

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 4

Prof. Edson Midorikawa

SÃO PAULO - SP

2021

SUMÁRIO

1	INTRODUÇÃO	4
1.1	Visão Geral: Interface com Sensor Ultrassônico de Distância	4
1.2	Funcionalidade	4
1.3	Não Funcionalidade	5
2	SOLUÇÃO TÉCNICA	6
2.1	Descrição Geral	6
3	ATIVIDADES	7
3.1	Atividade 1	7
3.2	Atividade 2	21
3.3	Atividade 3	21
3.4	Atividade 4	24
4	RESULTADOS	27
5	DESAFIO	28
5.1	Descrição Geral	28
6	APÊNDICE	31
6.1	Circuito Final	31
6.1.1	exp4_sensor	31
6.2	Circuito Base	33
6.2.1	interface_hcsr04	33
6.2.2	interface_hcsr04_fd	34
6.2.3	interface_hcsr04_uc	35
6.2.4	medidor_largura	37
6.2.5	gerador_pulso	38
6.2.6	contador_divisor	39
6.2.7	contador_bcd_4digitos	40
6.2.8	registraror_n	42

6.3 Circuito do Desafio.	42
6.3.1 exp4_sensor_desafio	42
6.3.2 controle_servo	45
6.4 Testbenches	46
6.4.1 interface_hcsr04_tb	46
7 REFERÊNCIAS BIBLIOGRÁFICAS	49

1 INTRODUÇÃO

1.1 Visão Geral: Interface com Sensor Ultrassônico de Distância

Durante este experimento, os alunos terão que aprender a trabalhar com o sensor ultrassônico HC-SR04, sintetizando uma interface para a utilização da peça.

O sensor funciona a partir do envio de 8 pulsos ultrassônicos que com sua reflexão em um obstáculo ou anteparo geram a medida da distância entre o sonar e o objeto, o acionamento dessa tomada de medida se da pela emissão de um pulso de $10\mu s$. O sonar utilizado apresenta um sinal de entrada e um de saída, sendo o primeiro por onde é mandado o pulso de acionamento e o último por onde devolve-se um sinal *Echo* com largura proporcional a distância medida.



Figura 1: Sensor HC-SR04

1.2 Funcionalidade

O circuito funcionará como forma de controle para o sensor sendo possível por meio dele mandar o sinal de medida para a peça e interpretar o sinal de saída devolvendo diretamente a medida feita por ele.

Arquiteturalmente sua composição é de uma unidade de controle e um fluxo de estado com as seguinte peças:

- **Gerador** de pulso trigger com $10\mu s$ de duração
- **Medidor** de largura do pulso Echo

- **Registrador** da saída

1.3 Não Funcionalidade

Durante este experimento, não será implementado: medidas de distâncias que não tenham como unidade básica 1 cm, casos com movimentação da aparelhagem do sonar e mapeamento de múltiplas distâncias.

2 SOLUÇÃO TÉCNICA

2.1 Descrição Geral

Neste experimento da disciplina de Laboratório Digital II, será desenvolvido o projeto 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 o projeto do circuito de interface com o sensor ultrassônico de distância, conforme especificação apresentada na seção 1.2. Apresente as decisões de projeto e detalhes do seu funcionamento no Planejamento. Anexe os diagramas de projeto referentes ao Fluxo de Dados e da Unidade de Controle

Durante o projeto, optou-se por começar pela estruturação do fluxo de dados do circuito. Desta forma, apresentam-se 3 principais componentes no fluxo de dados: **gerador** de pulso trigger, **medidor** de largura do pulso Echo e o **registrador** que armazena o dado de saída.

Desta forma, obtém-se o seguinte fluxo de dados:

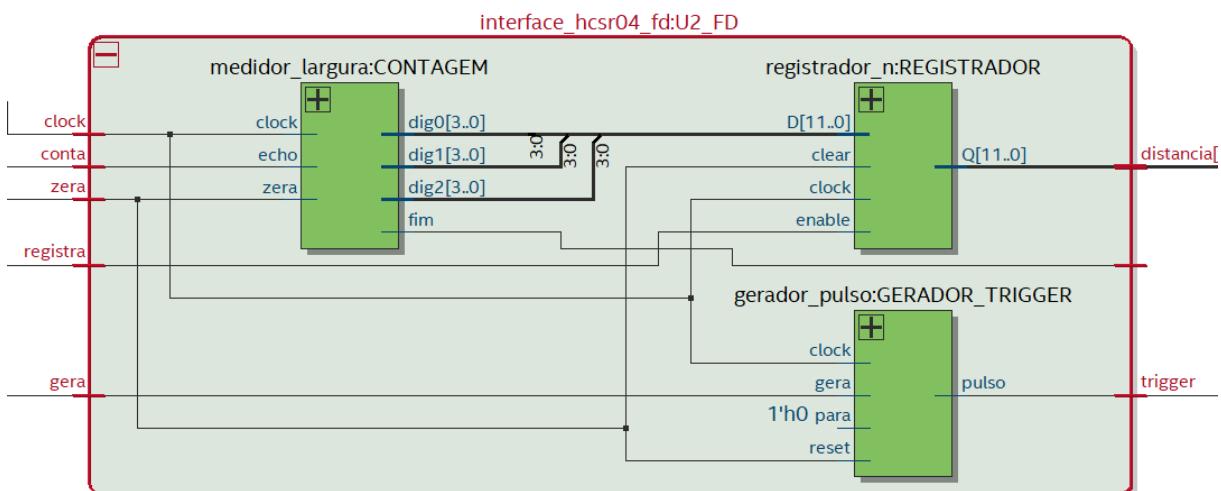


Figura 2: Diagrama RTL do fluxo de dados.

Gerador de pulso:

Este componente é responsável por gerar os pulsos para serem enviados ao componente HCSR04, necessários para gerar as ondas ultrassônicas utilizadas nas medições das distâncias. Para tanto, é necessário a entrada "clock" para determinar a largura destes pulsos, uma entrada "gera" que determina quando os pulsos devem ser gerados, e uma entrada "reset" para resetar o circuito, se necessário.

O gerador possui uma máquina de estados interna, representada pelo seguinte diagrama:

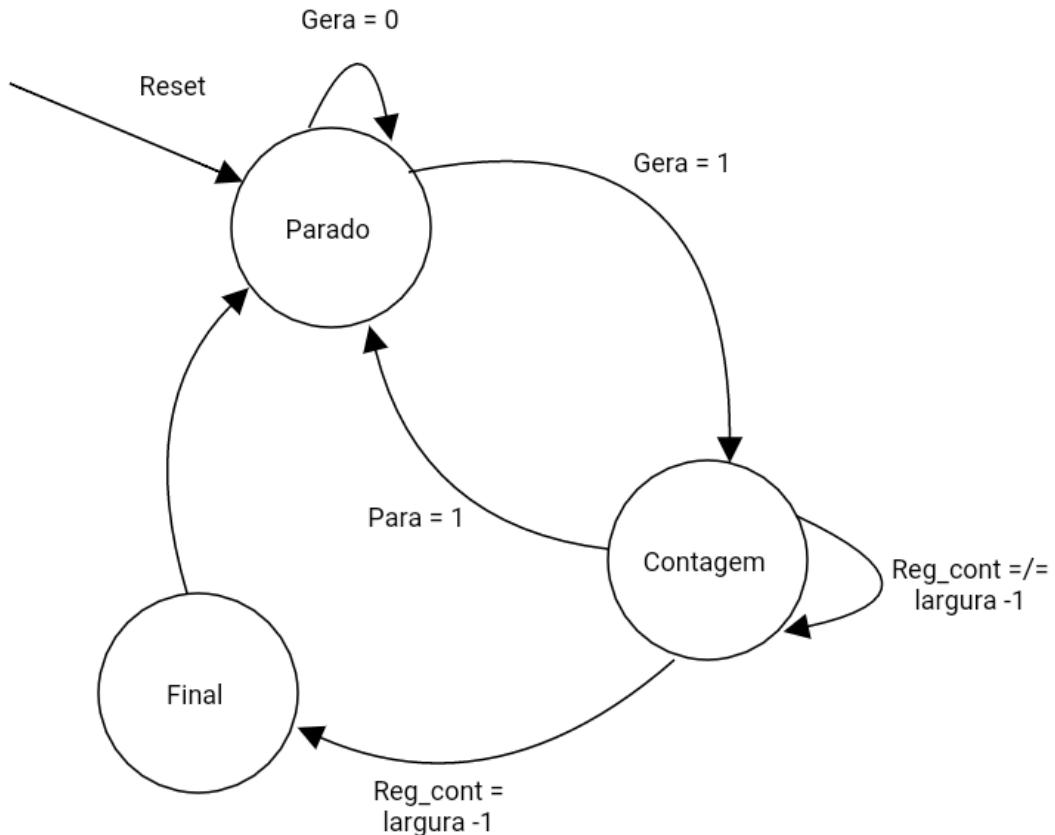


Figura 3: Máquina de estados do gerador de pulso.

Neste circuito, não utilizaremos o sinal *para* (isto é, ficará sempre em baixa), já que sempre queremos que o pulso enviado seja o mesmo (não há a necessidade de interrupções).

Medidor de Largura:

Este componente é responsável por calcular a quantidade de tempo que o pulso recebido pelo periférico HCSR04 (o sinal "*echo*") ficou em alta. Para isso, utilizam-se contadores que iniciam a contagem quando o pulso "*echo*" fica em alta. Além disso, para possibilitar a contagem, utiliza-se também o "*clock*". Como nesse circuito estamos trabalhando com uma frequência de 50MHz, temos que a cada 1 ciclo de *clock* que o pulso *echo* fica em alta, tem-se um total de 20ns que o sinal ficou em alta.

Além disso, este componente também é responsável por traduzir esta informação temporal em uma espacial. Em outras palavras, o medidor de largura também converte

o tempo que o pulso *echo* fica em alta em uma distância em cm. Para isso, utiliza-se o seguinte cálculo:

$$D = \frac{T}{58,82} \quad (1)$$

Com D em cm e T em μs . No entanto, para facilitar esta conta, vamos utilizar a seguinte relação:

$$1cm \equiv 58,82\mu s \quad (2)$$

Isto é, a cada $58,82\mu s$ que o pulso estiver em alta equivale a uma distância de 1cm. Logo, sabendo que:

$$\frac{58,82\mu s}{20ns} = 2941 \quad (3)$$

Temos então que a cada 2941 ciclos de clock a $50MHz$ que o pulso ficou em alta, a distância do objeto incrementa em 1cm.

Com essa simplificação, podemos considerar o circuito como dois contadores acoplados:



Figura 4: Diagrama RTL do medidor.

Além disso, o componente "*contador_bcd_4digitos*" fornecido pelo docente transforma sinais hexadecimais em decimais, para facilitar ainda mais a visualização do sinal na saída.

Registrador:

Por fim, no nosso fluxo de dados apresentamos um registrador de 12 bits, para armazenar 3 sinais decimais. Para isso, precisa-se da entrada D do registrador para armazenar esse dado, de um *enable* para saber quando armazenar o dado, um *clear* para quando for necessário limpar o dado, e uma saída Q para apresentar o que foi armazenado.

Posteriormente, continuou-se a montagem do circuito pela construção da unidade de controle. Antes de começar o código desta unidade, montou-se um diagrama de estados:

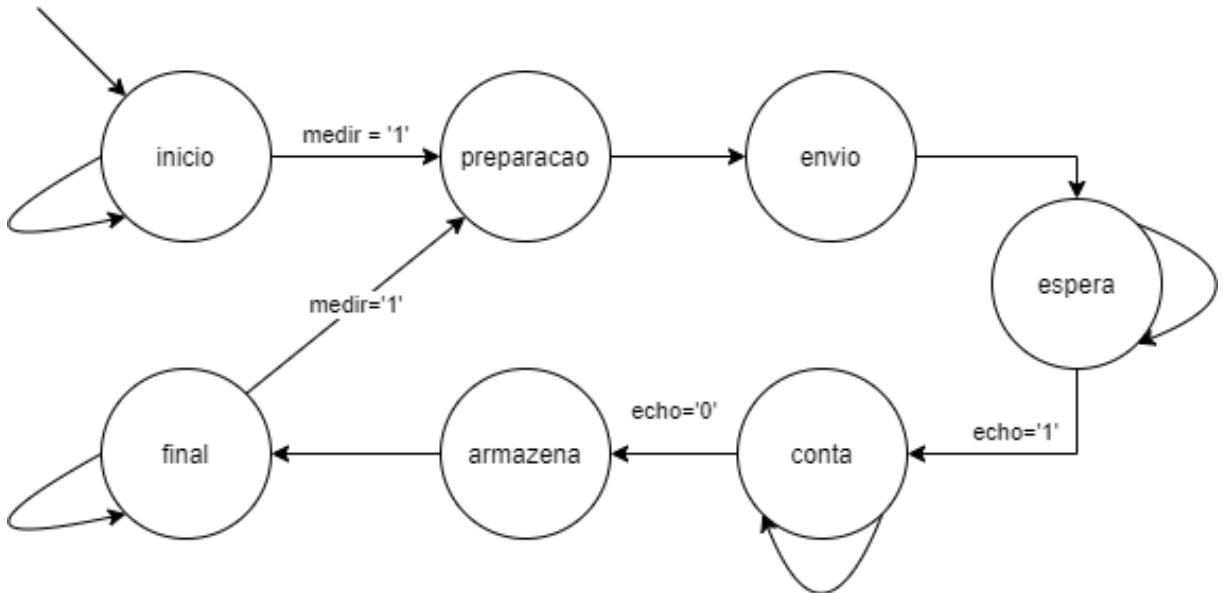


Figura 5: Máquina de estados da unidade de controle

- **inicio:** Durante este estado, a máquina aguarda o sinal de início de medição *medir*;
- **preparacao:** Neste estado, a máquina realiza as preparações necessárias para iniciar a medição (zera o circuito);
- **envio:** Quando a máquina chega neste ponto, a unidade de controle envia o sinal *gera* para o fluxo de dados, indicando para gerar os pulsos de **trigger**.
- **espera:** Durante este estado, a máquina espera o sinal de echo ficar em alta (indicando o início da contagem do pulso);
- **conta:** A unidade de controle fica neste estado enquanto o pulso *echo* estiver em alta, contando o tamanho do pulso;
- **armazena:** Após o fim do pulso *echo*, o resultado deve ser armazenado no registrador.
- **final:** A unidade de controle fica nesse estado no final de uma medição. Para medir mais um sinal, o circuito deve receber o sinal *medir = '1'*.

