Brian Engel

Module 4 Journal

When programming Embedded C there are several things that you should do to keep your code clean and secure. Use descriptive variables and function names to keep code readable and maintainable. Use named constants instead of hard coded numbers to make the code more maintainable. Functions should do one thing to make it easier to test and maintain. Use enum for state machines to make code less error prone. Avoid global variables to make code easier to understand. Use the “const” keyword to prevent accidental modification of variables. Check return values of functions for potential errors. Comments should explain "why" something is done rather than "what" is being done. The code itself should be clear enough to explain the "what". Modularize code as much as possible and use header and source files to improve readability and reusability. Use tools to detect potential bugs, memory leaks, or other issues (Garcia). While this is not an exhaustive list, I think it sums up the list we were given by Professor Esposito in week two.

There are also many common pitfalls to programming Embedded C. The first is not understanding the hardware you are programming for. If you don’t understand the I/O ports or peripherals it will be next to impossible to program anything using them. Another mistake is not using the right data types. Unsigned char is a lot different than an int and in functionality and memory. Lack of error checking is another pitfall. Without proper error checking, your program will likely not run as expected, or not at all. Optimizing code is more important with embedded systems than most other types of code. It can easily lead to performance issues, or even possibly cause the entire system to crash. Not using interrupts can cause slow response times to external events which can be disastrous (IIES, 2023). The last major pitfall for Embedded C is not using the best practices laid out in the first paragraph.

There are a few techniques to consider in Embedded C programming. Bit manipulation can be used in systems where memory and processing power are limited because it is efficient, but you need to make sure the code remains clear and easy to understand. Using the volatile keyword is necessary when coding to let the compiler know that the variable could change due to external factors, like hardware interrupts, memory-mapped I/O devices, or variables modified by other tasks in a multi-threaded environment. Memory optimization is maintained by avoiding unnecessary variables, using appropriate data types, and freeing up memory when no longer needed. An extension of this is stack management. You need to be careful of recursive functions as they can cause a stack overflow on a system with limited memory.

Garcia, Y. (2024, February 15). Best Practices for Embedded C Programming: Ensuring Efficiency, Reliability, and Maintainability. LinkedIn. [https://www.linkedin.com/pulse/best-practices-embedded-c-programming-](https://www.linkedin.com/pulse/best-practices-embedded-c-programming-ensuring-yamil-garcia-ibwje)ensuring-yamil-garcia-ibwje

IIES Indian Institute Of Embedded Systems. (2023, June 20). Common Mistakes to Avoid in Microcontroller Embedded C Programming. LinkedIn. Retrieved from [https://www.linkedin.com/pulse/common-mistakes-avoid-micro-controller-embedded-c-](https://www.linkedin.com/pulse/common-mistakes-avoid-micro-controller-embedded-c-iies#:~:text=Not%20Understanding%20the%20Hardware,O%20ports%2C%20and%20other%20peripherals)iies#:~:text=Not%20Understanding%20the%20Hardware,O%20ports%2C%20and%20other%20peripherals