Application of MISRA C:2012 in C code programming used in the Subway Arrival System of Mexico City

A Thesis

Submitted to the Faculty of Fachhochschule Aachen and Centro de Ingeniería y Desarrollo Industrial

BY

Maximiano Francisco Ruiz Páez

In Partial Fulfillment of the requirements
for the degree of Master of Science in Mechatronics

Santiago de Querétaro, Qro., México, September 2018

Abstract

This thesis work presents the application on the MISRA C:2012 Guidelines of the Modbus source code developed at CIDESI for use in the Mexico City's subway systems, especially in the arrival system, in which CIDESI has been working. The purpose of applying MISRA C on this source code is to increase its safety and to detect and erase any unwanted or unforeseen behavior in this code.

For the realization of this work, the recommendations within the MISRA guidelines were followed. As a starting point to this recommendations a C-style standard for written uniformity of code developed within any defined workgroup is proposed and then applied to the original Modbus files made at CIDESI.

Among the MISRA requirements developed during this project, "the compliance matrix" and the "deviation documentation format" were defined. The first helps to ensure and document how all rules are checked and to document changes done in the code to comply with a rule. The second document assists when due to the nature of the project requirements, a deviation from the rules is needed to be allowed.

Afterwards, for the modification of the source code in order to make it MISRA C complaint the IAR Embedded Workbench compiler version 8.22 and the C-STAT checking tool version 1.5.2 were used as the softwares for compiling the source code and for the MISRA C:2012 automatic static checking.

A record of the results of these analyses are presented, as well as all the modifications done on the code to make it MISRA C complaint.

Finally in the Appendixes to this work one can find the obtained compliance matrix for the Modbus code as well all deviations declared for this project.

Declaration of Authorship

Hereby, I, Maximiano Francisco Ruiz Páez, declare that this thesis "Application of MISRA C:2012 in C code programming used in the Subway Arrival System of Mexico City" is the result of my own work.

Any part of this dissertation has not been previously submitted, in part or whole, to any university or institution for a degree or other qualification.

I confirm that all consulted work from others is attributed and the source is always given. Furthermore, the data and the software employed have all been utilized in complete agreement to the copyright rules of the concerned establishments.

Signed: _____

Santiago de Querétaro, México, September 2018

Acknowledgments

I would like to thank everyone who helped me in Mexico and Germany during my studies, both Professors, administrative staff and friends of both CIDESI and FH Aachen who always were kind and willing to help.

I would also like to personally give my thanks to:
M. Sc. Sadot Arciniega and M. Sc. Hiram Hernández for their help, advice and recommendations for the improvement and realization of this thesis work.

Prof. Dr. Kämper and Prof. Dr. Woller for their guidance and help while I was in Germany

Dr. Salvador Acuña for the help granted through my Master's studies

To my Mother and Father for all their support, unconditional love and for taking care of my dog and cat while I was abroad.

And finally to CONACYT for the financial support to realize this Master's studies.

Index of contents

Index of tables

.

Index of figures

Chapter 1

Introduction

CIDESI is currently working on developing new systems to improve Mexico City's Subway. Among these systems is the subway arrival system, which will inform subway users of the time left for the next subway train to arrive. In order to develop this systems multiple sensors had to be placed along the subway line. For the communication of this sensors a Modbus protocol of master/slave is used. The correct coding of this Modbus communication protocol is vital to the correct operation of the system. In order to increase the safety and robustness of the source code used in this system the use of the MISRA C guidelines was decided.

1.1 Statement of the problem

Developing code is a complex matter in which causing unforeseen, unwanted behaviors is relatively easy. In embedded systems, especially in the ones where human life is involved, ensuring that no such behaviors can occur is critical to safety. To this effect multiple coding standards have been created but their implementation and application can be difficult and time consuming.

CIDESI's work at Mexico City's subway system relies heavily in the use of embedded systems programed in C language. Since these systems must be error free for the sake of safety and reliability the adaptation of C standards is useful to increase the robustness of the source code created.

Also, since many different programmers work in the development of the code for Mexico City's subway arrival systems and since most C coding standards, like MISRA, recommend a C style format guide is also needed to make the code made by different programmers more consistent in its written style making it easier and faster to understand between programmers.

1.2 Justification

MISRA C is the most widely used coding standard for coding in critical systems. Developing a methodology for its easy application can help improve the quality of the code developed at CIDESI for the new arrival systems of Mexico City's subway.

Using MISRA C and adapting a C style coding standard will help to make any code created more robust, unlikely to have errors or bugs and make it easier to maintain in cases where the code needs to be modified by a group of programmers other than the original creators.

1.3 Objectives

1.3.1 General objective

- Application of the MISRA C Guidelines in the Modbus communication C source code used in the new embedded systems developed by CIDESI for the Mexico City's subway.

1.3.2 Specific objectives

- Development of a methodology for applying MISRA C to a source code following the recommendations MISRA C itself proposes.
- Development of documentation formats to record deviations and compliance with MISRA rules.
- Development of a C style guide for making the source code written format standard at CIDESI.
- Revision and modifications to the Modbus source code used in the subway system to make it MISRA compliant.
- Elaboration and defense of a Master's Thesis.

1.4 Hypothesis

A method can be developed to apply MISRA C guidelines in efficient ways in the C code written at CIDESI. Specifically, in the Modbus source code developed for use in the new

embedded systems programed in C language developed by CIDESI for Mexico City's subway arrival system.

1.5 Methodology

While the MISRA C guidelines document doesn't enforce or specifies an obligatory way to apply its guidelines to a project or to any C source code. It gives a series of recommendations on its use.

It recommends to stablish a C-style guide with the purpose of making the C code more "alike" inside an organization or work group. This C-style guide should specify the text and formatting rules when writing C code, as well as standards for documenting and commenting in the code, such as readme files.

The MISRA guidelines also state that some formal documentation method should be stablished by any organization to document and authorize any deviations from the guidelines. MISRA also recommends to use a "Compliance matrix" which should list all the guidelines and document in which ways was each guideline verified.

A formal deviation document and compliance matrix were developed for this project based on the recommendations and example presented in the MISRA C guidelines.

Chapter 2

State of the Art

2.1 Coding standards

According to IEEE, C languages occupies the number one rank in the Top Ten Programming Languages for 2016 [1]. Together with C++, they are also the only high-level programming languages useful for low level and embedded programming.

The following is a brief review of the most commonly used C coding standards used actually to reduce vulnerabilities, errors, bugs and other common problems when programming in C language at the early stages of the software development process

2.1.1 MISRA C coding standard

MISRA, The Motor Industry Software Reliability Association is a collaboration between manufacturers, component suppliers and engineering consultancies which seeks to promote best practice in the development of safety-related embedded electronic systems. To this end MISRA publishes documents that provide accessible information for engineers and management, and holds events to permit the exchange of experiences between practitioners. [2]

MISRA C 2012 is the third revision of the MISRA C guidelines. The new guidelines document roughly double the size of the previous revision (MISRA 2004) although the number of guidelines was increased by around 10% only.
MISRA C 2012 has the following advantages versus previous version of itself and other C coding standards: [3]

- Better rationales for guidelines
- More precise descriptions
- More coding examples

MISRA C:2012 was published on 18 March 2013. MISRA C:2012 extends support to the C99 version of the language whilst maintaining guidelines for C90. It contains 143 rules and 16 "directives" (that is, rules whose compliance is more open to interpretation, or relates to process or procedural matters); each of which is classified as mandatory, required, or advisory. They are separately classified as either Single Translation Unit or System. Additionally, the rules are classified as Decidable or Undecidable.

2.1.2 SEI CERT C coding standard

SEI CERT describes itself as "Rules for secure coding in the C programming language. The goal of these rules and recommendations is to develop safe, reliable, and secure systems. Conformance to the coding rules defined in this standard are necessary (but not sufficient) to ensure the safety, reliability, and security of software systems developed in the C programming language."[4] Each rule consists of a title, a description, and noncompliant code examples and compliant solutions.

CERT's coding standards are being widely adopted by industry. Companies such as Cisco Systems, Inc. and Oracle have adopted this coding standard into their existing secure coding standards. [4]

The CERT C Secure Coding Standard is primarily intended for developers of C language programs but may also be used by software acquirers to define the requirements for software. This standard is of particular interest to developers who are building high-quality systems that are reliable, robust, and resistant to attack. The CERT C Secure Coding Standard was developed on the CERT Secure Coding wiki. [4]

2.1.3 BARR Group – Embedded C coding standard

The Barr group C coding standard objective is to reduce the number of software bugs present in new embedded software and in code added or modified later by maintainers. The guide

describes techniques to reduce or eliminate the number of bugs. While also improving the maintainability and portability of embedded software.

This coding standard details a set of guiding principles as well as specific naming conventions and other rules for the use of data types, functions, preprocessor macros and variables among others. Individual rules that have been demonstrated to reduce or eliminate certain types of bugs are highlighted. In its latest version, BARR-C: 2018, the coding rules have been fully harmonized with MISRA C: 2012, for helping in embedded system to reduce defects in firmware written in C and C++. [5]

2.1.4 CWE (Common weakness enumeration)

The CWE is a community-developed formal list of common software weaknesses. It serves as a common language for describing software security weaknesses, a standard measuring for software security tools targeting these vulnerabilities, and as a baseline standard for weakness identification, mitigation, and prevention efforts. If necessary, CWE can also be scoped to specific languages such as C. [6]

CWE is built from the diverse thinking on this topic from academia, the commercial sector, and government thanks to its community developed origin. Its objective is to help the code security assessment industry and also accelerate the use and utility of software assurance capabilities for organizations in reviewing the software systems they acquire or develop.

First developed by MITRE a not for profit organization which began working on the issue of categorizing software weaknesses as early 1999 when it launched the CVE List a dictionary of publicly disclosed cybersecurity vulnerabilities. As part of the development of CVE, MITRE's CVE Team developed a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts to help define common software weaknesses. [6]
Currently the most recent version is 3.0 which was released in November 2017.

This list has over 600 categories, not all of them C language relevant, including classes for buffer overflows, path/directory tree traversal errors, race conditions, cross-site scripting, hard-coded passwords, and insecure random numbers.

2.2 C-Style Guides

As said by Steve Oualline "Good programming style begins with the effective organization of code. By using a clear and consistent organization of the components of your programs, you make them more efficient, readable, and maintainable". [7]

Good style in software coding is defined as that which is organized, easy to read, easy to understand, maintainable and efficient.

2.2.1 C style guide by The Software Engineering Laboratory

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. [8]

This document discusses recommended practices and style for programmers using the C language in the Flight Dynamics Division environment. Guidelines are based on generally recommended software engineering techniques, industry resources, and local convention. It offers preferred solutions to common C programming issues and illustrates through examples of C code.

It was first published in August 1994. Although this document was written specifically for programmers in the SEL environment, the majority of these standards are generally applicable to all environments.

2.2.2 ESA Style Guide for 'C' coding

The Experts Solutions Australia (ESA) style guide for C is a guide to what is considered good style for C code. According to the ESA guide, style covers general rules to be applied when

designing and writing code. While designed for C, the ESA style guide can be applied to any programming language. [9]

It has recommendations for the use of comments, variable initialization, control flow, the use of returns and break statements, and the use of pointers, structures, functions and memory.

The document is structured into a number of general headings, and within each heading is a list of recommendations. The definitions used in the C Coding Standard are used in this document. It was original published in 1991.

2.2.3 Recommended C Style and Coding Standards Guide

This document is a modified version of a document from a committee formed at AT&T's Indian Hill labs to establish a common set of coding standards and recommendations for the Indian Hill community. The scope of this guide is C coding style. According to it, Good style should encourage consistent layout, improve portability, and reduce errors. This style guide does not cover functional organization. [10]

The document for this guide states that individual institutions or groups may adopt part or all of standards it proposes as a part of program code acceptance. It also recommends that when changing existing code it is better to conform to the style (indentation, spacing, commenting, and naming conventions) of the existing code than it is to blindly follow a style guide.

Ultimately, the goal of these standards is to increase portability, reduce maintenance, and above all improve clarity.

2.3 MISRA Checking Tools

While there exist many software tools that claim to check code for "MISRA conformance", there is no MISRA certification process and there are not checking tools endorsed or recommended by MISRA itself. [11]

Most of the guidelines can be checked using tools that perform static code analysis. One such tool is IAR C-STAT which is briefly described next:

2.3.1 IAR C-STAT

C-STAT is a static analysis tool that tries to find deviations from certain coding rules by performing one or more checks for the rule. The checks are grouped in packages. The various packages available are:

- STD_CHECKS: Contains checks for rules that come from CWE, as well as checks specific to C-STAT.
- CERT: Contains checks for CERT. In addition, some CERT rules and recommendations can be verified by checks for other standard rules.
- MISRA C:2004, C++:2008 and C:2012.

The checks for the mandatory and required rules of MISRA C:2012 are by default on, whereas the checks for the advisory rules are by default off. [12]

Chapter 3

Theoretical Framework

3.1 Appling MISRA C in a project

It is always recommended to adopt the use of the MISRA guidelines from the start of a project. Although it must be clarified that adherence to the MISRA guidelines does not in itself ensure error-free robust software or guarantee portability of the code.

