Politecnico di Torino

Master's degree in Automotive Engineering

Automatization of an HIL test bench for Engine Control Unit validation

Student

Guiso Riccardo

Professor

D'Ambrosio Stefano

Correlator

Finesso Roberto

Academic Year 2020-2021

Contents

List of Figures

1 Thesis overview

In a turbulent period where the complexity of the vehicle is increasing steeply day by day, it is important to find smarter ways to validate a software that can satisfy the needs of the client, and the new regulations imposed by the government in order to reduce the pollutant emissions.
In this thesis, it is possible to find a description of the techniques used in order to validate a software. Thanks to the interest on development of a new kind of approach of test validation of the company Westport Fuel System Italy, it was possible to discover the innovative way of the automatic tests: Automation Desk is the main software (developed by dSpace) that is used in order to test in an automatic way the software implemented in the control unit. The automatic test covers the test points decided a priori from the user and, it will give the result of the test. This SW covers the test points that are defined from the user and checks if the expectation were satisfied. A preliminary introduction of the chapters is presented in the following paragraph.
In the first chapter, a general discussion about the normative OBD-2 is done, it follows a punctual description of the V-cycle: a graphical representation of the steps that must be covered in order to validate a software. These two topics are very important in order to understand the process and the rules which the company follows.
The second chapter enlightens the different types of approach that can be used in order to validate a software. The types of tests that can be done are SiL, MiL, HiL and vehicle tests; in the thesis it is possible to find a particular focus on HiL and the hardware used during the activity.
The next chapter focuses its attention to the software functionality: a first description of the functions (Misfire detection and Pre-Catalyst Oxygen sensor) from the physical point of view is done, then, a description of the technical requirements is presented and an explanation of the Simulink logic concludes the chapter.
The fourth chapter covers the innovative part of the thesis, in which the new technique about the automatic test is introduced; the activity was done in parallel with Altran (Capgemini) using the software "Automation Desk" (developed by dSpace).
The last chapter is used for results and conclusion of the thesis: in this case, a discussion of the result and of the limits of this research is done.

1.1 Purpose of the thesis

The purpose of the thesis is to present a modern type of validation for a software that will be implemented in the control unit for a vehicle. In this work, a particular focus on two software components is done. The two software functionalities are compliant with OBD-2 legislation, and they are useful in order to give as output the result of a continuous check on the components of the engine. In particular, the first functionality analyzes the correct operating functioning of the catalytic converter in order to take under control the pollutant emissions and the state of the catalyst itself, instead, the second functionality can detect the absence of the ignition inside the combustion chamber (commonly known as misfire event). The new type of validation can be very useful in the future of the company in order to be more precise respect to a manual test, and to reduce the time necessary to validate a software.

2 Normative and organizational approach for software validation

This chapter is dedicated to point out the normative OBD-2 and the organization style that is used inside the company. In particular, a description of the V-cycle and the main steps that must be covered in order to validate a software is done; it follows a discussion about the history of the normative OBD and the main faults that are analyzed in order to alert the driver of a problem. Nowadays, the V-cycle is the most used approach in the majority of the companies, especially in the automotive sector and it can be very useful in order to have a schedule of the works.

2.1 V-cycle: organization of the software development inside Westport Fuel System Italy

The V-cycle, or V-model, is a graphical representation of a software development process. It is constituted by several steps that must be covered in order to validate a software. The definition of the steps can be different and depends on the ethic adopted by the company. A general description of the steps can be seen in figure 1.

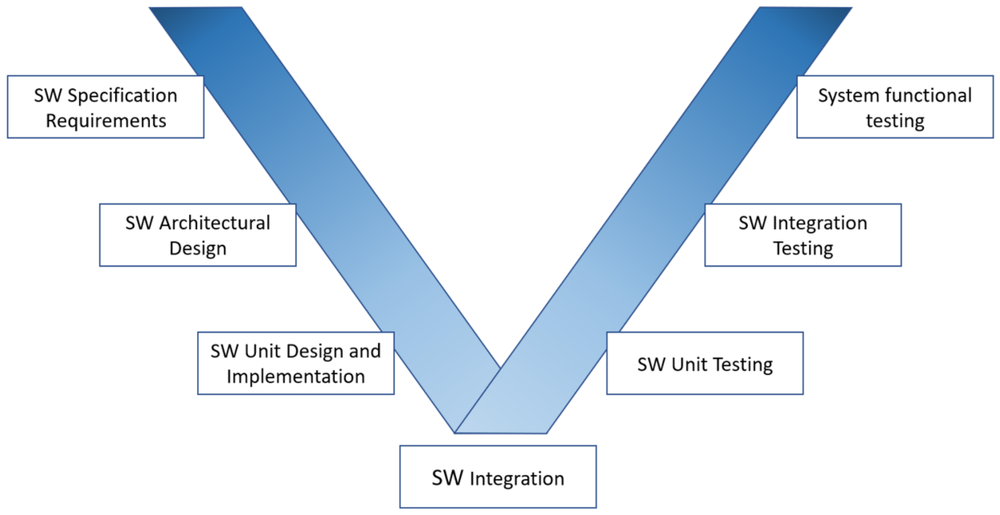

SW Specification Requirements
SW Architectural Design
SW Unit Design and Implementation
SW Integration
SW Unit Testing
SW Integration Testing
System functional testing

Figure 1: V-cycle scheme

The steps are:

- SW Specification Requirements: a document file represents what it is necessary in order to satisfy a particular need from the technical point of view;
- SW Architectural Design: the tools that can be used in order to represent the logic can be various, e.g. Excel, PowerPoint, or Simulink, but every tool have to describe what are the critical points in order to accomplish the requirements, from the logic point of view;
- SW Unit Design and Implementation: the software funcionality is created using Simulink as toll to create the logic, and to generate C-code, Embedded Coder is used (the code is generated automatically);

- SW Integration: the C-code is implemented in a microprocessor in order to be tested in different ways as SIL, HIL and on vehicle;
- SW Unit Testing: this is the first step in which the logic is tested on SIL or MIL and occasionally this is done in parallel with the step 'SW Unit Design and Implementation';
- SW Integration Testing: a more advanced step using a more realistic approach can be done considering a test on HiL (a particular model that simulate the engine and its functions is required);
- System functional testing: this is the last step and the unit is tested on a vehicle using predefined calibrations that simulate the critical points.

During the activity in Westport, the focus was on the step nr. 6 'SW Integration Testing', in which a new approach for the test was introduced. In order to develop this new technique, the training activity was done in parallel with the external company Altran. But, the focus of the stage was centered even to the previous steps, in fact, my personal contribution was to create two software functions basing on their requirements, and to test them using different approaches, as it was already stated in the introduction.
The general activity started with the analysis of the software requirements: this step is very important in order to understand the physical function of the single software functionality and to start the design of its logic in Simulink's environment. The same activity was done in a team where it was possible to discuss about the possible solutions and approaches to satisfy the requirements. The logic was built singularly and discuss with the same team in order to find discrepancies or inconsistency with the software requirements.
The same activity continues with the integration of the software functionality inside the control unit and with the tests in a more realistic environment respect to host PC.
The teamwork in all the stages was very important in order to communicate any problems to the other entities and discuss about the requirements themselves.

2.2 OBD-2 Legislation

The OBD system informs about the pollutant emissions of the vehicle. The monitoring of the vehicle and its emission is done using the lambda sensor: this is the main component that takes under control the quantity of the emissions emitted by the vehicle. The sensor is inside the catalyst and monitors the quantity of oxygen that is going out from the vehicle, so it is possible to know how much pollutant emissions are produced during the function.
Initially, the OBD system was used in order to take under control the pollutant emissions, because, in the past, it was thought that an increase of pollutant emission could be related to a deterioration of some components that were responsible of the conversion of the elements.
The first generation of diagnosis was OBD-1 (it can be simply defined as OBD) and it was implemented in 1988 just to monitor the pollutant emissions. The second generation (OBD-2) was developed in order to take under control the life components of the engine too. In fact, the new system does periodic or continuous checks of the operating conditions and of the single components. In case of a malfunction, the system will turn on a LED in order to advice the driver to stop the vehicle or to do maintenance.
The continuous checks are:

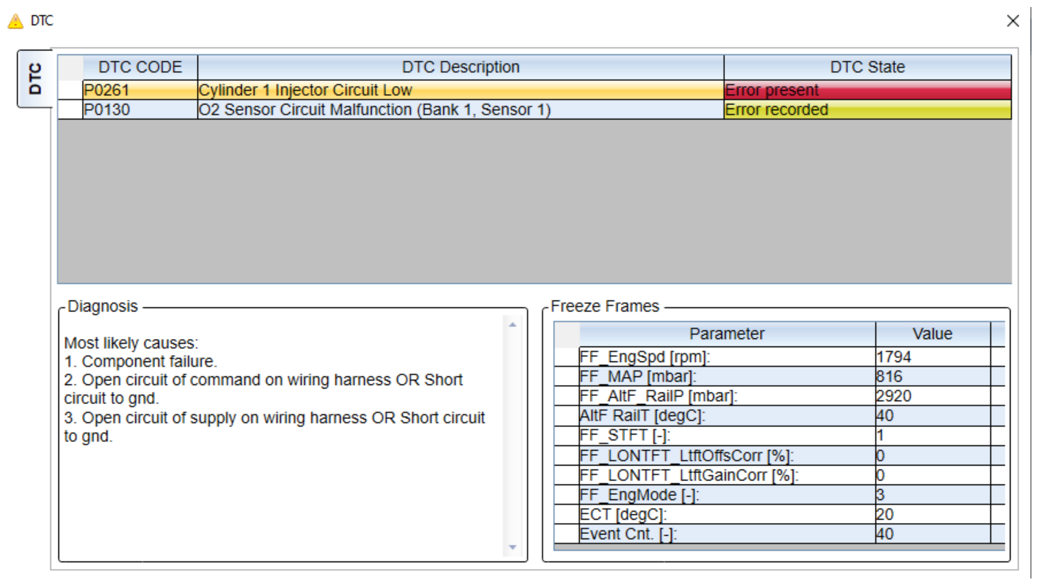
- unexpected loss of torque;
- erroneous quantity of injected fuel inside the combustion chamber;

- CCM (Carbon Ceramic Material, e.g. piston, crankshaft)

Once the system is on, the OBD checks continuously these factors, especially the request of fuel and the loss of torque.
The periodic checks are made in different components:

- EGR system;
- O2 sensors;
- heater O2 sensors;
- secondary air;
- catalytic converter.

