

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

PLANEJAMENTO - EXPERIÊNCIA 5

Prof. Edson Midorikawa

SÃO PAULO - SP

2021

SUMÁRIO

1	INTRODUÇÃO	3
1.1	Visão Geral: Sistema de Sonar	3
1.2	Funcionalidade	4
1.3	Não Funcionalidade	4
2	SOLUÇÃO TÉCNICA	5
2.1	Descrição Geral	5
3	ATIVIDADES	6
3.1	Atividade 1	6
3.2	Atividade 2	30
3.3	Atividade 3	33
4	RESULTADOS	39
5	DESAFIO	40
6	APÊNDICE	42
6.1	Testbenches	42
6.1.1	Comunicação Serial	43
6.1.2	Transmissão dos dados do sonar	50
7	REFERÊNCIAS BIBLIOGRÁFICAS	54

1 INTRODUÇÃO

1.1 Visão Geral: Sistema de Sonar

Nesse laboratório os alunos terão que desenvolver os componentes para a realização da varredura e detecção de objetos próximos com o uso de um sensor ultrassônico de distância e um servomotor.

Para a realização desse objetivo será projetado um circuito que ativa a movimentação do servomotor em modo de varredura em 8 posições, alternando entre elas a cada 1 segundo, e devolver uma saída serial com o ângulo e a distância do objeto até a aparelhagem, além de um alarme de proximidade no caso da distância ser menor o que 20 cm.

Foram utilizados na montagem componentes de laboratórios passados, sendo esses:

- Controle do servomotor
- Circuito de interface do sensor ultrassônico
- Circuitos de comunicação serial

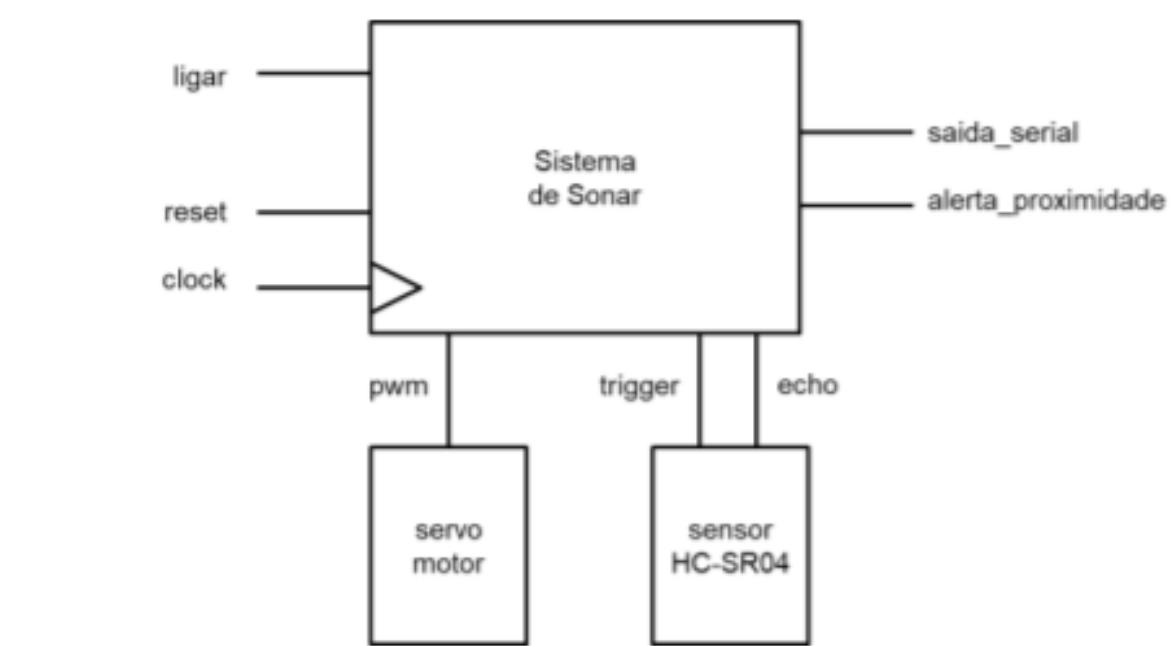


Figura 1: Diagrama do sistema do sonar

1.2 Funcionalidade

O circuito deve usar o sensor ultrassônico e o servomotor para retirar medidas de distâncias em 8 posições definidas e acionar um alarme de proximidade no caso de um anteparo a menos de 20 cm de distância.

1.3 Não Funcionalidade

O circuito não deve mapear num espectro contínuo de posições, nem criar algum tipo de visualização dos arredores com as medidas.

2 SOLUÇÃO TÉCNICA

2.1 Descrição Geral

Neste experimento da disciplina de Laboratório Digital II, será desenvolvido o projeto da interface de um sensor de distância por ondas ultrassônicas. As 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 *.vhdl*, 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 *.vhdl*, 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) Desenvolva a refatoração de código dos componentes desenvolvidos nas experiências anteriores. Documente no Planejamento os resultados desta refatoração, descrevendo as modificações que foram necessárias.

A) Revisão do circuito de controle do servomotor

Para a criação do módulo de movimentação do servomotor foi reaproveitado o código do desafio 2 do experimento 1, usando o período de 50000000 de clocks para aproximar-se ao valor de 1 segundo por posição.

Não foi necessário o uso do componente contador_g_updown_m, nem a separação entre um fluxo de dados e unidade de controle propriamente ditos, considerando a simplicidade do projeto e o formato de sua realização durante o laboratório. Porém foi necessário a adição de uma posição para sinal de *posição* igual a "000".

O controle do servo motor foi modificado durante o laboratório da experiência 1 para o desafio 2 de forma que o servomotor tivesse 8 posições com as seguintes larguras de pulso correspondentes:

Posição	Ciclos de Clock	Largura do Pulso
000	50000	1
001	57143	1.143
010	64300	1.286
011	71450	1.429
100	78550	1.571
101	85700	1.714
110	92850	1.857
111	100000	2

O diagrama de blocos do circuito em sua forma final fica da seguinte forma:

B) Revisão do circuito de interface com o sensor ultrassônico de distância

O circuito de interface funciona de acordo com o esperado, dado suas testagens e aplicações nos laboratórios anteriores. Os sinais de depuração *db_medir* e *db_reset* foram adicionados por sugestão do professor.

C) Revisão dos circuitos de comunicação serial

Os circuitos de comunicação serial assíncrona – isto é, o receptor *rx_serial_8n2.vhd* e o transmissor *tx_serial_8n2.vhd* – foram revisados com o propósito de serem utilizados como componentes internos do circuito do sonar a ser desenvolvido.

Para tanto, os elementos com interface para dispositivos externos – como adaptações para displays de sete segmentos, e detectores de borda de pulso – foram removidos.

Assim, as suas especificações de entidades VHDL, seguidas de seus diagramas RTL ficaram:

tx_serial_8n2.vhd

```

1 ENTITY tx_serial_8N2 IS
2   PORT (
3     clock : IN STD.LOGIC;
4     reset : IN STD.LOGIC;
5     partida : IN STD.LOGIC;
6     dados_ascii : IN STD.LOGIC_VECTOR(7 DOWNTO 0);
7     saida_serial : OUT STD.LOGIC;
8     pronto_tx : OUT STD.LOGIC;
9     db_partida : OUT STD.LOGIC;
10    db_saida_serial : OUT STD.LOGIC;
11    db_estado : OUT STD.LOGIC_VECTOR(3 DOWNTO 0)
12  );
13 END ENTITY;

```

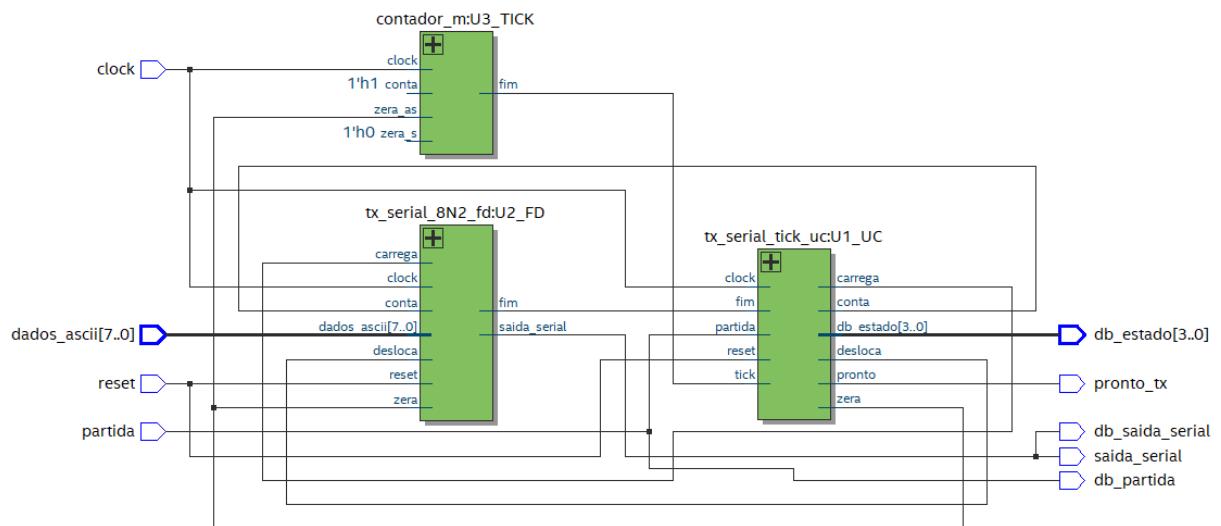


Figura 2: Diagrama RTL do componente transmissor de comunicação serial

rx_serial_8n2.vhd

```

1 ENTITY rx_serial_8N2 IS
2   PORT (
3     clock : IN STD.LOGIC;
4     reset : IN STD.LOGIC;
5     dado_serial : IN STD.LOGIC;
6     recebe_dado : IN STD.LOGIC;

```

```

7      dado_recebido : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
8      tem_dado : OUT STD_LOGIC;
9      pronto_rx : OUT STD_LOGIC;
10     db_recebe_dado : OUT STD_LOGIC;
11     db_dado_serial : OUT STD_LOGIC;
12     db_estado : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
13   );
14 END ENTITY;

```

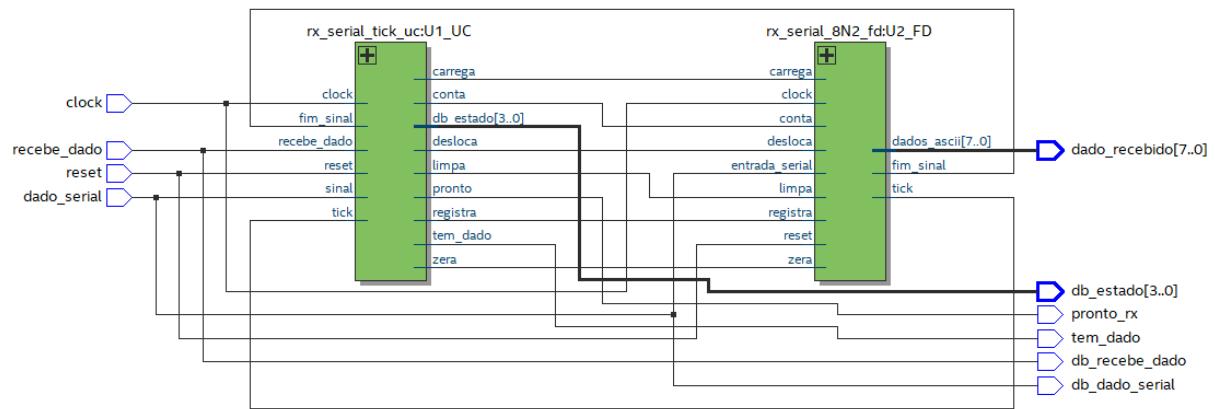


Figura 3: Diagrama RTL do componente receptor de comunicação serial

b) Documente a simulação dos componentes refatorados usando o ModelSim. Anexe as formas de onda obtidas, junto com uma breve descrição dos sinais das cartas de tempo. DICA: anote as figuras com observações dos resultados obtidos.

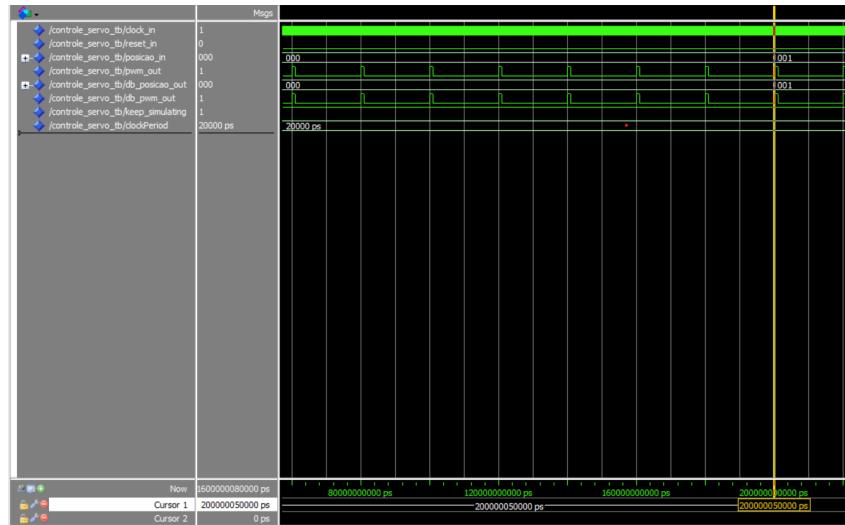
A) Testagem do módulo servomotor

Foram definidos durante sua síntese 3 sinais de depuração db_reset, db_pwm e db_posicao.

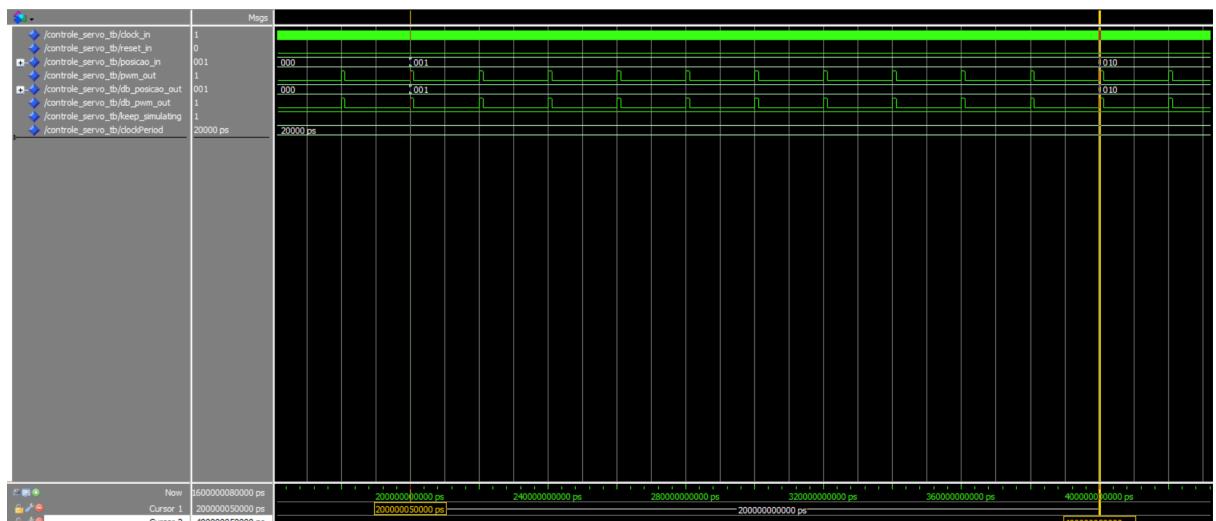
Foram desenvolvidos dois testbenches sendo um do controle do servomotor para checar seu funcionamento nas 8 posições e o outro para o controle do servomotor para que haja um movimento de varredura oscilando entre as 8 posições com o período de 1 segundo entre a mudança entre elas. Para a realização dos testes em tempo hábil foi utilizado o tempo de 250 períodos de clock entre cada transição entre as posições ao invés dos 50000000 necessários para ter 1 segundo de espera.

O teste do controle básico servomotor gerou a seguinte forma de onda:

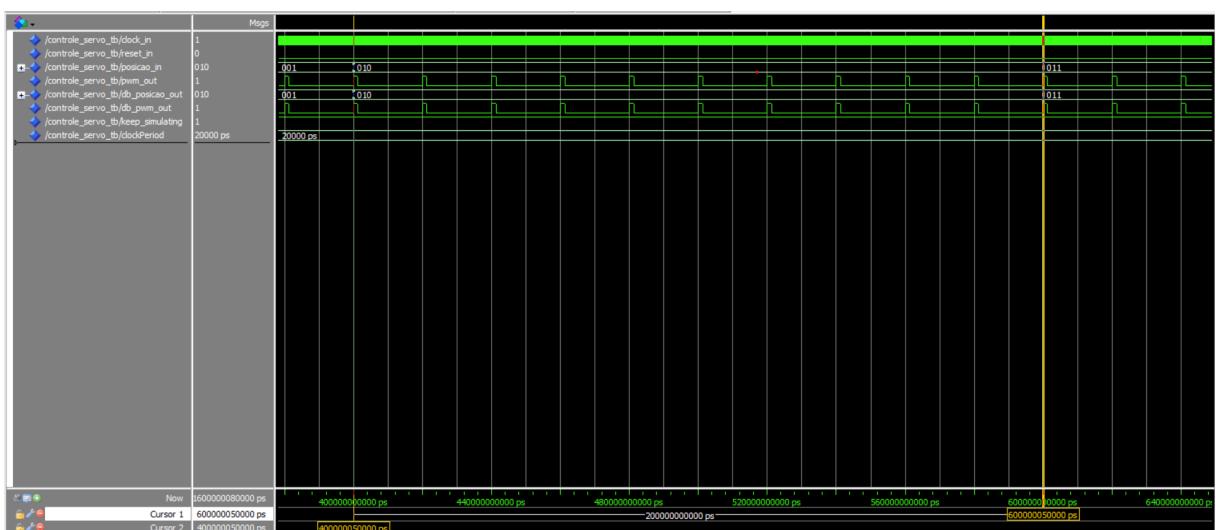
- Posição 000



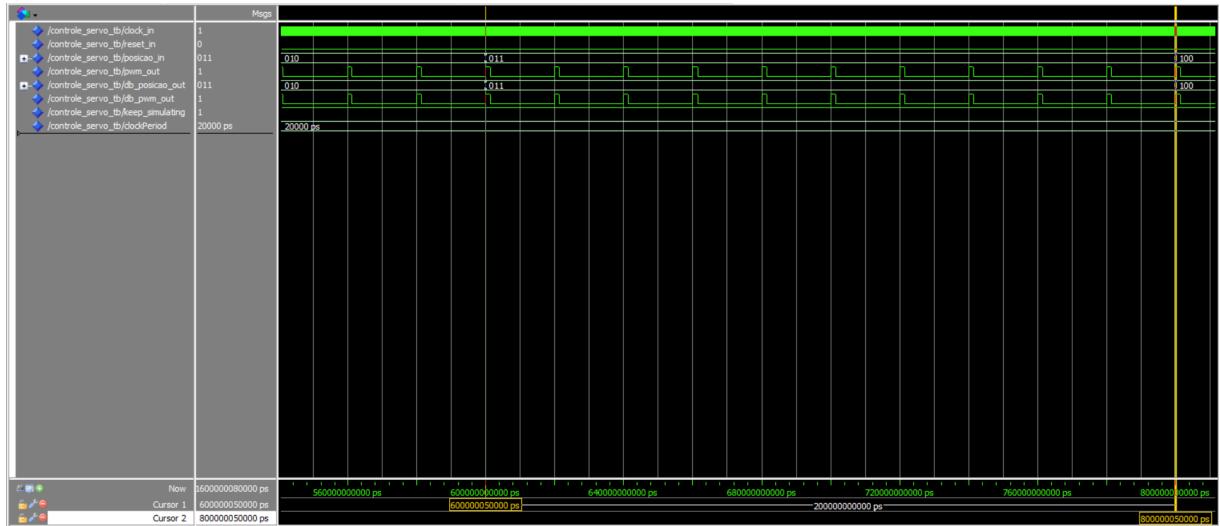
- Posição 001



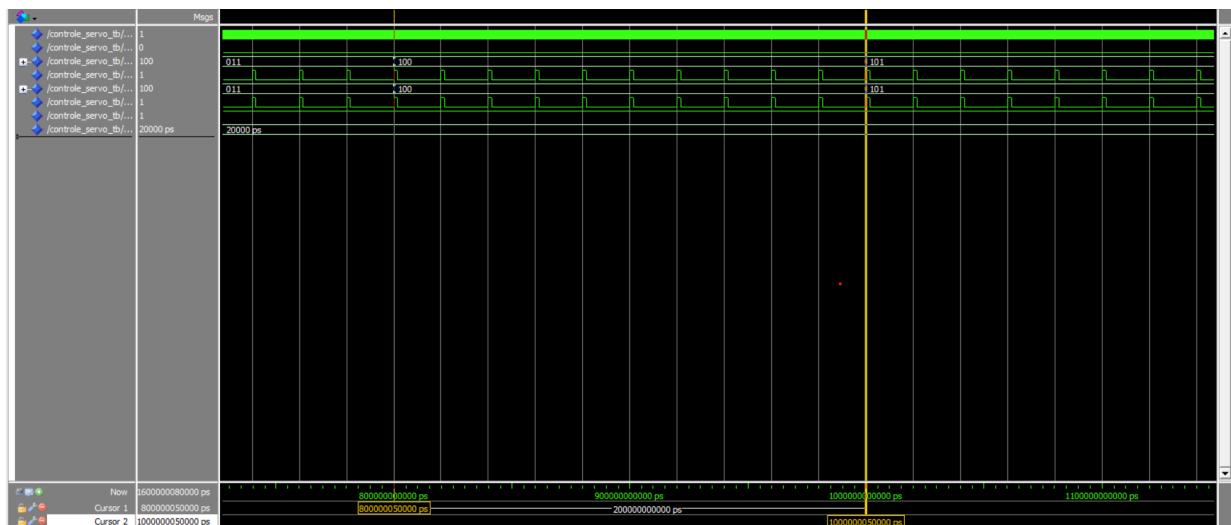
- Posição 010



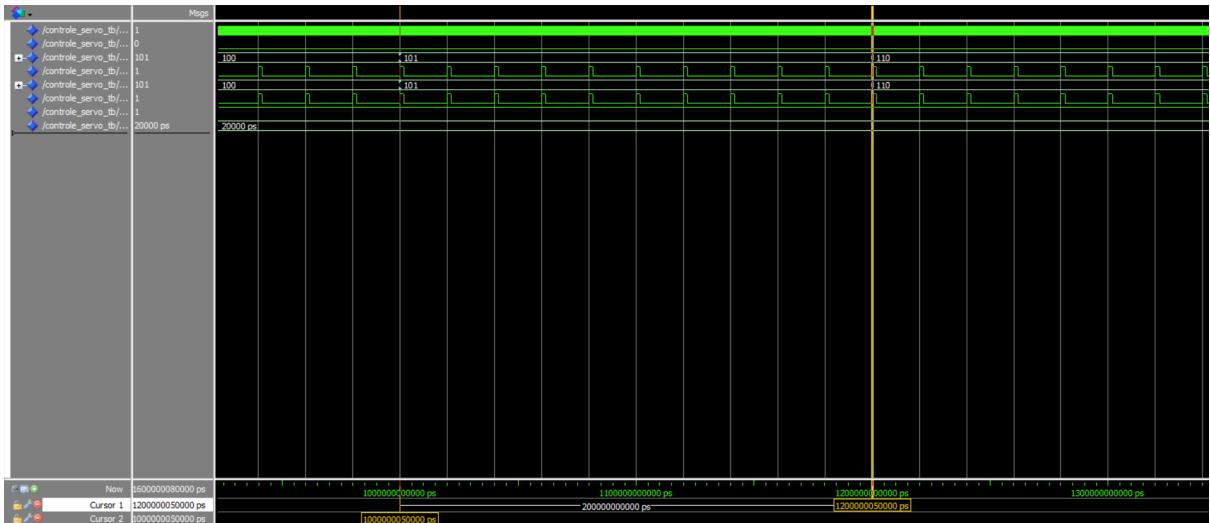
- Posição 011



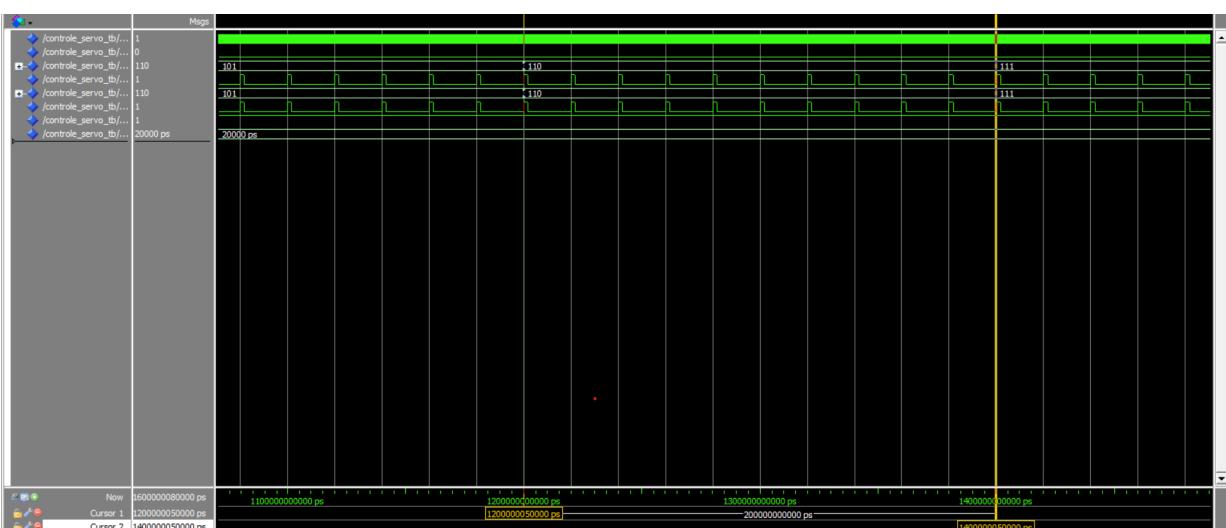
- Posição 100



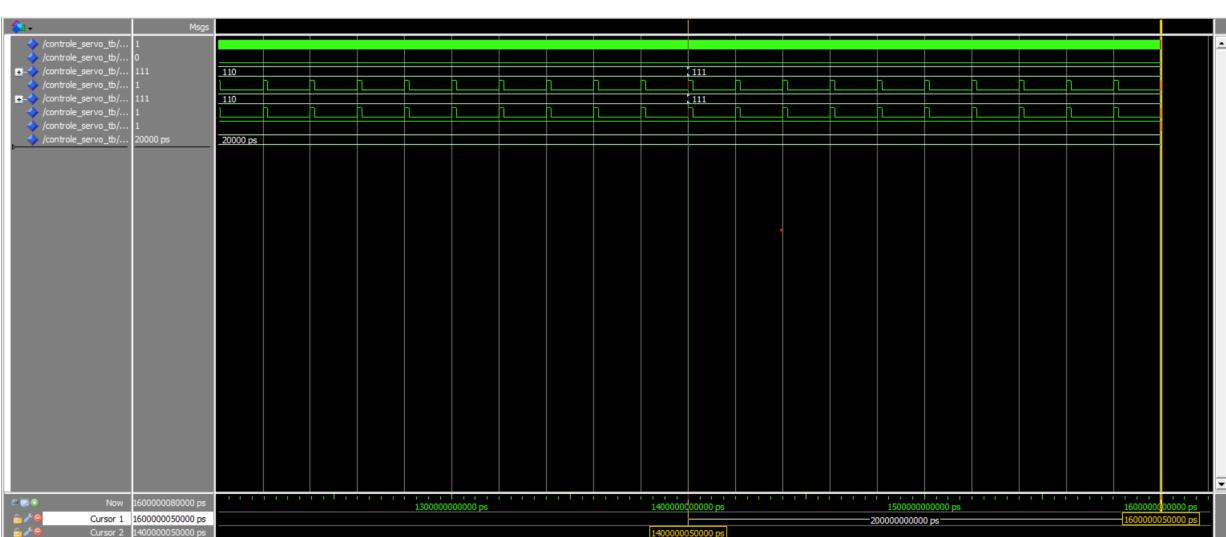
- Posição 101



- Posição 110



- Posição 111



Para o teste da movimentação do servomotor em varredura com 8 posições foram considerados os seguintes casos:

- Servo atingindo a posição 100
- Teste do signal *reset*
- Teste de varredura completa

Testagem da Interface Geral

Para testar a interface geral, vamos ter que testar os seguintes aspectos:

- Verificar se a unidade de controle troca de estados apropriadamente;
- Verificar se o tamanho dos pulsos gerados é adequado;
- Verificar se o aumento do tamanho do pulso *echo* gera uma medida maior.

1 - Teste para 4 cm

- Acionar *reset* e desativar, para resetar o circuito;
- Acionar *medir* para iniciar a medição do circuito;
- Esperar fim do envio dos pulsos de **trigger**;
- Enviar um pulso de *echo* com duração de 12000 clocks;
- Avaliar o sinal *medida*, verificar se equivale a *"0000 0000 0100"*

A simulação para este caso de teste no software ModelSim, resultou nas seguintes formas de ondas:

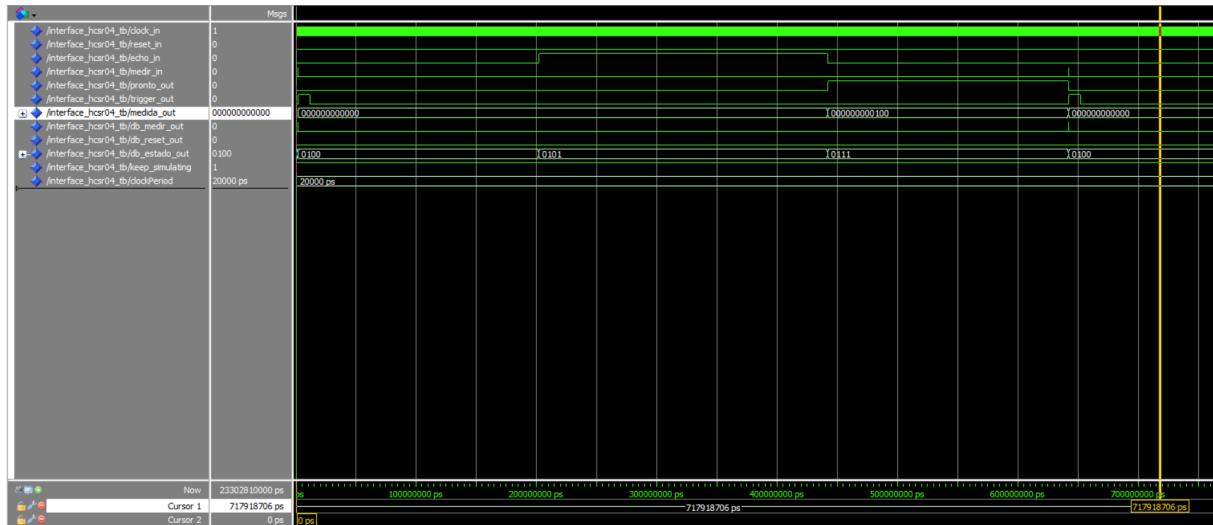


Figura 4: Forma de onda para o primeiro caso de teste

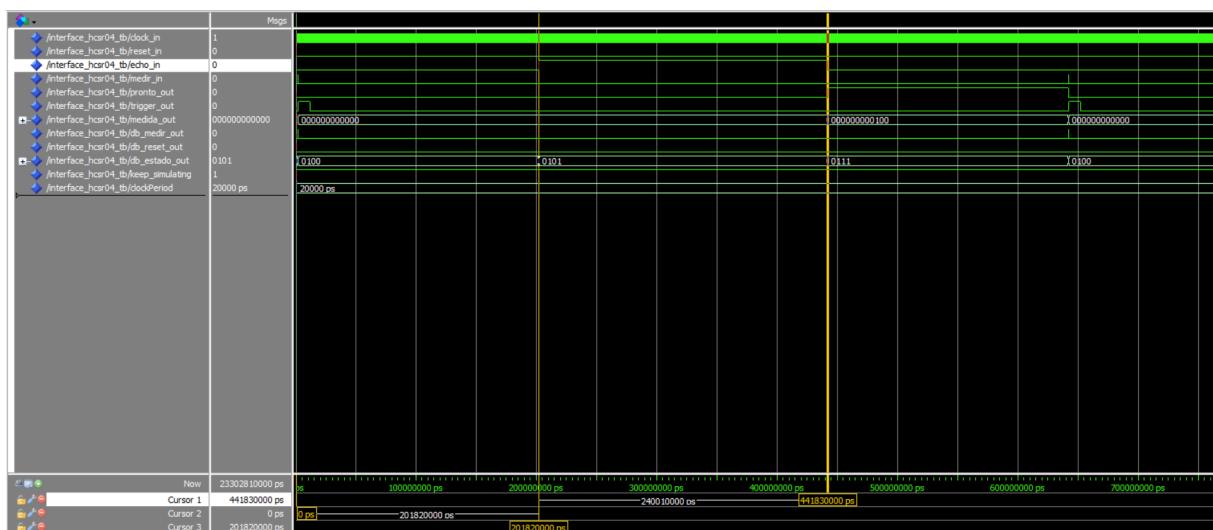


Figura 5: Largura do pulso "echo" para o primeiro teste

Observando o sinal de entrada *echo_in*, temos um pulso de **240 microssegundos**. Utilizando a fórmula para obter a distância a partir da largura do pulso, a distância medida é de:

$$D = \frac{240}{58,82} = 4,08\text{cm} \quad (1)$$

Como pode se ver, na forma de ondas o sinal *medida_out* possui o valor **"004"**, indicando 4 centímetros de distância, o que apresenta um caso de sucesso.

2 - Teste para 50 cm

- Acionar *reset* e desativar, para resetar o circuito;
- Acionar *medir* para iniciar a medição do circuito;
- Esperar fim do envio dos pulsos de **trigger**;
- Enviar um pulso de *echo* com duração de 148000 clocks;
- Avaliar o sinal *medida*, verificar se equivale a "0000 0101 0000"

A simulação para este caso de teste no software ModelSim, resultou nas seguintes formas de ondas:

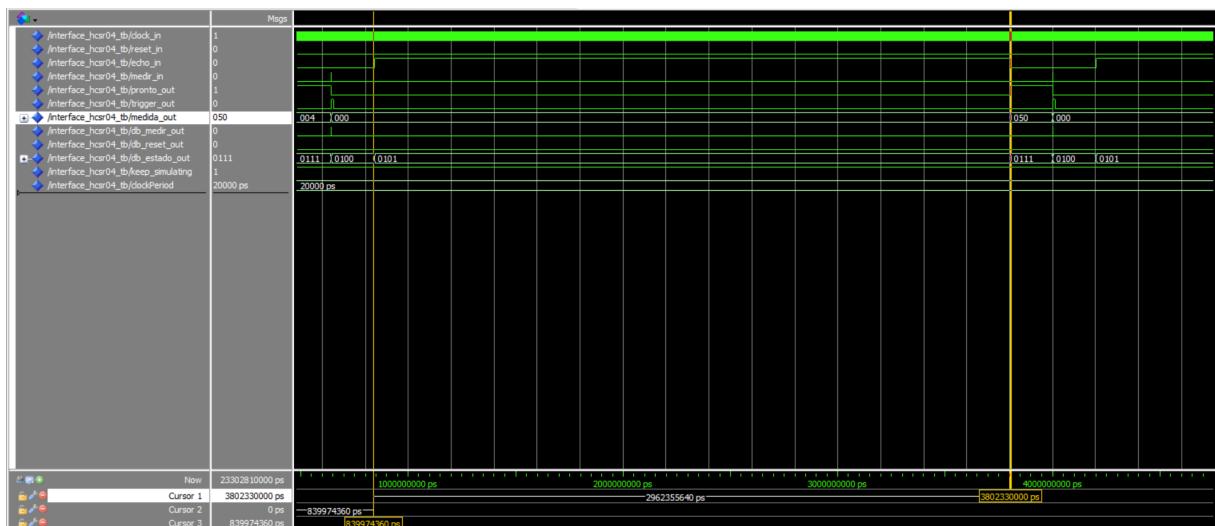


Figura 6: Forma de onda para o segundo caso de teste

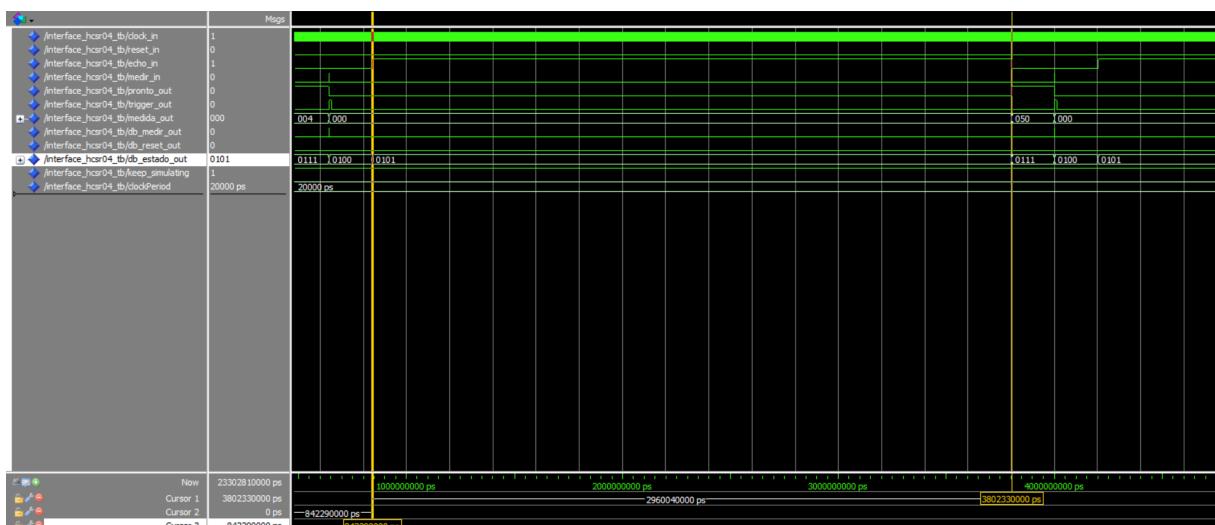


Figura 7: Largura do pulso "echo" para o segundo teste

Observando o sinal de entrada *echo_in*, temos um pulso de **2964 microssegundos**. Utilizando a fórmula para obter a distância a partir da largura do pulso, a distância medida é de:

$$D = \frac{2964}{58,82} = 50,391\text{cm} \quad (2)$$

Como pode se ver, na forma de ondas o sinal *medida_out* possui o valor **"050"**, indicando 50 centímetros de distância, o que apresenta um caso de sucesso.