In cases when a project is built over the code of a previous project or when MISRA is chosen to be used later in the development process of a project the benefits of compliance with MISRA C may be outweighed by the risk of creating defects when editing the code to make it compliant. In such case it is necessary to evaluate the implementation needs of the MISRA C guidelines. [1]

3.2 Compliance

In order to ensure that all of the MISRA C guidelines are followed. A compliance matrix should be produced. This matrix should list every guideline and indicate how the checks are performed.

In most guidelines the most efficient way for checking the rules will be using a static analysis tool, the compiler or a combination of both. [1]

When using a static checking tool, the following information should be recorded for each tool used for checking:

- Version number of the tool.
- Options and configuration of the checking tool when the code was verified.

3.2.1 Deviation procedure

In some case it must be necessary to deviate from the guidelines given in the document. It is then needed that such deviations are documented and authorized.

Such a deviation procedure should be formalized with the code development process. MISRA does not imposes any MISRA C deviation recording procedure, since the method requirements will vary between organizations. [1]

3.3 Guidelines classification and categories

MISRA C guidelines are divided in Directives and Rules.

A directive is a more general guideline. They cannot be specifically verified since they sometimes require additional information, such as design documents or requirements specifications.

A rule is a guideline with a complete description. It should be possible to check compliance with a rule by checking the code directly. Most rules are checkable using static analysis tools.

3.3.1 Guideline categories

Each guideline is classified in either “mandatory”, “required” or “advisory”.

It must be noted that an organization or work group can choose to treat any required guideline as mandatory and any advisory guideline as required or mandatory. [1]

3.3.1.1 Mandatory guidelines

To claim that any C code is MISRA C complaint every mandatory guideline must be obeyed. In the case of mandatory guidelines deviations are not permitted. [1]

3.3.1.2 Required guidelines

C Code which is claimed to be MISRA C complaint must obey every required guideline, with a formal documentation in case of deviations needed. [1]

3.3.1.3 Advisory guidelines

These guidelines are recommendations, still they must be followed as far as reasonably practical within the scope of the project. In the case of not complying with an advisory guidelines a formal deviation is not necessary, although it is always recommended to have some method of documenting non-compliances. [1]

3.4 Rules decidability

Each rules is either classified as Decidable (D) or Undecidable (U). Decidable rules are the ones that can be checked using a static analyzer to verify if the code complies with a rule or not. Undecidable rules much be manually checked and usually depend on the code's run-time properties to detect violations. [1]

3.5 Analysis Types

The analysis required for verifying a rule can be either "Single Translation Unit (STU)" or "System (Sys)". [1]

Rules are classified as "Single Translation Unit" if compliance to said rule can be verified by checking each code line or statements individually.

A rule is considered as "System" if the entire source code needs to be verified in other to claim compliance. This type of rules are more easily checked using a static analysis tool.

3.6 MISRA guidelines

The next are a list of all MISRA C:2012 directives and guidelines, this were studied and understood for the purpose of this project

3.5.1 Directives

There are a total of 16 directives, divided in 4 groups according to their scopes. A full list of the directives can be found in Appendix A

3.5.2 Rules

There are a total of 143 rules divided in 22 categories. A full list of the MISRA rules can be found in Appendix B of this document.

Chapter 4

C-style Standard for CIDESI

The suggestions in this document cover how code should be designed and written, as opposed to how the syntactic elements should be laid out, which is the scope of the MISRA C Coding Standard.

A workgroup standard style has the advantage that the intent and workings of a piece of code are easier to grasp because the way things are done are similar between al members of the work group. [9]

The following recommendations are based on the “C style guide by The Software Engineering Laboratory” and the “ESA Style Guide for 'C' coding” documents.

4.1 Use of indentation, blank lines and space.

The correct use of spacing can help make the source much easier to read and maintain.

4.1.1 Blank lines

It is recommended to divide de code in “paragraphs” grouping the “#include” preprocessing directives at the start and then leaving a blank space to declare the global variables followed by another blank line and then a function. For example:

```
#include "gpio.h"
#include "hw_memmap.h"

bool Data_ready   = false;
bool Process_data = false;
bool esperate     = false;

uint16_t get_Registro (void)
{
    return sDataMaster.Registro;
}
```

Inside a function is also helpful to organize the code paragraphs for variables and then statements. It is recommended to leave a blank line before a loop statement like if or else. For example, the next code segment shows how to use blank lines in a function:

```
uint8_t sendCoilMaster(sReadCoilMaster MDatos, uint8_t *Data)
{
    uint8_t  Data_to_send[100] = "";
    uint8_t  Datos_cadena[4];

    f = Armar_Cadena(Datos_cadena,MDatos.address, Data_to_send, i);

    while(intentos < 2)

    {
```

Only a single blank line should be used. The use of double blank lines should be avoided since it makes grouping consume too much space which can actually make the text harder to read.

4.1.2 Spacing

Correct spacing makes the readability of variables and operators easier and simpler. It also improves the overall presentation of the code text. The next examples illustrate how to correctly use spacing to make the text easier to read:

```
for(uint8_t k = 0; k < sMasterData -> NoData; k++)

uint32_t pendingBytes2 = uartGetPendingBytes(MODBUS);

if((MSBCRC == MSB_Calculado) && (LSBCRC == LSB_Calculado))
```

While not using spacing make the statements much harder to read as shown when the previous examples are modified without correct spacing:

```
for(uint8_t k=0;k<sMasterData->NoData;k++)

uint32_t pendingBytes2=uartGetPendingBytes(MODBUS);

if((MSBCRC==MSB_Calculado)&&(LSBCRC==LSB_Calculado))
```

When using commas a space must be added after the comma. For example:

```
preparare = Modbus(datos_ModbusRx, pendingBytes2, slave);
```

4.1.3 Indentation

Indentation is a tool that makes the structure and logic of a code faster to identify, and thus easier to read and understand. Four spaces is the recommended indent for readability and maintainability. In case where using four space would cause the code to reach the end of the line needing an additional line its acceptable to use less spaces for indentation.

An example on correct indentation is shown next fragment of code:

```
void decodificar_cadena_Slave(sReadCoilMaster *sMasterData, uint8_t slave)
{
    bool     preparar_informacion;

    if(pendingBytes2 != cero)
    {
        preparare = Modbus(datos_ModbusRx, pendingBytes2, slave);

        if(preparar_informacion == true)
        {
            sMasterData -> address     = datos_ModbusRx[i++];
            if(sMasterData -> funcion != 0x06)
            {
                registro    |= datos_ModbusRx[i++];
            }
        }
    }
}
```

4.2 Comments

Comments should be used to provide information which is not obvious from only reading the code. Comments can be added in various sections of the code or in a separate README file.

The README file should explain the code and also include and explanation of the code files organization.

A file prolog in the code itself can be used to as the first section of the code. It should explain the purpose of the code and give information for the code identification such as code version or original programmer.

A prolog can optionally be used before a function to give additional information or an explanation of it.

Through the document, wherever data is being declared or defined, comments can be used to explain the purpose of this various variables or definitions.

Three basic styles of commenting are recommended. Comments should be added to give extra information about the code or to section the code only. Avoid repeating or giving obvious information on the comments.

4.2.1 Boxed comments: used for prologs or section separators.
Example:

```
/*****************************************************
* FILE NAME                                          *
*                                                    *
* PURPOSE                                            *
*                                                    *
* OTHER INFORMATION                                  *
*****************************************************/
```

4.2.2 Block comments: used at the begging of a code section for description of the code.
Example:

```
/*
*Comments are written here, in full sentences
* This type of comments are used when more
* than one sentence is used.
*/
```

4.2.3 Short comments: written in the same line as the code which they describe.
Example:

```
int8_t  variable_X;     /* Info on variable_X comment*/
```

Additionally a comment line as shown in the next example can be used as separator. It is recommended to use this type of separator to indicate the end of functions.
Example separator:

```
/*****************************************************************************/
```

4.3 Recommendation for the use of names

For naming any file, function, constant or variable it is important to take into account the MISRA C:2012 rules on identifiers and naming. Besides following MISRA, names should be meaningful related to the object they belong to.

Through the document where any abbreviations are used they must be kept constant thought the code. For example if a name of a function is declared as DC_Function where DC is an abbreviation of "data check" the same abbreviation should be used through the code whenever an object related to "data check" is named.

An underscore should be used to write more elaborated names, for example "Modbus_Value" is better than "ModbusValue" for reading simplicity and clarity.

Avoid writing similar names or names that only differ in the use of letter case for differentiation (C is case sensitive). Avoid using names that differ in only one letter. Names should be unique and differ in at least two letter or have a number added to the names end to avoid confusions.

Do not assign a variable and a typedef (or struct) with the same name, even though C allows this. This type of redundancy can make the program difficult to follow.

4.3.1 Standard names

As described in the NASA C-style document when the use of a variable is obvious for the case of common practice short names, the following naming conventions can be followed:

c	characters
i, j, k	indices
n	counters
p, q	pointers
s	strings

4.3.2 Variable names

Names for global variables should not be same as the name of internal function variables to avoid creating hidden variables which could cause unwanted behaviors in the code. The matter of hidden variables is also considered in the MISRA C guidelines.

4.3.4 Capitalization

It is recommended to follow the next rules for the declaration of names to make it easier and faster to identify between variables, functions and constants within the source code.

Variables: Use lower case letters. Separate words with underscores.

Functions: Capitalize the first letter of each word, use underscores to separate words.

Constants: Use capital letters only. Separate each word with underscores.

Examples:

- Variable: answer_time
- Function: Time_Counting
- Constant: TIME_MIN

4.4 Data structures

A data structure defines the information about an object. All information about an object should be grouped into a struct. Structs should have more than one element.

The recommended format for a struct is:

```
struct
{
  uint8_t address;
  uint8_t no_registros;
  uint8_t data[100];
  uint16_t registro;
} struct_name;
```

4.5 Variable initialization

Avoid initializing an automatic variable where it is defined. Instead, initialize it closer to where the decision is made about what its value will be, or where it is used. This makes it easier that the variable is initialized correctly.

Chapter 5

C-Style standard modifications to the Modbus code

As a first step towards modification of the Modbus source code files to achieve MISRA C:2012 compliance the source files were edited to comply with the newly recommended C-style standard. This modification of the sources files to follow the C-style standard recommendations made the project files easier to read and easier to understand.

Most expressions and statements spacing and indentations were edited in order comply with the C-style standard, although no functions or variable identifier names were changed to avoid possible conflicts. For example, the code for the function decodificar_cadena_Slave written in Modbus.c was modified as follows, first the original code without the recommendations of C-style standard is presented:

```
void decodificar_cadena_Slave(sReadCoilMaster *sMasterData, uint8_t slave)
{
  bool preparar_informacion=false;
  uint8_t datos_ModbusRx[100]="";
  uint8_t i=0;
  uint16_t registro=0;
  uint16_t registerqty=0;
  uint32_t pendingBytes2=uartGetPendingBytes(MODBUS);
  if(pendingBytes2!=0)
  {
   uartGetData(MODBUS,datos_ModbusRx,pendingBytes2);
   preparar_informacion =comprobar_Datos_Modbus(datos_ModbusRx,
                                                pendingBytes2,slave);
   if(preparar_informacion==true)
   {
    sMasterData->address=datos_ModbusRx[i++];
    sMasterData->funcion=datos_ModbusRx[i++];
    if(sMasterData->funcion!=0x06)
    {
        registro|=datos_ModbusRx[i++];
        registro=registro<<8;
        registro|=datos_ModbusRx[i++];
        sMasterData->Registro=registro;
        registerqty|=datos_ModbusRx[i++];
        registerqty=registerqty<<8;
        registerqty|=datos_ModbusRx[i++];
        sMasterData->NoRegistros=registerqty;
    }
   }
  }
}
```

After the Proposed C-style rules are applied, mostly changing only indentation and spacing:

```
void decodificar_cadena_Slave(sReadCoilMaster *sMasterData, uint8_t slave)
{
    bool     preparar_informacion;
    uint8_t  datos_ModbusRx[100]  = "";
    uint8_t  i             = 0;
    uint16_t registro      = 0;
    uint16_t registerqty   = 0;
    uint32_t pendingBytes2 = uartGetPendingBytes(MODBUS);
    uint32_t cero          = 0;

    if(pendingBytes2 != cero)
    {
        (void) uartGetData(MODBUS, datos_ModbusRx, pendingBytes2);
                                  /* (void) added for rule 17.7 */
        preparar_informacion = comprobar_Datos_Modbus(datos_ModbusRx,
                                                      pendingBytes2, slave);

        if(preparar_informacion == true)
        {
            sMasterData -> address     = datos_ModbusRx[i++];
            sMasterData -> funcion     = datos_ModbusRx[i++];

            if(sMasterData -> funcion != 0x06)
            {
                registro    |= datos_ModbusRx[i++];
                registro     = registro << 8;
                registro    |= datos_ModbusRx[i++];
                sMasterData -> Registro = registro;
                registerqty |= datos_ModbusRx[i++];
                registerqty  = registerqty << 8;
                registerqty |= datos_ModbusRx[i++];
                sMasterData -> NoRegistros = registerqty;
            }
        }
    }
}
```

In this case indentation and spacing make the code logic and structure easier to follow, as well as making the operations and statements more readable. It must be noted that the modified code also presents the MISRA C Guidelines modifications explaining why the syntax is not exactly the same in both instances presented here of the same function.

