# TESTE DE REFLEXO

Uma aplicação de sistemas embarcados em RPGA

Ariadne Cecília Evangelista ○ Arthur Phelipe Queiroz ○ César Augusto Azevedo

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# INTRODUÇÃO

## Desenvolvimento

Desenvolvimento de um sistema digital embarcado com foco em tempo de reação.

## Implementação

Implementado em VHDL na plataforma DE2 (FPGA Cyclone II).

## Aplicações

Testes psicotécnicos, reabilitação motora e ensino de lógica digital.

## Estados principais

FSM com cinco estados principais: IDLE, WAITING, GO, SUCCESS e SHOW.

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# REQUISITOS TÉCNICOS

**3 botões**
Início, Reação e Reset

**4 LEDs**
estados do sistema (espera, pronto, erro, sucesso)

**2 Displays de 7 segmentos**
pontuação

**Comportamento lógico**
FSM com controle de tempo aleatório via LFSR

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# FUNCIONAMENTO DO SISTEMA

1

**Aguardando início** (KEY0), todos os LEDs desligados.

IDLE

2

**Tempo aleatório (2–7s) com LEDG0 aceso.**
Pressionou antes? → SHOW (Erro)
Tempo esgotado? → GO

WAITING

3

**LEDG2 aceso.**
Reagiu a tempo (KEY1)? → SUCCESS
Não reagiu? → SHOW

GO

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# FUNCIONAMENTO DO SISTEMA

4

**Todos os LEDs acesos, ponto contado.**
Ao soltar o botão, reinicia nova rodada com novo tempo.

SUCESS

5

**LEDG3 aceso, erro sinalizado.**
Pressionar KEY0 para reiniciar (volta ao IDLE).

SHOW

6

A cada acerto, a pontuação é atualizada nos displays HEX0 e HEX1

DISPLAY

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# CÓDIGO VHDL

000
1

```vhdl
-- BIBLIOTECAS USADAS ----------------------------------------------------------------------------
-- Esta seção importa as bibliotecas necessárias para o projeto.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- Define o tipo padrão STD_LOGIC e funções relacionadas. Essencial
para qualquer projeto VHDL.
use IEEE.STD_LOGIC_ARITH.ALL; -- Fornece funções aritméticas para os tipos STD_LOGIC_VECTOR.
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- Permite tratar vetores STD_LOGIC_VECTOR como números sem sinal,
facilitando comparações e operações.
----------------------------------------------------------------------------------------------
-- ENTIDADE --------------------------------------------------------------------------------------
-- A entidade "Jogo" define a interface do nosso circuito, ou seja, suas portas de entrada e saída.

entity Jogo is
    Port (
        -- ENTRADAS
        clk       : in  STD_LOGIC; -- Sinal de clock principal (relógio) que sincroniza todo o
circuito.
        reset     : in  STD_LOGIC; -- Sinal para reiniciar o jogo e todos os contadores.
        start_btn : in  STD_LOGIC; -- Botão para iniciar ou reiniciar uma partida.
        react_btn : in  STD_LOGIC; -- Botão de reação do jogador.

        -- SAÍDAS
        leds      : out STD_LOGIC_VECTOR(3 downto 0); -- Leds para indicar o estado atual do jogo.
        HEX0      : out STD_LOGIC_VECTOR(6 downto 0); -- Display de 7 segmentos para a unidade do
placar.
        HEX1      : out STD_LOGIC_VECTOR(6 downto 0); -- Display de 7 segmentos para a dezena do
placar.
        HEX2      : out STD_LOGIC_VECTOR(6 downto 0); -- Display não utilizado.
        HEX3      : out STD_LOGIC_VECTOR(6 downto 0); -- Display não utilizado.
        HEX4      : out STD_LOGIC_VECTOR(6 downto 0); -- Display não utilizado.
        HEX5      : out STD_LOGIC_VECTOR(6 downto 0); -- Display não utilizado.
        HEX6      : out STD_LOGIC_VECTOR(6 downto 0); -- Display não utilizado.
        HEX7      : out STD_LOGIC_VECTOR(6 downto 0)  -- Display não utilizado.
    );
end Jogo;
----------------------------------------------------------------------------------------------
```

