL de cada um dos componentes e módulos, uma explicação da integração destas partes e, finalmente, uma descrição do funcionamento do projeto completo.

Na visão global do circuito ele é composto por 5 componentes gerais, os 3 componentes de conversão para 7 segmentos, o fluxo de dados e a unidade de controle.

A unidade de controle é responsável pelos sinais de controle e a determinação do funcionamento do fluxo de dados em cada estado junto com a determinação do estado atual.

O fluxo de dados é composto por 3 tipos diferentes de componentes, há um deslocador com função de gerar os dados ascii a partir da agregação dos bits do sinal serial, um registrador responsável por guardar o resultado final do deslocador e 2 contadores, um que determina o fim da transmissão e outro que determina o tick para a superamostragem.

O circuito começa no estado inicial em que espera pelo bit de start '0', quando recebido esse bit muda-se o estado entre o de amostragem do bit e o de espera do bit seguinte. Terminado a transmissão do bit, o que é determinado pela contagem dos 8 bits totais esperados, o circuito deixa em alta um sinal indicando a disponibilidade do dado. Quando o sinal *recebe_dados* for ativado o circuito voltará para o estado inicial.

---

h) Defina os casos de testes devem ser executados para assegurar o correto funcionamento do circuito completo. Redija um Plano de Testes a ser empregado tanto na simulação como nos testes na placa FPGA.

Para os casos de teste foram adotados os exemplos dados no testbench *rx_serial_tb.vhd* fornecido, sendo esses:

- 00110101
- 01010101
- 10101010
- 11111111
- 00000000

com suas respectivas saídas esperadas:

- 10101100
- 10101010
- 01010101
- 11111111
- 00000000

Foram incluídos alguns sinais de depuração para facilitar o processo de debbuging do circuito, sendo esses *db_tick_out*, *db_estado_out*, *db_dado_recebido_0* e *db_dado_recebido_1*.

3.3 Atividade 3

Esta atividade envolve a simulação do projeto lógico do circuito de recepção serial assíncrona, usando o software **ModelSim**, disponível com o Intel Quartus Prime.

i) Estude a descrição do testbench fornecido para o circuito de recepção serial (arquivo rx_serial_tb.vhd).

---

j) Ajuste o testbench para verificar o funcionamento do circuito de recepção serial, incluindo os casos de teste especificadas no item h). Documente o código VHDL do testbench no Planejamento.

Seguiu-se os casos já disponibilizados no testbench.

---

k) Simule a recepção de dados com o software ModelSim e inclua as formas de onda no Planejamento. A figura 3 ilustra uma possível saída da simulação com ModelSim.

O resultado dos testes foi de acordo com o esperado como pode ser verificado abaixo

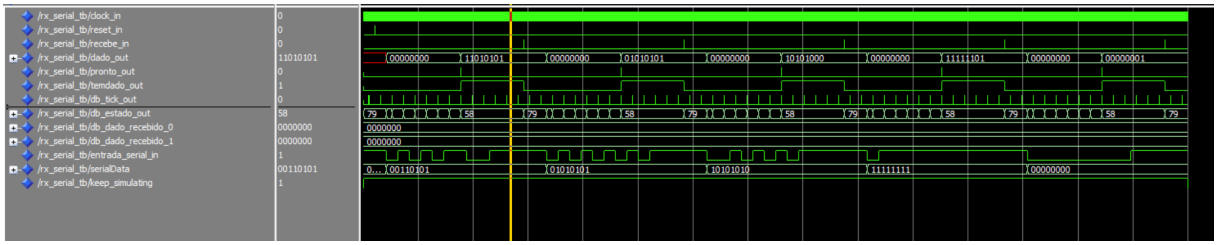

Figura 5: Resultado do testbench

---

l) Submeter o arquivo QAR (exp3-txby.qar) do projeto e o arquivo VHDL do testbench junto com o Planejamento.

3.4 Atividade 4

Neste item vamos implementar, no Laboratório Digital, o projeto do circuito de recepção serial com configuração em 8N2 e 9600 bauds na placa FPGA DE0-CV.

---

m) Inicialmente, realizaremos testes do circuito com auxílio do Analog Discovery. A entrada de dados seriais deverá ser gerada a partir da ferramenta Protocol (Analisador de Protocolos) configurada para o protocolo RS232C com comunicação em 8N2 na aba UART da ferramenta. O caractere ASCII a ser enviado deverá ser digitado no campo de transmissão (TX) e será transmitido ao pressionar o botão Send, conforme a figura 4 abaixo.

---

n) Sintetizar o circuito de recepção serial para a placa FPGA DE0-CV. Usar a seguinte designação de pinos mínima.

sinal	pino	Analog Discovery
CLOCK	CLOCK_50	-
RESET	**GPIO_0_D27**	***Static I/O – Button 0/1 – DIO7***
RECEBE_DADO	**GPIO_0_D29**	***Static I/O – Button 0/1 – DIO8***
DADO_SERIAL	**GPIO_0_D31**	***Protocol – UART – DIO9***
DADO_RECEBIDO	*leds LEDR[0-7]*	-
PRONTO_RX	*led* LEDR[8]	-
TEM_DADO	*led* LEDR[9]	-
DB_ESTADO	*display* HEX5	-

Figura 6: Tabela de Pinagem mínima para o Analog Discovery

A partir da designação de pinos mínima, os alunos montaram a designação de pinos contendo os sinais de depuração. Além disso, colocaram junto à tabela os pinos que serão utilizados na placa FPGA:

Sinal	Pinos	Analog Discovery	Pino FPGA
CLOCK	CLOCK_50	-	M9
RESET	GPIO_0_D27	Static I/O - Button 0/1 - DIO7	P18
DADO_SERIAL	GPIO_0_D31	Protocol - UART -DIO9	T20
RECEBE_DADO	GPIO_0_D29	Static I/O - Button 0/1 - DIO8	R17
PRONTO_RX	LEDR[8]	-	L2
TEM_DADO	LEDR[9]	-	L1
DADO_RECEBIDO	LEDR[0-7]	-	AA2, AA1, W2, Y3, N2, N1, U2, U1
DB_DADO_RECEBIDO_0	HEX0	-	U21, V21, W22, W21, Y22, Y21, AA22
DB_DADO_RECEBIDO_1	HEX1	-	AA20, AB20, AA19, AA18, AB18, AA17, U22
DB_TICK	-	-	-
DB_ESTADO	HEX2	-	Y19, AB17, AA10, Y14, V14, AB22, AB21
DB_DADO_SERIAL	GPIO_1_D27	Scope - CH1	F15

Figura 7: Tabela de Pinagem com depuração para o Analog Discovery

É notável que não foi inserido o sinal *"db_tick"* para a análise, dado que este sinal não seria visível ao olho nu na placa FPGA. Já que não seria visível, não seria um sinal de depuração útil.

Além disso, foi adicionado o sinal *"db_dado_serial"* para visualizar a forma de onda recebida, verificando se é coerente com o que o circuito está interpretando.

---

o) Programe a placa DE0-CV e execute os testes definidos para o circuito de recepção serial para vários dados seriais, conforme Plano de Teste elaborado no Planejamento. Anote os resultados obtidos.

Após a programação da placa DE0-CV, foi aberto o software de **Waveforms** para enviar o dado serial por meio da ferramenta **Protocol** e analisar o dado serial recebido com a ferramenta **Scope**.

Logo de início, notou-se que havia algo de errado com o circuito, já que a unidade de controle ficava preso no estado de número 3 (na máquina de estados, equivale ao estado "espera"). Os alunos então reavaliaram a forma de ondas gerada pelo **ModelSim** e não encontraram nenhum erro, o que gerou confusão.