Chapter 6

MISRA Application

For the modification of the code in order to make it MISRA C:2012 complaint the recommendations from the MISRA guidelines itself were followed. After the modification to the code to make it standard to the newly defined C-style the deviation documentation format and the MISRA C compliance matrix were created.

It must also be stated that creating code that is MISRA C complaint is much easier if MISRA is considered from the beginning of the source code creation. In the case of this work, MISRA C 2012 was applied to the Modbus code developed by CIDESI programmers for use in the arrival systems for Mexico City's subway.

6.1 MISRA C:2012 Compliance Matrix

To ensure that all directives and rules are checked, and to document any warning or errors during compilation of the code and the error given by the static analysis checking tool the compliance matrix must be used.

This compliance matrix was created following the MISRA guidelines recommendations for it. An example part of the matrix as well as an example way of filling it can be seen in the next figure. The matrix was written and filled used Microsoft Excel.

In the complete matrix table all 16 directives and 143 rules are listed. The background color for each rule and directive indicates if the rule is advisory (Pale blue), required (pale orange) or mandatory (orange). The complete Compliance Matrix used during the modifications to the Modbus source code can be seen in full in Appendix C of this work.

DIRECTIVE	COMPILER		CHECK TOOL		MANUAL REVISION
	IAR E.W. v8.22	Other	C-STAT v1.5.2	OTHER	
Dir 1.1	No errors	N/A	Not checkable	N/A	Directive followed.
Dir 4.4	No errors	N/A	No errors	N/A	Followed. Unused code erased.
Dir 4.5	No errors	N/A	No errors	N/A	Reviewed. Minor changes done.
Rule 1.1	No errors	N/A	Not checkable	N/A	Manual review. No errors found.
Rule 2.2	No errors	N/A	Multiple errors	N/A	Corrected: Unused code erased.
Rule 9.1	No errors	N/A	No errors	N/A	Manual check: No errors found.

Figure 1. MISRA C:2012 compliance matrix (fragment)

For the matrix filling method, under “compiler” the name as well as the version of the compiler used most be written. It is recommended to use at least two different compiler software’s to decrease the chance of errors in case of porting the code to other compiler software. In the case of the Modbus source code, due to licenses constraints, the code was only checked and compiled in IAR Embedded Workbench Version 8.22 so “IAR E.W. v8.22” is filled in as the first compiler and “other” is used as an empty filler name.

After compiling any warning given by the software must be registered in the matrix next to any rule that could be affected or causing it. Likewise for the Static MISRA check tool, the use of at least 2 different checking tools is recommended. The checking tool used in this case was C-STAT for MISRA 2012 which is a plugin extension for IAR Embedded Workbench. The C-STAT version used is 1.5.2.

Once the automatic checks are realized the results should be filled in the matrix. Short messages are recommend such as “error”, “no errors” and “multiple errors”. Once the results of the static check are added to the matrix one can continue to the next step, which is to manually correct the found rule violations. Tools that can’t be automatically checked by a software tool must be marked in the matrix, as “not checkable”.

For the Manual Revision part of the matrix one should briefly document any changes made to the code in order to correct the detected rule violation. Cases where the code is not modified because it was decided not to follow an advisory rule must be also be indicated as well as any formal deviation declared or any errors that could not be corrected. In the case of this project each rule was also checked manually even the ones that gave no errors by automated checking tools.

6.2 Deviation Documentation Format

When a rule has to be breached because of the project requirements, because the changes to code result too complex, unpractical or when it seems that a modification to a part of the code could cause more problems than benefits a formal rule deviation most be declared and documented.

Deviations can be classified in two types: project or specific. Project deviations occur when a guideline in particular is breached under certain circumstance through the entirety of the project source code, caused by the project requirements itself. A specific deviation occurs when a guideline is violated in only one instance of only one file of the project. A format for a formal deviation declaration, filled for a rule breach in the Modbus code can be found in Appendix B.

It's important to remark that only required rules need a formal deviation declaration, since mandatory rules cannot be violated to claim MISRA compliance for a project. Advisory rules deviations can be documented too but this not obligatory by MISRA standards.

The fields required for the formal deviation documentation are as follows:

- **Deviation number and source file:** A number most be assigned to the deviation for ease of identification. Also the source file where the rule violation occurs must be in declared.

- **Type of deviation**: state if the deviation is of specific or system type.
- **Description of the circumstances causing the deviation from the rule:** A brief but clear description of the causes and circumstances of the deviation must be written.
- **Justification to the rule deviation**: Include an evaluation of the possible dangers caused by the rule violation compared against other possible solutions.
- **Possible secondary effects due to the rule deviation**: If the rule violation could cause other problems they should be documented here.
- **Actions taken or needed to guarantee no other problems due to this deviation**: Any need test or evaluations done to guarantee no further problems should be documented here.

The full deviation document format, filled for the cases where it was necessary for the Modbus code modifications can be seen in full in Appendix B.

6.3 MISRA C static check tool C-STAT configuration and results

For the automatic check of the MISRA compliance of the Modbus code the following configuration for C-STAT was used; first a static analysis was used with the factory settings on the original version on of the Modbus.c and Modbus.h files using the libraries defined in this files.

For the analysis of the Modbus files a total of 48 functions, 8 header files and 4 source files were analyzed using C-STAT version 1.5.2 in two configurations: factory recommended MISRA checks and all MISRA checks enabled.

The factory settings use 173 of 213 C-STAT specific checks and 175 of 226 possible checks based on the MISRA C:2012 guidelines.

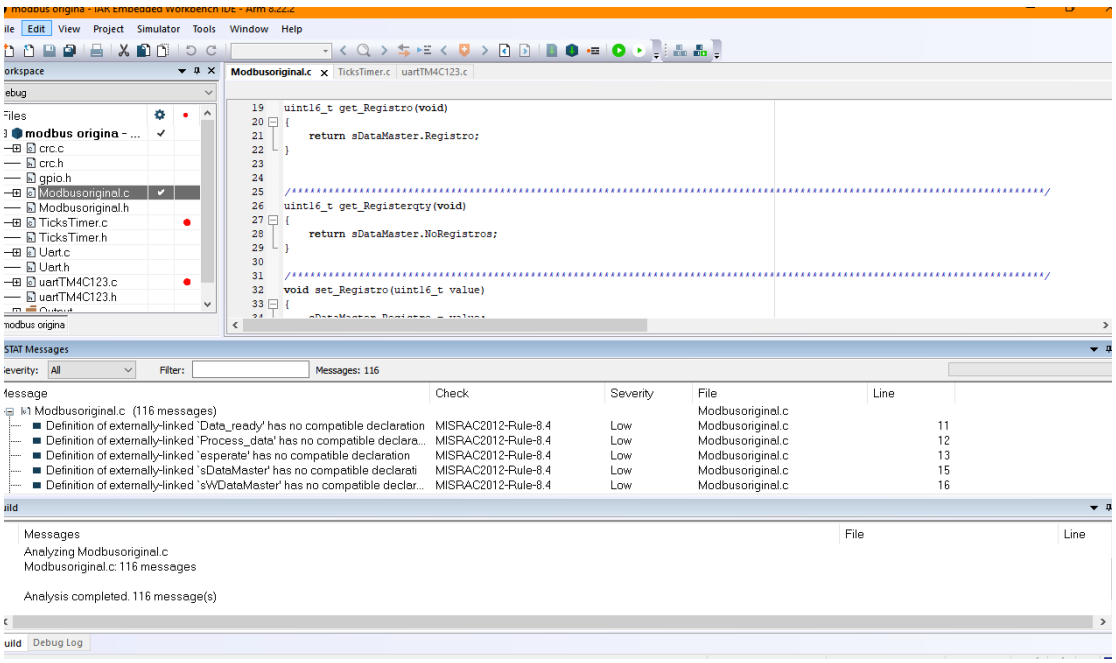

Figure 2. Original Modbus files analysis with factory settings.

A second analysis was made this time with the same recommended C-stat checks but using all 226 possible checks for the MISRA C:2012 guidelines.

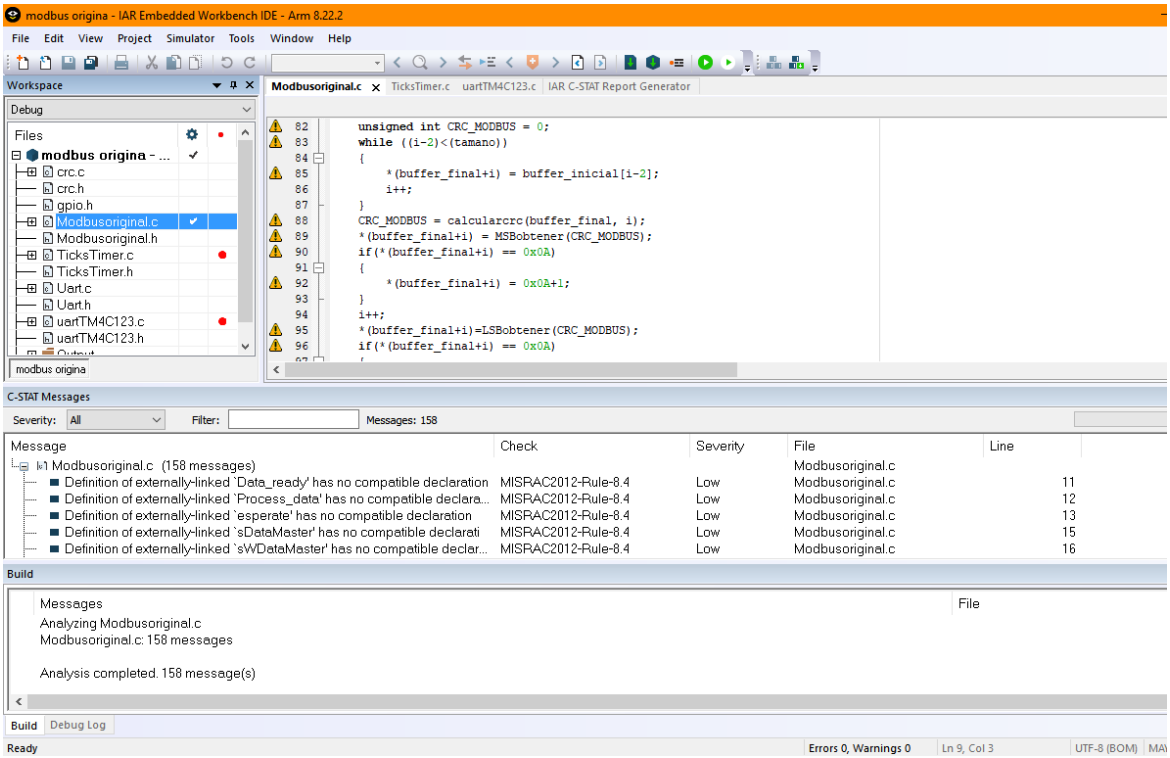

Figure 3. Original Modbus files with all MISRA checks enabled.

The results of these analysis were as follows:

For the factory settings analysis of the unmodified Modbus files the result was a total of 173 error messages. In Table 1 the total resulting messages and the MISRA violation they correspond to are shown.

Table 1. Factory settings C-stat analysis results for the original Modbus files.

Tag	Messages	Tag	Messages	Tag	Messages
MISRAC2012-Rule-10.3	59	MISRAC2012-Rule-10.4_a	31	MISRAC2012-Rule-2.2_c	23
MISRAC2012-Rule-18.4	16	MISRAC2012-Rule-8.4	12	MISRAC2012-Rule-17.7	11
MISRAC2012-Rule-21.1	7	MISRAC2012-Rule-17.3	6	MISRAC2012-Rule-8.5_a	4
MISRAC2012-Rule-10.1_R4	2	MISRAC2012-Rule-8.2_a	1	RED-unused-assign	1

The following figure shows additional data on the amount of times a same rule is violated, the source file of origin where the violations happen and the severity of the found violations.

