Dominic Drury

01/29/2024

CS-350

4-2 Journal: Best Coding Practices

The best coding practices in Embedded C are to

* only use logical operators with variables representing single-bit data, even though a bitwise operator may yield the same result
* define a variable as volatile if the variable may be changed at any time by an external entity
* Limit memory utalization as much as possible to improve performance of the system
* have an explicit transition point back to the same state with the proper condition, rather than relying on the implicit transition from a state to itself when coding state machines
* create a state named Init to carry out initializations of variables and also outputs when coding state machines
* ensure that actions don't wait on an external input value when coding state machines
* always include a default case for a switch statement, in case something bad happens
* start the main function by initializing all outputs
* plan to spend nearly as much time for testing a system as for designing a system
* name global variables using a "\_g" suffix, to distinguish them from a task's local variables, as was done above for "btn\_g"
* limit a variables scope to where it is needed, limiting it too much might not allow a function to utalize a variable it needs and not limiting it can cause issues with memory, variable naming, variable corruption, etc.

Common pitfalls in Embedded C are to

* accidentally type a bitwise operator like | when intending to type a logical operator like ||, and vice-versa
* not properly define a variable, which could allow a compiler to eliminate a statement entirely when trying to optimize the code
* either limit the memory too much as cause data to be lost or utilizing too much memory and causing the system to operate slowly or with errors
* create transitions leaving a state whose conditions are not mutually exclusive coding state machines
* create transitions such that sometimes no transition has a true condition when coding state machines
* forgetting to turn on the UART
* fail to read R when data is there, thus preventing the hardware from calling the RxISR function
* not ensure that tasks communicate robustly, leading to a system that may sometimes work, but sometimes fails depending on the variations that can occur with "simultaneous" ticks