Após estudo e testagem do problema, encontrou-se que o erro estava no contador dos ticks no fluxo de dados, cuja entrada *"conta"* estava ligada ao clock. No circuito ideal testado pelo *"ModelSim"*, isso não configurava um erro, mas ao aplicá-lo na placa FPGA, atrasos da aplicação real geravam erros.

Após esse estudo, o circuito foi modificado para que o funcionamento na placa fosse adequado.

---

p) Use ferramentas do Analog Discovery para a visualização dos sinais digitais e depuração do circuito. Adicione figuras com as saídas das ferramentas no Relatório

Exemplos dos resultados obtidos:

Testagem para o envio do sinal "a"

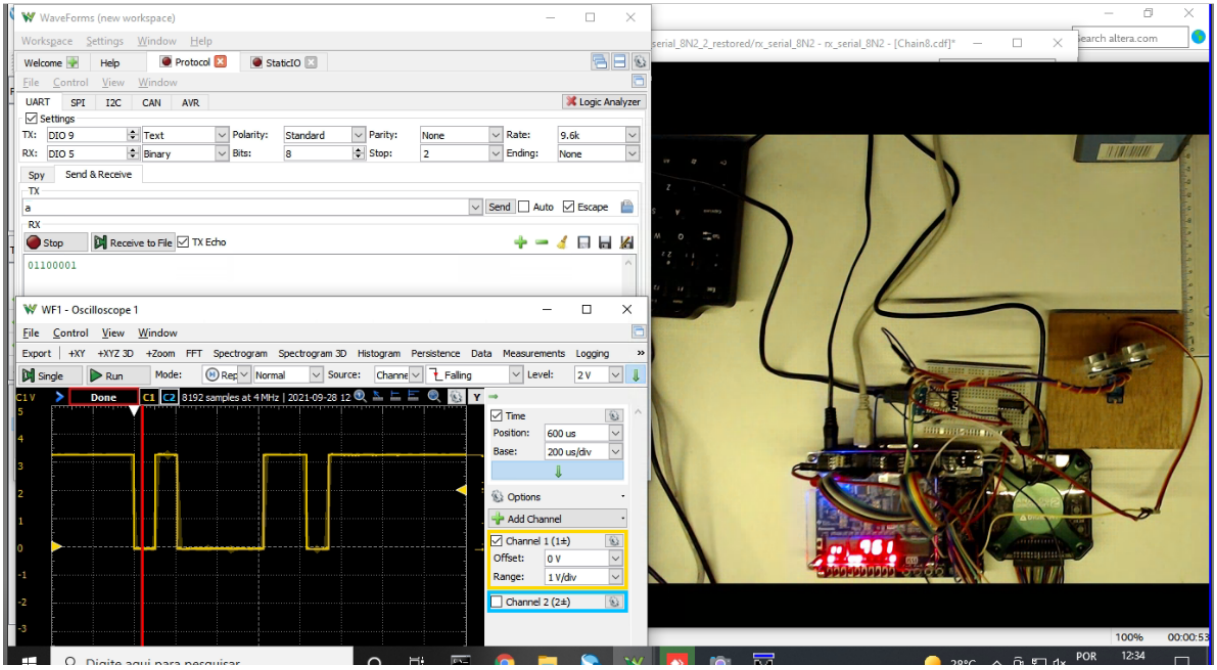

Figura 8: Tela do Anydesk para a testagem do sinal ”a”

Testagem para o envio do sinal ”5”

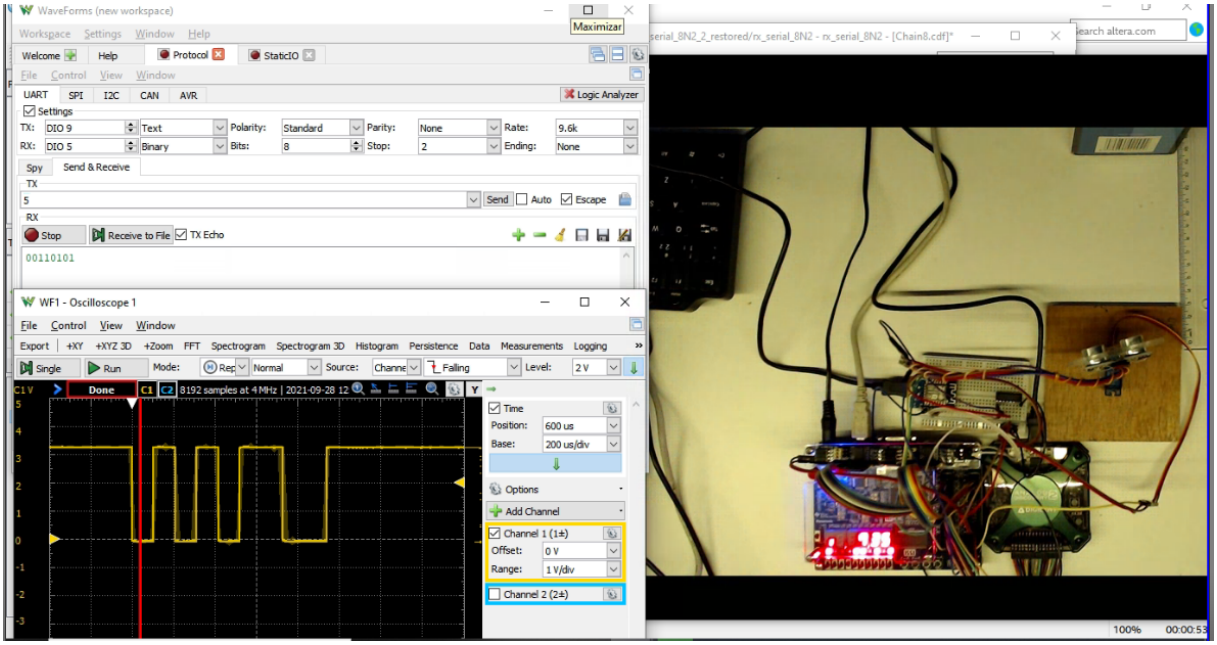

Figura 9: Tela do Anydesk para a testagem do sinal ”5”

Já que o circuito apresentou as saídas corretas para esta configuração, é possível dizer que a implementação foi um sucesso.

---

q) Na segunda parte desta atividade, passaremos a interfacear o circuito de recepção serial via protocolo MQTT. Configure o projeto do MQTT Dash e ajuste o circuito

na FPGA DE0-CV conforme a tabela de designação (mínima) abaixo.

sinal	pino	MQTT
CLOCK	CLOCK_50	-
RESET	**GPIO_0_D0**	**E0 (Switch/button)**
RECEBE_DADO	**GPIO_0_D1**	**E1 (Switch/button)**
DADO_SERIAL	**GPIO_0_D12**	**RX (Text)**
DADO_RECEBIDO	*leds LEDR[0-7]*	-
PRONTO_RX	*led* LEDR[8]	-
TEM_DADO	*led* LEDR[9]	-
DB_ESTADO	*display* HEX5	-

Figura 10: Tabela de Pinagem mínima para o MQTT Dash

A partir da pinagem mínima, os alunos montaram a pinagem com os sinais de depuração, para facilitar o entendimento e testagem do circuito. Além disso, adicionaram-se os pinos da placa FPGA.

Sinal	Pinos	MQTTDash	Pino FPGA
CLOCK	CLOCK_50	-	M9
RESET	GPIO_0_D0	E0 (Switch/Button)	N16
DADO_SERIAL	GPIO_0_D12	RX (Text)	R21
RECEBE_DADO	GPIO_0_D1	E1 (Switch/Button)	B16
PRONTO_RX	LEDR[8]	-	L2
TEM_DADO	LEDR[9]	-	L1
DADO_RECEBIDO	LEDR[0-7]	-	AA2, AA1, W2, Y3, N2, N1, U2, U1
DB_DADO_RECEBIDO_0	HEX0	-	U21, V21, W22, W21, Y22, Y21, AA22
DB_DADO_RECEBIDO_1	HEX1	-	AA20, AB20, AA19, AA18, AB18, AA17, U22
DB_TICK	-	-	-
DB_ESTADO	HEX2	-	Y19, AB17, AA10, Y14, V14, AB22, AB21
DB_DADO_SERIAL	GPIO_1_D27	Scope - CH1	F15

Figura 11: Tabela de Pinagem de depuração para o MQTT Dash

r) Sintetizar o circuito de recepção serial e programe o projeto na placa FPGA DE0-CV.

