Coursework Group 002 Report

High-Level overview of implementation

Decodable characters: [A-Z0-9] - Decodes standard Morse Code

There are three components to this program: morse decoding, driving the seven-segment display, and general program intialisation and control. We attempted to define the inter-component API early so that we could easily work on multiple components in parallel.

The morse component defines a structure which represents a morse code character in the process of being received, as well as useful methods to manipulate them. Specifically, creating a new blank morse code character, adding a signal to the character, and then decoding the character. We made used of the three most significant bits to store the length of the morse character and used the 5 least significant bits to store the value of morse character. This allowed the program to be concise while staying efficient.

Driving the seven-segment display is another facet of the program, with a simple API. We decided to only expose a single method, which puts a character onto the display. This way, all the complex logic is abstracted away, and the control component needs to only call a single function. The function makes use of an array that consists of binary mappings of LEDs to be turned on in the 7-segment display. Given a character input the function maps it to the correct index position in the array. The value of this index is then used to light the required LEDs on the display.

The control component calls into both the other components and performs the timing logic. It counts the number of milliseconds since the button input changed state, and depending on the state and length of time, either adds a signal to the in-progress morse code character, or decodes it and uses the functions exposed from the seven-segment display component to show it to the user.

We tried to maintain clear boundaries between the different logical components to improve several important attributes: readability, since there are limited ways of interacting with the components, maintainability, since there are no side effects between components, and efficiency of development, as we could work on each component on the same time without requiring other work to be finished.

Main challenges encountered during implementation

The representation of the morse code characters had the most discussion as we decided to go somewhat overboard! Following the discussion, it was decided that the inputs would be stored in a binary format. Initially, we used a structure that stored both the signals and the length in separate bytes, but later we realised that the most significant bits of the signals were being wasted, and so we encoded the length and signal into a single byte as this was more storage efficient. However, the trade-off with this method is that, if given too many inputs, an overflow will occur. Essentially, we halved the number of bytes needed to encode a character at the expense of some processing time to extract just the signals or length.

Another challenge was to do with the syntax of C. In the function to display a character to the seven-segment display, there's a check to see if the input character is a hyphen to display that, however we forgot to use the double equals sign to check for equality, and so instead had an assignment in the if

condition. This led to a strange bug where the if condition was always true, and so a hyphen was always displayed. This meant that some characters, specifically clearing the display would instead show a hyphen.

Contributions of group members

Muhammad developed the morse decoding component, creating the logic for both accepting signals into a morse code character and then later decoding them into a standard char. Ganesh developed the code which drove the seven-segment display, creating the logic which sets the LEDs on the display based on the input character. Leo handled the main control flow, the logic which handles the timings of the button presses and calls into the other components.