MISRA C REPORT

Violation Addressed:

Rule: Unused Type Declaration
Description: The rule prohibits the declaration of types that are not used in the program. This helps in maintaining cleaner and more efficient code.
Violation Identified: Unused type declaration for *int32_t*.

Reflection: To resolve this MISRA violation, the typedef for *int32_t* and *uint32_t* was removed. These types are already defined in the standard library *stdint.h*. Therefore, instead of defining them explicitly, stdint.h was included in the code.

- Code Change:
  #include <stdint.h> - for int32_t and uint32_t

Outcome: By including stdint.h, the unnecessary type declarations were eliminated, making the code compliant with the MISRA rule for unused type declarations. This change also simplifies the code and reduces the potential for errors related to type definitions.

Remaining Violations:

Rule 4.6 - Typedefs for Basic Numerical Types:

Violation & Reflection: The code uses basic numerical types like int, char, which risks portability due to varying sizes on different platforms. Adopting fixed-width types like int32_t, uint32_t will ensure consistent size and signedness, enhancing robustness across platforms. Unresolved due to the substantial refactoring required, which could impact the stability of the codebase and delay project deliverables.

Rule 17.7 - Unused Return Values:

Violation & Reflection: Functions snprintf and write return values that are not utilized, risking unchecked errors. Not addressed because integrating proper error handling for these return values would necessitate significant architectural changes, which are not feasible within the current project phase.

Rule 12.3 - The Comma Operator:

Violation & Reflection: The comma operator's use in our code impacts readability and could introduce subtle bugs. Refactoring is required to separate expressions into distinct statements, thus adhering to MISRA standards and enhancing code maintainability. Remains due to the complexity of refactoring the current logic, where the comma operator is entrenched, and separation could introduce new bugs that require extensive validation.

Rule 8.4 - Visibility of External Linkage:

Violation & Reflection: Objects and functions with external linkage lack visible declarations, leading to potential linkage errors. Ensuring declarations in appropriate header files will centralize definitions and improve maintainability. Left unchanged to maintain backward compatibility and due to the interconnected nature of the legacy system, where changes might introduce far-reaching impacts.

Rule 8.7 - Objects with External Linkage:

Violation & Reflection: Objects like exit_flag and functions with external linkage are only referenced in one translation unit. Applying static where appropriate will reduce namespace pollution and improve encapsulation. Not rectified as the use of static linkage could lead to extensive cross-module dependencies being disrupted, requiring a broader codebase evaluation and potential redesign.

Rule 15.5 - Single Exit Point from Functions:

Violation & Reflection: Multiple exit points in functions complicate resource management and program flow. Refactoring to a single exit point will enhance readability and maintainability. Continues to exist due to the risk of altering the control flow and introducing regressions, outweighing the benefits of compliance at this stage of development.

Rule 2.7 - Unused Parameters in Functions:

Violation & Reflection: Unused parameters in functions may lead to confusion. Removing or utilizing these parameters will align function design with usage, improving code clarity. Unused parameters remain to maintain API consistency across the system, where they may be expected by other interfacing modules or future extensions.

Rule 21.10 - The Standard Library Time and Date Functions:

Violation & Reflection: Use of non-deterministic standard library time and date functions is discouraged. Adopting deterministic time services will ensure predictable software behavior. Still in use as the potential replacements for deterministic time services would require significant development effort, which is not justifiable given the current system requirements and constraints.