---

s) Testar o circuito e documentar os resultados experimentais obtidos. Anexe figuras comprovando o funcionamento do circuito.

Após a organização do MQTT Dash, o funcionamento do circuito pode ser visto por meio destas imagens:

Testagem para o envio do sinal "a"

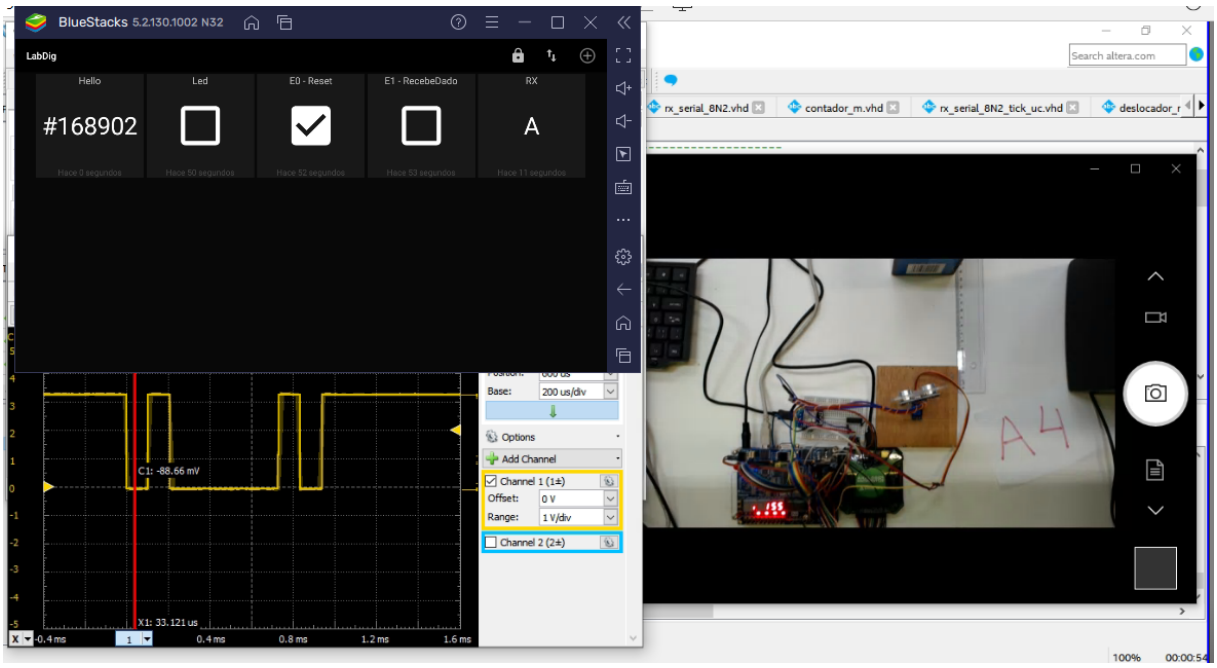

Figura 12: Funcionamento do circuito com o MQTT Dash para o sinal "a"

Testagem para o envio do sinal "5"

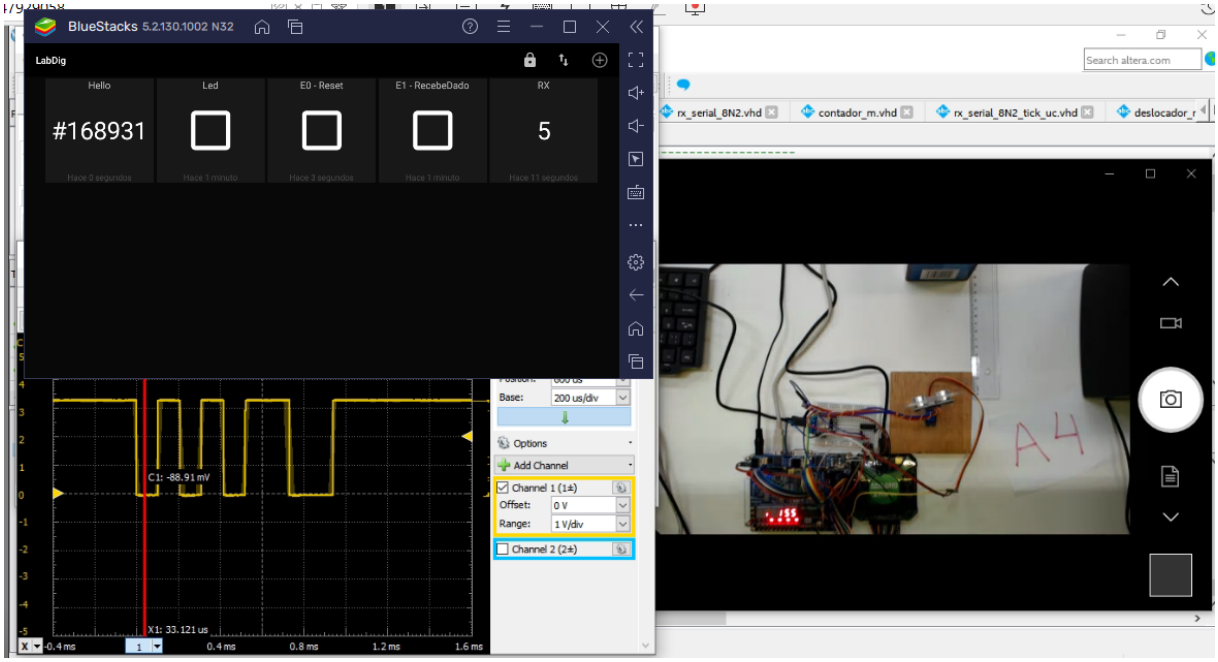

Figura 13: Funcionamento do circuito com o MQTT Dash para o sinal ”5”

---

t) Submeter o arquivo QAR final do projeto (exp3-final-txby.qar) junto com o Relatório.

---

4 RESULTADOS

Os resultados obtidos durante o planejamento foram bem satisfatórios, mas durante a aplicação do circuito na placa FPGA percebeu-se uma série de defeitos que só aconteceram na aplicação real.

Isso ressaltou a importância de sinais de depuração e da apresentação dos mesmos por meio dos LEDs e Displays da FPGA: sem estes, não seria possível a depuração e verificação do erro. Aprendeu-se, com esta inconveniência, que nem sempre a simulação de ondas é congruente com a realidade do circuito, e por isso, deve-se sempre verificar se o circuito não é sensível a essas alterações que ocorrem no sistema real.

Feitas as devidas correções, a aplicação do circuito foi um sucesso, o que leva a caracterizar um bom resultado durante a aplicação de um circuito de recepção serial neste experimento.

No entanto, essa correção tomou um tempo considerável, e impossibilitou a realização do desafio durante a aula de laboratório. Desta forma, o mesmo foi realizado posteriormente, e desenvolvido nos computadores pessoais dos alunos.

