A REPORT

ON

Firmware verification for Automotive Wireless Battery Monitoring Systems

BY

SAI KARTIK **2020A3PS0435P**

AT

Analog Devices India

A Practice School – II Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, PILANI CAMPUS

December 2023

A REPORT

ON

Firmware verification for Automotive Wireless Battery Monitoring Systems

BY

SAI KARTIK **2020A3PS0435P**

Prepared in partial fulfilment of
the Practice School-II Course BITS F412

AT

Analog Devices India

A Practice School – II Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, PILANI CAMPUS

December 2023

Abstract Sheet
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI - (RAJASTHAN)

Practice School Division

Station: Analog Devices India **Centre:** Bangalore
Duration: 04th July - 22nd December 2023

Title: Firmware verification for Automotive Wireless Battery Monitoring Systems

Name of student: Sai Kartik
ID Number: 2020A3PS0435P

Name of Experts	**Designation**
Mr. Abhinandan Subbaroao	Manager, Software Development Engineering
Mr. Sharath Bala Subramanya	Senior Engineer, Software Development Engineering

Name of PS Faculty	**Designation**
Dr. Satisha Shet K	Asst. Professor, Dept. of EEE

Keywords: Component Testing, Embedded Testing

Project Areas: Firmware & Embedded testing

Abstract

The goal of this project is to thoroughly assess the configuration of a Wireless Battery Management System (wBMS) that is meant to be used in automotive settings.
Of the various wBMS solutions ADI offers, the specific wBMS solution under consideration, which the author has thoroughly evaluated, is designed to comply with criteria that hold high regard for functional safety. This study provides an examination of the techniques used to verify the firmware against said functional safety standards which govern microcontroller operations at the embedded level.

Signature of Student **Signature of PS Faculty**

Acknowledgements

I would like to express my most sincere gratitude to the following people for their help in the work leading to this report:

- Mr. Sharath Bala Subramanya, my mentor, for his invaluable insights, encouragement, and constant support leading to my daily progress.
- Mr. Abhinandan Subbarao, team manager, for his encouragement and support throughout my project.
- I am also very grateful to the rest of SVG, DevOps and the Connectivity team for their constant support and guidance throughout my internship.
- Dr. Satisha Shet K, Instructor In-Charge, for his encouragement and support throughout my project.
- I am also sincerely grateful to the Practice School Division, BITS Pilani, for conducting this Program and allowing me to explore this field of work.

Contents

List of Figures

Abbreviations

API	**A**pplication **P**rogram **I**nterface
STLC	**S**oftware **T**est **L**ife **C**ycle
wBMS	**w**ireless **B**attery **M**onitoring **S**ystem
RTM	**R**esponsibility **T**raceablity **M**atrix

Chapter 1

About the project

1.1 Problem statement

Use automation concepts to test the software present on the embedded devices of a wBMS system and deliver it to customers quickly and efficiently without bugs.

1.2 Motive

There is a growing demand for advanced cockpit electronic systems as a result of the automotive industry's ongoing shift towards electric mobility. These technologies are essential for enabling comprehensive vehicle component monitoring and for providing the occupants with a clear, comprehensive overview of the current state and overall health of the vehicle.
The battery of the car is by far the most important of the crucial factors that need constant monitoring. This is particularly important for battery packs made mostly of Li-ion substrates because of the potential for severe, potentially dangerous consequences if their health and charge states are not strictly monitored.
Given these factors, Analog Devices Inc. provides a wBMS solution that is simple to integrate into the automotive setting with certain OEMs.

