UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

ENEL373: Digital Electronics and Devices

University of Canterbury

This is an Awesome Title for Our First Report

Philip Brand *(15776928)*
Michael Brown *(48571923)*
Boston Black *(24668421???)*

May 5, 2025

Contents

1 Introduction

Human reaction time is the interval between the a stimulus and the response to that stimulus. There are two main types of reaction time; simple, and choice. Simple reaction involves reacting to a singular stimulus, while choice reaction involves choosing the correct response from multiple stimuli. This project aims to implement a simple reaction time recorder on an FPGA in VHDL.

The reaction stimulus test starts with three LEDs that turn off sequentially, with a random delay between each LED turning off. Once the last LED turns off, the user must press a button, and the delay is displayed in milliseconds on 8 7-segment displays. The FPGA stores the reaction delays from the last three trials, and can display the fastest, slowest, and average reaction times. These values can be reset by the user.

2 Design Summary

The FPGA reaction stimulus test depends on four major components, those being the 8-digit counter, the ALU, the reaction countdown, and the output select and override. These components are shown in Figure 1. Each of these components are enabled and disabled via a finite state machine.

Figure 1: Overview of VHDL reaction timer structure

An 8-digit 5-bit-BCD counter is the primary mechanism used to track the reaction time from each stimulus test to the nearest millisecond. Each BCD digit is passed into an 8x5-to-5 mux. The purpose of this mux

is to sync the selected output digit with the currently active seven segment display anode, outputting each digit onto its associated display.

All the BCD digits from the 8-digit counter are passed directly to ALU module. This module converts the BCD numbers into a 10-bit binary number, then stores the last three reaction times in a circular buffer. This buffer is read by the ALU which calculates the average, minimum, and maximum of the last three reaction times. These times are converted back into BCD, and another 8x5-to-5 mux is used to select which digit is output.

A linear shift register is used to provide some randomisation for the countdown sequence at the start of every reaction stimulus test. This time randomisation acts as the trigger for the countdown state, Dotiey, and also has an 8x5-to-5 mux for digit selection.

All three 8x5-to-5 muxes from the three components described above are piped into an 8x5-to-5 mux used for output selection. This mux selects which module to take output from depending on what state is active. The mux output is piped into a text override, which overwrites some of the digits with letters based on state. This is finally piped into a 7 segment decoder, which takes a 5-bit BCD digit and converts it to a 8-bit seven segment display light combination which is piped to hardware.

3 Expanded Design Summary

3.1 Design Methods

The primary focus for the reaction timer project project was to develop VHDL code that is easy to maintain and develop. This led to a heavy focus on modular design and code reuse. Since the project was developed as a group, the other focus was for individual components to be developed and tested by individually, and then put together with minimal issues.

3.1.1 Diagramming

The primary program development method used in this project was to draw program or component flow diagrams before VHDL was written. These digrams detailed the communication between between individual modules, and outlined the overall program structure. Figure 1 shows one such digram outlining each major component in the design and how they're grouped into larger modules.

Clear communication between group members of the expected inputs and outputs from each component played a significant role in keeping the final assembly simple. Examples include the consistent transfer of characters to be displayed 8x5-to-5 muxes to transmit one character at a time unless otherwise required, such as for the ALU.

Philip - Michael, do you want to cover some of this?

3.1.2 Naming and Syntax Conventions

Another important aspect of the project ensuring high code quality, readability, and maintainability was strict rules around naming and syntax. This included consistent whitespace, indentation, case, and naming style. For example, all signals within the entity was postfixed with an underscore and whether that signal was an input or output. This can be seen in Listing 2 in Appendix A. This ensured when the components were used in the top-level Behavioural architecture with port mapping, it was easy to see which signals were inputs and outputs.

Add more examples here ...

3.1.3 Component Reuse

In order to reduce the number of components that needed to be created and maintained, components were reused where possible.

The most versitile example of this was the 8 channel 5 bit mux. The mux took up to 8 5-bit BCD inputs, and selected one of those inputs to be the output. The select lines and number of channels for the mux corresponded the segment display, allowing both to be controlled with the same signal. This capability made it easy to use for displaying timer digits and error message letters. The same mux was also used to select what output to send throught to the display depending on the state. While not all the channels of the mux were used in all usecases. It stil made sense to use the same component as it was proven to work and integrated well with the rest of the system.

