FPGA Reaction Timer
ECE 211 Course Project

Ali Sultan Sikandar

November 18th, 2020

1. Abstract

This two-part project incorporates a reaction timer for the student-athletes of Lafayette College and a banner displaying a scrolling animation of the motto 'GO PARDS' on the Nexys A7 FPGA board. This technical report describes in detail the functionally of every single element that goes into its development as well as lays out a testing plan to ensure the correct working. This report also lists out all the code that was written for the device's development and illustrates procedures with the help of various schematics and diagrams.

2. Introduction

In order for student-athletes to give their best performance out on the field, it is essential that they are alert and ready before the start of the league and other fixtures. Therefore, the college has developed hardware features of the Nexys A7 FPGA board. These include the switches, the LED lamp, and the 7 segment display. Every single module code is written in the System Verilog programming language through the Vivado software suite. The Vivado software suite is also used to create simulations of the modules before combining them and implementing them finally on the Nexys A7 board through a bitstream.

3. System Design

3.1Higher Level Design Diagram of the Project

The diagram below depicts the entire project as a whole. The project consists of two main modules: Reaction_Top and Scrolling_Text. In order to switch between these two modules, an input called sw0 is used. This input, when asserted high, triggers the Scrolling_Text module.

The Scrolling_Text module displays the phrase 'GO PARDS' onto the seven segment display with the scrolling speed of one second.

However, when the sw0 switch is asserted low, the Reaction_Top module, which provides the functionality of a reaction timer, is operational.

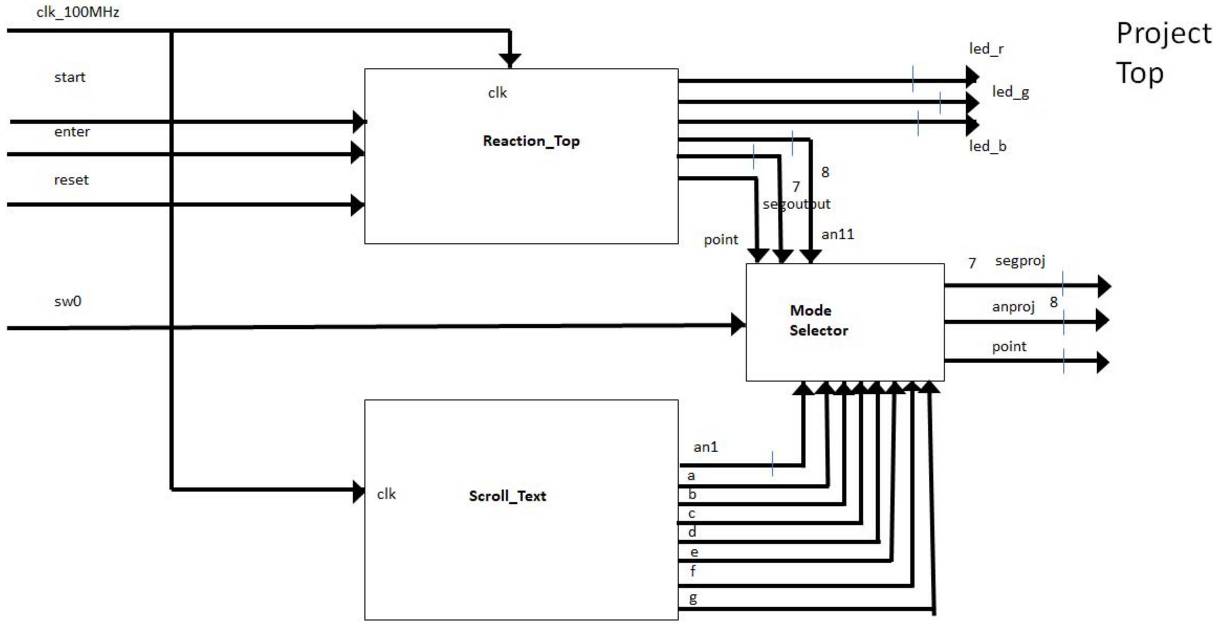


Fig 1. Project Top-Level Design

3.1.1 Inputs

This section describes the inputs of the whole project. Written inside the brackets is the hardware equivalent of these inputs on the Nexys A7 board.

- **clk_100MHz (clock signal**) - default clock signal which is 100MHz in frequency.
- **start (BUTNC)** - button to be pressed by the user of the device to start the reaction timer.
- **enter (BUTNL)** - button to be pressed by the user for the recording of his reaction time when prompted to do so
- **sw0 (SW0)** - mode selector to choose what needs to be displayed on the seven seg display: either the reaction time or the scrolling text
- **RESET (BTNL)** -button to be pressed by the user in order to reset every single module and clear the output display

3.1.2 Outputs

This section describes the inputs of the whole project. Written inside the brackets is the hardware equivalent of these inputs on the Nexys A7 board.

- **led_r (N15) -** having this asserted high only emits a red color light on the RGB lamp.
- **led_g (M16) -** having this asserted high only emits a green color light on the RGB lamp.
- **led_b (R12)-** having this asserted high only emits a blue color light on the RGB lamp.

Various combinations of these outputs being high would result in three colors: red, white, and yellow. Why and when these colors are displayed on the Nexys A7 board will be discussed further in the next section.

