COM1031 COURSEWORK

Camden Jones – 6809406
Rojin Kara – 6830924

High Level Overview

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

The program begins by initialising all global variables. The variables for timing are set up here so that they can be accessed throughout the program. To store the button inputs, a character array of length 99 is created. After that, the program prints "HELLO" to the terminal and displays the middle bar on the seven-segment display for 2 seconds. The Boolean "toDecode" is also defined here. This prevents it from being reset each time the while loop iterates.

The time is constantly defined in the "while (true)" loop. While the key is held down, a while loop is executed within the original loop. This causes the time overriding to stop. It also displays the "-" symbol on the display while held and sets toDecode to true. This ensures that the decoder function is only executed once.

The time is calculated once the button is released, as it then exits the inner while loop. If the time is between 50ms and 250ms, it is detected as a short press, and a 1 is passed through the function. If the time difference exceeds 700ms, an 8 appears on the display for 200ms and the character array is reset. If the time is greater than 250ms but less than 700ms, it is considered a long press, and checkButton is called with the value of 2. Within the inner while loop, a second time is constantly redefined. This means it will record the time when the user lets go of the button. After the button is released, the time difference between when it was released, and the current time is constantly checked. If it is greater than 400ms, the input is completed, and the decoder function is called. ToDecode is then set to false to ensure that it only runs once.

The CalcLength function checks which indexes of the character array aren't null and returns the length, wipeArray sets all values in the array back to null, and 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 possible 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 had assumed this would work the same way as Date.now() in javascript but it wasn't as simple as that.

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.