5 DESAFIO

5.1 Descrição Geral

O desafio deste experimento foi a criação de um circuito **UART (Universal Asynchronous Receiver Transmitter)**, que funciona basicamente como um **Receptor Serial** e um **Transmissor Serial** integrados. Existem modelos mais complexos e específicos para diferentes aplicações, mas o especificado para este laboratório foi o seguinte:

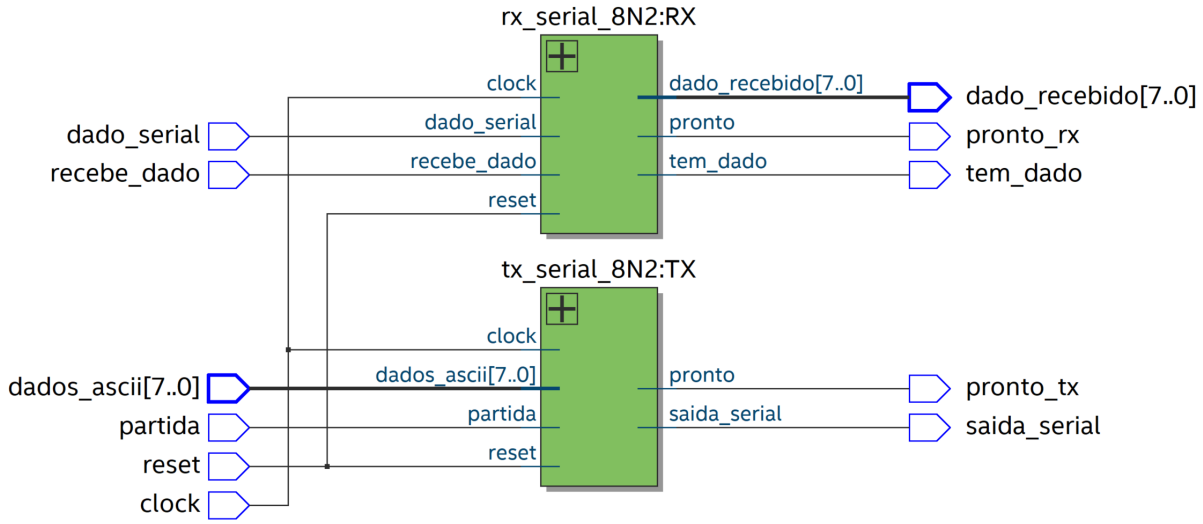

Figura 14: RTL de uma UART fornecida pelo docente

5.2 Montagem

Para a montagem, os alunos optaram por manter alguns dos sinais de depuração de ambos os circuitos. Utilizaram-se os códigos VHD desenvolvidos na segunda experiência (para o transmissor serial) e na terceira experiência (para o receptor serial).

O seguinte diagrama foi gerado no **RTL Viewer**:

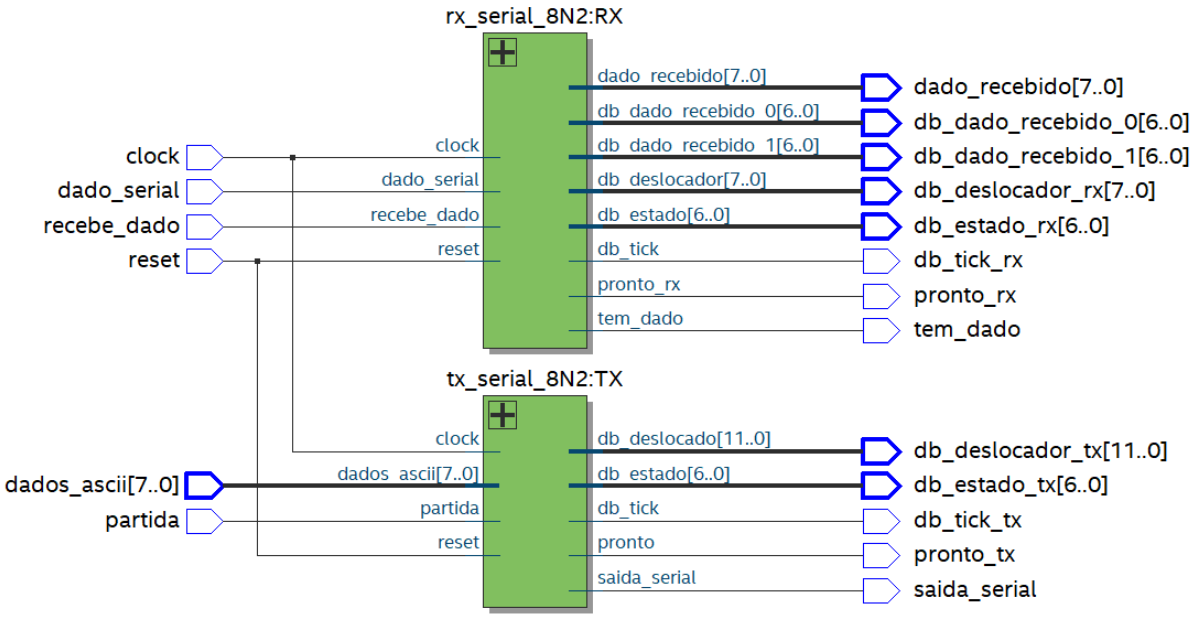

Figura 15: RTL da UART desenvolvida pelos alunos

5.3 Testagem

Para a testagem do sistema, a saída do transmissor serial (*saida_serial*) foi utilizada como entrada do receptor serial (*entrada_serial*). Dessa forma, se ambos os componentes estiverem funcionando corretamente, o sinal inserido em *dados_ascii* deve ser apresentado em *dado_recebido*.

Para isso, foi utilizado o testbench desenvolvido na segunda experiência para o circuito de transmissão serial, e analisou-se as formas de onda no ModelSim. Com isso, gerou-se um testbench específico para a UART: **uart_tb**.

Testou-se o circuito para os seguintes sinais de entrada para *"dados_ascii"*: **"00110101"** (o caractere '5') e **"00101001"** (o caractere ')'). As ondas obtidas foram as seguintes:

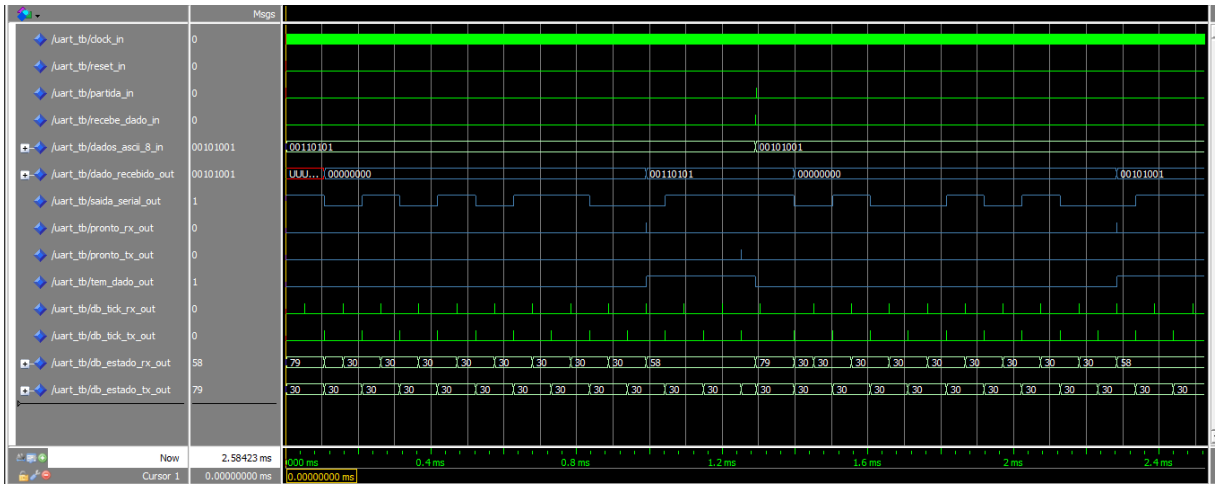

Figura 16: Formas de ondas obtidas no testbench da UART

Em especial, olhando as ondas azuis, é possível notar que o *"dado_recebido"* é igual ao dado enviado nos *"dados_ascii"*, e como o dado serial também é congruente, é possível concluir que a UART está funcionando corretamente.