- **segproj (T10,R10,K13,K16,P15,T11,L18)** - this seven-bit output contains the info regarding what needs to be displayed on the seven segment display.
- **anproj (J10,J18,T9,J14,P14,K14,U13)** - this eight-bit output dictates which one of the eight anodes of the seven-segment display needs to be chosen.

We will see further how segproj and anproj outputs are decided by the clock signal in a sequential manner.

- **point(H15)** - this output just asserts high for the very right decimal point on the display panel. We need this to be visible in our portrayal of the reaction time measured in seconds.

3.2 Higher Level Design Diagram of Scrolling Text

The diagram below depicts the implementation of the first major module: Scrolling Text. As mentioned before, this module is responsible for displaying the scrolling text 'GO PARDS' on the seven segment display panel.

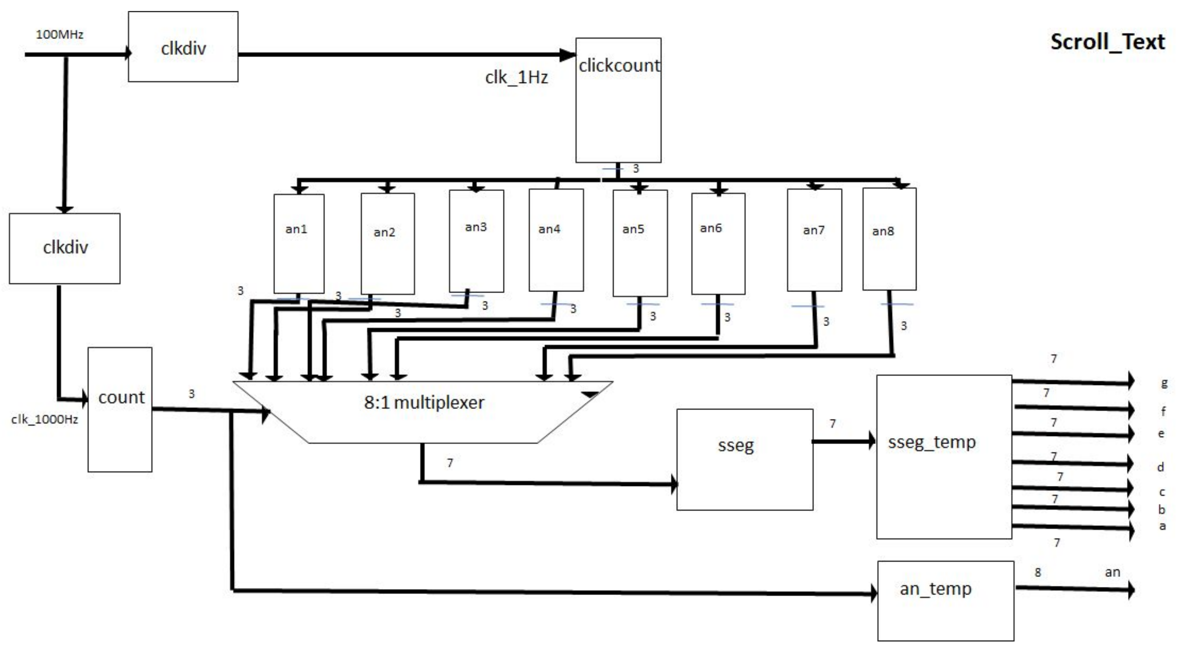


Fig 2. Scrolling_Text Top-Level Design

3.2.1 Inputs

This section describes the inputs of the Scrolling_Text module.

- **clk_100MHz (clock signal**) - default clock signal which is 100MHz in frequency is used as the only input to this module.

3.2.2 Outputs

This section describes the outputs of the Scrolling_Text module.

- **{a,b,c,d,e,f,g} -** these seven outputs constitute the seven segment display information .
- **an -** this eight-bit output just dictates which anode of the eight has to be asserted high. All the other 7 are asserted low.

3.2.3 Sub-Module of Scrolling_Text

3.2.2 Clock-Divider

Two clock divider modules are used in the Scrolling_Text main module. One of them is of 1Hz and the other is 100 Hz. The 1Hz clock signal is used for the shifting of the letters on the display while the 1000 Hz one selects the display anode. Since this 1000Hz is so fast we see all the anodes displaying some letters even though only one of them is asserted high at every clock edge.

3.2.3 ClickCount

This submodule is one of the two counters we used in this main module. It counts up to eight and therefore the bit-width counter within the module is of 3 bits. We count up to 8 bits because there are eight possible shifts that occur sequentially to display a scrolling text.

3.2.4 The eight an registers

Based on what the current count value of the click count module is, the 8 registers hold the LED values that need to be displayed at that shift. For example, if at count 1, the an1 storing the seven segment information for G, the an2 will store seven segment information for O, the an3 will store seven segment information for _, the an4 will store seven segment information for P, the an5 will store seven segment information for A, the an6 will store seven segment information for R, the an7 will store seven segment information for D, and the an2 will store seven segment information for S. At the next count, the an1 storing the seven segment information for S, the an2 will store seven segment information for G, the an3 will store seven segment information for O, the an4 will store seven segment information for _, the an5 will store seven segment information for P, the an6 will store seven segment information for A, the an7 will store seven segment information for R, and the an2 will store seven segment information for D. This will continue on till the eight-count after which it goes back to the count 0 with original orientation of the letters.