3 - Teste para 321 cm

- Acionar *reset* e desativar, para resetar o circuito;
- Acionar *medir* para iniciar a medição do circuito;
- Esperar fim do envio dos pulsos de **trigger**;
- Enviar um pulso de *echo* com duração de 945000 clocks;
- Avaliar o sinal *medida*, verificar se equivale a **"0011 0010 0001"**

A simulação para este caso de teste no software ModelSim, resultou nas seguintes formas de ondas:

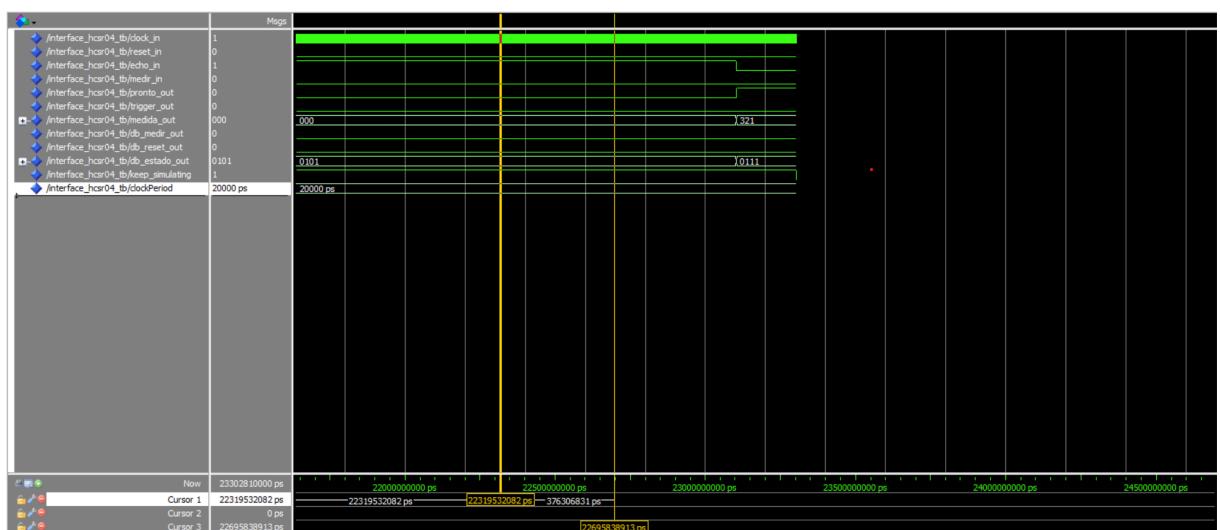


Figura 8: Forma de onda para o terceiro caso de teste

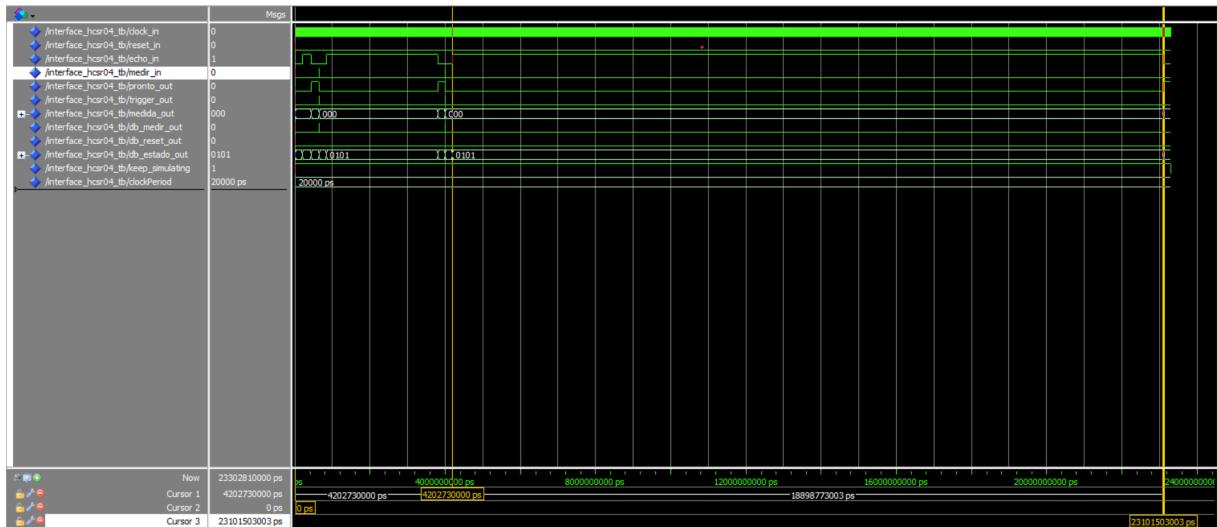


Figura 9: Largura do pulso "echo" para o terceiro teste

Observando o sinal de entrada *echo_in*, temos um pulso de **18894 microssegundos**. Utilizando a fórmula para obter a distância a partir da largura do pulso, a distância medida é de:

$$D = \frac{18894}{58,82} = 321,21cm \quad (3)$$

Como pode se ver, na forma de ondas o sinal *medida_out* possui o valor **"321"**, indicando 321 centímetros de distância, o que apresenta um caso de sucesso.

Além disso, analisando a largura do pulso de **trigger**:

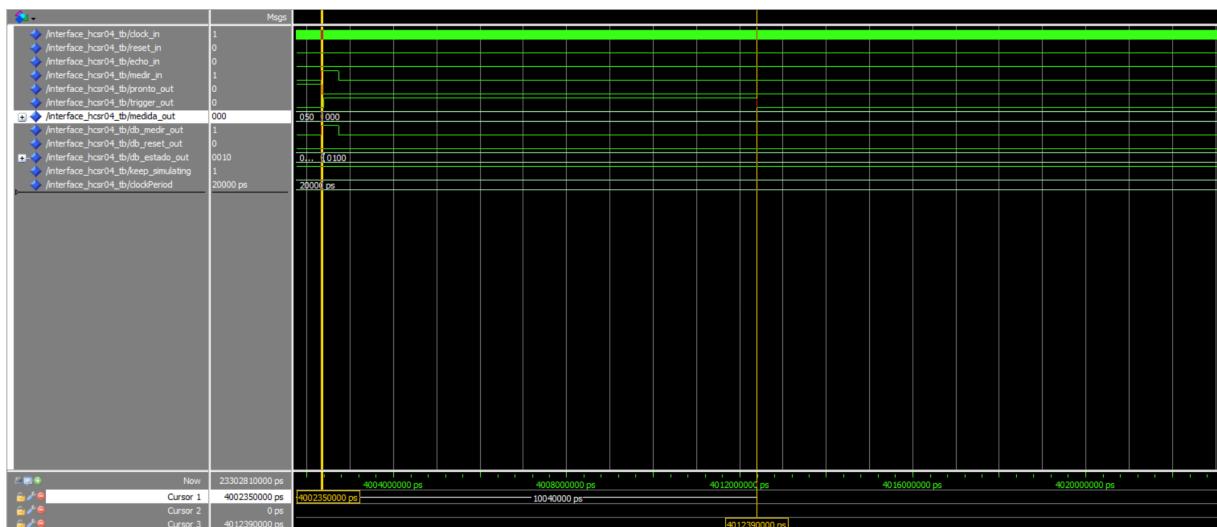


Figura 10: Forma de onda do pulso de trigger

De acordo com a medida realizada no MultiSim, sua largura é de **10 microsssegundos**, o que equivale o esperado

B) Testagem do circuito de interface com o sensor ultrassônico de distância

Para a testagem do circuito de interface foram considerados os seguintes casos:

C) Testagem dos circuitos de comunicação serial

Como não se alterou o funcionamento dos circuitos de comunicação serial, mantiveram-se as dinâmicas dos testes, apenas refatorando e ajustando os testbenches onde era devido.

O código desenvolvido para os testes será disponibilizado no apêndice.

Testagem do transmissor serial

Para o transmissor serial, a estratégia de testagem foi a mesma desenvolvida para o experimento 2, onde o teste foi dividido em dois: a testagem da transmissão de um sinal par e de um sinal ímpar.

Primeiramente, para a paridade par, realizou-se a transmissão do sinal **"00110101"**.

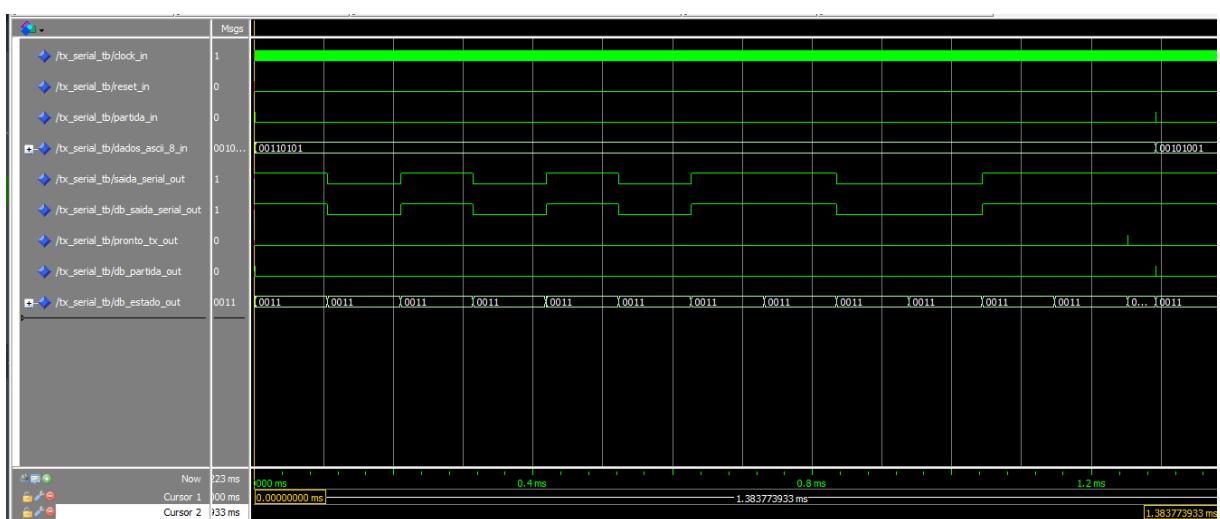


Figura 11: Testagem para paridade par

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)

O que caracterizou um teste bem sucedido.

Já para a paridade ímpar, realizou-se a transmissão do sinal: **"00101001"**.

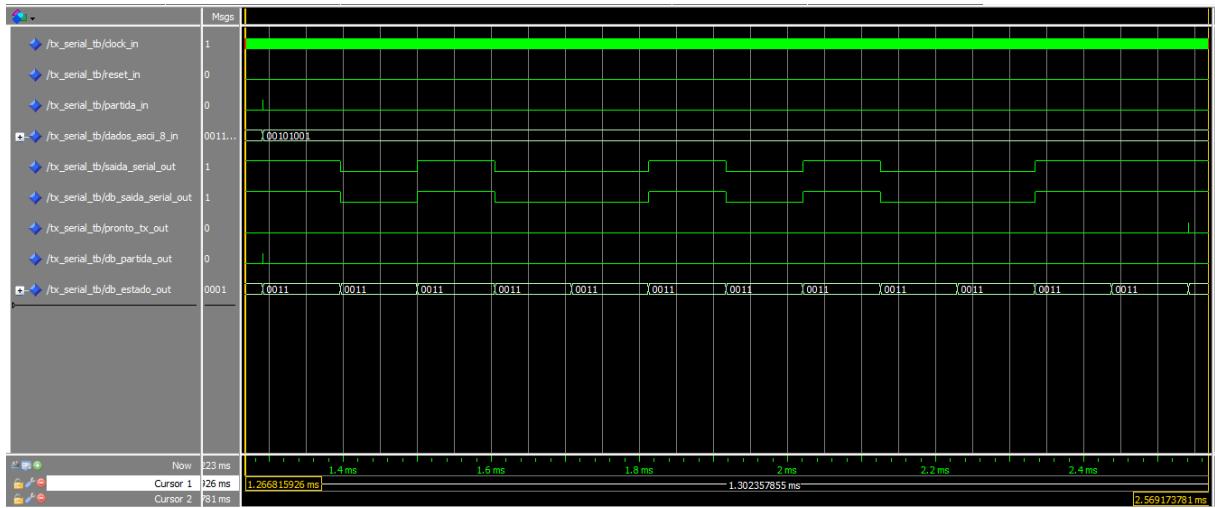


Figura 12: Testagem para paridade ímpar

Mais uma vez, nota-se o funcionamento correto, apresentando os seguintes sinais:

- 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)

Por fim, também caracteriza-se que o período apresentando por cada sinal é condizente, já que apresenta uma duração de $\frac{1}{9600}s = 0,10416ms$, como apresenta a imagem:

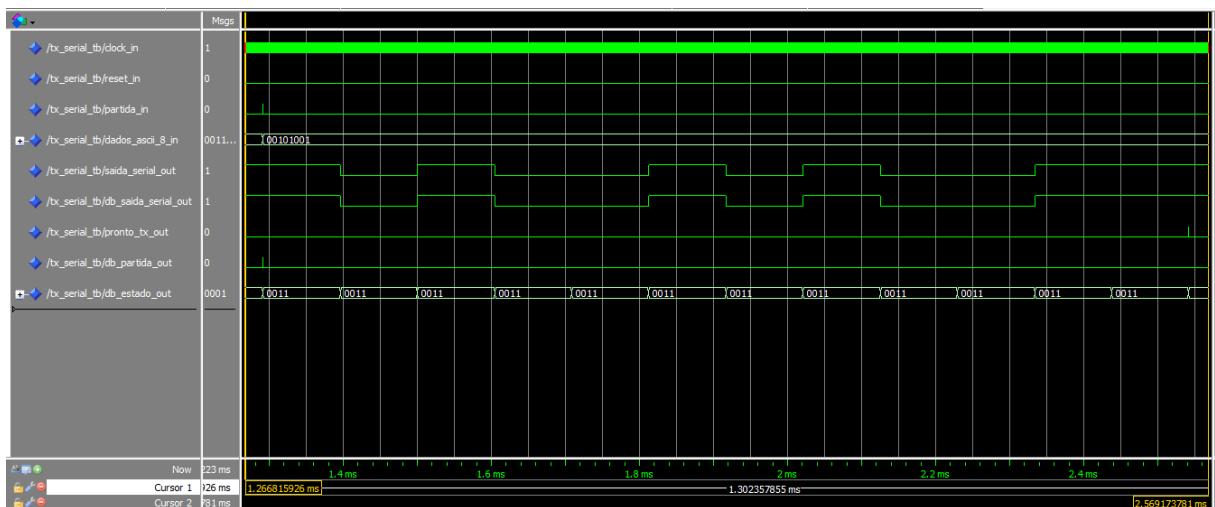


Figura 13: Tamanho do pulso da transmissão serial

Testagem do receptor serial

Para o receptor serial realizaram-se os mesmos testes que foram desenvolvidos no experimento 3, que consistem na transmissão desses sinais:

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

- 00110101

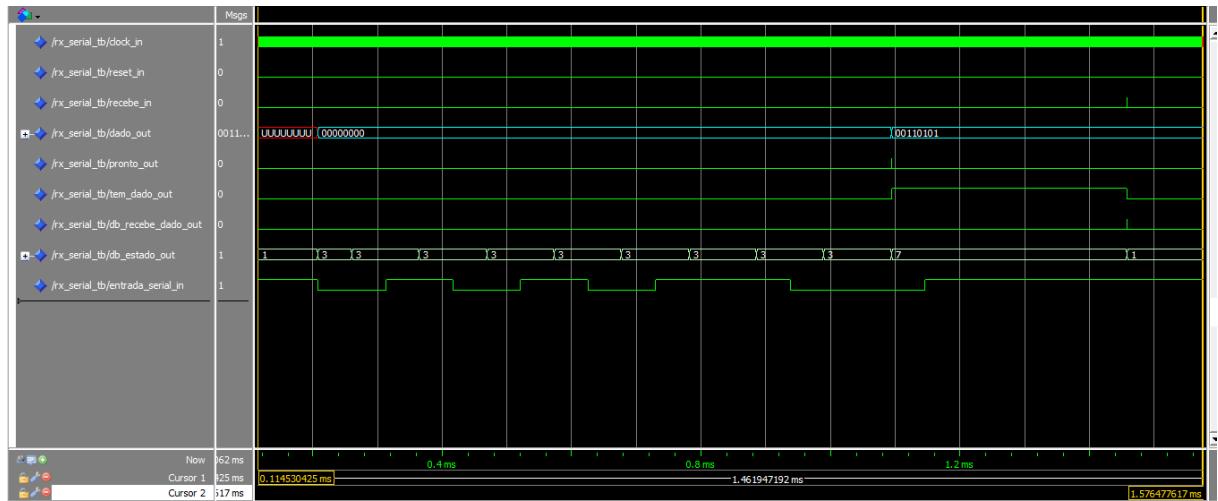


Figura 14: Primeiro caso de teste do receptor serial

- 01010101

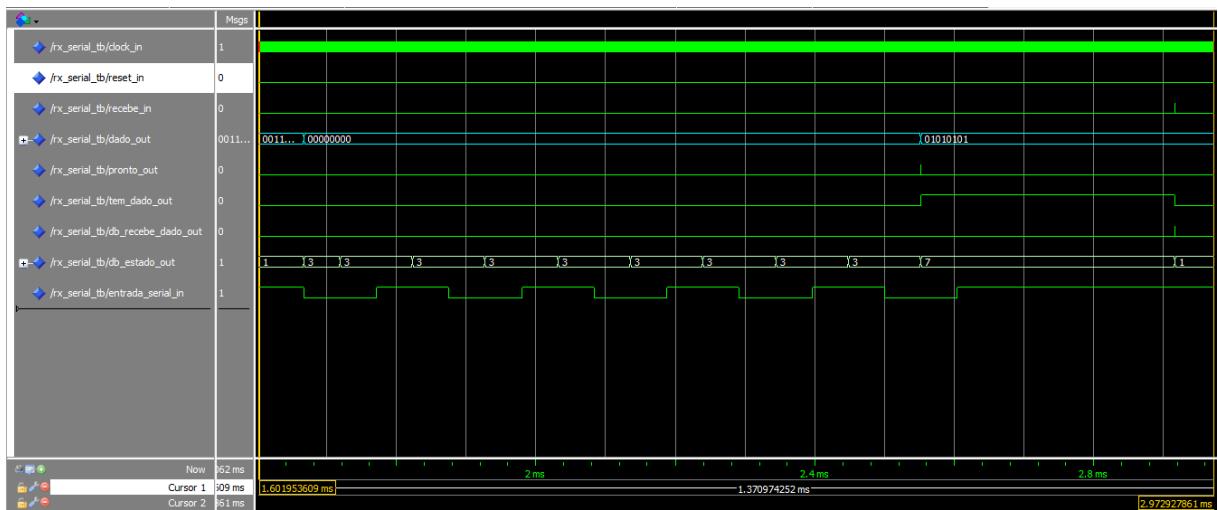


Figura 15: Segundo caso de teste do receptor serial

- 10101010

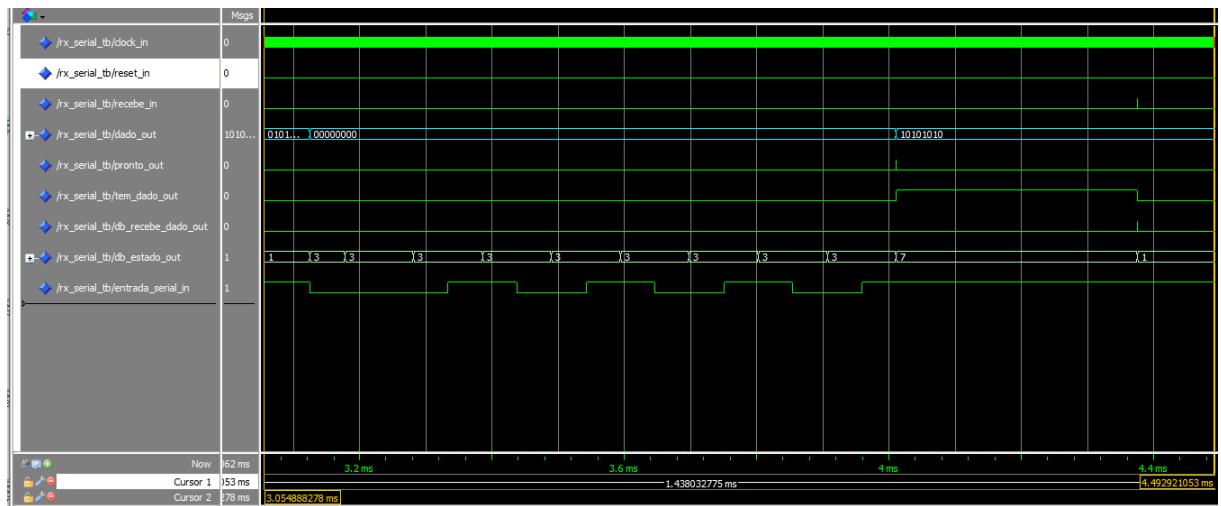


Figura 16: Terceiro caso de teste do receptor serial

- 11111111

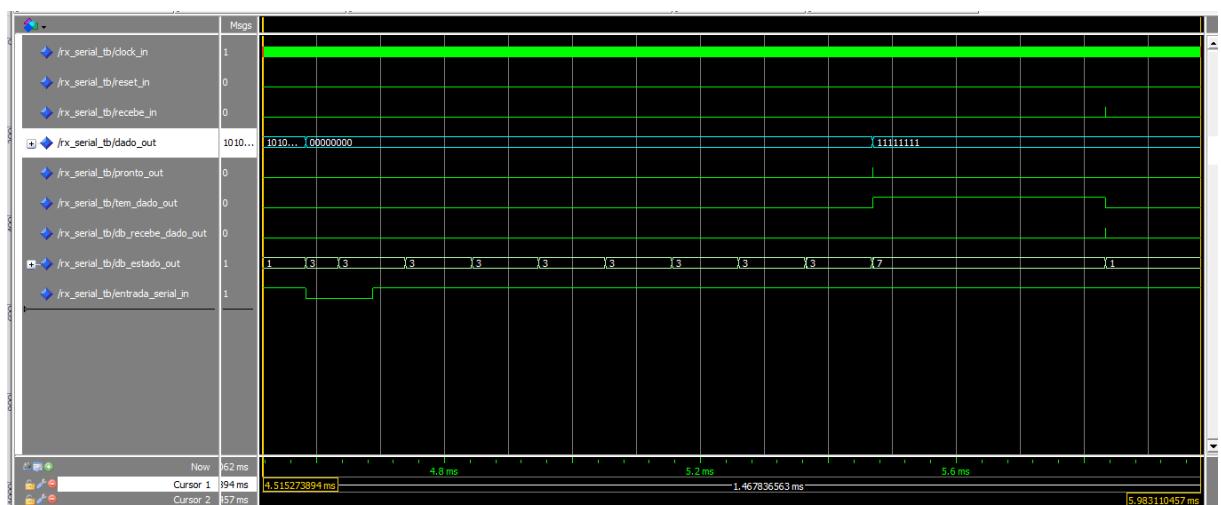


Figura 17: Quarto caso de teste do receptor serial

- 00000000

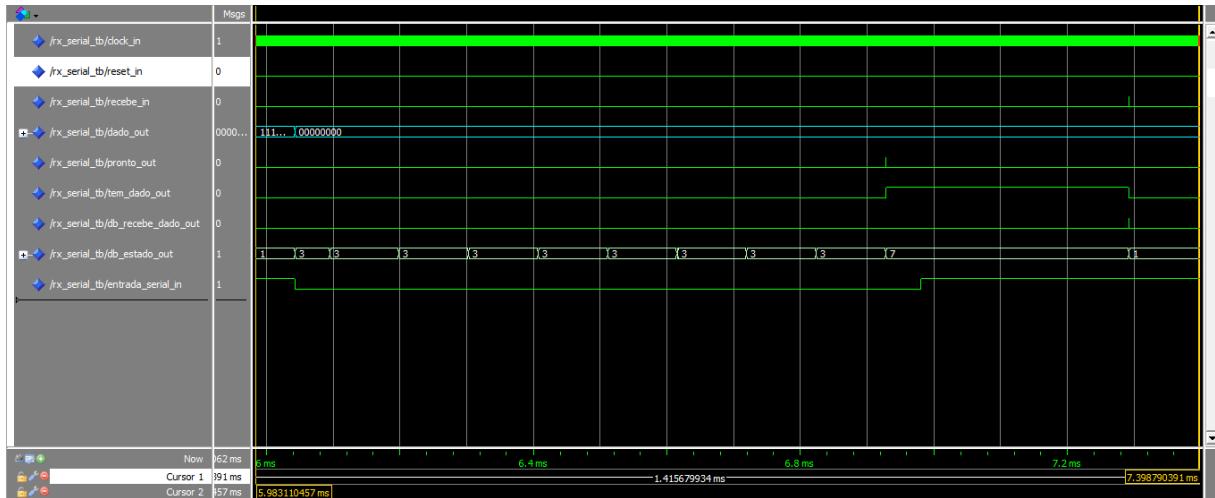


Figura 18: Quinto caso de teste do receptor serial

Como pode se visualizar, todos os dados enviados foram recebidos (visível na saída dado_out em ciano) o que constitui um caso de sucesso para a testagem deste componente.

c) Projete os novos componentes descritos na seção 1.4.2. Documente o funcionamento destes componentes.

D) Desenvolvimento do módulo UART

A especificação para o módulo UART fornecida pela apostila foi a seguinte:

```

1 ENTITY uart_8N2 IS
2   PORT (
3     clock : IN STD_LOGIC;
4     reset : IN STD_LOGIC;
5     transmite_dado : IN STD_LOGIC;
6     dados_ascii : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7     dado_serial : IN STD_LOGIC;
8     recebe_dado : IN STD_LOGIC;
9     saida_serial : OUT STD_LOGIC;
10    pronto_tx : OUT STD_LOGIC;
11    dado_recebido_rx : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
12    tem_dado : OUT STD_LOGIC;
13    pronto_rx : OUT STD_LOGIC;
14    db_transmite_dado : OUT STD_LOGIC;
15    db_saida_serial : OUT STD_LOGIC;
16    db_estado_tx : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
17    db_partida : OUT STD_LOGIC;
18    db_recebe_dado : OUT STD_LOGIC;
19    db_dado_serial : OUT STD_LOGIC;
20    db_estado_rx : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
21  );
22 END ENTITY;

```

Para a criação do módulo de UART, foi reaproveitado o código do desafio do experimento 3, no qual foi desenvolvida uma UART. No entanto, as entradas e saídas foram modificadas para atender as especificações fornecidas pela apostila – além das modificações indiretas de seus componentes (o transmissor e o receptor serial).

Assim, o diagrama de RTL da UART ficou da seguinte forma:

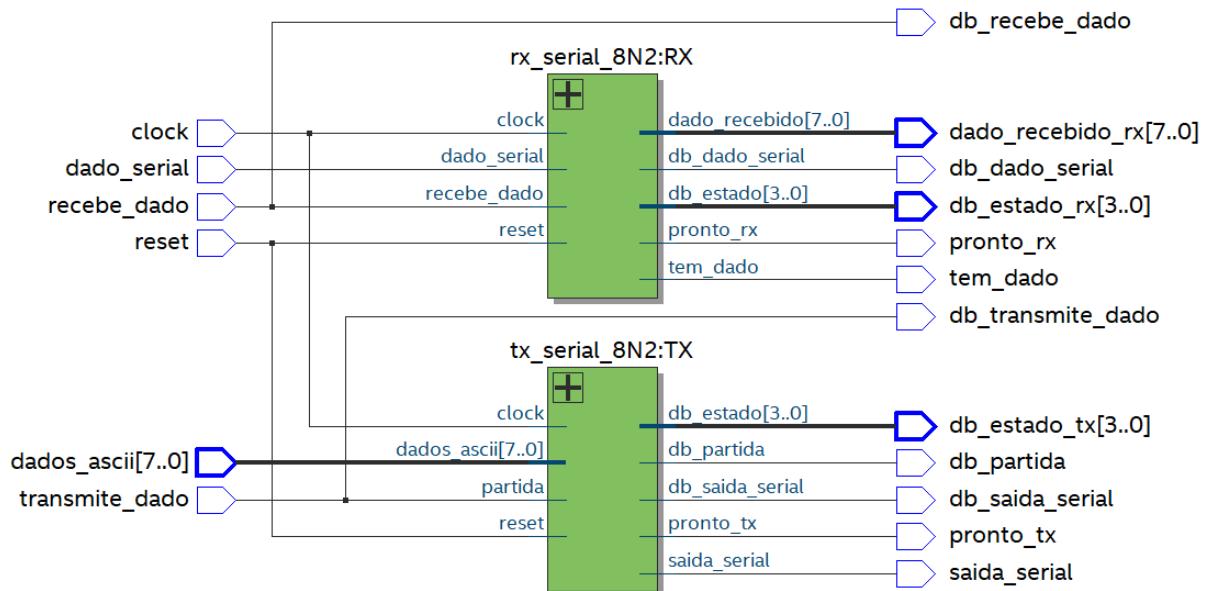


Figura 19: Diagrama RTL do módulo UART

E) Transmissão de dados seriais do sonar

Este módulo será responsável por receber os dígitos BCD relativos à distância e a angulação do sonar, e enviá-los em formato ASCII por meio de comunicação serial. A mensagem padrão conterá 8 caracteres: 3 dígitos para a angulação do sonar, 1 vírgula, 3 dígitos para a distância medida, e por último 1 ponto final.

Para fazer o envio dessa forma, será desenvolvida uma máquina de estados finita, permitindo assim a transmissão de cada caractere individualmente. Utilizará-se o módulo UART para transmitir as mensagens recebidas de forma serial.

Assim, a especificação da entidade VHD da unidade **tx_dados_sonar.vhd** ficou como:

```

1 ENTITY tx_dados_sonar IS
2     PORT (
3         clock : IN STD.LOGIC;
4         reset : IN STD.LOGIC;
5         transmitir : IN STD.LOGIC;
6         angulo2 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
7         angulo1 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
8         angulo0 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
9         distancia2 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
10        distancia1 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
11        distancia0 : IN STD.LOGIC.VECTOR(3 DOWNTO 0);
12        saída_serial : OUT STD.LOGIC;
13        pronto : OUT STD.LOGIC;
14        db_transmitir : OUT STD.LOGIC;
15        db_transmite_dado : OUT STD.LOGIC;
16        db_saida_serial : OUT STD.LOGIC;
17        db_estado_tx : OUT STD.LOGIC.VECTOR(3 DOWNTO 0);
18        db_estado_0 : OUT STD.LOGIC.VECTOR (2 DOWNTO 0)
19    );
20 END ENTITY;

```

A máquina de estado pode ser representada por meio deste diagrama:

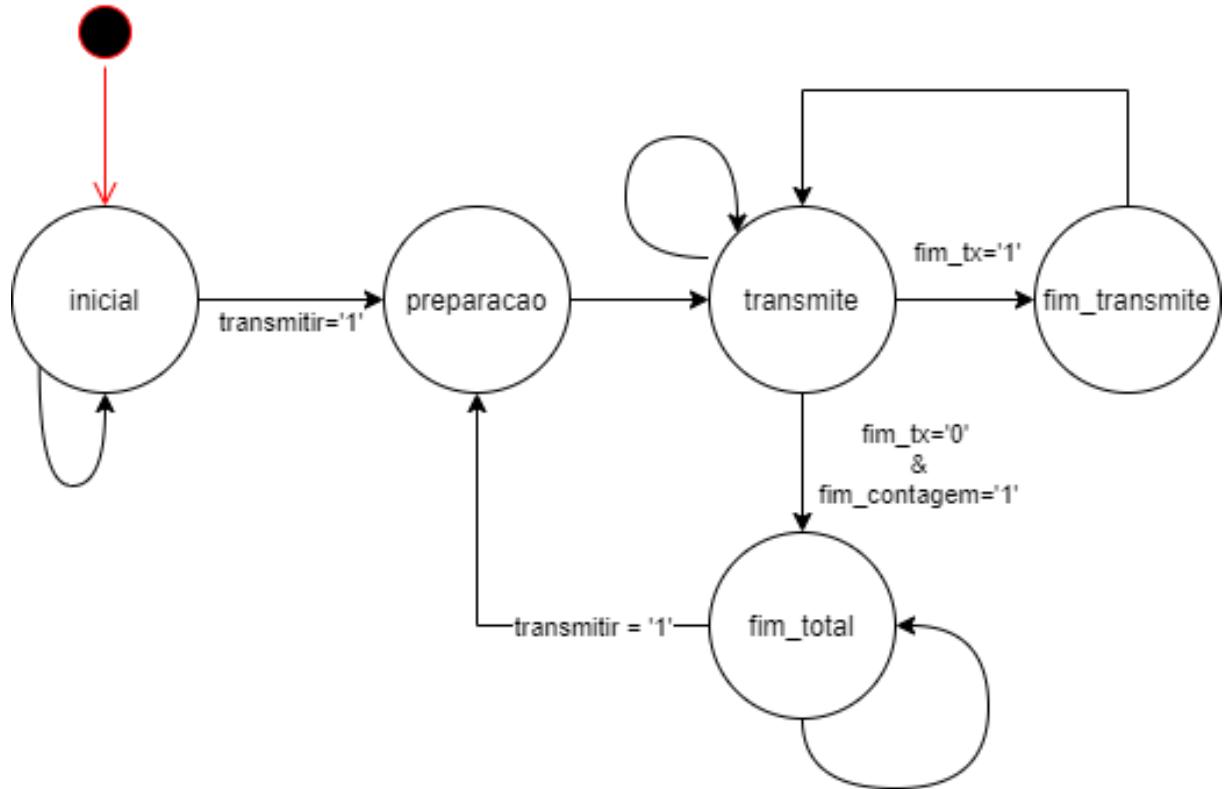


Figura 20: Máquina de estados do módulo de transmissão do sonar

A descrição de cada estado é:

- **inicial:** Estado inicial, que aguarda o comando para iniciar uma transmissão de sinal serial;
- **preparacao:** Estado preparatório, que zera contadores e faz a configuração para preparar o circuito com o objetivo de enviar um novo sinal;
- **transmite:** Estado de envio de sinal – a máquina fica neste estado durante a transmissão de um caractere;
- **fim_transmite:** Estado de fim de transmissão de um único caractere – responsável por iniciar a transmissão do próximo caractere;
- **fim_total:** Estado final da máquina de estados, atingido quando a palavra inteira (de 8 caracteres) foi transmitida. Neste estado, a máquina de estados ao receber um próximo sinal “*transmitir*”= ‘1’ inicia a próxima transmissão dos dados.

Para o funcionamento apropriado desta unidade de controle, é necessário também a existência de um fluxo de dados bem estruturado. Nele, será necessária a existência de:

- **UART**: para enviar sinais seriais;
- **Contador**: para contar quantas letras já foram enviadas por meio de comunicação serial, e selecionar a próxima letra a ser enviada;
- **Multiplexador de 8 entradas** : para selecionar quais das 8 palavras deve ser enviada pela UART.

O diagrama RTL do fluxo de dados desenvolvido ficou desta forma:

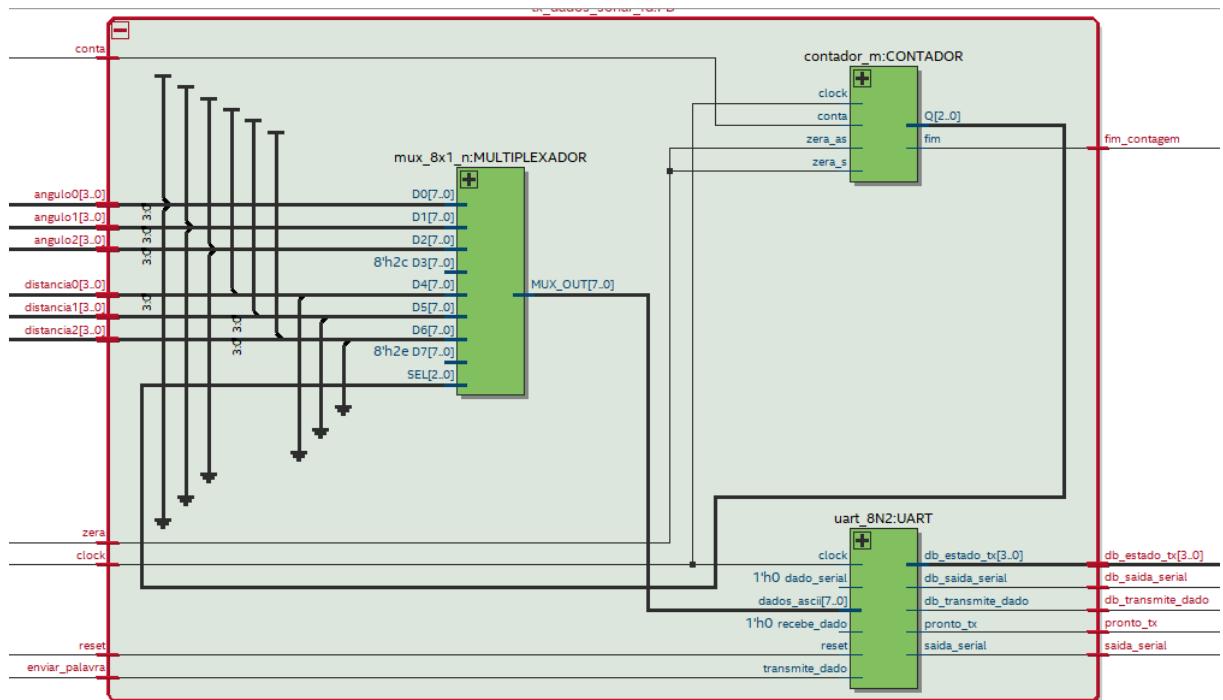


Figura 21: Diagrama RTL do fluxo de dados do circuito de transmissão dos dados do sonar

Por fim, o diagrama RTL do circuito **tx_dados_sonar.vhd** ficou como :

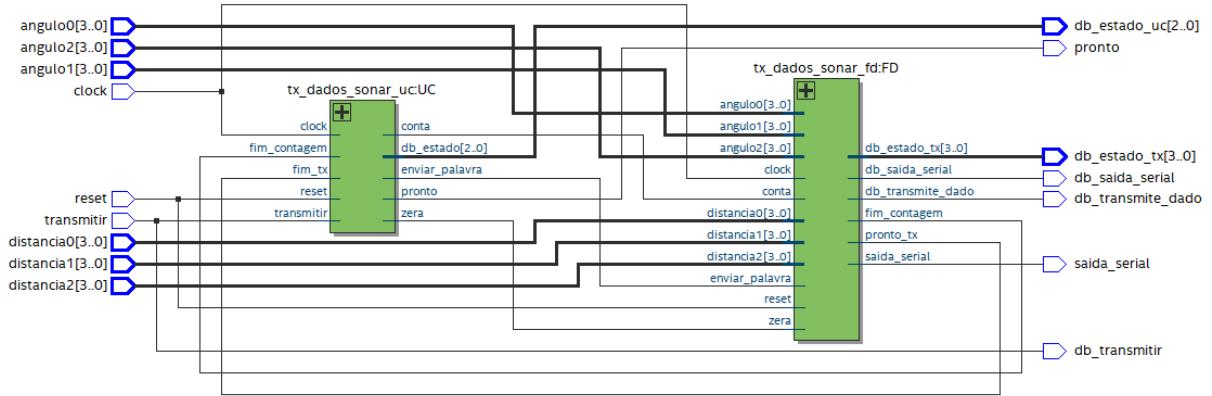


Figura 22: Diagrama RTL do circuito de transmissão dos dados do sonar

d) Defina casos de teste destes novos componentes.

D) Testagem do módulo UART

Para a testagem do módulo UART, aplicou-se o mesmo método utilizado no desafio do experimento 3: 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", 35 em hexadecimal (o caractere '5') e "00101001", 29 em hexadecimal (o caractere ')').

