Diploma thesis at Renault:

Development of a tool for automating the analysis of engine control unit error handling

ENSTA Bretagne
2 rue F. Verny
29806 Brest Cedex 9, France

PAURON, Valentin,

valentin.pauron-liverato@ensta-bretagne.org

Renault Group
1 Allée Cornuel
91510 Lardy
Tél : 01 60 82 21 52
11700@renault.com

AVARGUEZ, Frédéric,
frederic.avarguez@renault.com
DENAYROLLES, Manuel
manuel.denayrolles@renault.com

Acknowledgments

I would like to express my deepest gratitude to Frédéric Avarguez who accepted me in the team and has always been there to answer my questions and granted me the chance to visit the Crash test centre. I would also like to extend my sincere thanks to Manuel Denayrolles who offered to me the opportunity to join the Renault technical and test centre in Lardy.
Moreover, I would particularly like to thank Philippe Saillour, Patrick Leveau and Cuong Do-Hoang who guided me all along this internship. I also had the pleasure of working with Arnaud Lebaudy who helped me a lot to achieve this project.
I would also like to extend my sincere thanks to Mr. Vojtisek, my academic supervisor, who accompanied me and helped me to successfully complete this work.

Résumé

L'électronique embarquée automobile s'est massivement développée depuis la commercialisation de l'ABS électronique par Bosch en 1978. Ayant une place centrale dans l'architecture électronique des véhicules, les calculateurs automobiles permettent de contrôler et d'assurer le bon fonctionnement des véhicules.
Mon projet de fin d'études d'ingénieur au Centre Technique et d'Essais Renault Lardy en France, s'inscrit dans la conception et le développement d'un outil Python afin d'automatiser le post-traitement des résultats d'essais de validation inter-systèmes faits au banc HIL (Hardware In the Loop). Les essais inter-systèmes consistent à tester la bonne communication entre les différents calculateurs de la voiture et les bancs Hardware-in-the-loop (HIL) servent à tester le fonctionnement d'un système embarqué pour lequel l'environnement matériel est simulé. Les objectifs finaux de ce projet de fin d'études sont d'améliorer la qualité du développement des nouveaux projets de Renault Group et de réduire leurs coûts et délais de validation.
Tout d'abord, la conception de l'architecture de l'outil Python a permis d'établir le planning d'avancement du projet de manière à optimiser et organiser mon travail. De cette façon, j'ai pu créer les différentes fonctions qui composent l'outil.
Par la suite, la période de validation de l'outil montre que les résultats obtenus par l'analyse manuelle et automatique des essais sont identiques, à l'exception de quelques cas isolés. De plus, l'outil a été mis à disposition de mon équipe à la fin du stage de fin d'études ce qui a permis de confirmer que cet outil de post-traitement automatique des essais sur banc HIL permet de libérer des ressources humaines pour se concentrer sur d'autres tâches ayant une plus grande valeur ajoutée comme la résolution d'erreurs détectées par l'outil.

Abstract

Since Bosch launched electronic ABS in 1978, on-board automobile electronics have significantly improved. Automotive electronic control units (ECUs) are essential to the electronic architecture of automobiles, allowing them to be controlled and kept in proper working order.
In order to automate the post-processing of inter-system validation test results acquired on Hardware-in-the-loop (HIL) test benches, I designed and developed a Python tool for my Diploma Thesis at the Renault Lardy Technical and Test Centre in France. Inter-system tests involve assessing proper communication between the various electronic control units within the vehicle and Hardware-in-the-loop (HIL) benches are testing platforms which employ physical hardware components to mimic real-world interactions, facilitating the evaluation and validation of electronic control units.
The ultimate goals of this diploma thesis are to decrease validation times and costs while increasing the quality of new project development for the Renault Group.
First of all, I was able to plan the project's progress timetable and organize my work by creating the Python tool's architecture. I was able to develop the many functions that make up the tool in this way.
The results acquired by human and automatic test analysis were then identical, except for a few rare occurrences, as revealed by the tool's validation period. Additionally, the tool was made available to my team at the end of the diploma thesis, it confirmed that this automated post-processing tool for HIL bench tests frees up human resources to focus on activities which have a higher added value, including fixing tool-detected mistakes.

Table of contents

Acknowledgments 2

Résumé 3

Abstract 3

Introduction 6

1. Background 7

1.1. Presentation of the “applications-functions” team and inter-system environment 7

1.2. Realising an internship at Renault 9

2. Presentation of the former validation methodology for inter-system environment 9

2.1. Used tool: INCA / MDA module 9

2.2. Former test equipment: Vehicles 9

2.3. Newest test equipment: HIL benches 10

2.4. Inter-system (COMX) tests analysis on MDA module 12

3. Development of a tool for automatic analysis of engine control unit error handling 12

3.1. Definitions 12

3.2. Diploma thesis schedule 18

3.3. Overall tool architecture 19

3.4. Development of a function to read the PVAL HIL (Hardware-In-the-Loop Validation Plan) and extract relevant data 20

3.4.1. Reading the PVAL HIL (Hardware-In-the-Loop Validation Plan) and extracting relevant information on Python 20

3.4.2. Development of *liste_seq* function 22

3.5. Development of the automated Post-Processing function 23

3.5.1. Input files 23

3.5.2. CUT tests 24

3.5.3. CRC and Clock tests 29

3.5.4. Invalid values tests 31

3.5.5. *Post-process* function 35

3.5.6. Discussion, performances of the software 35

3.5.7. Problems and their resolution 36

3.5.7.1. “Ressemblance” function: Two methods to link variables with the same Invalid Value message 37

3.5.7.2. Different samplings between two variables in a frame 37

3.5.7.3. Changing the calculation method of the period of a frame 38

3.5.7.4. Time constraint margins 39
3.6. GUI 42
3.6.1. Development of the tool's GUI 42
3.6.2. GUI for every user 43
3.7. Validation of the tool 45
4. Tool improvements and maintenance 47
4.1. Filtering signals 47
4.2. Global variables for the AEMS methodology 48
4.3. Adaptability of the tool 48
Conclusion 50
Bibliography 51
Table of abbreviations 52
Table of figures 53
Appendix 1: Assessment report 55
Appendix 2: AEMS Methodology 56
Appendix 3: liste_seq code 57

Introduction

The conception of Electronic Control Unit (ECU) software is a really long and complex process, as the engine development engineers have to comply with all of the functional specifications, which means that the constraints coming from other development departments have to be taken into account. The production costs are also a major constraint. Furthermore, the variable customer needs and the green energies trends are highly to be considered. Moreover, the electronic architecture of a car plays more and more a critical role in the overall architecture of the vehicle, especially with electric ones.

At Renault, their development is handled by the powertrain tuning team, and as it plays such a centre role in the vehicle the process starts in the early phases of the car design. However, small changes in the ECU design can drastically impact the drivability, the power unit packaging or the architecture of the car. That is why the base models created in the early phases of development are modified as the car changes to find the perfect compromise between all the aspects of the vehicle.

I realised my diploma thesis in the applications/functions team. This team is responsible for the development, calibration and then vehicle tests of the new Renault Electronic Control Units software.

As a car enthusiast and a mechanical engineering student, realizing an internship at Renault was interesting because it allowed myself to get a better technical knowledge and also discover the automotive engineering world in a new country. Moreover, the courses at Czech Technical University in Prague in the second year of Master of Automotive engineering mainly focused on powertrains which matched perfectly with realizing an internship in the team which designs the ECU of a car.

This work begins with the background of the diploma thesis, with a brief presentation of the Renault team in which I realized it. It then focuses on the development of a Python tool to automate the post-processing of inter-system (COMX) validation test results. Inter-system (COMX) tests are tests to assess proper communication between the various control units within the vehicle. The project schedule and the architecture of the developed Python program are presented. The development of the tool's main functions and its

Figure 1 Renault Lardy Technical and Test Centre

validation are then presented. To complete this work, openings on new functionalities of the tool are proposed and then the maintenance programme of the tool is presented.

1. Background

1.1. Presentation of the "applications-functions" team and inter-system environment

The Renault Lardy Technical and Test Centre has three areas of expertise:
- Acoustics: studies of the vehicle's internal and external noise emissions
- Mechanical innovation: studies of the vehicle's structure, gearbox, and engine
- Electric vehicle: study of batteries and control systems

These studies include endurance, tightness, and performance tests, and are carried out using more than 200 test benches available to characterise vehicles and their equipment. In addition, the Lardy site has three tracks for testing vehicles and their control systems.

My diploma thesis was realised in the applications/functions team, which is in charge of the calibration and then validation of Electronic Control Units (ECUs) software. The goal of their work is to well calibrate the ECUs software to ensure the good communication between the different ECUs of the cars. The environment describing the communications between the different Electronic Control Units (ECUs) is called *inter-system* environment. It involves assessing proper communication between the various electronic control units within the vehicle [1].

This team is involved in the development of thermal, hybrid and electric projects. In their daily work, the team's engineers and technicians adjust and then test the ECUs calibrations on the test track or on Hardware-in-the-loop (HIL) test benches. While testing the ECU, they collect data using the INCA tool to then analyse it. Following this, they analyse these tests using INCA's MDA (Measure Data Analyzer) module to check that the calibrations are correct.

MDA is a measurement data analysis tool which is used to visualize, further process, analyse, and document measurement data [2].

The engineers and technicians carry out tests on various Electronic Control Units, such as Battery Management System (BMS) or Vehicle Dynamic Control (VDC), or even Adaptive Cruise Control (ACC), introducing errors to check the reaction of the various electronic control units (ECUs). Adaptive cruise control (ACC) is a type of advanced driver-assistance system for road vehicles that automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead.

As it is possible to notice on next figure, a lot of data flows between vehicles. The different lines represent the communication links between the different vehicle Electronic Control Units.

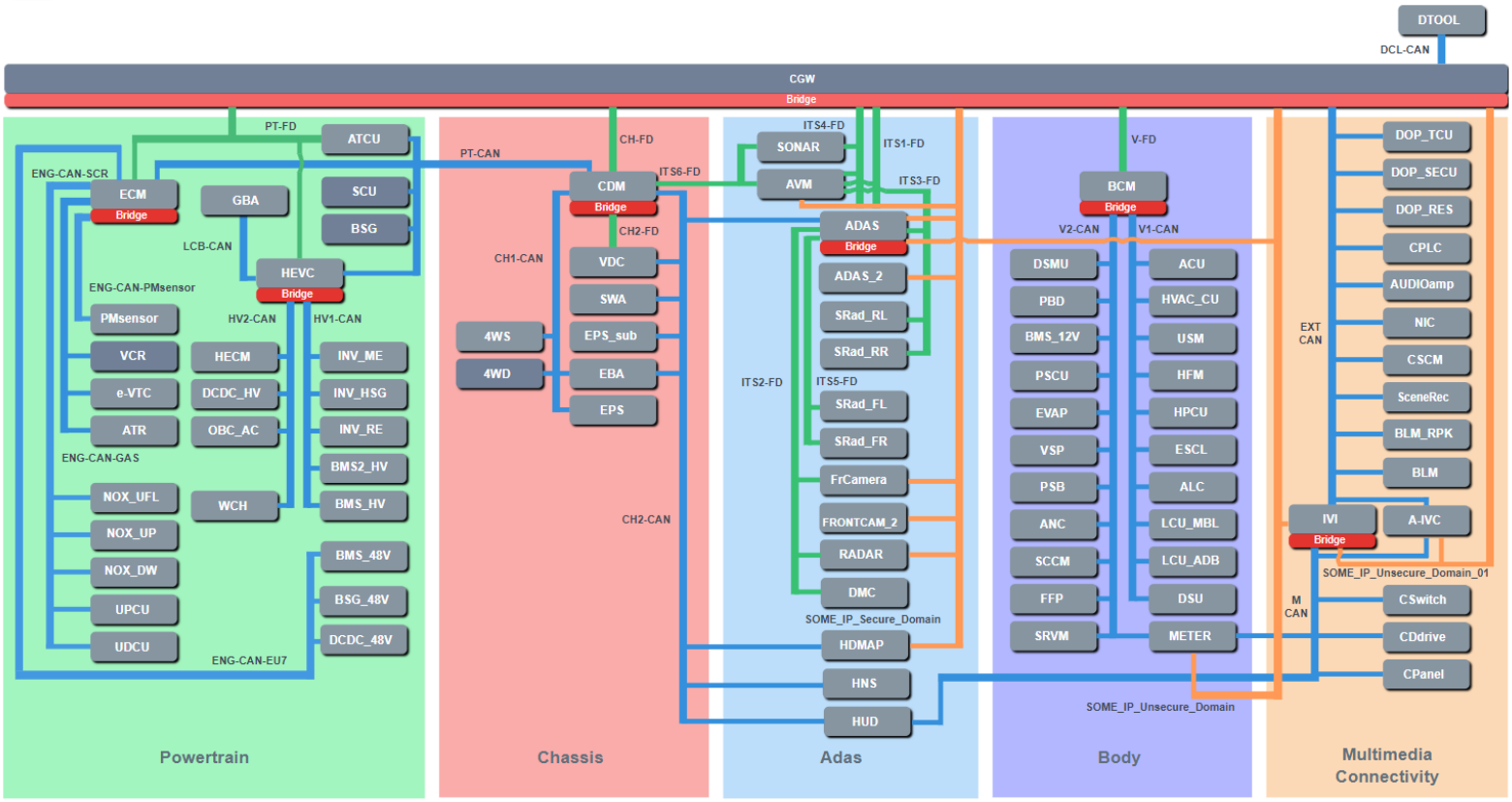


Figure 2 CAN (Controller Area Network) Architecture representing the communication between the different ECUs [5]

The used protocol to link the different car ECUs is the CAN bus (Controller Area Network).
A CAN (Controller Area Network) bus, is a communication protocol used in automotive and industrial applications to enable data transmission and communication between electronic control units (ECUs) and devices within a networked system. In this work, the communication environment between the different ECUs will be called *inter-system*.

An example of an inter-system connection failure is that an engineer from my team calibrated and then tested the new ECU software of the new Renault R5. But as the software of the car is still under development, there was a problem between the brakes and the ABS which caused the brakes to disengage if the ABS was used before. Therefore, to avoid any accident during the tests, all cars which use this software cannot be driven at more than 100 km/h. Moreover, to be able to drive this prototype vehicle, the driver has to get different Renault driving certifications as well piloting trainings, as predevelopment cars have never or almost never been tested on the road so anything could happen even if the parts of the car are oversized to ensure a better safety during the tests.

1.2.Realising an internship at Renault

In my opinion, team spirit is really important in this application/function team and in general in the whole Renault company. All the time when someone needs help, especially the interns, it is possible to ask everyone in the department and it is possible to get some advice or being redirected to new people who would have better answers.
Moreover, as the centre in which I realised my thesis is a test centre, employees must be in the office therefore it is very pleasant for new people who just arrived in the department to meet everyone and feel comfortable instead of contacting them on Teams if they are working from home for instance.

2. Presentation of the former validation methodology for inter-system environment

The validation of the ECUs' software is a long and evolving process. Nowadays, the trends tend to automate as much as possible the validation process to speed up the development of projects by making less errors due to humans during the validation. By automating the validation process, the validation team can spend more time on fixing errors and therefore increase the quality of the final product. The used tools to make the manual post processing will be presented and then the different methods and means of validation will be presented.

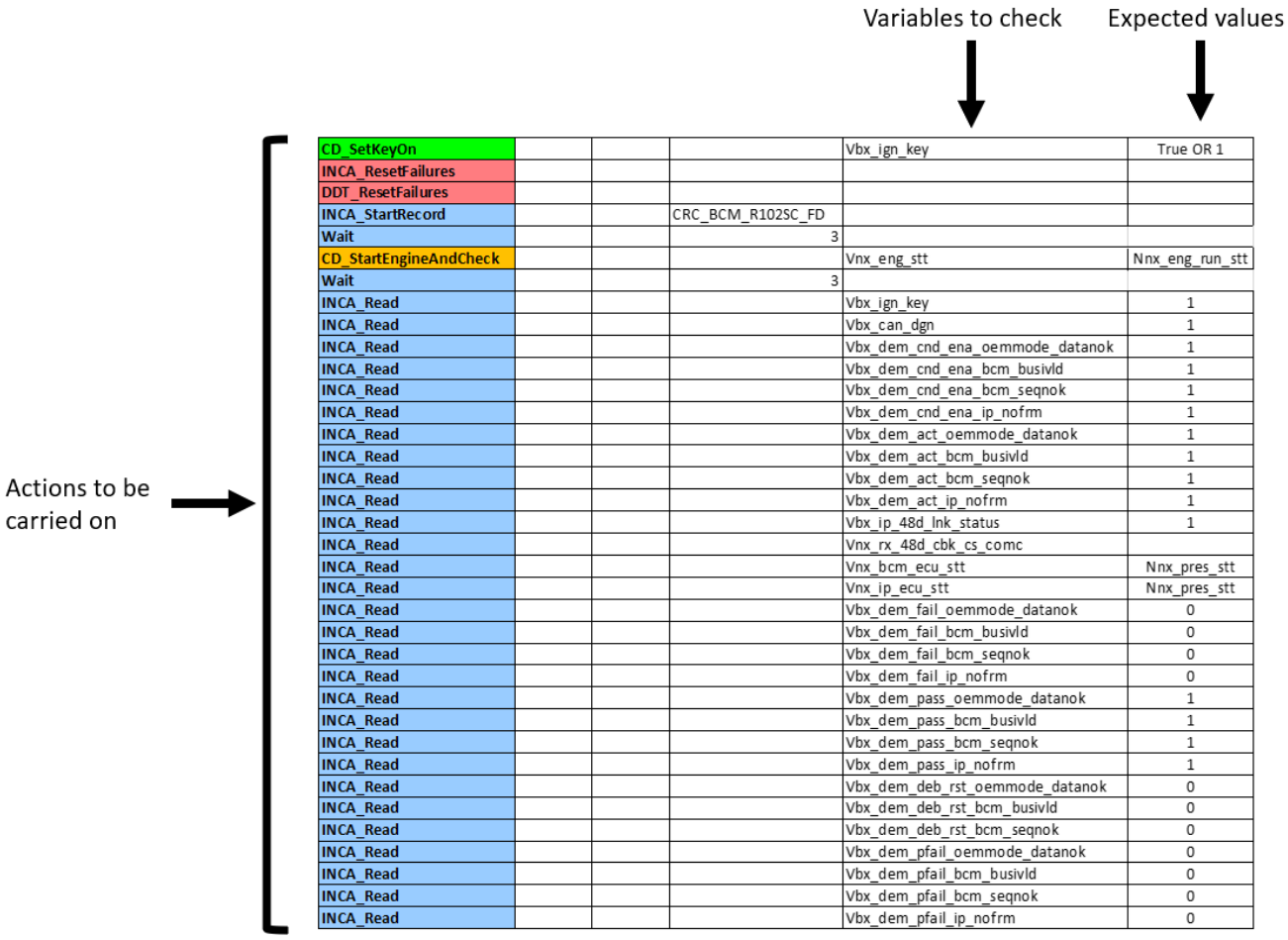
2.1.Used tool: INCA / MDA module

INCA [2] is a software package created in 2009 which enables to calibrate the strategies implemented in the vehicle's ECUs, to validate them on a test bench or using digital simulation, and to diagnose the electronic components fitted to the vehicle. Several functions are provided by the INCA software and its extensions, including:
o Flash re-programming of the ECU
o Measurement data analysis with the MDA (Measure Data Analyzer) extension
o Calibration management and measurement parameter setting
o Measurement acquisition
I used a lot this software during my internship to carry out my tasks both on the test bench and on the track. Interactive and versatile, the work page is an optimized interface for preparing tests and monitoring their progress. This work page allows the user to adjust ECU calibrations in real-time during the test.

