Departamento de Eletrónica Telecomunicações e Informática

Curso: 8309 - Mestrado em Engenharia Eletrónica e Telecomunicações

Unidade curricular: 41505 - Sistemas Embutidos e de Tempo Real

Ano letivo: 2024/2025

Test-Driven Development - Unity

Authors:

104995 Celina Brito

104081 Simão Ribeiro

Class: P2

Date 08/04/2025

Teacher: Paulo Pedreiras

1 Introduction

This document presents a summary of the design and validation of a UART Command Processing system developed in C, integrated into a smart sensing architecture. The module is responsible for decoding and handling structured commands related to sensor data acquisition, history retrieval, and system control. Using a structured testing approach based on the Unity framework, the implementation was evaluated under multiple scenarios to ensure functional correctness and protocol compliance. The goal was to confirm the system's stability and readiness for integration in embedded communication environments.

2 Configuration

To compile and run the program, a C compiler such as GCC is required. Additionally, the following files must be included in the working directory:

- `src folder`: Folder where the source files are located.
  - `main.c`: Entry point of the program where all test cases are executed using the Unity framework.
  - `cmdproc.c` : Source file implementing the UART command parsing and processing logic.
  - `cmdproc.h` : Header file declaring the interface for the command processor and defining related constants.
- `test folder`: Folder where the test files are located.
  - `test.c` : File containing the full set of unit tests for command processing, value validation, and buffer handling.
  - `test.h` : Header file declaring the test functions used in `test.c`.
- `unity folder`: Folder where the required UNITY files are located.
  - `unity.c`, `unity.h`, `unity_internals.h` : UNITY unit testing framework files used for test assertions and reporting.

The program can be compiled using the provided `Makefile` by running the `make` command. Once compiled, the resulting executable will run the full suite of tests automatically.

3 Compilation and Running

To compile the project, a `Makefile` is provided. From the root directory, simply execute the `make` command in the terminal. This will compile all source files and generate the object files inside the `build` folder, and generate the executable named `main` in the project folder. To run the program and execute the full test suite, execute the binary executable using `./main`.

In addiction to this method it is also possible to use `make run` to immediately compile and run the code. This was not working during the presentation but it was fixed.

4 Rationale of Tests

The tests were designed to ensure the correct functionality of the UART Command Processing Module, covering expected usage scenarios, boundary conditions, and error handling.

- **Test 1 – Initialization**: Confirms that buffers and internal state variables are correctly reset during initialization.
- **Test 2 – Command 'P'**: Validates the reception and parsing of individual sensor values (Temperature, Humidity, and CO_2). It checks the correct handling of valid data, the presence of the start and end frame symbols, and the checksum verification.
- **Test 3 – Command 'A'**: Verifies the module's ability to return the most recent values for all sensors in a single response. It also ensures correct formatting, checksum generation, and boundary checking of stored data.
- **Test 4 – Command 'L'**: Ensures the system can return the last 20 stored samples for each sensor type. This test confirms that the FIFO buffer is correctly maintained and the data is returned in the expected format.
- **Test 5 – Command 'R'**: Tests the module's ability to reset the memory and internal buffers, clearing all stored sensor data and resetting indexes.
- **Tests 6 – Wrong Command Error**: Intentionally corrupts messages to verify if wrong commands are correctly detected and flagged by the system.
- **Test 7 – Value Errors**: Sends messages with values outside the defined sensor ranges (e.g., temperature below -50°C or CO_2 below 400 ppm) to ensure that they are rejected with the appropriate error code.
- **Tests 8 – Wrong Checksum Error**: Intentionally corrupts messages to verify if wrong checksums are correctly detected and flagged by the system.
- **Tests 9, 10 – Conversions**: Tests the integer to char and char to integer conversion functions of the module.
- **Tests 11, 12, 13, 14 – Buffer Handling**: Validates the behavior of the RX and TX buffers. Ensures proper copying of buffer contents, correct length detection, and correct handling of edge conditions such as empty or full buffers.
- **Tests 15, 16 – Emulation Functions**: Tests the addValue and getsensor functions of the module, these functions are necessary for the correct emulation of the UART process.

The following figure presents the summary of all unit tests executed using the Unity framework. All tests passed successfully, confirming the reliability of the implemented module.

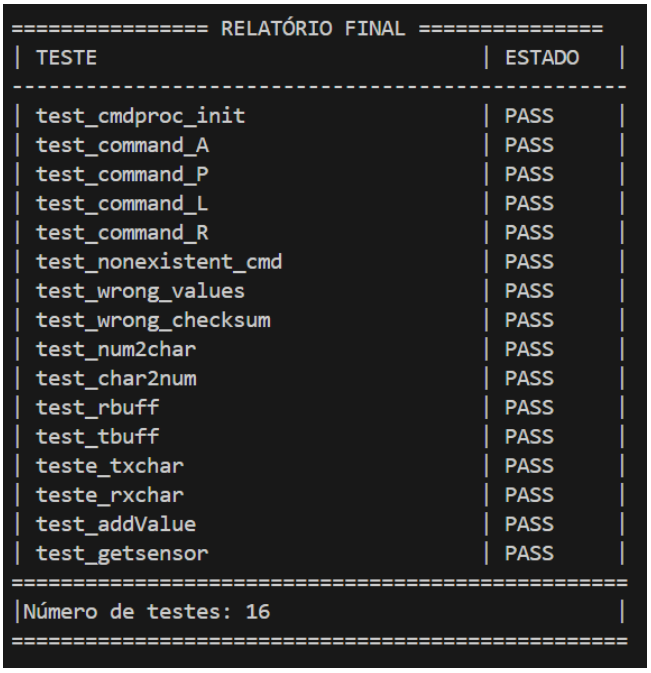

Figure 1: Validation results: All test cases passed successfully

During the presentation, we also had a user command test but after the feedback from the professor, we removed it from our tests because since it needs user input it can't be part of unit testing.

5 Conclusion

In conclusion, the UART command processing module was successfully developed and thoroughly validated using a structured set of unit tests. The implementation proved to be robust and reliable, handling a wide range of operational scenarios including command recognition, buffer management, data validation, and error detection. The test results confirm the correct interpretation and processing of temperature, humidity, and CO_2 values, as well as proper response to invalid data and communication errors. This ensures that the module can be confidently integrated into larger embedded systems requiring serial communication with sensor devices.

Repository Access

The full source code and test files for this project are available in the following GitHub repository:

https://github.com/CRB611/SETR_TRAB2