3.2.5 Count

The second counter of this main module will be counting up to 16 since it is 4 bit. Since it is incrementing its count at a 1000Hz frequency, we will not be noticing which anode is being selected as they will all appear to be on.

3.2.6 8:2 Multiplexer

This module will be using a case statement to choose what register input it needs to relay to the sseg module. The selector for this multiplexer is the count value outputted from the count counter.

3.2.7 SSEG/SSEG_Temp

The SSEG register basically holds the seven-segment information relayed from one of the 8th registers from an1 to an8. This is relayed to the SSEG_Temp which has the corresponding seven segment 8 bits to particular letters hard-coded into it. The output, then, is just the seven-segment info encoded into bits a,b,c,d,e,f, and g.

3.2.7An_Temp

This is an eight-bit register which uses the count value imputed into it to select which of the eight anodes is asserted high while others are asserted low at every clock edge. The output is the 8 bit which is relayed to the seven-segment display panel.

.

3.2 Higher Level Design Diagram of Reaction Timer

The diagram below depicts the implementation of the second major module: Reaction Timer. This module implements a human reaction timer. The order of operations is as follows:

1. Press the Start button
2. Wait for a random amount of time before the GO LED turns white.
3. When the LED is white, that is when the user of the device is prompted to press the enter button in order for his reaction time to be recorded and displayed on the seven segment display panel.

There are also additional errors that could occur. One of the errors that could occur is the user pressing the enter button while the reaction timer is still in the random wait state. When that happens, the GO led lamp displays red for five seconds. Another potential error that could occur is the user not pressing the enter button after he has pressed the start button at all. In that case, the GO Led will keep on displaying white for 10 seconds after which it goes on to display yellow color for 5 seconds.

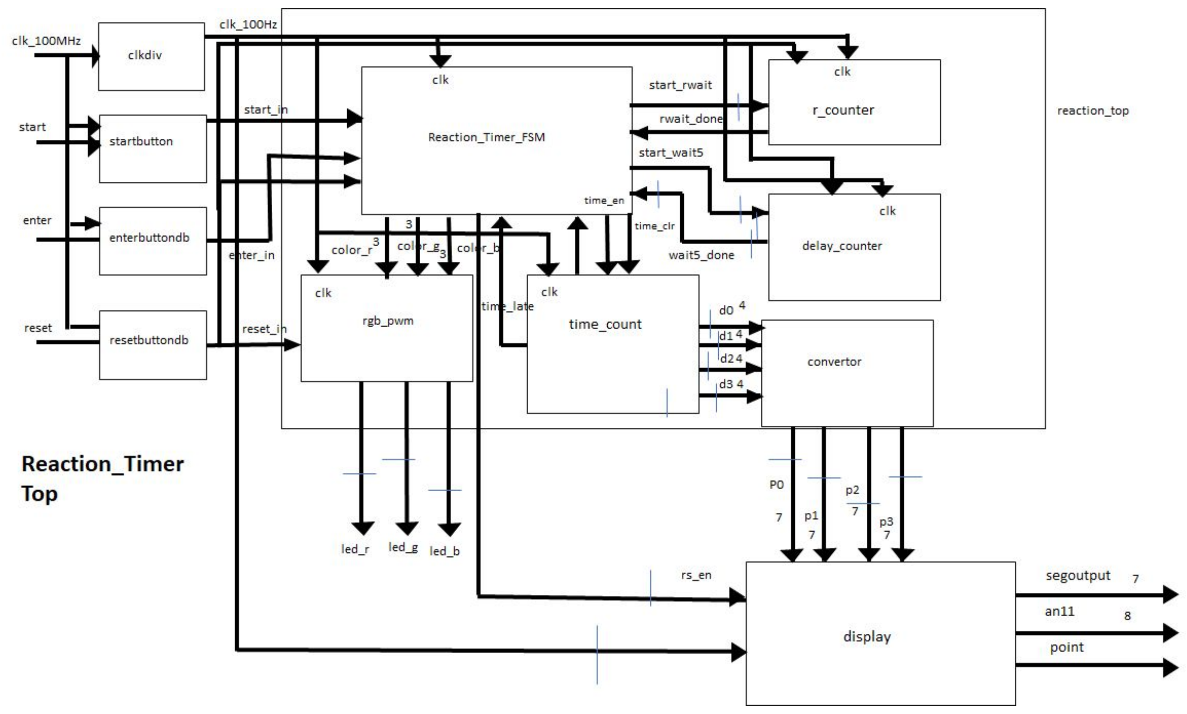


Fig 3. Reaction Timer Top-Level Design

3.2.1 Inner logic Inputs/Outputs of

This section describes the inputs and outputs within the Reaction Timer module. It must be noted that all the inputs and some of the outputs in this module are inner logic wires.

- **clk_100Hz (clock signal**) - 1000Hz clock signal is used by all the modules. This is because we will be counting time up to the 1000th unit which is one millisecond.