As ondas obtidas foram as seguintes:

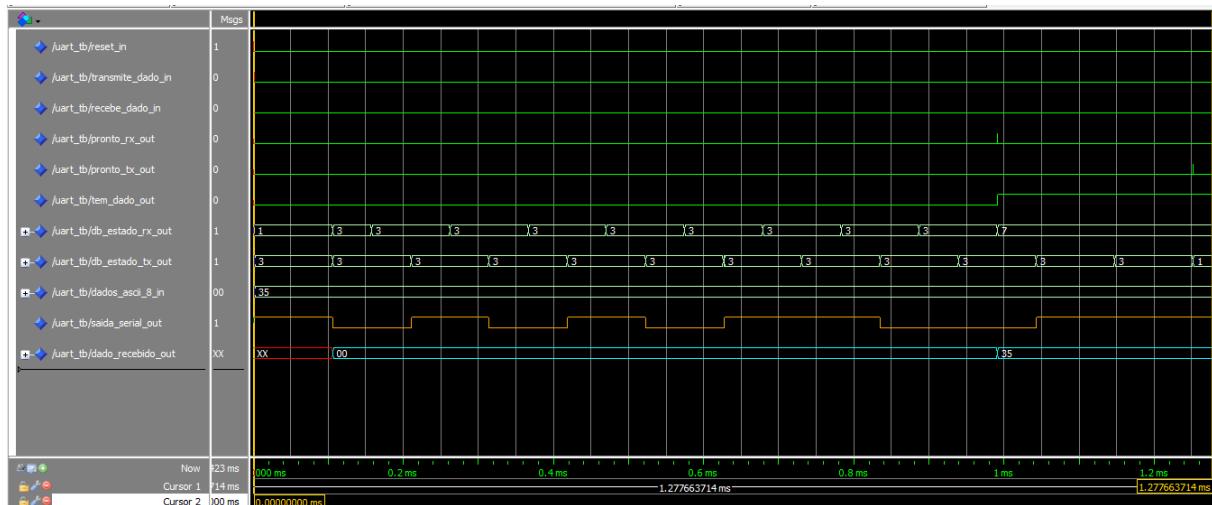


Figura 23: Testagem do sinal "00110101"para a UART

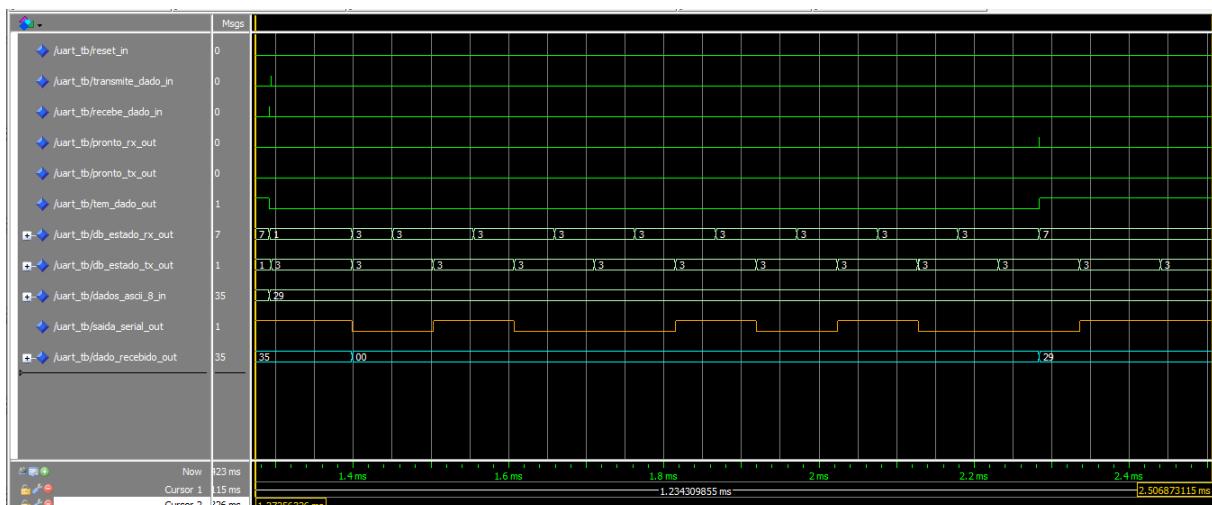


Figura 24: Testagem do sinal "00101001"para a UART

Como, nos dois casos, o sinal enviado pelo módulo interno transmissor da UART foi recebido apropriadamente pelo módulo receptor, pode-se concluir que ambos estão funcionando adequadamente, e a UART funciona como um todo.

E) Testagem do módulo de transmissão de dados do sonar

Para verificar-se o funcionamento da transmissão dos dados lidos pelo sonar, prepararam-se dois casos de teste:

- O envio da informação **"153,017."** (isto é, 153 medidas de angulação e 17 medidas de distância em cm) ;
- O envio da informação **"089,205."** (isto é, 89 medidas de angulação e 205 medidas de distância em cm) ;

Para realizar este teste no software de testagem de ondas ModelSim, utilizou-se um componente UART no testbench, cuja responsabilidade era receber os sinais seriais enviados e traduzi-los para uma forma mais legível – comprovando assim que o sinal serial está sendo enviado na forma correta.

Envio do sinal de "153,017."

Primeiramente, uma visão geral do primeiro caso de teste:

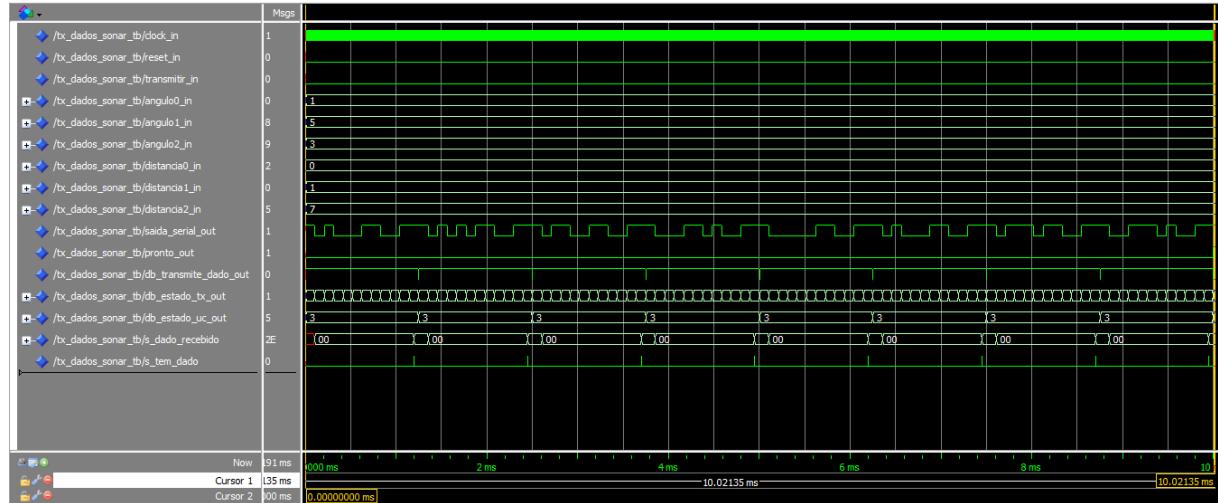


Figura 25: Visão geral do primeiro caso de teste

Em segundo lugar, uma *close* dos sinais, apresentando as saídas lidas em **s_dado_recebido**:

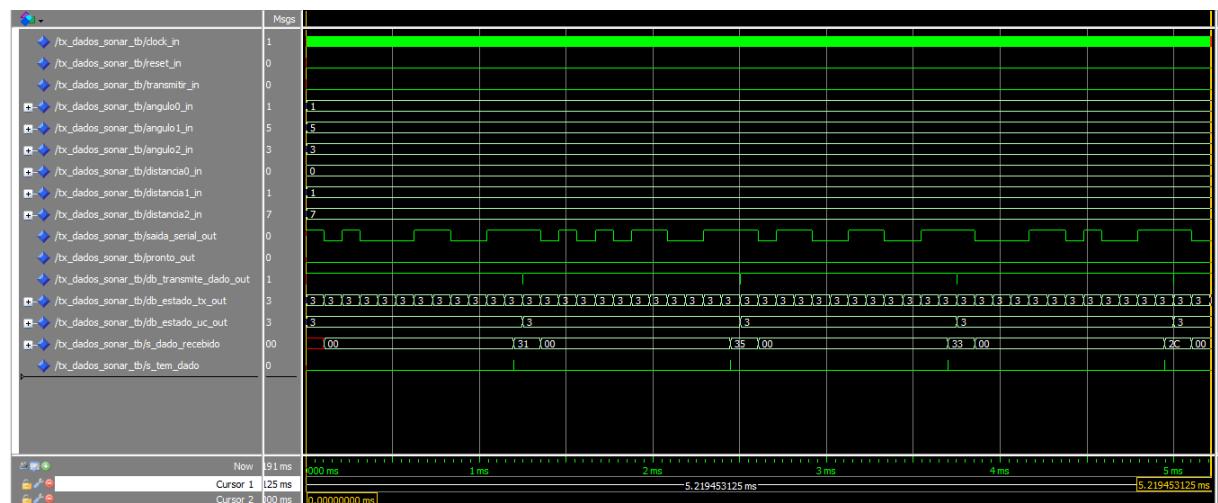


Figura 26: Visão dos 4 primeiros caracteres enviados no primeiro caso de teste

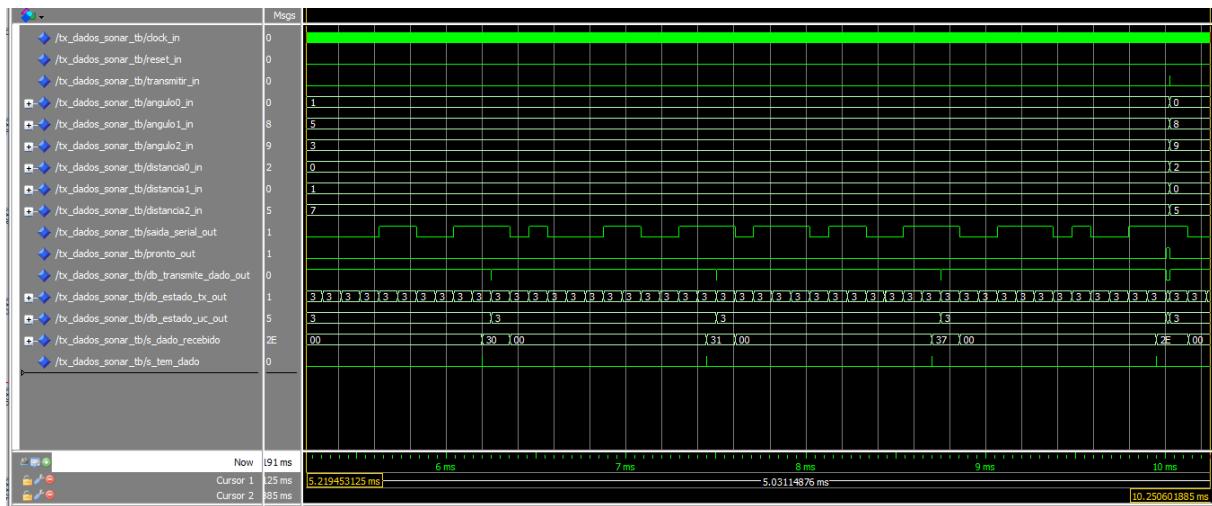


Figura 27: Visão dos 4 últimos caracteres enviados no primeiro caso de teste

Como pode-se ver, os sinais recebidos são equivalentes a **31, 35, 33, 2C, 30, 31, 37, 2E**, que em ASCII representam **"153,017."**, o que era o esperado. Assim, o primeiro caso de teste é considerado um sucesso.

Envio do sinal de **"089,205."**

Primeiramente, uma visão geral do segundo caso de teste:

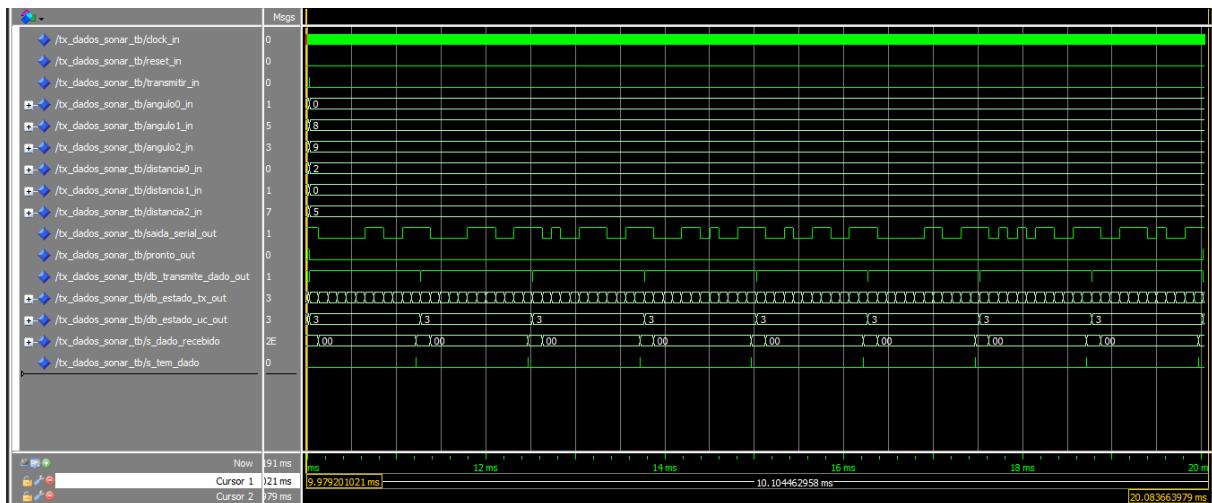


Figura 28: Visão geral do segundo caso de teste

Em segundo lugar, uma *close* dos sinais, apresentando as saídas lidas em **s_dado_recebido**:

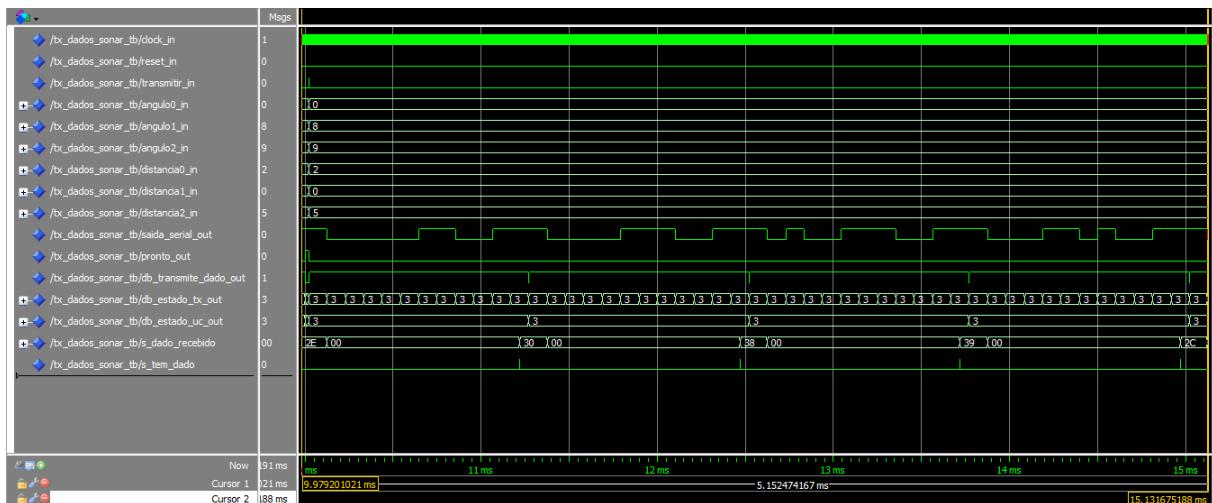


Figura 29: Visão dos 4 primeiros caracteres enviados no segundo caso de teste

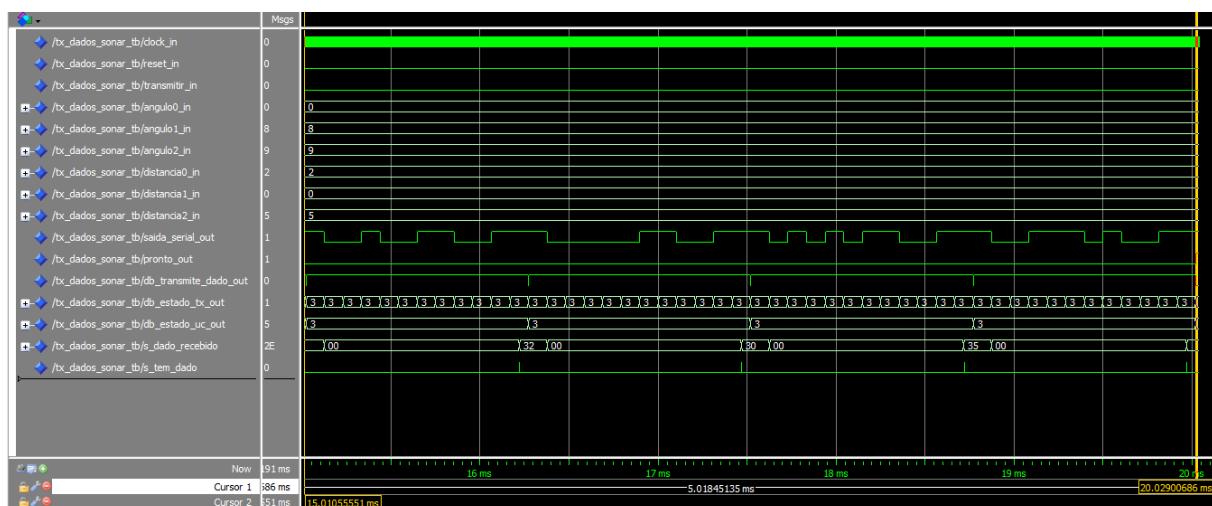


Figura 30: Visão dos 4 últimos caracteres enviados no segundo caso de teste

Como pode-se ver, os sinais recebidos são equivalentes a **30, 38, 39, 2C, 32, 30, 35, 2E**, que em ASCII representam **"089,205."**, o que era o esperado. Assim, o segundo caso de teste é também considerado um sucesso.

e) Execute a simulação dos casos de teste definidos para cada componente usando o ModelSim. Anote as figuras das ondas obtidas para mostrar o correto funcionamento

Este item já foi realizado no item anterior d)

f) Submeter os arquivos VHDL dos componentes refatorados, junto com os respectivos testbenches (arquivos ZIP)

3.2 Atividade 2

g) Projetar um circuito de teste que execute a movimentação contínua do servomotor percorrendo as 8 posições definidas. A movimentação deve seguir um padrão de movimento do “vai e volta”, com intervalo de 1 segundo em cada posição. DICA: o componente *contador_g_updown_m* será fornecido.

O circuito de teste desta movimentação do servomotor em varredura foi desenvolvido de maneira bem simples, já que o circuito por si só já funciona automaticamente, alterando as posições do servomotor. Para a análise do teste, utilizou-se o sinal **db_posicao**.

Desenvolveram-se dois casos de teste: o primeiro sendo uma varredura limitada, aguardando o tempo necessário para que o circuito fique na posição **”110”**, e verificando se o sinal **db_posicao** está correto. O segundo consiste em uma varredura geral, verificando se o circuito atinge todas as posições definidas de maneira correta.

h) Projetar um circuito de teste que envia a sequência de 8 caracteres ASCII com dados de ângulo e distância do sistema de sonar. DICA: o componente *mux_8x1_n* será fornecido.

O circuito de teste utilizado para a testagem será o próprio **tx_dados_sonar**, e para a verificação dos dados seriais enviados utilizará-se a ferramenta Protocol do Waveforms.

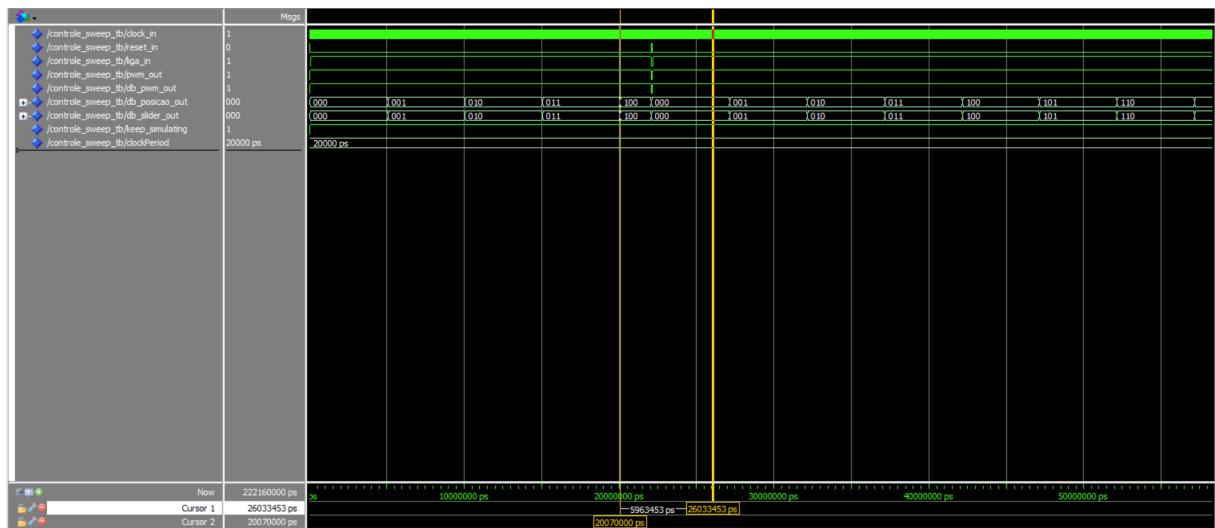
Para a testagem, enviará-se o sinal **”153,017.”** e avaliará-se se o sinal recebido pela ferramenta Protocol condiz com o que foi enviado.

i) Execute o teste de funcionamento destes circuitos de teste usando o ModelSim. DICA: nomeie os projetos dos circuitos de teste: *teste_movimentacao_servomotor* e *teste_tx_dados_sonar*.