1.3 Main objectives

The main objectives of this project are as follows:

1. To identify the test cases to be executed on the software of the wBMS system
2. To write scripts to perform manual testing of all the tests
3. To automate the running of the test suite created and the generation of a test report

Chapter 2

Methodology

2.1 Solution Outline

We have devised a systematic approach involving key steps to optimise our testing process. Initially, a robust deployment method ensures the swift installation of software onto designated testing devices, establishing the foundation for a streamlined testing procedure. Next, an intuitive interface facilitates smooth, bidirectional data exchange within the wireless testing network. This interface is a crucial communication link between the testing environment and the software under evaluation. Following deployment, our process involves meticulously executing a comprehensive test framework. This framework rigorously evaluates collected data, systematically examining the performance and functionality of the software. The subsequent data analysis phase culminates in generating a detailed bug report, a valuable resource for our development team.

2.2 Architechture

2.2.1 Hardware architecture

The basic hardware architecture of the wBMS involves multiple gateways and nodes. The nodes contain the sensors of the system and relay the information to the gateways who in turn, communicate with the host application to ultimately provide the collected information to the end user. To provide the firmware to the embedded devices, we flash the devices with J-Link

debuggers [1]. We then create the network between the various devices in the system with the help of software

FIGURE 2.1: J-Link lite debugger

2.2.2 Software architecture

2.2.2.1 System software architecture

The system software architecture is mainly present to control the network between the gateways and the nodes. This involves a GUI application with specific exposed API endpoints which will enable communication of the network with any test framework built. This will eventually allow us to automate certain tests instead of manually performing them from the GUI.

2.2.2.2 Testing software architecture

To make the test framework easy to access and maintain, python [2] was decided to be used to create the test framework. The initial design of this framework was done with a hybrid approach involving both scripting and Object Oriented Programming concepts. To improve this design, another framework was designed with Pytest [3] solely using Object Oriented concepts.

2.3 Workflow

2.3.1 Testing ideologies

To design test cases that exhaustively cover all possible scenarios, we must adhere to certain aspects of the Software Test Life Cycle (STLC). This mainly consists of 4 steps:

1. Test planning
2. Test case development
3. Test environment setup
4. Test execution

2.3.1.1 Test planning

During the initial step, the software testing team meticulously crafts essential testing strategies, making it the focal point of the process. This critical phase, marked by significance, involves formulating a comprehensive plan to guide the testing process. Typically, the responsibility for establishing project costs and required efforts rests with the team lead or manager [4]. This planning phase commences upon the conclusion of the requirement collection stage, signifying a pivotal transition in the testing lifecycle. The paramount outcome of this preparatory phase is the crystallization of a finalized test plan and strategy. This meticulously designed framework is the guiding document, dictating the subsequent testing procedures and ensuring adherence to the established protocols.

2.3.1.2 Test case development

After outlining a strategy for the tests, the team proceeds to gather the necessary data for implementation. The team then organizes the gathered data to align with various test cases, ensuring comprehensive coverage of all possible scenarios. After completing the design for individual test cases, each case is linked in a chain within the Responsibility Traceability Matrix [5]. This matrix serves as a structured framework, mapping the relationship between test cases and their associated responsibilities.

2.3.1.3 Test environment setup

The development team or the client actively undertakes a standalone task to determine the environment for software evaluation. Simultaneously, the testing team, taking an active role, formulates specific unit test cases to ensure the environment's readiness.

2.3.1.4 Text execution

In this phase, the testing team initiates the execution of test cases based on the decided strategy and environment. If certain tests fail, the team reports the specific defect to the development team using a bug-tracking system. Ideally, every failed test case should be linked to at least one problem, aiding in the identification of issues with the software [6] In the case of a test case being blocked by a design flaw, it is appropriately marked.¡br¿ Subsequently, a comprehensive report, encompassing all failed and blocked test cases, is prepared and provided to the development team for further action.

2.3.2 Test execution with the framework

As part of the test execution, we have chosen to work with pytest because of its simplicity and support. This is a widely used library and hence will also be easy for future managers to oversee any bugs within the code itself.

This library also provides support with common development environments such as VSCode to allow running tests without having to use multiple terminals to achieve the end goal.