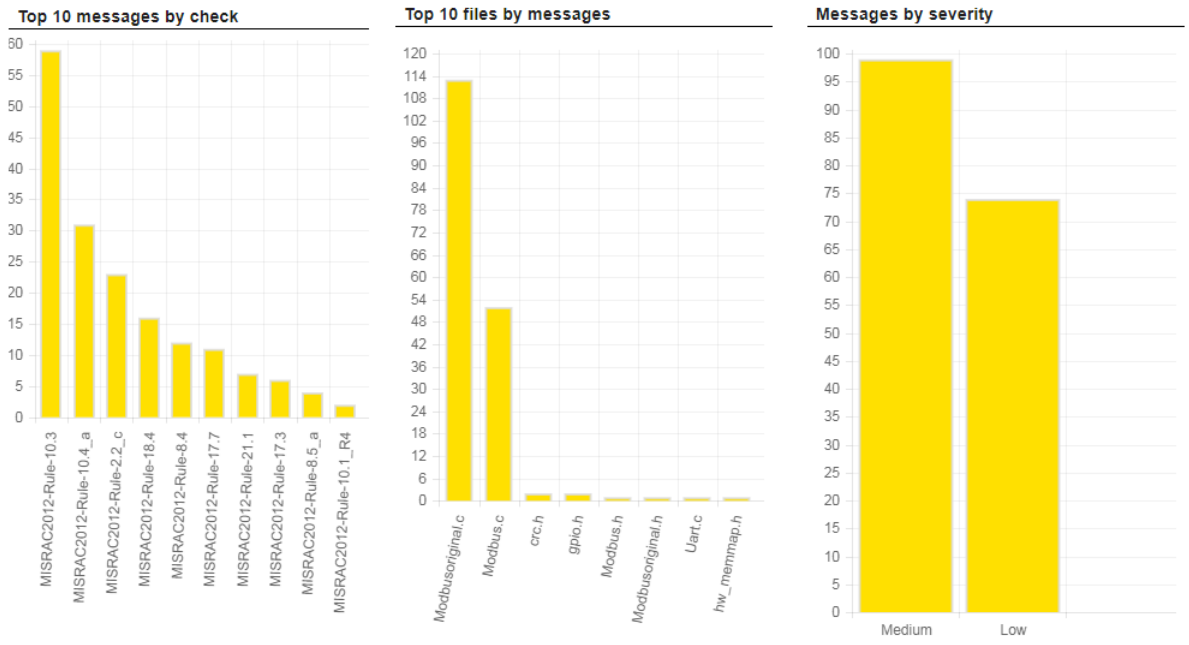

Figure 4. C-STAT Report result of the original files with factory settings

As can be seen Rule 10.3 has the most breaches, having almost double the breaches of rule 10.4. While the origin of the breaches is mostly the original Modbus.c file. In the case of the severity of the violations only medium and low severity breaches were found.

For the Full MISRA C checks analysis of the original Modbus files the result was a total of 215 error messages. In Table 2 the total resulting messages and the corresponding MISRA violation are shown.

Table 2. All MISRA checks enabled results for the original Modbus files.

Tag	Messages	Tag	Messages	Tag	Messages
MISRAC2012-Rule-10.3	59	MISRAC2012-Rule-10.4_a	31	MISRAC2012-Rule-13.3	28
MISRAC2012-Rule-2.2_c	23	MISRAC2012-Rule-18.4	16	MISRAC2012-Rule-8.4	12
MISRAC2012-Rule-17.7	11	MISRAC2012-Dir-4.6_a	8	MISRAC2012-Rule-21.1	7
MISRAC2012-Rule-17.3	6	MISRAC2012-Rule-8.5_a	4	MISRAC2012-Dir-4.4	3
MISRAC2012-Rule-10.1_R4	2	MISRAC2012-Dir-4.8	1	MISRAC2012-Rule-12.1	1
MISRAC2012-Rule-15.5	1	MISRAC2012-Rule-8.2_a	1	RED-unused-assign	1

The following figure shows the graphs generated by the C-STAT report which gives additional data on the amount of times a same rule is violated, the source file of origin where the violations happen and the severity of the found violations.

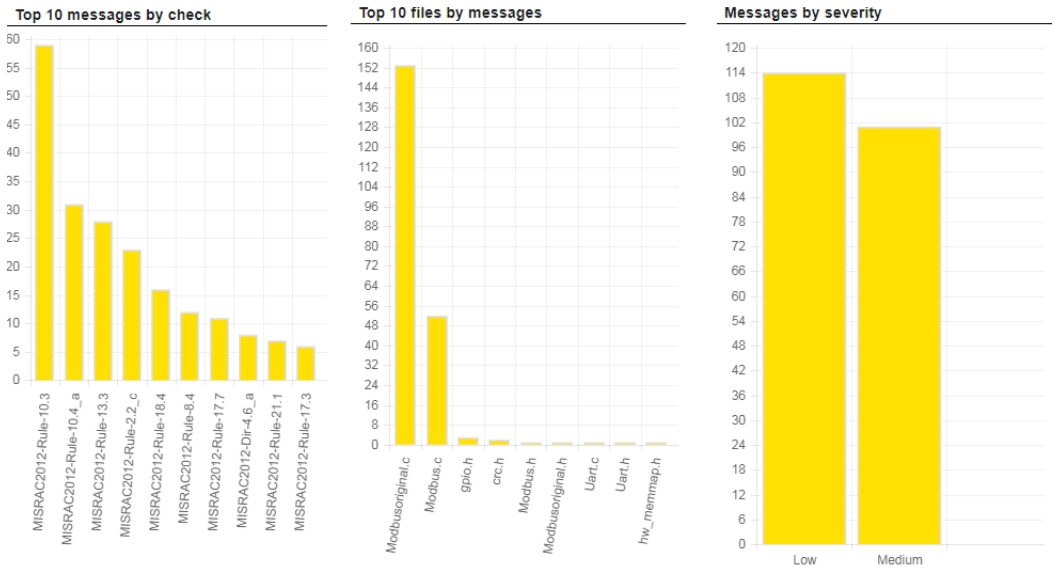

Figure 5. C-STAT Report result of the original files with all MISRA checks enabled.

As can be seen Rule 10.3 again has the most breaches, but in general the same breaches as in the first analysis still are the most common. While the original Modbus.c file was found to have almost 40 more breaches. Still no high severity violations are found.

Chapter 7

Modbus Source Code Modifications

The following are a list of changes done on the Modbus code to correct any rule breach. The changes specify which functions are being modified, but only the relevant part of the code that is being changed is shown.

For directive 4.5

Identifiers in the same name space with overlapping visibility should be typographically unambiguous.

Some single letter variables were changed to make them harder to confuse between them for example some “j” named variables were changed to “f” to avoid confusions with “i” variables.

For directive 4.6

typedefs that indicate size and signedness should be used in place of basic numerical types.

The following changes were made in the next functions:

```
uint8_t Armar_Cadena(uint8_t *buffer_inicial, FModbusType funcion, uint8_t
                              slave, uint8_t *buffer_final, uint32_t
tamano)
{
  uint32_t CRC_MODBUS  = 0;  /* unsigned char was changed for  uint32_t */

/*****************************************************************/

bool comprobar_Datos_Modbus (uint8_t *datos_modbus_string, char tamano2,
                                                          uint8_t slave)
{
  uint32_t crcmodbus = 0;      /* unsigned char was changed for uint32_t */

/*****************************************************************/
```

For rule 2.2:

There shall be no dead code

The unused initialization values for many declared variables was erased. It was verified that this variables were assigned a value before being used in other expressions. In in most cases in the following code extracts only the expressions " = 0;" which was never used was removed, where other changes were made it is specified in comments:

In the function write_data_master:

```
uint8_t  k;
uint16_t read_qtyregisters;
uint8_t  pending_bytesMod;
uint32_t initial_tick;
uint32_t actual_ticks;
bool     correctinfo;  /* "=false;" was removed */
```

In the function Armar_Cadena

```
uint8_t  datos_a_enviar;
uint32_t CRC_MODBUS;
```

In the function Answer_Modbus:

```
uint8_t k;
```

In the function comprobar_Datos_Modbus:

```
char slave_received;
char MSBCRC;
char LSBCRC;
char MSB_Calculado;
char LSB_Calculado;
uint32_t crcmodbus;

crcmodbus = 0;      /* this  line of code was erased */
```

In the function decodificar_cadena_Slave:

```
bool preparar_informacion;    /* "=false;" was removed */
```

In the functiondecodificar_cadena_Master:

```
uint32_t initial_tick;
uint32_t final_tick;
```

In the function decodificar_cadena_Slave_Data:

```
bool preparar_informacion;    /*  "=false;" was removed */
```

In the function decodificar_cadena_Master_

```
bool preparar_informacion;     /*  "=false;" was removed */
```

```
/* Also in this function the following unused variables and
   code were erased */
  uint32_t initial_tick;
  uint32_t final_tick;
  initial_tick = getTicks();
  final_tick   = getTicks();
```

In the function sendCoilMaster:

```
  uint32_t actual_ticks ;
  uint32_t initial_tick;
  uint8_t  f;
```

For rule 8.2:

Function types shall be in prototype form with named parameters.

Some functions not needing parameters were missing the void statement to be MISRA complaint.

In the Modbus.c file

```
/* the void sentence was added, as no parameters are needed*/
uint8_t getaddress(void)
```

In the TicksTimer.h file:

```
/* the void sentence was added, as no parameters are needed*/
void iniciarTimerTicks(void);
```

For rule 10.1:

Operands shall not be of an inappropriate essential type

In the function comprobar_Datos_modbus of the Modbus.c file the next changes were made to avoid using a char type variable in arithmetic subtraction:

```
int32_t tamano20 = tamano2 - '0'; /* convertion of the value of char
                                     tamano2 to int type          */
int32_t tamano23 = tamano20 - 3;     /* equivalent value to char
                                        tamano2 - 3               */
int32_t tamano22 = tamano20 - 2;     /* equivalent value to char
                                                tamano2 - 3     */
    if(slave_received == slave)
    {

 /* substitution of the value which previously was of type char   */
```

```
        MSBCRC = datos_modbus_string[tamano23];

 /* substitution of srithmetic substraction using a char variable */
       LSBCRC = datos_modbus_string[tamano22];
```

For rule 10.3:

The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.

This rule was the one with the most error messages thorough the code. Some of this errors were corrected but some were left as modifying the byte size of a global function parameter could cause further unforeseen problems. Deviation 1 in Appendix D covers this deviation.

However cases where local variables could be corrected by changing their byte size without causing new or other errors and where no bytes of data from the variables could be loss were corrected:

In the function Armar_Cadena for example:

```
uint32_t datos_a_enviar;  /* was uint8_t */
```

In the crc.h file the some function parameters were changed from uint8_t to uint_32:

```
uint8_t calcularcrc(uint8_t *buffercrc, uint32_t tamano);
uint8_t LSBobtener(uint32_t acomodar);
uint8_t MSBobtener(uint32_t acomodar);
```

For rule 10.4:

Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.

The violations to this rule were corrected as follows:

In the function Armar_Cadena:

```
uint8_t ten  = 10;      /* ten was added as a uint8_t variable  */
uint8_t once = 11;      /* once was added as a uint8_t variable */
uint32_t x    = 2;       /* two was added as a uint8_t variable  */
while ((i-two)<(tamano)) /* the signed constant 2 was changed for variable
                            two                                  */
{
```

```
    *(buffer_final+i) = buffer_inicial[i-x];
            /* The signed constant 2 was replaced by variable x to make the
         substraction between two variables of the same essential type*/
}

if(*(buffer_final+i) == ten)  /* the hexadecimal signed constant 0x0A was
                                 replaced by the uint_8 variable ten  */
{
    *(buffer_final+i) = once;    /* the expression "0x0A+1" was
                                 replaced by the uint_8 variable once */
}
```

Case where a value of zero was being assigned to a variable were verified and corrected as needed. For example:

In the function sendCoilMaster

```
while(intentos < 2) /* False positive; this is complaint due to a rule
                       exception explained in the rules for comparison
                       used signed constants */
```

In the function clear_data**:**

```
WDataMaster.NoData = cero; /* signed const 0 was changed to uint_8 cero */
```

In the function decodificar_cadena_Slave

```
uint32_t cero = 0;  /* variable cero added */

if(pendingBytes2 != cero) /* signed const 0 was changed to uint_8 cero */
```

For rule 12.1:

The precedence of operators within expression should be made explicit.

The only expression where adding a parenthesis made the operations precedence clearer was the following, only one set of parenthesis was added to the code:

```
for(uint8_t j = 4; j < (4 + recovering_bytes); j++)
     /* the parenthesis was added in (4 + recovering_bytes)*/
```

For rule 13.3:

A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator.

The breaches of this rule were found through the code in multiple expressions similar to the following:

```
sMasterData -> address  = datos_ModbusRx[i++];
sMasterData -> Function = datos_ModbusRx[i++];
```

To make this sentences complaint with rule13.3 the following change could be applied.

```
sMasterData -> address  = datos_ModbusRx[i];
i++;
sMasterData -> Function = datos_ModbusRx[i];
i++;
```

It was decided to keep the code without changes since the original code seems easier to read. Since this is an advisory rule no formal documentation of the deviation is needed.

For rule 15.5:

A function should have a single point of exit at the end.

There was only one breach to this rule in the function it was found in the next function:

```
bool get_Data(uint8_t *ui8Datos)
{
    uint8_t cero  = 0;

    if(sWDataMaster.NoData != cero)
    {
        for(uint8_t i = cero; i < sWDataMaster.NoData; i++)
        {
            ui8Datos[i] = sWDataMaster.Data[i];
        }
        return true;
    }
    return false;     /* Second return in a function, fault to rule 15.5 */
}
```

Since this is an advisory rule with no chances of causing unknown behaviors, no formal deviation was documented.

For rule 17.3:

A function shall not be declared implicitly.