Assim, com a montagem da unidade de controle, o RTL Viewer da interface **HC-SR04**:

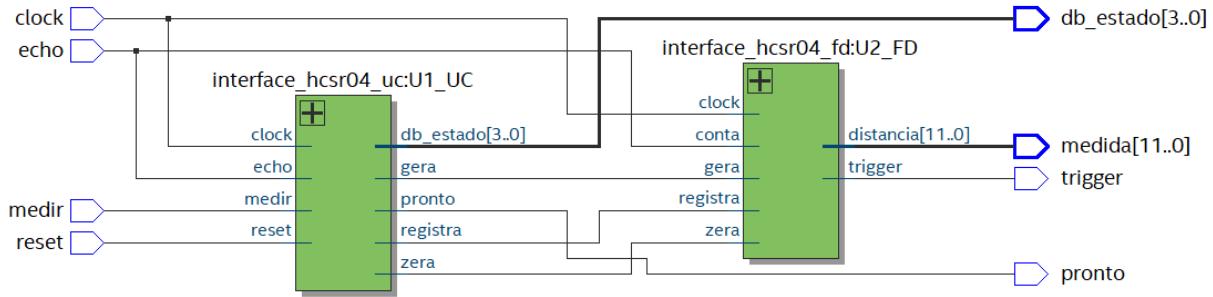


Figura 6: RTL Viewer da interface do circuito

Testagem dos componentes

Para a síntese desse circuito foi necessário o estudo dos componentes utilizados, para a realização dessa tarefa foram montados planos de testes.

Testes do gerador pulso

Para testar o gerador de pulso, vamos ter que testar os seguintes aspectos:

- Verificar se o pulso normal que deveria ser gerado funciona corretamente;
- Verificar o que ocorre no caso do sinal de geração não ser desligado no período do pulso ;
- Verificar o que ocorre caso o sinal de geração e o de parada fiquem ativos ao mesmo tempo.
- Verificar se o sinal de parada funciona como o esperado

1 - Pulso normal

- Acionar *gera* e desativar
- Usar um comando de assert para checar a saída *pronto* depois do tempo de 500 ciclos de clock esperado

2 - Pulso contínuo

- Acionar *gera*

- Esperar por 1000 ciclos de clock para visualização dos dois pulsos gerados

3 - Sinal de parada com sinal de geração

- Acionar *gera*
- Esperar 250 ciclos de clock
- Acionar *para*
- Esperar 250 ciclos de clock
- Desativar ambos os sinais *gera* e *para*

4 - Função de parada

- Acionar *gera* e desativar
- Esperar 250 ciclos de clock
- Acionar *para*
- Esperar 250 ciclos de clock
- Desativar *para*
- Esperar 100 ciclos de clock
- Usar um comando de assert para verificar a saída *pulso*

A saída da ferramenta multisim com todos os casos de teste foi a seguinte:

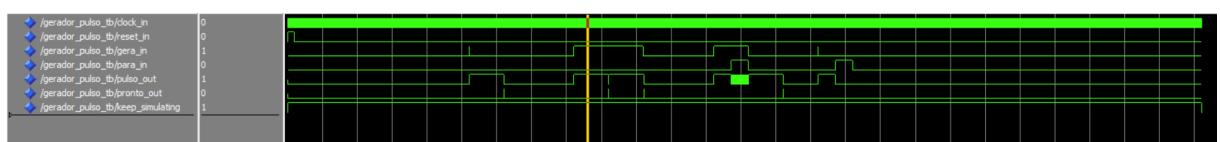


Figura 7: Forma de onda para os casos de testes do gerador de pulso

Pelas formas de onda pode-se visualizar que o pulso e o sinal de parada funcionam corretamente, em caso de ativação de ambos ao mesmo tempo o sinal de pulso oscila até a desativação de um dos dois e no caso do sinal *gera* em alta por tempo maior que o de 1 pulso outro pulso é gerado em sequência com uma baixa no sinal por 1 ciclo de clock.

Testagem do contador bcd de 4 dígitos

Para testar o contador, vamos ter que testar os seguintes aspectos:

- Se o contador atinge seu máximo e o que ocorre nesse caso
 - Teste da contagem sem ser até o final para ver se o valor corresponde ao esperado
-

1 - Teste da contagem

- Acionar *conta*
- Esperar por 9999 períodos de clock
- Desativar *conta*
- Ativar sinal *zera* e desativar para zerar a contagem

2 - Teste de contagem com parada

- Ativar sinal *conta*
- Esperar por 2941 períodos de clock
- Desativar *conta*
- Esperar 500 períodos de clock
- Ativar e desativar *zera*

A saída da ferramenta multisim com todos os casos de teste foi a seguinte:

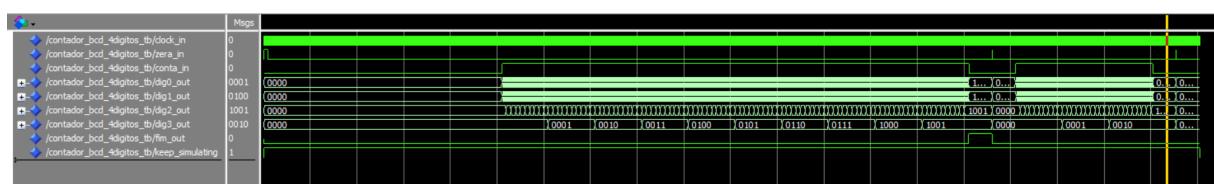


Figura 8: Forma de onda para os casos de testes do contador bcd

Pelas formas de onda pode-se perceber que o contador reinicia quando chega ao final de sua contagem (como esperado pela descrição vhdl) e conta corretamente até o fim do período especificado no *testbench*.

Testagem do medidor de largura

Para testar o medidor, considera-se os seguintes casos:

- Caso de medida de 1 cm
- Caso de pulso com valor não inteiro (4,08 cm)
- Caso de pulso com 6 cm
- Caso de pulso além da medida máxima do medidor (11000 cm)

1 - Teste de 1 cm

- Ativar *echo*
- Esperar 2941 ciclos de clock
- Desativar *echo*
- Ativar e desativar *zera*

2 - Teste de 4,08 cm

- Ativar *echo*
- Esperar 12000 ciclos de clock
- Desativar *echo*
- Ativar e desativar *zera*

3 - Teste de 6 cm

- Ativar *echo*
- Esperar 17646 ciclos de clock
- Desativar *echo*
- Ativar e desativar *zera*

4 - Teste de 11000 cm

- Ativar *echo*
- Esperar 32351000 ciclos de clock
- Desativar *echo*
- Ativar e desativar *zera*

Para esses testes foi necessário gerar duas formas de onda para facilitar a visualização:

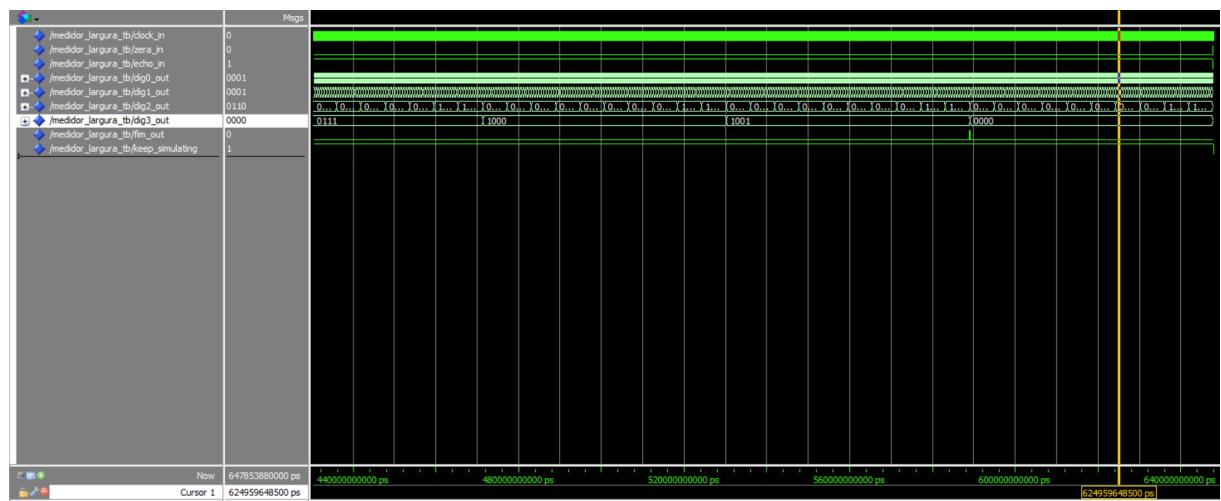


Figura 9: Forma de onda para o caso de teste de 11000cm

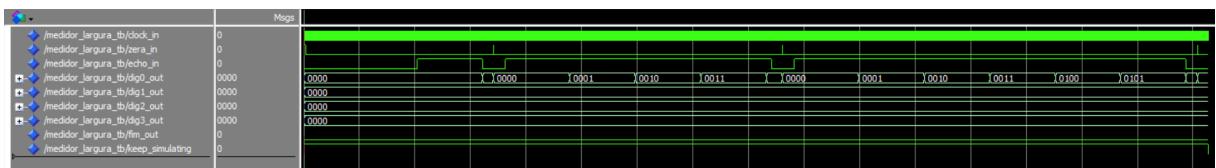


Figura 10: Forma de onda para os casos de testes do medidor

Pode-se perceber dos testes que os casos de valores inteiros e não inteiros funcionam (com arredondamento para baixo), além disso o medidor recomeça a contagem depois do seu máximo (como descrito no arquivo vhdl).

-
- b) Elabore um Plano de testes para verificar o funcionamento lógico do circuito e para a parte experimental, explicando a escolha dos casos de teste. DICA: planeje como os sinais do sensor HC-SR04 devem ser monitorados na montagem prática usando ferramentas do Analog Discovery (defina os sinais de depuração do circuito).**

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:

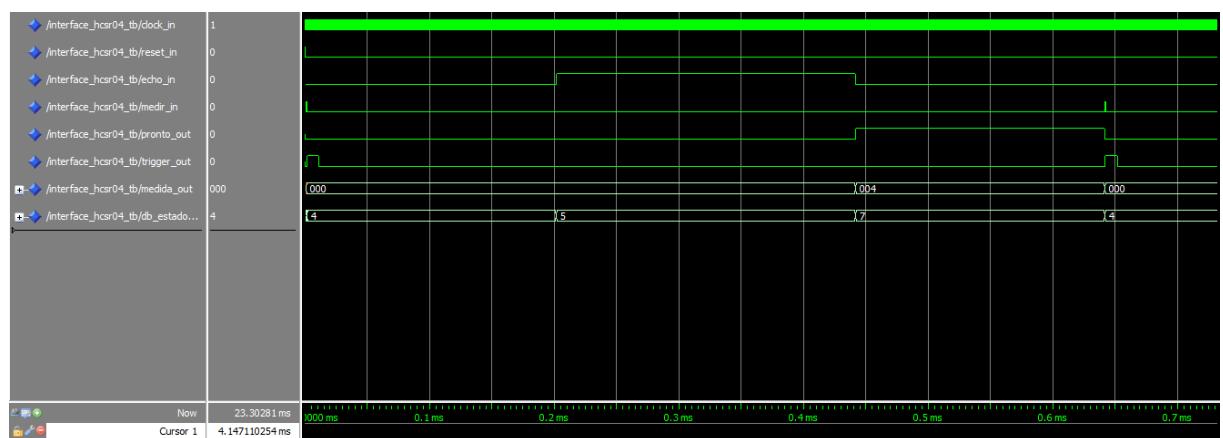


Figura 11: Forma de onda para o primeiro caso de teste

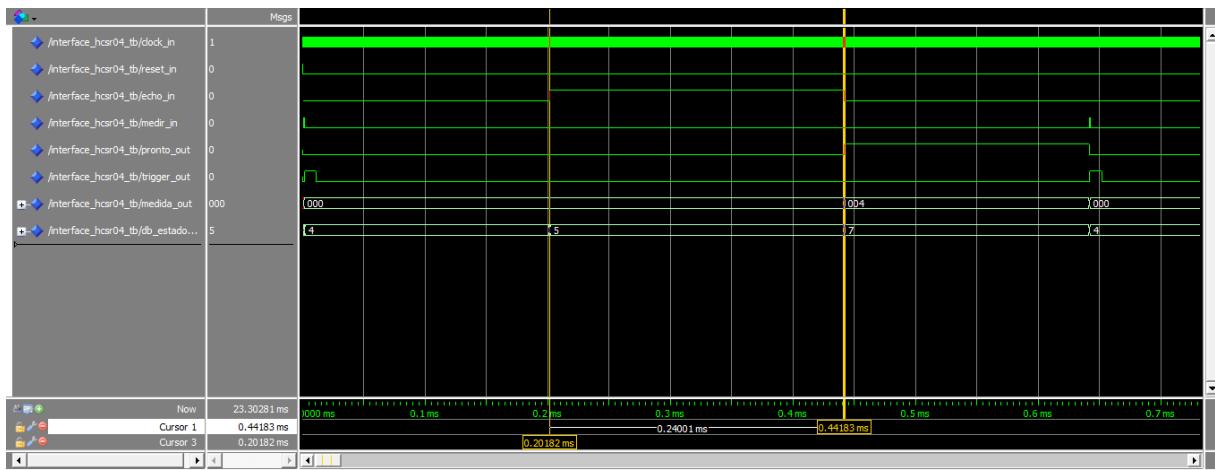


Figura 12: Largura do pulso "echo" para o primeiro teste

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:

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

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:

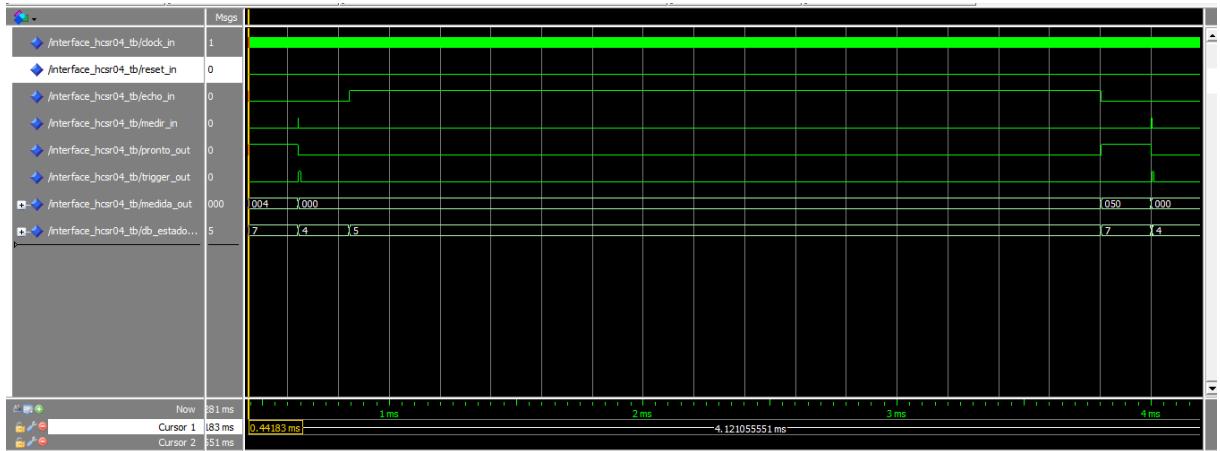


Figura 13: Forma de onda para o segundo caso de teste

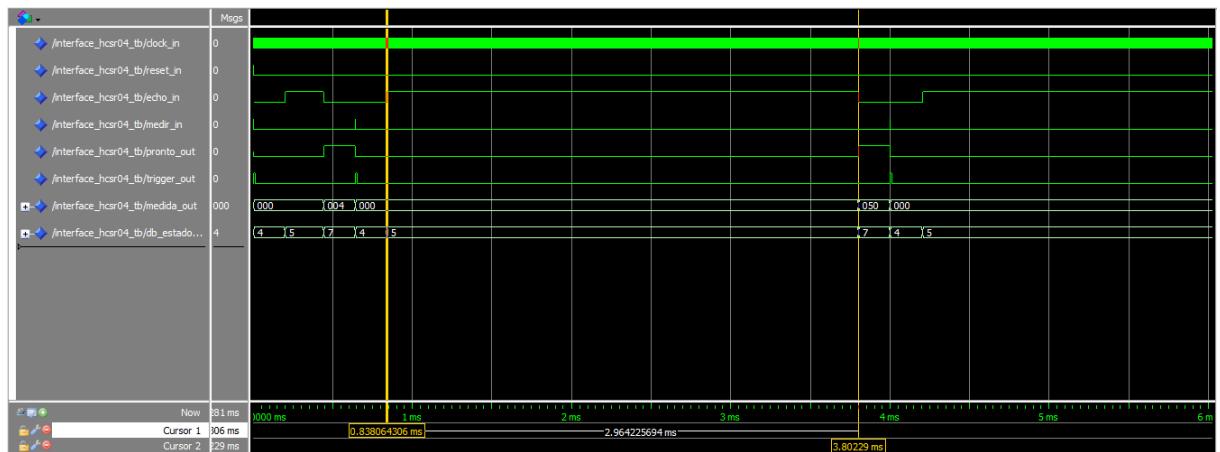


Figura 14: Largura do pulso "echo" para o segundo teste

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:

$$D = \frac{2964}{58,82} = 50,391 \text{ cm} \quad (5)$$

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:

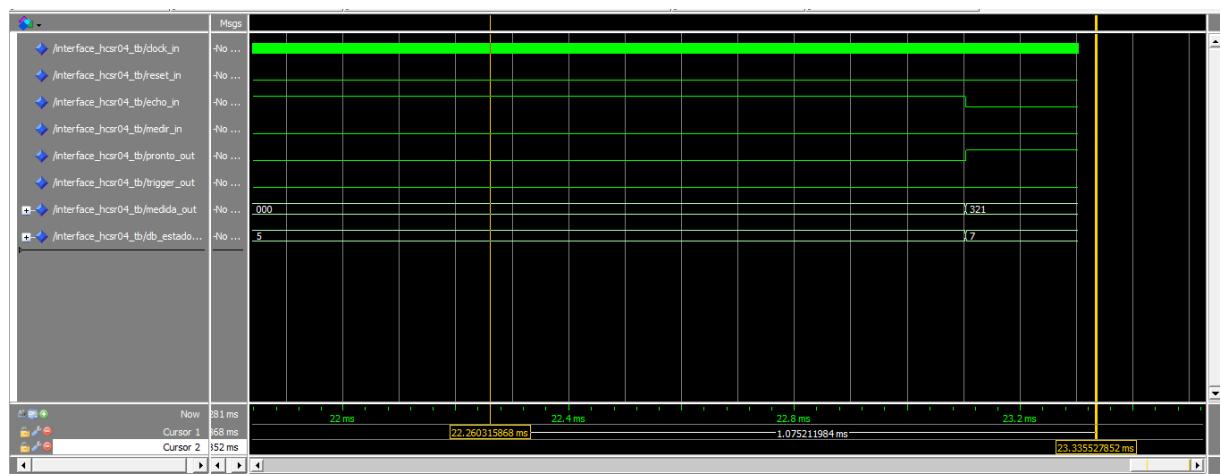


Figura 15: Forma de onda para o terceiro caso de teste

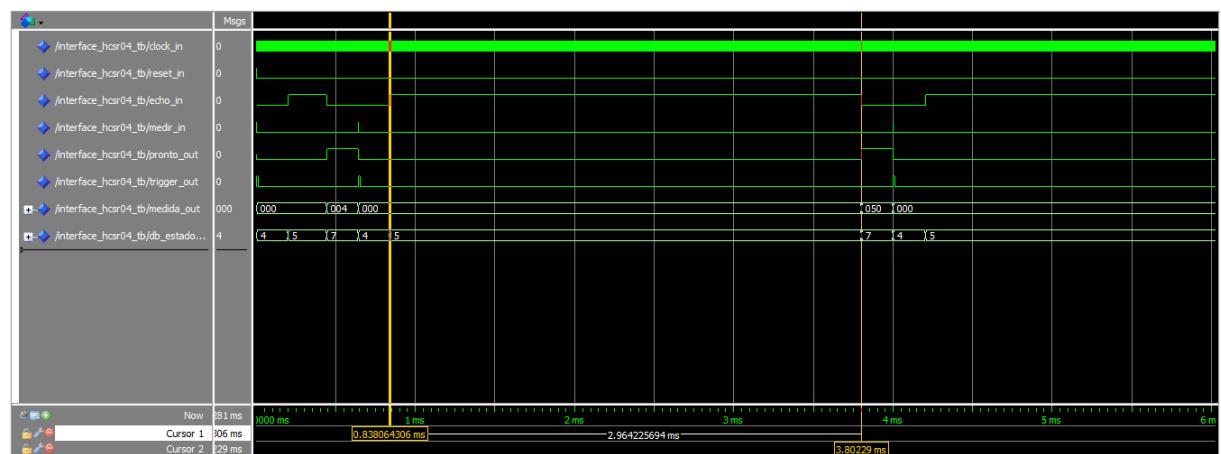


Figura 16: Largura do pulso "echo" para o terceiro teste

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:

$$D = \frac{18894}{58,82} = 321,21\text{cm} \quad (6)$$

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**:

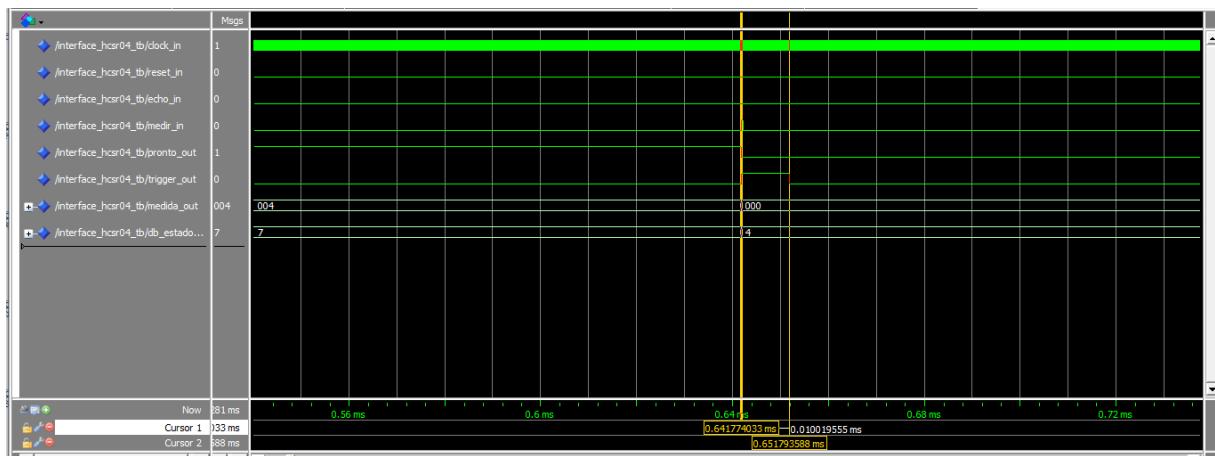


Figura 17: Forma de onda do pulso de trigger

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

-
- c) Use o software ModelSim para verificar o funcionamento do circuito de interface. Será disponibilizado um arquivo VHDL de um testbench básico com a estrutura geral para os testes. Recomenda-se que o grupo melhore os testes realizados pelo testbench para incluir os casos de teste definidos no item 2.1.b.
-
- d) Acrescente no Planejamento as formas de onda das simulações do sistema digital e de suas partes
-
- e) Submeta testbench desenvolvido pelo grupo (*interface_hcsr04_tb.vhd*) junto com o Planejamento
-

3.2 Atividade 2

f) Para a execução dos testes de funcionamento do circuito de interface com o sensor ultrassônico de distância em um ambiente remoto é necessário definir uma estratégia adequada. Para esta experiência, a montagem experimental na bancada remota do Laboratório Digital é composta por um módulo HC-SR04 e um conjunto de objetos dispostos a distâncias pré-estabelecidas. A figura 9 ilustra uma possível montagem física.

Para o circuito montado, precisamos testar dois aspectos do circuito: **precisão** e **estabilidade**.

Para testar a **precisão**, será testado 3 diferentes distâncias exatas para avaliar se o sensor está devolvendo medidas corretas.

Para testar **estabilidade**, cada teste de distância ocorrerá 6 vezes, e será calculado o desvio padrão para cada distância. Quanto menor o desvio, melhor a estabilidade.

Assim, no total, teremos 18 medições (6 medições para cada uma das três distâncias).

As distâncias escolhidas para a testagem foram:

- **5 centímetros de distância do sensor;**
- **50 centímetros de distância do sensor;**
- **200 centímetros de distância do sensor;**

Caso o último caso seja muito distante e não caiba no laboratório digital, alterar para 100 centímetros (ou qualquer distância menor possível).

