University of Canterbury

ENEL373 – Digital Electronics and Devices

Reaction Timer Project

Submitted by:

Benjamin Bush - 11930514

Connor Grant - 13367470

Rogan Ross – 54254447

Group:

RxnFriGroup7

Abstract

This project aims to develop a reaction timer within VHDL to be implemented the NEXYS-4 DDR FPGA board. The timer display s a countdown on the 7-segment displays and records the user's reaction in milliseconds. If the button is pressed too early, an error message is shown. The user can also choose to display the minimum, maximum, and average values of the past three times. The project had extensive amounts of testing and verification to ensure that the design resulted in the desired product.

Contents

1. Introduction

The aim of this project was to develop a reaction timer using a VHDL programmed NEXYS-4 DDR FPGA development kit. The reaction timer is to be designed to display a user prompt of three decimals counting down, utilising the 7 segment displays to indicate when the user should react. Each decimal should vary in how long it would take to turn off. The display would then show the timer counting in milliseconds. Upon pressing the centre button, the reaction time was to then be recorded and displayed on the 7-segment display. If the centre button were to be pressed before the last decimal point turned off an error message should be displayed. This process should be repeated when reset by the centre button. After multiple reactions, the minimum, maximum and average reactions should be displayed by use of the buttons.

Our team successfully developed a VHDL program that met all core specifications and added an additional feature of randomised starting LED countdown times. To achieve this, the team undertook extensive testing to refine the program and ensure it met the specifications.

2. Top-Level Block Diagram

The design of the reaction timer consists of 10 main modules that implement all the necessary functionality.

1. Clock Divider:
The clock divider is designed to take the boards internal clock and divide it down to the desired clock frequency by calculating and setting the required upper bound.

2. Display Selector:
The display selector is to take in a set clock rate and count from 0 to 7 outputting the full counter value. It is to be used to select which 7-segment should be lit up.

3. Anode Decoder:
The anode selector module is a combinational decoder used to provide power to one of the eight 7-segment displays anode dependent on which number is provided from the 3-bit counter input.

4. Cathode Decoder:
A cathode selection decoder is utilised to switch on the required segments in order to display a number, decimal point or other value corresponding to the binary coded decimal value inputted.

5. Decade Counter:
The decade counter module is used to count a decade (0-9). Once this decade is counted the module toggles and resets to count again. This module is critical for implementing the counting needed to measure reaction times.

6. MUX:

A Multiplexer module was implemented into the project to select the desired message for display on each digit of the BCD.

7. ALU Module:

The Arithmetic Logic Unit performs all the mathematical calculations needed to provide minimum, maximum, and average reaction times. This is a critical module to be able to perform the necessary reaction time statistics.

8. PRNG Module:

The PRNG begins with a seed and uses a linear feedback shift register in conjunction to generate new serial bits. This will then output a random sequence of bits.

9. Shift Register Module

The shift register consists of 3 different stages that each store a 16-bit vector. It loads a new value in when shift is enabled and can be reset.

10. Finite State Machine (FSM)

The finite state machine is to provide the countdown states, displaying dots corresponding to the countdown, as well as the timing and reaction time. The machine resets back to the countdown states from displaying reaction time state. A design of the finite state Machine can be found in Appendix B.

The top level integrates all the modules, where some are used multiple times so that the ENEL373 reaction time project can be implemented. A diagram of the Top-Level Block Design is included in Appendix A.

3. Expanded Design Summary

To create a working reaction timer FPGA design, the design had to meet all the desired specifications [1], be reliable against any edge cases, and be simple to understand. To achieve these goals a plan had to be created before any code was created, good VHDL design practises were followed to ensure that the design was reliable and efficient.

Before beginning any code writing, a top-level block diagram was created. This diagram would provide visual aid in understand how each module would connect to each other and made the functionality of each module clearer. Each module would provide essential behaviour for the reaction timer, and when connected would create a working reaction timer. This modular style of coding makes the development process easier as each module can be created and tested on their own without worrying about excessive moving parts.

To ensure reliability there were a few good practises considered so that the hardware in the FPGA performed reliably. First off, each process was controlled by a clock signal so that the programs tasks would happen sequentially. This allowed everything to change at the same time so that the behaviour can be controlled, and no steps are missed. It is important not to gate this clock signal to mitigate missed signals.

Secondly, flip flops were used instead of latches. Latches are controlled when the clock signal is high, which can lead to a large amount of transparency. Transparency can lead to missed signals that happen within the transparent regions. Flip flops use the clocks rising edge which reduces the transparency as much as possible.

