Quintin Lopez-Scarim
March 8th 2024
23533621

Report – Intro to Embedded Rust

Introduction

For this lab, hardware in the form of the Adafruit expansions for the rp2040 – and of course the rp2040 itself – were utilized. This includes a matrix of LED's and a board with an accelerometer embedded in it. This lab focuses on using a developing programming tool, rust, to write embedded systems code. As it is likely a new programming language to whoever does this lab, its build tools, flashing methods, syntax, and some other intricacies unique to rust are all new and introduced.

Design

A majority of the design for this lab was oriented around the design I used for the hardware drivers lab. I first created some kind of manipulatable array to send to the neomatrix, and also used I2C to instantiate and read from an accelerometer.

As there was more of a focus on what array was sent to the neomatrix for this lab, I first made a struct called node to store each LED's RGB values – and eventually the direction read from the accelerometer. I then made an array of those nodes and passed them to animation structs. These structs copied that array and then altered positions and RGB values to create "animations" (the details of which I will not go into as they are trivial) and then passed back the altered array to an array in main, which is the main array actually displayed. I passed the arrays by copy to the animation structs to prevent multiple instances from borrowing the same array, and to prevent them from affecting one another. This made my life substantially easier as I didn't have to worry about lifetimes which were unfamiliar concepts to me. The animations were declared as structs with their initialization, next, and to_list functions as described in the assignment document.

Testing for this lab was done mainly by using the blinking example to see if code reached a certain execution point, and then also flashing to the board and verifying that LED's lit up as desired. Due to the simplicity of the code base, this was more than enough, and by using the WHO_AM_I register and the blink example I was able to easily ensure my I2C was communicating correctly.

Most of my time for this lab was spent looking at crates and ensuring I met their dependencies, but two non-trivial pieces of code that are of note are the absolute value function, and shallow copying for the animations.

Because we were not using the std library, there's no function that gets the absolute value of floats, and as such I had to make my own with nested if statements comparing two floats.

As for the animations, whenever I was sliding either all columns or rows down or right, I first had to create a shallow copy of one row or column, otherwise that data would be corrupted. This is an idea from DSA which I don't always get the pleasure of employing, so I found this neat.

Those pieces of code were the most challenging parts of the lab for me besides for figuring out crate inter dependencies which felt very much like trial and error to me.

As for any suggestions I have for improving the structure of this project – I think four animations was kinda tedious, and instead maybe one animation or two could have been made with more emphasis on it having a more intricate implementation. This would expose more of the intricacies of rust in my opinion.

My main takeaway from rust is that it feels too much like just digging through an abstraction of a language developed by multiple developers rather than an actual language. The crates seem finicky and finding what you want is buried in many other implementations that almost accomplish the same thing but require different dependencies that what you have.

Conclusion

Overall some of the ideas of rust feel really unique, and its interesting to see a community come together to make a language that wasn't originally intended for embedded systems work on a high level, but I would personally like to not touch the language again. Integrating many small crates together did not feel smooth, and I think I prefer having a few large dependencies like numpy over a ton of small dependencies in the form of crates.