3.3 Atividade 3

g) O projeto a ser sintetizado na placa FPGA deverá incluir o circuito de interface com o sensor ultrassônico de distância e mais alguns componentes. A entidade VHDL do circuito deverá ser chamado `exp4_sensor` e deve seguir a seguinte interface mínima de sinal.

h) Na implementação do projeto do circuito de teste da interface com o sensor ultrassônico de distância na placa FPGA, o sinal medida deve ser apresentado em 3 displays de 7 segmentos. Isto deve ser implementado através da adição de codificadores para displays de 7 segmentos (entidade hex7seg) que fazem a conversão de sinais binários para o código de 7 segmentos. A entrada medir também pode ser processada por um circuito detector de borda para transformá-la em um pulso com 1 período de clock de duração. A figura 10 ilustra um diagrama de blocos do circuito.

Seguiu-se o estabelecido para a montagem do circuito *exp4_sensor* base e chegou-se no seguinte diagrama de blocos:

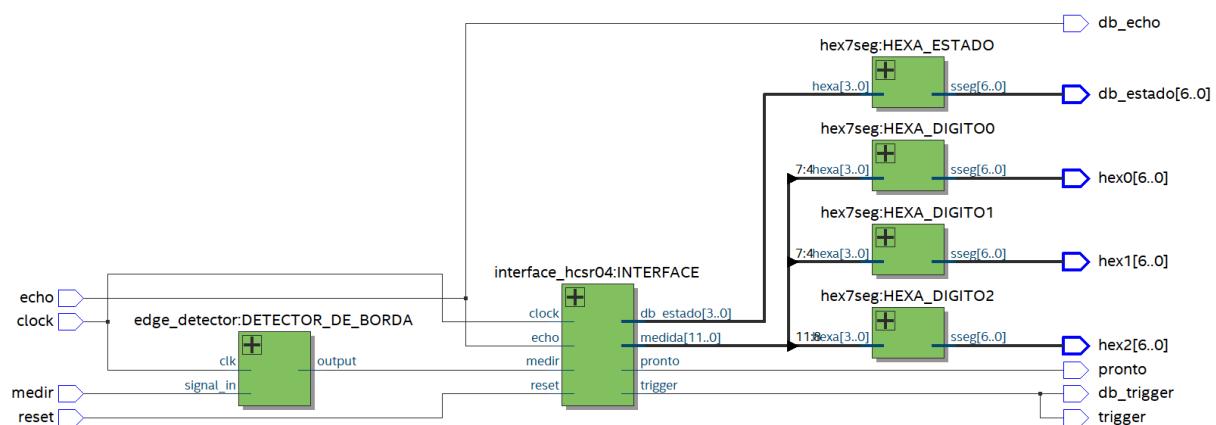


Figura 18: RTL do circuito *exp4_sensor* base

i) Defina um conjunto de sinais de depuração que podem ser mostrados em leds e displays disponíveis. O Planejamento deve conter uma tabela com a designação destes sinais adicionais. Observe que alguns sinais de depuração já foram definidos na tabela acima e devem ser observados.

Considerou-se os sinais *db_estado*, *db_echo* e *db_trigger* como suficientes para aferir o comportamento do circuito.

A partir disso a pinagem seguiu o seguinte formato:

sinal	pino	Analog Discovery	MQTT Dash
CLOCK	CLK_50	-	-
RESET	GPIO_0_D0	-	E0
MEDIR	GPIO_0_D1	-	E1
ECHO	GPIO_0_D34	-	-
TRIGGER	GPIO_1_D34	-	-
HEX0 a HEX2	displays HEX0 a HEX2	-	-
PRONTO	led LEDR[0]	-	-
ESTADO	display HEX5	-	-
db_trigger	-	CH1+ (Scope)	-
db_echo	-	CH2+ (Scope)	-

Figura 19: Tabela de pinagem

No Quartus a pinagem ficou efetivamente da seguinte forma:

in clock	Input	PIN_M9	3B	B3B_NO	PIN_M9	2.5 V		12mA ...ault	
out db_echo	Output	PIN_F12	7A	B7A_NO	PIN_F12	2.5 V		12mA ...ault	1 (default)
out db_estado[6]	Output	PIN_W19	4A	B4A_NO	PIN_W19	2.5 V		12mA ...ault	1 (default)
out db_estado[5]	Output	PIN_C2	2A	B2A_NO	PIN_C2	2.5 V		12mA ...ault	1 (default)
out db_estado[4]	Output	PIN_C1	2A	B2A_NO	PIN_C1	2.5 V		12mA ...ault	1 (default)
out db_estado[3]	Output	PIN_P14	4A	B4A_NO	PIN_P14	2.5 V		12mA ...ault	1 (default)
out db_estado[2]	Output	PIN_T14	4A	B4A_NO	PIN_T14	2.5 V		12mA ...ault	1 (default)
out db_estado[1]	Output	PIN_M8	3B	B3B_NO	PIN_M8	2.5 V		12mA ...ault	1 (default)
out db_estado[0]	Output	PIN_N9	3B	B3B_NO	PIN_N9	2.5 V		12mA ...ault	1 (default)
out db_trigger	Output	PIN_F15	7A	B7A_NO	PIN_F15	2.5 V		12mA ...ault	1 (default)
in echo	Input	PIN_T17	5A	B5A_NO	PIN_T17	2.5 V		12mA ...ault	
out hex0[6]	Output	PIN_AA22	4A	B4A_NO	PIN_AA22	2.5 V		12mA ...ault	1 (default)
out hex0[5]	Output	PIN_Y21	4A	B4A_NO	PIN_Y21	2.5 V		12mA ...ault	1 (default)
out hex0[4]	Output	PIN_Y22	4A	B4A_NO	PIN_Y22	2.5 V		12mA ...ault	1 (default)
out hex0[3]	Output	PIN_W21	4A	B4A_NO	PIN_W21	2.5 V		12mA ...ault	1 (default)
out hex0[2]	Output	PIN_W22	4A	B4A_NO	PIN_W22	2.5 V		12mA ...ault	1 (default)
out hex0[1]	Output	PIN_V21	4A	B4A_NO	PIN_V21	2.5 V		12mA ...ault	1 (default)
out hex0[0]	Output	PIN_U21	4A	B4A_NO	PIN_U21	2.5 V		12mA ...ault	1 (default)
out hex1[6]	Output	PIN_U22	4A	B4A_NO	PIN_U22	2.5 V		12mA ...ault	1 (default)
out hex1[5]	Output	PIN_AA17	4A	B4A_NO	PIN_AA17	2.5 V		12mA ...ault	1 (default)
out hex1[4]	Output	PIN_AB18	4A	B4A_NO	PIN_AB18	2.5 V		12mA ...ault	1 (default)
out hex1[3]	Output	PIN_AA18	4A	B4A_NO	PIN_AA18	2.5 V		12mA ...ault	1 (default)
out hex1[2]	Output	PIN_AA19	4A	B4A_NO	PIN_AA19	2.5 V		12mA ...ault	1 (default)
out hex1[1]	Output	PIN_AB20	4A	B4A_NO	PIN_AB20	2.5 V		12mA ...ault	1 (default)
out hex1[0]	Output	PIN_AA20	4A	B4A_NO	PIN_AA20	2.5 V		12mA ...ault	1 (default)
out hex2[6]	Output	PIN_AB21	4A	B4A_NO	PIN_AB21	2.5 V		12mA ...ault	1 (default)
out hex2[5]	Output	PIN_AB22	4A	B4A_NO	PIN_AB22	2.5 V		12mA ...ault	1 (default)
out hex2[4]	Output	PIN_V14	4A	B4A_NO	PIN_V14	2.5 V		12mA ...ault	1 (default)
out hex2[3]	Output	PIN_Y14	4A	B4A_NO	PIN_Y14	2.5 V		12mA ...ault	1 (default)
out hex2[2]	Output	PIN_AA10	3B	B3B_NO	PIN_AA10	2.5 V		12mA ...ault	1 (default)
out hex2[1]	Output	PIN_AB17	4A	B4A_NO	PIN_AB17	2.5 V		12mA ...ault	1 (default)
out hex2[0]	Output	PIN_Y19	4A	B4A_NO	PIN_Y19	2.5 V		12mA ...ault	1 (default)
in medir	Input	PIN_B16	7A	B7A_NO	PIN_B16	2.5 V		12mA ...ault	
in pronto	Output	PIN_AA2	2A	B2A_NO	PIN_AA2	2.5 V		12mA ...ault	1 (default)
in reset	Input	PIN_N16	5B	B5B_NO	PIN_N16	2.5 V		12mA ...ault	
out trigger	Output	PIN_J17	7A	B7A_NO	PIN_J17	2.5 V		12mA ...ault	1 (default)

Figura 20: Tabela de pinagem

j) Submeta o arquivo PDF do Planejamento e o arquivo QAR (exp4_txbyy.qar) do projeto.

3.4 Atividade 4

k) O teste na placa FPGA deve usar o MQTT Dash para acionar os sinais de entrada do circuito e iniciar a medida de distância. Para isto prepare o projeto no MQTT Dash com os seguintes widgets apresentados na figura 11.

A seguir, está o layout utilizado para realizar as testagens do circuito com a plataforma MQTT Dash:

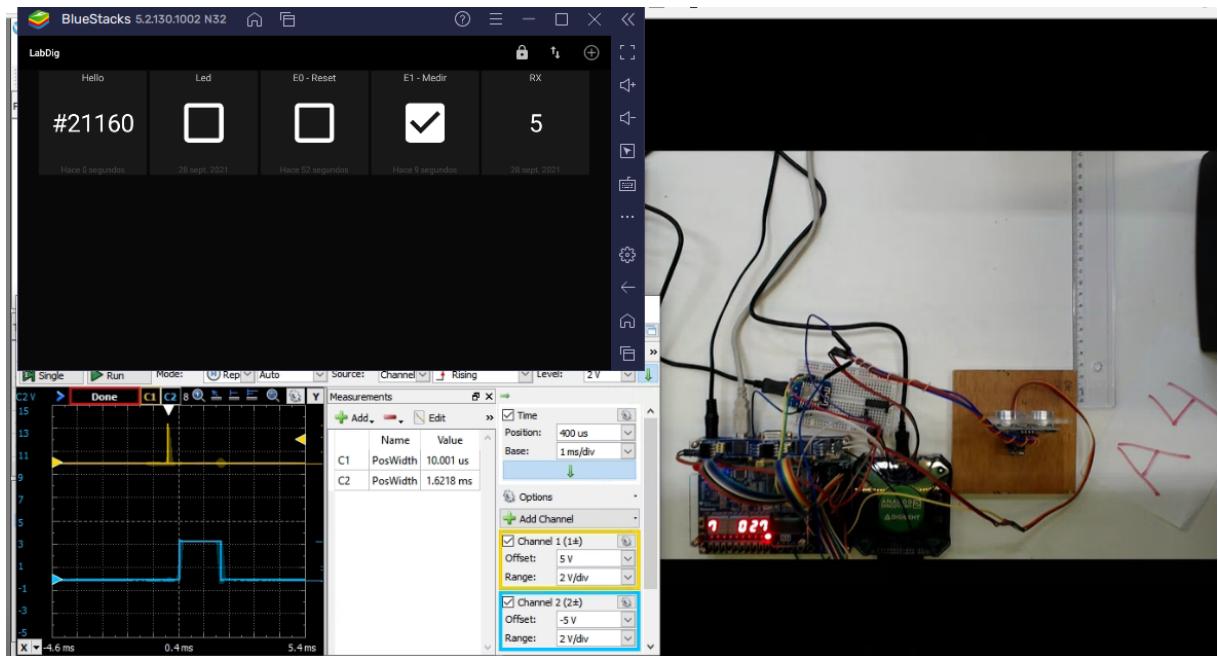


Figura 21: Montagem MQTT para testagem do circuito

Na imagem, é possível ver também a ferramenta *Scope* utilizada para analisar o formato de onda dos pulsos *trigger* e *echo*. Confirma-se que o *trigger* tem a largura de pulso correta de 10 microssegundos.

l) Programe o projeto sintetizado na placa DE0-CV no Laboratório Digital e teste o circuito. Descreva quaisquer ocorrências experimentais no Relatório.

O circuito funciona de acordo com o esperado, tendo algumas flutuações nas medidas ocasionalmente mas isso também fazia parte das expectativas considerando que o sensor

não tem uma exatidão na casa dos milímetros, apesar disso o sistema de arredondamento para baixo compensou a imprecisão do componente. Foram consideradas também as limitações do laboratório na escolha das medidas de teste e tabelamento.

m) Durante as atividades práticas, execute algumas medidas de sinais capturados do HC-SR04 (em função da distância e da posição do objeto de interesse) mapeando os sinais de depuração para os pinos da GPIO_1 da placa FPGA ligados no osciloscópio do Analog Discovery.

