# COM1031 COURSEWORK

Camden Jones – 6809406

Rojin Kara – 6830924

## High Level Overview

Our morse code decoder can represent characters A-Z and 0-9.

The program starts by initiating all the global variables. The variables used for timing are initiated here so they can be accessed throughout the whole program. A character array of length 99 is created to store the inputs by the buttons. The program then outputs “HELLO” to the terminal and shows the middle bar on the seven-segment display for 2 seconds. A Boolean called “toDecode” is also defined here. This is so it doesn’t get reset each time the while loop iterates.

In the “while (true)” loop, the time is constantly defined. While the key is held, a while loop is run within the original loop. This causes the overriding of the time to be stopped. It also shows the “- “on the display while held and sets toDecode to true. This makes sure the decoder function is only run once.

The time is then calculated once you let go of the button as it then exits the inner while loop. If the time is between 250ms and 50ms, it is detected as a short press and a 1 is sent into the checkButton function. If the time difference is greater than 700ms, an 8 is shown on the display for 200ms and the character array is reset. If the time is greater than 250ms but less than 700ms, it is detected as a long press and checkButton is called with the value of 2. Within the while pressed while loop, a second time is being constantly redefined. This means it will take the time as the user lets go of the button. After the button is released, the time difference between when it was let go and the current time is constantly checked. If its greater than 400ms, the input is done, and the decoder function is called. ToDecode is then set to false to make sure it only runs once.

CalcLength checks which indexes of the character array aren’t null and returns the length. wipeArray sets all values in the array back to null.

The checkButton function takes either 1 or 2. 1 represents a “.” and 2 represents a “– “and puts it in the first empty space in the character array.

The decoder function then checks the length and if it is too long, an error is printed to the terminal and the array is wiped. It then compares the individual characters in the array to each combination of “.” and “– “and if it’s a valid morse code, the character is shown on the display and printed to the terminal. The array is then wiped to remove the current morse code.

## Problems Faced

Originally, we went with an approach where we have an integer value that increments and then we could calculate time off that. We had calculated a value for the 250ms and 400ms but then we found out that when more logic is added, such as printing to the terminal, the program would run slower. This meant this method was too inconsistent for us to use.

We also tried to use the getTimeOfDay() which is an inbuilt c function. This was confusing to work with and didn’t work. We think this is due to it not knowing the time of day.

We also tried to use sleep\_ms(time). However, this meant that nothing else could run when it was sleeping, meaning we couldn’t press the button again or press the button in quick succession. We solved this from removing all the sleeps from the code and switching to another solution.

Finally, we found a time\_us\_64() function in the pico sdk. This allowed us to accurately calculate the time in microseconds, letting us have accurate timing for the decoder. The only problem we had with this was overriding the time it was let go for by defining the time let go as t2 then doing calculations by overriding the t2 variable. This was easily fixed though by not overriding the time it was let go and using a different variable for the comparison.

## Group Contribution and Engagement

Once the coursework was set, we sat down together and suggested ideas for possible implementation. We also spent a while setting up the software needed on Camden’s computer to allow us to work on it outside of the lab. Once the idea was set, we started working individually to get it working. Rojin started by defining all the binary values to represent all the characters. Rojin also completed all the individual comparisons in the decoder function, comparing each value of the array with morse code values. Camden implemented the logic to calculate the time held for and time released for. Camden also made the checkButton function to add the characters to the array. We realised that we would need a wipeArray function and checkLength function and implemented them together as they were quite simple.