BITS Pilani
Pilani Campus

Firmware verification for Automotive Wireless Battery Monitoring Systems

Author: Sai Kartik (2020A3PS0435P)
Manager: Mr. Abhinandan Subbarao
Professor: Dr. Sathisha Shet K
PS Station: Analog Devices, India

Outline

Aim and Problem statement

innovate achieve lead

Aim

This project aims to verify the firmware of battery monitor sensors for a car's wireless battery monitoring system (wBMS).

Problem statement

Use automation concepts to test the software and deliver it to customers quickly and efficiently without bugs.

Main Objectives

- To identify the testcases to be executed on wBMS system

Main Objectives

- To identify the testcases to be executed on wBMS system
- To write scripts to perform manual testing of all tests

Main Objectives

- To identify the testcases to be executed on wBMS system
- To write scripts to perform manual testing of all tests
- To automate the running of the test suite and generation of test report

Design Methodologies

- To perform manual testing, we flash the firmware onto the respective devices with JLink lite debuggers

Design Methodologies

- To perform manual testing, we flash the firmware onto the respective devices with JLink lite debuggers [1]
- Download files to the gateway and the nodes to facilitate communication between them

Design Methodologies

- To perform manual testing, we flash the firmware onto the respective devices with JLink lite debuggers [1]
- Download files to the gateway and the nodes to facilitate communication between them [2]
- Configure the front-end application (GUI) to control the network and test the functionality.

Design Methodologies

- To perform manual testing, we flash the firmware onto the respective devices with JLink lite debuggers [1]
- Download files to the gateway and the nodes to facilitate communication between them [2]
- Configure the front-end application (GUI) to control the network and test the functionality.
- Ensure the setup is RF-shielded fairly well

High level implementation

The software test cycle (STLC) mainly consists of 4 major steps to go through:

- Test planning

High level implementation

The software test cycle (STLC) mainly consists of 4 major steps to go through:

- Test planning
- Test case development

High level implementation

The software test cycle (STLC) mainly consists of 4 major steps to go through:

- Test planning
- Test case development
- Test environment setup

High level implementation

The software test cycle (STLC) mainly consists of 4 major steps to go through:

- Test planning
- Test case development
- Test environment setup
- Test execution

Test Planning

- This step is the most significant part of software testing where the required testing strategies are created.

Test Planning

- This step is the most significant part of software testing where the required testing strategies are created.
- It is typically the team lead's/manager's role to establish the project cost and efforts required. [4]

Test Planning

- This step is the most significant part of software testing where the required testing strategies are created.
- It is typically the team lead's/manager's role to establish the project cost and efforts required. [4]
- This phase begins once the requirement collection phase has been completed.

Test Planning

- This step is the most significant part of software testing where the required testing strategies are created.
- It is typically the team lead's/manager's role to establish the project cost and efforts required. [4]
- This phase begins once the requirement collection phase has been completed.
- The major outcome of this phase is the finalised test plan/strategy which has to be adhered to.

Test Case Development

- Once a strategy of the tests to be performed is outlined, the required data for it is gathered.

Test Case Development

- Once a strategy of the tests to be performed is outlined, the required data for it is gathered.
- This data is organised to fit various test cases to ensure coverage of all possible scenarios.

Test Case Development

- Once a strategy of the tests to be performed is outlined, the required data for it is gathered.
- This data is organised to fit various test cases to ensure coverage of all possible scenarios.
- Once the design of individual test cases is complete, each test case is linked in a chain in the Responsibility Traceability Matrix. [8]

Test Environment Setup

- This is a standalone task that is undertaken by the development team or the client to determine the environment the software is evaluated in.

Test Environment Setup

- This is a standalone task that is undertaken by the development team or the client to determine the environment the software is evaluated in.
- The testing team should also come up with certain unit test cases to ensure the environment is ready.

Test Execution

- In this phase, the testing team begins executing the test cases based on the strategy and environment decided.

Test Execution

- In this phase, the testing team begins executing the test cases based on the strategy and environment decided.
- If certain tests fail, the particular defect can be reported to the development team using a bug tracking system

Test Execution

- In this phase, the testing team begins executing the test cases based on the strategy and environment decided.
- If certain tests fail, the particular defect can be reported to the development team using a bug tracking system
- Every failed test case should ideally be linked to at least one problem. This linking can help identify issues with the software

Test Execution

- In this phase, the testing team begins executing the test cases based on the strategy and environment decided.
- If certain tests fail, the particular defect can be reported to the development team using a bug tracking system
- Every failed test case should ideally be linked to at least one problem. This linking can help identify issues with the software [7]
- If a test case is blocked by a design flaw, they can be marked as such

Test Execution

- In this phase, the testing team begins executing the test cases based on the strategy and environment decided.
- If certain tests fail, the particular defect can be reported to the development team using a bug tracking system
- Every failed test case should ideally be linked to at least one problem. This linking can help identify issues with the software [7]
- If a test case is blocked by a design flaw, they can be marked as such
- A report consisting of all failed and blocked test cases can then be prepared and provided to the development team for them to take further action.