2.2.Former test equipment: Vehicles

The standard process of validation is done with vehicles. The engineers and technicians who calibrate the software of the car test it on the track while doing a data acquisition on INCA and then they make the analysis of the acquisition on the INCA tool MDA (Measure Data Analyzer) to validate the ECU's calibrations and software [3].
To realize those vehicle tests, after calibrating the software, they have to follow and check each line of the Validation Plan (PVAL). The Validation Plan (PVAL) is an Excel file which contains the types of tests to be done,

the variables to check as well as the values they should move to during the different phases of the tests. This file was generated and provided by Manuel Denayrolles, who is an inter-system (COMX) expert.



CD_SetKeyOn				Vbx_ign_key	True OR 1
INCA_ResetFailures					
DDT_ResetFailures					
INCA_StartRecord			CRC_BCM_R102SC_FD		
Wait			3		
CD_StartEngineAndCheck				Vnx_eng_stt	Nnx_eng_run_stt
Wait			3		
INCA_Read				Vbx_ign_key	1
INCA_Read				Vbx_can_dgn	1
INCA_Read				Vbx_dem_cnd_ena_oemmode_datanok	1
INCA_Read				Vbx_dem_cnd_ena_bcm_busivld	1
INCA_Read				Vbx_dem_cnd_ena_bcm_seqnok	1
INCA_Read				Vbx_dem_cnd_ena_ip_nofrm	1
INCA_Read				Vbx_dem_act_oemmode_datanok	1
INCA_Read				Vbx_dem_act_bcm_busivld	1
INCA_Read				Vbx_dem_act_bcm_seqnok	1
INCA_Read				Vbx_dem_act_ip_nofrm	1
INCA_Read				Vbx_ip_48d_lnk_status	1
INCA_Read				Vnx_rx_48d_cbk_cs_comc	
INCA_Read				Vnx_bcm_ecu_stt	Nnx_pres_stt
INCA_Read				Vnx_ip_ecu_stt	Nnx_pres_stt
INCA_Read				Vbx_dem_fail_oemmode_datanok	0
INCA_Read				Vbx_dem_fail_bcm_busivld	0
INCA_Read				Vbx_dem_fail_bcm_seqnok	0
INCA_Read				Vbx_dem_fail_ip_nofrm	0
INCA_Read				Vbx_dem_pass_oemmode_datanok	1
INCA_Read				Vbx_dem_pass_bcm_busivld	1
INCA_Read				Vbx_dem_pass_bcm_seqnok	1
INCA_Read				Vbx_dem_pass_ip_nofrm	1
INCA_Read				Vbx_dem_deb_rst_oemmode_datanok	0
INCA_Read				Vbx_dem_deb_rst_bcm_busivld	0
INCA_Read				Vbx_dem_deb_rst_bcm_seqnok	0
INCA_Read				Vbx_dem_pfail_oemmode_datanok	0
INCA_Read				Vbx_dem_pfail_bcm_busivld	0
INCA_Read				Vbx_dem_pfail_bcm_seqnok	0
INCA_Read				Vbx_dem_pfail_ip_nofrm	0

Figure 3 Validation Plan (PVAL) example for one type of test

2.3.Newest test equipment: HIL benches

Nowadays, Hardware-in-the-Loop (HIL) benches are used to speed up and improve the development of new projects. In fact, the physical environment of the car is simulated to recreate the ECU's environment to then test all the types of failures faster [4].
Thanks to the time saved on testing, the engineers and technicians can spend more time on the analysis of the tests and then improve the quality of the development.

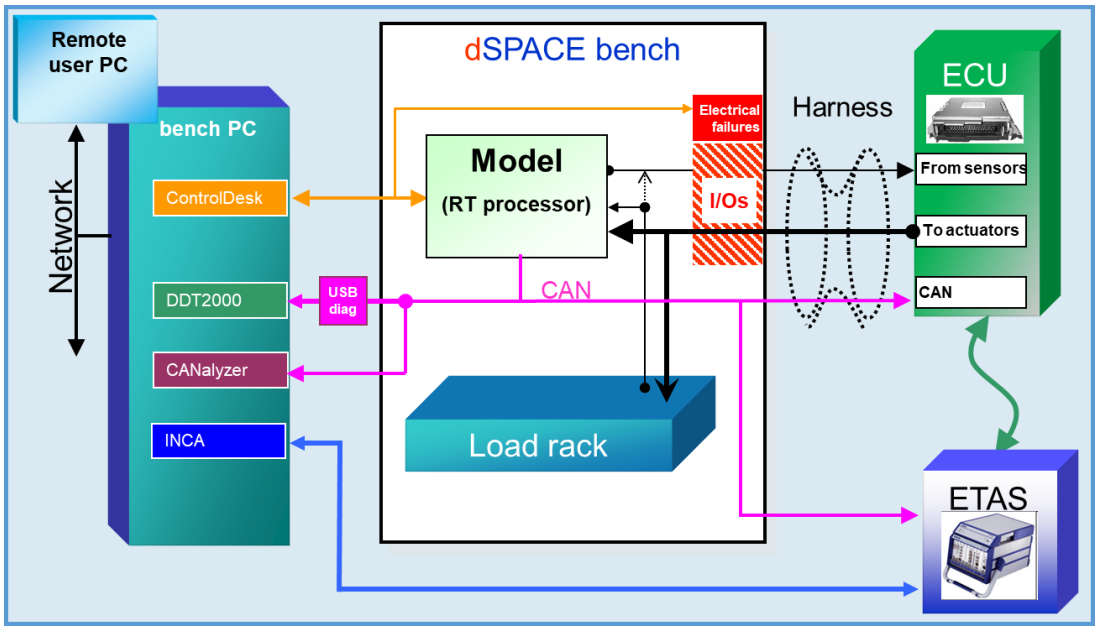


Figure 4 HIL bench architecture [9]

The HIL Environment consists in:

- Mathematical model (the simulation)
- HIL bench (hardware)
- ECU (hardware)
- Vehicle Harness (hardware)
- IS Tools (Control Desk, DDT2000, CANALYZER, INCA)

ECU (Electronic Control Unit): The ECU expects to receive sensors information and sends commands to actuators: it receives the signals from HIL bench (the white rectangle) and sends the commands through the harness (which is also present on the vehicle) connected to the HIL with the I/O board.

I/O board: The signals pass through Input/output boards that ensure the compatibility between the ECU and the Real time Processor (for both sensor and actuator signals). The signals may have electrical failures implemented which allows to validate the electrical diagnostics inside the ECU: open circuit, short circuit to power supply or short circuit to ground. The I/O boards communicate with the HIL processor where the real-time model is being run.

Control Desk: Interaction with the real-time model is done via the Control Desk.

CAN Network: The CAN Network is integrated to the harness. It ensures communication between all ECUs present in the vehicle. On the HIL test bench, the CAN Network is connected to the Real-Time model which simulates the other ECUs of the vehicle.
The ECU, the harness, the load rack, and the model are the only parts specific to a project, all the other elements remain generic. [5]

An example of using an HIL bench in a concrete case to show that we gain time and can reproduce every situation which could be really complex to create on a vehicle: such as simulating the overheating of the vehicle

to see if it is well detected and if the connected components well react to this situation by implementing a downgraded mode for example.

Then, after making the tests on the HIL benches or previously on the cars, the data coming from the tests have to be analyzed on MDA.

2.4.Inter-system (COMX) tests analysis on MDA module

Whether it is a test on vehicle or on HIL benches, data has to be analysed on the MDA (Measure Data Analyzer) tool to check if during all the different types of errors, the detection, confirmation and disconfirmation variables well reacted in accordance with the Alliance Engine Management Software (AEMS) methodology. The disconfirmation of a failure is the confirmation that the error has been cleared and no longer persists. The Alliance Engine Management Software (AEMS) methodology is a is a group of rules shared by Renault and Nissan to ensure that Electronic Control Units (ECUs) behave correctly when they are subjected to a failure. It determines the time constraints that must be respected by the detection, confirmation and disconfirmation variables.
For each test, all variables are selected on MDA and then, the time constraints are checked and then the test is validated or not. If the test is validated, the calibrations of the ECU's software linked to the checked variables are validated. If not, the calibrations linked to the faulty variables have to be checked separately.

3. Development of a tool for automatic analysis of engine control unit error handling

3.1. Definitions

ECU: Electronic Control Unit [6].

Hardware-in-the-Loop benches: HIL bench involves simulating vehicle and environmental inputs for the electronic control unit (ECU) under test, causing it to believe that it is reacting to real-world driving conditions on the open road.

COMX tests: Communication Across all Electronic Control Units CAN (Controller Area Network) tests i.e., exchanges between all the electronic control units (ECUs) inside the vehicle. Those inter-system tests involve assessing proper communication between the various control units within the vehicle. [5]

Specifications of a software:

The specifications of an ECU's software are the architecture and the links between all the variables of the software to respect the customer's requirements design specifications [7]. In our case, this diploma thesis will study the behavior of the detection, confirmation, and disconfirmation of failures in inter-system **(COMX).**

Therefore, in inter-system (**COMX)**, the main variables which will be checked are the detection, confirmation, and disconfirmation variables of a failure to assess whether the time constraints are respected or not to detect and then report potential communication problems between two ECUs.

The architecture of an ECU's software is basically this:

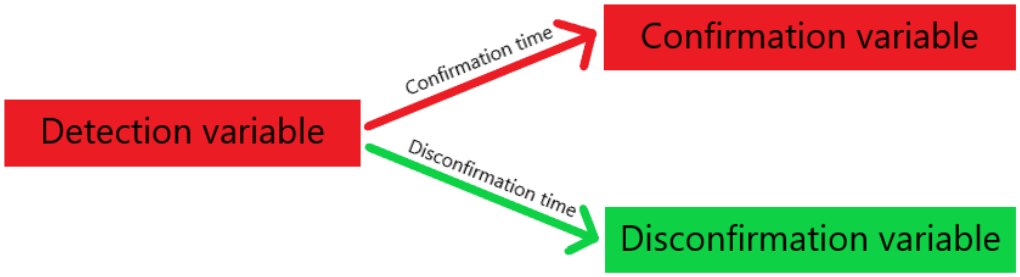


Figure 5 ECU software architecture

The detection variable detects the failure and transmits this information to the confirmation and disconfirmation variables which will respectively switch to 1 or 0 after the confirmation and disconfirmation time.
Those three variables are reported by the ECU.

Detection variable:
It is a Boolean which is released by the ECU as soon as the failure has been implemented by the experimenter.

Confirmation variable:
The confirmation variable is a Boolean released by the ECU after a delay after the detection of the error to confirm that the error is present.

Disconfirmation variable:
The disconfirmation of a failure is in fact the confirmation that the error has been cleared and no longer persists. This variable is released by the ECU after a delay after the detection of the error is over.

INCA/MDA: INCA (Integrated Calibration and Application Tool) is a measurement, calibration and diagnostic software. [2]

HIL Bench (hardware-in-the-loop): HIL tests help validate embedded software on automotive ECUs using simulation and modeling techniques to shorten test times and increase coverage, especially for test cases that are hard to reliably replicate in physical lab, track, or field testing. [4]

Frame:

Each frame of the CAN architecture is composed of a data field which contains the information to transmit but it is also composed of other fields such as the CRC one that we will study later or other ones to know some information about the importance of the frame for example.

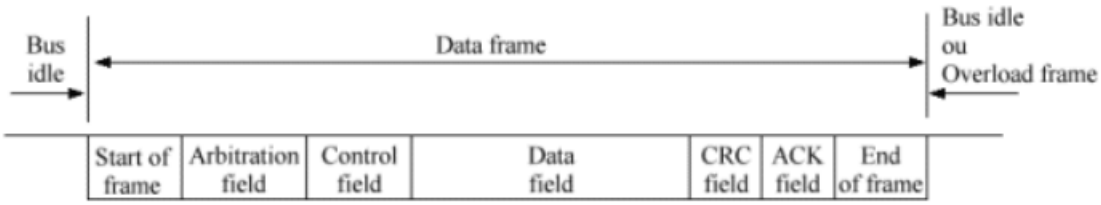


Figure 6 Frame architecture [9]

Frame emission period:

Period of transmission of the frame from a sending to a receiving ECU.

Types of tests:

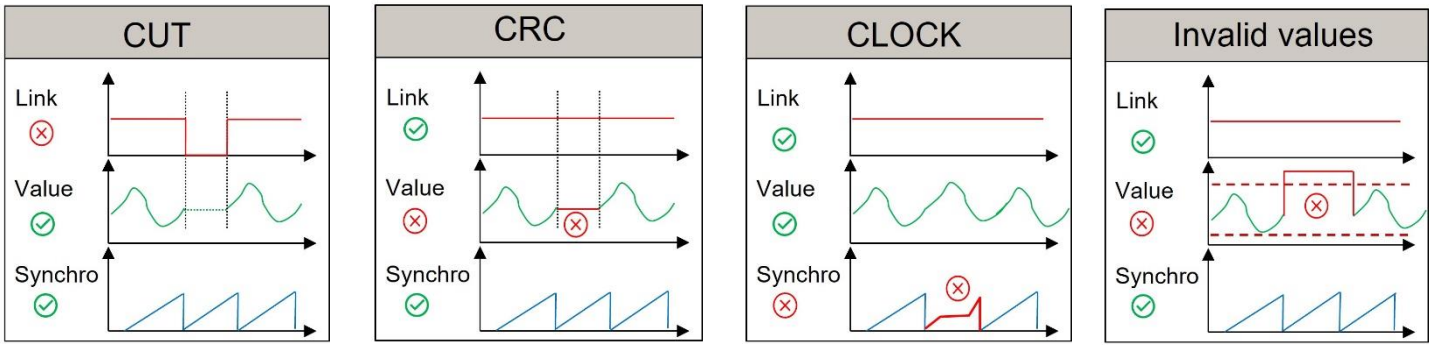


Figure 7 Different types of tests in COMX

CUT: The CUT of a frame is a type of test used to assess the robustness of a communication system in the case of a frame break. In this test, communication between two ECUs is intentionally interrupted to check that the system reacts correctly in this abnormal situation. Later in this thesis, a frame break will be called a CUT.

CRC: (Cyclic Redundancy Check) CRC is an error detection method used to check the integrity of transmitted data.

Clock: The Clock test's goal is to desynchronize the clock of one control unit to another one and then see if the receiver control unit detects the problem of clock desynchronization.

Invalid Value: The goal of this test is to change the value encoded into the data field of a frame by putting a value out of the valid boundaries of the frame to then transmit it from an emitter ECU to a receiver ECU and see if the receiver detects and reacts properly to this invalid value.

AEMS (Alliance Engine Management Software) Methodology:

The Alliance Engine Management Software (**AEMS**) methodology is a is a group of rules shared by Renault and Nissan to ensure that Electronic Control Units (ECUs) behave correctly when they are subjected to a failure. It determines the time constraints that must be respected by the detection, confirmation and disconfirmation variables.

Diagnostic			Level 1
Absence of frame (CUT)		Detection	Detection time
		Confirmation	Confirmation time
		Disconfirmation	Disconfirmation time
Error contained in the frame	Frame break (CUT)	Detection	Detection time
		Confirmation	Confirmation time
		Disconfirmation	Disconfirmation time
	Clock failure	Detection	Detection time
		Confirmation	Confirmation time
		Disconfirmation	Disconfirmation time
	CRC failure	Detection	Detection time
		Confirmation	Confirmation time
		Disconfirmation	Disconfirmation time
	Invalid Value failure	Detection	Detection time
		Confirmation	Confirmation time
		Disconfirmation	Disconfirmation time

Figure 8 Time constraints in the AEMS methodology [Appendix 2]

For inter-system **(COMX)** tests, there are 3 phases in the diagnostic of a failure: detection, confirmation, and disconfirmation.
However, before the diagnostic of the failure can start, there are 4 variables which must switch to 1:
-Ignition key variable: It is a Boolean which switches to 1 when the ignition starts.
-Engine status variable: It is a state variable which has the value of 0 when the engine is off, 1 when the engine is starting and 2 when the engine is running,
-Can diagnosis variable: It is a Boolean which switches to 1 when the diagnostics are authorized on the can network.
-Conditions of enabling the diagnostic variable: It is a Boolean variable which switches to 1 when all the conditions are right for activating the diagnostic.
-Link status variable: It is a Boolean variable which switches to 1 to confirm the presence of the frame received by the transmitting ECU.

Then, the diagnostic must be activated so that the confirmation and disconfirmation variables can change value.

The detection consists in making the system aware of an anomaly. Then the confirmation consists in switching the confirmation variable from 0 to 1 after the detection of the failure for more than a few emission periods

or a few seconds. The emission period Finally, after the detection variable switches back to 0 when the failure is over, after a few periods of frames or a few seconds when the value of the detection variable is at 0, the confirmation variable will switch to 0 as well and the disconfirmation variable will switch back to 1 and therefore the warning light on the dashboard will also switch off.