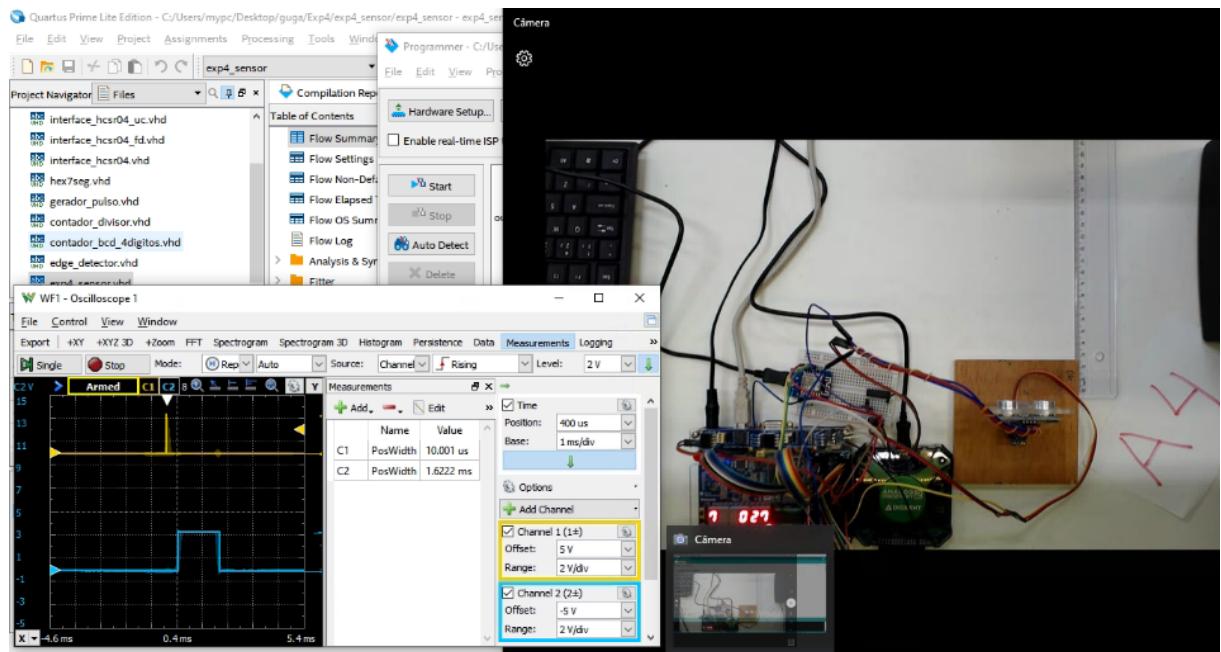


Figura 22: Tabela das medidas

Como pode ser observado na figura acima os sinais de depuração geraram formas de onda que correspondem ao esperado, com o pulso trigger de $10 \mu\text{s}$ e o echo correspondente a distância determinada.

n) Repita medidas para cada posição. Verifique a precisão e estabilidade das medidas realizadas. Se necessário, faça ajustes nas configurações do circuito. Reporte os resultados obtidos no Relatório.

	Posição 1 (5 cm)			Posição 2 (16,7 cm)			Posição 3 (27 cm)		
	Largura do Pulso	Calculado	Medido	Largura do Pulso	Calculado	Medido	Largura do Pulso	Calculado	Medido
1	0,33588	5,710302618	5	1,01	17,17103026	17	1,6224	27,58245495	27
2	0,33566	5,706562394	5	1,0103	17,17613057	17	1,6222	27,57905474	27
3	0,33589	5,710472628	5	1,01	17,17103026	17	1,6223	27,58075485	27
4	0,3359	5,710642639	5	1,0101	17,17273036	17	1,6223	27,58075485	27
5	0,33591	5,710812649	5	1,0102	17,17443047	17	1,622	27,57565454	27
6	0,33568	5,706902414	5	1,0102	17,17443047	17	1,6223	27,58075485	27
7	0,33583	5,709452567	5	1,0101	17,17273036	17	1,6224	27,58245495	27
8	0,3355	5,703842231	5	1,0101	17,17273036	17	1,6489	28,03298198	28
9	0,33578	5,708602516	5	1,0104	17,17783067	17	1,6223	27,58075485	27
10	0,33571	5,707412445	5	1,0102	17,17443047	17	1,6223	27,58075485	27

Figura 23: Tabela das medidas

Os resultados obtidos foram satisfatórios: embora as posições definidas no planejamento de teste foram diferentes das medidas, isso decorre de um erro humano do posicionamento dos objetos com a régua.

Fora isso, é notável que a precisão do sensor é suficientemente boa, dado que a medida sempre foi bem próxima da posição colocada (para 5 centímetros, a medida foi de 5, para 16,7 foi de 17, para 27 foi de 27).

Por fim, conclui-se também que a estabilidade do sistema é alta, considerando que as medidas tiveram pouquíssimo desvio - com exceção da oitava medida na terceira posição, que mediu 28 enquanto todas as outras medidas na mesma posição mediram 27. Esse desvio pode ser decorrente de vários fatores - uma interferência no circuito, uma interferência no sinal ultrassônico. No entanto, como se trata de um caso à parte (**outlier**) podemos desconsiderá-lo – sem falar também que foi um desvio razoavelmente pequeno (de somente 1 centímetro).

-
- o) Submeta o arquivo PDF (**PCS3645-TurmaX-BancadaYY-Relatorio4.pdf**), o arquivo **QAR** do projeto testado no Laboratório Digital (**exp4_final_txbyy.qar**), junto com o arquivo **VHDL** do testbench final do circuito.

4 RESULTADOS

Ao longo deste experimento, desenvolveram-se testes e *testbenches* para diversos componentes do circuito assim como o circuito como um todo. Geraram-se ondas no software ModelSim e a depuração do sistema permitiu eliminar erros quando foram encontrados.

Sobre o desenvolvimento de um circuito de controle do sensor HC-SR04, determina-se que foi um sucesso: a montagem do circuito por partes ao longo de testes parciais foram essenciais para identificar erros de maneira pontual e incisiva. Isso permitiu uma montagem de sucesso e uma implementação sem problemas do circuito durante o laboratório.

Durante a aula, não houveram grandes mudanças no circuito – o momento foi mais direcionado ao uso das ferramentas do Waveforms para verificar se o funcionamento estava de acordo com o que foi projetado, o que foi confirmado (como foi apresentado nas capturas de tela).

Este desenvolvimento será reaproveitado nos próximos experimentos, para a montagem do sonar.

5 DESAFIO

5.1 Descrição Geral

A modificação a ser implementada foi a adição do controle pwm do servo motor para que se realizasse medidas em diferentes posições e distâncias. A montagem do circuito do desafio ficou da seguinte forma:

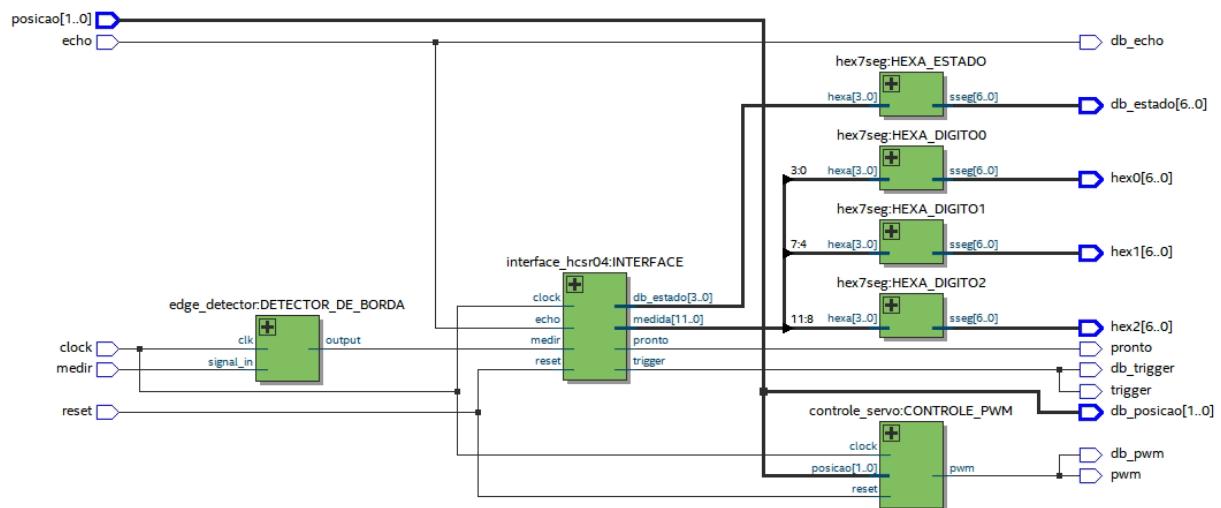


Figura 24: Diagrama de blocos do desafio

Como pode ser visto no diagrama RTL, também foram adicionadas saídas de depuração úteis para a análise do funcionamento do servomotor: *db_pwm* e *db_posicao*. A primeira foi alocada na ferramenta *Scope*, para analisar o tamanho do pulso em alta da PWM, e a segunda foi alocada nos dois LEDs, ao lado do LED do sinal de *pronto*. Assim, é possível depurar o circuito.

A testagem das ondas de PWM podem ser visualizadas no relatório do Experimento 1, onde foi desenvolvido e apropriadamente testado.

Além disso, adicionaram-se os sinais de posicionamento do servomotor no MQTT Dash, para poder controlá-lo por meio do aplicativo:

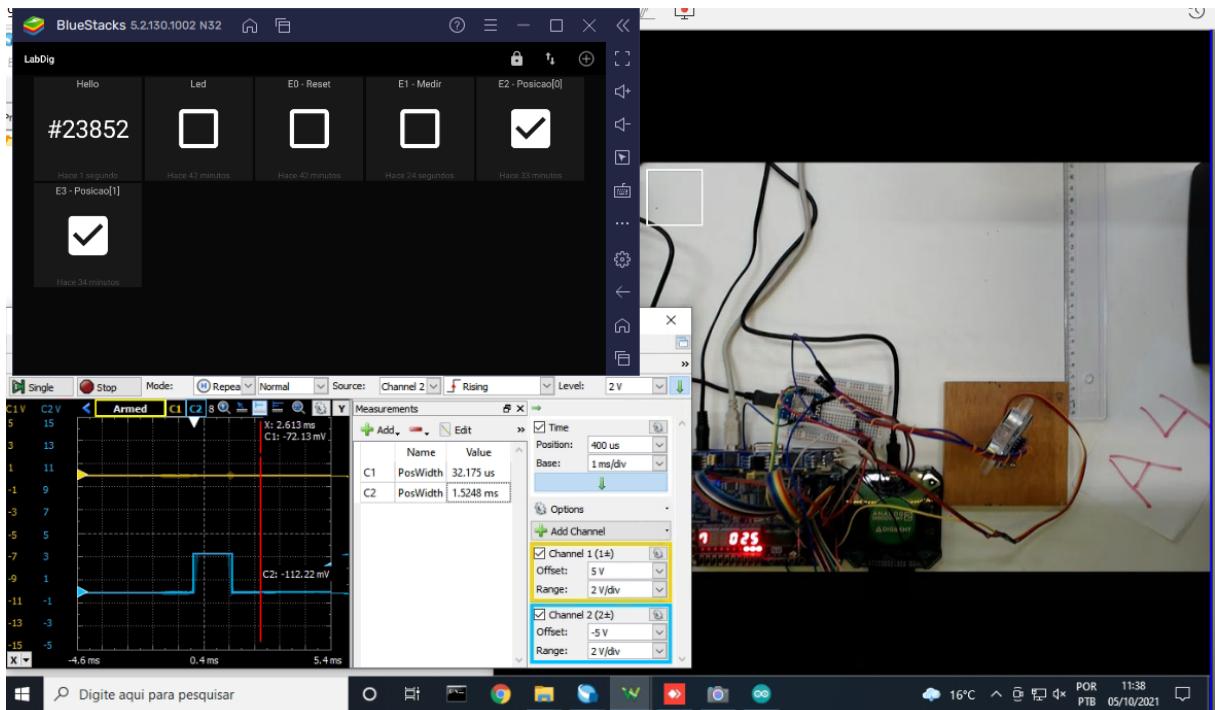


Figura 25: Montagem do MQTT Dash para o desafio

Terminada a montagem partiu-se então para a testagem do circuito nas diferentes posições. Suas configurações podem ser visualizadas nas seguintes imagens:

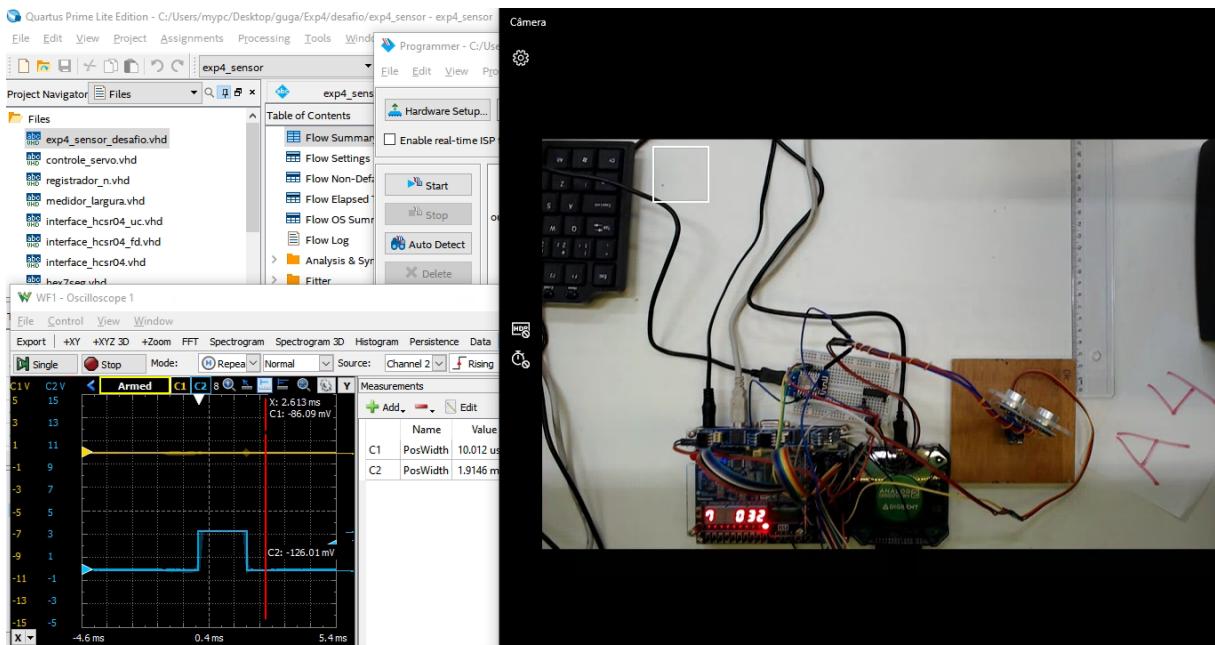


Figura 26: Servo na posição mínima

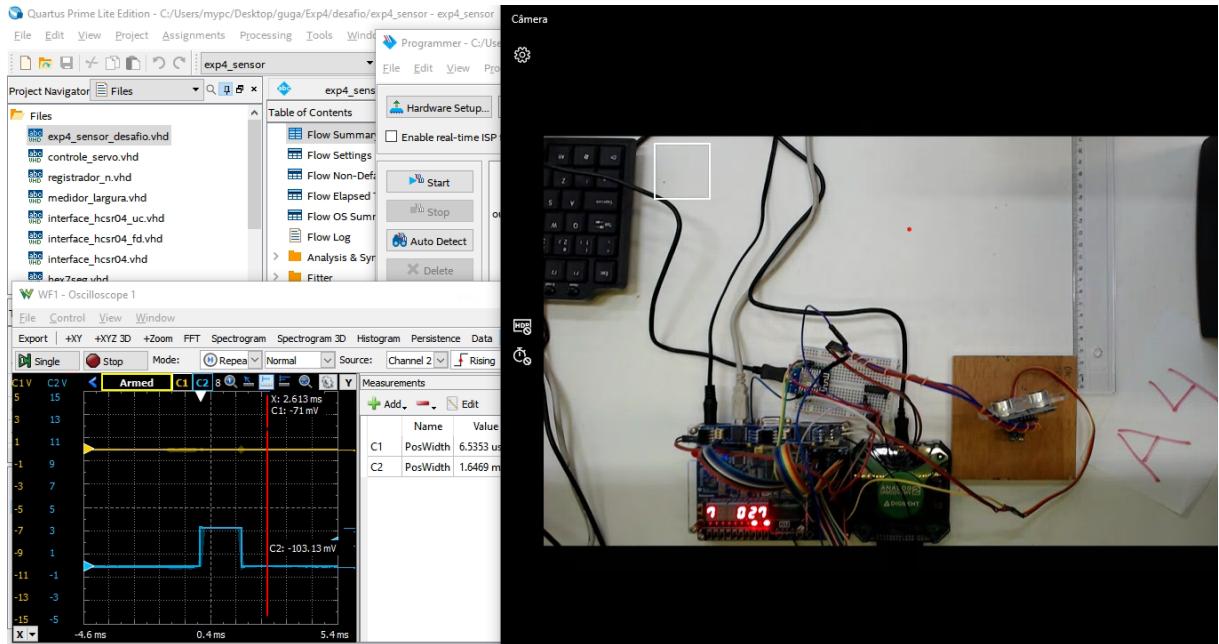


Figura 27: Servo na posição média

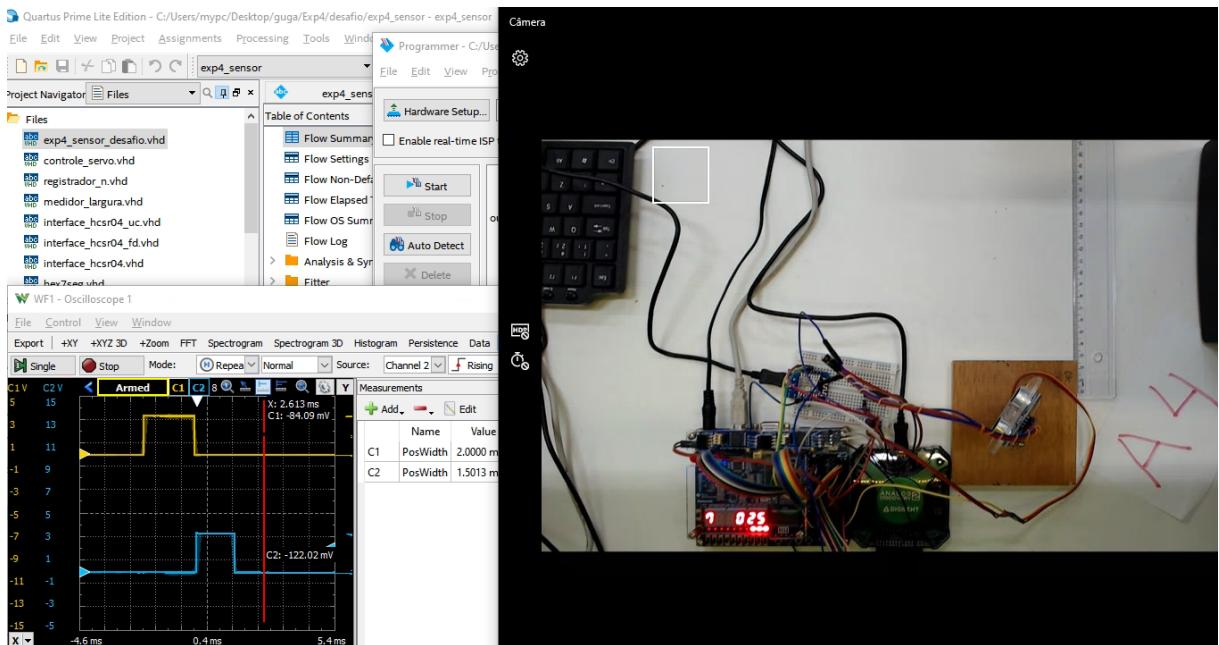


Figura 28: Servo na posição máxima

Como pode ser visto, as medidas variavam de acordo com a posição do servomotor, indicando que o circuito integrado estava com um funcionamento adequado.

6 APÊNDICE

6.1 Circuito Final

6.1.1 exp4_sensor

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use IEEE.math_real.all;
5
6 entity exp4_sensor is
7   port (
8     clock: in std_logic;
9     reset: in std_logic;
10    medir: in std_logic;
11    echo: in std_logic;
12    trigger: out std_logic;
13    hex0: out std_logic_vector(6 downto 0);
14    hex1: out std_logic_vector(6 downto 0);
15    hex2: out std_logic_vector(6 downto 0);
16    pronto: out std_logic;
17    db_trigger: out std_logic;
18    db_echo: out std_logic;
19    db_estado: out std_logic_vector(6 downto 0)
20  );
21 end entity exp4_sensor;
22
23 architecture arch of exp4_sensor is
24
25 component interface_hcsr04
26   port (
27     clock : in std_logic;
28     reset : in std_logic;
29     echo : in std_logic;
30     medir : in std_logic;
31     pronto : out std_logic;
32     trigger : out std_logic;
33     medida : out std_logic_vector(11 downto 0);
34     db_estado : out std_logic_vector(3 downto 0)
35   );
36 end component;
37
38
39 component edge_detector
40   port ( clk : in std_logic;
41           signal_in : in std_logic;
42           output : out std_logic
43         );
44 end component;
45
46
47 component hex7seg
48   port (
49     hexa : in std_logic_vector(3 downto 0);
50     sseg : out std_logic_vector(6 downto 0)
51   );

```

```

52 end component;
53
54 signal s_db_estado, s_medida0, s_medida1, s_medida2: std_logic_vector(3 downto 0);
55 signal s_medida: std_logic_vector(11 downto 0);
56 signal s_medir, s_db_echo, s_trigger: std_logic;
57
58 begin
59
60 DETECTOR_DE_BORDA: edge_detector port map(
61   clock,
62   medir,
63   s_medir
64 );
65
66 INTERFACE: interface_hcsr04 port map(
67   clock,
68   reset,
69   echo,
70   s_medir,
71   pronto,
72   s_trigger,
73   s_medida,
74   s_db_estado
75 );
76
77 HEXA_ESTADO: hex7seg port map(
78   s_db_estado,
79   db_estado
80 );
81
82 HEXA_DIGITO0: hex7seg port map(
83   s_medida0,
84   hex0
85 );
86
87 HEXA_DIGITO1: hex7seg port map(
88   s_medida1,
89   hex1
90 );
91
92 HEXA_DIGITO2: hex7seg port map(
93   s_medida2,
94   hex2
95 );
96
97 s_medida0 <= s_medida (3 downto 0);
98 s_medida1 <= s_medida (7 downto 4);
99 s_medida2 <= s_medida (11 downto 8);
100 db_echo <= echo;
101 db_trigger <= s_trigger;
102 trigger <= s_trigger;
103
104
105
106
107
108 end arch;

```

6.2 Circuito Base

6.2.1 interface_hcsr04

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.numeric_std.ALL;
4 USE IEEE.math_real.ALL;
5
6 ENTITY interface_hcsr04 IS
7 PORT (
8     clock : IN STD.LOGIC;
9     reset : IN STD.LOGIC;
10    echo : IN STD.LOGIC;
11    medir : IN STD.LOGIC;
12    pronto : OUT STD.LOGIC;
13    trigger : OUT STD.LOGIC;
14    medida : OUT STD.LOGIC_VECTOR(11 DOWNTO 0);
15    db_estado : OUT STD.LOGIC_VECTOR(3 DOWNTO 0)
16 );
17 END ENTITY;
18
19 ARCHITECTURE interface_hcsr04_arch OF interface_hcsr04 IS
20
21 COMPONENT interface_hcsr04_uc
22 PORT (
23     clock : IN STD.LOGIC;
24     reset : IN STD.LOGIC;
25     medir : IN STD.LOGIC;
26     echo : IN STD.LOGIC;
27     registra : OUT STD.LOGIC;
28     gera : OUT STD.LOGIC;
29     zera : OUT STD.LOGIC;
30     pronto : OUT STD.LOGIC;
31     db_estado : OUT STD.LOGIC_VECTOR(3 DOWNTO 0)
32 );
33 END COMPONENT;
34 COMPONENT interface_hcsr04_fd
35 PORT (
36     clock, conta, zera, registra, gera : IN STD.LOGIC;
37     distancia : OUT STD.LOGIC_VECTOR (11 DOWNTO 0);
38     trigger, fim : OUT STD.LOGIC
39 );
40 END COMPONENT;
41
42 SIGNAL s_pronto, s_zera, s_gera, s_registra : STD.LOGIC;
43 BEGIN
44 — unidade de controle
45 U1_UC : interface_hcsr04_uc PORT MAP(
46     clock,
47     reset,
48     medir,
49     echo,
50     s_registra,
51     s_gera,
52     s_zera,
53     pronto,
54     db_estado
55 );
56
57 — fluxo de dados
58 U2_FD : interface_hcsr04_fd PORT MAP(
59     clock,

```

```

60      echo ,
61      s_zera ,
62      s_registra ,
63      s_gera ,
64      medida ,
65      trigger ,
66      OPEN
67      );
68 END ARCHITECTURE;

```

6.2.2 interface_hcsr04_fd

```

1
2
3 LIBRARY ieee;
4 USE ieee.std_logic_1164.ALL;
5 USE ieee.numeric_std.ALL;
6 USE IEEE.math_real.ALL;
7
8 ENTITY interface_hcsr04_fd IS
9  PORT (
10    clock , conta , zera , registra , gera : IN STD.LOGIC;
11    distancia : OUT STD.LOGIC_VECTOR (11 DOWNTO 0);
12    trigger , fim : OUT STD.LOGIC
13  );
14 END ENTITY;
15
16 ARCHITECTURE arch OF interface_hcsr04_fd IS
17  COMPONENT medidor_largura
18    PORT (
19      clock , zera , echo : IN STD.LOGIC;
20      dig3 , dig2 , dig1 , dig0 : OUT STD.LOGIC_VECTOR(3 DOWNTO 0);
21      fim : OUT STD.LOGIC
22    );
23  END COMPONENT;
24
25  COMPONENT registrador_n
26    GENERIC (
27      CONSTANT N : INTEGER
28    );
29    PORT (
30      clock : IN STD.LOGIC;
31      clear : IN STD.LOGIC;
32      enable : IN STD.LOGIC;
33      D : IN STD.LOGIC_VECTOR (N - 1 DOWNTO 0);
34      Q : OUT STD.LOGIC_VECTOR (N - 1 DOWNTO 0)
35    );
36  END COMPONENT;
37
38  COMPONENT gerador_pulso
39    PORT (
40      clock : IN STD.LOGIC;
41      reset : IN STD.LOGIC;
42      gera : IN STD.LOGIC;
43      para : IN STD.LOGIC;
44      pulso : OUT STD.LOGIC;
45      pronto : OUT STD.LOGIC
46    );
47  END COMPONENT;
48  SIGNAL s_dig2 , s_dig1 , s_dig0 : STD.LOGIC_VECTOR(3 DOWNTO 0);
49  SIGNAL s_distancia : STD.LOGIC_VECTOR(11 DOWNTO 0);
50
51 BEGIN

