## Report on Reaction Timer System

**Table of Contents**

[Report on Reaction Timer System 1](#__RefHeading___Toc105_2121440065)

[1. Introduction 1](#__RefHeading___Toc143_2121440065)

[2. System Overview 1](#__RefHeading___Toc107_2121440065)

[3. Top-Level Design 1](#__RefHeading___Toc109_2121440065)

[3.1. Entity Declaration 2](#__RefHeading___Toc111_2121440065)

[4. Component Descriptions 2](#__RefHeading___Toc113_2121440065)

[4.1. Clock Generation Modules 2](#__RefHeading___Toc115_2121440065)

[4.2. Key Press Detection 2](#__RefHeading___Toc117_2121440065)

[4.3. FSM Controller 2](#__RefHeading___Toc119_2121440065)

[4.3.1. Entity Declaration 2](#__RefHeading___Toc121_2121440065)

[4.4. Display Modules 3](#__RefHeading___Toc123_2121440065)

[4.4.1. Bin to BCD Conversion 3](#__RefHeading___Toc125_2121440065)

[4.4.2. BCD to Seven-Segment Conversion 3](#__RefHeading___Toc127_2121440065)

[5. FSM Controller Implementation 3](#__RefHeading___Toc129_2121440065)

[5.1. State Register Process 3](#__RefHeading___Toc131_2121440065)

[5.2. Combinational Logic Process 3](#__RefHeading___Toc133_2121440065)

[6. Display Logic 4](#__RefHeading___Toc135_2121440065)

[7. Test Bench 4](#__RefHeading___Toc137_2121440065)

[7.1. Test Bench Structure 5](#__RefHeading___Toc139_2121440065)

[8. Conclusion 6](#__RefHeading___Toc141_2121440065)

### 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.