- **start_in, enter_in, and reset_in -** these signals coming from the enter, start, and reset input of the project except that they have been eliminated of all the noise by being passed through the debounced module.
  - **d0** (unit of the time count)
  - **d1** (a tenth of the time count)
  - **d2** (hundredth of the time count)
  - **d3** (hundredth of the time count)

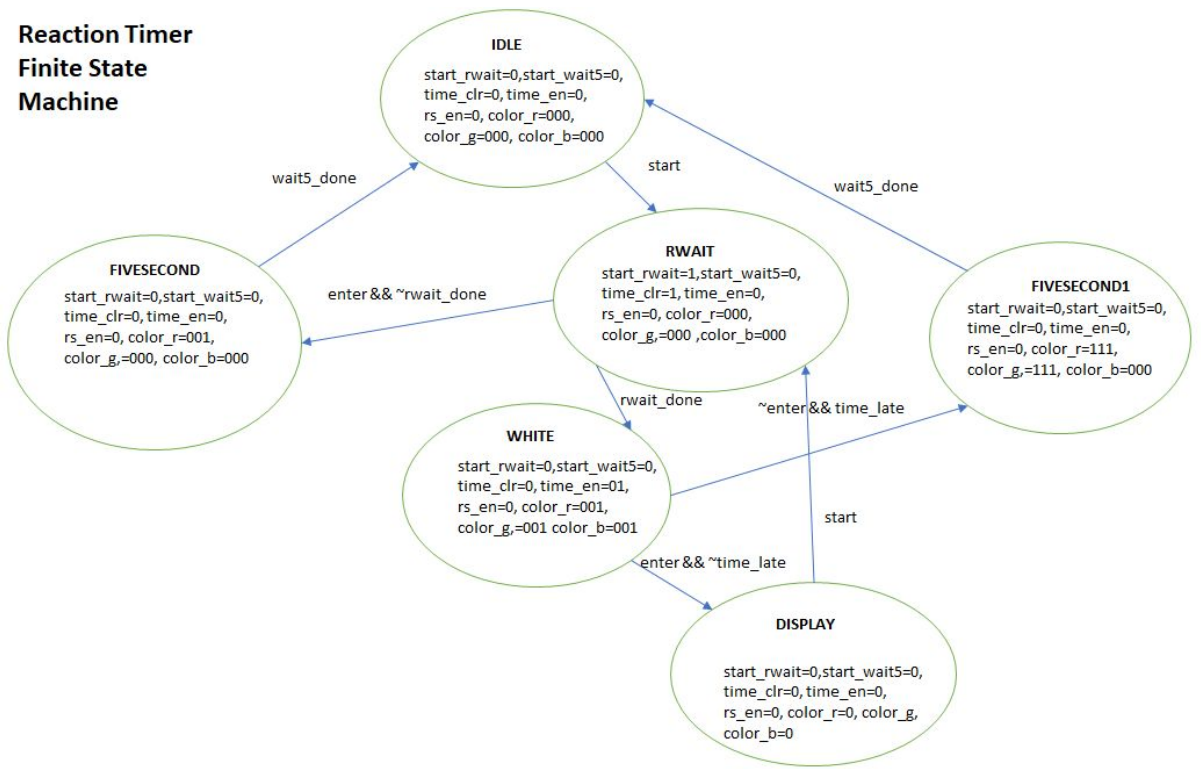
These outputs represent the 4 units of the time count

- **rs_en -** when this output this high, the seven-segment display is triggered to display the result (reaction time)
- **start_rwait -** when asserted high triggers the rcounter module to count up to 7000 since we have assigned the random wait as 7 seconds.
- **rwait_done -** rwait_done logic serves as an input to the FSM of the Reaction timer and is outputted from the r_counter module only when the waiting period is done.
- **start_wait5 -** when asserted high triggers the delay_counter module to count up to 5000 since we have assigned the error wait time as 5 seconds.
- **wait5_done -** wait5_done logic serves as an input to the FSM of the Reaction timer and is outputted from the delay_counter module only when the waiting period is done.
- **p0,p1,p2,p3 -** these four bits are logic signals that are 7 bit long. These are the conversions of d0,d1,d2,d3 in the form of seven-segment representation. They are outputted by the converter module.
- **color_r,color_g,color_b -** these are the 3-bit representation of the three different color components in the RGB. The rgb_pwm module uses these as inputs to assign the output LEDs according to their duty cycle
- **segoutput -** this is basically the seven-segment information bit that is relayed to the output of the project.
- **an11 -** as used in the other main module, we use this 8-bit output to select which anode of the 8 anodes in the display panel are asserted high. For this module, we are just using the first four anodes since our output time is just four digits.

3.2.3 Sub-Module of Reaction_Timer

3.2.2 Reaction_Timer_FSM

This finite state machine is the most essential submodule of the reaction timer since it dictates the order at which outputs are asserted high. The state transition diagram below shows all the states that the reaction timer would be hopping in and out of according to the user's actions.