# CODIGO VHDL

```
38  -- ARQUITETURA ------------------------------------------------------------------------------
39  -- A arquitetura "Behavioral" descreve como o circuito funciona internamente.
40
41  architecture Behavioral of Jogo is
42
43      -- CONSTANTES DE TEMPO E FREQUÊNCIA
44      constant CLK_FREQ_HZ          : integer := 45_000_000;  -- Frequência do clock da placa (ex: 45
45  MHz).
46      constant MIN_WAIT_S           : real    := 2.0;         -- Tempo mínimo de espera antes do
47  sinal para reagir (2 segundos).
48      constant MAX_WAIT_S           : real    := 7.0;         -- Tempo máximo de espera (7 segundos).
49      constant REACTION_WINDOW_S    : real    := 1.0;         -- Janela de tempo que o jogador tem
50  para reagir (1 segundo).
51
52      -- CONVERSÃO DE TEMPO PARA CICLOS DE CLOCK
53      -- O hardware opera em ciclos de clock, não em segundos. Convertemos os tempos para a
54  quantidade
55      -- de ciclos correspondente, com base na frequência do clock.
56      constant MAX_WAIT_CYCLES      : integer := integer(MAX_WAIT_S * real(CLK_FREQ_HZ)); -- Nº
57  máximo de ciclos de espera.
58      constant MIN_WAIT_CYCLES      : integer := integer(MIN_WAIT_S * real(CLK_FREQ_HZ)); -- Nº
59  mínimo de ciclos de espera.
60      constant WAIT_RANGE_CYCLES    : integer := integer((MAX_WAIT_S - MIN_WAIT_S) *
```

0011

# CODIGO VHDL

```
real(CLK_FREQ_HZ)); -- Faixa de ciclos de espera aleatória.
    constant REACTION_WINDOW_CYCLES : integer := integer(REACTION_WINDOW_S * real(CLK_FREQ_HZ)); --
Nº de ciclos para a janela de reação.

    -- GERAÇÃO DE NÚMERO ALEATÓRIO
    -- Para tornar o tempo de espera imprevisível, usamos um gerador de números pseudoaleatórios
(LFSR).
    -- Este valor divide a faixa de espera em 16 passos (0 a 15) para simplificar a geração do
tempo alvo.
    constant RANDOM_STEP_CYCLES   : integer := WAIT_RANGE_CYCLES / 15;

    -- MÁQUINA DE ESTADOS
    -- A máquina de estados controla o fluxo do jogo.
    type state_type is (IDLE, WAITING, GO, SUCCESS, SHOW); -- Define os 5 estados possíveis do
jogo.
    signal state : state_type := IDLE; -- Sinal que armazena o estado atual. O jogo começa em IDLE
(ocioso).

    -- SINAIS INTERNOS
    -- Sinais são como "fios" internos que conectam diferentes partes do nosso circuito.
    signal wait_count     : integer range 0 to MAX_WAIT_CYCLES := 0; -- Contador de ciclos de clock
para os tempos de espera e reação.
    signal random_target  : integer range 0 to MAX_WAIT_CYCLES := 0; -- Armazena o tempo de espera
aleatório (em ciclos) para a rodada atual.
    signal lfsr_counter   : STD_LOGIC_VECTOR(25 downto 0) := (others => '0'); -- Registrador de
deslocamento para gerar números pseudoaleatórios.

    signal start_btn_prev : STD_LOGIC := '1'; -- Armazena o estado anterior do botão 'start' para
detectar a borda de descida (quando o botão é pressionado).

    -- SINAIS PARA O PLACAR
    signal score_count    : integer range 0 to 99 := 0; -- Armazena a pontuação do jogador (0 a
99).
    signal digit_unidades : integer range 0 to 9;       -- Armazena o dígito das unidades do
placar.
    signal digit_dezenas  : integer range 0 to 9;       -- Armazena o dígito das dezenas do placar.

begin
```