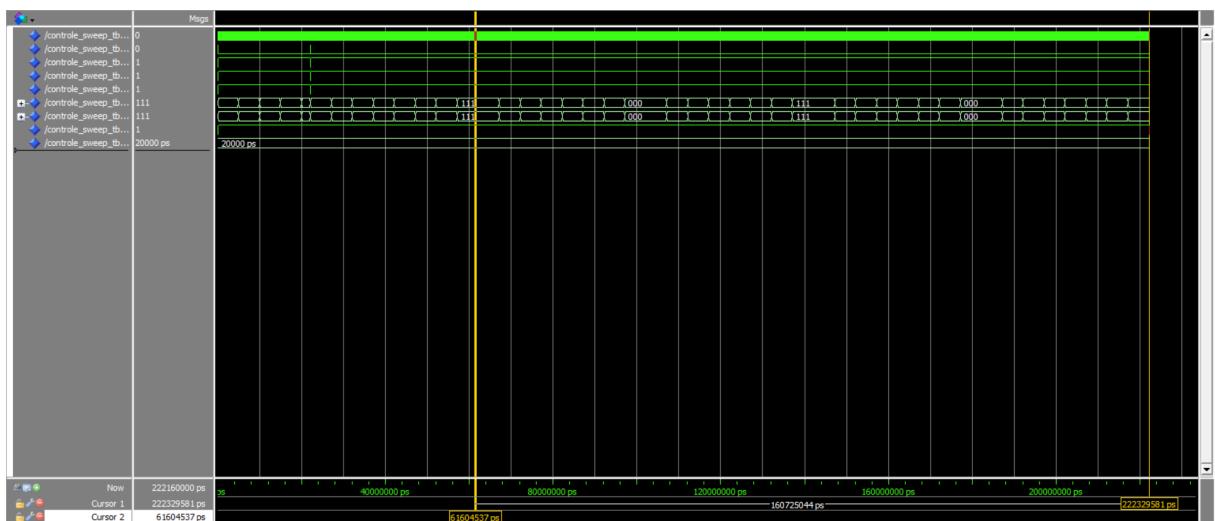
Teste da movimentação do Servomotor

A seguir, a testagem do circuito do servomotor em varredura, conforme foi descrito no planejamento de testes:

Visualização do primeiro teste com reset



Visualização da varredura geral



Teste da transmissão dos dados do sonar

A seguir, a testagem do circuito com o envio do sinal **"153,017."**, conforme foi descrito no cenário de teste:

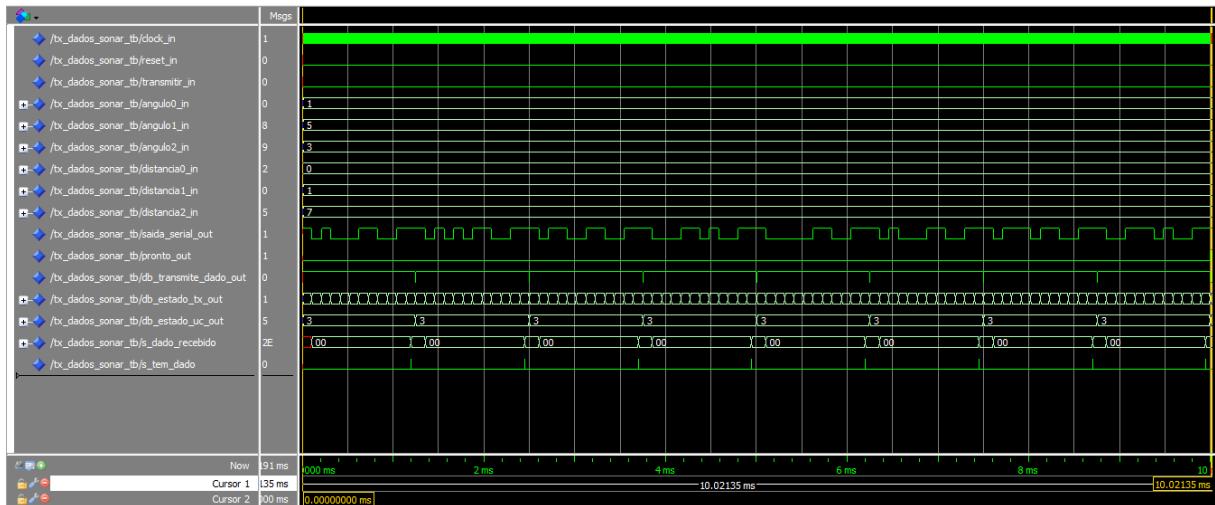


Figura 31: Visão geral do primeiro caso de teste

Em segundo lugar, uma *close* dos sinais, apresentando as saídas lidas em `s_dado_recebido`:

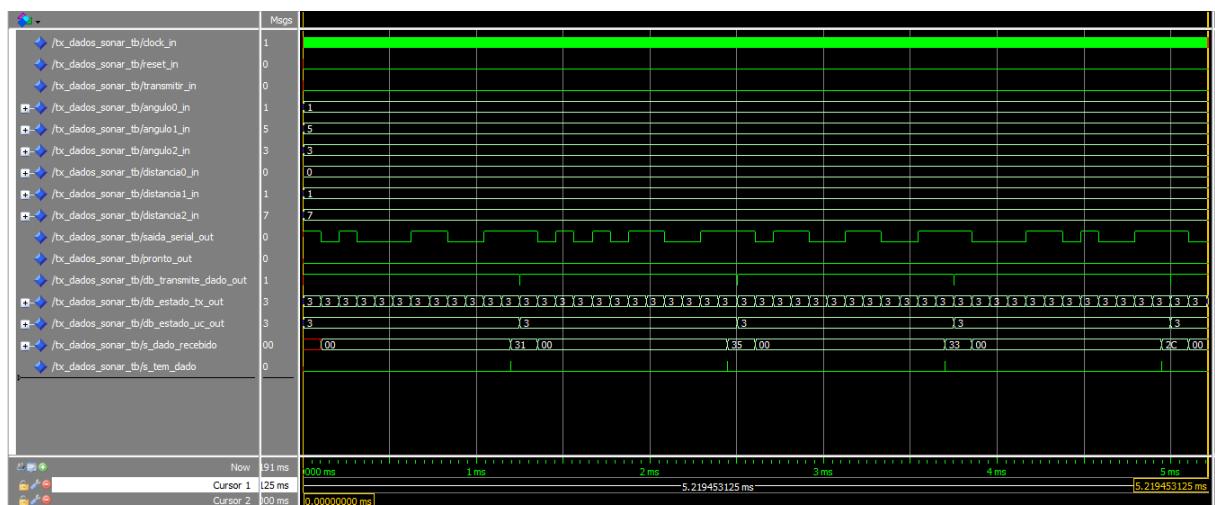


Figura 32: Visão dos 4 primeiros caracteres enviados no primeiro caso de teste

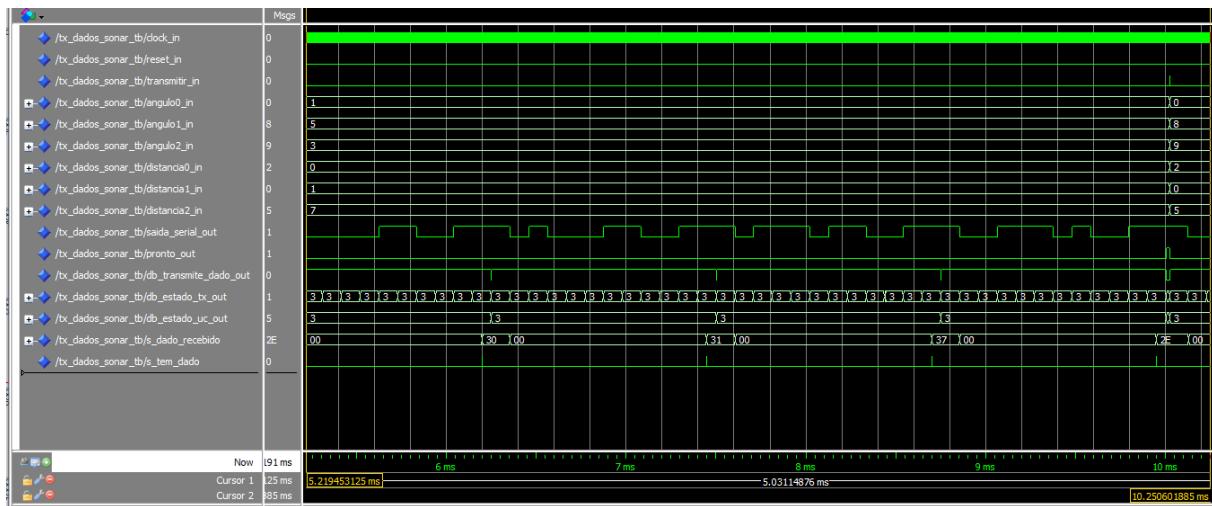


Figura 33: Visão dos 4 últimos caracteres enviados no primeiro caso de teste

Como pode-se ver, os sinais recebidos são equivalentes a **31, 35, 33, 2C, 30, 31, 37, 2E**, que em ASCII representam **"153,017."**, o que era o esperado. Assim, o primeiro caso de teste é considerado um sucesso.

j) Submeta os arquivos QAR dos projetos dos circuitos de teste com o Planejamento. DICAS: nomeie os arquivos como `exp5_txbyy teste_movimentacao_servomotor.qar` e `exp5_txbyy teste_tx_dados_sonar.qar`.

3.3 Atividade 3

k) Os circuitos de teste projetados devem ser implementados na bancada remota no Laboratório Digital usando a placa FPGA DE0-CV. Defina também as designações de sinais aos pinos da FPGA e aos pinos dos projetos no MQTT Dash (se for usado)

Para o circuito de teste da movimentação do servomotor, a pinagem deverá ser a seguinte:

in clock	Input	PIN_M9
out db_posicao[2]	Output	PIN_W2
out db_posicao[1]	Output	PIN_AA1
out db_posicao[0]	Output	PIN_AA2
out db_pwm	Output	PIN_F15
out db_slider[2]	Output	PIN_H15
out db_slider[1]	Output	PIN_A12
out db_slider[0]	Output	PIN_H16
in liga	Input	PIN_B16
out pwm	Output	PIN_K16
in reset	Input	PIN_N16

Figura 34: Pinagem para o controle do servomotor na aplicação

Já para os sinais de MQTT utilizados, serão necessários os botões E0 como reset e E1 como ligar para operar o circuito e a saída slide como forma de visualização, a montagem pode ser vista na imagem abaixo:

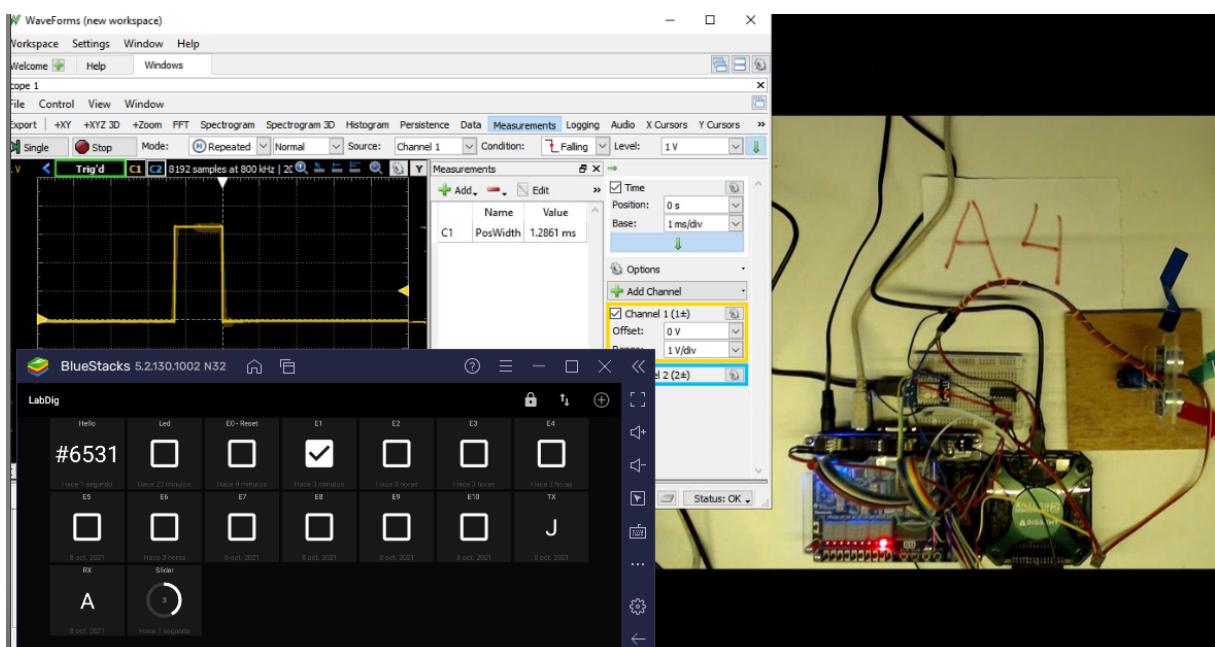


Figura 35: Pinagem para o controle do servomotor na aplicação

Para a testagem do circuito no relatório foram considerados os casos usados no

testbench, detalhado na sessão de testagem do servomotor. Pode-se observar a testagem dos circuitos no laboratório por meio das imagens abaixo:

Caso do servo na posição 010

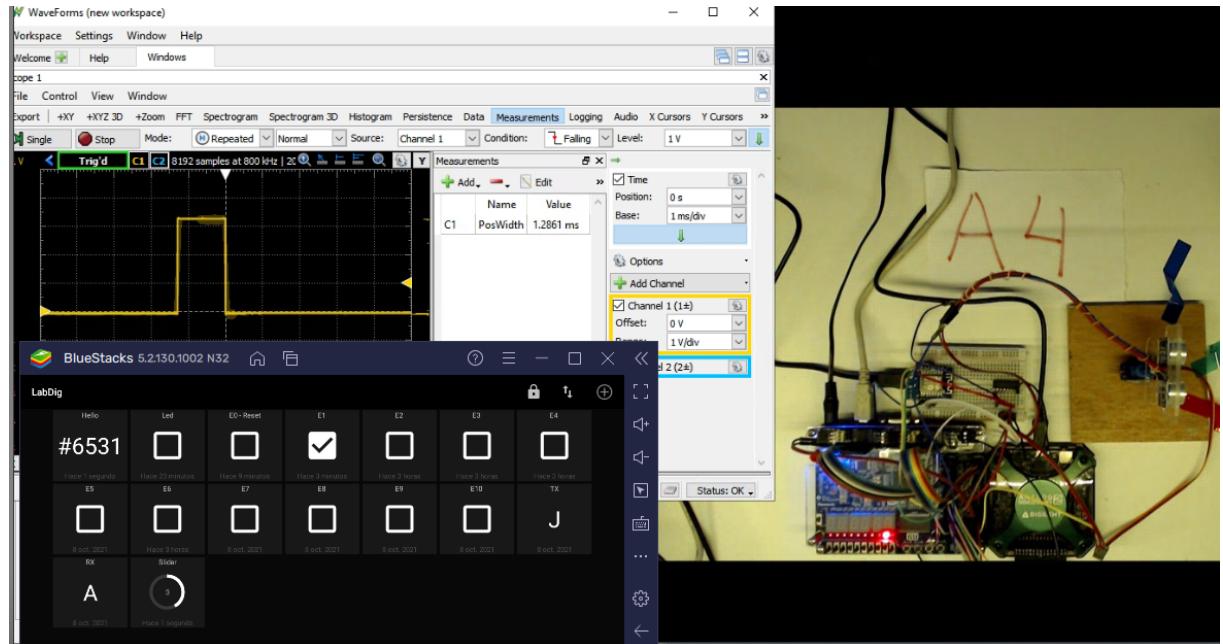


Figura 36: Servomotor na posição 010

Caso do servo na posição 100

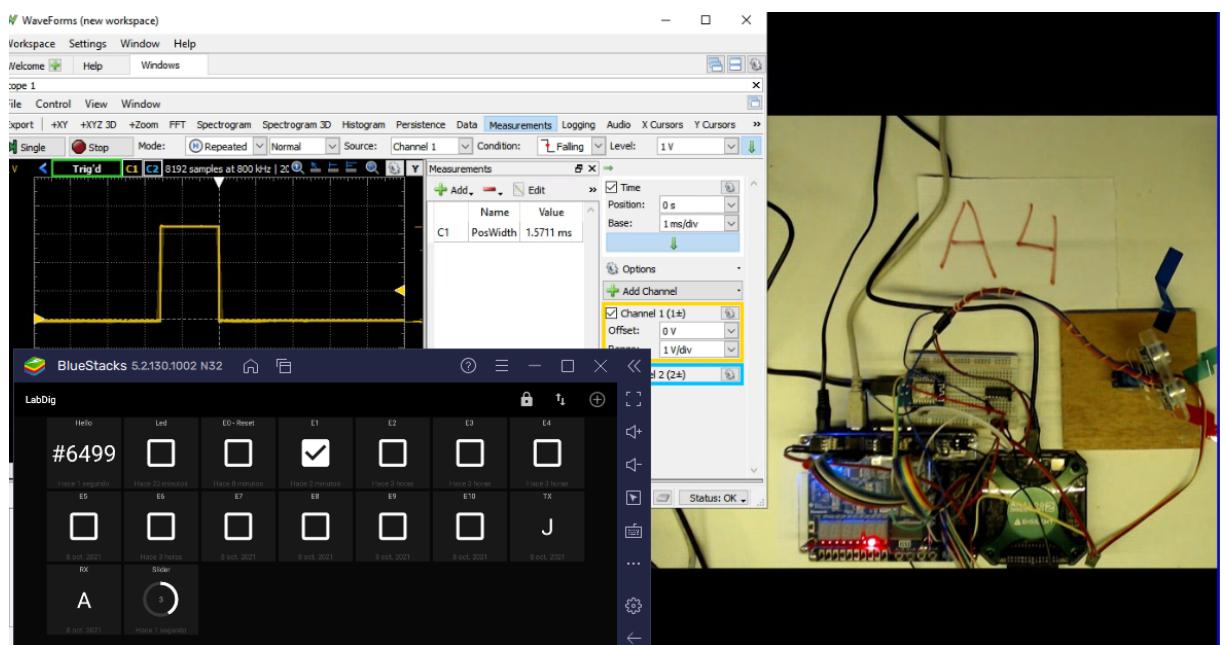


Figura 37: Servomotor na posição 100

Para o circuito do funcionamento do *tx_dados_sonar*, a pinagem deverá ser a seguinte:

in clock	Input	PIN_M9
out db_estado_tx[3]	Output	PIN_U2
out db_estado_tx[2]	Output	PIN_N1
out db_estado_tx[1]	Output	PIN_N2
out db_estado_tx[0]	Output	PIN_Y3
out db_estado_uc[2]	Output	PIN_W2
out db_estado_uc[1]	Output	PIN_AA1
out db_estado_uc[0]	Output	PIN_AA2
out db_saida_serial	Output	PIN_F15
out db_transmite_dado	Output	PIN_L2
out db_transmitir	Output	PIN_U1
out pronto	Output	PIN_L1
in reset	Input	PIN_N16
out saída_serial	Output	PIN_P18
in transmitir	Input	PIN_B16

Figura 38: Pinagem da transmissão de dados

Para a montagem do circuito de teste do circuito foi necessário a adição de um detector de borda para que se evitasse um envio contínuo dos dados para as saídas. Pode-se visualizar a montagem final do circuito de testes abaixo:

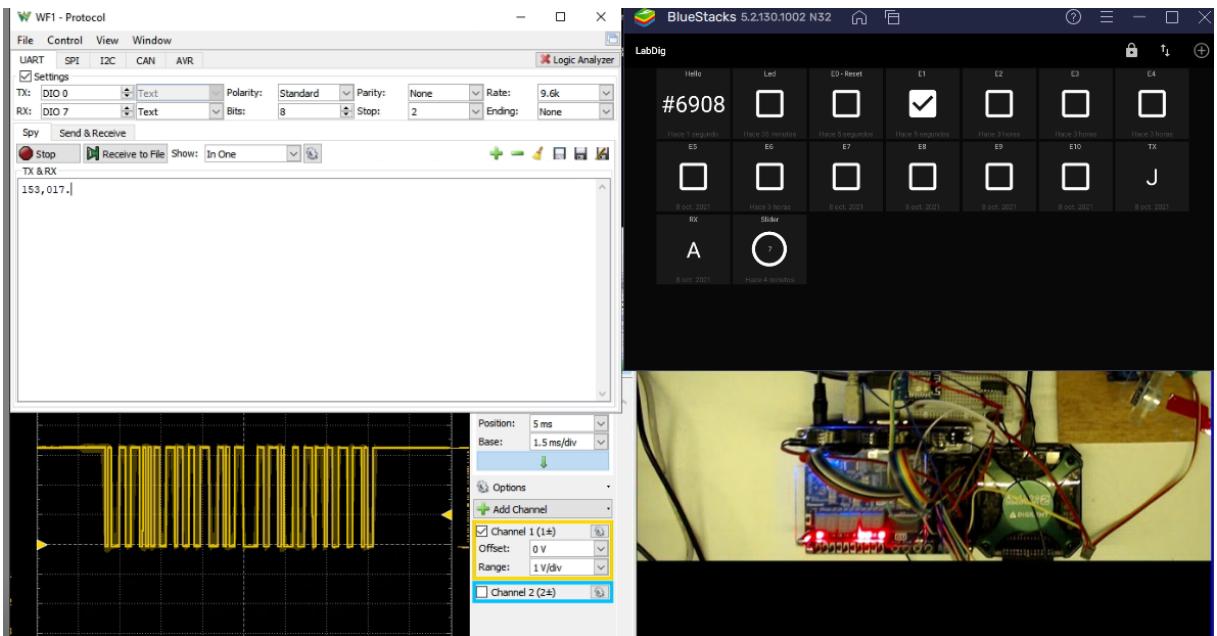


Figura 39: Montagem do circuito de testes

Tendo feito o circuito de teste passou-se para a aplicação e testagem. Verificou-se então os formatos de onda dos sinais na ferramenta *Scope* e o funcionamento do circuito com a ferramenta *Protocol*. Pode-se visualizar ambas as verificações abaixo:

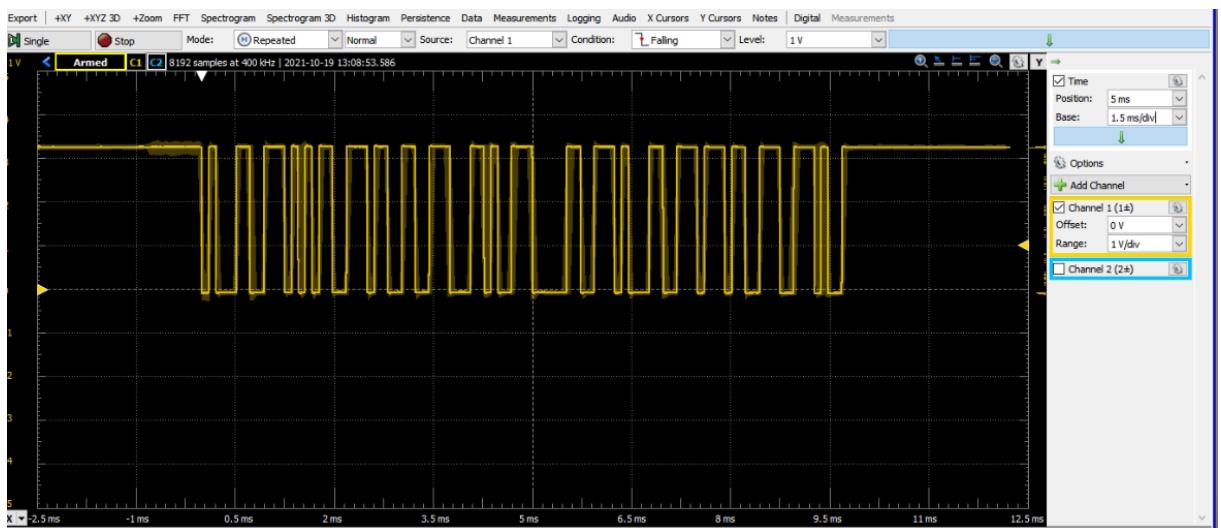


Figura 40: Onda obtida pelo Scope

Os sinais de MQTT utilizados serão o E0 como reset, o E1 como transmitir e uma visualização TX. Eles podem ser visualizados na imagem da montagem do circuito.

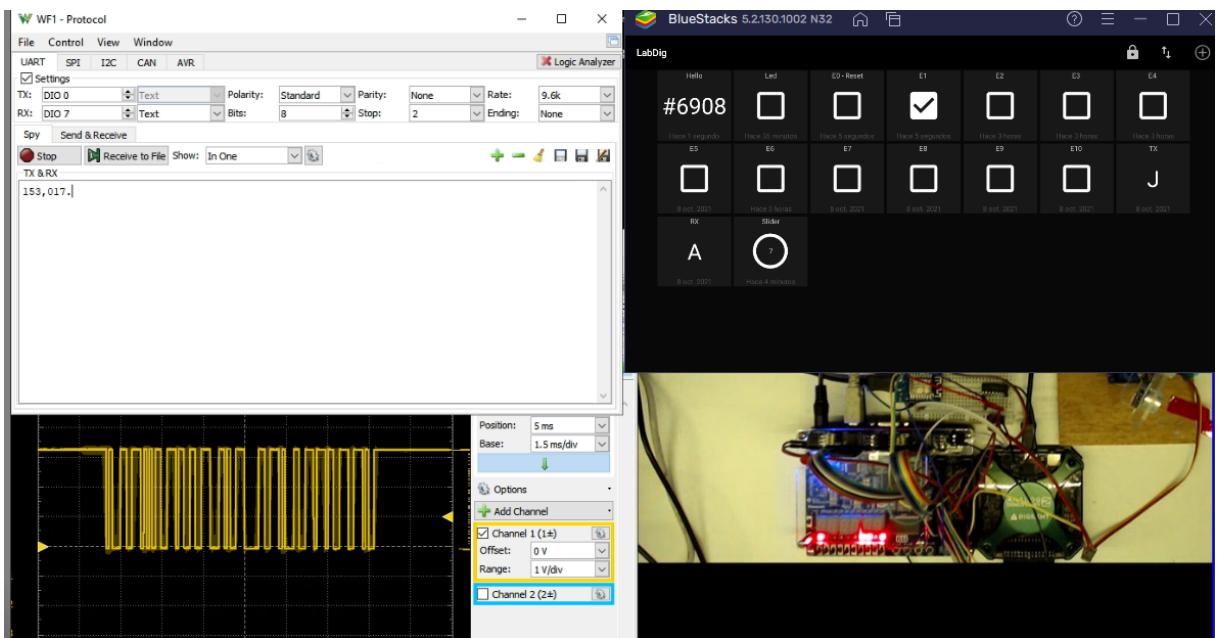


Figura 41: Montagem do circuito e MQTT

1) Relate quaisquer ocorrências experimentais no Relatório

Durante o experimento, notou-se que o módulo de transmissão *tx_dados_sonar* enviava vários pacotes de sinais (8 caracteres) para um único acionamento do sinal de transmitir. Isso se deve ao fato de que, já que a transmissão é muito rápida, o circuito interpretava que era para se repetir a transmissão do sinal. Para resolver isso, o grupo incluiu no circuito de teste o componente *edge-detector*, que reduzia o pulso de input a um único ciclo de clock.

m) Submeta os arquivos QAR dos projetos finais da experiência com o Relatório

Durante a testagem do circuito a saída do *tx_dados_sonar* teve múltiplas ocorrências seguidas do sinal recebido ao invés de uma única saída após o recebimento, isso foi resolvido através da adição de um detector de borda como relatado na montagem do experimento.

4 RESULTADOS

Durante este experimento, foi realizada a refatoração, assimilação e união de diversos circuitos desenvolvidos durante a disciplina de Laboratório Digital II. Com isso, desenvolvemos as seguintes habilidades, cruciais para o âmbito de engenharia:

- Refatoração de projetos;
- Adaptação de projetos antigos para ampliá-los;
- Testagem de novas alterações em um componente modificado;
- União de componentes com o objetivo de produzir um outro;

Como os circuitos foram testados e re-desenvolvidos com sucesso, considera-se que o grupo executou as tarefas com qualidade e habilidade, sempre mantendo ou até mesmo melhorando o padrão de funcionalidade dos projetos anteriores, garantindo a excelência com uma testagem meticulosa e adequada.

Os circuitos desenvolvidos serão reaproveitados para o desenvolvimento do experimento 6, e por isso, o funcionamento ótimo destes é ideal, para não obstruir o desenvolvimento no futuro.

5 DESAFIO

O desafio proposto aos alunos consistia em unir o circuito de controle do servomotor em varredura e a interface do sonar: criando assim, um circuito que mede valores de distância periodicamente a cada posição assumida pelo servomotor.

Assim, para a montagem deste circuito, pensou-se em utilizar o próprio contador para o controle do servomotor (que periodicamente altera as posições do motor) para enviar os sinais de captura de distância para a interface do sensor de distância.

Além disso, para melhor visualização dos resultados obtidos pelo circuito, utilizaram-se componentes que transformam sinais hexadecimais e BCD (que são devolvidos pela interface do sensor de distância), em sinais de sete segmentos, para melhorar a visualização dos sinais.

Assim, o RTL produzido para o desafio foi o seguinte:

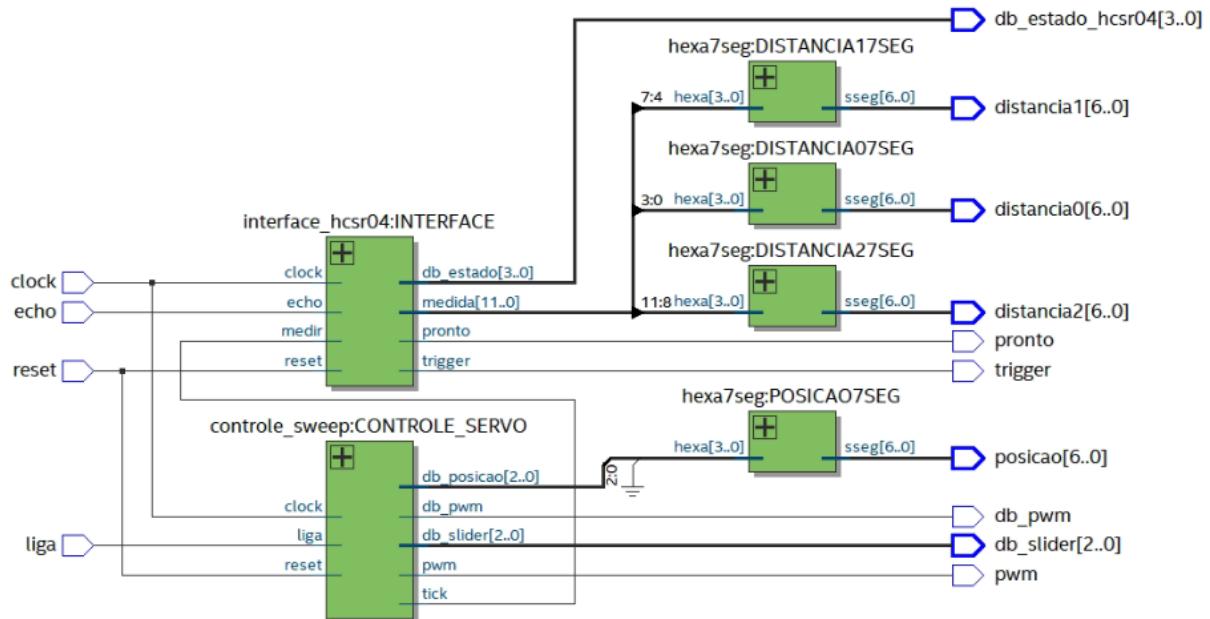


Figura 42: Diagrama RTL do circuito do desafio

Para o funcionamento apropriado do circuito, os sinais de **echo** e **trigger** foram conectados ao componente do sensor de distância. A saída de **pwm** foi repassada para o servomotor, com o objetivo de controlar seu posicionamento. Além disso, as saídas dos 7 segmentos foram colocados nos displays da placa FPGA. O sinal de **pronto** foi alocado

para um dos LEDs da placa FPGA, e a saída **db_pwm** foi repassada para a ferramenta Scope do Waveforms, para a depuração da onda.

Já para os sinais de input, o **clock** é recebido pela própria placa FPGA, enquanto os sinais de **reset** e **liga** são recebidos via MQTT Dash, alocados aos sinais E0 e E1, respectivamente.

Assim, a montagem do circuito para testagem no laboratório digital foi:

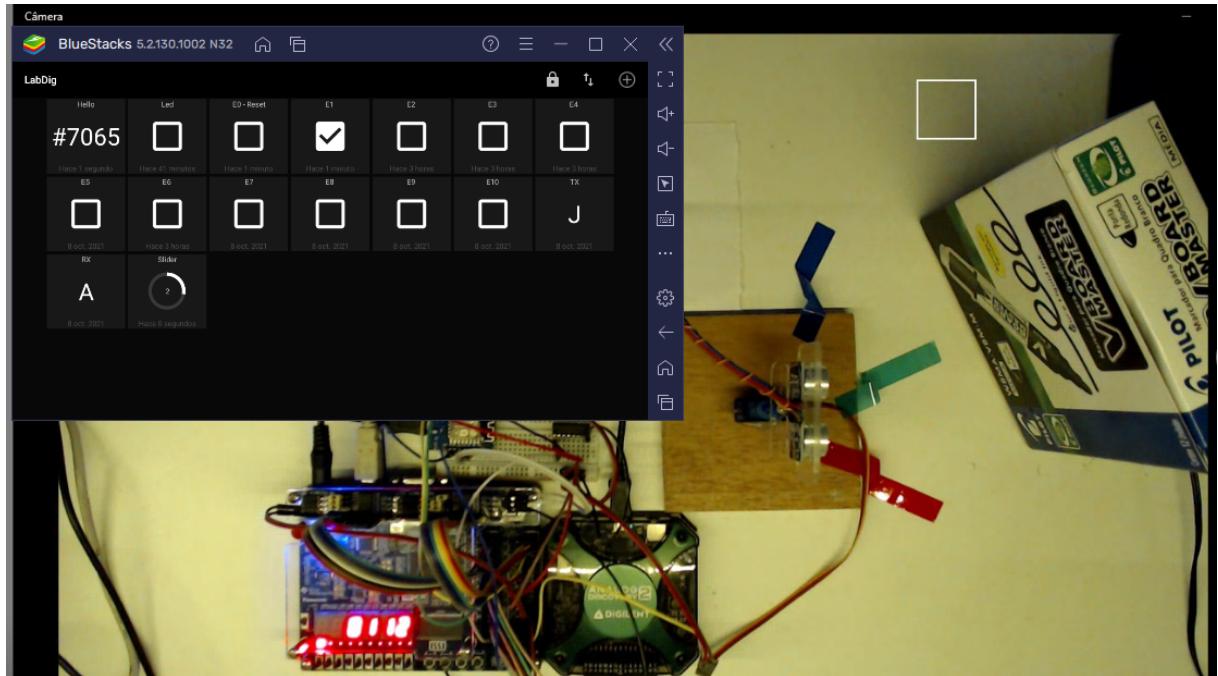


Figura 43: Montagem para testagem do desafio

O funcionamento do circuito, em vídeo, pode ser visualizado no seguinte link: https://www.youtube.com/watch?v=_hJ2RTC0dMU

6 APÊNDICE

6.1 Testbenches

Controle do Servomotor

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3
4 ENTITY controle_sweep_tb IS
5 END ENTITY;
6
7 ARCHITECTURE tb OF controle_sweep_tb IS
8
9 COMPONENT controle_sweep IS
10 PORT (
11     clock : IN STD_LOGIC; — 50MHz
12     reset : IN STD_LOGIC;
13     liga : IN STD_LOGIC;
14     db_posicao : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
15     db_slider : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
16     pwm : OUT STD_LOGIC;
17     db_pwm : OUT STD_LOGIC
18 );
19 END COMPONENT;
20
21 SIGNAL clock_in : STD.LOGIC := '0';
22 SIGNAL reset_in : STD.LOGIC := '0';
23 SIGNAL liga_in : STD.LOGIC := '0';
24 SIGNAL pwm_out : STD.LOGIC;
25 SIGNAL db_pwm_out : STD.LOGIC;
26 SIGNAL db_posicao_out : STD.LOGIC_VECTOR(2 DOWNTO 0);
27 SIGNAL db_slider_out : STD.LOGIC_VECTOR(2 DOWNTO 0);
28
29 SIGNAL keep_simulating : STD.LOGIC := '0';
30 CONSTANT clockPeriod : TIME := 20 ns;
31
32 BEGIN
33
34     clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
35
36     dut : controle_sweep PORT MAP(
37         clock => clock_in,
38         reset => reset_in,
39         liga => liga_in,
40         pwm => pwm_out,
41         db_pwm => db_pwm_out,
42         db_posicao => db_posicao_out,
43         db_slider => db_slider_out
44 );
45
46     stimulus : PROCESS IS
47     BEGIN
48
49         ASSERT false REPORT "Inicio da simulacao" & LF & "... Simulacao ate 800 ms. Aguarde o final da
50             simulacao..." SEVERITY note;
51         keep_simulating <= '1';

```

```

52      —— inicio: reset ———
53      reset_in <= '1';
54      WAIT FOR 2 * clockPeriod;
55      reset_in <= '0';
56      WAIT FOR 2 * clockPeriod;
57
58      —— casos de teste
59
60      —— servo indo para posicao 100
61
62      liga_in <= '1';
63
64      WAIT FOR 1000 * clockPeriod;
65
66      ASSERT db_posicao_out = "100" REPORT "Servo n o demora o tempo esperado para atingir uma pos i o
67      esperada" SEVERITY failure;
68
69      —— perodo para visualiza o do estado
70      WAIT FOR 100 * clockPeriod;
71
72      liga_in <= '0';
73
74      —— teste do comando de reset
75
76      reset_in <= '1';
77      WAIT FOR 2 * clockPeriod;
78      reset_in <= '0';
79      WAIT FOR 2 * clockPeriod;
80
81      ASSERT db_posicao_out = "000" REPORT "Reset n o funciona como esperado" SEVERITY failure;
82
83      —— teste de varredura completa
84
85      liga_in <= '1';
86
87      WAIT FOR 10000 * clockPeriod;
88
89      —— final dos casos de teste da simulacao
90      ASSERT false REPORT "Fim da simulacao" SEVERITY note;
91      keep_simulating <= '0';
92
93      WAIT;
94  END PROCESS;
95 END ARCHITECTURE;

```

6.1.1 Comunicação Serial

tx_serial_tb

```

1 —— Arquivo : tx_serial_tb.vhd
2 —— Projeto : Experiencia 2 — Transmissao Serial Assincrona
3
4 —— Descricao : circuito da experiencia 2
5 ——           > modelo de testbench para simulacao do circuito
6 ——           > de transmissao serial assincrona
7 ——           >
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 tx_serial_tb IS
20 END ENTITY;
21
22 ARCHITECTURE tb OF tx_serial_tb IS
23
24 -- Componente a ser testado (Device Under Test — DUT)
25 COMPONENT tx_serial_8N2
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     saida_serial : OUT STD_LOGIC;
32     pronto_tx : OUT STD_LOGIC;
33     db_partida : OUT STD_LOGIC;
34     db_saida_serial : OUT STD_LOGIC;
35     db_estado : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
36 );
37 END COMPONENT;
38
39 -- Declara o de sinais para conectar o componente a ser testado (DUT)
40 -- valores iniciais para fins de simulacao (ModelSim)
41 SIGNAL clock_in : STD_LOGIC := '0';
42 SIGNAL reset_in : STD_LOGIC := '0';
43 SIGNAL partida_in : STD_LOGIC := '0';
44 SIGNAL dados_ascii_8_in : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
45
46 SIGNAL saida_serial_out : STD_LOGIC;
47 SIGNAL pronto_tx_out : STD_LOGIC;
48
49 SIGNAL db_partida_out : STD_LOGIC;
50 SIGNAL db_saida_serial_out : STD_LOGIC;
51 SIGNAL db_estado_out : STD_LOGIC_VECTOR (3 DOWNTO 0);
52
53 -- Configura es do clock
54 SIGNAL keep_simulating : STD_LOGIC := '0'; -- delimita o tempo de gera o do clock
55 CONSTANT clockPeriod : TIME := 20 ns; -- clock de 50MHz
56
57 BEGIN
58     -- Gerador de clock: executa enquanto 'keep_simulating = 1', com o periodo
59     -- especificado. Quando keep_simulating=0, clock interrompido, bem como a
60     -- simula o de eventos
61     clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
62
63     -- Conecta DUT (Device Under Test)
64     dut : tx_serial_8N2
65     PORT MAP
66     (
67         clock => clock_in,
68         reset => reset_in,
69         partida => partida_in,
70         dados_ascii => dados_ascii_8_in,
71         saida_serial => saida_serial_out,
72         pronto_tx => pronto_tx_out,
73         db_partida => db_partida_out,
74         db_saida_serial => db_saida_serial_out,
75         db_estado => db_estado_out
76     );
77
78     -- geracao dos sinais de entrada (estimulos)

```