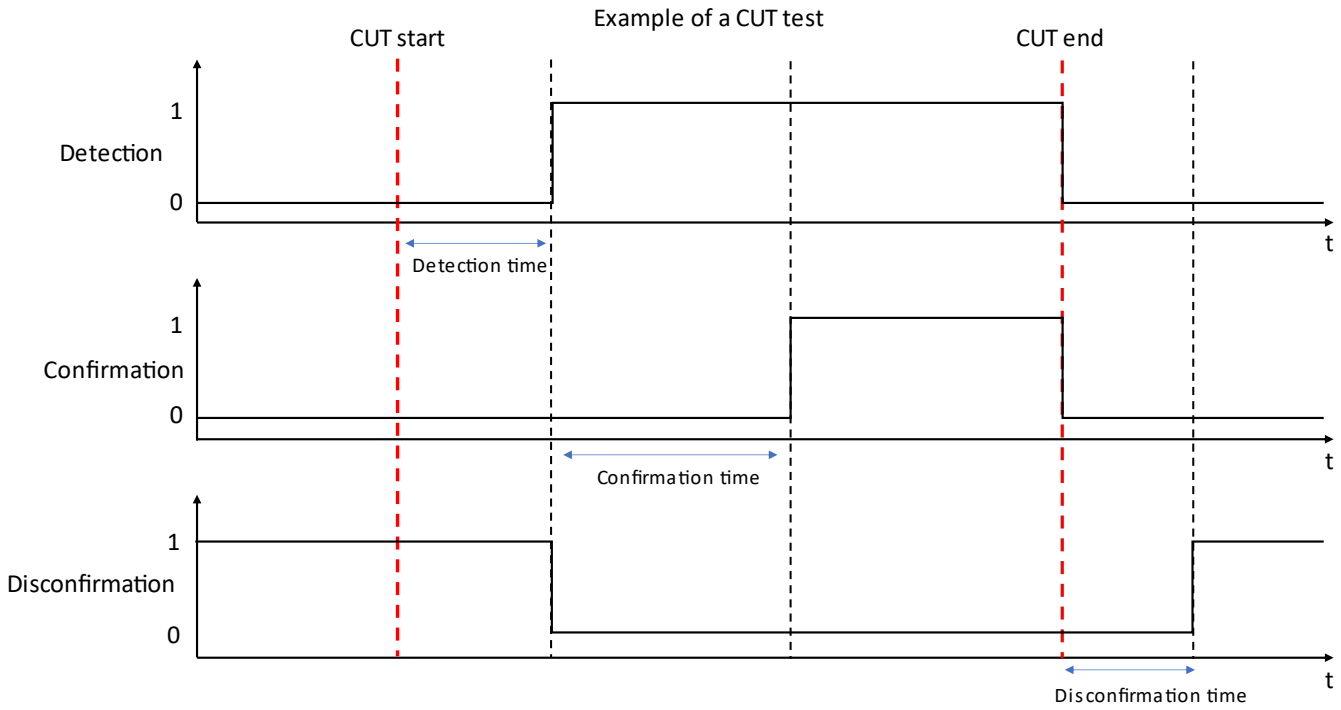


Figure 9 Example of a frame behavior during a CUT test

There are also 2 categories of diagnostics: the nofrm (No frame) variables are relative to the frame itself whereas the datanok (Data NOK) variables are relative to the data contained in the frame.

Nofrm (No frame) variables change value according to the state of the entire frame i.e., whether it is present or not, whereas datanok (Data NOK) variables change according to the contents of the frame i.e. its datafield or crc.

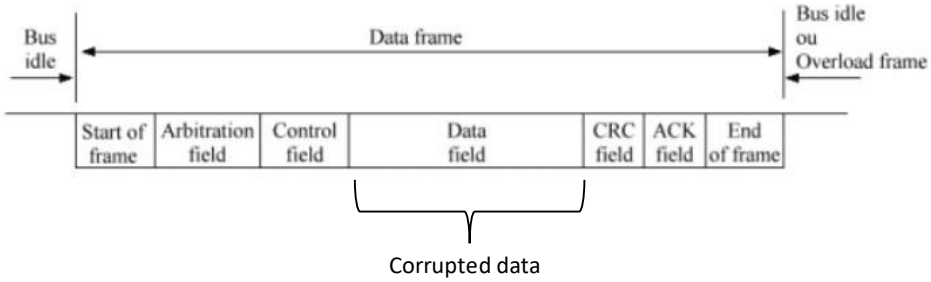


Figure 10 Datanok failure

Datanok variables are raised when the values in the datafield are wrong, i.e., when the frame datafield is corrupted. This is the case when there is an invalid value (because a value is set in the datafield outside the limits), when there is a CRC because the CRC is activated.
There is a “no frame” transmission error when there is a communication break because there is no longer a connection between the receiver and sender, so the receiver necessarily receives a replacement value.

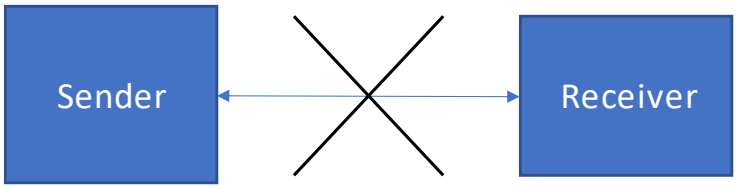


Figure 11 No more connection between sender and receiver

PVAL HIL: PVAL means “Validation Plan”. It is an excel file which contains all the different steps to realize on the HIL (hardware-in-the-loop) test bench during a test. Firstly, there is the normal phase where all the variables are compared to their initial value. Then there is the “error” phase during which the perturbation is imposed and therefore the corresponding variables are being checked to assess if they switch at the right moment according to the Alliance Engine Management Software (AEMS) methodology. Finally, there is the third and last phase which consists in getting back to the initial situation to check if the detection, confirmation and disconfirmation variables change at the right moment. This file was generated and then provided by Manuel Denayrolles, who is an inter-system expert, he was my diploma thesis supervisor.

3.2. Diploma thesis schedule

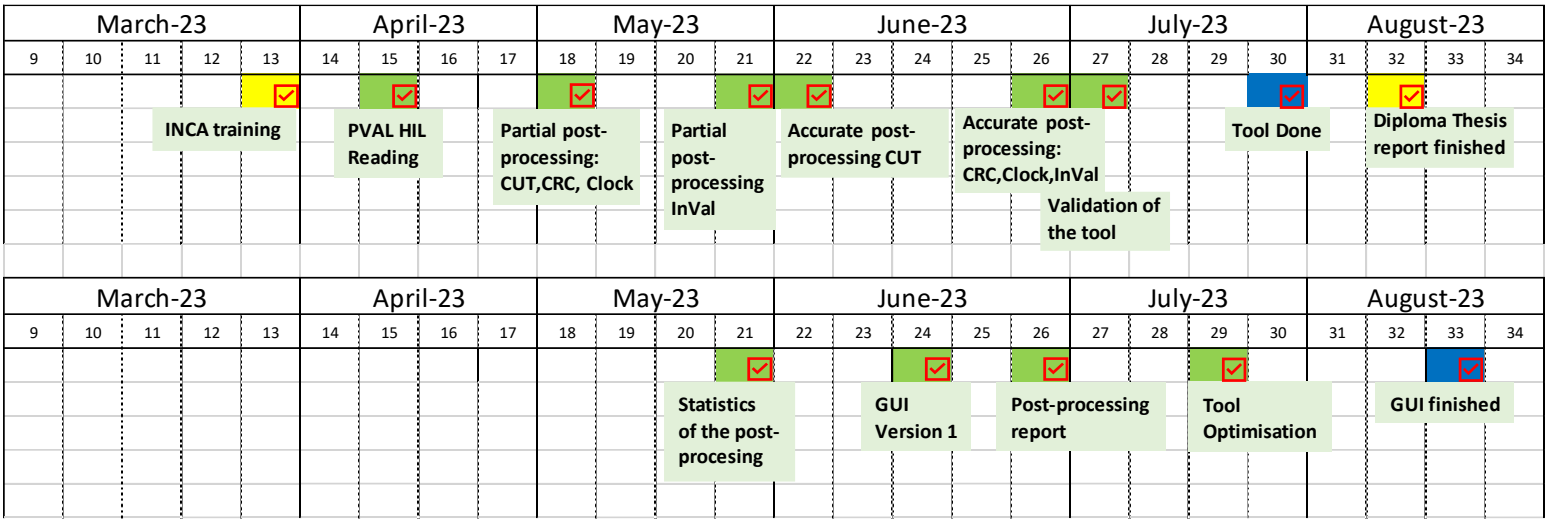


Figure 12 Diploma thesis schedule

: Professional training and university work

: Project milestones

: Final project milestones

The development of an automation tool software for analysis of engine control unit error handling is decomposed in different parts.

On the upper part of the schedule, it is possible to notice the different milestones of the development of the post processing tool. The schedule is based on the architecture of the post-processing tool, which breaks down into several parts: Reading the PVAL HIL (Hardware-In-the-Loop Validation Plan) on Python to extract the important data, then partial post-processing of the 4 types of tests without taking into account the time constraints imposed by the Alliance Engine Management Software (AEMS) methodology and finally the precise post-processing of the 4 types of tests by taking into account the time constraints.
On the lower part it is possible to notice the milestones of the development of the tool which are related to the creation of the Excel report containing the valid frames and also the GUI of the tool so that the software can be used by every user in the future.

3.3. Overall tool architecture

Before presenting the tool's main functions, this illustration summarises the general operation of the tool, including the main inputs, the main functions as well as the results provided by the tool.

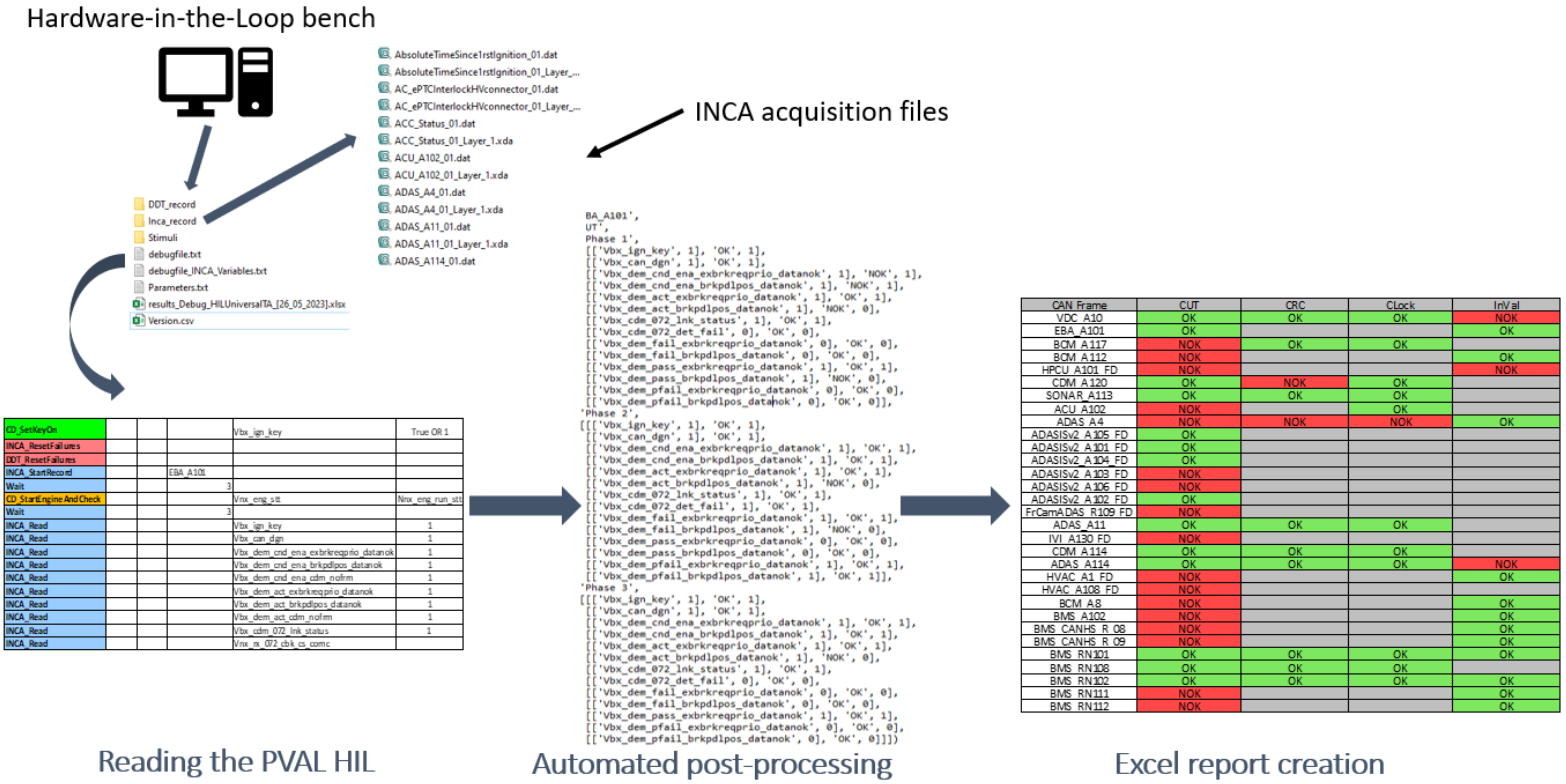


Figure 13 General operating principle of the tool

3.4. Development of a function to read the PVAL HIL (Hardware-In-the-Loop Validation Plan) and extract relevant data

3.4.1. Reading the PVAL HIL (Hardware-In-the-Loop Validation Plan) and extracting relevant information on Python

The PVAL HIL (Hardware-In-the-Loop Validation Plan) of a software is the schedule of all the tests to perform to validate the correct reaction of a diagnostic:

- Detection of errors.
- Confirmation of errors.
- Disconfirmation of errors.

Step	CAN Frame	CAN Message/Variable	Value to be given	Variable to check	Value to be checked
Wait			3		
TA_SetDelayBetweenSteps			0		
CD_SetKeyOn				Frm_Detection_Cnd	True OR 1
INCA_ResetFailures					
DDT_ResetFailures					
INCA_StartRecord		FileName	Frame Name		
Wait			3		
CD_StartEngineAndCheck				Vnx_eng_stt	Nnx_eng_run_stt
Wait			3		
INCA_Read				Frame_Detected	1
INCA_Read				dem_cnd_datanok	1
INCA_Read				Status (Vbx_Var)	0
INCA_Read				rx_asil	0
INCA_Read				frm_stt	Nnx_frm_ok_vld_stt
INCA_Read				vld_stt_mux	Nnx_frm_ok_vld_stt
INCA_Read				Status_mux(det_fail_mux)	0
INCA_Read				dem_fail_datanok	0
INCA_Read				dem_pass_datanok	1
INCA_Read				dem_act_datanok	1
INCA_Read				Rx_cal_bck_comc	
CD_CANUncheckFrame	Frame Name				
INCA_Read				Frame_Detected	1
INCA_Read				dem_cnd_datanok	1
INCA_Read				Status (Vbx_Var)	1
INCA_Read				rx_asil	1
INCA_Read				frm_stt	Nnx_frm_nok_det_stt
INCA_Read				vld_stt_mux	Nnx_frm_nok_det_stt
INCA_Read				frm_stt	Nnx_frm_nok_vld_stt
INCA_Read				vld_stt_mux	Nnx_frm_nok_vld_stt
INCA_Read				Status_mux(det_fail_mux)	**1**
INCA_Read				dem_fail_datanok	1
INCA_Read				dem_pass_datanok	0
INCA_Read				dem_act_datanok	1
INCA_Read				Rx_cal_bck_comc	
CD_CANcheckFrame	Frame Name				
INCA_Read				Rx_cal_bck_comc	
INCA_Read				Frame_Detected	1
INCA_Read				dem_cnd_datanok	1
INCA_Read				Status (Vbx_Var)	**0**
INCA_Read				rx_asil	**0**
INCA_Read				frm_stt	Nnx_frm_ok_vld_stt
INCA_Read				vld_stt_mux	Nnx_frm_ok_vld_stt
INCA_Read				Status_mux(det_fail_mux)	0
INCA_Read				dem_fail_datanok	0
INCA_Read				dem_pass_datanok	1
INCA_Read				dem_act_datanok	1
INCA_StopRecord					
CD_SetKeyOff					
INCA_ResetFailures					
DDT_ResetFailures					

Figure 14 PVAL HIL template

This excel sheet is generated and provided by Manuel Denayrolles, who is an inter-system (COMX) expert.

As it is possible to notice on this PVAL HIL template, the first column represents the list of the actions during the test.
The second one contains the name of the different frames.
The third one contains the name of the invalid message for the Invalid Values tests.
The fourth one contains the value of the invalid message for the Invalid Values tests.
The fifth and the sixth contain the list of the variables to check and the value to be checked during the test.

To analyze the results of the tests, thanks to pandas Python library, I created a function *liste_seq* which goes through all the PVAL HIL and then gets the name of the different frames, the type of test and the variables to check as well as the values to be checked.

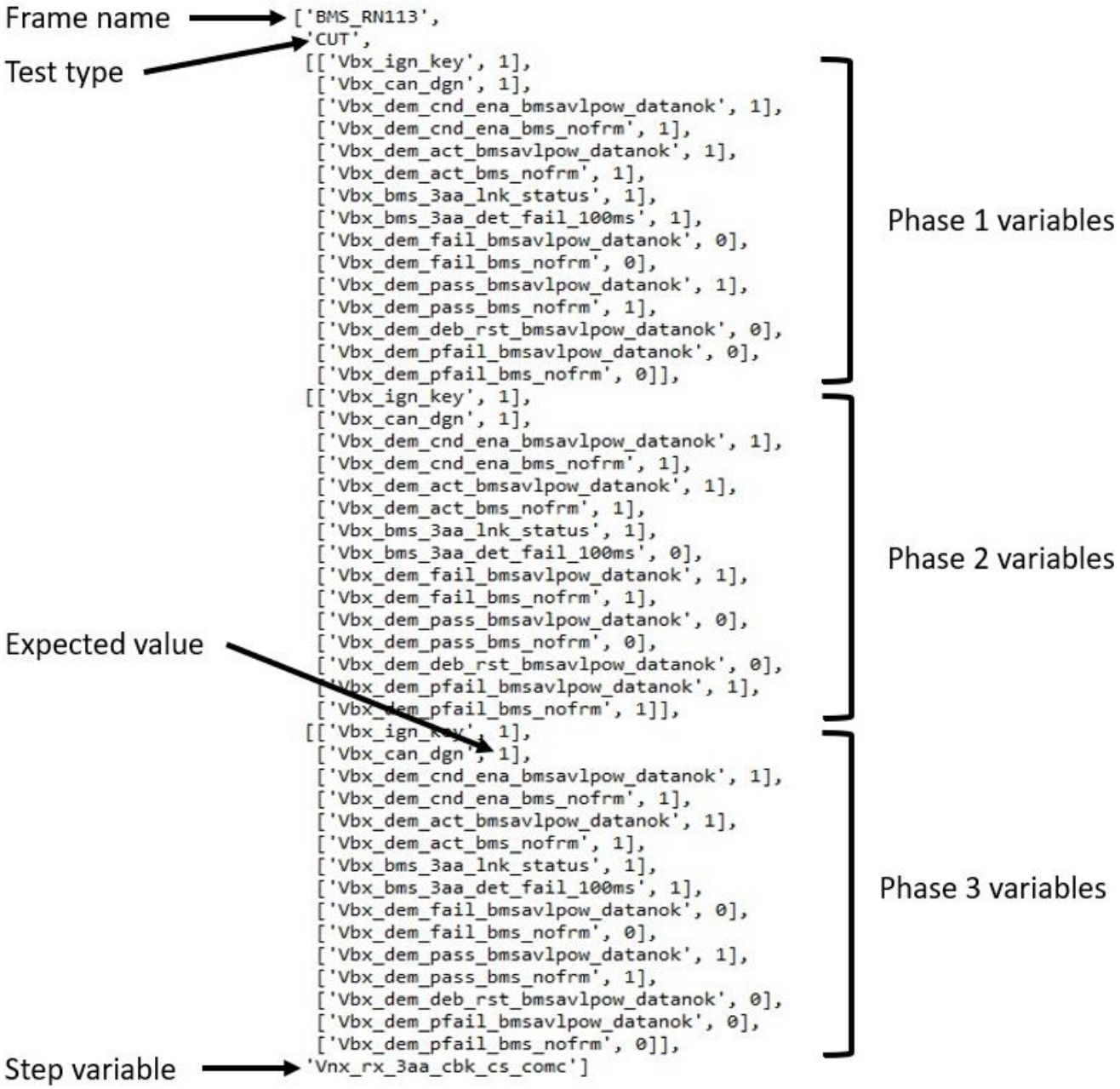


Figure 15 Extracted PVAL HIL of a CUT test in Python

Then this list will be used to make the analysis of the test by getting the variables to check in the .dat file and then comparing them with the value that they should take in the PVAL.