Most likely a false positive due to lack of libraries. The breaches were found in the following functions:

```
iniciarEnable();
Driver_enable();
```

They were not modified.

For rule 17.7:

The value returned by a function having non-void return type shall be used.

Multiple breaches of this rule were found. In all the cases the "(void)" sentence was added to the function to make the code MISRA complaint.

```
(void)uartGetData(MODBUS, datos_ModbusRx, pendingBytes2);
(void)uartGetData(MODBUS, Received_Data, pending_bytesMod);
(void)Driver_enable();
(void)Receiver_enable();
(void)iniciarEnable();
```

For rule 18.4:

The +, -, += and -= operators should not be applied to an expression of pointer type.

While no -, +=, or -= operators were used, multiple breaches to this rule were found. In all cases the breaching code was found to be the operation "*(buffer_final + i)".

A formal deviation was not declared since the rule is advisory so it was decided to keep the original code without changes.

For rule 21.1:

#define and #undef shall not be used on a reserved identifier or reserved macro name.

The violations to this rule were found as follows:

In Modbus.h:

```
#ifndef _UART_PROCESS_H
#define _UART_PROCESS_H
```

In hw_memmap.h:

```
#ifndef __HW_MEMMAP_H__
#define __HW_MEMMAP_H__
```

In crc.h:

```
#ifndef _CRC_H_
#define _CRC_H_
```

MISRA doesn't allow the use of macro names that start with underscore. Also it was decided to leave this macros unmodified since them could be referenced on other files and changing the names could give further errors.

A correction to this rule would be to remove the underscore from the macro's names or change the macro names entirely, but since this violations occur in a #ifndef structure and because of possible cross references of this macro names in other files it was decide to leave it unchanged.

Chapter 8

C-STAT Final Results after MISRA C Modifications to the Source Code

After the modifications were done on the original Modbus files, C-STAT analyses were made again to the changed files. In this instance two different configurations were selected for the analyses: One with all MISRA checks selected and one without checks for the rules that were declared as deviations and the advisory rules that were not followed.

For the case with all MISRA C checks are enabled, a total of 56 violations were found.
The results can be seen in the next table

Table 3. C-STAT results with all MISRA checks enabled.

Tag	Messages	Tag	Messages	Tag	Messages
MISRAC2012-Rule-13.3	28	MISRAC2012-Rule-10.4_a	10	MISRAC2012-Rule-8.4	6
MISRAC2012-Rule-21.1	4	MISRAC2012-Dir-4.4	3	MISRAC2012-Rule-17.3	3
MISRAC2012-Rule-8.5_a	2	MISRAC2012-Dir-4.8	1	MISRAC2012-Rule-15.5	1

The following graphs show the final result of the current modifications with all the possible MISRA checks for the C-STAT analysis. As one can see from the graphs and table most of this rules violations were declared as formal deviations, and a majority of them are of a low severity.

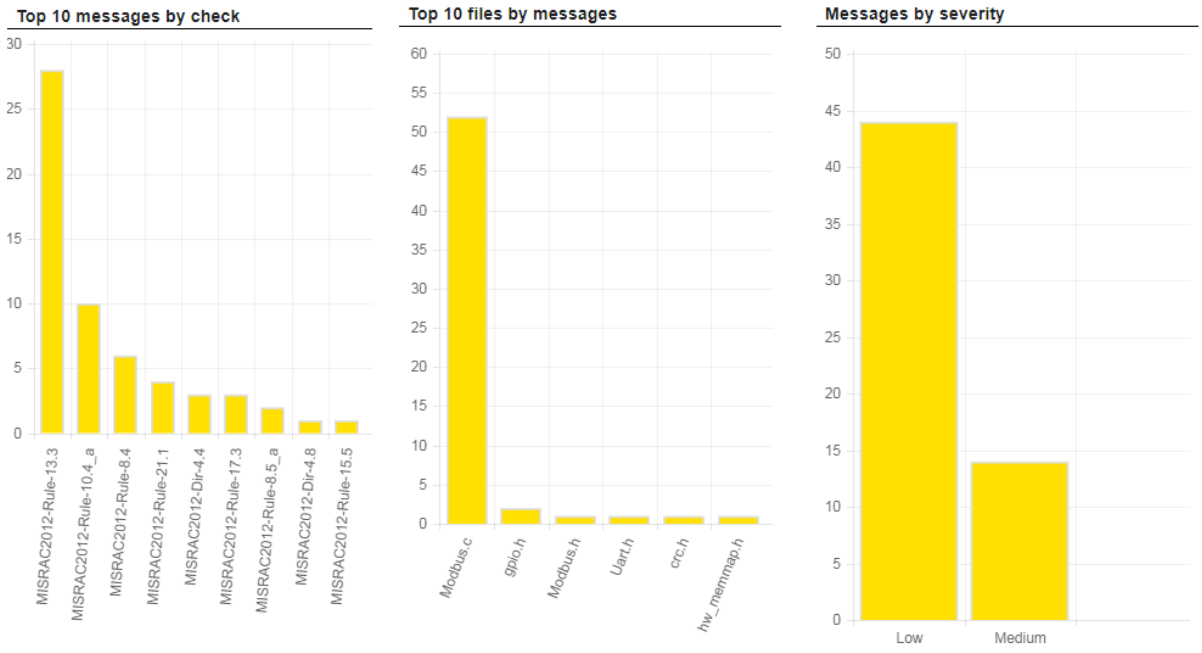

Figure 6. C-STAT report graphs for all MISRA checks enabled.

For the case with the disabled checks for rules with deviations and advisory rules which were not followed a total of 7 error messages were found. All of them were checked and either originated from an advisory rule or an apparent false positive.

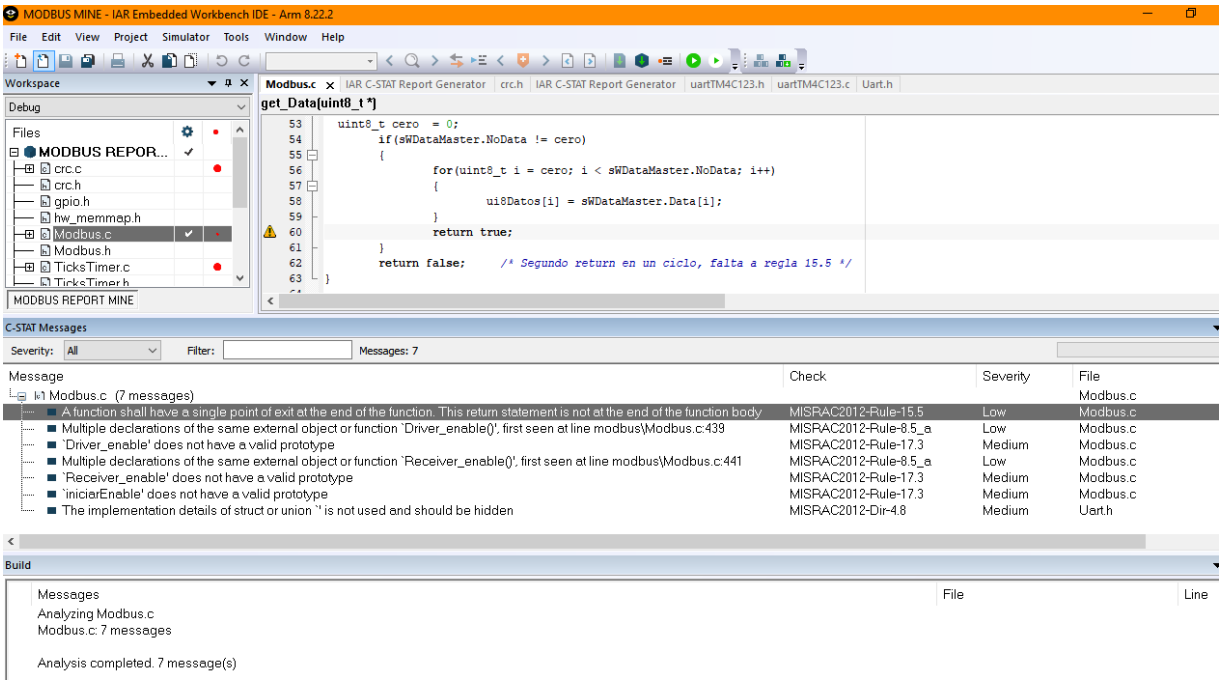

Figure 7. IAR screen after analysis execution with deviation considered.

The results for this C-stat analysis configuration can be seen in the next table:

Table 4. C-STAT results for modified Modbus files.

Tag	Messages
MISRAC2012-Rule-17.3	3
MISRAC2012-Rule-8.5_a	2
MISRAC2012-Dir-4.8	1
MISRAC2012-Rule-15.5	1
RED-unused-assign	1

Finally the following graphics show the origin files for these 7 messages, with one warning in the uart.c and uart.h being the macro warning to directive 4.8 of an uderscore at the begining of a macro name.

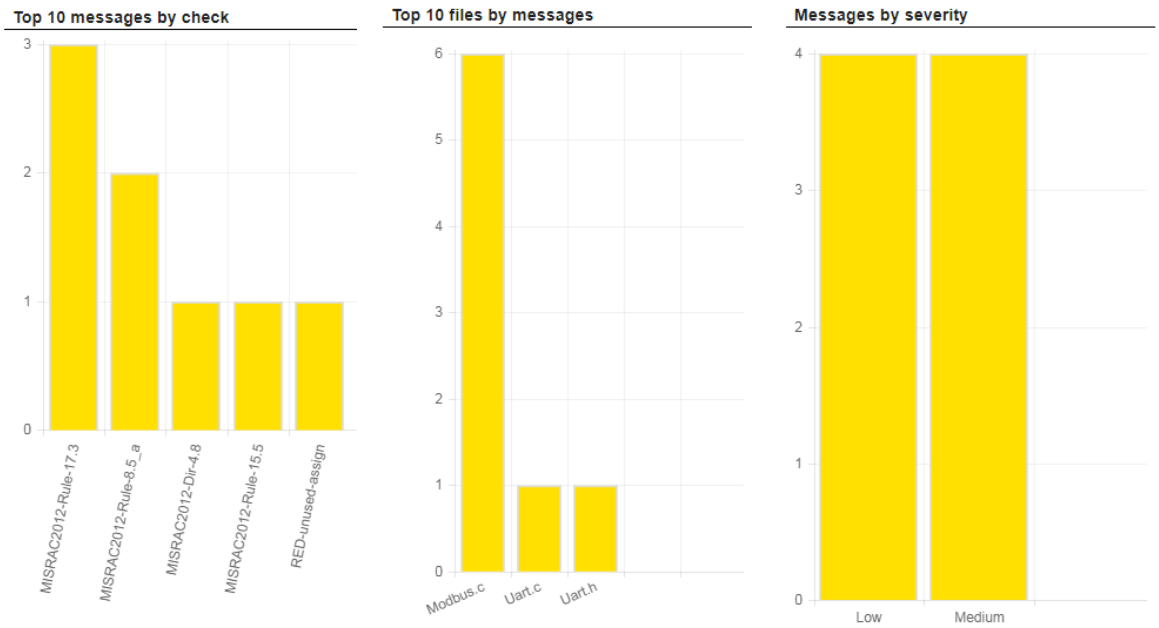

Figure 8. Final C-STAT report graphs with deviations considered.

The final version of the modified Modbus code can be seen in full in appendix E.

Conclusions

Even if the MISRA recommendation of starting the creation of code with the MISRA C guidelines in mind from the begging of a project were not followed in this project's case, the results for the modification of the Modbus source files demonstrate that any code can be "ported" to make it MISRA complaint, although this task can turn to be more complex and time consuming than what it seems.

It can also be validated that the use of a C-style format standard for the sake of simplifying the lecture, flow, logic and structure of a source file written code between members of a project workgroup is a helpful tool to the development of source code since it makes easier to understand and read the code created by other members of the workgroup.

One of the harder parts of modifying the code was the lack of documentation of its functionality, and limited knowledge of the C language which difficulted the initial work. It can be concluded that previous skills in C coding can greatly help when applying MISRA. Still the changes made should help make the code more safe and unlikely to present unknown behaviors.

Finally it can be said that from the work of this project, the final Modbus modified code should be safer, less unlikely to bugs, easier to maintain and to port and that the developed method to apply MISRA with its documentation formats, the compliance matrix and the deviation documentation format, should make using MISRA more efficient, structured and easier.

Future work

As a future work a study for coming with a set of easy to follow rules or guidelines to avoid breaches of rule 13.3 should be proven very useful in reducing the time and work needed for a project, to make its source code more efficiently MISRA C complaint since it can be seen from the results of this project that keeping a uniform and MISRA C complaint use of the essential type for variables and functions declarations and operations is probably the best solution to avoid any kind of violations to rule 13.3.

Finally, the creation of some development process for the review of code during execution could help even further in the quest for high safety and security by avoiding errors of data overflow, invalid shifting of values, memory errors due to accessing invalid memory regions and accesses to uninitialized data.

Whatever method is decided for runtime checking it would certainly need a standardized record and documentation protocol just like the one that was produced for this MISRA C project.