Reaction Timer Finite State Machine
IDLE
start_rwait=0,start_wait5=0, time_clr=0, time_en=0, rs_en=0, color_r=000, color_g=000, color_b=000
start
wait5_done
wait5_done
FIVESECOND
start_rwait=0,start_wait5=0, time_clr=0, time_en=0, rs_en=0, color_r=001, color_g,=000, color_b=000
enter && ~rwait_done
RWAIT
start_rwait=1,start_wait5=0, time_clr=1, time_en=0, rs_en=0, color_r=000, color_g,=000 ,color_b=000
FIVESECOND1
start_rwait=0,start_wait5=0, time_clr=0, time_en=0, rs_en=0, color_r=111, color_g,=111, color_b=000
rwait_done
~enter && time_late
WHITE
start_rwait=0,start_wait5=0, time_clr=0, time_en=01, rs_en=0, color_r=001, color_g,=001 color_b=001
start
enter && ~time_late
DISPLAY
start_rwait=0,start_wait5=0, time_clr=0, time_en=0, rs_en=0, color_r=0, color_g, color_b=0

Fig 4. State Transition Diagram of the Reaction_Timer_FSM

The initial state is IDLE and the machine always goes back to this state when reset is asserted high. When the start button is pressed by the user, we go to the RWAIT state where we wait for 7 seconds. After this, we go to the WHITE state where the color_r, color_g, and color_b are given some values according to their duty cycle. We stay in this state until the enter button is pressed by the user within the allowed time which is 10 seconds. Once that is done, we move on to the DISPLAY state whereupon the reaction time of the user is displayed on the display panel. If enter is pressed by the user before the random wait period is done, the color_r output is given some value so that only the led_r can be asserted high and there is a red light on the GO lamp. This happens all in the FIVESECOND state. This error period is only for 5 seconds after we hop on back to the IDLE state. When the user does not press the enter button within the 10 second time period, we move on to the FIVESECOND1 state where the color_r and color_g are given some values according to their duty cycle in order to display a yellow color on the GO lamp for 5 seconds.

3.2.3 Time_Count

This module uses the 1000Hz frequency as a clock signal and uses 4 BCD counters to count the time in decimal. Every digit counted is of 4 bits which is outputted to the converter.

3.2.4 Convertor

This module simply converts the 4-bit count values coming from the time_count module into equivalent 7 bit that can be displayed on the seven segment display.

3.2.5 Display

This module uses the 4 seven-bit representations of the time and uses the same principles we discussed earlier to select the specific anode for the time to be displayed. The module also outputs the point which must be asserted high in the very right of seven-segment panel in order to display the time

4. System Validation and Performance

Test	**Expected Output**	**Actual Output**	**Pass/Fail**
Waits for a random Time	Should wait for 7 seconds before the LED goes white	Waits for 7 seconds and makes the LED white	**PASS**
White LED turns on	Should turn on for 10 seconds maximum	Turns on and only turns off while the enter button is pressed	**PASS**
Red LED turns on when an error occurs	Should turn on for 5 seconds	Turns on for 5 seconds when the user presses enter during the random wait period	**PASS**

Yellow LED turns on when an error occurs	Yellow light must be displayed if 10 seconds have passed and the user does not press enter	Yellow light is displayed for 5 seconds	<u>**PASS**</u>

<u>**Test**</u>	<u>**Expected Output**</u>	<u>**Actual Output**</u>	<u>**Pass/Fail**</u>
Time is displayed in 4 digits	The reaction time must be displayed on the display panel and it must be accurate.	Accurate time was measured and compared with a phone stopwatch	<u>**PASS**</u>
Scrolling Text shows GO PARDS	There should be a scrolling text displaying 'GO PARDS' moving at the speed of 1 second.	The moving text saying 'GO PARDS' is seen.	<u>**PASS**</u>
Sw0 is used to select the mode	Used to switch between the operations of the two major modules	Switches between the two selections.	<u>**PASS**</u>
The display is off on RESET and is off when the reaction time does not have to be displayed	Should show the scrolling text when sw0 is high and should show noting initially when sw0 is low.	The display is off when the reaction timer mode is initially selected. Otherwise, it shows the scrolling text.	<u>**PASS**</u>

5. Conclusion

The Reaction Timer/ Scrolling text device is fully operational and reusable as was described along every step of this report. There were some limitations, however, to this project and it could be improved much more to become a better version of itself. The random time is hardcoded but for the device to be fully working with the project specifications, it must be a random wait time. Additionally, when an error occurs, those specific errors and the next steps pertaining to them could also have been displayed on the display panel. Despite all these, the device calculated accurate reaction time and will prove to be very useful for the Athletic Facility of Lafayette College.

6. Appendices

6.2 Specifications

Taken for the ECE 211 Lab 10 Manual:

The specification of the Human Reaction Timer's inputs, outputs, and functions is as follows:

Inputs

• Mode select switch (slide switch SW0)

• Reaction time START button start (pushbutton BUTNC)

• Reaction time STOP button (pushbutton BUTNL)

• System RESET Outputs

• 8-digit seven-segment display (anode_l, segs_l)

• Reaction Timer "Go" Lamp (RGB LED LD17) Operation

• This reaction timer has two different functions: (a) when the mode select switch SW0 is on, it displays a scrolling text on a seven-segment display, and (b) When the mode select switch SW0 is off, it tests the user's reaction time

. • Scrolling Text o When the SW0 is turned on, the seven-segment display has to scroll the text "GO PArdS". o You can also use any of the previously designed modules from the past labs if required

. • Reaction Timer o When the player presses the START button, the seven-segment display should be turned off (if it isn't already). The circuit should then wait for a random amount of time between roughly 1 and 9 seconds. The wait time should be randomly selected from at least eight different delay values in this range. o After the random wait, turn on the GO LED and record the amount of time elapsed before the player presses the

STOP button. The LED should be off except when waiting for the player to press STOP. o Depending on when (and if) the player presses the STOP button, the seven-segment display and LED will display the result of the reaction time test, as follows: If the STOP button is pressed up to 9.999 seconds after the GO LED turns on, the seven-segment display should be turned on and display the reaction time in the format x.xxx (in seconds). The circuit will continue to display this time until the START button is pressed again. If the STOP button is pressed before the GO LED turns on, the seven-segment display should remain off and the LED color should change to red for five seconds to indicate an error. It should remain lit for five seconds after which it should be turned off and the system should return to waiting for the START button to be pressed.