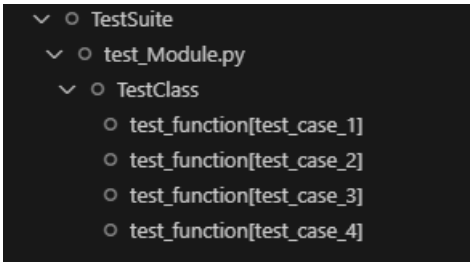


FIGURE 2.2: Untested test cases as collected by pytest on VSCode

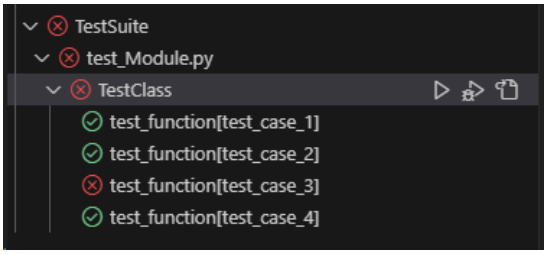
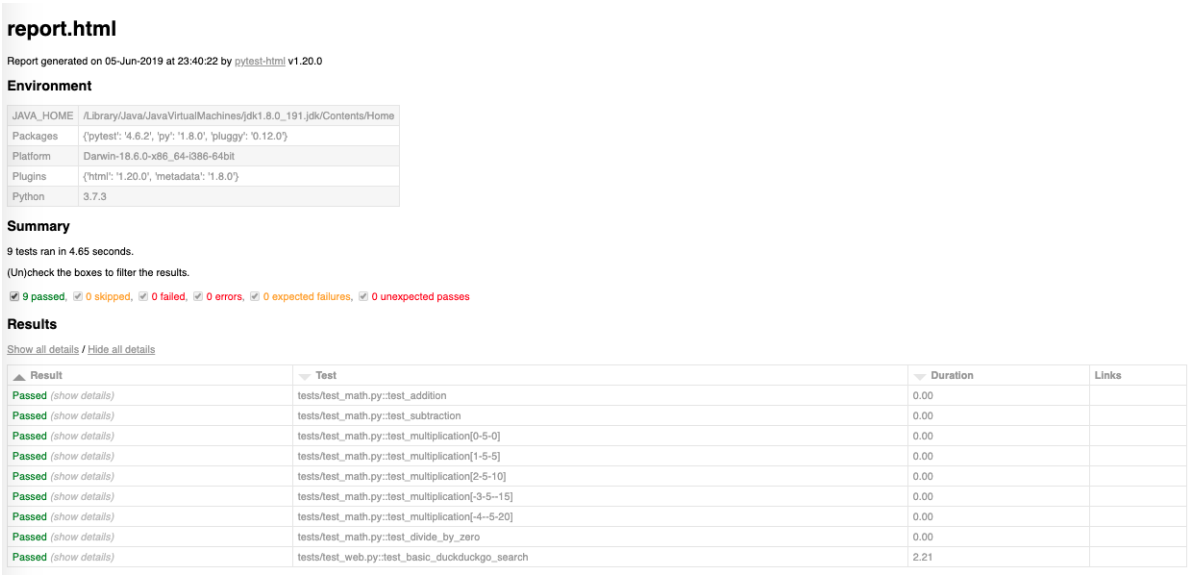


FIGURE 2.3: After testing on VSCode

Pytest also allows us to use various plugins like pytest-html [7] that will allow us to automatically generate HTML and XML reports. These can be used by other teams to create consolidated reports if needed.



report.html

Report generated on 05-Jun-2019 at 23:40:22 by pytest-html v1.20.0

Environment

JAVA_HOME	/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home
Packages	{'pytest': '4.6.2', 'py': '1.8.0', 'pluggy': '0.12.0'}
Platform	Darwin-18.6.0-x86_64-i386-64bit
Plugins	{'html': '1.20.0', 'metadata': '1.8.0'}
Python	3.7.3

Summary

9 tests ran in 4.65 seconds.

(Un)check the boxes to filter the results.