Respect to the continuous checks, these can not have a continuous monitor, this is due to the fact that they can operate only with particular conditions, so, for this reason, they are called periodic checks.
In order to have the complete usage of the OBD-2, it is necessary to work on different conditions. These conditions can be created during a dedicated driving cycle that includes highways and urban roads. For what concern the lighting of the MIL (led or symbol that comes on only when it is detected a fault), it comes on only when a particular operating condition is present, and it comes off only if the condition is not more present (so the system considers the damage to one component not so important to stop the vehicle) or there is maintenance of the vehicle (the damage is considered important and a more precise check is needed).
The faults generated by malfunction of the engine or another component are named DTC (Diagnostic Trouble Codes). When this error is detected, the control unit stores this information and the MIL comes on, and if the damage was just a temporary not critic condition, then, at the successive crank of the engine, the MIL is off (or in the successive trips).
The two software functionalities that were designed and validated during the stage in Westport, are very important from the OBD legislation point of view, in fact, the first software (Pre-Catalyst Oxygen Sensor) takes under control the air/fuel ratio in function of the operating conditions, while the second software (Misfire) detects the absence of ignition or a partial ignition inside the combustion chamber. The misfire detection is a permanent check because it can be a cause of loss of torque, while the analysis of the air/fuel ratio is a periodic check, as it was stated before.
The detection of a fault can be done just watching when the MIL comes on in the dashboard of the driver, but, in order to identify what is the correct factor that produced that fault, it is necessary to connect the host PC with a CAN-line and, for the case of my activity, use the BRC Calibration Tool. This software allows to program the control unit, so to update the version of the software in the ECU, and, at the same time, identifies the particular fault that enable the lighting of the MIL.


DTC

DTC

DTC CODE	DTC Description	DTC State
P0261	Cylinder 1 Injector Circuit Low	Error present
P0130	O2 Sensor Circuit Malfunction (Bank 1, Sensor 1)	Error recorded

Diagnosis

Most likely causes:
1. Component failure.
2. Open circuit of command on wiring harness OR Short circuit to gnd.
3. Open circuit of supply on wiring harness OR Short circuit to gnd.

Freeze Frames

Parameter	Value
FF_EngSpd [rpm]:	1794
FF_MAP [mbar]:	816
FF_AltF_RailP [mbar]:	2920
AltF RailT [degC]:	40
FF_STFT [-]:	1
FF_LONTFT_LtftOffsCorr [%]:	0
FF_LONTFT_LtftGainCorr [%]:	0
FF_EngMode [-]:	3
ECT [degC]:	20
Event Cnt. [-]:	40

Figure 2: Example of a fault recorded in the control unit and detected by software

In figure 2, it is presented an example of the result given by the BRC Calibration Tool. In the next chapter, a more specific discussion about the approaches of how to validate the software is done.

3 Validation types

This chapter is dedicated to the different approaches to validate a software, considering the V-cycle, this part points out the right branch. The diffusion of the ICE in the automotive sector pushes the different companies to develop their technologies and to find new ways of validations to be more precise and faster. Nowadays, this aspect is very important in order to respect the more stringent limits on pollutant emissions. In this period, the most diffused approach in order to validate a software is the V-cycle and, in particular, it takes in consideration different techniques, such as: MiL (Model-In-The-Loop), SiL (Software-In-The-Loop) and HiL (Hardware-In-The-Loop). After these steps, it is necessary to use the most realistic approach: vehicle test.

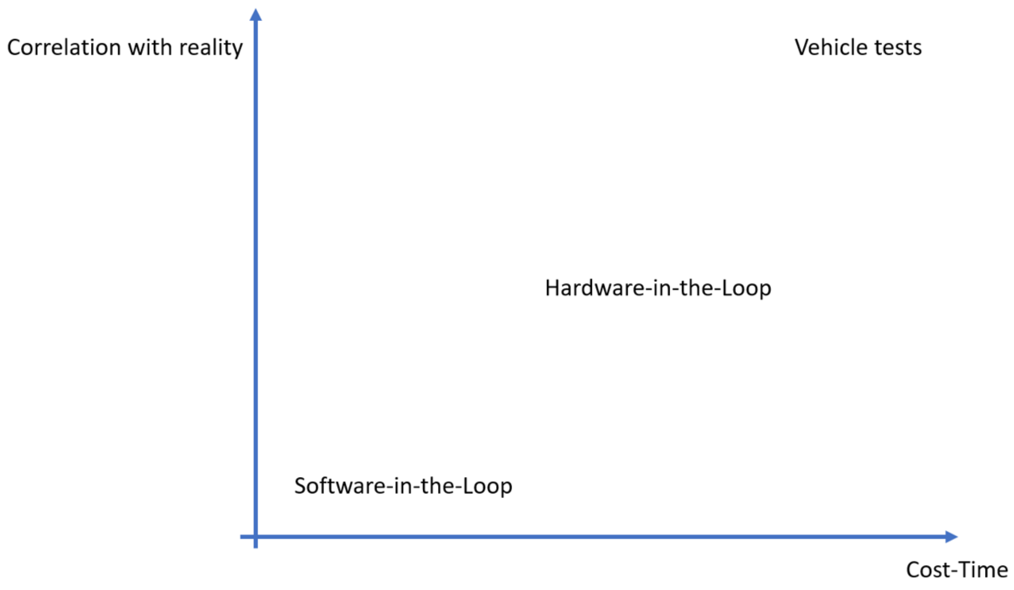


Figure 3: Scheme for correlation with reality of the different approaches

As it is possible to see from figure 3, there is a direct proportionality between cost-time and real world correlation, in fact, the most complex and long approach is the best (vehicle test) but, for preliminary tests, it is better to check errors in a simple PC and occasionally use a hardware that can simulate the real world (HiL).
The compromise is always based on the time and on the complexity of the system in order to validate the software: generally, as it is possible to deduce from figure 3, the choice is always between a simple model with a low simulation time and a complex system with a long simulation time. Obviously, this choice is dependent on the available hardware and on the performance of the same. On the next sections a discussion about the different approaches that were used in order to validate the software functionalities is done. In particular, an analysis on the components and on the performances that they can have in order to validate the software is done.

3.1 MiL, SiL and HiL

The first type of approach that can be used in order to validate a software is the MiL: this step can be covered using a dedicated environment like Simulink. In this thesis, Simulink plays a major role for the design of the software, in fact, there is a dedicated section where it is used in order to create the logic that is useful to accomplish the technical requirements of the SW functionality.

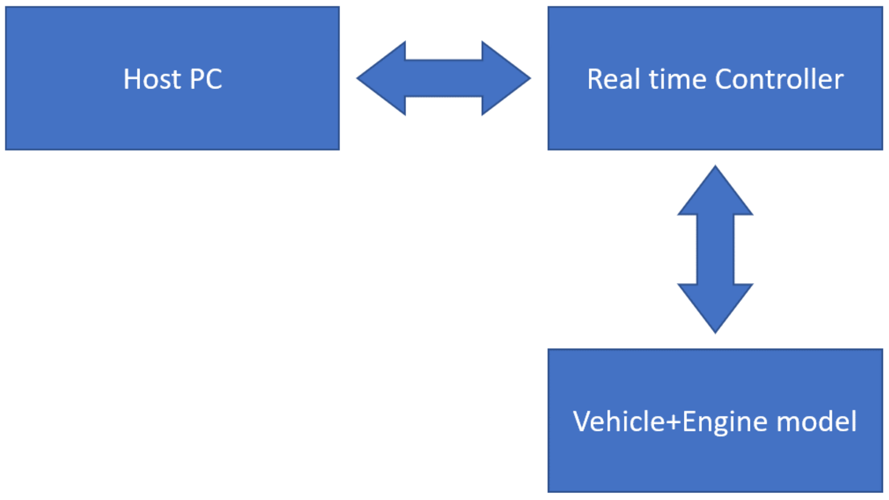


Figure 4: Software-in-the-Loop scheme

As it is possible to see from figure 4, no hardware or ECU is used to validate the software (except for the PC). The validation can be done in Simulink emulating the inputs that are useful for the software functionality. At this level, there is not so much complexity in terms of input and there is not the possibility to record a result in an external memory (Not Volatile Memory). The model runs in processor-time, which is much faster than real-time.