3.1.4 Programming Techniques

Another method used to implement the FPGA design was pair programming. This had to be done with care, as since the project was done in three-person groups, it would have been easy to leave one person out of the process and result in a lack of their understanding of the code. However, pair programming did help ensure that minimal time was spent debugging, as the observer frequently caught subtleties of VHDL that the programmer missed. Pair programming was used to develop the circular buffer, counters, and FSM.

Since the project the significantly modularised, it was important that individual components could be developed and tested without hindering the development of other components. The method used to achieve this was to use git branching and merging. Individual components were developed on separate branches, tested, and then merged into the main branch. This ensured that the main branch was always stable, and that broken changes could be easily rolled back.

3.2 Design Justification

1. it did what it was meant to do
2. it was not coded into an unchangeable spaghetti
3. it was incredibly modular

4 Module Testing

Philip - I believe this was going to the random number generator and then we can discuss maximal outputs? (to do on Tuesday?)

5 Design & Implementation Problems

1. Steven Weddell
2. Vivado
3. Vivado
4. Vivado
5. VHDL
6. Steven Weddell
7. Steven Weddell
8. Vivado
9. Vague Error messages
10. Decimal point issue
11. Randomisation delay insufficient
12. Not case sensitive code
13. Steven Weddell
14. Vivado

One issue that arose during the development of the seven segment display module which mapped a BCD input to a combination of segments on a display to represent that BCD number was that the provided mappings did not represent the numbers correctly. The solution approach was to first determine whether the BCD input or mapping was incorrect. This was done by tying the BCD input to the green LEDs on the FPGA. This asserted BCD input was correct. To rectify the incorrect mappings, individual segments on a board were tied to the FPGA user switches. Combinations of switches were tested, and when the segments represented a number, the combination of switches was recorded as a mapping.

However, once all the mappings were implemented, we found that our countdown timer failed to display the dots. Alteration of the countdown timer to display temporary digits instead of dots revealed the countdown to be functional. Stepping through the code by hand revealed that, due to the use of the unfamiliar provided seven segment decoder, a line of code was overlooked that overwrote the decimal place value. Disabling this line of code allowed the countdown timer to show dots.

A significantly unexpected issue that appeared during the testing of the finite state machine was deemed as the "Schrödinger's States" problem. This originated from the FSM running lines of code designed to change states even while they were commented out. In addition, an unwanted state change wouldn't necessarily complete, instead running half of a state's instructions before reverting back to the original state. This was fixed by removing the FSM inputs from the sensitivity list of the FSM process, instead tying it to a 10 kHz clock instead. CONFIRM THE CLOCK SPEED.

There were issues with the simulation software built into the Vivado toolchain. During the design of the BCD-to-Binary numeric converter, the Vivado simulator would stop the simulation at the beginning of the numeric conversion, but would not give an error message to indicate an issue. However, generating a bitstream and running it on hardware led to no such issues with the BCD-to-Binary module, and the numeric conversion operated as intended.

Another issue that arose was regarding case sensitivity in VHDL. Although VHDL is not case sensitive, the constraints file the VHDL interfaces with is. Early on during the project, while developing the clock divider, an error message "insert error message here" was encountered. The approach to solving this problem was to ask the TAs in the lab what the message meant. However, they were unable to explain the meaning of the message. From there, the clock divider code we wrote was thoroughly compared to the provided clock divider code, and every small difference tested to see if it repaired the error. This determined the issue to be the case of the clock input declared in entity of main module, as lowercase had been used rather than uppercase. This correct case can be seen in Listing 2.

6 Conclusions

7 Appendix A: Code Listings

```
-- Define module IO
entity counter_decade is
    Port ( EN_IN : in STD_LOGIC;
           RESET_IN : in STD_LOGIC;
           INCREMENT_IN : in STD_LOGIC;
           COUNT_OUT : out STD_LOGIC_VECTOR (3 downto 0);
           TICK_OUT : out STD_LOGIC);
end counter_decade;
```

Code Listing 1: Entity naming convention example.

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

entity main is
    Port (  CLK100MHZ : in STD_LOGIC; -- This must be capitalised.
            -- This, and the rest, must be too.
            AN : out STD_LOGIC_VECTOR (7 downto 0) := X"00";
            SEVEN_SEG : out STD_LOGIC_VECTOR (7 downto 0) := X"00";
            BTNC : in STD_LOGIC;
            BTNR : in STD_LOGIC;
            BTNL : in STD_LOGIC;
            BTNU : in STD_LOGIC;
            BTND : in STD_LOGIC);
end main;
```

Code Listing 2: Capitalised port names to avoid errors.