Recommended Usage:
- Assign each section below to a specific reviewer, giving two or three sections to each reviewer.
- Ensure that each question has been considered for every piece of code.
- Review 100-400 lines of code per 1-2 hour review session. Do the review in person.

FUNCTION
- F-1. Does the code match the design and the system requirements?
- F-2. Does the code do what it should be doing?
- F-3. Does the code do anything it should not be doing?
- F-4. Can the code be made simpler while still doing what it needs to do?
- F-5. Are available building blocks used when appropriate? (algorithms, data structures, types, templates, libraries, RTOS functions)
- F-6. Does the code use good patterns and abstractions? (e.g., state charts, no copy-and-paste)
- F-7. Can this function be written with a single point of exit? (no returns in middle of function)
- F-8. Are all variables initialized before use?
- F-9. Are there unused variables?
- F-10. Is each function doing only one thing? (Does it make sense to break it down into smaller modules that each do something different?)

STYLE
- S-1. Does the code follow the style guide for this project?
- S-2. Is the header information for each file and each function descriptive enough?
- S-3. Is there an appropriate amount of comments? (frequency, location, and level of detail)
- S-4. Is the code well structured? (typographically and functionally)
- S-5. Are the variable and function names descriptive and consistent in style?
- S-6. Are "magic numbers" avoided? (use named constants rather than numbers)
- S-7. Is there any “dead code” (commented out code or unreachable code) that should be removed?
- S-8. Is it possible to remove any of the assembly language code, if present?
- S-9. Is the code too tricky? (Did you have to think hard to understand what it does?)
- S-10. Did you have to ask the author what the code does? (code should be self-explanatory)

ARCHITECTURE
- A-1. Is the function too long? (e.g., longer than fits on one printed page)
- A-2. Can this code be reused? Should it be reusing something else?
- A-3. Is there minimal use of global variables? Do all variables have minimum scope?
- A-4. Are classes and functions that are doing related things grouped appropriately? (cohesion)
- A-5. Is the code portable? (especially variable sizes, e.g., “int32” instead of “long”)
- A-6. Are specific types used when possible? (e.g., “unsigned” and typedef, not just "int")
- A-7. Are there any if/else structures nested more than two deep? (consecutive “else if” is OK)
- A-8. Are there nested switch or case statements? (they should never be nested)
EXCEPTION HANDLING
- E-1. Are input parameters checked for proper values (sanity checking)?
- E-2. Are error return codes/exceptions generated and passed back up to the calling function?
- E-3. Are error return codes/exceptions handled by the calling function?
- E-4. Are null pointers and negative numbers handled properly?
- E-5. Do switch statements have a default clause used for error detection?
- E-6. Are arrays checked for out of range indexing? Are pointers similarly checked?
- E-7. Is garbage collection being done properly, especially for errors/exceptions?
- E-8. Is there a chance of mathematical overflow/underflow?
- E-9. Are error conditions checked and logged? Are the error messages/codes meaningful?
- E-10. Would an error handling structure such as try/catch be useful? (depends upon language)

TIMING
- T-1. Is the worst case timing bounded? (no unbounded loops, no recursion)
- T-2. Are there any race conditions? (especially multi-byte variables modified by an interrupt)
- T-3. Is appropriate code thread safe and reentrant?
- T-4. Are there any long-running ISRs? Are interrupts masked for more than a few clocks?
- T-5. Is priority inversion avoided or handled by the RTOS?
- T-6. Is the watchdog timer turned on? Is the watchdog kicked only if every task is executing?
- T-7. Has code readability been sacrificed for unnecessary optimization?

VALIDATION & TEST
- V-1. Is the code easy to test? (how many paths are there through the code?)
- V-2. Do unit tests have 100% branch coverage? (code should be written to make this easy)
- V-3. Are the compilation and/or lint checks 100% warning-free? (are warnings enabled?)
- V-4. Is special attention given to corner cases, boundaries, and negative test cases?
- V-5. Does the code provide convenient ways to inject faulty conditions for testing?
- V-6. Are all interfaces tested, including all exceptions?
- V-7. Has the worst case resource use been validated? (stack space, memory allocation)
- V-8. Are run-time assertions being used? Are assertion violations logged?
- V-9. Is there commented out code (for testing) that should be removed?

HARDWARE
- H-1. Do I/O operations put the hardware in correct state?
- H-2. Are min/max timing requirements met for the hardware interface?
- H-3. Are you sure that multi-byte hardware registers can’t change during read/write?
- H-4. Does the software ensure that the system resets to a well defined hardware system state?
- H-5. Have brownout and power loss been handled?
- H-6. Is the system correctly configured for entering/leaving sleep mode (e.g. timers)?
- H-7. Have unused interrupt vectors been directed to an error handler?
- H-8. Has care been taken to avoid EEPROM corruption? (e.g., power loss during write)

This document is placed in the public domain. Credit to the original authors is appreciated.