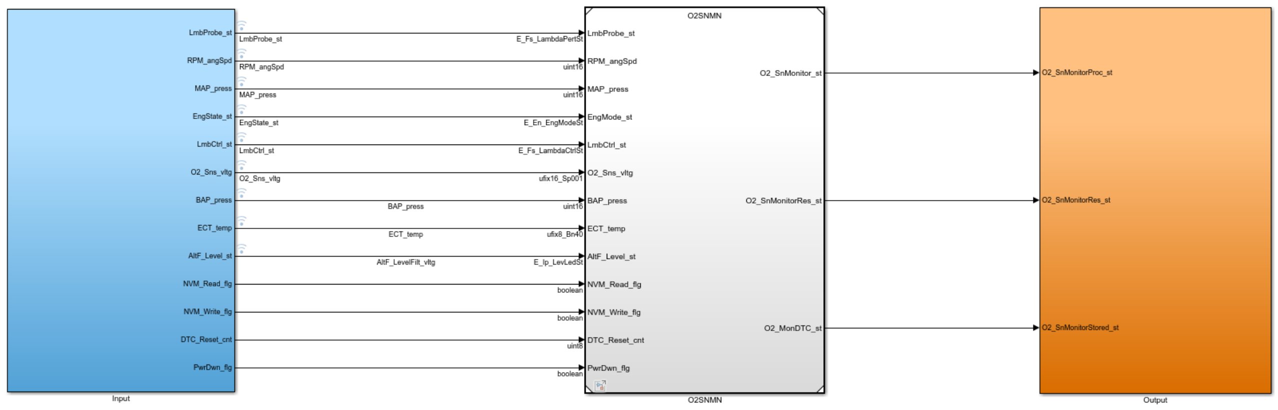


Figure 5: Software-in-the-Loop Simulink blocks

As it is possible to see from figure 5, the tests that can be done in MiL are very simple: the inputs are simulated as signals (blue block) and they are analyzed in the software (white block) in order to produce an output (orange block). This is a limited analysis of the logic, because the input that are produced are not the same of a real driving cycle, and it is not possible to store an information inside the ECU or in another not-volatile memory.
The next step is to use the Software-In-The-Loop: there are no differences for what concern the types of components, but now the model is compiled into C-code and run with a fixed time-step. In this step, it is necessary to see if there are differences between the results from the

SiL and the results from MiL, and, in case of discrepancies, it is necessary to modify the model in Simulink to correct them. So for this step, the same figure utilized for MiL can be used to describe the architecture of SiL.

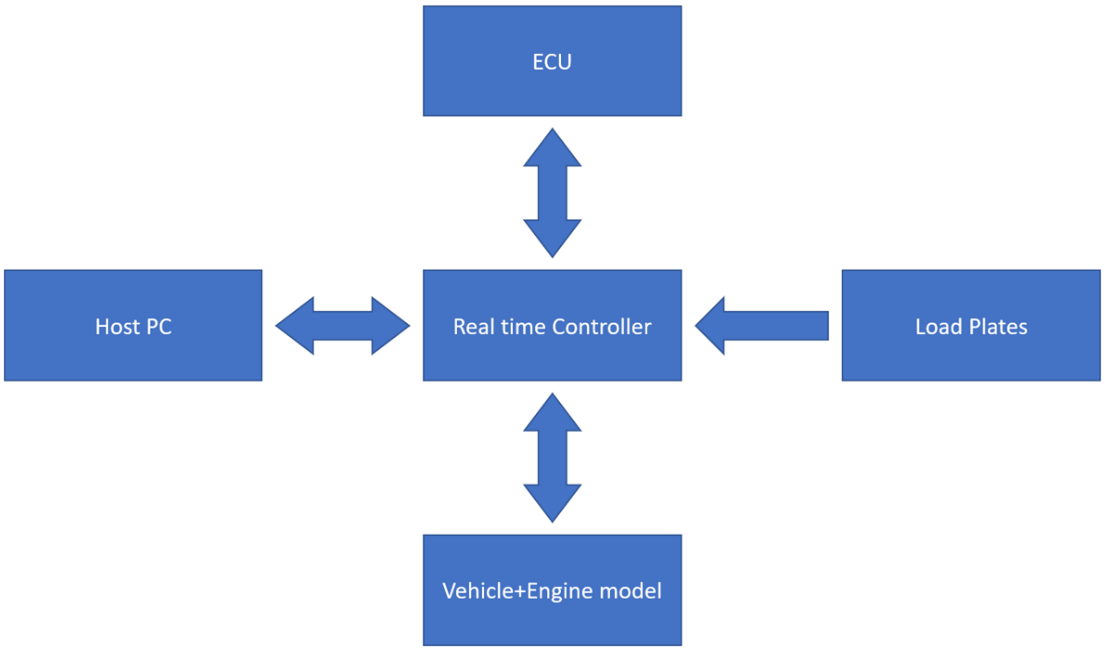


Figure 6: Hardware-in-the-Loop scheme

The next important step is to check with a more complex system if there are any problems with the logic of the software functionality. From figure 6, it is possible to see the differences between the components exploited in order to validate the software by using SiL or MiL, and the components used with the HiL: the software is installed inside the ECU, and the external condition are simulated using a load plates. The core of the system is the real-time controller, because it is able to guarantee a real-time simulation. The computing power is very important, because, if the system were to check only some fault of the ECU, a high computing power is not necessary, while if a simulation of engine and vehicle is requested, then, it is necessary to increase the computing power of the HiL.
The load plates is a dedicated section where it is possible to install real hardware to provide a more realistic functionality of the simulator. The component that are installed inside the hardware are (not all of the component installed inside the hardware are present in this list):

- throttle valves;
- solenoid valves;
- spark plugs;
- accelerator pedal;
- dashboard;
- changeover switch.

The dimension and the integration in the system of the load plates are directly proportional to its complexity. However, the possibilities of the system to simulate other phenomenons are

high, so the composition of the load plates depends by the choices of the company and its budget.
This is a Master & Slave system, so a periphery is present and it can control the bus of the entire system: in the time when a command is sent to a particular unit, other units, or the same unit, can send a feedback about the output. In this particular case, the host PC programs the control unit with the wanted software (the software can be upgraded any time it is considered necessary: for example, if there is a particular need to change the software because of a little upgrade, then, a new release of the software will be produced). The Scalexio (name of the hardware unit that simulate the real world conditions) can simulate engine, vehicle , environment and sends signals to ECU and to the loads, and, at the same time, it can read input given by the ECU and by the loads. So a communication between the units must be present. The system must be calibrated, so an information file is used to send data to the system. In order to accomplish this task, two dedicated software (BRC Calibration Tool and INCA) are used: the first is used to program the ECU, this is managed by the company and it provides information about the faults given by the software (e.g. injector malfunction, high voltage); INCA is used to manage the calibration (a connection with the control unit is necessary) and to take under control the signals.

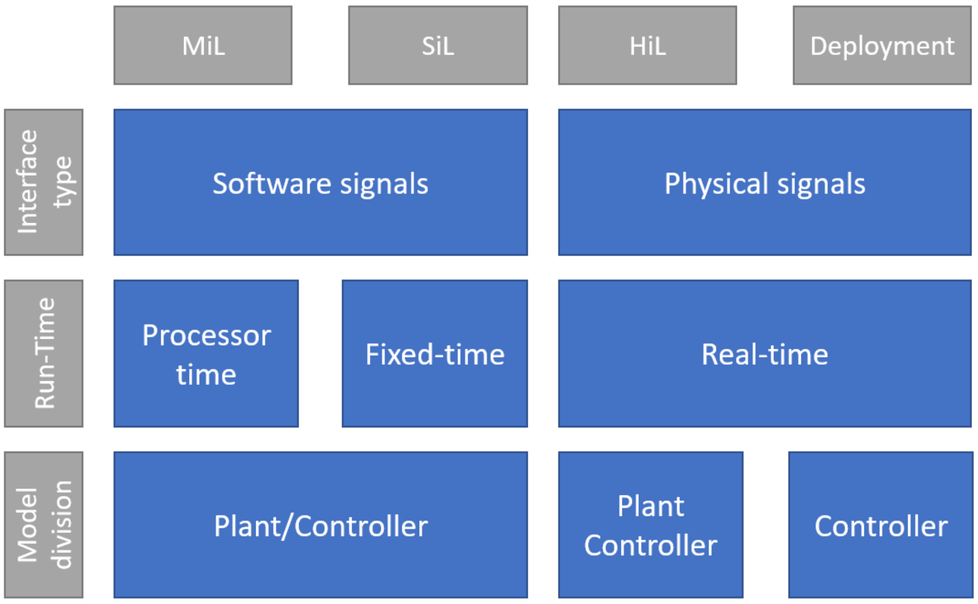


Figure 7: Model variation throughout development

In figure 7, it is possible to see the differences in terms of run time and hardware type between the different levels of validation through the development cycle. The progressive alteration of the model to meet the changing requirements at each stage of development is a core concern in system modeling since it impacts developments overhead, re-usability and model fidelity. To be more clear, HiL and vehicle test can be more precise respect to MiL and SiL because of the fact that the input signal can be considered valid or even real. Then, the other important difference between these is the run-time: now, a real-time is used.
From figure 7, it is also possible to confirm what it was stated before for what concern the physical components of MiL and SiL, in fact, the only difference between them is how they are simulated: the run-time for MiL is the processor time, while for SiL is fixed. For what concern the HiL and the Deployment (vehicle test), it is necessary to use the real time as run-

time and the inputs must be provided from a physical source. The only difference between the two systems of test is just the hardware and how the input signals can be managed: in the vehicle,e.g. it is not possible to control perfectly the speed or the load as it is desired.