```

79  stimulus : PROCESS IS
80  BEGIN
81
82    ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
83    keep_simulating <= '1';
84
85    —— inicio da simulacao: reset ——
86    partida_in <= '0';
87    reset_in <= '1';
88    WAIT FOR 20 * clockPeriod; — pulso com 20 periodos de clock
89    reset_in <= '0';
90    WAIT UNTIL falling_edge(clock_in);
91    WAIT FOR 50 * clockPeriod;
92
93    —— dado de entrada da simulacao (caso de teste #1)
94    dados_ascii_8_in <= "00110101"; — x35 = '5'
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_tx_out = '1';
105
106   —— final do caso de teste 1
107
108   —— intervalo entre casos de teste
109   WAIT FOR 2000 * clockPeriod;
110
111   —— dado de entrada da simulacao (caso de teste #2)
112   dados_ascii_8_in <= "00101001"; — x29 = ')'
113   WAIT FOR 20 * clockPeriod;
114
115   —— acionamento da partida (inicio da transmissao)
116   partida_in <= '1';
117   WAIT UNTIL rising_edge(clock_in);
118   WAIT FOR 5 * clockPeriod; — pulso partida com 5 periodos de clock
119   partida_in <= '0';
120
121   —— espera final da transmissao (pulso pronto em 1)
122   WAIT UNTIL pronto_tx_out = '1';
123
124   —— final do caso de teste 2
125
126   —— intervalo
127   WAIT FOR 2000 * clockPeriod;
128
129   —— final dos casos de teste da simulacao
130   ASSERT false REPORT "Fim da simulacao" SEVERITY note;
131   keep_simulating <= '0';
132
133   WAIT; — fim da simulacao: aguarda indefinidamente
134 END PROCESS;
135 END ARCHITECTURE;

```

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.WRITEBYTE)
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);
31 SIGNAL pronto_out : STD.LOGIC;
32 SIGNAL tem_dado_out : STD.LOGIC;
33 SIGNAL db_recebe_dado_out : STD.LOGIC;
34 SIGNAL db_estado_out : STD.LOGIC_VECTOR(3 DOWNTO 0);
35
36 — para procedimento UART.WRITEBYTE
37 SIGNAL entrada_serial_in : STD.LOGIC := '1';
38 SIGNAL serialData : STD.LOGIC_VECTOR(7 DOWNTO 0) := "00000000";
39
40 — Configuraes do clock
41 CONSTANT clockPeriod : TIME := 20 ns; — 50MHz
42 CONSTANT bitPeriod : TIME := 5208 * clockPeriod; — 5208 clocks por bit (9.600 bauds)
43 — constant bitPeriod : time := 454*clockPeriod; — 454 clocks por bit (115.200 bauds)
44
45 — Procedimento para geracao da sequencia de comunicacao serial 8N2
46 — adaptacao de codigo acessado de:
47 — https://www.nandland.com/goboard/uart-go-board-project-part1.html
48 PROCEDURE UART.WRITEBYTE (
49     Data_In : IN STD.LOGIC_VECTOR(7 DOWNTO 0);
50     SIGNAL Serial_Out : OUT STD.LOGIC) IS
51 BEGIN
52
53     — envia Start Bit
54     Serial_Out <= '0';
55     WAIT FOR bitPeriod;
56
57     — envia Dado de 8 bits
58     FOR ii IN 0 TO 7 LOOP
59         Serial_Out <= Data_In(ii);
60         WAIT FOR bitPeriod;
61     END LOOP; — loop ii
62
63     — envia 2 Stop Bits
64     Serial_Out <= '1';
65     WAIT FOR 2 * bitPeriod;
66
67 END UART.WRITEBYTE;
68 — fim procedure
69
70 —— Array de casos de teste

```

```

71  TYPE caso_teste_type IS RECORD
72    id : NATURAL;
73    data : STD.LOGIC.VECTOR(7 DOWNTO 0);
74  END RECORD;
75
76  TYPE casos_teste_array IS ARRAY (NATURAL RANGE <>) OF caso_teste_type;
77  CONSTANT casos_teste : casos_teste_array :=
78  (
79    (1, "00110101"), — 35H
80    (2, "01010101"), — 55H
81    (3, "10101010"), — AAH
82    (4, "11111111"), — FFH
83    (5, "00000000") — 00H
84    — inserir aqui outros casos de teste (inserir "," na linha anterior)
85  );
86
87  SIGNAL keep_simulating : STD.LOGIC := '0'; — delimita o tempo de geração do clock
88
89 BEGIN
90
91  —— Gerador de Clock
92  clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
93
94  —— Instantancia o direta DUT (Device Under Test)
95  DUT : ENTITY work.rx_serial_8N2
96    PORT MAP(
97      clock => clock_in,
98      reset => reset_in,
99      dado_serial => entrada_serial_in,
100     recebe_dado => recebe_in,
101     dado_recebido => dado_out,
102     tem_dado => tem_dado_out,
103     pronto_rx => pronto_out,
104     db_recebe_dado => db_recebe_dado_out,
105     db_estado => db_estado_out
106   );
107
108 —— Geracao dos sinais de entrada (estimulo)
109 stimulus : PROCESS IS
110 BEGIN
111
112  —— inicio da simulacao
113  ASSERT false REPORT "inicio da simulacao" SEVERITY note;
114  keep_simulating <= '1';
115  —— reset
116  reset_in <= '0';
117  WAIT FOR bitPeriod;
118  reset_in <= '1', '0' AFTER 2 * clockPeriod;
119  WAIT FOR bitPeriod;
120
121  —— loop pelos casos de teste
122  FOR i IN casos_teste'RANGE LOOP
123    ASSERT false REPORT "Caso de teste " & INTEGER'image(casos_teste(i).id) SEVERITY note;
124    serialData <= casos_teste(i).data; — caso de teste "i"
125    WAIT FOR 10 * clockPeriod;
126
127    —— 1) envia bits seriais para circuito de recepcao
128    UART.WRITE_BYT( Data_In => serialData, Serial_Out => entrada_serial_in );
129    entrada_serial_in <= '1'; — repouso
130    WAIT FOR bitPeriod;
131
132    —— 2) simula recebimento do dado (p.ex. circuito principal registra saida)
133    WAIT UNTIL falling_edge(clock_in);
134    recebe_in <= '1', '0' AFTER 2 * clockPeriod;
135
136    —— 3) intervalo entre casos de teste

```

```

137      WAIT FOR 2 * bitPeriod;
138  END LOOP;
139
140  —— final dos casos de teste da simulacao
141  ASSERT false REPORT "fim da simulacao" SEVERITY note;
142  keep_simulating <= '0';
143
144  WAIT; — fim da simulacao : aguarda indefinidamente
145
146  END PROCESS stimulus;
147
148 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     transmite_dado : IN STD_LOGIC;
30     dados_ascii : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
31     dado_serial : IN STD_LOGIC;
32     recebe_dado : IN STD_LOGIC;
33     saida_serial : OUT STD_LOGIC;
34     pronto_tx : OUT STD_LOGIC;
35     dado_recebido_rx : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
36     tem_dado : OUT STD_LOGIC;
37     pronto_rx : OUT STD_LOGIC;
38     db_transmite_dado : OUT STD_LOGIC;
39     db_saida_serial : OUT STD_LOGIC;
40     db_estado_tx : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
41     db_recebe_dado : OUT STD_LOGIC;
42     db_dado_serial : OUT STD_LOGIC;
43     db_estado_rx : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
44   );
45 END COMPONENT;
46
47 — Declara o de sinais para conectar o componente a ser testado (DUT)
48 — valores iniciais para fins de simulacao (ModelSim)
49 SIGNAL clock_in : STD_LOGIC := '0';

```

```

50      SIGNAL reset_in : STD_LOGIC := '0';
51      SIGNAL transmite_dado_in : STD_LOGIC := '0';
52      SIGNAL recebe_dado_in : STD_LOGIC := '0';
53      SIGNAL dados_ascii_8_in : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
54      SIGNAL dado_recebido_out : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
55      SIGNAL saida_serial_out : STD_LOGIC := '1';
56      SIGNAL pronto_rx_out : STD_LOGIC := '0';
57      SIGNAL pronto_tx_out : STD_LOGIC := '0';
58      SIGNAL tem_dado_out : STD_LOGIC := '0';
59      SIGNAL db_recebe_dado_out, db_dado_serial_out : STD_LOGIC := '0';
60      SIGNAL db_transmite_dado_out, db_saida_serial_rx_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 (3 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 periodo
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          transmite_dado => transmite_dado_in,
81          dados_ascii => dados_ascii_8_in,
82          dado_serial => saida_serial_out,
83          recebe_dado => recebe_dado_in,
84          saida_serial => saida_serial_out,
85          pronto_tx => pronto_tx_out,
86          dado_recebido_rx => dado_recebido_out,
87          tem_dado => tem_dado_out,
88          pronto_rx => pronto_rx_out,
89          db_transmite_dado => db_transmite_dado_out,
90          db_saida_serial => db_saida_serial_rx_out,
91          db_estado_tx => db_estado_tx_out,
92          db_recebe_dado => db_recebe_dado_out,
93          db_dado_serial => db_dado_serial_out,
94          db_estado_rx => db_estado_rx_out
95      );
96
97      — geracao dos sinais de entrada (estimulos)
98      stimulus : PROCESS IS
99      BEGIN
100
101         ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
102         keep_simulating <= '1';
103
104         — inicio da simulacao: reset —————
105         transmite_dado_in <= '0';
106         reset_in <= '1';
107         WAIT FOR 20 * clockPeriod; — pulso com 20 periodos de clock
108         reset_in <= '0';
109         WAIT UNTIL falling_edge(clock_in);
110         WAIT FOR 50 * clockPeriod;
111
112         — dado de entrada da simulacao (caso de teste #1)
113         dados_ascii_8_in <= "00110101"; — x35 = '5'
114         WAIT FOR 20 * clockPeriod;
115

```

```

116      —— acionamento da transmite_dado (inicio da transmissao)
117      transmite_dado_in <= '1';
118      WAIT UNTIL rising_edge(clock_in);
119      WAIT FOR 5 * clockPeriod; — pulso transmite_dado com 5 periodos de clock
120      transmite_dado_in <= '0';
121
122      —— espera final da transmissao (pulso pronto em 1)
123      WAIT UNTIL pronto_tx_out = '1';
124
125      —— final do caso de teste 1
126
127      —— intervalo entre casos de teste
128      WAIT FOR 2000 * clockPeriod;
129
130      —— dado de entrada da simulacao (caso de teste #2)
131      recebe_dado_in <= '1';
132      WAIT FOR 20 * clockPeriod;
133      recebe_dado_in <= '0';
134      dados_ascii_8_in <= "00101001"; — x29 = ')'
135      WAIT FOR 100 * clockPeriod;
136
137      —— acionamento da transmite_dado (inicio da transmissao)
138      transmite_dado_in <= '1';
139      WAIT UNTIL rising_edge(clock_in);
140      WAIT FOR 5 * clockPeriod; — pulso transmite_dado com 5 periodos de clock
141      transmite_dado_in <= '0';
142
143      —— espera final da transmissao (pulso pronto em 1)
144      WAIT UNTIL pronto_tx_out = '1';
145
146      —— final do caso de teste 2
147
148      —— intervalo
149      WAIT FOR 2000 * clockPeriod;
150
151      —— final dos casos de teste da simulacao
152      ASSERT false REPORT "Fim da simulacao" SEVERITY note;
153      keep_simulating <= '0';
154
155      WAIT; — fim da simulacao: aguarda indefinidamente
156  END PROCESS;
157 END ARCHITECTURE;

```

6.1.2 Transmissão dos dados do sonar

tx_dados_sonar_tb

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.numeric_std.ALL;
4
5 ENTITY tx_dados_sonar_tb IS
6 END ENTITY;
7
8 ARCHITECTURE tb OF tx_dados_sonar_tb IS
9
10 — Componente a ser testado (Device Under Test — DUT)
11 COMPONENT tx_dados_sonar
12   PORT (
13     clock : IN STD_LOGIC;
14     reset : IN STD_LOGIC;
15     transmitir : IN STD_LOGIC;

```

```

16     angulo2 : IN STD.LOGIC_VECTOR(3 DOWNTO 0); — digitos BCD
17     angulo1 : IN STD.LOGIC_VECTOR(3 DOWNTO 0); — de angulo
18     angulo0 : IN STD.LOGIC_VECTOR(3 DOWNTO 0);
19     distancia2 : IN STD.LOGIC_VECTOR(3 DOWNTO 0); — digitos de distancia
20     distancial : IN STD.LOGIC_VECTOR(3 DOWNTO 0);
21     distancia0 : IN STD.LOGIC_VECTOR(3 DOWNTO 0);
22     saida_serial : OUT STD.LOGIC;
23     pronto : OUT STD.LOGIC;
24     db_transmitir : OUT STD.LOGIC;
25     db_transmite_dado : OUT STD.LOGIC;
26     db_saida_serial : OUT STD.LOGIC;
27     db_estado_tx : OUT STD.LOGIC_VECTOR(3 DOWNTO 0);
28     db_estado_uc : OUT STD.LOGIC_VECTOR (2 DOWNTO 0)
29 );
30 END COMPONENT;
31
32 COMPONENT uart_8n2 PORT (
33     clock : IN STD.LOGIC;
34     reset : IN STD.LOGIC;
35     transmite_dado : IN STD.LOGIC;
36     dados_ascii : IN STD.LOGIC_VECTOR (7 DOWNTO 0);
37     dado_serial : IN STD.LOGIC;
38     recebe_dado : IN STD.LOGIC;
39     saida_serial : OUT STD.LOGIC;
40     pronto_tx : OUT STD.LOGIC;
41     dado_recebido_rx : OUT STD.LOGIC_VECTOR (7 DOWNTO 0);
42     tem_dado : OUT STD.LOGIC;
43     pronto_rx : OUT STD.LOGIC;
44     db_transmite_dado : OUT STD.LOGIC;
45     db_saida_serial : OUT STD.LOGIC;
46     db_estado_tx : OUT STD.LOGIC_VECTOR(3 DOWNTO 0);
47     db_partida : OUT STD.LOGIC;
48     db_recebe_dado : OUT STD.LOGIC;
49     db_dado_serial : OUT STD.LOGIC;
50     db_estado_rx : OUT STD.LOGIC_VECTOR (3 DOWNTO 0)
51 );
52 END COMPONENT;
53
54 — Declara o de sinais para conectar o componente a ser testado (DUT)
55 — valores iniciais para fins de simulacao (ModelSim)
56 SIGNAL clock_in : STD.LOGIC := '0';
57 SIGNAL reset_in : STD.LOGIC := '0';
58
59 SIGNAL transmitir_in : STD.LOGIC := '0';
60 SIGNAL angulo2_in, angulo1_in, angulo0_in : STD.LOGIC_VECTOR (3 DOWNTO 0) := "0000";
61 SIGNAL distancia2_in, distancial_in, distancia0_in : STD.LOGIC_VECTOR (3 DOWNTO 0) := "0000";
62
63 SIGNAL saida_serial_out : STD.LOGIC;
64 SIGNAL pronto_out : STD.LOGIC;
65 SIGNAL db_transmite_dado_out : STD.LOGIC;
66 SIGNAL db_estado_tx_out : STD.LOGIC_VECTOR(3 DOWNTO 0);
67 SIGNAL db_estado_uc_out : STD.LOGIC_VECTOR(2 DOWNTO 0);
68
69 SIGNAL s_dado_recebido : STD.LOGIC_VECTOR (7 DOWNTO 0);
70 SIGNAL s_tem_dado : STD.LOGIC;
71
72 — Configura es do clock
73 SIGNAL keep_simulating : STD.LOGIC := '0'; — delimita o tempo de gera o do clock
74 CONSTANT clockPeriod : TIME := 20 ns; — clock de 50MHz
75
76 BEGIN
77     — Gerador de clock: executa enquanto 'keep_simulating = 1', com o per odo
78     — especificado. Quando keep_simulating=0, clock interrompido, bem como a
79     — simula o de eventos
80     clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
81

```

```

82 — Conecta DUT (Device Under Test)
83 dut : tx_dados_sonar PORT MAP(
84     clock => clock_in ,
85     reset => reset_in ,
86     transmitir => transmitir_in ,
87     angulo2 => angulo2_in ,
88     angulo1 => angulo1_in ,
89     angulo0 => angulo0_in ,
90     distancia2 => distancia2_in ,
91     distancia1 => distancia1_in ,
92     distancia0 => distancia0_in ,
93     saida_serial => saida_serial_out ,
94     pronto => pronto_out ,
95     db_transmitir => OPEN,
96     db_transmite_dado => db_transmite_dado_out ,
97     db_saida_serial => OPEN,
98     db_estado_tx => db_estado_tx_out ,
99     db_estado_uc => db_estado_uc_out
100 );
101
102 uart : uart_8n2 PORT MAP(
103     clock => clock_in ,
104     reset => reset_in ,
105     transmite_dado => '0',
106     dados_ascii => "00000000",
107     dado_serial => saida_serial_out ,
108     recebe_dado => s_tem_dado ,
109     saida_serial => OPEN,
110     pronto_tx => OPEN,
111     dado_recebido_rx => s_dado_recebido ,
112     tem_dado => s_tem_dado ,
113     pronto_rx => OPEN,
114     db_transmite_dado => OPEN,
115     db_saida_serial => OPEN,
116     db_estado_tx => OPEN,
117     db_partida => OPEN,
118     db_recebe_dado => OPEN,
119     db_dado_serial => OPEN,
120     db_estado_rx => OPEN
121 );
122
123 — geracao dos sinais de entrada (estimulos)
124 stimulus : PROCESS IS
125 BEGIN
126
127     ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
128     keep_simulating <= '1';
129
130     —— inicio da simulacao: reset ————
131     transmitir_in <= '0';
132     reset_in <= '1';
133     WAIT FOR 20 * clockPeriod; — pulso com 20 periodos de clock
134     reset_in <= '0';
135     WAIT UNTIL falling_edge(clock_in);
136     WAIT FOR 20 * clockPeriod;
137
138     —— Envio de 153,017. (caso de teste 1)
139     angulo0_in <= "0001";
140     angulo1_in <= "0101";
141     angulo2_in <= "0011";
142     distancia0_in <= "0000";
143     distancia1_in <= "0001";
144     distancia2_in <= "0111";
145
146     WAIT FOR 10 * clockPeriod;
147

```

```

148      transmitir_in <= '1';
149
150      WAIT FOR 20 * clockPeriod;
151
152      transmitir_in <= '0';
153
154      WAIT UNTIL pronto_out = '1';
155
156      —— Intervalo
157      WAIT FOR 1000 * clockPeriod;
158
159      —— Envio de 089,205. (caso de teste 1)
160      angulo0_in <= "0000";
161      angulo1_in <= "1000";
162      angulo2_in <= "1001";
163      distancia0_in <= "0010";
164      distancia1_in <= "0000";
165      distancia2_in <= "0101";
166
167      WAIT FOR 10 * clockPeriod;
168
169      transmitir_in <= '1';
170
171      WAIT FOR 20 * clockPeriod;
172
173      transmitir_in <= '0';
174
175      WAIT UNTIL pronto_out = '1';
176
177      —— Intervalo
178      WAIT FOR 1000 * clockPeriod;
179
180      —— final dos casos de teste da simulacao
181      ASSERT false REPORT "Fim da simulacao" SEVERITY note;
182      keep_simulating <= '0';
183
184      WAIT; — fim da simula o: aguarda indefinidamente
185  END PROCESS;
186 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.