References

[1] Stephen Cass and Parthasaradhi Bulusu. “https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018”, IEEE Spectrum, 2018

[2] *MISRA Compliance:2016 Achieving compliance with MISRA Coding Guidelines*, HORIBA MIRA Limited, 2016.

[3] *MISRA C:2012 Guidelines for the use of the C language in critical systems*, MIRA Limited, March 2013

[4] “SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems”. Software Engineering Institute. Carnegie Mellon University. 2016 Edition.

[5] Michael Barr, “Embedded C Coding Standard”. Barr group PDF document. 2013

[6] “https://cwe.mitre.org/about/index.html”, CWE official web page.

[7] Rich Quinnell “Enhanced Guidelines Appear for Safe, Secure C Programs. MISRA updates guidelines”, EE times 2016

[8] Software Engineering Laboratory Series. Sel-94-003. “*C Style Guide*”, National Aeronautics and Space Administration. August 1994.

[9] “ESA Style Guide for 'C' coding”. Expert Solutions Australia Pty. Ltd. 1991.

[10] Mark Brader, David Keppel, Henry spencer. “Recommended C Style and Coding Standards”, Revision 6.0, February 1997

[11] “MISRA C FAQ list” MISRA Consortium

[12] “C-STAT Static Analysis Guide”, IAR Systems AB and Synopsys, Inc. 7th edition, October 2017

Appendix A: MISRA Directives

Table 5. The Implementation

Directive	Category	Applies to	Analysis	Description
Dir 1.1	Required	C90, C99		Any implementation-defined behavior on which the output of the program depends shall be documented and understood.

Table 6. Compilation and build

Directive	Category	Applies to	Analysis	Description
Dir 2.1	Required	C90, C99		All source files shall compile without any compilation errors.

Table 7. Requirements traceability

Directive	Category	Applies to	Analysis	Description
Dir 3.1	Required	C90, C99		All code shall be traceable to documented requirements.

Table 8. Code design

Directive	Category	Applies to	Analysis	Description
Dir 4.1	Required	C90, C99		Run-time failures shall be minimized
Dir 4.2	Advisory	C90, C99		All usage of assembly language should be documented
Dir 4.3	Required	C90, C99		Assembly language shall be encapsulated and isolated.
Dir 4.4	Advisory	C90, C99		Sections of code should not be “commented out.”
Dir 4.5	Advisory	C90, C99		Identifiers in the same name space with overlapping visibility should be typographically unambiguous.

Dir 4.6	Advisory	C90, C99	Typedefs that indicate size and signedness should be used in place of the basic numerical types.
Dir 4.7	Required	C90, C99	If a function returns error information, then the error information shall be tested.
Dir 4.8	Advisory	C90, C99	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden.
Dir 4.9	Advisory	C90, C99	A function should be used in preference to a function-like macro where they are interchangeable.
Dir 4.10	Required	C90, C99	Precautions shall be taken in order to prevent the contents of a header file being included more than once.
Dir 4.11	Required	C90, C99	The validity of values passed to library functions shall be checked.
Dir 4.12	Required	C90, C99	Dynamic memory allocation shall not be used.
Dir 4.13	Advisory	C90, C99	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.

Appendix B: MISRA Rules

Table 9. A standard C environment

Rule	Category	Applies to	Analysis	Description
1.1	Required	C90, C99	D, STU	Any implementation-defined behavior on which the output of the program depends shall be documented and understood.
1.2	Advisory	C90, C99	U, STU	Language extensions should not be used.
1.3	Required	C90, C99	U, Sys	There shall be no occurrence of undefined or critical unspecified behavior.

Table 10. Unused code

Rule	Category	Applies to	Analysis	Description
2.1	Required	C90, C99	U, Sys	A project shall not contain unreachable code.
2.2	Required	C90, C99	U, Sys	There shall be no dead code.
2.3	Advisory	C90, C99	D, Sys	A project should not contain unused type declarations.
2.4	Advisory	C90, C99	D, Sys	A project should not contain unused tag declarations.
2.5	Advisory	C90, C99	D, Sys	A project should not contain unused macro declarations.
2.6	Advisory	C90, C99	D, STU	A function should not contain unused label declarations.
2.7	Advisory	C90, C99	D, STU	There should be no unused parameters in functions.

Table 11. Comments

Rule	Category	Applies to	Analysis	Description
3.1	Required	C90, C99	D, STU	The character sequences /* and // shall not be used within a comment.
3.2	Required	C99	D, STU	Line-splicing shall not be used in // comments.

Table 12. Character sets and lexical conventions

Rule	Category	Applies to	Analysis	Description
4.1	Required	C90, C99	D, STU	Octal and hexadecimal escape sequences shall not be terminated.
4.2	Advisory	C90, C99	D, STU	Trigraphs should not be used.

Table 13. Identifiers

Rule	Category	Applies to	Analysis	Description
5.1	Required	C90, C99	D, Sys	External identifiers shall be distinct.
5.2	Required	C90, C99	D, STU	Identifiers declared in the same scope and name space shall be distinct.
5.3	Required	C90, C99	D, STU	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.
5.4	Required	C90, C99	D, STU	Macro identifiers shall be distinct.
5.5	Required	C90, C99	D, STU	Identifiers shall be distinct from Marco names.
5.6	Required	C90, C99	D, Sys	A typedef name shall be unique identifier.
5.7	Required	C90, C99	D, Sys	A tag name shall be a unique identifier.
5.8	Required	C90, C99	D, Sys	Identifiers that define objects or functions with external linkage shall be unique.
5.9	Advisory	C90, C99	D, Sys	Identifiers that define objects or functions with internal linkage should be unique.

Table 14. Types

Rule	Category	Applies to	Analysis	Description
6.1	Required	C90, C99	D, STU	Identifiers that define objects or functions with external linkage shall be unique.
6.2	Advisory	C90, C99	D, STU	Identifiers that define objects or functions with internal linkage should be unique.

Table 15. Literals and constants

Rule	Category	Applies to	Analysis	Description
7.1	Required	C90, C99	D, STU	Octal constants shall not be used.
7.2	Required	C90, C99	D, STU	A “u” or “U” suffix shall be applied to all integer constants that are represented in an unsigned type.
7.3	Required	C90, C99	D, STU	The lowercase “l” shall not be used in a literal suffix.
7.4	Required	C90, C99	D, STU	A string literal shall not be assigned to an object unless the object’s type is pointer to “const-qualified char.”

Table 16. Declarations and definitions

Rule	Category	Applies to	Analysis	Description
8.1	Required	C99	D, STU	Type shall be explicitly specified.
8.2	Required	C90, C99	D, STU	Function types shall be in prototype form with named parameters.
8.3	Required	C90, C99	D, Sys	All declarations of an object or function shall use the same names and types qualifiers.

8.4	Required	C90, C99	D, STU	A compatible declaration shall be visible when an object or function with external linkage is defined.
8.5	Required	C90, C99	D, Sys	An external object or function shall be declared once in one and only one file.
8.6	Required	C90, C99	D, Sys	An identifier with external linkage shall have exactly one external definition.
8.7	Advisory	C90, C99	D, Sys	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.
8.8	Required	C90, C99	D, STU	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.
8.9	Advisory	C90, C99	D, Sys	An object should be defined at block scope if its identifier only appears in a single function.
8.10	Required	C99	D, STU	An infinite function shall be declared with the static storage class.
8.11	Advisory	C90, C99	D, STU	When an array with external linkage is declared, its size should be explicitly specified.
8.12	Required	C90, C99	D, STU	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.
8.13	Advisory	C90, C99	U, Sys	A pointer should point to a const-qualified type whenever possible.
8.14	Required	C99	D, STU	The restrict type qualifier shall not be used.

Table 17. Initialization

Rule	Category	Applies to	Analysis	Description
9.1	Mandatory	C90, C99	U, Sys	The value of an object with automatic storage duration shall not be read before it has been set.
9.2	Required	C90, C99	D, STU	The initializer for an aggregate or union shall be enclosed in braces.
9.3	Required	C90, C99	D, STU	Arrays shall not be partially initialized.
9.4	Required	C99	D, STU	An element of an object shall not be initialized more than once.
9.5	Required	C99	D, STU	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.

Table 18. The essential type model

Rule	Category	Applies to	Analysis	Description
10.1	Required	C90, C99	D, STU	Operands shall not be of an inappropriate essential type.
10.2	Required	C90, C99	D, STU	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.
10.3	Required	C90, C99	D, STU	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.
10.4	Required	C90, C99	D, STU	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.
10.5	Advisory	C90, C99	D, STU	The value of an expression should not be cast to an inappropriate essential type.
10.6	Required	C90, C99	D, STU	The value of a composite expression shall not be assigned to an object with wider essential type.

10.7	Required	C90, C99	D, STU	If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have a wider essential type.
10.8	Required	C90, C99	D, STU	The value of a composite expression shall not be cast to a different essential type category or a wider essential type.

Table 19. Pointer type conversions

Rule	Category	Applies to	Analysis	Description
11.1	Required	C90, C99	D, STU	Conversions shall not be performed between a pointer to a function and any other type.
11.2	Required	C90, C99	D, STU	Conversions shall not be performed between a pointer to an incomplete type and any other type.
11.3	Required	C90, C99	D, STU	A cast shall not be performed between a pointer to a pointer to object type and a pointer to a different object type.
11.4	Advisory	C90, C99	D, STU	A conversion should not be performed between a pointer to object and integer type.
11.5	Advisory	C90, C99	D, STU	A conversion should not be performed from pointer to void into pointer to object.
11.6	Required	C90, C99	D, STU	A cast shall not be performed between pointer to void and an arithmetic type.
11.7	Required	C90, C99	D, STU	A cast shall not be performed between pointer to object and a non-integer arithmetic type.
11.8	Required	C90, C99	D, STU	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.
11.9	Required	C90, C99	D, STU	The macro NULL shall be the only permitted from of integer null pointer constant.

Table 20. Expressions

Rule	Category	Applies to	Analysis	Description
12.1	Advisory	C90, C99	D, STU	The precedence of operators within expressions should be made explicit.
12.2	Required	C90, C99	U, Sys	The right-hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left-hand operand.
12.3	Advisory	C90, C99	D, STU	The comma operator should not be used.
12.4	Advisory	C90, C99	D, STU	Evaluation of constant expressions should not lead to unsigned integer wrap-around.

Table 21. Side effects

Rule	Category	Applies to	Analysis	Description
13.1	Required	C99	U, Sys	Initializer list shall not contain persistent side effects.
13.2	Required	C90, C99	U, Sys	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.
13.3	Advisory	C90, C99	D, STU	A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator.
13.4	Advisory	C90, C99	D, STU	The result of an assignment operator should not be used.
13.5	Required	C90, C99	U, Sys	The right hand operand of a logical && or || operator shall not contain persistent side effects.
13.6	Mandatory	C90, C99	D, STU	The operand of the sizeof operator shall not contain any expression which has potential side effects.

Table 22. Control statement expressions

Rule	Category	Applies to	Analysis	Description
14.1	Required	C90, C99	U, Sys	A loop counter shall not have essentially floating type.
14.2	Required	C90, C99	U, Sys	A for loop shall be well-formed.
14.3	Required	C90, C99	U, Sys	Controlling expressions shall be invariant.
14.4	Required	C90, C99	D, STU	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type.

Table 23. Control flow

Rule	Category	Applies to	Analysis	Description
15.1	Advisory	C90, C99	D, STU	The goto statement should not be used.
15.2	Required	C90, C99	D, STU	The goto statement shall jump to a label declared later on the same function.
15.3	Required	C90, C99	D, STU	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.
15.4	Advisory	C90, C99	D, STU	There should be no more than one break or goto statement used to terminate any iteration statement.
15.5	Advisory	C90, C99	D, STU	A function should have a single point of exit at the end.
15.6	Required	C90, C99	D, STU	The body of an iteration-statement or a selection-statement shall be a compound-statement.
15.7	Required	C90, C99	D, STU	All if… else if constructs shall be terminated with an else statement.

Table 24. Switch statements

Rule	Category	Applies to	Analysis	Description
16.1	Required	C90, C99	D, STU	All Switch statements shall be well formed.
16.2	Required	C90, C99	D, STU	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.
16.3	Required	C90, C99	D, STU	An unconditional break statement shall terminate every switch-clause.
16.4	Required	C90, C99	D, STU	Every switch statement shall have a default label.
16.5	Required	C90, C99	D, STU	A default label shall appear as either the first or the last switch label of a switch statement.
16.6	Required	C90, C99	D, STU	Every switch statement shall have at least two switch clauses.
16.7	Required	C90, C99	D, STU	A switch-expression shall not have essentially Boolean type.

Table 25. Functions

Rule	Category	Applies to	Analysis	
17.1	Required	C90, C99	D, STU	The features of <stdarg.h> shall not be used.
17.2	Required	C90, C99	U, Sys	Functions shall not call themselves, either directly or indirectly.
17.3	Mandatory	C90	D, STU	A function shall not be declared implicitly.
17.4	Mandatory	C90, C99	D, STU	All exit paths from a function with a non-void return type shall have an explicit return statement with an expression.
17.5	Advisory	C90, C99	U, Sys	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.