3.2 Vehicle tests

The vehicle test is the last step to be covered in order to validate a software, but, in terms of cost and time, is the worst because it has the highest level of correlation with reality; this statement was explained correctly in a more efficient way in the introduction of the third chapter of the thesis, in fact, the vehicle test stays on the farthest part of the scheme in figure 3. Some tests were conducted on a vehicle, but they are not reported in the thesis because the main topic is the automation test. In order to accomplish the test requirements, a real vehicle Apè was used as testing unit. In the vehicle an OBD-2 unit was equipped in order to take under control the correct functionality of the catalytic converter and of the emissions produced by the engine. The operating conditions of the test depend by the software function that is wanted to test: for example, the Pre-Catalyst monitor could be checked in idle or during acceleration condition, instead, the Misfire monitor must be checked only during acceleration condition. The connection with the control unit is the same, so in terms of components and their connection there are not differences with the HiL, the only difference is that the load plates is substituted with the vehicle itself and the model to simulate the vehicle with the engine is not more necessary because there is not more the need to do a simulation but it is just a record of what is happening in the real situation.

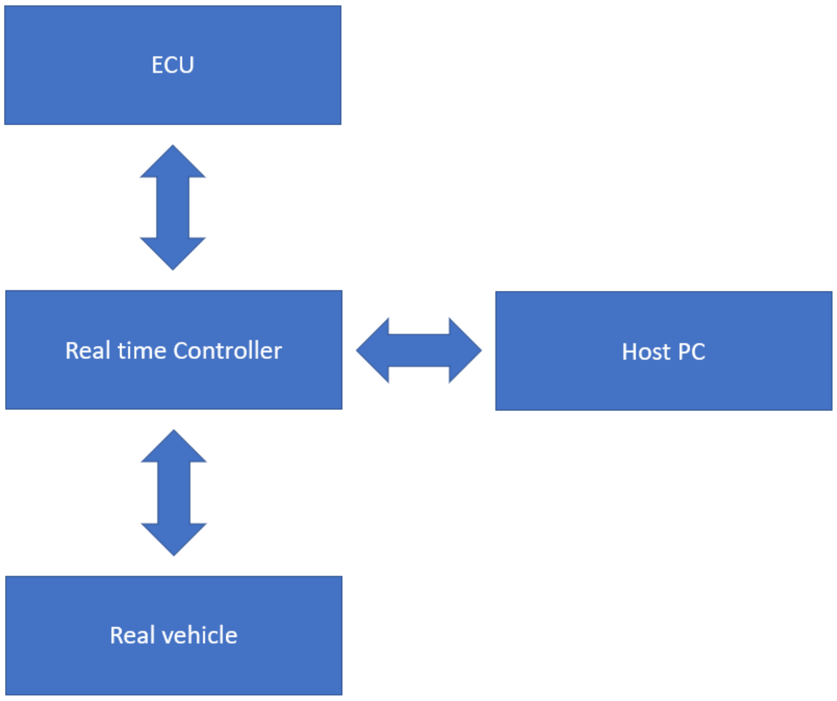


Figure 8: Vehicle test scheme

In figure 8, it is possible to see a representation of the scheme for the vehicle test: the inputs are produced at vehicle level. The misfire is a particular condition that could happen

inside the engine, but this depends by specific operating conditions. In order to avoid this particular problem and if there is the necessity to create tests that imply a misfire event, there is the possibility to create it manually from the vehicle, exploiting a particular function that can create a systematic misfire with a loss of torque and engine speed. In opposition, the oxygen monitor strategy can be tested easily because of the fact it has to check only the correct functioning of the catalyst. This means that the same manual test that were used on HiL, can be used for the vehicle test to check any software requirements.
For the thesis activity, the vehicle tests were not essential, but they could add more precision in terms of validation for the two software functionalities. The next chapter introduce the innovative part about the automatic tests describing the software functionalities that were implemented in the control unit from the physical point of view and then from the logical point of view.

4 Software functions

In order to validate a software, it is not possible to consider an ECU based only on a single software functionality, but it is necessary to integrate the so-called "Big Model": it is the merge of multiple functionality in which some inputs are produced from the BSWL and they are used in order to produce a result (for example, during the training activity, it was necessary to produce a command that enables the opening of the solenoid valve for the injection of alternative fuel). The input signals can be produced with different velocities and the velocity depends on the requirements (generally the speed in order to produce the result is a requirement stated on the document). For example, in order to detect the presence of a misfire in a driving cycle, the analysis must be done every 180° (cylinder event), because it is necessary to focus on the time passed during combustion phase and during intake phase. For the other software functionalities that were analyzed during the activity, a run-time of 10 ms or 100 ms was enough because the analysis of the inputs were temporal and not dependent on the cylinder event.
On the next subsections, a discussion about the single software functionalities will be done.

4.1 Pre Cat Oxygen Sensor

In order to satisfy the more stringent regulations on the pollutant emissions, during the years, a new way of conversion and control of the exhaust gases was developed.
The oxygen sensors are positioned before and after the catalyst, so it is possible to control the quantity of exhaust gases that are converted into non-harmful emissions. A communication between the different components such as injector, ECU and spark plug is always needed, so, as it is possible to check from figure 9, all the operating components are connected to create a closed-loop control, in order to check the correct quantity of fuel that must be injected to the combustion chamber every cycle.

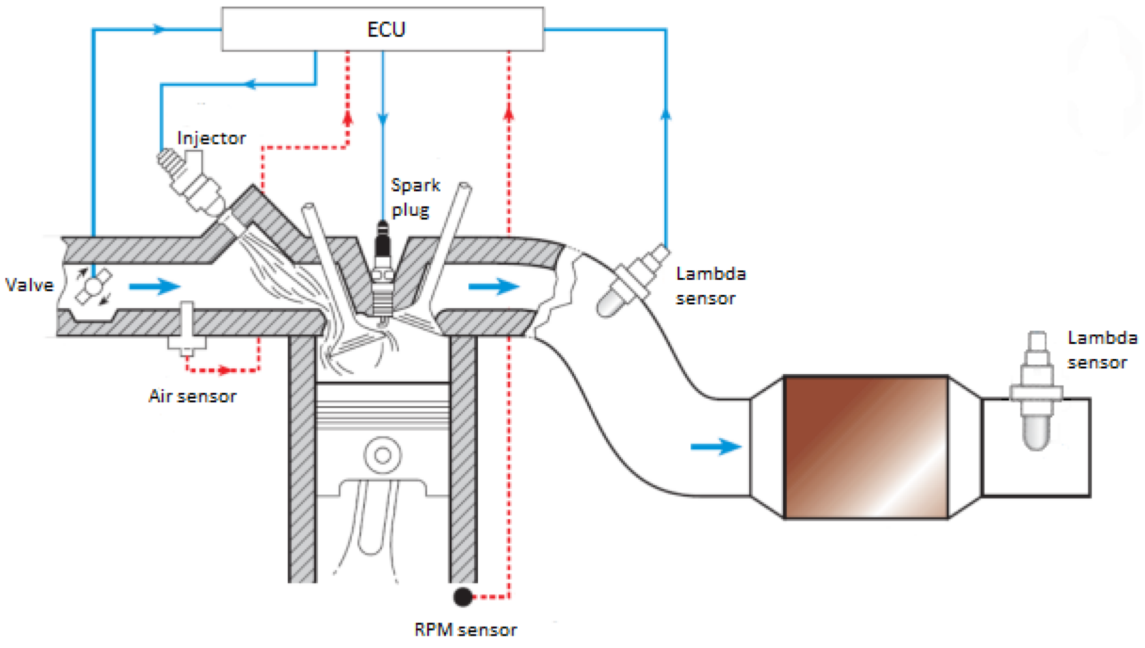


Figure 9: Closed loop control system for air/fuel ratio

In figure 9, it is possible to see the position of the two oxygen sensors; the first software

functionality is made in function of the preliminary oxygen sensor.
The closer the air/fuel ratio of the exhaust gases are to stoichiometry, the higher will be the catalyst's efficiency at converting emissions to water, nitrogen and carbon dioxide. The figure 9 shows the most common way of how it is possible to communicate and adjust the mixture of air and fuel that is going inside the combustion chamber. The first oxygen sensor is used in parallel with the air mass sensor in order to communicate to the ECU the adjustment of the mixture, while the second sensor provides the result of the catalytic converter, so there is a continuous broadcast communication within the units. Generally, the changes on the mixture are very small, so it is possible to be precise on mixture's quality. A practical example of the calculus made by the hardware Scalexio in order to correct quantity of fuel injected in the combustion chamber can be seen in figure 10.