9 passed, 0 skipped, 0 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Passed *(show details)*	tests/test_math.py::test_addition	0.00	
Passed *(show details)*	tests/test_math.py::test_subtraction	0.00	
Passed *(show details)*	tests/test_math.py::test_multiplication[0-5-0]	0.00	
Passed *(show details)*	tests/test_math.py::test_multiplication[1-5-5]	0.00	
Passed *(show details)*	tests/test_math.py::test_multiplication[2-5-10]	0.00	
Passed *(show details)*	tests/test_math.py::test_multiplication[-3-5--15]	0.00	
Passed *(show details)*	tests/test_math.py::test_multiplication[-4--5-20]	0.00	
Passed *(show details)*	tests/test_math.py::test_divide_by_zero	0.00	
Passed *(show details)*	tests/test_web.py::test_basic_duckduckgo_search	2.21	

FIGURE 2.4: Sample pytest HTML report [8]

2.4 Automation of the workflow

The test framework has only solved a part of our problem to automate testing of the firmware. In order to achieve complete automation, we can use CI/CD pipeline software like Jenkins [9] to trigger the test framework to build based on new releases. This will allow us to generate test reports very quickly after each release. This will allow the development team to patch any issues with the firmware and will eventually ensure that the end user of the system will not have to experience buggy software.

Chapter 3

Techonology used

3.1 Hardware

- SDP-K1 [10]
- ADBMS6817 [11, Public reference]

3.2 Software

- Version Control: Bitbucket [12]
- Bug and issue tracking: Jira [13]
- CI/CD pipeline: Jenkins [9]
- Documentation: Confluence [14]

Chapter 4

Conclusion

In summary, the exploration of software testing has revealed valuable insights into the ways to automate it using custom APIs or packages. Through a comprehensive analysis of the architecture of the wBMS, it becomes evident that such extensive testing is necessary to deliver a perfect product. This exploration not only deepens our understanding of firmware testing but also underscores the significance of it in an automotive setting where the slightest of errors can have the largest consequences.

Moving forward, these insights can serve as a foundation for better testing methods to achieve greater coverage in terms of test cases Ultimately, the knowledge gained from this exploration contributes to the broader conversation on the automotive industry, emphasizing the importance of ongoing inquiry and exploration into wireless Battery Monitoring Systems.

Bibliography

[1] *Jlink debuggers*. URL: https:%20//%20www.%20segger%20.com%20/products%20/debug-%20probes%20/j%20-%20link/models/j-link-lite/overview/..

[2] *Python docs*. URL: https://docs.python.org/3.9/.

[3] *Python test scripts*. URL: https://docs.pytest.org/en/7.1.x/contents.html.

[4] Minkyu Lee et al. "Wireless battery management system". In: *2013 World Electric Vehicle Symposium and Exhibition (EVS27)*. 2013, pp. 1–5. DOI: 10.1109/EVS.2013.6914889.

[5] Wei Xie et al. "Vulnerability Detection in IoT Firmware: A Survey". In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. 2017, pp. 769–772. DOI: 10.1109/ICPADS.2017.00104.

[6] Yang Shuaishuai et al. "An automatic testing framework for embedded software". In: *2017 12th International Conference on Computer Science and Education (ICCSE)*. 2017, pp. 269–274. DOI: 10.1109/ICCSE.2017.8085501.

[7] *pytest-html*. URL: https://pypi.org/project/pytest-html/.

[8] *Sample pytest HTML report*. URL: https://blog.testproject.io/2019/07/16/create-pytest-html-test-reports/.

[9] *Automation of testing: (CI/CD Pipeline)*. URL: https://www.jenkins.io/.

[10] *SDP-K1*. URL: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/sdp-k1.html.

[11] *ADBMS6817 public reference*. URL: https://www.analog.com/en/products/adbms6817.html.

[12] *Bitbucket*. URL: https://bitbucket.org/product.

[13] *Jira*. URL: https://www.atlassian.com/software/jira.

[14] *Confluence*. URL: https://www.atlassian.com/software/confluence.