# CODIGO VHDL

```
100     -- GERADOR DE NÚMEROS PSEUDOALEATÓRIOS (LFSR)
101     -- Este processo implementa um Linear Feedback Shift Register (LFSR).
102     -- A cada ciclo de clock, ele gera um novo padrão de bits, criando uma sequência
103 pseudoaleatória.
104     -- Usamos essa sequência para determinar o tempo de espera em cada rodada.
105     process(clk) -- O processo é sensível apenas à borda de subida do clock.
106     begin
107         if rising_edge(clk) then
108             -- A lógica de realimentação (XOR entre os bits 25 e 3) gera o novo bit de entrada.
109             lfsr_counter(0) <= lfsr_counter(25) xor lfsr_counter(3);
110             -- O registrador é deslocado para a esquerda.
111             lfsr_counter(25 downto 1) <= lfsr_counter(24 downto 0);
112         end if;
113     end process;
114 
115     -- PROCESSO PRINCIPAL: MÁQUINA DE ESTADOS DO JOGO
116     -- Controla a transição entre os estados e a lógica principal.
117     process(clk)
118         variable random_base : integer range 0 to 15; -- Variável temporária para o cálculo do
119 tempo aleatório.
120     begin
121         if rising_edge(clk) then -- Toda a lógica é sincronizada com a borda de subida do clock.
122 
123             start_btn_prev <= start_btn; -- Atualiza o estado anterior do botão 'start' a cada
124 ciclo.
125 
```

# CODIGO VHDL

0101

```
126             if reset = '1' then -- Se o botão de reset for pressionado, o jogo volta ao estado
127 inicial.
128                 state <= IDLE;
129                 wait_count <= 0;
130                 leds <= "0000";
131                 start_btn_prev <= '1';
132                 score_count <= 0;
133             else
134                 -- A estrutura CASE seleciona a lógica a ser executada com base no estado atual.
135                 case state is
136 
137                     -- ESTADO IDLE: Ocioso, esperando o início do jogo.
138                     when IDLE =>
139                         leds <= "0000"; -- Todos os leds apagados.
140                         -- Detecta o pressionar do botão 'start' (transição de '1' para '0').
141                         if start_btn_prev = '1' and start_btn = '0' then
142                             state <= WAITING; -- Muda para o estado de espera.
143                             wait_count <= 0;  -- Zera o contador de tempo.
144                             -- Gera um novo tempo de espera aleatório.
145                             random_base  := conv_integer(lfsr_counter(23 downto 20)); -- Pega 4
146 bits do LFSR (valor de 0 a 15).
147                             random_target<= MIN_WAIT_CYCLES + (random_base * RANDOM_STEP_CYCLES);
148 -- Calcula o tempo alvo.
149                         end if;
150 
151                     -- ESTADO WAITING: Esperando o tempo aleatório para acender o led de reação.
152                     when WAITING =>
153                         leds <= "0001"; -- Acende um led para indicar que o jogo está em andamento.
154                         wait_count <= wait_count + 1; -- Incrementa o contador de tempo a cada
155 ciclo.
156                         -- Verifica se o jogador pressionou o botão de reação cedo demais.
157                         if react_btn = '0' then
158                             state <= SHOW; -- Se sim, o jogador perdeu. Vai para o estado de
159 exibição de resultado.
160                             leds <= "1000"; -- Acende o led de "falha".
161                         -- Verifica se o tempo de espera aleatório foi atingido.
162                         elsif wait_count >= random_target then
163                             state <= GO; -- Se sim, muda para o estado 'GO', onde o jogador deve
164 reagir.
```