Low Level Implementation

- In order to exhaustively cover the testing of the software, there will be hundreds of test cases.

Low Level Implementation

- In order to exhaustively cover the testing of the software, there will be hundreds of test cases.
- Maunally testing each case by using GUI elements is completely infeasible as it will consume a lot of time and human resources.

Low Level Implementation

- In order to exhaustively cover the testing of the software, there will be hundreds of test cases.
- Maunally testing each case by using GUI elements is completely infeasible as it will consume a lot of time and human resources.
- In order to overcome this issue, frameworks are built to test the software using various exposed API calls to the software.

Low Level Implementation

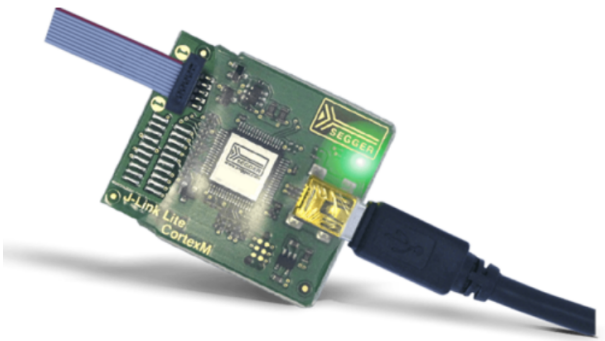
- In order to exhaustively cover the testing of the software, there will be hundreds of test cases.
- Maunally testing each case by using GUI elements is completely infeasible as it will consume a lot of time and human resources.
- In order to overcome this issue, frameworks are built to test the software using various exposed API calls to the software.
- As a major objective of this project, the frameworks are designed to be easily automated.

Low Level Implementation

- In order to exhaustively cover the testing of the software, there will be hundreds of test cases.
- Maunally testing each case by using GUI elements is completely infeasible as it will consume a lot of time and human resources.
- In order to overcome this issue, frameworks are built to test the software using various exposed API calls to the software.
- As a major objective of this project, the frameworks are designed to be easily automated.
- To allow everyone to access the test cases and modify them as necessary, open source software alternatives like pytest and python have been considered. [5, 6]

Hardware Setup

As discussed earlier, the gateways and nodes of the network are flashed with the software provided by the development team.
The development boards housing the neessary components are equipped with the required hardware pins to allow quick flashing through internal applications developed by ADI.



J-link Lite debugger

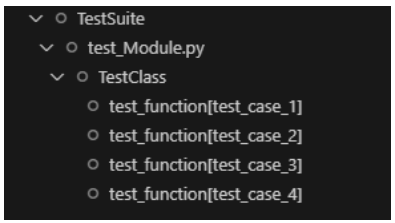
Once the relevant firmware has been flashed onto the respective devices, the nodes are made to join the network created by the gateways

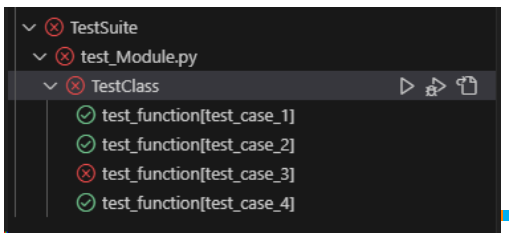
Software setup: Pytest framework

Once the hardware is completely setup and the network is formed, the configuration for the devices in the network are uploaded to them through a GUI tool. This will ensure the reliance of the messages transmitted in the network.
In order to allow automation, this tool can be designed to expose certain API endpoints which can be used by python scripts to communicate and carry tasks at a much faster pace compared to manual interaction with the GUI tool.

Software setup: Pytest framework I

To fast-track the process further in terms of validation and testing, we can use the pytest library along with certain packages that allow communication with the wireless network.
Since this is a very popular framework, there are a lot of tools engineered to make our lives easy. This framework integrates easily with development applications such as VSCode and provides a clean user interface to the tests.





Software setup: Pytest framework II

The user can decide to run specific tests to test various functionalites of the system as and when needed.
If we want to automate a larger batch of tests, we can use the command line options that pytest provides. This tool combined with a CI-CD tool like Jenkins to set triggers for various times to test [3]

Conclusion

We have seen how with the use of certain freely available tools along with standard industry practices, we were able to interface with a complicated wireless battery monitoring system and verify that the software complies with industry standards

References[1]

I

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

[2] URL: https://jusst.org/wp-content/uploads/2021/06/Validation-of-Wireless-Battery-Management-System-wBMS-Gen2.pdf.

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

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

References[2]
II

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

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

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

[1] Other documents regarding specific hardware/software architecture are for internal use only and cannot be shared as open sources/references

[2] Other documents regarding specific hardware/software architecture are for internal use only and cannot be shared as open sources/references