6 APÊNDICE

Os códigos completos podem ser encontrados no repositório **Github:** https://github.com/Gabrie

6.1 Circuito Base

6.1.1 rx_serial_8N2

```
1  ------------------------------------------------------------------
2  -- Arquivo    : rx_serial_8N2.vhd
3  -- Projeto    : Experiencia 3 - Recepcao Serial Assincrona
4  ------------------------------------------------------------------
5  -- Descricao : fluxo de dados do circuito da experiencia 3
6  --              > implementa configuracao 8N2
7  ------------------------------------------------------------------
8  -- Revisoes  :
9  --      Data        Versao  Autor             Descricao
10 --      24/09/2021  1.0     Gabriel Kishida   versao inicial
11 ------------------------------------------------------------------
12
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15 USE ieee.numeric_std.ALL;
16 USE IEEE.math_real.ALL;
17
18 ENTITY rx_serial_8N2 IS
19     PORT (
20         clock : IN STD_LOGIC;
21         reset : IN STD_LOGIC;
22         dado_serial : IN STD_LOGIC;
23         recebe_dado : IN STD_LOGIC;
24         pronto_rx : OUT STD_LOGIC;
25         tem_dado : OUT STD_LOGIC;
26         dado_recebido : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
27         db_dado_recebido_0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
28         db_dado_recebido_1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
29         db_tick : OUT STD_LOGIC;
30         db_estado : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
31         db_dado_serial : OUT STD_LOGIC;
32         db_deslocador : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
33     );
34 END ENTITY;
35
36 ARCHITECTURE rx_serial_8N2_arch OF rx_serial_8N2 IS
37
38     COMPONENT rx_serial_tick_uc PORT (
39         clock : IN STD_LOGIC;
40         reset : IN STD_LOGIC;
41         sinal : IN STD_LOGIC;
42         tick : IN STD_LOGIC;
43         fim_sinal : IN STD_LOGIC;
44         recebe_dado : IN STD_LOGIC;
45         zera : OUT STD_LOGIC;
46         conta : OUT STD_LOGIC;
47         carrega : OUT STD_LOGIC;
```

```
48          pronto : OUT STD_LOGIC;
49          tem_dado : OUT STD_LOGIC;
50          registra : OUT STD_LOGIC;
51          limpa : OUT STD_LOGIC;
52          desloca : OUT STD_LOGIC;
53          db_estado : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
54          );
55      END COMPONENT;
56
57      COMPONENT rx_serial_8N2_fd PORT (
58          clock, reset : IN STD_LOGIC;
59          zera, conta, carrega, desloca : IN STD_LOGIC;
60          limpa, registra : IN STD_LOGIC;
61          entrada_serial : IN STD_LOGIC;
62          tick, fim_sinal : OUT STD_LOGIC;
63          dados_ascii : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
64          db_deslocador : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
65          );
66      END COMPONENT;
67      COMPONENT hex7seg PORT (
68          hexa : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
69          sseg : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
70          );
71      END COMPONENT;
72
73      SIGNAL s_zera, s_conta, s_carrega, s_desloca, s_tick, s_conta_baud : STD_LOGIC;
74      SIGNAL s_fim_sinal, s_registra, s_limpa, s_dado_serial : STD_LOGIC;
75
76      SIGNAL s_dados_ascii : STD_LOGIC_VECTOR(7 DOWNTO 0);
77      SIGNAL s_db_estado : STD_LOGIC_VECTOR (3 DOWNTO 0);
78
79  BEGIN
80      -- unidade de controle
81      U1_UC : rx_serial_tick_uc PORT MAP(
82          clock,
83          reset,
84          s_dado_serial,
85          s_tick,
86          s_fim_sinal,
87          recebe_dado,
88          s_zera,
89          s_conta,
90          s_carrega,
91          pronto_rx,
92          tem_dado,
93          s_registra,
94          s_limpa,
95          s_desloca,
96          s_db_estado
97      );
98
99      -- fluxo de dados
100     U2_FD : rx_serial_8N2_fd PORT MAP(
101         clock,
102         reset,
103         s_zera,
104         s_conta,
105         s_carrega,
106         s_desloca,
107         s_limpa,
108         s_registra,
109         s_dado_serial,
110         s_tick,
111         s_fim_sinal,
112         s_dados_ascii,
113         db_deslocador
```

```
114     );
115
116     DB_ESTADO_7SEG : hex7seg PORT MAP(
117         hexa => s_db_estado(3 DOWNTO 0),
118         sseg => db_estado
119     );
120
121     DB_DADO_7SEG_0 : hex7seg PORT MAP(
122         hexa => s_dados_ascii(3 DOWNTO 0),
123         sseg => db_dado_recebido_0
124     );
125
126     DB_DADO_7SEG_1 : hex7seg PORT MAP(
127         hexa => s_dados_ascii(7 DOWNTO 4),
128         sseg => db_dado_recebido_1
129     );
130     s_dado_serial <= dado_serial;
131     db_dado_serial <= s_dado_serial;
132     dado_recebido <= s_dados_ascii;
133     db_tick <= s_tick;
134
135 END ARCHITECTURE;
```

6.1.2 rx_serial_8N2_fd

```
1
2  ------------------------------------------------------------------
3  -- Arquivo    : rx_serial_8N2_fd.vhd
4  -- Projeto    : Experiencia 3 - Recepcao Serial Assincrona
5  ------------------------------------------------------------------
6  -- Descricao : fluxo de dados do circuito da experiencia 3
7  --              > implementa configuracao 8N2
8  ------------------------------------------------------------------
9  -- Revisoes  :
10 --     Data        Versao  Autor             Descricao
11 --     24/09/2021  1.0     Gabriel Kishida   versao inicial
12 ------------------------------------------------------------------
13 --
14
15 LIBRARY ieee;
16 USE ieee.std_logic_1164.ALL;
17 USE ieee.numeric_std.ALL;
18 USE IEEE.math_real.ALL;
19
20 ENTITY rx_serial_8N2_fd IS
21     PORT (
22         clock, reset : IN STD_LOGIC;
23         zera, conta, carrega, desloca : IN STD_LOGIC;
24         limpa, registra : IN STD_LOGIC;
25         entrada_serial : IN STD_LOGIC;
26         tick, fim_sinal : OUT STD_LOGIC;
27         dados_ascii : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
28         db_deslocador : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
29     );
30 END ENTITY;
31
32 ARCHITECTURE rx_serial_8N2_fd_arch OF rx_serial_8N2_fd IS
33
34     COMPONENT deslocador_n
35         GENERIC (
36             CONSTANT N : INTEGER
37         );
38         PORT (
```

```
39              clock, reset : IN STD_LOGIC;
40              carrega, desloca, entrada_serial : IN STD_LOGIC;
41              dados : IN STD_LOGIC_VECTOR (N - 1 DOWNTO 0);
42              saida : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0)
43          );
44      END COMPONENT;
45
46      COMPONENT contador_m
47          GENERIC (
48              CONSTANT M : INTEGER
49          );
50          PORT (
51              clock, zera_as, zera_s, conta : IN STD_LOGIC;
52              Q : OUT STD_LOGIC_VECTOR (NATURAL(ceil(log2(real(M)))) - 1 DOWNTO 0);
53              fim, meio : OUT STD_LOGIC
54          );
55      END COMPONENT;
56
57      COMPONENT registrador_n
58          GENERIC (
59              CONSTANT N : INTEGER
60          );
61          PORT (
62              clock : IN STD_LOGIC;
63              clear : IN STD_LOGIC;
64              enable : IN STD_LOGIC;
65              D : IN STD_LOGIC_VECTOR (N - 1 DOWNTO 0);
66              Q : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0)
67          );
68      END COMPONENT;
69
70      SIGNAL s_reset_cont : STD_LOGIC;
71      SIGNAL s_reset_cont_final : STD_LOGIC;
72      SIGNAL s_dados : STD_LOGIC_VECTOR(7 DOWNTO 0);
73
74  BEGIN
75
76      DESLOCADOR : deslocador_n GENERIC MAP(
77          N => 8) PORT MAP (
78          clock,
79          reset,
80          '0',
81          desloca,
82          entrada_serial,
83          "00000000",
84          s_dados
85      );
86
87      CONTADOR_SINAL : contador_m GENERIC MAP(
88          M => 10) PORT MAP (
89          clock,
90          reset,
91          zera,
92          conta,
93          OPEN,
94          fim_sinal,
95          OPEN
96      );
97
98      CONTADOR_TICK : contador_m GENERIC MAP(
99          M => 5208) PORT MAP (
100         clock,
101         reset,
102         zera, -- s_reset_cont_final,
103         '1', -- clock,
104         OPEN,
```

```
105         OPEN, -- s_reset_cont,
106         tick
107     );
108     REGISTRADOR : registrador_n GENERIC MAP(
109         N => 8) PORT MAP(
110         clock,
111         limpa,
112         registra,
113         s_dados,
114         dados_ascii
115     );
116
117     db_deslocador <= s_dados;
118     s_reset_cont_final <= s_reset_cont OR zera;
119
120 END ARCHITECTURE;
```