# CODIGO VHDL

0110

```
165                         wait_count <= 0; -- Zera o contador para medir o tempo de reação.
166                     end if;
167
168                 -- ESTADO GO: Sinaliza para o jogador reagir.
169                 when GO =>
170                     leds <= "0100"; -- Acende o led de "reaja agora!".
171                     wait_count <= wait_count + 1; -- Começa a contar o tempo de reação.
172                     -- Verifica se o jogador pressionou o botão.
173                     if react_btn = '0' then
174                         state <= SUCCESS; -- Se sim, o jogador acertou!
175                         wait_count <= 0; -- Zera o contador.
176                     -- Verifica se o tempo para reagir (1 segundo) esgotou.
177                     elsif wait_count >= REACTION_WINDOW_CYCLES then
178                         state <= SHOW; -- Se o tempo esgotou, o jogador perdeu.
179                         leds <= "1000"; -- Acende o led de "falha".
180                     end if;
181
182                 -- ESTADO SUCCESS: O jogador reagiu a tempo.
183                 when SUCCESS =>
184                     leds <= "1111"; -- Acende todos os leds para comemorar.
185                     -- Espera o jogador soltar o botão de reação para evitar múltiplos
186 incrementos no placar.
187                     if react_btn = '1' then
188                         -- Incrementa o placar. Se chegar a 99, volta para 0.
189                         if score_count = 99 then score_count <= 0; else score_count <=
190 score_count + 1; end if;
```

# CODIGO VHDL

0111

```
                        wait_count <= 0; -- Zera o contador para medir o tempo de reação.
                    end if;

                -- ESTADO GO: Sinaliza para o jogador reagir.
                when GO =>
                    leds <= "0100"; -- Acende o led de "reaja agora!".
                    wait_count <= wait_count + 1; -- Começa a contar o tempo de reação.
                    -- Verifica se o jogador pressionou o botão.
                    if react_btn = '0' then
                        state <= SUCCESS; -- Se sim, o jogador acertou!
                        wait_count <= 0; -- Zera o contador.
                    -- Verifica se o tempo para reagir (1 segundo) esgotou.
                    elsif wait_count >= REACTION_WINDOW_CYCLES then
                        state <= SHOW; -- Se o tempo esgotou, o jogador perdeu.
                        leds <= "1000"; -- Acende o led de "falha".
                    end if;

                -- ESTADO SUCCESS: O jogador reagiu a tempo.
                when SUCCESS =>
                    leds <= "1111"; -- Acende todos os leds para comemorar.
                    -- Espera o jogador soltar o botão de reação para evitar múltiplos
incrementos no placar.
                    if react_btn = '1' then
                        -- Incrementa o placar. Se chegar a 99, volta para 0.
                        if score_count = 99 then score_count <= 0; else score_count <=
score_count + 1; end if;
```