Thirdly, named association was used when defining the port map. This is better practise than to used positional so that each signal connection is done correctly. And lastly, every signal that could “trigger” a process was listed within the sensitivity list.

For the code to be understandable, the ports, signals, variables, blocks, and programs must have meaningful names. Having purposeful names provides many benefits such as making the code easier to read and understand for both the developer and team members. When encountering issues, it makes it clearer what the purpose of each part of the code is based on their names, which in turn reduces the time required to understand the code when needing to debug or develop further. It also provides indirect documentation which reduces the need for a lot of comments within the code. Improving code quality makes sure the ideas communicated clearly, speeding up the coding development.

When working as a group in a software project, it is especially important to uphold code quality and use of modules. Each member in the team made sure every part of the code was clear so that other members could work on, use, test and debug the code. The use of modules also made splitting up the workload easier as each member was delegated a few modules each. These methods along with a clear plan meant the team could be on the same page for the project and made the development of the project efficient.

The project began with focusing on the modules needed for milestone one. Each module was tested using a testbench to ensure that they performed correctly. Each module had small tasks

to perform making it easier to write the modules functionality. The modules used the behavioural VHDL style. This language style worked well for the individual modules as behavioural coding describes the behaviour of how the module should be perform. This style uses sequential processes triggered by signals to determine the functionality. These modules were then joined together in a top-level file. The top level used structural VHDL style. This VHDL style is used to connect signals together using a port map. The signals are used to connect internal parts of the design like a wire.

When choosing upper bounds for the clocks, the counter clock needed to have a frequency of 1000Hz so that each tick was equal to 1 millisecond. The display clock needed to be more than the refresh rate of a human eye which has a maximum of 90Hz. Because there were 8 segment displays, the display clock needed to be set to at least 90Hz × 8 = 720Hz. After some testing, it was found that though 720Hz work well enough for the whole display to shown at the same time, using a frequency of 1000Hz made the display look less "glitchy." For the pseudo random number generator as very fast clock of 1000Hz was used to make the number generation more random as the clock could cycle through a lot of different random numbers before choosing one.

Once the milestone was completed there were more specifications needing to be met for the final design. These specifications were to be integrated in project created thus far. This process of starting with a simpler design then adding more complex parts made the development process easier to envision as it started with a base that could be added on to.

For the extension specification, the group decided on making the time between decimal points turning off to vary randomly. This behaviour would mitigate the user's ability to pre-empt when the reaction timer would begin making the reaction timer test more accurate. This specification was chosen as it made intuitively more sense to the group to implement and add to the design.

For the random number module, its behaviour would be to output a random number that could be used as an upper bound for the millisecond clock to count to. The process to create this was to research how random numbers could be generated. From the research, pseudo-random number generators (PRNGs) are a typical way of creating random numbers in hardware. The PRNG created used a linear feedback shift register, with four of the flip flops in the shift register connected to some XOR gates for a new serial bit to be generated and shifted into the register [2]. A diagram of the pseudo random number generator can be found in Appendix C. The PRNG outputted a 9-bit sequence meaning that it could output a random number ranging from slightly higher than 0 to slightly less than 512. The PRNG begins with a seed of "000010001" so that it can begin with a random number and does not allow the output to be 0 because it would remain at zero without any bits as "1" and cannot reach 512 due to the use of XOR gates. A lower bound of 480 was added so that the dots were not turned off to quickly, making the random times range from roughly 500 to 950 milliseconds. This range made the random times noticeably different, and all fit in a range that was not too fast to react to, but not too slow that the user was waiting too long.

The final design of the reaction timer FPGA project embodies a robust, reliable, and user-centric solution that meets all the specification. Through thoughtful planning, a lot of verification, and adherence to best practises in VHDL design, the design fulfils its intended purpose.

4. Module Testing

The pseudo random number generator required a testbench to confirm its functionality. The testbench was designed to test the pseudo-number generator's ability to produce random values within a range of slightly more than 0 and less than 512. The test bench toggles the shift register clock, causing each bit to shift by one and a newly generated serial bit is appended to the start of the shift register. The test bench did indeed prove that numbers were being changed and varied in randomness. The testbench successfully running can be found in Appendix E.

5. Conclusion

In conclusion, a Reaction Timer was able to be developed by the team on the NEXYS DDR4 development board. The timer successfully prompts a user, records their reaction time, provides a range of statistics relative to the times and challenges them by prompting them with random countdown intervals. Therefore, meeting all specified criteria.