17.6	Mandatory	C99	D, STU	The declaration of an array parameter shall not contain the static keyword between the [ ].
17.7	Required	C90, C99	D, STU	The value returned by a function having non-void return type shall be used.
17.8	Advisory	C90, C99	U, Sys	A function parameter should not be modified.

Table 26. Pointer and arrays

Rule	Category	Applies to	Analysis	
18.1	Required	C90, C99	U, Sys	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.
18.2	Required	C90, C99	U, Sys	Subtraction between pointers shall only be applied to pointers that address elements of the same array.
18.3	Required	C90, C99	U, Sys	The relationship operators >, >=, < and =< shall not be applied to objects of pointer type except where they point into the same object.
18.4	Advisory	C90, C99	D, STU	The +, -, += and -= operators should not be applied to expressions of pointer type.
18.5	Advisory	C90, C99	D, STU	Declarations should contain no more than two levels of pointer nesting.
18.6	Required	C90, C99	U, Sys	The address of an object with automatic storage shall not be copied to another object that persist after the first object has ceased to exist.
18.7	Required	C99	D, STU	Flexible array members shall not be declared.
18.8	Required	C99	D, STU	Variable-length array types shall not be used.

Table 27. Overlapping storage

Rule	Category	Applies to	Analysis	
19.1	Mandatory	C90, C99	U, Sys	An object shall not be assigned or copied to an overlapping object.
19.2	Advisory	C90, C99	D, STU	The union keyword should not be used.

Table 28. Preprocessing directives

Rule	Category	Applies to	Analysis	
20.1	Required	C90, C99	D, STU	#include directives should only be preceded by preprocessor directives or comments.
20.2	Required	C90, C99	D, STU	The ‘, “ or \ characters and the /* or // character sequences shall not occur in a header file name.
20.3	Required	C90, C99	D, STU	The #include directive shall be followed by either a <filename> or “filename” sequence.
20.4	Required	C90, C99	D, STU	A macro shall not be defined with the same name as a keyboard.
20.5	Advisory	C90, C99	D, STU	#undef should not be used.
20.6	Required	C90, C99	D, STU	Tokens that look like a preprocessing directive shall not occur within a macro argument.
20.7	Required	C90, C99	D, STU	Expression resulting from the expansion of macro parameters shall be enclosed in parenthesis.
20.8	Required	C90, C99	D, STU	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.
20.9	Required	C90, C99	D, STU	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define’d before evaluation.
20.10	Advisory	C90, C99	D, STU	The # and ## preprocessor operations should not be used.

20.11	Required	C90, C99	D, STU	A macro parameter used as an operand to the # or ## operators shall not be immediately followed by a ## operator.
20.12	Required	C90, C99	D, STU	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators.
20.13	Required	C90, C99	D, STU	A line whose first token is # shall be a valid preprocessing directive.
20.14	Required	C90, C99	D, STU	All #else, #elif, and #endif preprocessor directives shall reside in the same file as the #if, #ifedf or #ifndef directive to which they are related.

Table 29. Standard libraries

Rule	Category	Applies to	Analysis	
21.2	Required	C90, C99	D, STU	#define and #undef shall not be used on a reserved identifier or reserved macro name.
21.2	Required	C90, C99	D, STU	A reserved identifier or macro name shall not be declared.
21.3	Required	C90, C99	D, STU	The memory allocation and deallocation functions of <stdlib.h> shall not be used.
21.4	Required	C90, C99	D, STU	The standard header file <setjmp.h> shall not be used.
21.5	Required	C90, C99	D, STU	The standard header file <signal.h> shall not be used.
21.6	Required	C90, C99	D, STU	The standard Library input/output functions shall not be used.
21.7	Required	C90, C99	D, STU	The atof, atoi, atoll, and atoll functions of <stdlib.h> shall not be used.
21.8	Required	C90, C99	D, STU	The library functions abort, exit, getenv and system of <stdlib.h> shall not be used.
21.9	Required	C90, C99	D, STU	The library functions bsearch and qsort of <stdlin.h> shall not be used.

21.10	Required	C90, C99	D, STU	The Standard Library time and date functions shall not be used.
21.11	Required	C99	D, STU	The standard header file <tgmath.h> shall not be used.
21.12	Advisory	C99	D, STU	The exception handling features of <fenv.h> should not be used.

Table 30. Resources

Rule	Category	Applies to	Analysis	
22.1	Required	C90, C99	U, Sys	All resources obtained dynamically by means of the standard Library functions shall be explicitly released.
22.2	Mandatory	C90, C99	U, Sys	A block of memory shall only be freed if it was allocated by means of a Standard library function.
22.3	Required	C90, C99	U, Sys	The same file shall not be open for read and write access at the same time on different streams.
22.4	Mandatory	C90, C99	U, Sys	There shall be no attempt to write to a stream which has been opened as read only.
22.5	Mandatory	C90, C99	U, Sys	A pointer to a FILE object shall not be dereferenced.
22.6	Mandatory	C90, C99	U, Sys	The value of a pointer to a FILE shall not be used after the associated stream has been closed.

Appendix C: Modbus MISRA C compliance matrix

DIRECTIVE	COMPILER		CHECK TOOL		MANUAL REVISION
	IAR E.W. v8.22	Other	C-STAT v1.5.2	OTHER	
Dir 1.1	No errors	N/A	Not checkable	N/A	Directive followed
Dir 2.1	No errors	N/A	Not checkable	N/A	Directive followed
Dir 3.1	No errors	N/A	Not checkable	N/A	Directive followed
Dir 4.1	No errors	N/A	Not checkable	N/A	Considered
Dir 4.2	No errors	N/A	Not checkable	N/A	Directive followed, not used
Dir 4.3	No errors	N/A	No errors	N/A	Directive followed, not used
Dir 4.4	No errors	N/A	No errors	N/A	Followed. Unused code erased
Dir 4.5	No errors	N/A	No errors	N/A	Reviewed. Minor changes done.
Dir 4.6	No errors	N/A	Errors	N/A	Directive followed
Dir 4.7	No errors	N/A	No errors	N/A	Directive followed
Dir 4.8	No errors	N/A	No errors	N/A	Directive followed
Dir 4.9	No errors	N/A	No errors	N/A	Directive followed
Dir 4.10	No errors	N/A	No errors	N/A	Directive followed
Dir 4.11	No errors	N/A	No errors	N/A	General review done. No obvious faults
Dir 4.12	No errors	N/A	No errors	N/A	Directive followed
Dir 4.13	No errors	N/A	No errors	N/A	General review done. No obvious faults
Rule 1.1	No errors	N/A	Not checkable	N/A	Manual review. No errors found
Rule 1.2	No errors	N/A	Not checkable	N/A	Manual review. No errors found
Rule 1.3	No errors	N/A	No errors	N/A	Manual review. No errors found
Rule 2.1	No errors	N/A	No errors	N/A	Checked
Rule 2.2	No errors	N/A	Multiple Errors	N/A	Corrected: Unused code erased.
Rule 2.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 2.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 2.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 2.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 2.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 3.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 3.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 4.1	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 4.2	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 5.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.6	No errors	N/A	No errors	N/A	Manual check: No errors found

Rule 5.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 5.9	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 6.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 6.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 7.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 7.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 7.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 7.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.2	No errors	N/A	2 Errors	N/A	Corrected: Parameters added
Rule 8.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.4	No errors	N/A	Error (low severity)	N/A	Manual check: Libraries related
Rule 8.5	No errors	N/A	1 Error	N/A	False positive
Rule 8.6	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 8.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.8	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 8.9	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.10	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.11	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.12	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 8.13	No errors	N/A	No errors	N/A	Not followed. No changes
Rule 8.14	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 9.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 9.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 9.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 9.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 9.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 10.1	No errors	N/A	Errors	N/A	Errors successfully corrected
Rule 10.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 10.3	No errors	N/A	Multiple Errors	N/A	Formal Deviation 1
Rule 10.4	No errors	N/A	No errors	N/A	Not all instances of the error corrected
Rule 10.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 10.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 10.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 10.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.3	No errors	N/A	No errors	N/A	Manual check: No errors found

Rule 11.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 11.9	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 12.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 12.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 12.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 12.4	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 13.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 13.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 13.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 13.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 13.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 13.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 14.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 14.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 14.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 14.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.5	No errors	N/A	Error	N/A	Manual check: No errors found
Rule 15.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 15.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 16.1	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.2	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.3	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.4	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.5	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.6	No errors	N/A	No errors	N/A	Checked. Not used
Rule 16.7	No errors	N/A	No errors	N/A	Checked. Not used
Rule 17.1	No errors	N/A	No errors	N/A	Checked. Not used
Rule 17.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 17.3	No errors	N/A	Error	N/A	Error likely caused by missing library
Rule 17.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 17.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 17.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 17.7	No errors	N/A	Multiple	N/A	Errors successfully corrected

			errors		
Rule 17.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.4	No errors	N/A	Multiple errors	N/A	Advisory rule, not followed
Rule 18.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 18.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 19.1	No errors	N/A	No errors	N/A	Checked
Rule 19.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.1	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.3	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.6	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.8	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.9	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.10	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 20.11	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.12	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.13	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 20.14	No errors	N/A	Not checkable	N/A	Manual check: No errors found
Rule 21.1	No errors	N/A	Errors	N/A	Checked. Violation not relevant.
Rule 21.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.6	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.7	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.8	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.9	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule	No errors	N/A	No errors	N/A	Manual check: No errors found

21.10					
Rule 21.11	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 21.12	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 22.1	No errors	N/A	Error in various files	N/A	Manual check: No errors found
Rule 22.2	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 22.3	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 22.4	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 22.5	No errors	N/A	No errors	N/A	Manual check: No errors found
Rule 22.6	No errors	N/A	No errors	N/A	Manual check: No errors found

Mandatory rule
Required rule
Advisory rule

Appendix D: MISRA C Formal deviation documentation format

DEVIATION DECLARATION FORMAT FOR MISRA C:2012

Deviation number, rule number and source file: Deviation #1, Modbus.c

Rule or Directive number: rule 10.3

Type of deviation: Specific

Description of the circumstances causing the deviation from the rule:

In the function "uint8_t cadena_slave" has the final expressions:

datos_a_enviar **=** i**;**

return datos_a_enviar**;**

The variable datos_a_enviar is a uint8_t type while the variable "i" is of type uint_32t.

In this section of the code the variable "i" also is used as a counter and is compared with other variables such as tamano, "i" is also declared as a parameter for the function "uint8_t calcularcrc(uint8_t *buffercrc, uint8_t tamano);" which is of type uint_8t. Changing i type causes further rule 10.3 violations in the function.

Justification to the rule deviation. Include an evaluation of the possible dangers caused by the rule violation compared against other possible solutions:

It's impossible to define I maximum value i can reach. An overflow could cause a wrong value return for the cadena_slave function.

Possible secondary effects due to the rule deviation:

Loss of data counted by i if value goes over 255

Actions taken or needed to guarantee no other problems due to this deviation:

The maximum reachable value for I needs to be documented in other to guarantee no overflow and data losses.