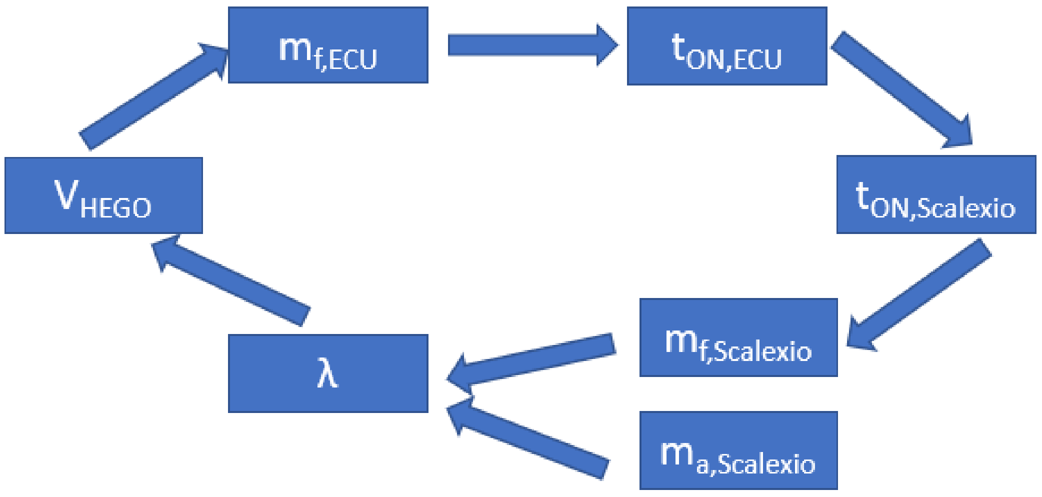


Figure 10: Scheme of closed loop control system for air/fuel ratio

As first, a quantity of fuel is evaluated by the ECU, this value corresponds to an opening time of the valve evaluated by the ECU (this value must coincide with the opening time evaluated from the hardware Scalexio). A determined value of air, dependent by the angular speed of the engine and the load, is used in order to evaluate λ as in equation (1).

$$\lambda = \frac{m_{fuel}/m_{air}}{(m_{fuel}/m_{air})_{st}} \tag{1}$$

At this value of λ, corresponds a voltage from the HEGO sensor that provides a correction to the value of the fuel injected to the combustion chamber evaluated by the ECU. The correction factor is known as KO2, and it is used in order to adjust the level of fuel in the combustion chamber: if KO2 is lower than 1, the fuel mass should be decreased because of a too rich mixture, instead, if the KO2 is higher than 1, the fuel mass generated by the ECU should be increased because of a too lean mixture.

4.1.1 Physical meaning

The oxygen sensor is a product of the evolution from the Nerst principle. The Nerst principle describes the thermochemical behaviour of a galvanic element. The two main types of oxygen

sensors are the universal exhaust gas oxygen (UEGO) sensors and the switching-type (EGO) sensors. The first can measure the ratio between the air and the fuel that is injected inside the combustion chamber, while the second can indicate to the ECU only if the mixture of air and fuel is lean or rich (so only if the ratio is higher or lower than the stoichiometric value). The problem associated with this type of sensor is based on the fact that they measure the air/fuel ratio only at the sensor's surface, instead they should measure the actual ratio of the exhaust gases.
Nowadays, there are three types of EGO sensors, and the distinction is based on material, principle of operation or configuration, and are referred to as:

- potentiometric;
- amperometric;
- resistive semiconductor.

The potentiometric EGO is widely used and is based on the Nerst cell principle, where the output varies logarithmically with the air/fuel ratio. Because of the fact that there is an abrupt response of the sensor, this is referred to as the "switch-type" EGO sensor.
The amperometric sensor, commonly referred to as wide band or broad band lambda sensor, is based on the limiting current principle, where an additional cell is used for oxygen pumping. The pumping current is a function of the oxygen concentration, like the potentiometric sensor. The advantage of this type of sensor is the wide range of measurement: in fact, it is possible to detect an air/fuel ratio range 0.7-4.
The resistive sensor is not so much used in the automotive field because of poor durability. The basic principle is a change in the semiconductor material conductivity that reacts with the change in the oxygen concentration, then, there is a direct correlation with voltage output of the signal.
The oxygen sensor is made up of a solid electrolyte that current is carried through by oxygen ions, a ceramic electrolyte separates two platinum electrodes and the difference in partial pressure between the two electrodes causes the oxygen ions to carry current between the chambers. So there is a direct relation between the difference in pressure and the output current (actually, the real output of this component is a voltage, so there is an associated resistance, and the product between resistance and current gives as result a voltage). For example, a rich mixture such that $\lambda < 1$, produces a higher output voltage due to the difference in oxygen concentrations between the two electrodes of the Nerst cell. Conversely, a value of lambda that equates to a relatively high oxygen concentration (lean mixture), creates a lower output voltage due to the smaller difference in oxygen concentration among the electrodes. The sensor is very sensible to the oxygen concentration and, as it is possible to see from the figure 11, there is a thin threshold that separates lean voltage output from rich voltage output.

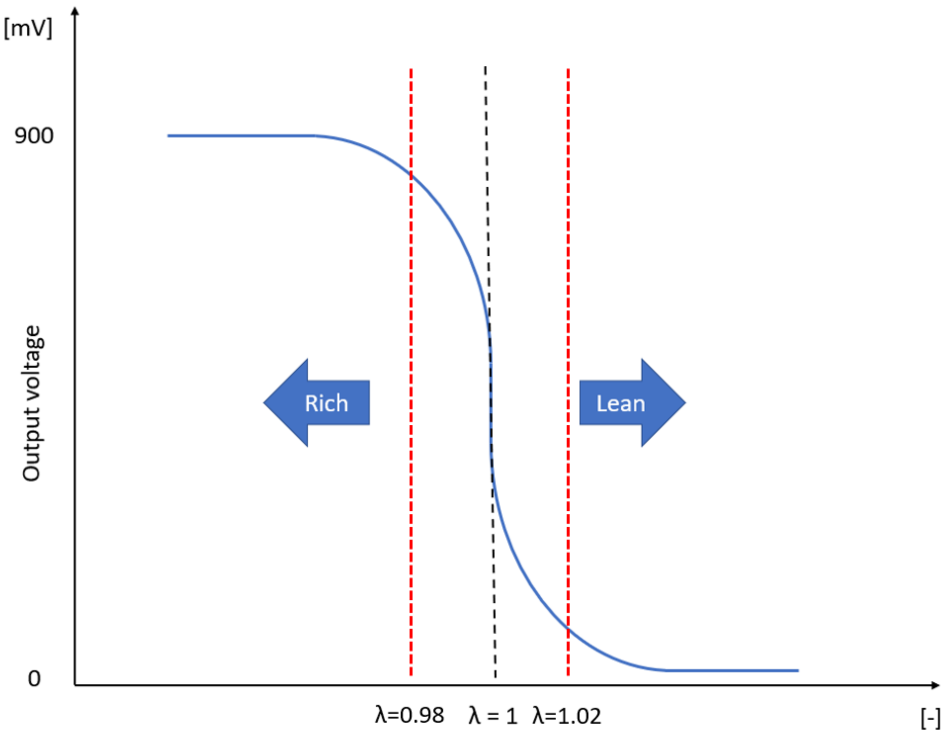


Figure 11: Typical output voltage of a lambda sensor

The output of an oxygen sensor is a voltage and the efficiency of the catalyst can be evaluated based on the result of the two oxygen sensors (pre and post): the correction of the lambda sensor made upstream of the duct are very frequent, while the correction made by a normal lambda sensor in the downstream of the duct are less frequent than the pre-catalyst sensor. The efficiency of an oxygen sensor can be assumed as the comparison between the frequency of corrections made the downstream catalyst and the upstream catalyst: if the two results are similar, the efficiency can be considered low, on the contrary, if the two results are very different with each other, the efficiency can reach high values.

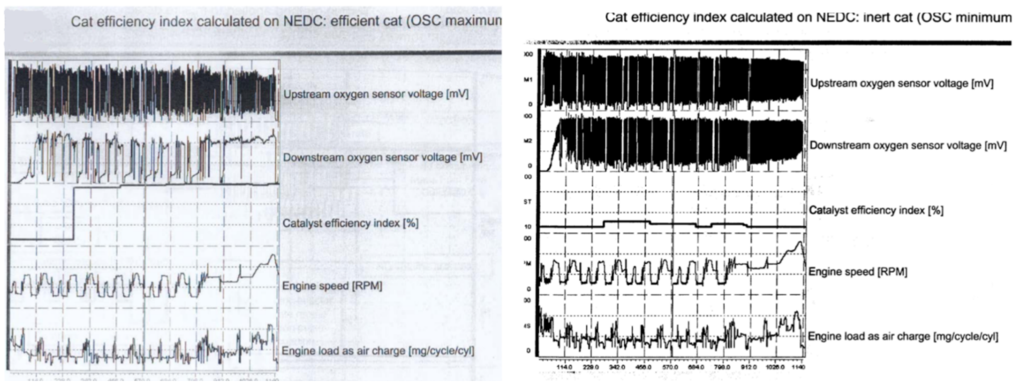


Figure 12: Catalyst efficiency monitored by oxygen sensors

In figure 12 it is possible to see a comparison between a good and a bad catalyst where the response of the lambda sensors can be analyzed: the first is not good because the post-catalyst's output is bouncing between a rich and a lean mixture.

4.1.2 Requirements for the software

This software functionality aims to detect a performance deterioration of the Pre-Catalyst oxygen sensor, in order to fulfill the OBD-2 compliances. It consists in 3 oxygen sensor signal checks:

- frequency check: the response time of an aged sensor can be deteriorated and, as consequence, the frequency at which we have a rising and a falling edge is decreasing; during a monitoring window, if the value of the frequency is under a threshold for a certain amount of time, the check will result failed;
- drift check: the response time of a sensor in rich can be different respect to the time spent in lean, and the operating condition influences these time; however, in stoichiometric conditions, it is possible to see an asymmetric behaviour; during the monitor time, the ratio between the time spent in lean and the time spent in rich is calculated and compared to a minimum and a maximum threshold, and, if the time of failure is higher than a certain percentage threshold, the check will result failed;
- offset check: the signal of an aged sensor can have a different value for the span, so in this check the difference between the maximum and minimum value of the voltage is compared to a minimum threshold, and if this is lower than it, the check will result failed.