During the project, there were a few issues that needed to be fixed. For example, the program would skip straight to the error state when the centre button was pushed to reset the reaction timer. This was due to switch debouncing, and the button pushed signal carried over during the warning_3 state, triggering the error message. Initially the idea to fix this was to create a switch debounce in conjunction with an edge detector so that the signal would not carry over. After some review the idea seemed to be too complex, and a simpler solution would be to create a buffer state to wait for the switch to become “0.” This would create even more randomness to the time 3 decimal points were shown as the time the mechanical switch stops bouncing is random.

The project overall was well run and gave clear instructions on what needed to be completed. The improvements the group suggested would be to provide more intuitive examples on how VHDL works, and to explain at the beginning how the project could be better split up between group members.

6. References & Appendices

[1] Department of Electrical and Computer Engineering and University of Canterbury, Eds., “ENEL373: Reaction Timer Project.”

[2] “Pseudo-Random Number Generation Routine for the MAX765x Microprocessor,” *Analog.com*, 2024. https://www.analog.com/en/resources/technical-articles/pseudorandom-number-generation-routine-for-the-max765x-microprocessor.html (accessed May 16, 2024).

Appendix A:

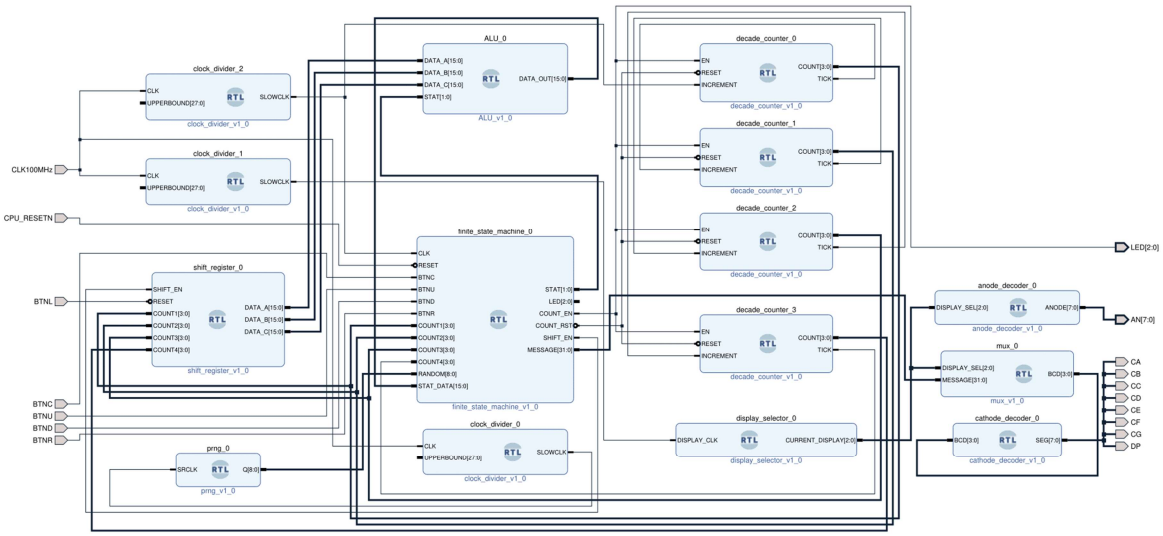

Figure 1. *Top-Level-Block Diagram*

Appendix B:

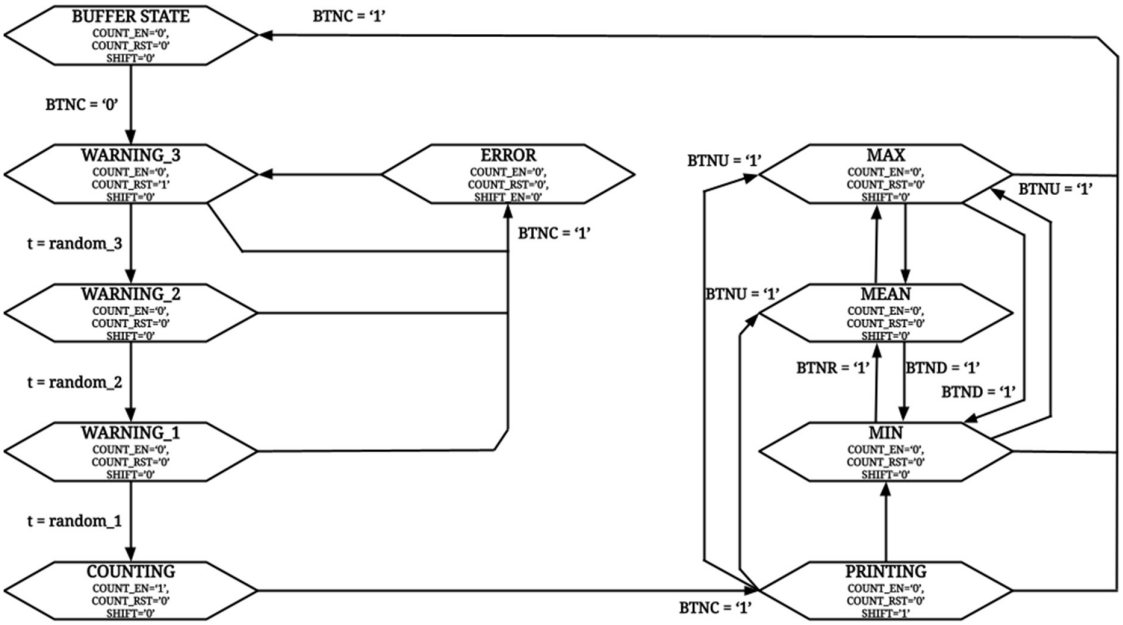

Figure 2: *Finite State Machine Diagram*

Appendix C:

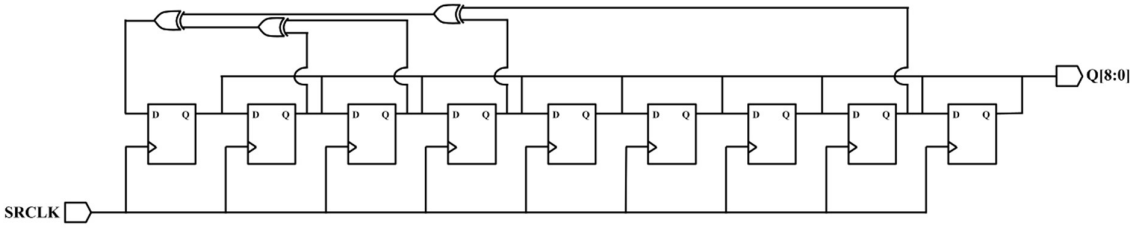

Figure 3: *Pseudo random number generator circuit*

Appendix D:

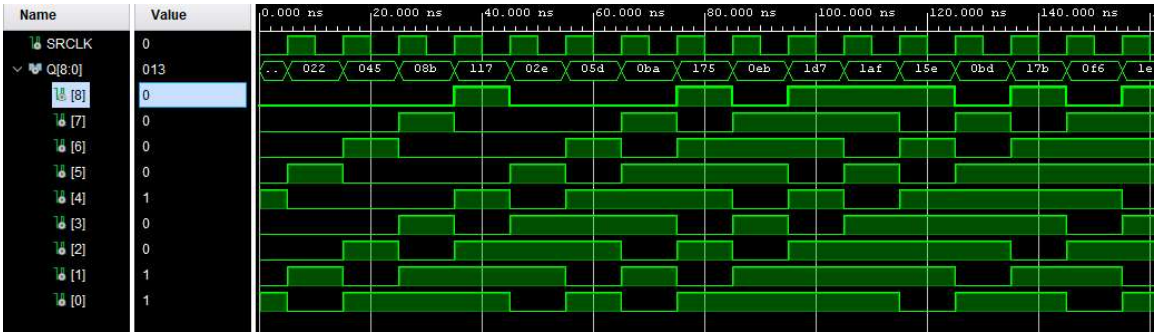

Figure 4: *Pseudo random number generator test bench*

Appendix E:

```
----------------------------------------------------------------------------
-------
-- Create Date: 06.05.2024 21:19:10:
-- Module Name: prng_tb - Behavioral
-- Revision 0.01 - File Created
----------------------------------------------------------------------------
-------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity prng_tb is
end prng_tb;

architecture Behavioral of prng_tb is

signal SRCLK : std_logic := '0';
signal Q : std_logic_vector(8 downto 0); -- Output

begin

SRCLK <= not SRCLK after 5ns;            -- toggles clock every 5
nanoseconds

inst_prng : entity work.prng(Behavioral)
port map(SRCLK => SRCLK, Q => Q);

end Behavioral;
```

***Code Snippet 1**: Pseudo Random Number Generator Testbench*