# CODIGO VHDL

100
0

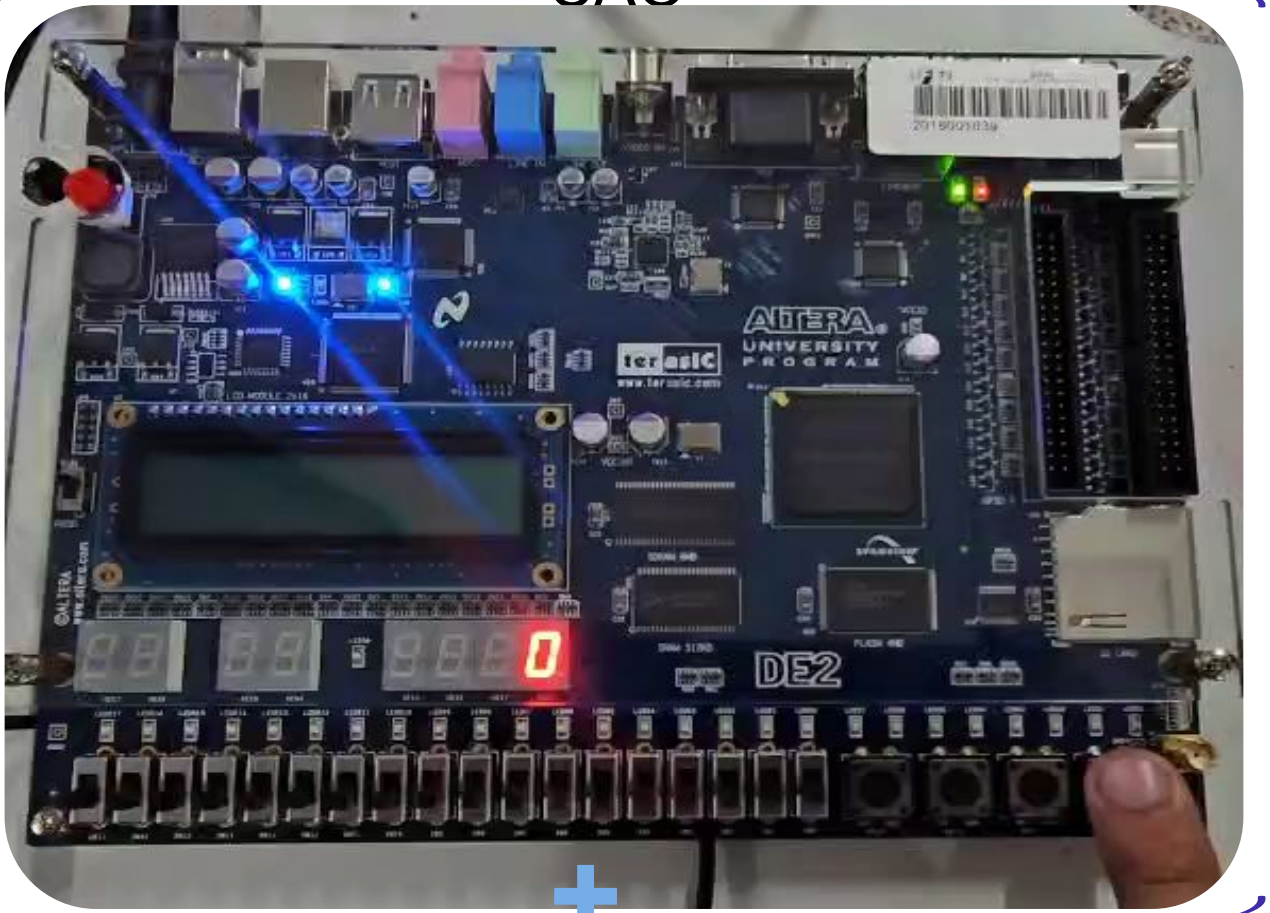
```
                        -- Prepara para a próxima rodada, voltando ao estado WAITING.
                        state <= WAITING;
                        wait_count <= 0;
                        -- Gera um novo tempo de espera aleatório para a próxima rodada.
                        random_base   := conv_integer(lfsr_counter(23 downto 20));
                        random_target<= MIN_WAIT_CYCLES + (random_base * RANDOM_STEP_CYCLES);
                    end if;

                -- ESTADO SHOW: Exibe o resultado de uma falha (reagiu cedo ou tarde demais).
                when SHOW =>
                    leds <= "1000"; -- Mantém o led de "falha" aceso.
                    -- Espera o jogador pressionar 'start' para recomeçar do zero.
                    if start_btn_prev = '1' and start_btn = '0' then
                        state <= IDLE; -- Volta para o estado inicial.
                        score_count <= 0; -- Zera o placar.
                    end if;

            end case;
        end if;
    end if;
end process;

-- LÓGICA DO PLACAR E CONTROLE DOS DISPLAYS ----------------------------------------------------
-- Este bloco de código é executado continuamente para manter o placar atualizado.
-- Separa o número do placar (0-99) em dois dígitos separados: dezenas e unidades.
digit_dezenas  <= score_count / 10;   -- Divisão inteira para obter a dezena.
digit_unidades <= score_count mod 10; -- Operador de módulo (resto da divisão) para obter a
unidade.

-- LÓGICA PARA O DISPLAY DE DEZENAS (HEX1)
-- Este processo controla o que é exibido no display da dezena.
process(digit_dezenas, score_count) -- O processo é reavaliado sempre que o placar muda.
begin
    -- SUPRESSÃO DE ZERO À ESQUERDA: Se o placar for menor que 10, o display da dezena fica
apagado.
```