For this software functionality, it is implemented the so-called machine state logic, in which, depending on the operating conditions, the monitor will have a different value. The different states are: "UNABLE_TO_PERFORM", "PENDING", "ACTIVE" and "COMPLETE". The first state is when the monitor is temporarily disabled because the enabling conditions are not satisfied, then, when they are satisfied, the state will pass to "PENDING", in which the monitor waits for the activation conditions (the operating conditions of the engine characterize the activation of the monitor). The last state is accomplished only when a certain time has passed. The requests in order to enable the monitor are:

- an enable command must be present;
- if particular faults are detected (e.g. injector faults, too rich system, ignition fault, O2 sensor malfunction . . .), monitor will be disabled for the current trip;
- the coolant temperature at crank must be limited by a minimum and a maximum threshold;
- if the barometric pressure is lower than a calibration, the monitor must be disabled for the current trip;
- if the level of the fuel in the storage is lower than a calibration, then the monitor is disabled for the current trip;
- the monitor will start only if is completed a warm-up cycle; it is defined considering the temperature of the cooling fluid:
  - ECT at crank must be lower than a threshold;
  - the current ECT must be greater than a minimum;
  - the ECT must be risen by a certain calibrated value;

- a certain time from the crank has to be passed in order to enable the monitor.

Once the enabling condition are satisfied, in order to activate the monitor, it is necessary to satisfied the so-called activation condition: steady state condition: the engine speed (expressed in rpm) and the Manifold Absolute Pressure (MAP, expressed in mbar) are controlled and it is necessary to compare the raw signal with the filtered signal (2):

$$|x(t) - x(t)_{filt}| \leq \delta \tag{2}$$

the monitor shall be enabled only in a region where the engine speed and the load has the lowest risk to produce a failure: this is expressed limiting the two input by a minimum and a maximum threshold; the monitor can be activated only if the lambda closed loop control is active and the engine is producing a positive torque or it is idle. Once the monitor is active, as it was stated before, a certain minimum time has to pass in order to go to “COMPLETE”. The other state that is necessary to consider, is the result of the monitor: until the state of the monitor is “COMPLETE”, the result will be “DATA_NOT_READY”, and, in base of the result, the state will be “FAIL” or “PASS”.

4.1.3 Simulink logic

In figure 13 it is possible to see an example of implementation of machine state logic in the software dedicated to check the operating functionality of the oxygen sensor. The logic is based on the Simulink block "Switch-case", that is used to define the status of the monitor and the status of its result. The monitor will pass through a different state every time some conditions are satisfied, for example, if the initial state of the monitor is "UNABLE_TO_PERFORM" and the enabling conditions are satisfied, then the state of the monitor will become "PENDING". So it is possible to recognize an evolution of the monitor through these states.

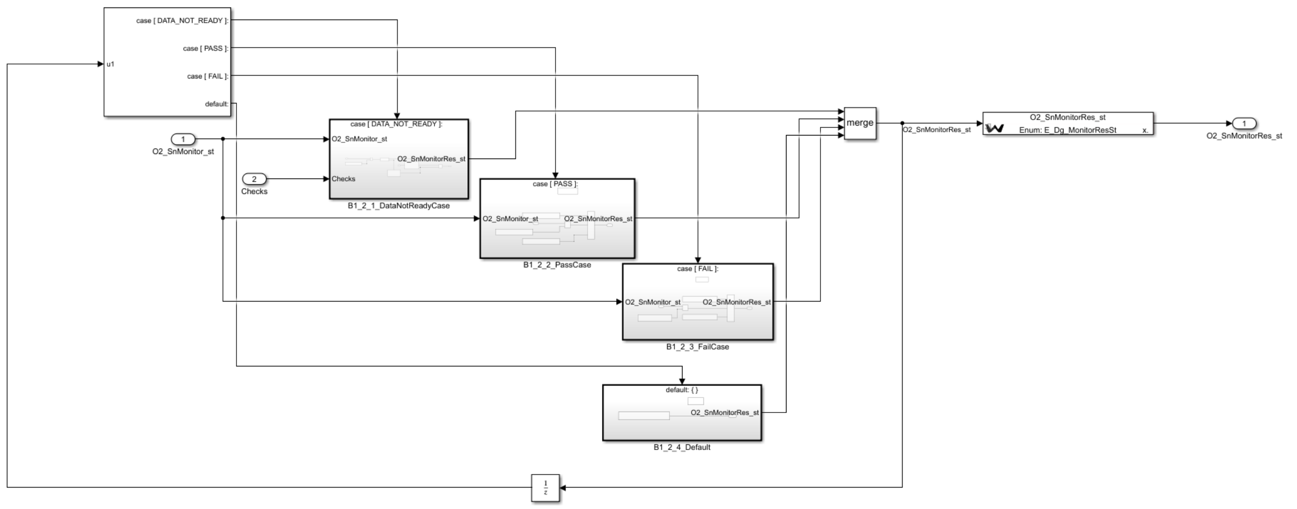


Figure 13: Example of a machine state for the result of O2 monitor

In figure 14 and figure 15, enabling conditions and activation conditions Simulink blocks are presented: they are very similar for what concerns the logic of implementation, in fact, each of them is based on an "AND" block. The common block is a boolean block, so the output can be only 1 or 0, if the conditions are satisfied, the output will be positive, otherwise it will be negative (the description of the conditions to be accomplished is presented in chapter 4.1.2).

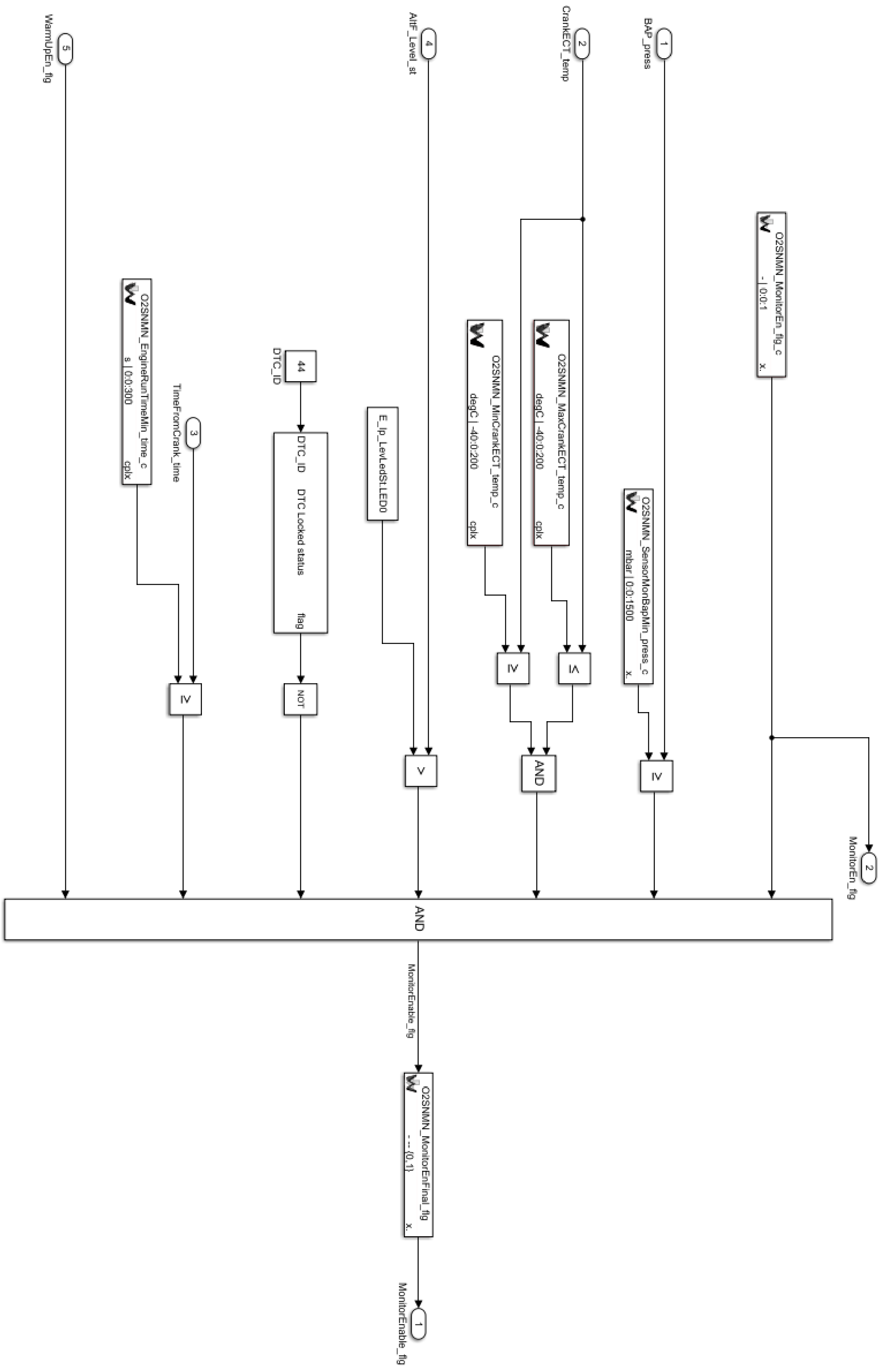

1 BAP_press
2 CrankECT_temp
4 AltF_Level_st
5 WarmUpEn_flg
3 TimeFromCrank_time
O2SNMN_MonitorEn_flg_c - | 0:0:1 x.
O2SNMN_SensorMonBapMin_press_c mbar | 0:0:1500 x.
O2SNMN_MaxCrankECT_temp_c degC | -40:0:200 cplx
O2SNMN_MinCrankECT_temp_c degC | -40:0:200 cplx
E_lp_LevLedSt.LED0
44
DTC_ID
DTC_ID DTC Locked status flag
O2SNMN_EngineRunTimeMin_time_c s | 0:0:300 cplx
NOT
AND
AND
2 MonitorEn_flg
MonitorEnable_flg
O2SNMN_MonitorEnFinal_flg - -- {0,1} x.
1 MonitorEnable_flg