If the STOP button has not been pressed 10 seconds after the GO LED turns on, the seven-segment display should remain off and the LED color should change to yellow for five seconds to indicate an error. It should remain lit for five seconds after which it should be turned off and the system should return to waiting for the START button to be pressed. Additional requirements and constraints

• The circuit must be implemented as a fully synchronous circuit using a 1 kHz clock generated by a clock divider.
• All sequential logic (except the clock divider and single pulser circuits) should include a synchronous reset and be connected to a single master RESET input.
• All storage in the circuit must be implemented using flip-flops - the circuit must contain no latches. To check whether your circuit contains latches, use the Vivado Synthesis Report (or watch for warnings about latch inferences in the "messages" pane).
• The RGB LED should display outputs at a comfortable intensity and all colors should be displayed at approximately equal intensity.
• Unused digits in the 7-segment display should be blank in both modes of operation.

6.3 Code Listings

This section will have included the codes of all the modules:

6.3.1 Scrolling_Text

```
`timescale 1ns / 1ps
```

```
module Scroll_Text(
input clk_100MHz,
input reset,
output a,
output b,
output c,
output d,
output e,
output f,
output g,
output [7:0] an
);

reg [3:0] an1,an2,an3,an4,an5,an6,an7,an8; // registers to
hold the LED values
logic clk_1000Hz;

clkdiv #(.DIVFREQ(10000)) U_CLKDIV( .clk(clk_100MHz),
.reset(1'b0), .sclk(clk_1000Hz) );
clkdiv #(.DIVFREQ(1)) U_CLKDIV1( .clk(clk_100MHz),
.reset(1'b0), .sclk(clk_1Hz) );


reg [3:0] clickcount;

always @ (posedge clk_1Hz)
begin
if(reset)
clickcount <= 0;
else if(clickcount == 8)
clickcount <= 0;
else
clickcount <= clickcount + 1;

end

always @ (*) //always block that will scroll or move the
text. Accomplished with case
```

```
begin
case(clickcount)
0:
begin
an8 = 1; //G
an7 = 3; //O
an6 = 7; //-
an5 = 0; //P
an4 = 2; //A
an3 = 8; //r
an2 = 4; //D
an1 = 5; //S
end

1:
begin
an8 = 5; //S
an7 = 1; //G
an6 = 3; //O
an5 = 7; //-
an4 = 0; //P
an3 = 2; //A
an2 = 8; //r
an1 = 4; //d
end

2:
begin
an8 = 4; //d
an7 = 5; //S
an6 = 1; //G
an5 = 3; //0
an4 = 7; //-
an3 = 0; //p
an2 = 2; //A
an1 = 8; //r
end
```

```
3:
begin
an8 = 8; //r
an7 = 4; //d
an6 = 5; //S
an5 = 1; //G
an4 = 3; //O
an3 = 7; //-
an2 = 0; //P
an1 = 2; //A
end

4:
begin
an8 = 2; //A
an7 = 8; //r
an6 = 4; //d
an5 = 5; //S
an4 = 1; //G
an3 = 3; //O
an2 = 7; //-
an1 = 0; //P
end

5:
begin
an8 = 0; //P
an7 = 2; //A
an6 = 8; //r
an5 = 4; //d
an4 = 5; //S
an3 = 1; //G
an2 = 3; //O
an1 = 7; //-
end
6:
begin
```

```
an8 = 7; //-
an7 = 0; //P
an6 = 2; //A
an5 = 8; //r
an4 = 4; //D
an3 = 5; //S
an2 = 1; //G
an1 = 3; //O
end

7:
begin
an8 = 3; //O
an7 = 7; //-
an6 = 0; //P
an5 = 2; //A
an4 = 8; //R
an3 = 4; //D
an2 = 5; //S
an1 = 1; //G
end

8:
begin
an8 = 1; //G
an7 = 3; //O
an6 = 7; //-
an5 = 0; //P
an4 = 2; //A
an3 = 8; //r
an2 = 4; //d
an1 = 5; //s
end

endcase

end
```

```
localparam N = 3;

reg [N-1:0]count;

always @ (posedge clk_1000Hz or posedge reset)
begin
if (reset)
count <= 0;
else
count <= count + 1;
end

reg [6:0] sseg;
reg [7:0 ]an_temp;

always @ (*)
begin
case(count)//[N-3:N-1])

3'b000 :
begin
sseg =an1;
an_temp = 8'b11111110;
end

3'b001 :
begin
sseg =an2;
an_temp = 8'b11111101;
end

3'b010 :
begin
sseg =an3;
an_temp = 8'b11111011;
end
```

```
3'b011 :
begin
sseg =an4;
an_temp = 8'b11110111;
end

3'b100 :
begin
sseg =an5;
an_temp = 8'b11101111;
end

3'b101 :
begin
sseg =an6;
an_temp = 8'b11011111;
end

3'b110 :
begin
sseg =an7;
an_temp = 8'b10111111;
end

3'b111 :
begin
sseg =an8;
an_temp = 8'b01111111;
end
endcase
end
assign an = an_temp;

reg [6:0] sseg_temp;
always @ (*)
begin
case(sseg)
1 : sseg_temp = 7'b0100000; //to display G
```

```
3 : sseg_temp = 7'b0000001; //to display 0
7 : sseg_temp = 7'b1111111; //to display -
0 : sseg_temp = 7'b0011000; //to display P
2 : sseg_temp = 7'b0001000; //to display A
8 : sseg_temp = 7'b0111001; //to display r
4 : sseg_temp = 7'b1000010; //to display d
5 : sseg_temp = 7'b0100100; //to display S


default : sseg_temp = 7'b1111111; //blank
endcase
end
assign {g, f, e, d, c, b, a} = sseg_temp;

endmodule
```