3.4.2. Development of *liste_seq* function

As it has been presented in the previous part, this function is used to extract all the relevant information to then carry out the automated post-processing of the INCA acquisition files.
This function takes the path of the HIL Validation Plan (PVAL) as an argument as well as the transmission rate of the analysed frame.
Then, thanks to Pandas Python library, which enables a user to read an Excel file on Python, the Validation Plan (PVAL) HIL is imported in Python.
Then, the PVAL HIL, an Excel file, is converted to an array thanks to Numpy Python library. This library is used for numerical and mathematical operations. It provides support for working with arrays and matrices. Moreover, the functions provided by this library are very fast which makes this library very useful for data analysis.

```
def liste_seq(ch,frame_rate):

    File=read(ch)
    PVAL_HIL = File.to_numpy()
    Liste_Seq=find_index(ch)
```

Figure 16 list_seq function code beginning

The start and end indexes of each PVAL HIL test are then found using the *find_index* function.
The start and end indexes of a test can be easily recognised in the PVAL HIL Excel file as each start is at the same line as a "*key_on*" cell and each end is at the same line as a "*key_off*" cell.
Thus, this function is based on this principle.

CD_SetKeyOn				Vbx_ign_key	True OR 1
INCA_ResetFailures					
DDT_ResetFailures					
INCA_StartRecord			IVI_A129_FD		
Wait			3		
CD_StartEngineAndCheck				Vnx_eng_stt	Nnx_eng_run_stt
Wait			3		
INCA_Read				Vbx_ign_key	1
INCA_Read				Vnx_can_dgn_ena_mux	1
⋮					
INCA_Read				Vbx_ivi_64e_lnk_status	1
INCA_Read				Vnx_rx_64e_cbk_cs_comc	
INCA_Read				Vnx_bcm_ecu_stt	Nnx_pres_stt
INCA_Read				Vnx_ivi_ecu_stt	Nnx_pres_stt
INCA_Read				Vnx_ivi_64e_nofrm_stt	Nnx_pres_stt
INCA_StopRecord					
CD_SetKeyOff					
INCA_ResetFailures					
DDT_ResetFailures					

Figure 17 Example of an Excel Validation Plan (PVAL) HIL showing start and end of a test

Then, a loop goes through the indexes of the starts and ends of the tests and for each test, the name of the frame is found and then the test type. Then, the 3 parts of the test are found in the Validation Plan (PVAL) using specific cells. And finally, once the structure of the test is established, *liste_seq* function will go through the 3 parts of the test contained in the Excel file PVAL HIL to extract relevant information.
It is possible to look at the code in Appendix 3.

3.5. Development of the automated Post-Processing function

Once all the important data have been extracted from the PVAL HIL, it is now possible to read the acquisition files on Python to then analyze the tests and validate or not the reaction of the different variables of the diagnosis when a frame is cut or when an error is imposed.

The post-processing of the different variables is not based on seconds but on the number of iterations for each sampling of the variables. I.e., a common variable in a test is chosen as the reference, it is called the step variable of every frame for every test. Then, all the other variables which were analyzed had to be sampled at the same rate as the reference because it would be very tricky to compare two variables precisely which are not sampled at the same rate. Therefore, if a variable is not sampled at the same rate as the reference's one, it is skipped and automatically set to NOK but the post-processing of the test will continue anyway.
In the remainder of this report, an **OK** test will refer to a successful one whereas a **NOK** test will refer to a fail one.

3.5.1. Input files

PVAL HIL: Validation plan of a software to schedule of all the tests to perform to validate the correct reaction of a diagnostic. All the important information from this file is extracted by *liste_seq* function.

Database: This Excel file is mainly used to make the link between the names of the messages, which will be voluntarily distorted during the Invalid Values tests, and its associated frames, the platform type as well as the name of the shortened message.

Bridge file: It is an Excel file which makes the link between the frames to check and their associated detection, confirmation, and disconfirmation variables. It also makes the link between the frame and its emission rate.

Data acquisition: I created the function *list_frames_and_files* which gets all the names of the frames in the HIL Validation Plan (PVAL) and link them to their INCA acquisition file in the INCA Record folder. Thanks to this function, it is possible to automate the post processing of the data by automatically analyzing every frame from the PVAL HIL one after another.

```
[[['HVAC_A1_FD', 'CUT'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\HVAC_A1_FD_01.dat'],
 [['HVAC_A1_FD', 'InVal'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\AC_ePTCInterlockHVconnector_01.dat'],
 [['HVAC_A108_FD', 'CUT'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\HVAC_A108_FD_01.dat'],
 [['HVAC_A108_FD', 'InVal'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\BattCoolingDiag_01.dat'],
 [['BCM_A8', 'CUT'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\BCM_A8_01.dat'],
 [['BCM_A8', 'InVal'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\AbsoluteTimeSince1rstIgnition_01.dat'],
 [['BMS_A102', 'CUT'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\\BMS_A102_01.dat'],
 [['BMS_A102', 'InVal'],
  'C:\\Users\\p119720\\OneDrive - Alliance\\Valentin\\Fichiers Python COMX\\HJB\
\Debug_HILUniversalTA[03_06_2023]_part_9\\Inca_record\
\CumulatedEnergyCharged;CumulatedEnergyDisharged;AccumulatedCurrentCharge_01.dat'],
```

Figure 18 List of data path linked to frames names and types of tests

The principle of this function is that thanks to the PVAL HIL, the list of the frames and types of tests is created. Then, thanks to this list, the function reads the file names of the .dat files in the acquisition folder to then compare each frame and type of test to the name of the .dat file.

3.5.2. CUT tests

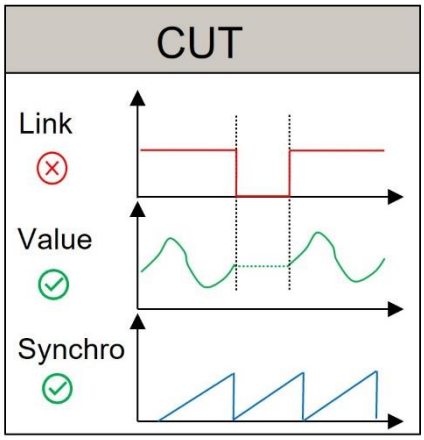


Figure 19 CUT test principle

A CUT test is a frame break test in which the tuner cuts the link between a master ECU and a slave ECU to then see if the master ECU well reacts to this CUT by detecting it, confirming it and then disconfirming it when the ECUs are reconnected.
The imposed time constraints during the test relate to the values of the step variable as well as the values of the detection variable.

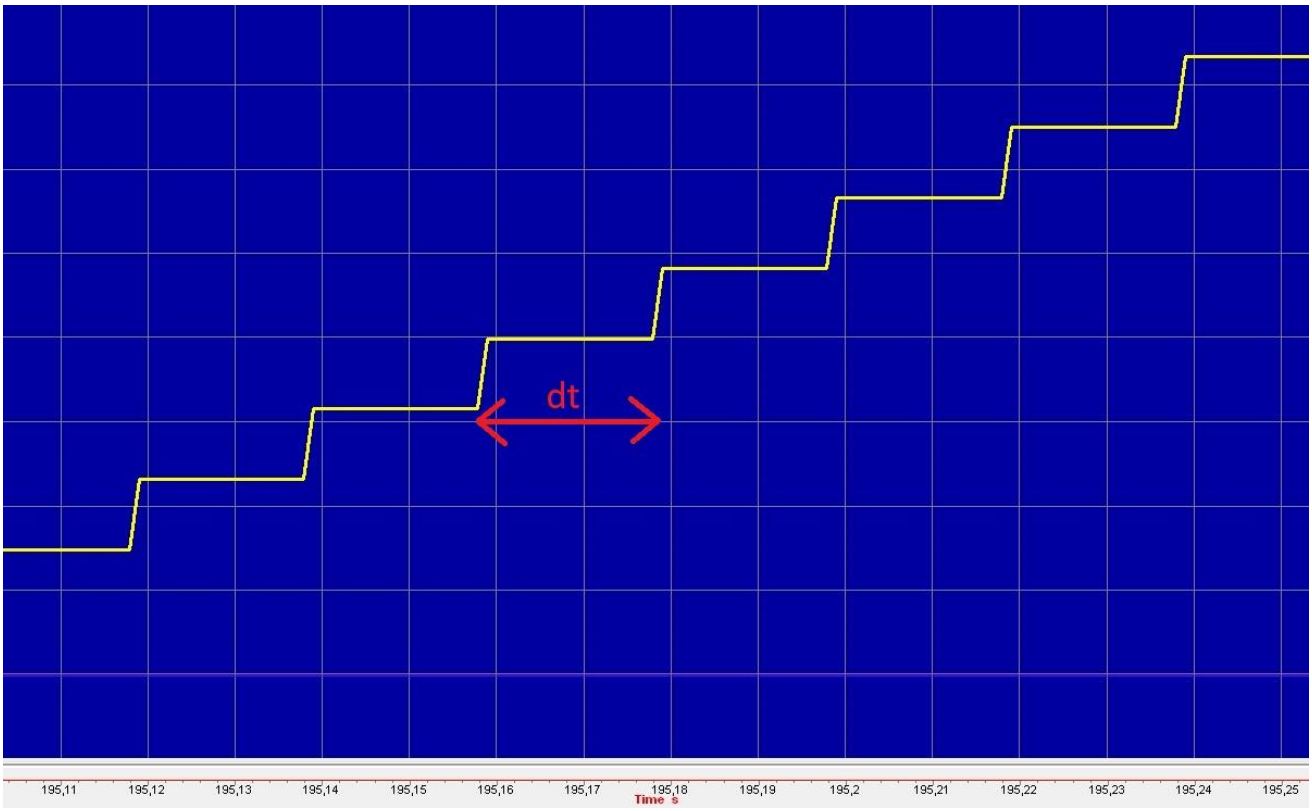


Figure 20 Step variable example

The step variable is a state variable which characterizes the frame activity i.e., every time the frame is sent from the sender to the receiver, a rising edge is raised.

To analyse the CUT tests, I created the function *check_var_CUT_New_Time* which takes as arguments the important data extracted from the PVAL HIL for the analysed test, the acquisition file path as well as the bridge file.
The bridge file is used to get the emission rate "dt" of the frame.
In this function, firstly, the acquisition path is found thanks to the *find_data* function.
Then, the acquisition data is imported in Python thanks to mdfreader library.
Mdfreader is a Python library which is used to import Measured Data Format (MDF) files which are acquisition files coming from the INCA software.
Once the data is imported in Python, the extracted emission rate dt is used to compute the time constraints on the different variables of the frame.
Thanks to the data of the "step variable", the beginning and the end of the CUT are computed thanks to *find_CUT_New* function by determining the iterations when the "step variable" starts to be constant and when it finishes to be constant, i.e., when the CUT is over.

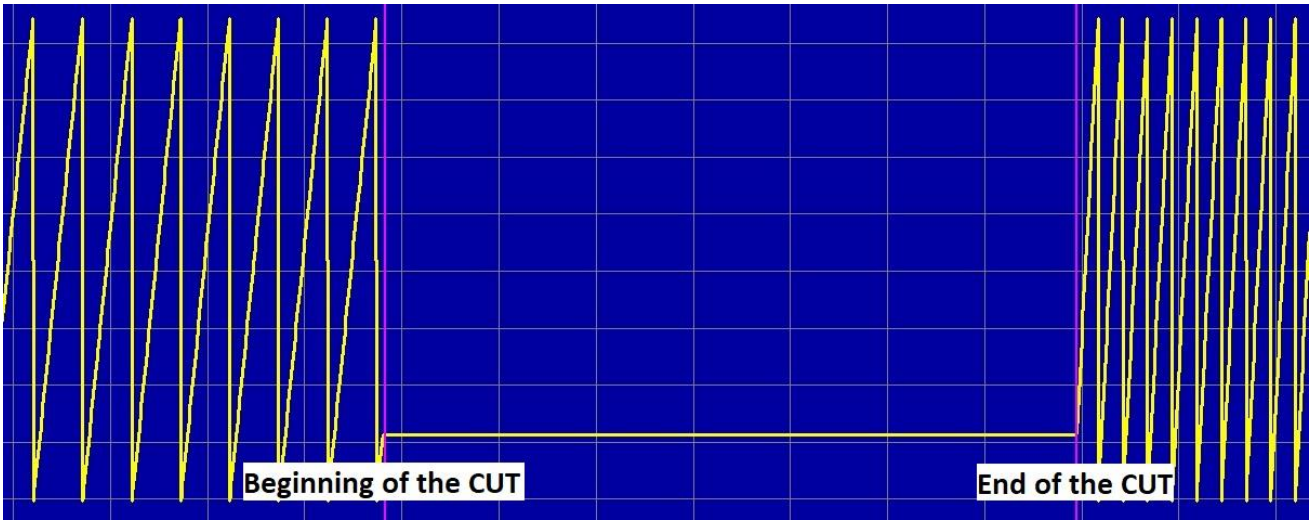


Figure 21 Step variable reaction during a CUT

Once the beginning and the end of CUT have been computed, a test is done to assess if the step variable is constant or not. If it is constant, that means that the frame is not active, therefore the test is automatically set as NOK. If not, the analysis of the test can begin.

The *check_var_CUT_New_Time* function will now go through all the variables to check in the PVAL HIL during the 3 phases of the chosen test.

```
['BMS_RN113',
 'CUT',
  [['Vbx_ign_key', 1],
   ['Vbx_can_dgn', 1],
   ['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1],
   ['Vbx_dem_cnd_ena_bms_nofrm', 1],
   ['Vbx_dem_act_bmsavlpow_datanok', 1],
   ['Vbx_dem_act_bms_nofrm', 1],
   ['Vbx_bms_3aa_lnk_status', 1],
   ['Vbx_bms_3aa_det_fail_100ms', 1],
   ['Vbx_dem_fail_bmsavlpow_datanok', 0],
   ['Vbx_dem_fail_bms_nofrm', 0],
   ['Vbx_dem_pass_bmsavlpow_datanok', 1],
   ['Vbx_dem_pass_bms_nofrm', 1],
   ['Vbx_dem_deb_rst_bmsavlpow_datanok', 0],
   ['Vbx_dem_pfail_bmsavlpow_datanok', 0],
   ['Vbx_dem_pfail_bms_nofrm', 0]],
  [['Vbx_ign_key', 1],
   ['Vbx_can_dgn', 1],
   ['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1],
   ['Vbx_dem_cnd_ena_bms_nofrm', 1],
   ['Vbx_dem_act_bmsavlpow_datanok', 1],
   ['Vbx_dem_act_bms_nofrm', 1],
   ['Vbx_bms_3aa_lnk_status', 1],
   ['Vbx_bms_3aa_det_fail_100ms', 0],
   ['Vbx_dem_fail_bmsavlpow_datanok', 1],
   ['Vbx_dem_fail_bms_nofrm', 1],
   ['Vbx_dem_pass_bmsavlpow_datanok', 0],
   ['Vbx_dem_pass_bms_nofrm', 0],
   ['Vbx_dem_deb_rst_bmsavlpow_datanok', 0],
   ['Vbx_dem_pfail_bmsavlpow_datanok', 1],
   ['Vbx_dem_pfail_bms_nofrm', 1]],
  [['Vbx_ign_key', 1],
   ['Vbx_can_dgn', 1],
   ['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1],
   ['Vbx_dem_cnd_ena_bms_nofrm', 1],
   ['Vbx_dem_act_bmsavlpow_datanok', 1],
   ['Vbx_dem_act_bms_nofrm', 1],
   ['Vbx_bms_3aa_lnk_status', 1],
   ['Vbx_bms_3aa_det_fail_100ms', 1],
   ['Vbx_dem_fail_bmsavlpow_datanok', 0],
   ['Vbx_dem_fail_bms_nofrm', 0],
   ['Vbx_dem_pass_bmsavlpow_datanok', 1],
   ['Vbx_dem_pass_bms_nofrm', 1],
   ['Vbx_dem_deb_rst_bmsavlpow_datanok', 0],
   ['Vbx_dem_pfail_bmsavlpow_datanok', 0],
   ['Vbx_dem_pfail_bms_nofrm', 0]],
  'Vnx_rx_3aa_cbk_cs_comc']
```

Figure 22 Last version of extracted PVAL HIL of a CUT test in Python

During the first phase, which is the normal phase, the variables are compared to their initial expected values. I.e., the variables concerning the ignition key, the conditions to activate the diagnosis, the activation of the diagnosis, the link status between the two ECUs and evidently the detection, confirmation, and disconfirmation variables to ensure that they are set to their initial values. Moreover, during this first phase, when the detection variable is being checked, the beginning and the end of the detection are computed at the same time as the time constraints on the confirmation and disconfirmation variables are based on failure detection.

Then, during the second phase which is the CUT phase, the detection variable is firstly checked to notice if there are some parts during the CUT when it oscillates between 0 and 1 thanks to *test_det_fail* function.