Figure 14: Enabling condition for O2 monitor

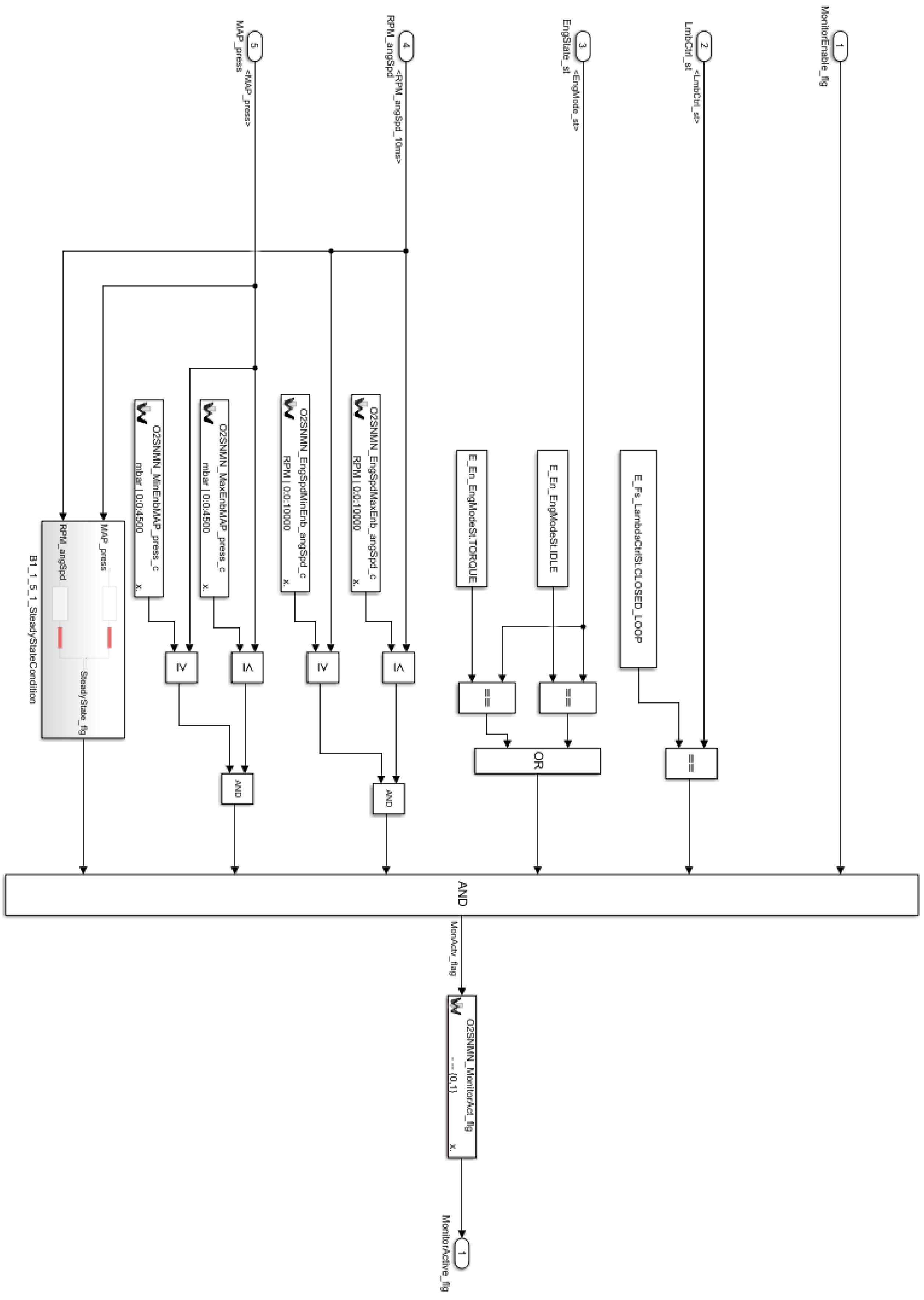


Figure 15: Activation condition for O2 monitor

In the figures 16, 17 and 18, it is possible to recognize the representation of the checks used in order to control the correct functioning of the lambda sensor.