6.1.3 rx_serial_8N2_tick_uc

```
1  ------------------------------------------------------------------
2  -- Arquivo    : rx_serial_tick_uc.vhd
3  -- Projeto    : Experiencia 2 - Transmissao Serial Assincrona
4  ------------------------------------------------------------------
5  -- Descricao : circuito da experiencia 2
6  --              > unidade de controle para o circuito
7  --              > de recepcao serial assincrona
8  --              > usa a tecnica de superamostragem com o uso
9  --              > de sinal de tick para recepcao de dados
10 ------------------------------------------------------------------
11 -- Revisoes  :
12 --     Data        Versao  Autor             Descricao
13 --     24/09/2021  1.0     Gabriel Kishida   versao inicial
14 ------------------------------------------------------------------
15 --
16
17 LIBRARY ieee;
18 USE ieee.std_logic_1164.ALL;
19
20 ENTITY rx_serial_tick_uc IS
21     PORT (
22         clock : IN STD_LOGIC;
23         reset : IN STD_LOGIC;
24         sinal : IN STD_LOGIC;
25         tick : IN STD_LOGIC;
26         fim_sinal : IN STD_LOGIC;
27         recebe_dado : IN STD_LOGIC;
28         zera : OUT STD_LOGIC;
29         conta : OUT STD_LOGIC;
30         carrega : OUT STD_LOGIC;
31         pronto : OUT STD_LOGIC;
32         tem_dado : OUT STD_LOGIC;
33         registra : OUT STD_LOGIC;
34         limpa : OUT STD_LOGIC;
35         desloca : OUT STD_LOGIC;
36         db_estado : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
37     );
38 END ENTITY;
39
40 ARCHITECTURE rx_serial_tick_uc_arch OF rx_serial_tick_uc IS
41
42     TYPE tipo_estado IS (
43         inicial,
44         preparacao,
```

```
45          espera,
46          recepcao,
47          armazena,
48          final,
49          dado_presente
50      );
51
52      SIGNAL Eatual : tipo_estado; -- estado atual
53      SIGNAL Eprox : tipo_estado; -- proximo estado
54
55  BEGIN
56
57      -- memoria de estado
58      PROCESS (reset, clock)
59      BEGIN
60          IF reset = '1' THEN
61              Eatual <= inicial;
62          ELSIF clock'event AND clock = '1' THEN
63              Eatual <= Eprox;
64          END IF;
65      END PROCESS;
66
67      -- logica de proximo estado
68      PROCESS (sinal, tick, fim_sinal, Eatual, recebe_dado)
69      BEGIN
70          CASE Eatual IS
71              WHEN inicial => IF sinal = '0' THEN
72                  Eprox <= preparacao;
73              ELSE
74                  Eprox <= inicial;
75          END IF;
76
77          WHEN preparacao => Eprox <= espera;
78
79          WHEN espera => IF (tick = '1') THEN
80          Eprox <= recepcao;
81      ELSIF (fim_sinal = '1') THEN
82          Eprox <= armazena;
83      ELSE
84          Eprox <= espera;
85      END IF;
86      WHEN recepcao => Eprox <= espera;
87      WHEN armazena => Eprox <= final;
88
89      WHEN final => Eprox <= dado_presente;
90
91      WHEN dado_presente => IF recebe_dado = '1' THEN
92      Eprox <= inicial;
93  ELSE
94      Eprox <= dado_presente;
95  END IF;
96
97  WHEN OTHERS => Eprox <= inicial;
98
99  END CASE;
100 END PROCESS;
101
102 -- logica de saida (Moore)
103 WITH Eatual SELECT
104     carrega <= '1' WHEN preparacao, '0' WHEN OTHERS;
105
106 WITH Eatual SELECT
107     limpa <= '1' WHEN preparacao, '0' WHEN OTHERS;
108
109 WITH Eatual SELECT
110     registra <= '1' WHEN armazena, '0' WHEN OTHERS;
```

```
111
112 WITH Eatual SELECT
113     zera <= '1' WHEN preparacao, '0' WHEN OTHERS;
114
115 WITH Eatual SELECT
116     conta <= '1' WHEN recepcao, '0' WHEN OTHERS;
117
118 WITH Eatual SELECT
119     pronto <= '1' WHEN final, '0' WHEN OTHERS;
120
121 WITH Eatual SELECT
122     tem_dado <= '1' WHEN dado_presente, '0' WHEN OTHERS;
123
124 WITH Eatual SELECT
125     desloca <= '1' WHEN recepcao, '0' WHEN OTHERS;
126
127 WITH Eatual SELECT
128     db_estado <= "0001" WHEN inicial,
129     "0010" WHEN preparacao,
130     "0011" WHEN espera,
131     "0100" WHEN recepcao,
132     "0101" WHEN armazena,
133     "0110" WHEN final,
134     "0111" WHEN dado_presente,
135     "0001" WHEN OTHERS;
136
137 END rx_serial_tick_uc_arch;
```

