Report on Reaction Timer System

Table of Contents

1. Introduction

The Reaction Timer System is a hardware-based project designed to measure the reaction times of two players and determine a winner based on their performances. The system utilizes an FSM-based controller to manage the state transitions and handle the timing and scoring logic. The project is implemented using VHDL and targets FPGA devices.

2. System Overview

The system comprises the following key components:

- **Clock Generation Modules**: Generate the required clock signals.
- **Key Press Detection**: Detects and debounces key presses.
- **FSM Controller**: Manages the overall system state and logic.
- **Display Modules**: Converts binary values to BCD and then to seven-segment display format.

3. Top-Level Design

The top-level design, `Reaction_Timer_System`, integrates various components to build the complete reaction timer system. The entity declaration includes inputs for the clock signal, reset, and player inputs, as well as outputs for LEDs and seven-segment displays.

3.1. Entity Declaration

```
entity Reaction_Timer_System is
  Port (
    clk_50mhz : in std_logic;
    reset     : in std_logic;
    KEYs      : in std_logic_vector(3 downto 0);
    LEDs      : out std_logic_vector(7 downto 0);
    HEX0      : out std_logic_vector(6 downto 0);
    HEX1      : out std_logic_vector(6 downto 0);
    HEX2      : out std_logic_vector(6 downto 0);
    HEX3      : out std_logic_vector(6 downto 0);
    HEX4      : out std_logic_vector(6 downto 0);
    HEX5      : out std_logic_vector(6 downto 0);
    HEX6      : out std_logic_vector(6 downto 0);
    HEX7      : out std_logic_vector(6 downto 0)
  );
end Reaction_Timer_System;
```

4. Component Descriptions

4.1. Clock Generation Modules

The system includes two clock generation modules: `clock_ms` and `clock_1ms_1hz`. These modules derive a 1ms clock signal from the 50MHz input clock and subsequently generate a 1Hz clock signal from the 1ms clock.

4.2. Key Press Detection

The `keypressed` component detects and debounces the key presses. It generates an enable signal when a key is pressed, ensuring stable input to the FSM controller.

4.3. FSM Controller

The `controller` component is the core of the system, managing the game logic through various states. It controls the timers, scores, and stimuli for players A and B.

4.3.1. Entity Declaration

```
component controller
  port(
    clock           : in std_logic;
    reset           : in std_logic;
    clk_1ms         : in std_logic;
    clk_1hz         : in std_logic;
    player_A        : in std_logic;
    player_B        : in std_logic;
    start           : in std_logic;
    target_confirm  : in std_logic;
    target_score    : out unsigned(5 downto 0);
    config_enable_o : out std_logic;
    test_cycles     : out unsigned(5 downto 0);
    score_playerA   : out unsigned(5 downto 0);
    score_playerB   : out unsigned(5 downto 0);
    stimulus_playerA: out std_logic_vector(3 downto 0);
    stimulus_playerB: out std_logic_vector(3 downto 0)
  );
end component;
```

4.4. Display Modules

The system uses several display modules to convert binary scores to BCD and then to seven-segment display format.

4.4.1. Bin to BCD Conversion

The `Bin_to_BCD` component converts a 6-bit binary number to BCD format, separating it into tens and units.

4.4.2. BCD to Seven-Segment Conversion

The `BCD_to_7Segment` component converts a BCD digit to the corresponding seven-segment display pattern.

5. FSM Controller Implementation

The FSM controller handles the game states, transitioning based on inputs and internal signals. Key states include `IDLE`, `CONFIG`, `WAIT_for_START`, `DELAY_TIMER`, `STIMULUS`, `POINTS_ASSIGN`, `INC_SCORE`, `COMPARE_SCORE`, `SEC_DELAY`, `WINNER`, `SP1_2`, `SP2_2`, and `PENALTY`.

5.1. State Register Process

The state register process updates the current state based on the clock and reset signals.

```
state_reg: process(clock, reset)
begin
  if(reset ='1') then
    current_state <= IDLE;
  elsif (clock'event and clock='1') then
    current_state <= next_state;
  end if;
end process;
```

5.2. Combinational Logic Process

The combinational logic process determines the next state and outputs based on the current state and inputs.

```
comb_logic: process(current_state, start, player_A, player_B, target_confirm)
begin
  case current_state is
    when IDLE =>
      -- Initialize outputs
      -- State transition logic
    when CONFIG =>
      -- Configuration state logic
    when WAIT_for_START =>
      -- Wait for start signal
    when DELAY_TIMER =>
```

```
      -- Delay timer logic
    when STIMULUS =>
      -- Stimulus generation logic
    when POINTS_ASSIGN =>
      -- Points assignment logic
    when INC_SCORE =>
      -- Increment score logic
    when COMPARE_SCORE =>
      -- Compare scores
    when SEC_DELAY =>
      -- Second delay logic
    when WINNER =>
      -- Winner determination logic
    when SP1_2 =>
      -- Player A penalty logic
    when SP2_2 =>
      -- Player B penalty logic
    when PENALTY =>
      -- Penalty state logic
    when others =>
      next_state <= IDLE;
  end case;
end process;
```