```

```

52
53  CONTAGEM : medidor_largura
54  PORT MAP(
55      clock ,
56      zera ,
57      conta ,
58      OPEN,
59      s_dig2 ,
60      s_dig1 ,
61      s_dig0 ,
62      fim
63  );
64
65  REGISTRADOR : registrador_n GENERIC MAP(
66      N => 12) PORT MAP(
67      clock ,
68      zera ,
69      registra ,
70      s_distancia ,
71      distancia
72  );
73
74  GERADOR_TRIGGER : gerador_pulso
75  PORT MAP(
76      clock ,
77      zera ,
78      gera ,
79      '0',
80      trigger ,
81      OPEN
82  );
83
84  s_distancia <= s_dig2 & s_dig1 & s_dig0 ;
85 END ARCHITECTURE;

```

6.2.3 interface_hcsr04_uc

```

1
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4
5 ENTITY interface_hcsr04_uc IS
6     PORT (
7         clock : IN STD_LOGIC;
8         reset : IN STD_LOGIC;
9         medir : IN STD_LOGIC;
10        echo : IN STD_LOGIC;
11        registra : OUT STD_LOGIC;
12        gera : OUT STD_LOGIC;
13        zera : OUT STD_LOGIC;
14        pronto : OUT STD_LOGIC;
15        db_estado : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
16    );
17 END ENTITY;
18
19 ARCHITECTURE interface_hcsr04_uc_arch OF interface_hcsr04_uc IS
20
21     TYPE tipo_estado IS (
22         inicio ,
23         preparacao ,
24         envio ,
25         espera ,
26         conta ,

```

```

27      armazena ,
28      final
29  );
30
31  SIGNAL Eatual : tipo_estado; — estado atual
32  SIGNAL Eprox : tipo_estado; — proximo estado
33
34 BEGIN
35
36      — memoria de estado
37  PROCESS (reset , clock)
38  BEGIN
39      IF reset = '1' THEN
40          Eatual <= inicio ;
41      ELSIF clock'event AND clock = '1' THEN
42          Eatual <= Eprox ;
43      END IF ;
44  END PROCESS;
45
46      — logica de proximo estado
47  PROCESS (medir, echo, reset , Eatual)
48  BEGIN
49      CASE Eatual IS
50          WHEN inicio =>
51              IF medir = '1' THEN
52                  Eprox <= preparacao;
53              ELSE
54                  Eprox <= inicio ;
55              END IF ;
56
57          WHEN preparacao => Eprox <= envio ;
58
59          WHEN envio => Eprox <= espera ;
60
61          WHEN espera =>
62              IF (echo = '1') THEN
63                  Eprox <= conta;
64              ELSE
65                  Eprox <= espera ;
66              END IF ;
67
68          WHEN conta =>
69              IF (echo = '0') THEN
70                  Eprox <= armazena;
71              ELSE
72                  Eprox <= conta;
73              END IF ;
74
75          WHEN armazena => Eprox <= final ;
76
77          WHEN final =>
78              IF medir = '1' THEN
79                  Eprox <= preparacao;
80              ELSE
81                  Eprox <= final ;
82              END IF ;
83
84          WHEN OTHERS => Eprox <= inicio ;
85
86      END CASE;
87  END PROCESS;
88
89      — logica de saida (Moore)
90  WITH Eatual SELECT
91      gera <= '1' WHEN envio , '0' WHEN OTHERS;
92

```

```

93      WITH Eatual SELECT
94          registra <= '1' WHEN armazena, '0' WHEN OTHERS;
95
96      WITH Eatual SELECT
97          zera <= '1' WHEN preparacao, '0' WHEN OTHERS;
98
99      WITH Eatual SELECT
100         pronto <= '1' WHEN final, '0' WHEN OTHERS;
101
102     WITH Eatual SELECT
103         db_estado <= "0001" WHEN inicio,
104             "0010" WHEN preparacao,
105             "0011" WHEN envio,
106             "0100" WHEN espera,
107             "0101" WHEN conta,
108             "0110" WHEN armazena,
109             "0111" WHEN final,
110             "0001" WHEN OTHERS;
111
112 END interface_hcsr04_uc_arch;

```

6.2.4 medidor_largura

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.numeric_std.ALL;
4 USE IEEE.math_real.ALL;
5
6 ENTITY medidor_largura IS
7     PORT (
8         clock, zera, echo : IN STD_LOGIC;
9         dig3, dig2, dig1, dig0 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
10        fim : OUT STD_LOGIC
11    );
12 END medidor_largura;
13
14 ARCHITECTURE medidor_arch OF medidor_largura IS
15
16     COMPONENT contador_bcd_4digitos
17         PORT (
18             clock, zera, conta : IN STD_LOGIC;
19             dig3, dig2, dig1, dig0 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
20             fim : OUT STD_LOGIC
21         );
22     END COMPONENT;
23
24     COMPONENT contador_divisor
25         PORT (
26             clock, zera_as, zera_s, conta : IN STD_LOGIC;
27             Q : OUT STD_LOGIC_VECTOR (NATURAL(ceil(log2(real(2941)))) - 1 DOWNTO 0);
28             fim, meio : OUT STD_LOGIC
29         );
30     END COMPONENT;
31
32     SIGNAL dig3_final, dig2_final, dig1_final, dig0_final : STD_LOGIC_VECTOR(3 DOWNTO 0);
33     SIGNAL s_cm, s_fim : STD_LOGIC;
34
35 BEGIN
36
37     CONTADOR_INICIAL : contador_divisor PORT MAP(
38         clock,
39         zera,
40         zera,

```

```

41     echo ,
42     OPEN,
43     s_cm ,
44     OPEN
45   );
46
47 CONTADOR_FINAL : contador_bcd_4digitos PORT MAP(
48   clock ,
49   zera ,
50   s_cm ,
51   dig3_final ,
52   dig2_final ,
53   dig1_final ,
54   dig0_final ,
55   s_fim
56 );
57
58   fim <= s_fim;
59   dig3 <= dig3_final;
60   dig2 <= dig2_final;
61   dig1 <= dig1_final;
62   dig0 <= dig0_final;
63 END medidor_arch;

```

6.2.5 gerador_pulso

```

1 -----
2 — Arquivo   : gerador_pulso.vhd
3 — Projeto   : Experiencia 4 – Interface com sensor de distancia
4 -----
5 — Descricao : gera pulso de saida com largura pulsos de clock
6 —
7 —           1) descricao na forma de uma maquina de estados
8 —
9 -----
10 — Revisoes  :
11 —     Data      Versao   Autor           Descricao
12 —     19/09/2021  1.0     Edson Midorikawa  versao inicial
13 —
14 —
15
16 LIBRARY ieee;
17 USE ieee.std_logic_1164.ALL;
18 USE ieee.numeric_std.ALL;
19
20 ENTITY gerador_pulso IS
21   PORT (
22     clock : IN STD_LOGIC;
23     reset : IN STD_LOGIC;
24     gera : IN STD_LOGIC;
25     para : IN STD_LOGIC;
26     pulso : OUT STD_LOGIC;
27     pronto : OUT STD_LOGIC
28   );
29 END gerador_pulso;
30
31 ARCHITECTURE fsm_arch OF gerador_pulso IS
32
33   TYPE tipo_estado IS (parado, contagem, final);
34   SIGNAL reg_estado, prox_estado : tipo_estado;
35   SIGNAL reg_cont, prox_cont : INTEGER RANGE 0 TO 500 - 1;
36
37 BEGIN

```

```

38
39 — estado e contagem
40 PROCESS (clock, reset)
41 BEGIN
42   IF (reset = '1') THEN
43     reg_estado <= parado;
44     reg_cont <= 0;
45   ELSIF (clock'event AND clock = '1') THEN
46     reg_estado <= prox_estado;
47     reg_cont <= prox_cont;
48   END IF;
49 END PROCESS;
50
51 — logica de proximo estado e contagem
52 PROCESS (reg_estado, gera, para, reg_cont)
53 BEGIN
54   pulso <= '0';
55   pronto <= '0';
56   prox_cont <= reg_cont;
57
58   CASE reg_estado IS
59
60     WHEN parado =>
61       IF gera = '1' THEN
62         prox_estado <= contagem;
63       ELSE
64         prox_estado <= parado;
65       END IF;
66       prox_cont <= 0;
67
68     WHEN contagem =>
69       IF para = '1' THEN
70         prox_estado <= parado;
71       ELSE
72         IF (reg_cont = 500 - 1) THEN
73           prox_estado <= final;
74         ELSE
75           prox_estado <= contagem;
76           prox_cont <= reg_cont + 1;
77         END IF;
78       END IF;
79       pulso <= '1';
80
81     WHEN final =>
82       prox_estado <= parado;
83       pronto <= '1';
84   END CASE;
85 END PROCESS;
86
87 END fsm_arch;

```

6.2.6 contador_divisor

```

1
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.ALL;
4 USE ieee.numeric_std.ALL;
5 USE IEEE.math_real.ALL;
6
7 ENTITY contador_divisor IS
8
9   PORT (
10     clock, zera_as, zera_s, conta : IN STD.LOGIC;

```

```

11      Q : OUT STD_LOGIC_VECTOR (NATURAL(ceil(log2(real(2941)))) - 1 DOWNTO 0);
12      fim, meio : OUT STD_LOGIC
13  );
14 END ENTITY contador_divisor;
15
16 ARCHITECTURE comportamental OF contador_divisor IS
17   SIGNAL IQ : INTEGER RANGE 0 TO 2941 - 1;
18 BEGIN
19
20   PROCESS (clock, zera_as, zera_s, conta, IQ)
21   BEGIN
22     IF zera_as = '1' THEN
23       IQ <= 0;
24     ELSIF rising_edge(clock) THEN
25       IF zera_s = '1' THEN
26         IQ <= 0;
27       ELSIF conta = '1' THEN
28         IF IQ = 2941 - 1 THEN
29           IQ <= 0;
30         ELSE
31           IQ <= IQ + 1;
32         END IF;
33       ELSE
34         IQ <= IQ;
35       END IF;
36     END IF;
37
38     — fim de contagem
39     IF IQ = 2941 - 1 THEN
40       fim <= '1';
41     ELSE
42       fim <= '0';
43     END IF;
44
45     — meio da contagem
46     IF IQ = 2941/2 - 1 THEN
47       meio <= '1';
48     ELSE
49       meio <= '0';
50     END IF;
51
52     Q <= STD_LOGIC_VECTOR(to_unsigned(IQ, Q'length));
53
54   END PROCESS;
55
56 END comportamental;

```

6.2.7 contador_bcd_4digitos

```

1 —————
2 — Arquivo : contador_bcd_4digitos.vhd
3 — Projeto : Experiencia 4 – Interface com sensor de distancia
4 —————
5 — Descricao : contador bcd com 4 digitos (modulo 10.000)
6 —           descricao VHDL comportamental
7 —           1) reset sincrono
8 —           2) saida de fim de contagem para 9999
9 —————
10 — Revisoes :
11 —   Data      Versao  Autor          Descricao
12 —   19/09/2020 1.0    Edson Midorikawa versao inicial
13 —   26/09/2020 1.1    Edson Midorikawa revisao
14 —   19/09/2021 1.2    Edson Midorikawa revisao

```

```

15 -----
16 -
17
18 LIBRARY ieee;
19 USE ieee.std_logic_1164.ALL;
20 USE ieee.numeric_std.ALL;
21
22 ENTITY contador_bcd_4digitos IS
23     PORT (
24         clock, zera, conta : IN STD.LOGIC;
25         dig3, dig2, dig1, dig0 : OUT STD.LOGIC_VECTOR(3 DOWNTO 0);
26         fim : OUT STD.LOGIC
27     );
28 END contador_bcd_4digitos;
29
30 ARCHITECTURE comportamental OF contador_bcd_4digitos IS
31
32     SIGNAL s_dig3, s_dig2, s_dig1, s_dig0 : unsigned(3 DOWNTO 0);
33
34 BEGIN
35
36     PROCESS (clock)
37     BEGIN
38         IF (clock'event AND clock = '1') THEN
39             IF (zera = '1') THEN — reset sincrono
40                 s_dig0 <= "0000";
41                 s_dig1 <= "0000";
42                 s_dig2 <= "0000";
43                 s_dig3 <= "0000";
44             ELSIF (conta = '1') THEN
45                 IF (s_dig0 = "1001") THEN
46                     s_dig0 <= "0000";
47                     IF (s_dig1 = "1001") THEN
48                         s_dig1 <= "0000";
49                         IF (s_dig2 = "1001") THEN
50                             s_dig2 <= "0000";
51                             IF (s_dig3 = "1001") THEN
52                                 s_dig3 <= "0000";
53                             ELSE
54                                 s_dig3 <= s_dig3 + 1;
55                             END IF;
56                         ELSE
57                             s_dig2 <= s_dig2 + 1;
58                         END IF;
59                     ELSE
60                         s_dig1 <= s_dig1 + 1;
61                     END IF;
62                 ELSE
63                     s_dig0 <= s_dig0 + 1;
64                 END IF;
65             END IF;
66         END IF;
67     END PROCESS;
68
69     — fim de contagem (comando VHDL when else)
70     fim <= '1' WHEN s_dig3 = "1001" AND s_dig2 = "1001" AND
71         s_dig1 = "1001" AND s_dig0 = "1001" ELSE
72         '0';
73
74     — saidas
75     dig3 <= STD.LOGIC_VECTOR(s_dig3);
76     dig2 <= STD.LOGIC_VECTOR(s_dig2);
77     dig1 <= STD.LOGIC_VECTOR(s_dig1);
78     dig0 <= STD.LOGIC_VECTOR(s_dig0);
79
80 END comportamental;

```

6.2.8 registrador_n