The principle of this function is that when it detects a 0, it checks if detection variable stays at 0 for more than one emission period. If so, this oscillation could disturb the other variables, therefore the entire test is set as NOK. Else, detection is OK and the test analysis can keep on.

Following this, all the other variables can be checked during this phase as their time constraints all depend on the time of the beginning of the CUT or on the time of detection (which should occur a few periods after the beginning of the CUT).

For the third and last phase, the same process as for the second one is applied i.e., the detection, the confirmation and disconfirmation variables are checked to notice if they come back to their initial value,

depending on the time constraints of the end of the CUT and the end of the detection for the confirmation and detection.
Finally, the result of the analysis of the test is returned by *check_var_CUT_New_Time*.

```
('BMS_RN113',
 'CUT',
 ['Phase 1',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_act_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_bms_nofrm', 1], 'OK', 1],
   [['Vbx_bms_3aa_lnk_status', 1], 'OK', 1],
   [['Vbx_bms_3aa_det_fail_100ms', 1], 'NOK', 0],
   [['Vbx_dem_fail_bmsavlpow_datanok', 0], 'OK', 0],
   [['Vbx_dem_fail_bms_nofrm', 0], 'OK', 0],
   [['Vbx_dem_pass_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_pass_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_pfail_bmsavlpow_datanok', 0], 'OK', 0],
   [['Vbx_dem_pfail_bms_nofrm', 0], 'OK', 0]],
  'Phase 2',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_act_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_bms_nofrm', 1], 'OK', 1],
   [['Vbx_bms_3aa_lnk_status', 1], 'OK', 1],
   [['Vbx_bms_3aa_det_fail_100ms', 0], 'NOK', 1],
   [['Vbx_dem_fail_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_fail_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_pass_bmsavlpow_datanok', 0], 'OK', 0],
   [['Vbx_dem_pass_bms_nofrm', 0], 'OK', 0],
   [['Vbx_dem_pfail_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_pfail_bms_nofrm', 1], 'OK', 1]],
  'Phase 3',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_act_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_bms_nofrm', 1], 'OK', 1],
   [['Vbx_bms_3aa_lnk_status', 1], 'OK', 1],
   [['Vbx_bms_3aa_det_fail_100ms', 1], 'NOK', 0],
   [['Vbx_dem_fail_bmsavlpow_datanok', 0], 'OK', 0],
   [['Vbx_dem_fail_bms_nofrm', 0], 'OK', 0],
   [['Vbx_dem_pass_bmsavlpow_datanok', 1], 'OK', 1],
   [['Vbx_dem_pass_bms_nofrm', 1], 'OK', 1],
   [['Vbx_dem_pfail_bmsavlpow_datanok', 0], 'OK', 0],
   [['Vbx_dem_pfail_bms_nofrm', 0], 'OK', 0]]])
```

Figure 23 Analysis result of a CUT test

3.5.3. CRC and Clock tests

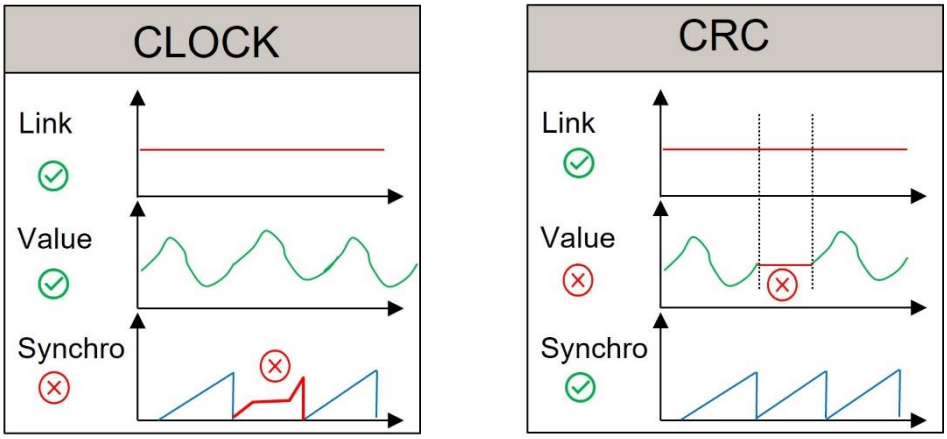


Figure 24 Clock and CRC principles

A Clock test is a desynchronisation test in which the tuner desynchronizes a master ECU and a slave ECU to then see if the master ECU well reacts to this desynchronisation by detecting it, confirming it and then disconfirming it when the ECUs are reconnected.
The time constraints imposed during the test relate to the values of the of the synchronisation detection variable.
The synchronisation detection variable is a Boolean variable which characterizes a problem of synchronization between two ECUs.

The CRC (Cyclic Redundancy Check) test is an error detection method used to check the integrity of transmitted data.
The CRC detection variable is a Boolean variable which characterizes a problem of data integrity between two ECUs.

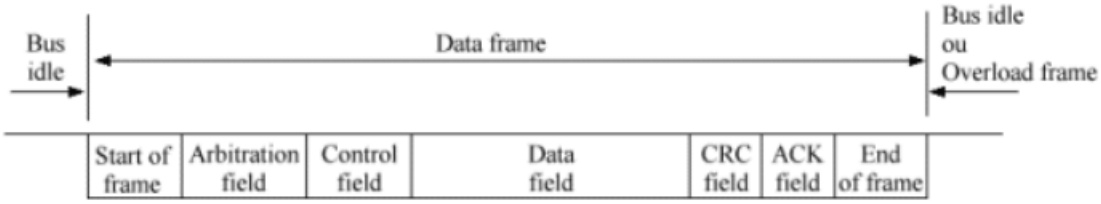


Figure 25 Frame architecture

As it is possible to notice on this scheme representing the architecture of a frame, some bits of each part of the data frame are used to compute the CRC of a frame. When the frame is emitted, the CRC is also computed and sent by the emitter. Then, the receiver computes the CRC with the received frame and compares this value with the CRC calculated by the emitter. Thanks to this method, it is eventually possible to spot an error during the transmission of the frame.

However, this method is not unerring because during the calculation of the CRC, not all the bits are selected to compute this value therefore if an error appears on a bit which is not in the calculation of the CRC, the frame can be transmitted without detecting any malfunction.

The analysis of the CRC and Clock tests are exactly the same. Therefore, I created a common function, *check_var_CRC_Clock*, to analyse both the CRC and Clock tests.
The only difference between those two tests is the name of the detection variable which is "CRC_det_fail" for a CRC test and "Clock_ det_fail" for the Clock tests.
As the variables time constraints depend only on the detection of the failure, I created a test at the beginning of the program so that the function knows if the test is a CRC one or a Clock one.

check_var_CRC_Clock function takes as arguments the important data extracted from the PVAL HIL for the analysed test, the acquisition file path as well as the bridge file.
The bridge file is used to get the emission rate "dt" of the frame.
In this function, firstly, the acquisition path is found thanks to the *find_data* function.
Then, the acquisition data is imported in Python thanks to mdfreader library.
Once the data is imported in Python, the emission rate dt is extracted from the bridge file to use it to compute the time constraints on the different variables of the frame.
Thanks to the "step variable" data, which is the detection variable for those tests, the beginning and the end of the CRC or Clock failure are computed thanks to *find_CRC* function by determining the iterations when the "step variable" switches to 1 and then returns to 0, i.e., when the failure is over.

Once the beginning and the end of the CRC or Clock failure have been computed, a test is done to assess if the frame is active or not thanks to the callback variable (which is the step variable in CUT tests). If not, the test is automatically set as NOK. Else, the test analysis can begin.

Figure 26 Callback variable : represents the frame activity, one step every period

check_var_CRC_Clock function will now go through all the variables to check in the PVAL HIL during the 3 phases of the chosen test.

During the first phase, which is the normal phase, the variables are compared to their initial expected values. I.e., the variables concerning the ignition key, the conditions to activate the diagnosis, the activation of the diagnosis, the link status between the two ECUs and evidently the detection, confirmation, and disconfirmation variables to ensure that they are set to their initial values.
Then, during the second phase which is the CRC or Clock failure phase, the detection variable is firstly checked to notice if it oscillates between 0 and 1 during the failure. Variable oscillations are monitored in the same way as for CUT tests, thanks to *test_det_fail* function. Then, if the detection oscillates, the detection variable is directly set as NOK. Following this, all the other variables can be checked during this phase as their time constraints all depend on detection time. That is why for this test it is very important to make a precise diagnosis on this detection variable as all the analysis computations are based on it.
For the third and last phase, the same process as for the second one is applied i.e., the detection, the confirmation and disconfirmation variables are checked to notice if they come back to their initial value, depending on the time constraints based on the end of detection for confirmation and disconfirmation.
Finally, the result of the analysis of the test is returned by *check_var_CRC_Clock*.

3.5.4. Invalid values tests

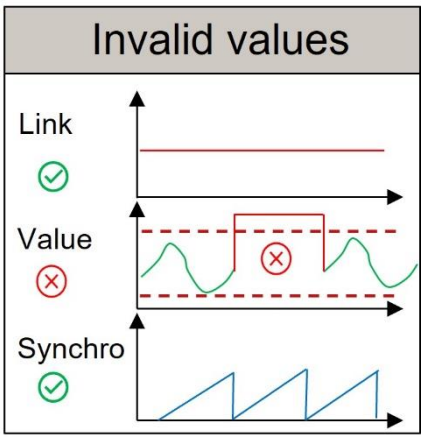


Figure 27 Invalid values test principle

An Invalid value is a replacement value enabling the system to continue operating without crashing, by activating a degraded mode for example, but meaning that the transmitter is no longer able to calculate and send a valid value. Link with the transmitter is still present i.e. the frame is present and active, synchronisation is OK i.e. Clock is OK, data is not corrupted i.e. CRC is OK but the transmitter can no longer calculate a valid value.
For example, if one of the transmitter's sensors is out of order and it needs to transmit this measurement or perform a calculation using this measurement: a replacement value is sent instead, known as the invalid value.

In the remainder of this report, an **invalid value message** will refer to the signal name of the faulty invalid value which is linked to its detection, confirmation and disconfirmation variables.

For example, the frame “ADAS_A2” which is a driving aids frame contains different signals such as “ACC_PWTWheelTorqueRequest” or “ACC_PWTWheelTorqueOrder” which manage the adaptative cruise control.

CD_CANImpose	Frame Name	Faulty Message 1 (Signal 1)	Invalid Value 1
CD_CANImpose		Faulty Message 2 (Signal 2)	Invalid Value 2

Figure 28 Example of an invalid value PVAL containing 2 invalid values

The goal of this test is to change the data field of a frame in the frame structure by putting a value out of the valid bounds of the frame to then transmit it from an emitter ECU to a receiver ECU and see if the receiver detects and reacts properly to this invalid value.

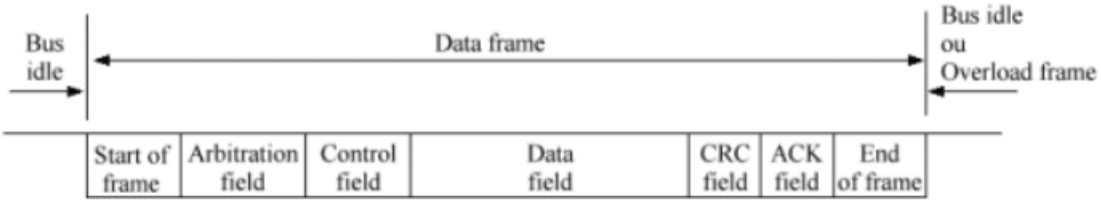


Figure 29 Frame architecture

In fact, there is also three parts during the test. Firstly, all the variables are being recorded during the first phase, which is the normal phase. Then, during the second phase, one or more messages of the frame are voluntarily changed with invalid values and then the detection and confirmation variables are being recorded precisely to notice if the system well reacts to the perturbation.
Then during the last phase, the third one, the changed messages containing invalid values are removed and the situation is finally set back to the initial situation to notice if the detection, confirmation, and disconfirmation variables well react.

To analyse Invalid Values tests, I created the function *check_var_InVal* which takes as arguments the important data extracted from the PVAL HIL for the analysed test, the acquisition file path as well as the bridge file.
The bridge file is used to get the emission rate “dt” of the frame.
In this function, firstly, the acquisition path is found thanks to the *find_data* function.
Then, the acquisition data is imported in Python thanks to mdfreader library.
Once the data is imported in Python, the extracted emission rate dt is used to compute the time constraints on the different variables of the frame.
Then, thanks to *find_Msg_InVal* and the database, a dictionary which links all Invalid Values messages to the variables names is created.
Finally, the initialization phase is over, therefore, the analysis can start.

As an invalid value test can contain several invalid values messages, the post-processing is done message by message as invalid values messages are sent at a different time. Consequently, the beginning and the end of phase 2 during the analysis should differ from one message to another.

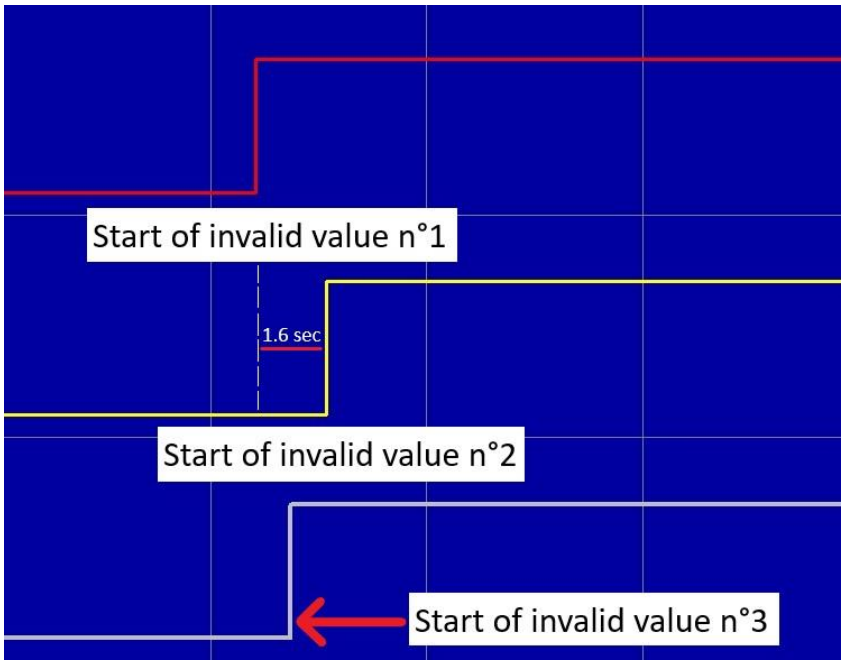


Figure 30 Example of 3 invalid values detection variables with different starts

That is why every invalid value test analysis is carried out faulty message by faulty message.

Then for every faulty message analysis, the activation, activation conditions, detection, confirmation and disconfirmation variables are found in the variables to check list and then loaded in Python thanks to *search*

function and the dictionary which links the shortened message name to the corresponding variables.
Once all the relevant variables have been loaded, the post-processing of the faulty message can begin.
During the first phase, which is the normal phase, the variables are compared to their initial expected values.
Then, during the second phase which is the invalid value failure phase, the invalid value message detection variable is firstly checked to notice if it oscillates between 0 and 1 during the failure. Variable oscillations are monitored in the same way as for CUT tests, thanks to *test_det_fail* function. Then, if the detection oscillates, the detection variable is directly set as NOK. Following this, all the other variables can be checked during this phase as their time constraints all depend on detection time.
For the third and last phase, the same process as for the second one is applied i.e., the detection, the confirmation and disconfirmation variables are checked to notice if they return to their initial value, depending on the time constraints based on the end of detection for confirmation and disconfirmation.
Finally, the analysis of the variables linked to the faulty message is saved in the “Result” list which contains the analysis of the other faulty messages.
Then, the analysis process is applied to all the other faulty messages variables and finally the “Result” list is returned, containing all the faulty messages analysis.

Example of a faulty Invalid Value test: In this test, it will be shown that the time constraints were not respected between the detection of the failure and then the confirmation of the failure: There was 20 sec between the detection and the confirmation whereas it should have been (3+1) dt according to the AEMS methodology i.e. about 4 sec to confirm the failure.

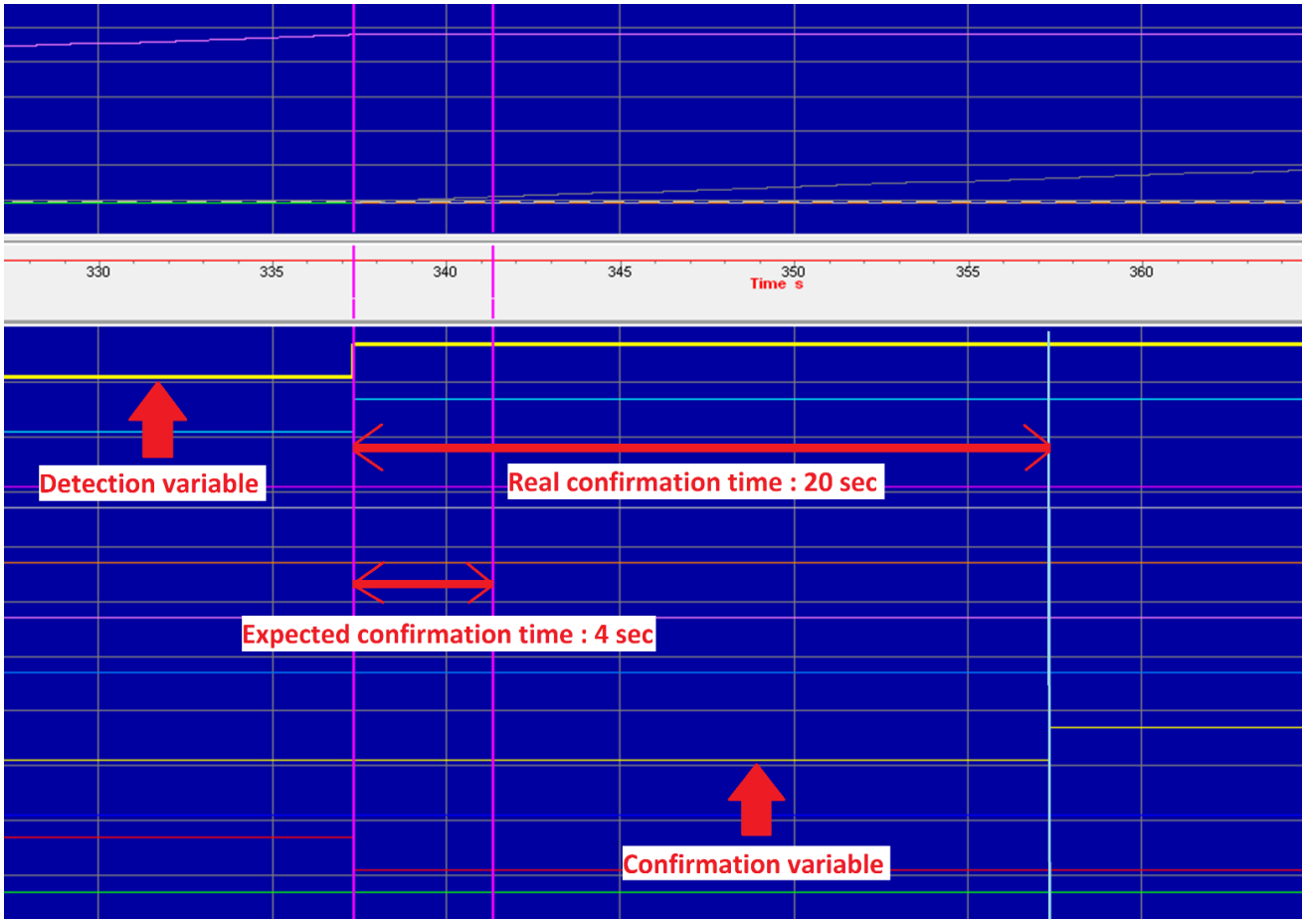


Figure 31 Faulty Invalid Value test

Therefore, following this situation, the tool notices the wrong confirmation time and then set the confirmation variable as NOK which will result as a NOK test in the Excel report, with the confirmation variable to check.