# CODIGO VHDL

1001

```
226         if score_count < 10 then
227             HEX1 <= "1111111"; -- Envia um valor que apaga todos os segmentos do display.
228         else
229             -- Se o placar for 10 ou mais, exibe o dígito da dezena correspondente.
230             case digit_dezenas is
231                 when 1      => HEX1 <= "1111001"; -- Mostra "1"
232                 when 2      => HEX1 <= "0100100"; -- Mostra "2"
233                 when 3      => HEX1 <= "0110000"; -- Mostra "3"
234                 when 4      => HEX1 <= "0011001"; -- Mostra "4"
235                 when 5      => HEX1 <= "0010010"; -- Mostra "5"
236                 when 6      => HEX1 <= "0000010"; -- Mostra "6"
237                 when 7      => HEX1 <= "1111000"; -- Mostra "7"
238                 when 8      => HEX1 <= "0000000"; -- Mostra "8"
239                 when 9      => HEX1 <= "0010000"; -- Mostra "9"
240                 when others=> HEX1 <= "1111111"; -- Apagado por segurança (não deve acontecer).
241             end case;
242         end if;
243     end process;
244
245     -- LÓGICA PARA O DISPLAY DE UNIDADES (HEX0)
246     -- Esta é uma atribuição condicional concorrente. É uma forma mais compacta de um 'case'.
247     -- Ele mapeia o valor do dígito da unidade para o padrão de 7 segmentos correspondente.
248     with digit_unidades select
249         HEX0 <=
250             "1000000" when 0, -- "0"
251             "1111001" when 1, -- "1"
252             "0100100" when 2, -- "2"
253             "0110000" when 3, -- "3"
254             "0011001" when 4, -- "4"
255             "0010010" when 5, -- "5"
256             "0000010" when 6, -- "6"
257             "1111000" when 7, -- "7"
258             "0000000" when 8, -- "8"
259             "0010000" when 9, -- "9"
260             "1111111" when others; -- Apagado por segurança.
261
262     -- DESLIGAR DISPLAYS NÃO UTILIZADOS
263     -- Atribuímos um valor constante a estes displays para mantê-los sempre apagados.
264     HEX2 <= "1111111";
265     HEX3 <= "1111111";
266     HEX4 <= "1111111";
267     HEX5 <= "1111111";
268     HEX6 <= "1111111";
269     HEX7 <= "1111111";
270
271 end Behavioral;
```

# DEMONSTRA CÃO

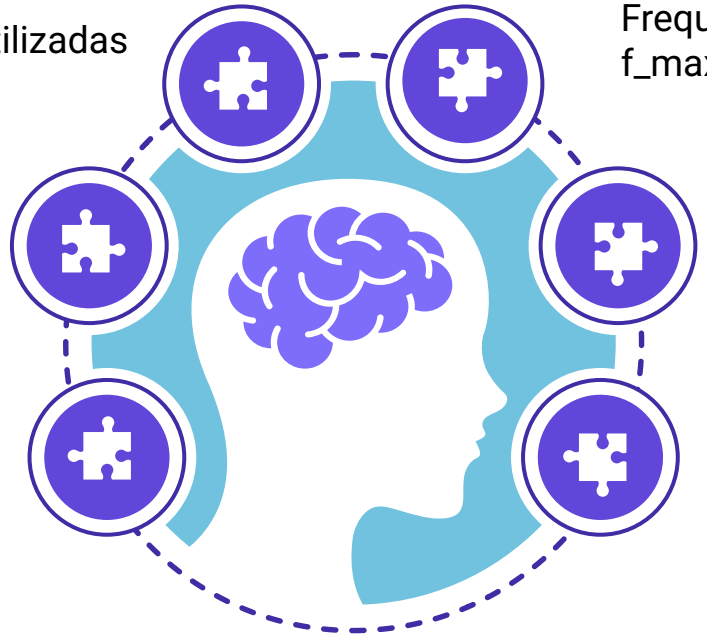


# BLOCOS LÓGICOS, CONSUMO E DESEMPENHO

293 células lógicas utilizadas

95 registradores dedicados

64 pinos de I/O



Frequência alvo: 45 MHz |
f_max: 133,12 MHz

Folga de tempo (slack):
+14,710 ns

Projeto estável e com ótima margem de segurança

Resultados em dados

# TESTE DE REFLEXO

Uma aplicação de sistemas embarcados em RPGA

Ariadne Cecília, Arthur Phelipe e César Augusto