COM1031 | Morse Code Logic

High-Level Implementation Overview

Our initial goal within the group was to define the pins and then describe the alphabet underneath directives .text and .global main to display the alphabet to the 7-segment. We did this by defining a function for each letter and using pins alongside an output-input function, both defined within the library gpiolib.s underneath the .data directive. This enabled us to turn the 7-segment display on and off and soon after, print the welcome message to the terminal after the display was turned off and print an error message if the segment failed to turn on.

We defined a function *convert* that looks at r6 and branches to our functions *length(1-4)* to allow the morse code that we defined to be converted and then reset r4, r5 and r6 using the *reset_counters* function and return to where *reset_counters* was called.

We decided it was best to follow the pseudocode top to bottom and write functions after they had been called. After initializing the button our plan was to use registers r4 and r5 as counters to keep track of the time when the button is pressed and not pressed respectively, and register r6 as a buffer to track the number of dots and dashes inputted using the button. To achieve this, a *loop* function was created to read the value within r4 and r5. We incremented the registers by 1 or resetted to 0 to allow the program to check whether to call the function *check_button* or loop back to the beginning and call the *convert* function by comparing the value in r5.

However, we realized this could not be done without defining the *convert and check_button* functions as we followed the pseudocode top to bottom.. The group initially struggled in defining *check_button* as we were unsure how to create a cmp statement that matched a similar result to

the pseudocode, but managed to conclude that we had to use r8, r9, r10 and r11 to store the dots and dashes in the correct order, rather than using a buffer, where #0 means a dot and #1 means a dash so that they can be used in later functions *length(1-4)*.

To interact with the display and print the morse code, the code had to load the pin and read the value of the button. If r0, #0 the button has been pressed which increases counter r4 and resets r5. If not equal, r5 is incremented and branches to the *check_button* function. If r5 is greater than 0, call the *convert* function.

To encode the morse code, we used functions *length(1-4)* to map letters to their combinations of morse. The length functions will compare to a register r8, r9, r10 and r11 and use the set_letters *(set_(a-z))* function to allow them to be displayed on the display and terminal.