6.2 Desafio

6.2.1 uart_8N2

```
1  ------------------------------------------------------------------
2  -- Arquivo    : uart_8N2.vhd
3  -- Projeto    : Experiencia 3 - Recepcao Serial Assincrona
4  ------------------------------------------------------------------
5  -- Descricao : Circuito principal da UART
6  --              > implementa configuracao 8N2
7  ------------------------------------------------------------------
8  -- Revisoes  :
9  --     Data        Versao  Autor             Descricao
10 --     24/09/2021  1.0     Gabriel Kishida   versao inicial
11 ------------------------------------------------------------------
12
13 LIBRARY ieee;
14 USE ieee.std_logic_1164.ALL;
15 USE ieee.numeric_std.ALL;
16 USE IEEE.math_real.ALL;
17
18 ENTITY uart_8N2 IS
19     PORT (
20         clock : IN STD_LOGIC;
21         reset : IN STD_LOGIC;
22         partida : IN STD_LOGIC;
23         dados_ascii : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
24         recebe_dado : IN STD_LOGIC;
25         dado_serial : IN STD_LOGIC;
26         dado_recebido : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
27         pronto_rx : OUT STD_LOGIC;
28         tem_dado : OUT STD_LOGIC;
29         pronto_tx : OUT STD_LOGIC;
```

```
30          saida_serial : OUT STD_LOGIC;
31          db_dado_recebido_0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
32          db_dado_recebido_1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
33          db_tick_rx : OUT STD_LOGIC;
34          db_tick_tx : OUT STD_LOGIC;
35          db_estado_rx : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
36          db_estado_tx : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
37          db_deslocador_rx : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
38          db_deslocador_tx : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
39      );
40  END ENTITY;
41
42  ARCHITECTURE uart_8N2_arch OF uart_8N2 IS
43
44      COMPONENT tx_serial_8N2 IS
45          PORT (
46              clock, reset, partida : IN STD_LOGIC;
47              dados_ascii : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
48              saida_serial, pronto : OUT STD_LOGIC;
49              db_deslocado : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
50              db_deslocado_7seg_0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
51              db_deslocado_7seg_1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
52              db_deslocado_7seg_2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
53              db_tick : OUT STD_LOGIC;
54              db_estado : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
55          );
56      END COMPONENT;
57
58      COMPONENT rx_serial_8N2 IS
59          PORT (
60              clock : IN STD_LOGIC;
61              reset : IN STD_LOGIC;
62              dado_serial : IN STD_LOGIC;
63              recebe_dado : IN STD_LOGIC;
64              pronto_rx : OUT STD_LOGIC;
65              tem_dado : OUT STD_LOGIC;
66              dado_recebido : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
67              db_dado_recebido_0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
68              db_dado_recebido_1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
69              db_tick : OUT STD_LOGIC;
70              db_estado : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
71              db_dado_serial : OUT STD_LOGIC;
72              db_deslocador : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
73          );
74      END COMPONENT;
75
76  BEGIN
77      TX : tx_serial_8n2 PORT MAP(
78          clock => clock,
79          reset => reset,
80          partida => partida,
81          dados_ascii => dados_ascii,
82          saida_serial => saida_serial,
83          pronto => pronto_tx,
84          db_deslocado => db_deslocador_tx,
85          db_deslocado_7seg_0 => OPEN,
86          db_deslocado_7seg_1 => OPEN,
87          db_deslocado_7seg_2 => OPEN,
88          db_tick => db_tick_tx,
89          db_estado => db_estado_tx
90      );
91
92      RX : rx_serial_8n2 PORT MAP(
93          clock => clock,
94          reset => reset,
95          dado_serial => dado_serial,
```

```
96          recebe_dado => recebe_dado,
97          pronto_rx => pronto_rx,
98          tem_dado => tem_dado,
99          dado_recebido => dado_recebido,
100         db_dado_recebido_0 => db_dado_recebido_0,
101         db_dado_recebido_1 => db_dado_recebido_1,
102         db_tick => db_tick_rx,
103         db_estado => db_estado_rx,
104         db_dado_serial => OPEN,
105         db_deslocador => db_deslocador_rx
106     );
107
108 END ARCHITECTURE;
```

6.2.2 Testbench

rx_serial_tb

```
1  ----------------------------------------------------------------
2  -- Arquivo    : rx_serial_tb.vhd
3  -- Projeto    : Experiencia 3 - Recepcao Serial Assincrona
4  ----------------------------------------------------------------
5  -- Descricao : testbench para circuito de recepcao serial
6  --              contem recursos adicionais que devem ser aprendidos
7  --              1) procedure em VHDL (UART_WRITE_BYTE)
8  --              2) array de casos de teste
9  --
10 ----------------------------------------------------------------
11 -- Revisoes  :
12 --     Data        Versao  Autor             Descricao
13 --     09/09/2021  1.0     Edson Midorikawa  versao inicial
14 ----------------------------------------------------------------
15 --
16 library ieee;
17 use ieee.std_logic_1164.all;
18 use ieee.numeric_std.all;
19
20 entity rx_serial_tb is
21 end entity;
22
23 architecture tb of rx_serial_tb is
24
25   -- Declaracao de sinais para conectar o componente a ser testado (DUT)
26   signal clock_in:    std_logic  := '0';
27   signal reset_in:    std_logic  := '0';
28   signal recebe_in:   std_logic  := '0';
29   -- saidas
30   signal dado_out:    std_logic_vector(7 downto 0) := "00000000";
31   signal pronto_out:  std_logic  := '0';
32   signal temdado_out: std_logic  := '0';
33   signal db_tick_out: std_logic  := '0';
34   signal db_estado_out : std_logic_vector(6 downto 0) := "0000000";
35   signal db_dado_recebido_0  : std_logic_vector (6 downto 0) := "0000000";
36   signal db_dado_recebido_1  : std_logic_vector (6 downto 0) := "0000000";
37
38   -- para procedimento UART_WRITE_BYTE
39   signal entrada_serial_in: std_logic := '1';
40   signal serialData:        std_logic_vector(7 downto 0) := "00000000";
41
42   -- Configuraes do clock
43   constant clockPeriod : time := 20 ns;             -- 50MHz
44   constant bitPeriod :   time := 5208*clockPeriod; -- 5208 clocks por bit (9.600 bauds)
```

```
45  -- constant bitPeriod : time := 454*clockPeriod; -- 454 clocks por bit (115.200 bauds)
46
47  -- Procedimento para geracao da sequencia de comunicacao serial 8N2
48  -- adaptacao de codigo acessado de:
49  -- https://www.nandland.com/goboard/uart-go-board-project-part1.html
50  procedure UART_WRITE_BYTE (
51      Data_In : in  std_logic_vector(7 downto 0);
52      signal Serial_Out : out std_logic) is
53  begin
54
55      -- envia Start Bit
56      Serial_Out <= '0';
57      wait for bitPeriod;
58
59      -- envia Dado de 8 bits
60      for ii in 0 to 7 loop
61          Serial_Out <= Data_In(ii);
62          wait for bitPeriod;
63      end loop;  -- loop ii
64
65      -- envia 2 Stop Bits
66      Serial_Out <= '1';
67      wait for 2*bitPeriod;
68
69  end UART_WRITE_BYTE;
70  -- fim procedure
71
72  ---- Array de casos de teste
73  type caso_teste_type is record
74      id    : natural;
75      data : std_logic_vector(7 downto 0);
76  end record;
77
78  type casos_teste_array is array (natural range <>) of caso_teste_type;
79  constant casos_teste : casos_teste_array :=
80      (
81        (1, "00110101"), -- 35H
82        (2, "01010101"), -- 55H
83        (3, "10101010"), -- AAH
84        (4, "11111111"), -- FFH
85        (5, "00000000")  -- 00H
86        -- inserir aqui outros casos de teste (inserir "," na linha anterior)
87      );
88
89  signal keep_simulating: std_logic := '0'; -- delimita o tempo de  g e r a o  do clock
90
91 begin
92
93  ---- Gerador de Clock
94  clock_in <= (not clock_in) and keep_simulating after clockPeriod/2;
95
96  --  I n s t a n c i a o  direta DUT (Device Under Test)
97  DUT: entity work.rx_serial_8N2
98       port map (
99                 clock=>           clock_in,
100                reset=>           reset_in,
101                dado_serial=>     entrada_serial_in,
102                recebe_dado=>     recebe_in,
103                pronto_rx=>       pronto_out,
104                tem_dado=>        temdado_out,
105                dado_recebido=>   dado_out,
106           db_dado_recebido_0 => db_dado_recebido_0,
107           db_dado_recebido_1 => db_dado_recebido_1,
108                db_estado=>       db_estado_out,
109           db_tick=>       db_tick_out
110       );
```

```
111
112   ---- Geracao dos sinais de entrada (estimulo)
113   stimulus: process is
114   begin
115
116     ---- inicio da simulacao
117     assert false report "inicio da simulacao" severity note;
118     keep_simulating <= '1';
119     -- reset
120     reset_in <= '0';
121     wait for bitPeriod;
122     reset_in <= '1', '0' after 2*clockPeriod;
123     wait for bitPeriod;
124
125     ---- loop pelos casos de teste
126     for i in casos_teste'range loop
127         assert false report "Caso de teste " & integer'image(casos_teste(i).id) severity note;
128         serialData <= casos_teste(i).data; -- caso de teste "i"
129         wait for 10*clockPeriod;
130
131         -- 1) envia bits seriais para circuito de recepcao
132         UART_WRITE_BYTE ( Data_In=>serialData, Serial_Out=>entrada_serial_in );
133         entrada_serial_in <= '1'; -- repouso
134         wait for bitPeriod;
135
136         -- 2) simula recebimento do dado (p.ex. circuito principal registra saida)
137         wait until falling_edge(clock_in);
138         recebe_in <= '1', '0' after 2*clockPeriod;
139
140         -- 3) intervalo entre casos de teste
141         wait for 2*bitPeriod;
142     end loop;
143
144     ---- final dos casos de teste da simulacao
145     assert false report "fim da simulacao" severity note;
146     keep_simulating <= '0';
147
148     wait; -- fim da  s i m u l a o : aguarda indefinidamente
149
150   end process stimulus;
151
152 end tb;
```