6.3.2 Reaction_Top

```
`timescale 1ns / 1ps

module reaction_top_test(input logic clk_100MHz, input logic
reset,input logic start,
input logic enter , output logic [7:0] an11, output logic
[6:0] segoutput,output logic led_r,led_g,led_b, output logic
point);
clkdiv #(.DIVFREQ(1000)) U_CLKDIV( .clk(clk_100MHz),
.reset(1'b0), .sclk(clk_1000Hz) );
logic [6:0] p0,p1,p2,p3;
logic start_in;
logic enter_in;
logic reset_in;
logic rs_en;
debounce
startbutton(.clk(clk_100MHz),.pb(start),.pb_debounced(start_i
n));
debounce
enterbutton(.clk(clk_100MHz),.pb(enter),.pb_debounced(enter_i
n));
```

```
debounce
resetbutton(.clk(clk_100MHz),.pb(reset),.pb_debounced(reset_i
n));
reaction_top RT(.clk(clk_1000Hz),.start(start_in),
.RESET(reset_in),
.enter(enter_in),.led_r(led_r),.led_g(led_g),.led_b(led_b),.r
s_en(rs_en),.p0(p0),.p1(p1),.p2(p2),.p3(p3));
display
dt(.clk(clk_1000Hz),.reset(reset_in),.rs_en(rs_en),.p0(p0),.p
1(p1),.p2(p2),.p3(p3), .an1(an11),
.sseg(segoutput),.point(point));
ndmodule
```

6.3.3 Reaction_Timer_FSM

```
`timescale 1ns / 1ps

module Reaction_Timer_FSM(input logic clk,start, RESET,
enter, rwait_done, wait5_done, time_late,
output logic start_rwait, start_wait5, time_clr, time_en,
rs_en,
output logic [2:0] color_r, color_g, color_b
); //module description

typedef enum logic [2:0]{

IDLE=3'b000,RWAIT=3'b001,WHITE=3'b010,FIVESECOND=3'b011,DISPL
AY=3'b100,FIVESECOND1=3'b110
} state_t;
state_t state, next_state; //states declaration

always_ff @(posedge clk)
begin
if (RESET) state <=IDLE;
else state <= next_state;
end
//states condition
always_comb
```

```
begin
//initial outputs
start_rwait <=1'b0;
start_wait5 <=1'b0;
time_clr <=1'b0;
time_en <=1'b0;
rs_en <=1'b0;
color_r <=3'd0;
color_g <=3'd0;
color_b <=3'd0;
next_state = IDLE;

//CASES
case(state)
IDLE:
begin

start_rwait <=1'b0;
start_wait5 <=1'b0;
time_clr <=1'b0;
time_en <=1'b0;
rs_en <=1'b0;
color_r <=3'd0;
color_g <=3'd0;
color_b <=3'd0;

if (start)
next_state = RWAIT;
else
next_state = IDLE;
end

RWAIT:
begin
start_rwait <=1'b1;
start_wait5 <=1'b0;
//hhjvuvyuc
time_clr <=1'b1;
```

```
time_en <=1'b0;
color_r <=3'd0;
color_g <=3'd0;
color_b <=3'd0;
rs_en <=1'b0;

if (rwait_done)
next_state = WHITE;
else if (enter && ~rwait_done )
next_state = FIVESECOND;
else next_state=RWAIT;
end


WHITE:
begin
start_rwait <=1'b0;
start_wait5 <=1'b0;
time_clr <=1'b0;
time_en <=1'b1;
color_r <=3'b001;
color_g <=3'b001;
color_b <=3'b001;
rs_en <=1'b0;

if (~enter && time_late)
next_state = FIVESECOND1;

else if (enter && ~time_late)
next_state = DISPLAY;
else next_state=WHITE;
end

FIVESECOND:
begin
start_rwait <=1'b0;
start_wait5 <=1'b1;
time_clr <=1'b0;
```

```
time_en <=1'b0;
color_r <=3'b111;
color_g <=3'd0;
color_b <=3'd0;
rs_en <=1'b0;

if (wait5_done)
next_state = IDLE;
else next_state=FIVESECOND;
end

FIVESECOND1:
begin
start_rwait <=1'b0;
start_wait5 <=1'b1;
time_clr <=1'b0;
time_en <=1'b0;
color_r <=3'b111;
color_g <=3'b111;
color_b <=3'd0;
rs_en <=1'b0;

if (wait5_done)
next_state = IDLE;
else next_state=FIVESECOND1;
end

DISPLAY:
begin
start_rwait <=1'b0;
start_wait5 <=1'b0;
time_clr <=1'b0;
time_en <=1'b0;
color_r <=3'd0;
color_g <=3'd0;
color_b <=3'd0;
rs_en <=1'b1;
```

```
if (start)
next_state = RWAIT;
else next_state=DISPLAY;
end

endcase
end
Endmodule
```