Variables to check	Phase 1	Phase 2	Phase 3
Vbx_dem_fail_abs1ignition_datanok	0	0	0
Vbx_dem_pass_abs1ignition_datanok	1	0	0
Result	NOK		
nofrm	-	-	
Status of observed datanok variables	NOK	6	

Figure 32 Faulty invalid value test Excel report

As it is possible to notice on this Excel report, this diagnosis is well established as it is possible to see that the "dem_fail" variable (confirmation variable) is still at 0 at the time when it is supposed to switch to 1 because of the detection of the failure. Finally, the test is set to NOK at the right error code 6 which means that the confirmation time is not respected.

3.5.5. *Post-process* function

Finally, after developing and testing the three post-processing functions adapted to the 4 types of tests, i.e. *check_var_CUT_New_Time*, *check_var_CRC_Clock* or *check_var_InVal*, I created the *Post-process* function which takes as main argument the extracted Validation Plan (PVAL) HIL to then go through it and read the name and test type of each frame to then apply the right post-processing function I just mentioned.

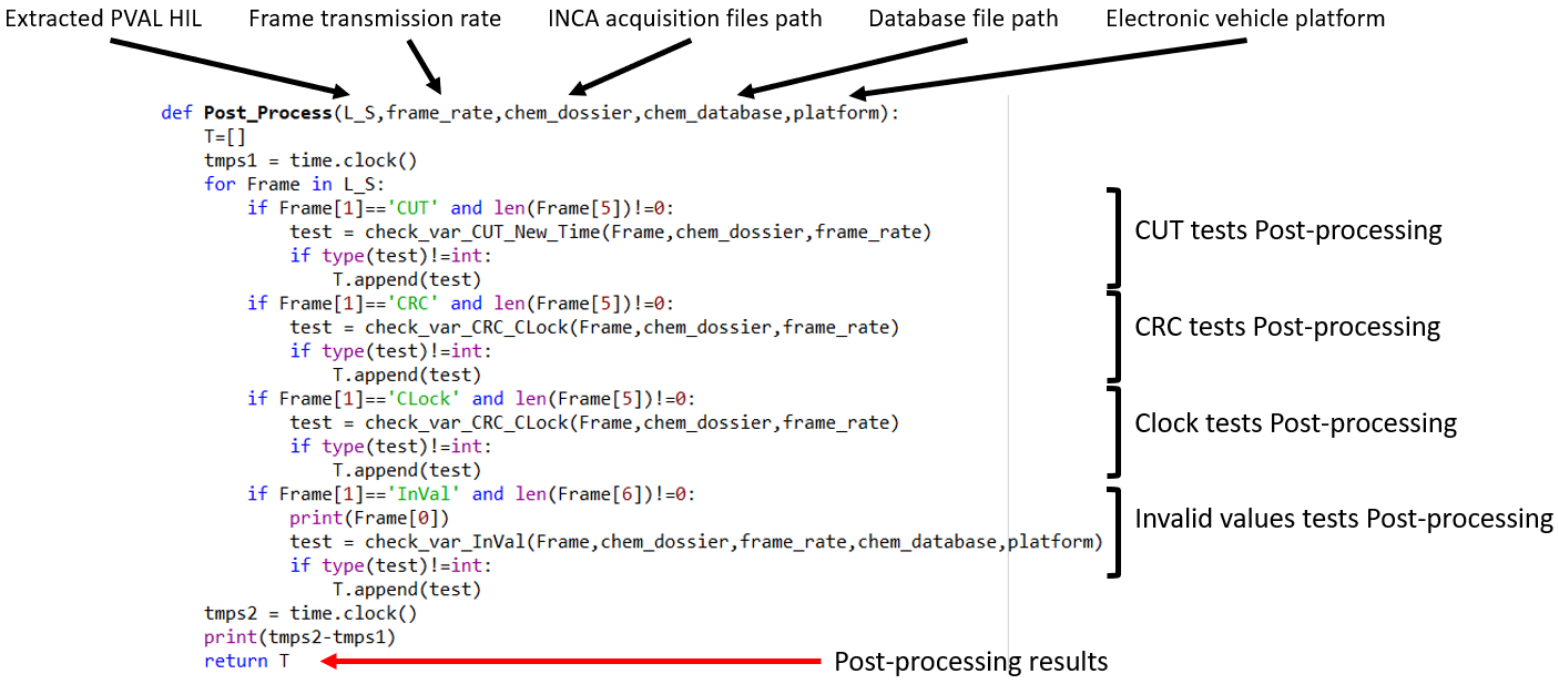


Figure 33 Post-processing function code in Python

Post-process function is the main function which is called in the GUI to make the post-processing of all the tests.

3.5.6. Discussion, performances of the software

It was highly practical to create the *Post-process* function which calls upon four other functions rather than a single, large function. This made the development and testing of these four functions much simpler, unlike a function which would handle all four types of tests and be more complex to debug. Furthermore, using multiple small functions rather than a single main one offers several advantages for tool maintenance and updates, as it facilitates modifications and improvements to the functions and simplifies problem-solving.
Additionally, breaking down the program's structure into various small functions is well-suited for maintenance and future upgrades by other engineers or technicians from my team, as it will enable them to better

understand the tool's operation compared to dealing with a single, monolithic function, which can be challenging to grasp.

The advantage of creating Post-process function is that it is possible to test all types of tests or select only one if improvements have been made to a specific post-processing function, for example.

CAN Frame	CUT	CRC	CLock	InVal
VDC_A121	NOK	-	-	-
ADAS_A104	NOK	-	-	-
HFM_A104	OK	-	-	-
USM_A102_FD	OK	-	-	-
IVI_A132_FD	NOK	-	-	-
IVI_R103_FD	NOK	-	-	-
ParkOut_SCU_Frame	NOK	-	-	-
EPKB_A1	NOK	-	-	-
FRCAMERA_A110	OK	-	-	-
BCM_R102SC_FD	NOK	-	-	-
CDM_A107	OK	-	-	-
CDM_A103_FD	NOK	-	-	-
HVAC_A114_FD	NOK	-	-	-
VDC_A5	NOK	-	-	-
HVAC_A2_FD	NOK	-	-	-
USM_A101	OK	-	-	-

Figure 34 Excel results containing only the post-processing of CUT type tests

Moreover, in a future project, the Python list derived from data extracted from the PVAL HIL could be transformed into classes. This transformation would create an even more understandable structure for a new user. For instance, each frame from the PVAL HIL could be directly accessed by its name in Python instead of its index, thereby containing different lists of variables to control based on each type of test.

An also important part of this project was to optimize the software so that the tool can analyse the data coming from the HIL tests as fast as possible to be able to speed up the development of new Renault projects.
In early June, the tool was able to analyse 61 tests per hour.
Then, by using some existing and fast libraries on Python, such as Numpy or Pandas, it was possible to reduce computation time up to the end of the thesis.
That is why in early August, after optimizing the tool, it was possible to analyse more than 100 tests per hour.

3.5.7. Problems and their resolution

During the development of the tool, I faced different problems, some were hard to fix but not very significant on the results of analysis and some were less hard to fix but very influent on the results.
Therefore, in this part, the different problems will be presented in the order of their influence on the results, and not according to the difficulty of solving them. Always had to modify little parts of the tool to correct it and took a long time to calibrate the time constraints for example because I always had to look at the variables on MDA to set new time constraints margins.

3.5.7.1. "Ressemblance" function: Two methods to link variables with the same Invalid Value message

As it is possible to notice on this example of PVAL HIL with 3 invalid values faulty messages, the signal names of the two first messages are very similar which created a problem in the program. As once the invalid message is obtained, all the linked variables such as detection, confirmation or disconfirmation are found in the PVAL HIL but in this case, the linked variables were not the good ones but the ones of the other similar faulty message. Therefore, all the analysis of some invalid values tests were wrong as the post processing of every invalid value signal depends on its variables and not on the variables of another one which could have been triggered later for example.

CD_CANImpose	VDC_A9	WheelSpeedFR	65535
CD_CANImpose	VDC_A9	WheelSpeedFL	65535
CD_CANImpose	VDC_A9	VehicleSpeed	65535

Figure 35 Invalid values signals in a PVAL HIL

Finally, the other method has been chosen, whose aim is to import the big database file which links all the diagnostics variables to Invalid Values messages.

3.5.7.2. Different samplings between two variables in a frame

```
C:\Users\p119720\OneDrive - Alliance\Valentin\Fichiers Python
COMX\HCB\Debug_HILUniversalTA[25_05_2023]_part_8\Inca_record\HECM_RN10_01.dat
time [9.49425000e-02 ... 8.33429686e+02]
452263 647271 3000
time_confirm_no_frm_it 10000
464421 647321
Traceback (most recent call last):

  File "C:\Users\p119720\AppData\Local\Temp/ipykernel_18608/3390051449.py", line 1, in <module>
    res4 = Post_Process(C4,frame_rate,chem_dossier4,chem_database,'C1E')

  File "C:\Users\p119720\.spyder-py3\temp.py", line 1680, in Post_Process
    test = check_var_CUT_New_Time(Frame,chem_dossier,frame_rate)

  File "C:\Users\p119720\.spyder-py3\temp.py", line 558, in check_var_CUT_New_Time
    if abs(data[var[0]]['data'][i]-var[1])>0.2:
```

Figure 36 Error message in Python

Error message: out of bounds of the pfail_clock variable.

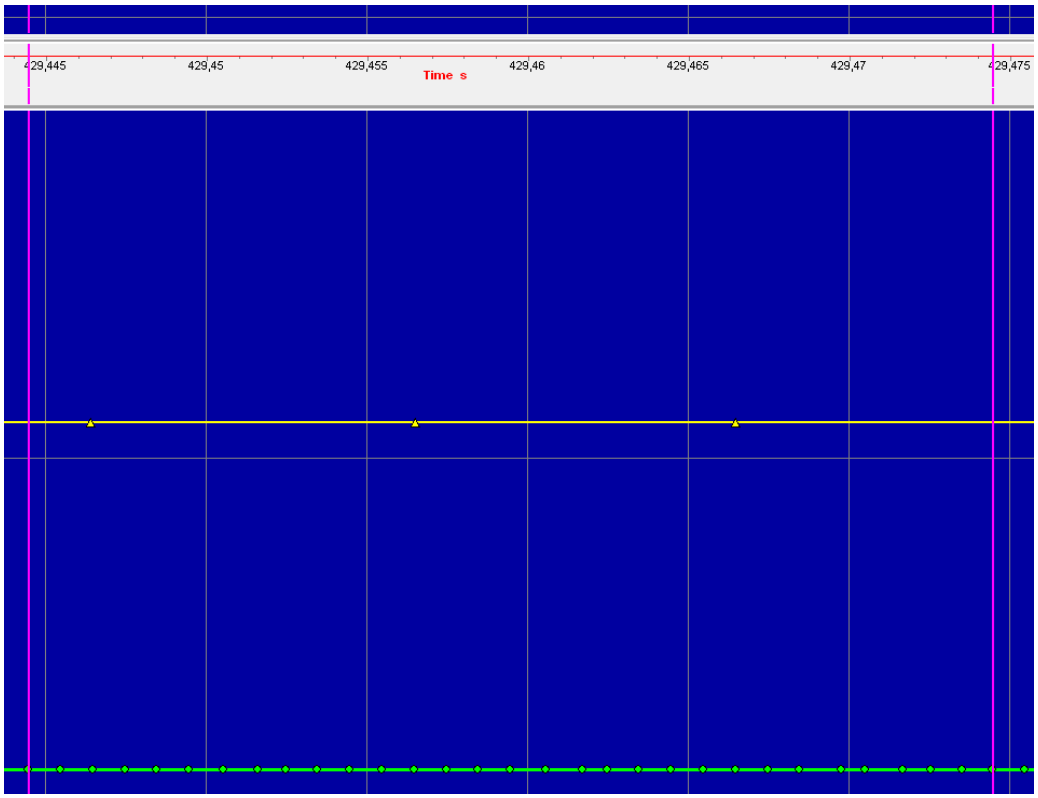


Figure 37 Different samplings of 2 variables in MDA

I ran into a problem concerning the difference of sampling between the step variable and a pfail_clock variable during a CUT test therefore the analysis failed because the iteration of the beginning of the CUT was about 80.000 whereas the length of the pfail_clock variable was about 8.000 samples therefore the test was performed outside the pfail_clock variable's bounds.
Is was a problem because it failed the test whereas it was ok because this pfail_clock variable was not relevant for a CUT test.

```
if var[0] in data and ('deb' not in var[0] and 'sfty' not in var[0]) and data[var[0]]['master']==time_name:
```

finally, we now check if the time sampling (master of the variable) of the variable to check is the same as the one of the step variable.

3.5.7.3. Changing the calculation method of the period of a frame

In the first phase of development of the tool, I was computing the refresh rate of every frame in order not to import too many files in the Post_Process tool. I was computing the refresh rate of each frame thanks to the data of the step variable (Vnx rx) to then make the analysis of the test and apply the time constraints of the AEMS methodology to the variables of the frame based on the computed refresh rate.

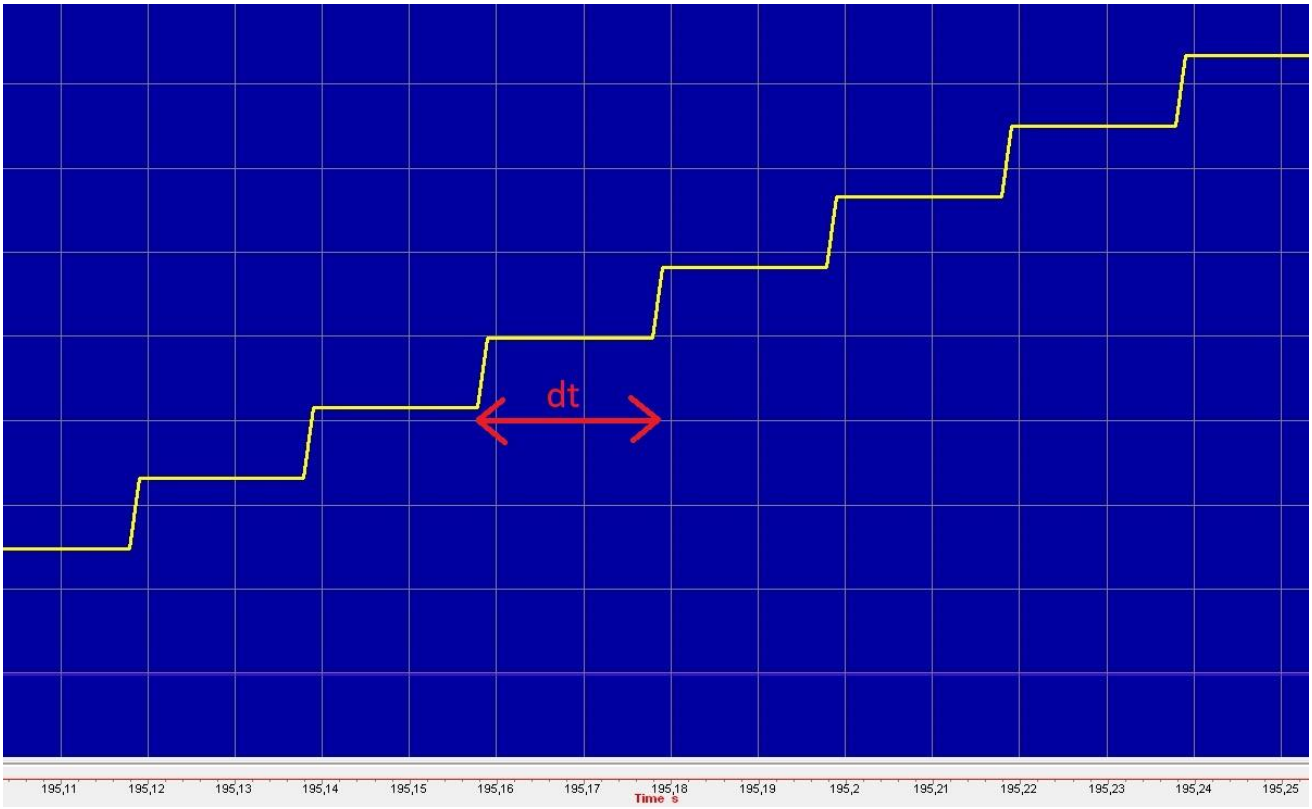


Figure 38 Calculation method of frame emission period

Finally, I chose to use the bridge file to get the dt instead of computing it because there was a problem of acquisition in some .dat file which distorted the computation of dt (In this case, 10 iterations instead of 1000). Consequently, this wrong dt disrupted the post-processing of the frame and released a NOK instead of an OK because the tool assessed if the time constraints between the beginning of the CUT and its detection were respected but they were not as the dt was wrong. Therefore, it is finally a better choice not to compute the dt for every frame because the process of getting the dt of each frame in the bridge file is finally faster.

3.5.7.4. Time constraint margins

It is very important to well calibrate the time constraint margins of the tool not to get a wrong NOK i.e. to assess a test as NOK instead of OK and therefore distort the statistics.