uart_tb

```
1  ------------------------------------------------------------------
2  -- Arquivo    : tx_serial_tb.vhd
3  -- Projeto    : Experiencia 2 - Transmissao Serial Assincrona
4  ------------------------------------------------------------------
5  -- Descricao : circuito da experiencia 2
6  --              > modelo de testbench para simulacao do circuito
7  --              > de transmissao serial assincrona
8  --              >
9  ------------------------------------------------------------------
10 -- Revisoes  :
11 --     Data        Versao  Autor             Descricao
12 --     09/09/2021  1.0     Edson Midorikawa  versao inicial
13 ------------------------------------------------------------------
14 --
15 LIBRARY ieee;
16 USE ieee.std_logic_1164.ALL;
17 USE ieee.numeric_std.ALL;
18
19 ENTITY uart_tb IS
```

```
20 END ENTITY;
21
22 ARCHITECTURE tb OF uart_tb IS
23
24     -- Componente a ser testado (Device Under Test -- DUT)
25     COMPONENT uart_8N2 IS
26         PORT (
27             clock : IN STD_LOGIC;
28             reset : IN STD_LOGIC;
29             partida : IN STD_LOGIC;
30             dados_ascii : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
31             recebe_dado : IN STD_LOGIC;
32             dado_serial : IN STD_LOGIC;
33             dado_recebido : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
34             pronto_rx : OUT STD_LOGIC;
35             tem_dado : OUT STD_LOGIC;
36             pronto_tx : OUT STD_LOGIC;
37             saida_serial : OUT STD_LOGIC;
38             db_dado_recebido_0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
39             db_dado_recebido_1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
40             db_tick_rx : OUT STD_LOGIC;
41             db_tick_tx : OUT STD_LOGIC;
42             db_estado_rx : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
43             db_estado_tx : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
44             db_deslocador_rx : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
45             db_deslocador_tx : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
46         );
47     END COMPONENT;
48
49     -- Declara  o de sinais para conectar o componente a ser testado (DUT)
50     --   valores iniciais para fins de simulacao (ModelSim)
51     SIGNAL clock_in : STD_LOGIC := '0';
52     SIGNAL reset_in : STD_LOGIC := '0';
53     SIGNAL partida_in : STD_LOGIC := '0';
54     SIGNAL recebe_dado_in : STD_LOGIC := '0';
55     SIGNAL dados_ascii_8_in : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
56     SIGNAL dado_recebido_out : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
57     SIGNAL saida_serial_out : STD_LOGIC := '1';
58     SIGNAL pronto_rx_out : STD_LOGIC := '0';
59     SIGNAL pronto_tx_out : STD_LOGIC := '0';
60     SIGNAL tem_dado_out : STD_LOGIC := '0';
61     SIGNAL db_tick_rx_out, db_tick_tx_out : STD_LOGIC := '0';
62     SIGNAL db_estado_rx_out, db_estado_tx_out : STD_LOGIC_VECTOR (6 DOWNTO 0);
63
64     -- Configura  es do clock
65     SIGNAL keep_simulating : STD_LOGIC := '0'; -- delimita o tempo de gera  o do clock
66     CONSTANT clockPeriod : TIME := 20 ns; -- clock de 50MHz
67
68 BEGIN
69     -- Gerador de clock: executa enquanto 'keep_simulating = 1', com o per odo
70     -- especificado. Quando keep_simulating=0, clock    interrompido, bem como a
71     -- simula  o de eventos
72     clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
73
74     -- Conecta DUT (Device Under Test)
75     dut : uart_8N2
76     PORT MAP
77     (
78         clock => clock_in,
79         reset => reset_in,
80         partida => partida_in,
81         dados_ascii => dados_ascii_8_in,
82         recebe_dado => recebe_dado_in,
83         dado_serial => saida_serial_out,
84         dado_recebido => dado_recebido_out,
85         pronto_rx => pronto_rx_out,
```

```
86          tem_dado => tem_dado_out,
87          pronto_tx => pronto_tx_out,
88          saida_serial => saida_serial_out,
89          db_dado_recebido_0 => OPEN,
90          db_dado_recebido_1 => OPEN,
91          db_tick_rx => db_tick_rx_out,
92          db_tick_tx => db_tick_tx_out,
93          db_estado_rx => db_estado_rx_out,
94          db_estado_tx => db_estado_tx_out,
95          db_deslocador_rx => OPEN,
96          db_deslocador_tx => OPEN
97      );
98
99      -- geracao dos sinais de entrada (estimulos)
100     stimulus : PROCESS IS
101     BEGIN
102
103         ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
104         keep_simulating <= '1';
105
106         ---- inicio da simulacao: reset ----------------
107         partida_in <= '0';
108         reset_in <= '1';
109         WAIT FOR 20 * clockPeriod; -- pulso com 20 periodos de clock
110         reset_in <= '0';
111         WAIT UNTIL falling_edge(clock_in);
112         WAIT FOR 50 * clockPeriod;
113
114         ---- dado de entrada da simulacao (caso de teste #1)
115         dados_ascii_8_in <= "00110101"; -- x35 = '5'
116         WAIT FOR 20 * clockPeriod;
117
118         ---- acionamento da partida (inicio da transmissao)
119         partida_in <= '1';
120         WAIT UNTIL rising_edge(clock_in);
121         WAIT FOR 5 * clockPeriod; -- pulso partida com 5 periodos de clock
122         partida_in <= '0';
123
124         ---- espera final da transmissao (pulso pronto em 1)
125         WAIT UNTIL pronto_tx_out = '1';
126
127         ---- final do caso de teste 1
128
129         -- intervalo entre casos de teste
130         WAIT FOR 2000 * clockPeriod;
131
132         ---- dado de entrada da simulacao (caso de teste #2)
133         recebe_dado_in <= '1';
134         WAIT FOR 20 * clockPeriod;
135         recebe_dado_in <= '0';
136         dados_ascii_8_in <= "00101001"; -- x29 = ')'
137         WAIT FOR 100 * clockPeriod;
138
139         ---- acionamento da partida (inicio da transmissao)
140         partida_in <= '1';
141         WAIT UNTIL rising_edge(clock_in);
142         WAIT FOR 5 * clockPeriod; -- pulso partida com 5 periodos de clock
143         partida_in <= '0';
144
145         ---- espera final da transmissao (pulso pronto em 1)
146         WAIT UNTIL pronto_tx_out = '1';
147
148         ---- final do caso de teste 2
149
150         -- intervalo
151         WAIT FOR 2000 * clockPeriod;
```

```
152
153         ---- final dos casos de teste da simulacao
154         ASSERT false REPORT "Fim da simulacao" SEVERITY note;
155         keep_simulating <= '0';
156
157         WAIT; -- fim da simula  o: aguarda indefinidamente
158     END PROCESS;
159 END ARCHITECTURE;
```

7 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.