6.3.4 Time_Count

```
`timescale 1ns / 1ps


module time_count(input logic clk, time_clr,time_en,  output
logic [3:0] d0,d1,d2,d3, output logic time_late );

logic w0,w1,w2,w3; //inner logic
bcd_counter
D0(.clk(clk),.rst(time_clr),.enb(time_en),.q(d0),.carry(w0));
bcd_counter
D1(.clk(clk),.rst(time_clr),.enb(w0),.q(d1),.carry(w1));
bcd_counter
D2(.clk(clk),.rst(time_clr),.enb(w1),.q(d2),.carry(w2));
bcd_counter
D3(.clk(clk),.rst(time_clr),.enb(w2),.q(d3),.carry(w3));
//case when it takes more than 10 seconds
always_comb
if ( ~time_en)
begin
time_late=0;
end
else if (w3 && time_en)
time_late=1;


endmodule
```

6.3.5 Convertor

```
`timescale 1ns / 1ps
module convertor(input logic [3:0] d0,d1,d2,d3, output logic
[6:0] p0,p1,p2,p3);//ins ds outs are ps
localparam [6:0]
zero = 7'b0000001,
one = 7'b1001111,
two = 7'b0010010,
three = 7'b0000110,
four = 7'b1001100,
five = 7'b0100100,
six =  7'b0100000,
seven = 7'b0001111,
eight = 7'b0000000,
nine = 7'b0001100;

always_comb
begin
case(d0)
4'b0000 : p0 = zero; //0
4'b0001 : p0 = one; //1
4'b0010 : p0 = two;//2
4'b0011 : p0 = three;//3
4'b0100 : p0 = four;//4
4'b0101 : p0 = five;//5
4'b0110 : p0 = six;//6
4'b0111 : p0 = seven;//7
4'b1000 : p0 = eight;//8
4'b1001 : p0 = nine;//5
endcase

case(d1)
4'b0000 : p1 = zero; //0
4'b0001 : p1 = one; //1
```

```
4'b0010 : p1 = two;//2
4'b0011 : p1 = three;//3
4'b0100 : p1 = four;//4
4'b0101 : p1 = five;//5
4'b0110 : p1 = six;//6
4'b0111 : p1 = seven;//7
4'b1000 : p1 = eight;//8
4'b1001 : p1 = nine;//5
endcase


case(d2)
4'b0000 : p2 = zero; //0
4'b0001 : p2 = one; //1
4'b0010 : p2 = two;//2
4'b0011 : p2 = three;//3
4'b0100 : p2 = four;//4
4'b0101 : p2 = five;//5
4'b0110 : p2 = six;//6
4'b0111 : p2 = seven;//7
4'b1000 : p2 = eight;//8
4'b1001 : p2 = nine;//5
endcase


case(d3)
4'b0000 : p3 = zero; //0
4'b0001 : p3 = one; //1
4'b0010 : p3 = two;//2
4'b0011 : p3 = three;//3
4'b0100 : p3 = four;//4
4'b0101 : p3 = five;//5
4'b0110 : p3 = six;//6
4'b0111 : p3 = seven;//7
4'b1000 : p3 = eight;//8
4'b1001 : p3 = nine;//5
endcase
end
```

```
endmodule
```

6.3.6 Display

```
`timescale 1ns / 1ps


module display(input clk, input logic reset,rs_en,input logic
[6:0] p0,p1,p2,p3, output logic [7:0] an1, output logic [6:0]
sseg, output logic point);
logic [1:0] q;
k_counter(.clk,.reset(reset),.rs_en(rs_en),.q(q));
seven_seg
ss(.p0(p0),.p1(p1),.p2(p2),.p3(p3),.q(q),.segs_1(sseg),.point
(point));
dec_3_8 dc (.q(q),.rs_en(rs_en),.y_1(an1));


Endmodule
```

RGB_PWM

```
`timescale 1ns / 1ps

module rgb_pwm(input logic clk, rst, input logic [2:0]
color_r,color_g,color_b,
output logic  led_r, led_g, led_b );
logic [3:0] q;

always @(posedge clk)
begin
if (rst)
q<=0;
else
q<=q+1;
end

assign led_r = (q < {color_r});
assign led_g = (q < {color_g});
assign led_b = (q < {color_b});



endmodule
```

6.3.7 Delay_Counter

```
`timescale 1ns / 1ps


module delay_counter( input logic clk,rst, start_wait5,
output logic wait5_done);
logic [12:0] q;
always @(posedge clk)
if (rst)
begin
q<=0;
wait5_done<=0;
end
else if (start_wait5)
begin
q<=q+13'd1;
if (q==13'd5000)
begin
q<=0;
wait5_done<=1'b1;
end
else
wait5_done<=1'b0;
end

endmodule
```