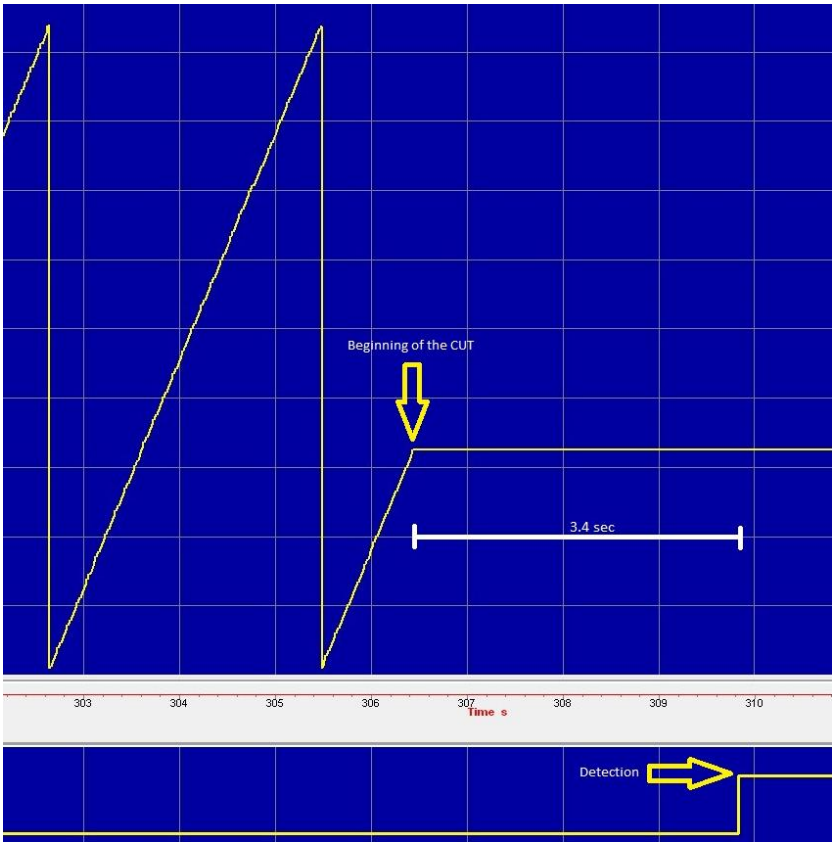


Figure 39 Step and detection variables on MDA

For example, this test was OK because the time period dt of this frame is 1 sec and according to the methodology, the detection should happen about 3*dt after the beginning of the CUT i.e. about 3 sec after the CUT in this case. As it is possible to notice here, the detection occurs 3.4 seconds after the beginning of the CUT which is more than 3 seconds despite the tolerance margin of 20 % of a dt that I had implemented. Therefore, the test was NOK. However, the right tolerance margin which is used by the **tuners** is in fact 50 % of a dt period. Finally, after correcting this tolerance margin, lots more than half of the COMX tests switched to OK.

CAN Frame	CUT
VDC_A114	NOK
METER_A107_FD	NOK
METER_A103_FD	NOK
CSCM_A102_FD	NOK
CSCM_A104_FD	NOK
CSCM_A103_FD	NOK
CSCM_A105_FD	NOK
CSCM_A106_FD	NOK
TIME_R1_FD	NOK
METER_UserSetPref_A1_FD	NOK
CDM_A117	OK
CPLC_R4_FD	NOK
HVAC_A113_FD	OK
CPLC_A103_FD	NOK
CPLC_A107_FD	OK
CPLC_A102_FD	NOK
CPLC_A106_FD	OK
CPLC_A104_FD	NOK
HFM_A101_FD	NOK
BCM_A114_FD	NOK
BCM_R14_FD	NOK
CDM_A111	NOK
VDC_A6	NOK

CAN Frame	CUT
VDC_A114	NOK
METER_A107_FD	OK
METER_A103_FD	NOK
CSCM_A102_FD	OK
CSCM_A104_FD	NOK
CSCM_A103_FD	NOK
CSCM_A105_FD	OK
CSCM_A106_FD	OK
TIME_R1_FD	OK
METER_UserSetPref_A1_FD	NOK
CDM_A117	OK
CPLC_R4_FD	OK
HVAC_A113_FD	OK
CPLC_A103_FD	OK
CPLC_A107_FD	NOK
CPLC_A102_FD	OK
CPLC_A106_FD	OK
CPLC_A104_FD	OK
HFM_A101_FD	NOK
BCM_A114_FD	NOK
BCM_R14_FD	NOK
CDM_A111	NOK
VDC_A6	OK

Figure 40 Results of tests analysis on the Excel report

As it is possible to notice, by putting the right margins, many tests finally switched to OK by putting the right time constraints.

3.6.GUI

3.6.1. Development of the tool's GUI

```
('EBA_A101',
 'CUT',
 ['Phase 1',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_exbrkreqprio_datanok', 1], 'NOK', 1],
   [['Vbx_dem_cnd_ena_brkpdlpos_datanok', 1], 'NOK', 1],
   [['Vbx_dem_act_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_brkpdlpos_datanok', 1], 'NOK', 0],
   [['Vbx_cdm_072_lnk_status', 1], 'OK', 1],
   [['Vbx_cdm_072_det_fail', 0], 'OK', 0],
   [['Vbx_dem_fail_exbrkreqprio_datanok', 0], 'OK', 0],
   [['Vbx_dem_fail_brkpdlpos_datanok', 0], 'OK', 0],
   [['Vbx_dem_pass_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_pass_brkpdlpos_datanok', 1], 'NOK', 0],
   [['Vbx_dem_pfail_exbrkreqprio_datanok', 0], 'OK', 0],
   [['Vbx_dem_pfail_brkpdlpos_datanok', 0], 'OK', 0]],
  'Phase 2',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_brkpdlpos_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_brkpdlpos_datanok', 1], 'NOK', 0],
   [['Vbx_cdm_072_lnk_status', 1], 'OK', 1],
   [['Vbx_cdm_072_det_fail', 1], 'OK', 1],
   [['Vbx_dem_fail_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_fail_brkpdlpos_datanok', 1], 'NOK', 0],
   [['Vbx_dem_pass_exbrkreqprio_datanok', 0], 'OK', 0],
   [['Vbx_dem_pass_brkpdlpos_datanok', 0], 'OK', 0],
   [['Vbx_dem_pfail_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_pfail_brkpdlpos_datanok', 1], 'OK', 1]],
  'Phase 3',
  [[['Vbx_ign_key', 1], 'OK', 1],
   [['Vbx_can_dgn', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_cnd_ena_brkpdlpos_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_act_brkpdlpos_datanok', 1], 'NOK', 0],
   [['Vbx_cdm_072_lnk_status', 1], 'OK', 1],
   [['Vbx_cdm_072_det_fail', 0], 'OK', 0],
   [['Vbx_dem_fail_exbrkreqprio_datanok', 0], 'OK', 0],
   [['Vbx_dem_fail_brkpdlpos_datanok', 0], 'OK', 0],
   [['Vbx_dem_pass_exbrkreqprio_datanok', 1], 'OK', 1],
   [['Vbx_dem_pfail_exbrkreqprio_datanok', 0], 'OK', 0],
   [['Vbx_dem_pfail_brkpdlpos_datanok', 0], 'OK', 0]]])
```

Figure 41 First Python's GUI for test analysis

For each analysis of a frame, I created a GUI which contains:

- The name of the frame
- The type of test
- The results of the analysis of the frame during the 3 different phases

```
[['Vbx_dem_pass_exbrkreqprio_datanok', 1], 'OK', 1],
```

Figure 42 Frame's variable analysis result

Every line of the results contains the name of the variable which has been checked, the value to be checked, the result of the comparison between the expected value and the real value (OK or NOK) and then the real value during the test.

3.6.2. GUI for every user

The aim was to create a Human Machine Interface (HMI) to enable users to use the data analysis tool without needing to know the Python language. The aim was therefore to develop a HMI that combined Arnaud's work with my own. The initial specifications were quite simple: to create a GUI that would allow everyone to obtain the automated report.

The first step was to create a GUI class containing all the necessary elements. The PyQt5 library was chosen, a library dedicated to HMIs in Python, because it offers more functionality than Tkinter.

The first approach was to create a very simple window with three buttons: "Load CNIB file", "Load PVAL Excel file" and "Start analysis". The choice of trial type was made via a drop-down menu, with no additional message displayed. The only objective was to carry out the analysis, and the report was generated at the location of the Python file. This version is called "HMI version 1".

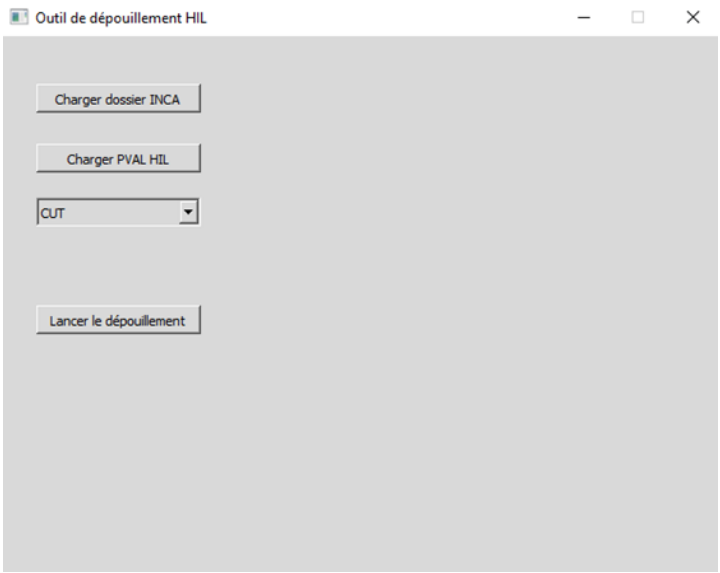


Figure 43 HMI V1

As the counting code evolved, this first GUI became obsolete, although it did the job it was supposed to do. New functions were therefore added. The user now had to load the software database as well as the vehicle platform, as these are used in the counting process. Arnaud and I also wanted to visualize the calculation and progress of the counting in real time. Version 2 of the GUI was therefore created, incorporating two progress bars to display the reading of the PVAL HIL file and the counting of the frames. Visual changes have also been made (Version of 5 June).

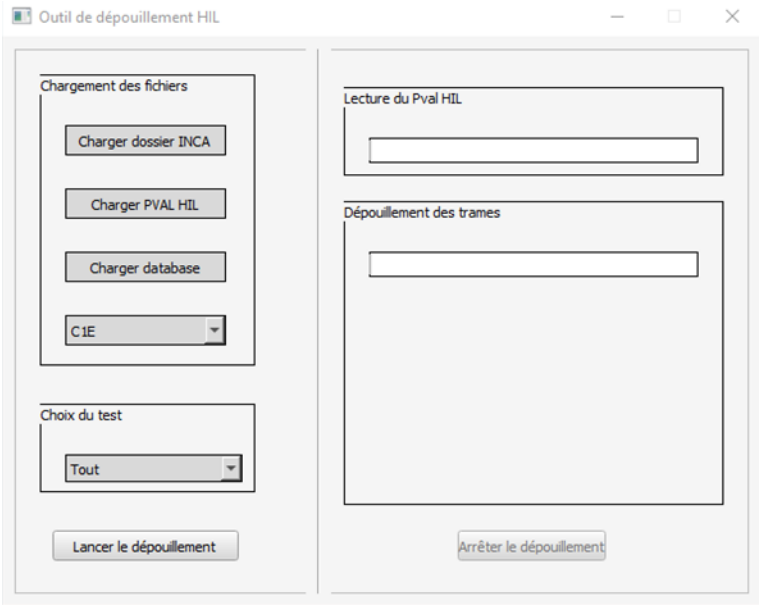


Figure 44 HMI V2

Following the development of the code, the final interface has been completed. To obtain the "lab" file, the user must now select both the software variables file and the software "labels" file. To avoid having too many buttons, the functions for loading the CNIB file and the PVAL HIL file have been merged. The load button can now be used to locate the INCA folder and the PVAL HIL file in the results file sent directly by Romania. A button for loading the bridge file has also been added for post-processing. A display of processed frames has also been integrated into the HMI. What's more, the report and text file are now automatically opened when they are generated at the end of the automatic processing, eliminating the need to search for them and allowing you to choose the download location. This version is dated 31 July 2023."

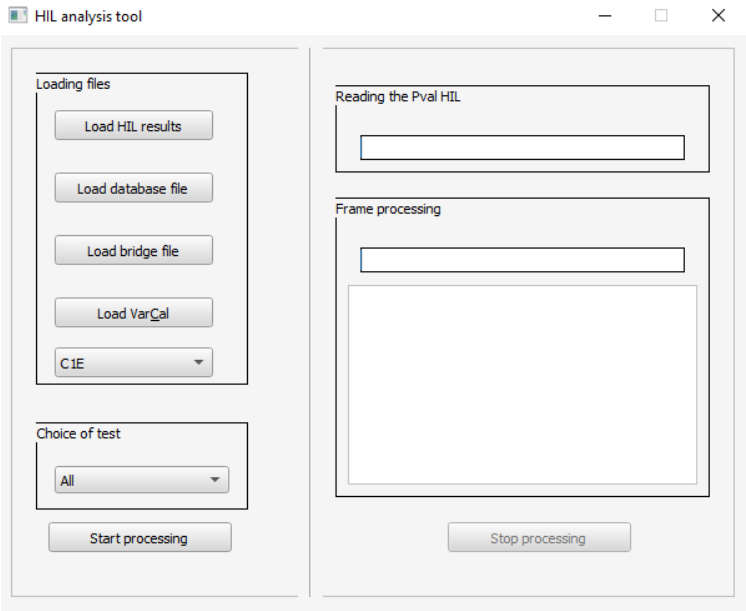


Figure 45 HMI V3

3.7. Validation of the tool

The validation process is a pretty long development phase of the tool because it consists in realising a lot of automated post processing with the tool to then compare the results to the manual post processing of the data on MDA (Measure Data Analyser) module. It is very important to create a safe tool so that to get a right OK or NOK not to compromise the development of a car. A safe tool is one that only enables calibration validation if and when all associated tests are completely successful. Designing a tool which generates false negatives rather than false positives is considerably preferable since if the outcome is a false negative, it will be quickly reviewed and validated manually. If, however, it turns out to be a false positive, this result won't be examined, and any calibrations associated to it could potentially result in future ECU failures, endangering the driver of the car in the future.

Consequently, to validate the tool, I used the HIL tests realized by Renault in Romania on new cars ECUs [8]. Then those HIL (Hardware-in-the-Loop) tests have been manually analysed by the engineers and technicians

of my team as well as subcontractors. Following this period of manual analysis, I ran the automated post-processing tool on the same HIL tests to then compare automated post-processing results with manual post-processing results.

Name of the analyzed frame

Test type

CAN Frame	CUT	CRC	CLock	InVal
USM_A102_FD	NOK	-	-	OK
IVI_A132_FD	NOK	-	-	OK
IVI_R103_FD	OK	-	-	-
ParkOut_SCU_Frame	NOK	-	-	-
EPKB_A1	NOK	NOK	OK	OK
FRCAMERA_A110	OK	-	-	NOK
BCM_R102SC_FD	NOK	-	-	-
CDM_A107	OK	-	-	-
CDM_A103_FD	NOK	-	-	OK
HVAC_A114_FD	OK	-	-	-
VDC_A5	NOK	OK	NOK	-
HVAC_A2_FD	OK	-	-	-
USM_A101	NOK	OK	OK	OK
HVAC_A107_FD	OK	-	-	-
CGW_A101_FD	OK	-	-	-
HVAC_R4_FD	NOK	-	-	-

Automated
Post-processing results

Figure 46 Example of automated post-processing

To achieve the tool validation and so that the tool can be used by every user, even the ones who cannot use Python, an executable has been created to start the tool easily and then directly start a post processing.

4. Tool improvements and maintenance

It is really important to keep on improving the tool after releasing a first working version. Because it is still possible to make it faster by improving the code by using new libraries. But it is also possible to improve it by filtering the signals during the post processing so that to detect the errors at the right times even for irregular signals. It could also be possible to create a code which adapts itself to the new AEMS methodology.
Adaptability in reading the PVAL is also very important to be able to analyze as much data as possible.

4.1. Filtering signals

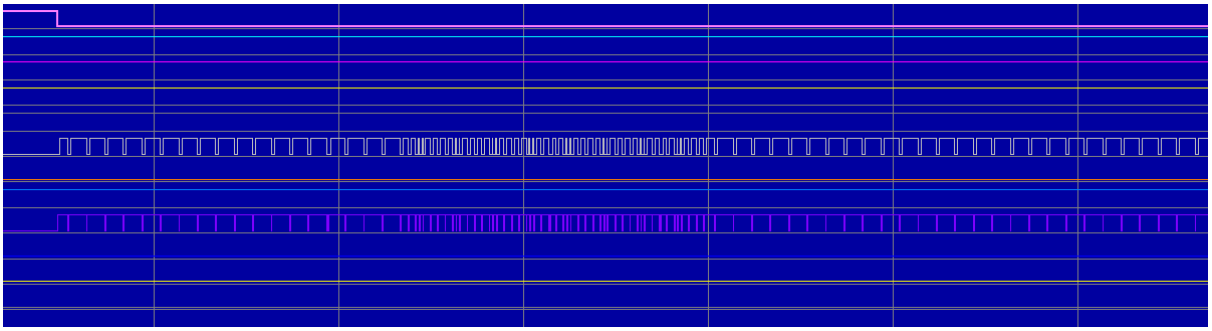


Figure 47 Example of irregular signals

Currently, when a test contains irregular signals like the ones on the graph above, the test is automatically set as NOK because it means that there was a failure during the test. So that means that every test which has technical problem is set as NOK.
However, it could be interesting in the future to adapt the tool to irregular signals so that analyzing the tests could be possible even with technical issues.
Thanks to this approach it could be another way to improve the development of new ECUs software by validating their software even if the signals in the test were not perfectly regular.

4.2. Global variables for the AEMS methodology

In the next years, new rules will force Renault to adapt its AEMS methodology and therefore change time constraints for detection, confirmation and disconfirmation of errors for example. Thanks to this code, it is possible to modify the time constraints of the tool without modifying the whole code in every analysis function.

```
### Time constraints ###

#CUT no_frm#
dt_CUT_detec_nofrm=3
dt_CUT_confirm_nofrm=2    # unit is seconds
dt_CUT_disconfirm_nofrm=0

#CUT datanok#
dt_CUT_detec_datanok=3
dt_CUT_confirm_datanok=3
dt_CUT_disconfirm_datanok=2

#Clock#
dt_Clock_detec_datanok=0
dt_Clock_confirm_datanok=3
dt_Clock_disconfirm_datanok=2

#CRC#
dt_CRC_detec_datanok=0
dt_CRC_confirm_datanok=3
dt_CRC_disconfirm_datanok=2

#Invalid Values#
dt_InVal_detec_datanok=0
dt_InVal_confirm_datanok=3
dt_InVal_disconfirm_datanok=2
```

Figure 48 Setting time constraints in Python

Then, a next step in the improvement of the tool could be to use AI to read the new time constraints in the AEMS methodology and then update the tool in a few seconds.