6. Display Logic

The display logic uses the converted BCD values to drive the seven-segment displays, showing the scores and system states.

```
disp_process: process(clk_50mhz, reset, config_enable_o_w, test_cycles_w)
begin
  if reset = '1' then
    HEX3 <= "0000000";
    HEX2 <= "0000000";
    HEX1 <= "0000000";
    HEX0 <= "0000000";
  elsif rising_edge(clk_50mhz) then
    if config_enable_o_w = '1' then
      -- Display "ConF"
    elsif test_cycles_w = "000000" then
      -- Display "tESt"
    else
      HEX3 <= HEX3_w;
      HEX2 <= HEX2_w;
      HEX1 <= HEX1_w;
      HEX0 <= HEX0_w;
    end if;
  end if;
end process;
```

7. Test Bench

A comprehensive test bench should be created to validate the functionality of the `controller` component. The test bench will simulate various scenarios and check the outputs against expected results.

7.1. Test Bench Structure

- **Clock and Reset Generation**: Generate clock and reset signals.
- **Input Stimulus**: Apply different input sequences to the FSM.

- **Output Monitoring**: Check the outputs and state transitions.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity tb_controller is
end tb_controller;

architecture Behavioral of tb_controller is

  -- Component Declaration for the Unit Under Test (UUT)
  component controller
    port(
      clock           : in std_logic;
      reset           : in std_logic;
      clk_1ms         : in std_logic;
      clk_1hz         : in std_logic;
      player_A        : in std_logic;
      player_B        : in std_logic;
      start           : in std_logic;
      target_confirm  : in std_logic;
      target_score    : out unsigned(5 downto 0);
      config_enable_o : out std_logic;
      test_cycles     : out unsigned(5 downto 0);
      score_playerA   : out unsigned(5 downto 0);
      score_playerB   : out unsigned(5 downto 0);
      stimulus_playerA: out std_logic_vector(3 downto 0);
      stimulus_playerB: out std_logic_vector(3 downto 0)
    );
  end component;

  signal clock           : std_logic := '0';
  signal reset           : std_logic := '0';
  signal clk_1ms         : std_logic := '0';
  signal clk_1hz         : std_logic := '0';
  signal player_A        : std_logic := '0';
  signal player_B        : std_logic := '0';
  signal start           : std_logic := '0';
  signal target_confirm  : std_logic := '0';
  signal target_score    : unsigned(5 downto 0);
  signal config_enable_o : std_logic;
  signal test_cycles     : unsigned(5 downto 0);
  signal score_playerA   : unsigned(5 downto 0);
  signal score_playerB   : unsigned(5 downto 0);
  signal stimulus_playerA: std_logic_vector(3 downto 0);
  signal stimulus_playerB: std_logic_vector(3 downto 0);

begin

  -- Clock generation
  clock <= not clock after 10 ns;
  clk_1ms <= not clk_1ms after 5000000 ns; -- 1ms clock period
  clk_1hz <= not clk_1hz after 500000000 ns; -- 1Hz clock period

  -- Instantiate the Unit Under Test (UUT)
  uut: controller
    port map(
      clock           => clock,
      reset           => reset,
      clk_1ms         => clk_1ms,
      clk_1hz         => clk_1hz,
      player_A        => player_A,
      player_B        => player_B,
      start           => start,
      target_confirm  => target_confirm,
      target_score    => target_score,
      config_enable_o => config_enable_o,
      test_cycles     => test_cycles,
      score_playerA   => score_playerA,
      score_playerB   => score_playerB,
      stimulus_playerA=> stimulus_playerA,
      stimulus_playerB=> stimulus_playerB
    );

  -- Test process
  stimulus: process
  begin
    -- Apply reset
    reset <= '1';
    wait for 20 ns;
    reset <= '0';

    -- Apply start signal
    start <= '1';
    wait for 20 ns;
    start <= '0';

    -- Player A reaction
    wait for 200 ns;
    player_A <= '1';
    wait for 20 ns;
    player_A <= '0';

    -- Player B reaction
    wait for 200 ns;
    player_B <= '1';
    wait for 20 ns;
    player_B <= '0';

    -- Wait for the simulation to end
    wait;
  end process;

end Behavioral;
```

8. Conclusion

The Reaction Timer System successfully integrates various hardware components to create a fully functional reaction timer. The project demonstrates the use of FSM for state management and employs clock division techniques, key debounce logic, and binary-to-BCD conversion for display purposes. The comprehensive design and test bench ensure that the system operates as intended, providing accurate reaction time measurements for two players.