Appendix E: MISRA C:2012 complaint Modbus.c code with C-style format changes

```
#include "Modbus.h"
#include "crc.h"
#include <stdint.h>
#include <stdbool.h>
#include "Uart.h"
#include "TicksTimer.h"
#include "uartTM4C123.h"
#include "gpio.h"
#include "hw_memmap.h"

bool Data_ready   = false;
bool Process_data = false;
bool esperate     = false;

sReadCoilMaster sDataMaster;
sWriteMaster    sWDataMaster;

/*****************************************************************************/

uint16_t get_Registro (void)
{
    return sDataMaster.Registro;
}
/*****************************************************************************/

uint16_t get_Registerqty(void)
{
    return sDataMaster.NoRegistros;
}
/*****************************************************************************/

void set_Registro (uint16_t value)
{
    sDataMaster.Registro = value;
}
/*****************************************************************************/

void set_Registerqty(uint16_t value)
{
    sDataMaster.NoRegistros = value;
}
/*****************************************************************************/

void clear_data(void)
{
    uint8_t cero  = 0;
    sWDataMaster.NoData = cero;
}

bool get_Data(uint8_t *ui8Datos)
```

```
{
    uint8_t cero  = 0;

    if(sWDataMaster.NoData != cero)
    {
        for(uint8_t i = cero; i < sWDataMaster.NoData; i++)
        {
            ui8Datos[i] = sWDataMaster.Data[i];
        }
        return true;
    }
    return false;     /* Second return in a function, fault to rule 15.5 */
}

/**************************************************************************/
* @brief     {Function to build the MODBUS sent string}                   *
*                                                                         *
* @param buffer_inicial information wanted to be send by MODBUS           *
* @param[in] funcion    Value of the MODBUS function which is planned to  *
*                       be used                                           *
*                                                                         *
* @param[in] slave      Address of the responding slave                   *
*                                                                         *
* @param      buffer_final    The uart sent data string (MODBUS)          *
* @param[in]  tamano          The amount of data to send                  *
*                                                                         *
* @return         {Description_of_the_return_value }                     *
/**************************************************************************/

uint8_t Armar_Cadena(uint8_t *buffer_inicial, FModbusType funcion, uint8_t
                            slave, uint8_t *buffer_final, uint32_t tamano)
{
    uint32_t     datos_a_enviar;
    uint32_t     i              = 2;        /* Deviation 1, rule 10.3*/
    uint32_t CRC_MODBUS;
    buffer_final[0]             = slave;
    buffer_final[1]             = funcion;  /* Variable enum Fmodbustype*/
    uint32_t     x              = 2;
    uint8_t ten                 = 10;
    uint8_t once                = 11;

    while ((i-x)<(tamano))
    {
        *(buffer_final+i) = buffer_inicial[i-x];
        i++;
    }
    CRC_MODBUS = calcularcrc(buffer_final, i); /* Deviation 1, rule 10.3 */
    *(buffer_final+i) = MSBobtener(CRC_MODBUS);

    if(*(buffer_final+i) == ten)
    {
        *(buffer_final+i) = once;
    }
    i++;

    *(buffer_final+i) = LSBobtener(CRC_MODBUS);
```

```
    if(*(buffer_final+i) == ten)
    {
        *(buffer_final+i) = once;
    }
    i++;
    *(buffer_final+i) = '\n';
    i++;
    datos_a_enviar = i;
    return datos_a_enviar;
 }
/*****************************************************************************/

void set_data(void)
{
    Data_ready = true;
}
/*****************************************************************************/

uint8_t getaddress(void)
{
    return sDataMaster.address;
}
/*****************************************************************************/

void set_data_modbus (void)
{
    Process_data = true;
}
/*****************************************************************************/

void data_recepcion_Modbus(void)
{
    decodificar_cadena_Slave(&sDataMaster, SLAVEADDRESS);
    decodificar_cadena_Slave_Data(&sWDataMaster,SLAVEADDRESS);
}

/*****************************************************************************/
*                                                                           *
* @param      datos_modbus_string  The data that is wanted to be compared*
*                                  To know which information a device.    *
*                                  Holds                                  *
* @param[in]  tamano2              The amount of data to compare.         *
*                                                                         *
* @param[in]  slave                Address which is wanted to receive the*
*                                  Information.                           *
*                                                                         *
*@return     {Value that indicates if the information is right and the   *
*               device is the correct wanted one.}                        *
/*****************************************************************************/

bool comprobar_Datos_Modbus (uint8_t *datos_modbus_string, char tamano2,
                                                            uint8_t slave)
{
    unsigned char slave_received; /* changed from char to unsigned char */
    unsigned char MSBCRC;
    unsigned char LSBCRC;
```

```
    unsigned char MSB_Calculado;
    unsigned char LSB_Calculado;
    bool Data_recuperada = false;
    uint32_t crcmodbus;
    int32_t tamano20 = tamano2  - '0';
    int32_t tamano23 = tamano20 - 3;
    int32_t tamano22 = tamano20 - 2;
    uint8_t ten = 10;

    slave_received = datos_modbus_string[0];
    if(slave_received == slave)
    {
        MSBCRC = datos_modbus_string[tamano23];
        LSBCRC = datos_modbus_string[tamano22];
        crcmodbus = calcularcrc(datos_modbus_string, tamano20);
        MSB_Calculado = MSBobtener(crcmodbus);

        if(MSB_Calculado == ten)      /* if MSB_Calculado equals 10 */
        {
            MSB_Calculado = 0x0B;     /* MSB_Calculado = 11          */
        }
        LSB_Calculado = LSBobtener(crcmodbus);

        if(LSB_Calculado == ten)      /* if MSB_Calculado equals 10 */
        {
            LSB_Calculado = 0x0B;     /* MSB_Calculado = 11          */
        }

        if((MSBCRC == MSB_Calculado) && (LSBCRC == LSB_Calculado))
        {
             Data_recuperada = true;
        }
        else
        {
            Data_recuperada = false;
        }
    }
    return Data_recuperada;
}
/*************************************************************************
*/

void decodificar_cadena_Slave(sReadCoilMaster *sMasterData, uint8_t slave)
{
    bool     preparar_informacion;
    uint8_t  datos_ModbusRx[100]  = "";
    uint8_t  i             = 0;
    uint16_t registro      = 0;
    uint16_t registerqty   = 0;
    uint32_t pendingBytes2 = uartGetPendingBytes(MODBUS);
    uint32_t cero          = 0;

    if(pendingBytes2 != cero)
    {
        (void) uartGetData(MODBUS, datos_ModbusRx, pendingBytes2);
                                  /* (void) added for rule 17.7 */
```

```
        preparar_informacion = comprobar_Datos_Modbus(datos_ModbusRx,
                                                      pendingBytes2, slave);

        if(preparar_informacion == true)
        {
            sMasterData -> address     = datos_ModbusRx[i++];
            sMasterData -> funcion     = datos_ModbusRx[i++];

            if(sMasterData -> funcion != 0x06)
            {
                registro    |= datos_ModbusRx[i++];
                registro     = registro << 8;
                registro    |= datos_ModbusRx[i++];
                sMasterData -> Registro = registro;
                registerqty |= datos_ModbusRx[i++];
                registerqty  = registerqty << 8;
                registerqty |= datos_ModbusRx[i++];
                sMasterData -> NoRegistros = registerqty;
            }
        }
    }
}
/*****************************************************************************
*/

void decodificar_cadena_Slave_Data(sWriteMaster *sMasterData, uint8_t
                                                                    slave)
{
    bool     preparar_informacion;
    uint8_t  datos_ModbusRx[100]  = "";
    uint8_t  i                    = 0;
    uint16_t registro             = 0;
    uint16_t registerqty          = 0;
    uint32_t pendingBytes2        = uartGetPendingBytes(MODBUS);
    uint8_t cero  = 0;

    if(pendingBytes2 != cero)
    {
        (void)uartGetData(MODBUS, datos_ModbusRx, pendingBytes2);
                                        /* (void) added for rule 17.7 */
        preparar_informacion    = comprobar_Datos_Modbus(datos_ModbusRx,
                                                      pendingBytes2, slave);

        if(preparar_informacion == true)
        {
            sMasterData -> address   = datos_ModbusRx[i++];
            sMasterData -> Function  = datos_ModbusRx[i++];

            if(sMasterData->Function == 0x06)
            {
                registro                |= datos_ModbusRx[i++];
                registro                 = registro << 8;
                registro                |= datos_ModbusRx[i++];
                sMasterData -> Register  = registro;
                registerqty             |= datos_ModbusRx[i++];
                registerqty              = registerqty << 8;
```

```
                registerqty              |= datos_ModbusRx[i++];
                sMasterData -> NoRegister = registerqty;
                sMasterData -> NoData     = datos_ModbusRx[i++];

                for(uint8_t k = 0; k < sMasterData -> NoData; k++)
                {
                    sMasterData->Data[k] = datos_ModbusRx[i++];
                }
            }
        }
    }
}

/*************************************************************************
*/

uint8_t decodificar_cadena_Master(uint8_t *buffer_recuperar_uart, uint8_t
                                                                    slave)
{
    bool     preparar_informacion;
    uint8_t  datos_ModbusRx[100]  = "";
    uint8_t  recovering_bytes     = 0;
    uint8_t  data_to_send         = 0;
    uint32_t pendingBytes2        = uartGetPendingBytes(MODBUS);
    uint8_t cuatro = 4;

    if(pendingBytes2 != 0)
    {
        (void)uartGetData(MODBUS, datos_ModbusRx, pendingBytes2);
                          /* (void) added for rule 17.7 */
        Uart_Transmit(RASPBERRY, datos_ModbusRx, pendingBytes2);
        preparar_informacion = comprobar_Datos_Modbus(datos_ModbusRx,
                                                pendingBytes2, slave);

        if(preparar_informacion == true)
        {
            recovering_bytes = datos_ModbusRx[2];

            for(uint8_t j = cuatro; j < (cuatro + recovering_bytes); j++)
            {
                buffer_recuperar_uart[j-4] = datos_ModbusRx[j-1];
                data_to_send++;
            }
        }
    }
    return recovering_bytes;
}
/*************************************************************************
*/

uint8_t sendCoilMaster(sReadCoilMaster MDatos, uint8_t *Data)
{
    uint8_t  Data_to_send[100] = "";
    uint8_t  Datos_cadena[4];
    uint8_t  i                 = 0;
    uint8_t  f;
```

```
    uint8_t  intentos       = 0;
    uint8_t  received_bytes = 0;
    uint32_t actual_ticks;
    uint32_t initial_tick;
    Datos_cadena[i++] = MDatos.Registro >> 8;
    Datos_cadena[i++] = MDatos.Registro;
    Datos_cadena[i++] = MDatos.NoRegistros >> 8;
    Datos_cadena[i++] = MDatos.NoRegistros;

    f = Armar_Cadena(Datos_cadena, MDatos.funcion, MDatos.address,
Data_to_send, i);
    while(intentos < 2)    /* False positive due to exception*/
    {
        (void)Driver_enable();    /* (void) added for rule 17.7 */
        Uart_Transmit(MODBUS, Data_to_send, f);
        Uart_Transmit(RASPBERRY, Data_to_send, f);
        (void)Receiver_enable();  /* (void) added for rule 17.7 */
        initial_tick = getTicks();

        while((Process_data == false))
        {
            actual_ticks = getTicks();
            if(actual_ticks > (initial_tick + TIMEOUT_ANSWER_MASTER))
            {
                break;
            }
        }

        if(Process_data == true)
        {
            Process_data = false;
            received_bytes = decodificar_cadena_Master(Data,
                                                MDatos.address);

            if(received_bytes!= 0)
            break;
        }
        else
        {
              intentos++;
        }
    }
    return received_bytes;
}
/*****************************************************************************/

bool write_data_master(sWriteMaster SdataWrite)
{
    uint8_t  Data_to_send[100]  = "";
    uint8_t  i                  = 0;
    uint8_t  f                  = 0;
    uint8_t  k;
    uint8_t  Datos_cadena[50]   = "";
    uint8_t  Received_Data[100] = "";
    uint8_t  intentos           = 0;
    uint16_t read_register      = 0;
```

```
uint16_t read_qtyregisters;
uint32_t pending_bytesMod;   /* before uint8_t */
uint32_t initial_tick;
uint32_t actual_ticks;
bool     transferied        = false;

Datos_cadena[i++] = SdataWrite.Register >> 8;
Datos_cadena[i++] = SdataWrite.Register;
Datos_cadena[i++] = SdataWrite.NoRegister >> 8;
Datos_cadena[i++] = SdataWrite.NoRegister;
Datos_cadena[i++] = SdataWrite.NoData;

while(f < SdataWrite.NoData)
{
    Datos_cadena[i++] = SdataWrite.Data[f++];
}
k =Armar_Cadena(Datos_cadena, SdataWrite.Function, SdataWrite.address,
                                                    Data_to_send, i);
while(intentos < 2)
{
    bool correctinfo;
    (void)Driver_enable();        /* (void) added for rule 17.7 */
    Uart_Transmit(MODBUS, Data_to_send, k);
    (void)Receiver_enable();      /* (void) added for rule 17.7 */
    initial_tick = getTicks();

    while((Process_data == false))
    {
        actual_ticks    =  getTicks();
        if(actual_ticks > (initial_tick + TIMEOUT_ANSWER_MASTER))
        {
            break;
        }
    }

    if(Process_data == true)
    {
        pending_bytesMod = uartGetPendingBytes(MODBUS);
        (void)uartGetData(MODBUS, Received_Data, pending_bytesMod);
                                  /* (void) added for rule 17.7 */
        correctinfo = comprobar_Datos_Modbus(Received_Data,
                            pending_bytesMod ,SdataWrite.address);

        if(correctinfo == true)
        {
            read_register |= Received_Data[2];

            if(read_register == SdataWrite.NoData)
            {
                transferied = true;
            }
        }
        break;
    }
    else
    {
```

```
            intentos++;
        }
    }
    return transferied;
}
/*****************************************************************************/

void Answer_Modbus (slaveModbusStruct SDatos)
{
    uint8_t Data_to_send[250] = "";
    uint8_t Datos_cadena[100];
    uint8_t i                 = 0;
    uint8_t f                 = 0;
    uint8_t k;

    Datos_cadena[i++] = SDatos.NoRegistros;

    while(f < (SDatos.NoRegistros))
    {
        Datos_cadena[i++] = SDatos.Data[f++];
    }

    k = Armar_Cadena(Datos_cadena, SDatos.funcion, SDatos.address,
                                                    Data_to_send, i);
    (void)Driver_enable();              /* (void) added by rule 17.7 */
    Uart_Transmit(MODBUS, Data_to_send, k);
    (void)Receiver_enable();            /* (void) added by rule 17.7 */
}
/*****************************************************************************/

void Inicializar_Modbus(void)
{
    inicializarUart1();
    inicializaBufferUart(MODBUS);
    (void)iniciarEnable(); /* (void) added to comply with rule 17.7 */
}
/*****************************************************************************/

void procesar_Modbus(void)
{
    #ifdef SLAVE

    if(Process_data == true)
    {
        Process_data = false;
        data_recepcion_Modbus();
    }
    #endif
}
```