4.3. Adaptability of the tool

It is also really important to make an adaptable tool as sometimes the input files may contain errors that the tool should recognize to overcome this problem and keep on analysing the file.
For instance, I encountered a problem in one part of the New Captur's PVAL HIL. In fact, when the tool reads the PVAL HIL to extract all the relevant information, it detects the transition from one vehicle to another using the cell "**CD_SetKeyOn**" to know the beginning of a test and "**CD_SetKeyOff**" to know the end of a test.

On the New Captur's PVAL HIL, the "**CD_SetKeyOn**" cell was missing for one test, therefore when the tool wanted to detect the next test, it never detected it as the condition needed to know the beginning of a test is this "**CD_SetKeyOn**" cell.
Finally, as this error occurred only once I corrected the PVAL HIL by adding the missing "**CD_SetKeyOn**" cell.

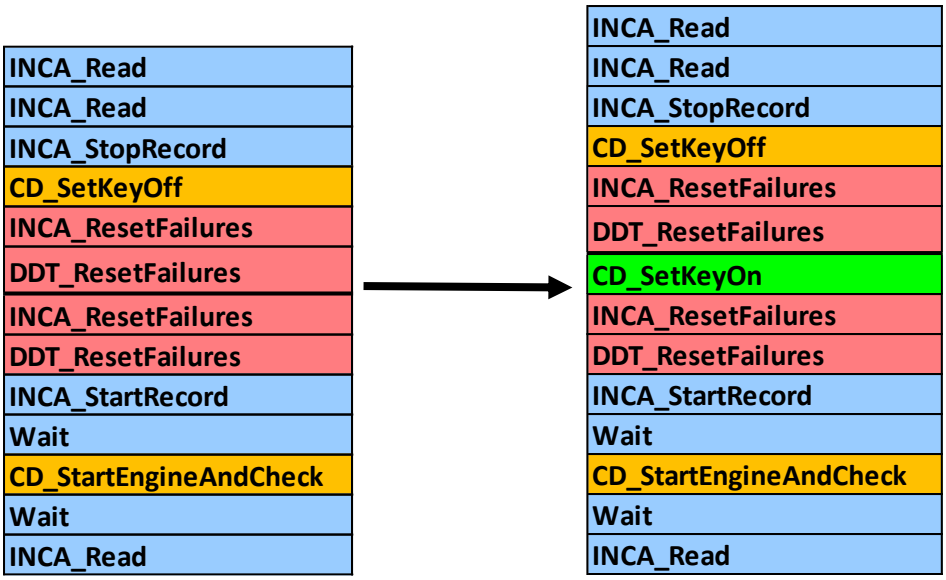


Figure 49 PVAL HIL error leading to an error during post processing

Conclusion

This study focused on the development of an automatic analysis tool as part of the validation of ECU software. After studying and understanding the manual post-processing method, the main functions required to implement the post-processing tool have been created.
By developing the three main post-processing functions *check_var_CUT_New_Time, check_var_CRC_Clock, check_var_InVal* as well as *Post-process* function, I have been able to automatically post-process the four test types i.e., CUT, CRC, Clock and Invalid values. While validating the tool, its produced results were exactly the same as those obtained by manual post-processing.
Moreover, to analyse all the frames of an ECU, it takes half a day by doing it automatically instead of doing it manually during two weeks by three engineers and technicians.
Consequently, this automatic post-processing tool for HIL bench tests frees up human resources so that they can concentrate on tasks with greater added value, such as resolving errors detected by the tool to realize a better ECU software. In addition, a GUI has been developed to enable the tool to be used by everyone.
In conclusion, the tool is already being deployed on new Renault projects to improve productivity and quality.

This end of studies internship gave me the opportunity to work in the applications/functions team of the Renault Lardy Technical and Test Centre. It gave me an overview of how car ECU development is carried out, as I have developed a tool to validate the correct operation of ECUs for new Renault projects. Through this work, it is possible to notice that I gained both technical and human skills thanks to the opportunity to realise an internship at Renault.

Working in such an environment of car enthusiasts was a pleasure given my passion for cars, I was learning without feeling like I was working. I find it fascinating to design a vehicle and then to be able to try out its functions directly afterwards on the track, like at the Lardy centre. I also had the privilege to get an insight into the world of crash tests which was very interesting to understand better the development of cars.

Moreover, developing the tool for automating the analysis of engine control units for our team and our colleagues in Romania and Spain was very stimulating and rewarding. Working in this environment gave me the opportunity to discover the main steps in the conception of new ECUs software as well as to gain technical knowledge in this field. I also had to face several challenges, in particular the development of a tool under pressure from project teams which needed it to improve their development of ongoing projects as fast as possible but I adapted myself and I can only have incredible memories of this experience.

Furthermore, this internship introduced me to the organisation of a large and French automotive company. Living in a city where the car industry has to adapt drastically to take account of the growing ecological challenges was very interesting and motivating to maintain and strengthen my desire to work in the automotive industry.

Bibliography

[1] L. Jeannier et Y. Reneme, *Méthodologie de calibration et de validation : Méthodologie pour l'architecture C1H en A-EMS,* Paris: Internal document of Renault, 2020.

[2] ETAS, «INCA – Produits logiciels,» 2023. [En ligne]. Available: https://www.etas.com/fr/produits/inca_software_products.php. [Accès le 2023].

[3] ETAS, *INCA V6.2 Didacticiel,* Paris: ETAS, 2008.

[4] Aptiv, "What is hardware-in-the-loop testing?," 24 March 2022. [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-hardware-in-the-loop-testing. [Accessed July 2023].

[5] M. Denayrolles, *HIL SW & Calib VALIDATION,* Paris: Internal document of Renault, 2023.

[6] IFP: Institut Français du Pétrole (French Institute of Petroleum), *Fondamentaux sur le fonctionnement des moteurs (Fundamentals of engine operation),* Rueil-Malmaison: IFP school, 2019.

[7] Renault Group, *A-EMS Software specification, Engine Control Strategies: Conventional, hybrid and EV systems,* Paris: Internal document of Renault, 2022.

[8] Renault Group, *HIL Bench usage,* Paris: Internal document of Renault, 2015.

[9] M. Denayrolles, *Methodology for using ATCL Appli COMX,* Paris: Internal document of Renault, 2023.

Table of abbreviations

AEMS: Alliance Engine Management Software

ECU: Electronic Control Unit

MDA: Measure Data Analyzer

COMX: Communication Across all Electronic Control Units

HIL: Hardware-in-the-loop

DATANOK: Data not OK

CUT: Frame break

CRC: Cyclic Redundancy Check

CAN: Controller Area Network

OBD: On-Board-Diagnostic

PVAL: Validation Plan

SW: Software

Calib: Calibration

BMS: Battery Management System

VDC: Vehicle Dynamic Control

ACC: Adaptive Cruise Control

NOK: Not OK

PFAIL: Pre-failure i.e., before failure

Table of figures

Figure 1 Renault Lardy Technical and Test Centre 6
Figure 2 CAN (Controller Area Network) Architecture representing the communication between the different ECUs [5] 8
Figure 3 Validation Plan (PVAL) example for one type of test 10
Figure 4 HIL bench architecture [9] 11
Figure 5 ECU software architecture 13
Figure 6 Frame architecture [9] 14
Figure 7 Different types of tests in COMX 14
Figure 8 Time constraints in the AEMS methodology [Appendix 2] 15
Figure 9 Example of a frame behavior during a CUT test 16
Figure 10 Datanok failure 16
Figure 11 No more connection between sender and receiver 17
Figure 12 Diploma thesis schedule 18
Figure 13 General operating principle of the tool 19
Figure 14 PVAL HIL template 20
Figure 15 Extracted PVAL HIL of a CUT test in Python 21
Figure 16 list_seq function code beginning 22
Figure 17 Example of an Excel Validation Plan (PVAL) HIL showing start and end of a test 22
Figure 18 List of data path linked to frames names and types of tests 24
Figure 19 CUT test principle 24
Figure 20 Step variable example 25
Figure 21 Step variable reaction during a CUT 26
Figure 22 Last version of extracted PVAL HIL of a CUT test in Python 27
Figure 23 Analysis result of a CUT test 28
Figure 24 Clock and CRC principles 29
Figure 25 Frame architecture 29
Figure 26 Callback variable : represents the frame activity, one step every period 30
Figure 27 Invalid values test principle 31
Figure 28 Example of an invalid value PVAL containing 2 invalid values 32
Figure 29 Frame architecture 32
Figure 30 Example of 3 invalid values detection variables with different starts 33
Figure 31 Faulty Invalid Value test 34
Figure 32 Faulty invalid value test Excel report 34
Figure 33 Post-processing function code in Python 35
Figure 34 Excel results containing only the post-processing of CUT type tests 36
Figure 35 Invalid values signals in a PVAL HIL 37
Figure 36 Error message in Python 37
Figure 37 Different samplings of 2 variables in MDA 38
Figure 38 Calculation method of frame emission period 39
Figure 39 Step and detection variables on MDA 40
Figure 40 Results of tests analysis on the Excel report 41
Figure 41 First Python's GUI for test analysis 42

Figure 42 Frame's variable analysis result 43
Figure 43 HMI V1 43
Figure 44 HMI V2 44
Figure 45 HMI V3 45
Figure 46 Example of automated post-processing 46
Figure 47 Example of irregular signals 47
Figure 48 Setting time constraints in Python 48
Figure 49 PVAL HIL error leading to an error during post processing 49
Figure 50 AEMS Methodology 56

Appendix 1: Assessment report

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Pauron Valentin** Personal ID number: **514461**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Automotive, Combustion Engine and Railway Engineering**

Study program: **Master of Automotive Engineering**

Branch of study: **Advanced Powertrains**

II. Master's thesis details

Master's thesis title in English:

Development of a software automation tool for analysis of engine control unit error handling

Master's thesis title in Czech:

Softwarový nástroj pro automatizaci reakci řídící jednotky motoru na chybové stavy

Guidelines:

Engine control units (ECU) have sophisticated diagnostic tools which allow for various malfunctions and undesired anomalies to be detected and communicated to the driver. Hardware in the loop (HIL) tests, where part or all inputs and outputs are simulated, are performed on ECU before being fully integrated into the vehicle during the design process.

The goal of the diploma thesis is to design and implement a software tool, which will automate the analysis of HIL tests and test drives, during which various anomalous or errorneous signals are generated, from invalid checksum and other communication errors to values which are unrealistic or out of the normal range. The response of the ECU, including messages delivered to the driver or other actions, will be recorded and assessed. The software will be written in Python or other high-level language.

The experimental work will take place at Renault facility in Lardy, France, under the supervision of Frédéric Avarguez. The thesis and its defense will be public; confidential information can be withheld, but sufficient details need to be presented to assess the quality of work.

Name and workplace of master's thesis supervisor:

Vojtíšek Michal, prof., M.S., Ph.D.

Name and workplace of second master's thesis supervisor or consultant:

Frédéric Avarguez, Renault facility in Lardy, France

Date of master's thesis assignment: **27.06.2023** Deadline for master's thesis submission: **11.08.2023**

Assignment valid until: _____

Vojtíšek Michal, prof., M.S., Ph.D.	doc. Ing. Oldřich Vítek, Ph.D.	doc. Ing. Miroslav Španiel, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

CVUT-CZ-ZDP-2015.1

Appendix 2: AEMS Methodology

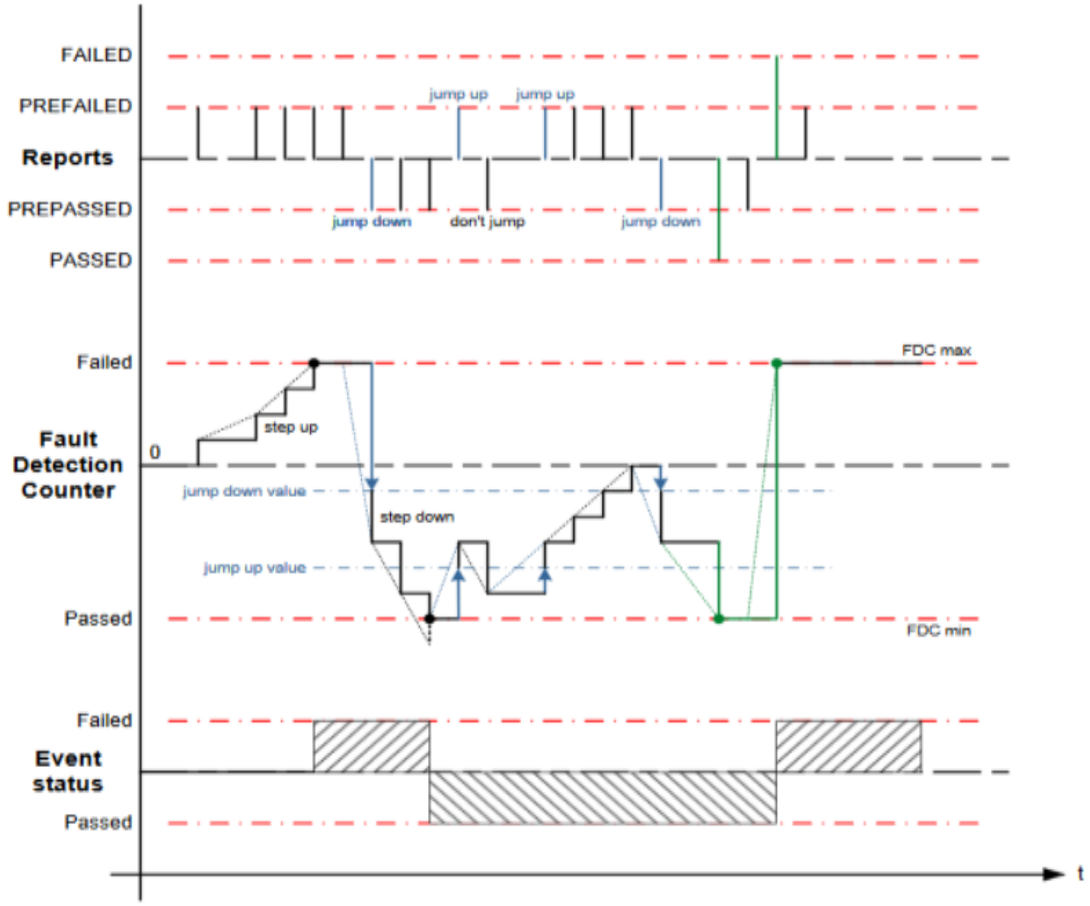


Figure 50 AEMS Methodology

Appendix 3: liste_seq code

```
def liste_seq(ch,frame_rate):            #Returns the list of the frames, the type of test and th

    File=read(ch)
    PVAL_HIL = File.to_numpy()
    Liste_Seq=find_index(ch)
    L_S=[]                               #Final list of sequences
    for seq in Liste_Seq:
        phase1_start=0
        phase2_start=0
        phase3_start=0
        phase3_end=0
        Frame_Name=""
        Err_Name=""
        P=[]
        P1=[]
        P2=[]
        P3=[]
        step_CUT=""
        Msg_InVal=""
        key_on=seq[0]
        key_off=seq[1]

        for i in range(key_on,key_off+1):
            if type(PVAL_HIL[i][4])==str:
                if len(PVAL_HIL[i][4])>=19:
                    if PVAL_HIL[i][4][0:6]=='Vnx_rx' and (PVAL_HIL[i][4][-11:])=='cbk_cs_comc':
                        if len(step_CUT)==0:
                            step_CUT=PVAL_HIL[i][4]
                            # print(step_CUT)

            if PVAL_HIL[i][0]=='CD_SetKeyOff':
                phase3_end=i

            if PVAL_HIL[i][0]=='CD_SetKeyOn':
                phase1_start=i
            if PVAL_HIL[i][0]=='CD_CANUncheckFrame':      #If this cell has this name, it means t
                phase2_start=i
                Err_Name="CUT"

            if PVAL_HIL[i][0]=='CD_CANcheckFrame':
                phase3_start=i

            if PVAL_HIL[i][0]=='CD_CANImpose':
                phase2_start=i
```

```
                phase2_start=i
                if type(PVAL_HIL[i][2])==str and PVAL_HIL[i][2][0:3]=="CRC":
                    Err_Name="CRC"
                if type(PVAL_HIL[i][2])==str and PVAL_HIL[i][2][0:5]=="CLock":
                    Err_Name="CLock"
                if type(PVAL_HIL[i][2])==str and PVAL_HIL[i][2][0:3]!="CRC" an
                    Err_Name="InVal"
                    # Msg_InVal=PVAL_HIL[key_on+3][3]    #Getting the messages
                    # Msg_InVal = Msg_InVal.split(";")
                    # print(Msg_InVal)


            if PVAL_HIL[i][0]=='CD_CANResetMessage':
                phase3_start=i

            if len(Frame_Name)==0:
                if type(PVAL_HIL[i][1])==str:
                    Frame_Name=PVAL_HIL[i][1]

        if Err_Name=="InVal":
            Msg_InVal=frame_rate[Frame_Name][2]
            Msg_InVal = Msg_InVal.split(";")

        for i in range(phase1_start,phase2_start):
            if type(PVAL_HIL[i][4])==str and type(PVAL_HIL[i][5])==int:
                P1.append([PVAL_HIL[i][4],PVAL_HIL[i][5]])

        for i in range(phase2_start,phase3_start):
            if type(PVAL_HIL[i][4])==str and type(PVAL_HIL[i][5])==int:
                P2.append([PVAL_HIL[i][4],PVAL_HIL[i][5]])

        for i in range(phase3_start,phase3_end):
            if type(PVAL_HIL[i][4])==str and type(PVAL_HIL[i][5])==int:
                P3.append([PVAL_HIL[i][4],PVAL_HIL[i][5]])
        error=0
        last_itt=0
        if Err_Name == "InVal":
            P=[Frame_Name,Err_Name,P1,P2,P3,Msg_InVal,step_CUT]
        else:
            P=[Frame_Name,Err_Name,P1,P2,P3,step_CUT]

        if phase2_start!=0:                    #Condition to fulfill to check
            L_S.append(P)
    return L_S
```