```

1 ----- Laboratorio Digital -----
2 — Arquivo : registrador_n.vhd
3 — Projeto : Experiencia 3 - Recepcao Serial Assincrona
4
5 — Descricao : registrador com numero de bits N como generic
6 —           com clear assincrono e carga sincrona
7 —           baseado no codigo vreg16.vhd do livro
8 —           J. Wakerly, Digital design: principles and practices 4e
9
10 — Revisoes :
11 —     Data      Versao Autor      Descricao
12 —     09/09/2019 1.0   Edson Midorikawa  criacao
13 —     08/06/2020 1.1   Edson Midorikawa  revisao e melhoria de codigo
14 —     09/09/2020 1.2   Edson Midorikawa  revisao
15 —     09/09/2021 1.3   Edson Midorikawa  revisao
16
17 —
18
19 LIBRARY IEEE;
20 USE IEEE.std_logic_1164.ALL;
21
22 ENTITY registrador_n IS
23     GENERIC (
24         CONSTANT N : INTEGER := 8
25     );
26     PORT (
27         clock : IN STD.LOGIC;
28         clear : IN STD.LOGIC;
29         enable : IN STD.LOGIC;
30         D : IN STD.LOGIC_VECTOR (N - 1 DOWNTO 0);
31         Q : OUT STD.LOGIC_VECTOR (N - 1 DOWNTO 0)
32     );
33 END registrador_n;
34
35 ARCHITECTURE registrador_n OF registrador_n IS
36     SIGNAL IQ : STD.LOGIC_VECTOR(N - 1 DOWNTO 0);
37 BEGIN
38
39     PROCESS (clock, clear, enable, IQ)
40     BEGIN
41         IF (clear = '1') THEN
42             IQ <= (OTHERS => '0');
43         ELSIF (clock'event AND clock = '1') THEN
44             IF (enable = '1') THEN
45                 IQ <= D;
46             ELSE
47                 IQ <= IQ;
48             END IF;
49         END IF;
50         Q <= IQ;
51     END PROCESS;
52
53 END registrador_n;

```

6.3 Circuito do Desafio

6.3.1 exp4_sensor_desafio

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use IEEE.math_real.all;
5
6 entity exp4_sensor_desafio is
7 port (
8 clock: in std_logic;
9 reset: in std_logic;
10 medir: in std_logic;
11 echo: in std_logic;
12 posicao: in std_logic_vector(1 downto 0);
13 trigger: out std_logic;
14 hex0: out std_logic_vector(6 downto 0);
15 hex1: out std_logic_vector(6 downto 0);
16 hex2: out std_logic_vector(6 downto 0);
17 pronto: out std_logic;
18 pwm: out std_logic;
19 db_pwm: out std_logic;
20 db_trigger: out std_logic;
21 db_echo: out std_logic;
22 db_posicao: out std_logic_vector(1 downto 0);
23 db_estado: out std_logic_vector(6 downto 0)
24 );
25 end entity exp4_sensor_desafio;
26
27 architecture arch of exp4_sensor_desafio is
28
29 component interface_hcsr04
30 port (
31 clock : in std_logic;
32 reset : in std_logic;
33 echo : in std_logic;
34 medir : in std_logic;
35 pronto : out std_logic;
36 trigger : out std_logic;
37 medida : out std_logic_vector(11 downto 0);
38 db_estado : out std_logic_vector(3 downto 0)
39 );
40 end component;
41
42
43 component edge_detector
44 port ( clk : in std_logic;
45 signal_in : in std_logic;
46 output : out std_logic
47 );
48 end component;
49
50
51 component hex7seg
52 port (
53 hexa : in std_logic_vector(3 downto 0);
54 sseg : out std_logic_vector(6 downto 0)
55 );
56 end component;
57
58 component controle_servo is
59 port (
60 clock : in std_logic;
61 reset : in std_logic;
62 posicao : in std_logic_vector(1 downto 0);
63 pwm : out std_logic );
64 end component;
65
66 signal s_db_estado, s_medida0, s_medida1, s_medida2: std_logic_vector(3 downto 0);

```

```

67 signal s_medida: std_logic_vector(11 downto 0);
68 signal s_medir, s_db_echo, s_trigger, s_pwm: std_logic;
69
70 begin
71
72 DETECTOR_DE_BORDA: edge_detector port map(
73     clock,
74     medir,
75     s_medir
76 );
77
78 INTERFACE: interface_hcsr04 port map(
79     clock,
80     reset,
81     echo,
82     s_medir,
83     pronto,
84     s_trigger,
85     s_medida,
86     s_db_estado
87 );
88
89 CONTROLE_PWM: controle_servo port map (
90     clock,
91     reset,
92     posicao,
93     s_pwm
94 );
95
96 HEXA_ESTADO: hex7seg port map(
97     s_db_estado,
98     db_estado
99 );
100
101 HEXA_DIGITO0: hex7seg port map(
102     s_medida0,
103     hex0
104 );
105
106 HEXA_DIGITO1: hex7seg port map(
107     s_medida1,
108     hex1
109 );
110
111 HEXA_DIGITO2: hex7seg port map(
112     s_medida2,
113     hex2
114 );
115
116 s_medida0 <= s_medida (3 downto 0);
117 s_medida1 <= s_medida (7 downto 4);
118 s_medida2 <= s_medida (11 downto 8);
119
120 pwm <= s_pwm;
121 db_pwm <= s_pwm;
122 db_posicao <= posicao;
123 db_echo <= echo;
124 db_trigger <= s_trigger;
125 trigger <= s_trigger;
126
127 end arch;

```

6.3.2 controle_servo

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity controle_servo is
6 port (
7     clock      : in  std_logic; — 50MHz
8     reset      : in  std_logic;
9     posicao   : in  std_logic_vector(1 downto 0); — 00=0ms, 01=1ms 10=1.5ms 11=2ms
10    pwm        : out std_logic );
11 end controle_servo;
12
13 architecture rtl of controle_servo is
14
15  constant CONTAGEMMAXIMA : integer := 1000000;
16  signal contagem      : integer range 0 to CONTAGEMMAXIMA-1;
17  signal posicao_pwm  : integer range 0 to CONTAGEMMAXIMA-1;
18  signal s_posicao     : integer range 0 to CONTAGEMMAXIMA-1;
19
20 begin
21
22  process(clock,reset,posicao)
23 begin
24      — inicia contagem e posicao
25  if(reset='1') then
26      contagem <= 0;
27      pwm      <= '0';
28      posicao_pwm <= s_posicao;
29  elsif(rising_edge(clock)) then
30      — saida
31      if(contagem < posicao_pwm) then
32          pwm <= '1';
33      else
34          pwm <= '0';
35      end if;
36      — atualiza contagem e posicao
37      if(contagem=CONTAGEMMAXIMA-1) then
38          contagem <= 0;
39          posicao_pwm <= s_posicao;
40      else
41          contagem <= contagem + 1;
42      end if;
43  end if;
44 end process;
45
46 process(posicao)
47 begin
48  case posicao is
49  when "01" => s_posicao <= 50000; — pulso de 1ms
50  when "10" => s_posicao <= 75000; — pulso de 1.5 ms
51  when "11" => s_posicao <= 100000; — pulso de 2 ms
52  when others => s_posicao <= 0; — nulo saida 0
53  end case;
54 end process;
55
56 end rtl;

```

6.4 Testbenches

6.4.1 interface_hcsr04_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 interface_hcsr04_tb IS
20 END ENTITY;
21
22 ARCHITECTURE tb OF interface_hcsr04_tb IS
23
24 — Componente a ser testado (Device Under Test — DUT)
25 COMPONENT interface_hcsr04 IS
26     PORT (
27         clock : IN STD_LOGIC;
28         reset : IN STD_LOGIC;
29         echo : IN STD_LOGIC;
30         medir : IN STD_LOGIC;
31         pronto : OUT STD_LOGIC;
32         trigger : OUT STD_LOGIC;
33         medida : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
34         db_estado : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
35     );
36 END COMPONENT;
37
38 — Declara o de sinais para conectar o componente a ser testado (DUT)
39 — valores iniciais para fins de simulacao (ModelSim)
40 SIGNAL clock_in : STD_LOGIC := '0';
41 SIGNAL reset_in : STD_LOGIC := '0';
42 SIGNAL echo_in : STD_LOGIC := '0';
43 SIGNAL medir_in : STD_LOGIC := '0';
44 SIGNAL pronto_out : STD_LOGIC := '0';
45 SIGNAL trigger_out : STD_LOGIC := '0';
46 SIGNAL medida_out : STD_LOGIC_VECTOR(11 DOWNTO 0) := "000000000000";
47 SIGNAL db_estado_out : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
48
49 — Configura es do clock
50 SIGNAL keep_simulating : STD_LOGIC := '0'; — delimita o tempo de gera o do clock
51 CONSTANT clockPeriod : TIME := 20 ns; — clock de 50MHz
52
53 BEGIN
54     — Gerador de clock: executa enquanto 'keep_simulating = 1', com o per odo
55     — especificado. Quando keep_simulating=0, clock interrompido, bem como a
56     — simula o de eventos
57     clock_in <= (NOT clock_in) AND keep_simulating AFTER clockPeriod/2;
58
59     — Conecta DUT (Device Under Test)

```

```

60      dut : interface_hcsr04
61      PORT MAP
62      (
63          clock => clock_in ,
64          reset => reset_in ,
65          echo => echo_in ,
66          medir => medir_in ,
67          pronto => pronto_out ,
68          trigger => trigger_out ,
69          medida => medida_out ,
70          db_estado => db_estado_out
71      );
72
73      — geracao dos sinais de entrada (estimulos)
74      stimulus : PROCESS IS
75      BEGIN
76
77          ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
78          keep_simulating <= '1';
79
80          —— inicio da simulacao: reset ——————
81          reset_in <= '1';
82          WAIT FOR 20 * clockPeriod; — pulso com 20 periodos de clock
83          reset_in <= '0';
84          WAIT UNTIL falling_edge(clock_in);
85          WAIT FOR 50 * clockPeriod;
86
87          —— caso de teste #1
88          medir_in <= '1';
89          WAIT FOR 20 * clockPeriod;
90          medir_in <= '0';
91
92          —— acionamento da partida (inicio da transmissao)
93          WAIT FOR 10000 * clockPeriod;
94          echo_in <= '1';
95          WAIT UNTIL rising_edge(clock_in);
96          WAIT FOR 12000 * clockPeriod; — pulso partida com 5 periodos de clock
97          echo_in <= '0';
98
99          —— espera final da transmissao (pulso pronto em 1)
100         WAIT UNTIL pronto_out = '1';
101
102         —— final do caso de teste 1
103
104         — intervalo entre casos de teste
105         WAIT FOR 10000 * clockPeriod;
106
107         —— caso de teste #2
108         medir_in <= '1';
109         WAIT FOR 20 * clockPeriod;
110         medir_in <= '0';
111
112         —— acionamento da partida (inicio da transmissao)
113         WAIT FOR 10000 * clockPeriod;
114         echo_in <= '1';
115         WAIT UNTIL rising_edge(clock_in);
116         WAIT FOR 14000 * clockPeriod; — pulso partida com 5 periodos de clock
117         echo_in <= '0';
118
119         —— espera final da transmissao (pulso pronto em 1)
120         WAIT UNTIL pronto_out = '1';
121
122         —— final do caso de teste 2
123
124         — intervalo entre casos de teste
125         WAIT FOR 10000 * clockPeriod;

```

```

126
127      —— caso de teste #3
128      medir_in <= '1';
129      WAIT FOR 20 * clockPeriod;
130      medir_in <= '0';
131
132      —— acionamento da partida (inicio da transmissao)
133      WAIT FOR 10000 * clockPeriod;
134      echo_in <= '1';
135      WAIT UNTIL rising_edge(clock_in);
136      WAIT FOR 945000 * clockPeriod; — pulso partida com 5 periodos de clock
137      echo_in <= '0';
138
139      —— espera final da transmissao (pulso pronto em 1)
140      WAIT UNTIL pronto_out = '1';
141
142      —— final do caso de teste 3
143
144      — intervalo entre casos de teste
145      WAIT FOR 10000 * clockPeriod;
146
147      —— final dos casos de teste da simulacao
148      ASSERT false REPORT "Fim da simulacao" SEVERITY note;
149      keep_simulating <= '0';
150
151      WAIT; — fim da simulacao: aguarda indefinidamente
152  END PROCESS;
153 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.