EE4930 Embedded Systems
Section 011/021, Winter 2022/23

Professor: Kerry R. Widder, Ph.D.
Electrical Engineering and Computer Science Department
Milwaukee School of Engineering

Guidelines for Source Code

Good formatting practices when writing source code make it easier for someone else to understand and follow your code. It is important for me when I am grading your code, it will be important at work (since co-workers may need to use/modify/debug your code, and it is important to you when you need to use/modify/debug your code next week/month/quarter. Good formatting practices include:

- Start each code file with a header comment block. Include the file name, your name, date of creation, course, description of what the code does, any assumptions made, any connections required, revision info (date, initials, changes made), etc.
- USE COMMENTS! Add meaningful comments to help the reader understand what the code is doing.
- Use a mono-spaced (fixed width) font, like Courier New, for printing.
- Don't write long lines of code that will wrap when printing – break comments up into proper length lines for better readability. Printing in landscape mode may also help.
- Consider using spaces instead of tabs (some editors will automatically put spaces for tabs) for better transportability to other editors. Also be sparing with the number of spaces when indenting (use about 3 spaces for each level) to keep your code lines from getting too far to the right.
- Use consistent indenting.

Part of the lab grade will be an evaluation of the format of your code.

Here are some good practices to follow when writing code:

- Perform initialization steps at the beginning of the main program, not in loops or ISRs. (Note: sometimes it is necessary to do one or two initialization steps here (like enabling a timer, etc.), but in those cases the rest of the initialization should be done once at the start of the program.) Putting these steps in loops or ISRs is inefficient since these instructions usually only need to run once.
- Minimize the amount of code in loops – only put statements there if they need to be done every pass through the loop – for greater efficiency.
- Keep ISRs as short as possible to avoid missing interrupts due to delays in processing another interrupt. Avoid function calls in ISRs.
- Don't write dead code – i.e., code that will never actually get executed, hence is unnecessary, or code that really isn't doing anything new. For example, an if statement condition that is never true, hence the statement(s) associated with the if will never execute, or an if statement condition that is always true, hence the statement(s) associated with the if will always execute and the if is pointless.
- Make sure variables are initialized before using their value.
- Put the __enable_interrupt(); instruction after all the other initialization.