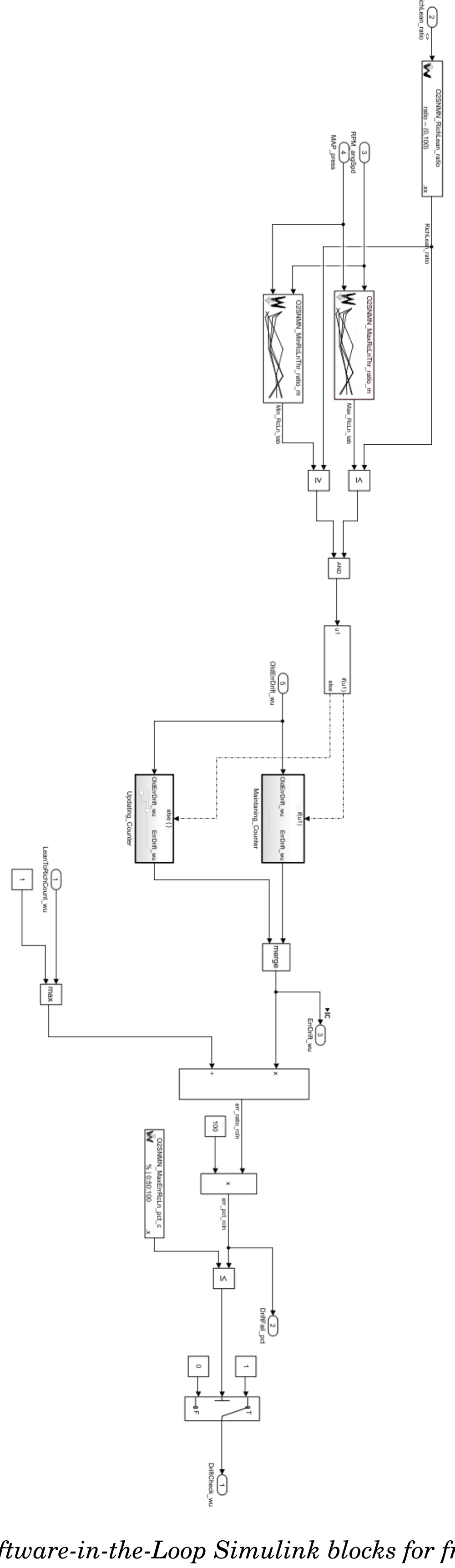


Figure 16: Software-in-the-Loop Simulink blocks for frequency check

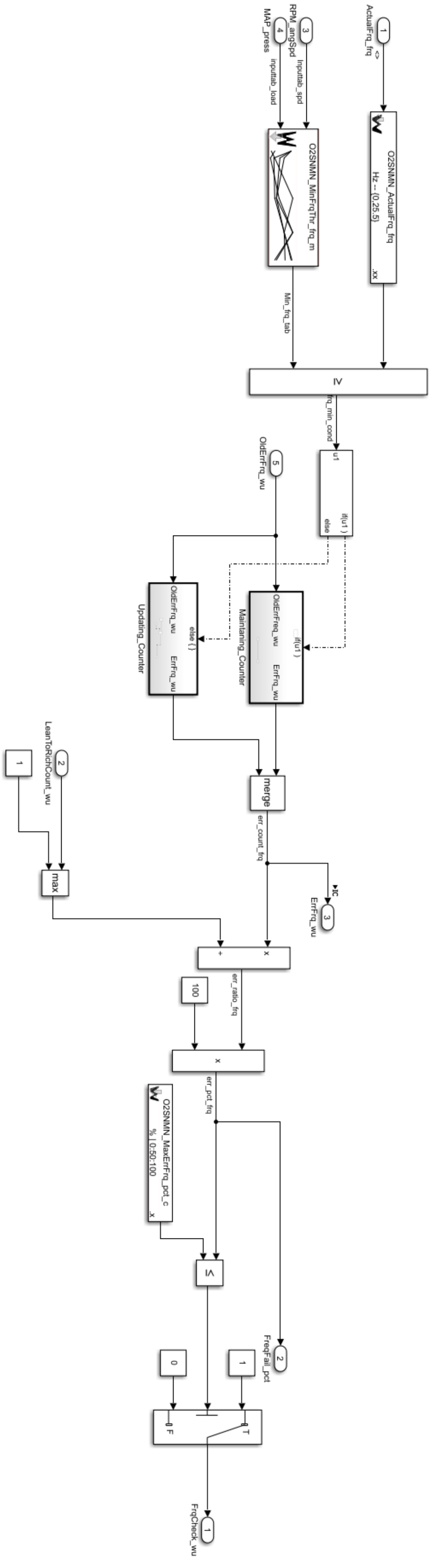


Figure 17: Software-in-the-Loop Simulink blocks for drift check

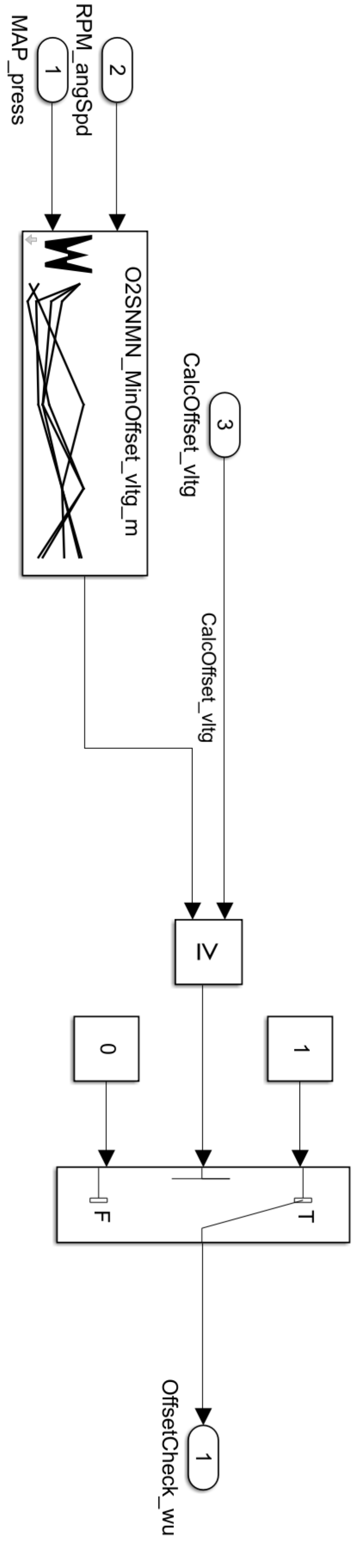


Figure 18: Software-in-the-Loop Simulink blocks for offset check

As it was discussed in the second chapter for the software requirement analysis, all the checks are analyzed during the monitoring window, and if one of these three checks is "FAIL", it means that there was a fault in that window. The system is realised in order to choose even what of the checks can be enabled, so, just changing some parameters in the calibration file, it is possible to select what type of controls must be done in that window. This conclude the first part of the fourth chapter; in the next part, a similar discussion about the second software functionality implemented inside the "big model" is present.

4.2 Misfire detection

For a typical engine cycle, it is possible to distinguish two important phases(referencing to figure 19: intake and combustion. During the intake phase, a preparation for the combustion is present, so, inside the combustion chamber, particular operating conditions that favours the combustion are generated. The thesis is based on the analysis of a Spark Ignition engine, so the ignition is created through a spark plug that ignites the mixture and creates the so-called flash of a combustion.
When the ignition is ready, the fuel is already vaporized due to the high level of pressure that is generated in the combustion chamber and mixed with the air already inside and measured in function of the operating condition. From the spark, it is possible to recognize the core's flame, where the flame is begun; then, the flame expands gradually creating the so-called "flame front", that is an irregular surface that separates the combusted mixture from the uncombusted mixture. This process ends with the total consumption of the mixture in the combustion chamber, without any waste or uncombusted mixture, so the development of energy of the mixture that is burning is progressive and regular.
Above, a description of a normal combustion is present, but, it is possible that the combustion can be abnormal (not perfect combustion) because there are so many factors that influence it: for example, turbulence, temperature, pressure, air/fuel ratio, presence of previous exhaust gases and more other. So, there are multiple factors that can create an abnormal combustion.
One of the most dangerous and common problems that can be present during an engine cycle, is the misfire. On the next chapters, a discussion about the detection of this phenomenon is present.

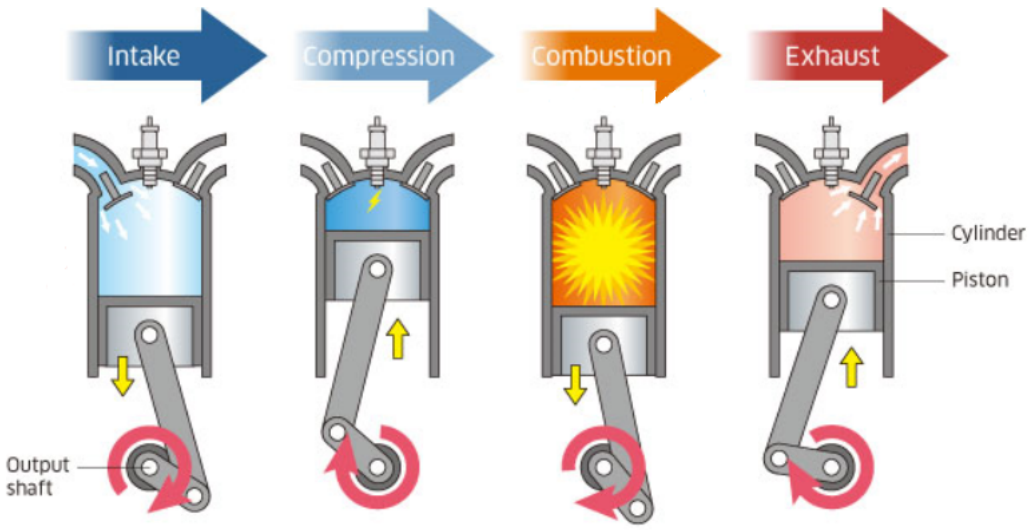


Figure 19: Phases of a ICE with 4 strokes

4.2.1 Physical meaning

The misfire is the loss of combustion in a cylinder of a spark ignition engine, this can be due to lack of spark, fault on the mixture level, poor decompression or other faults linked to injection or spark plug. This is an unwanted phenomenon because it produces a null or lower torque, and it is dangerous for the life of the catalyst. The mixture, in presence of a misfire, can reach the catalyst and the oxidation of the mixture could produce an increase of the temperature with serious problems on catalyst's life material. Another consequence related to the misfire, is the increase of pollutant emissions because the performances of the catalyst are worse than

the original situation, so there is a poor conversion efficiency of the catalyst. The abnormal increase of pollutant emissions is the reason why it is very important to detect the presence of misfire during the engine cycle.
Generally, it is possible to recognize three modalities of lack of torque due to faults related to injection or ignition:

- slow-burn cycles: the loss of torque is not so much high, but it can not be negligible, and it can be accounted with a range of 15-55%;
- partial-burn cycles: the loss of torque is higher, and generally is higher than 55% of the medium torque;
- misfiring cycles: the torque is negative or zero.

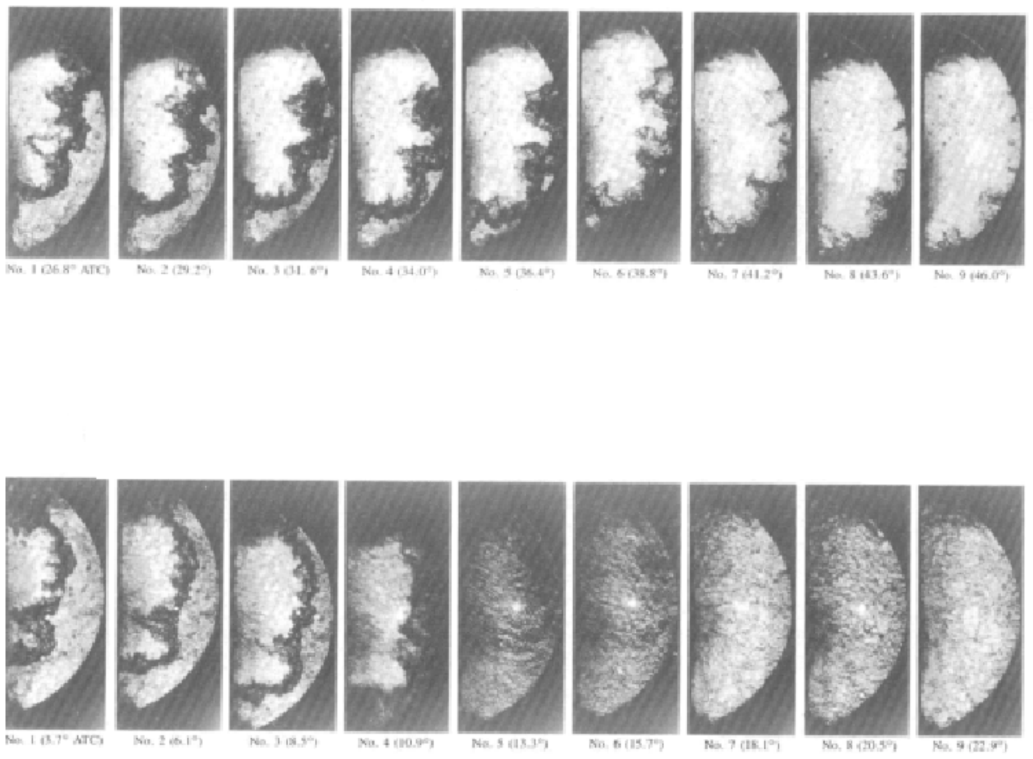


Figure 20: Photographic sets of the combustion's evolution

In figure 20, it is possible to see the evolution of an engine cycle when the misfire is not present and when it is present. The first set of figures shows that the light is spread among the cylinder, so the combustion has not problem, while the other set, shows that the light is not spreading after the spark ignition, so a misfire is present and the mixture can not be burned in the right way.
The OBD-2 systems must detect a misfire in two main cases:

- misfire damages the catalyst: for any operating conditions, it is necessary to detect the percentage of misfire over 200 cycles that increases the temperature and could damage the catalytic converter;
- misfire increases the pollutant emissions by 50% more than the regulations: it is necessary to detect the percentage of misfire over 1000 cycles that causes an increase of the pollutant emission by 1.5 times the ordinary condition.

If one of these conditions are satisfied, the MIL turns on. The same system, in case of a single misfire, has to be able to detect the cylinder where the misfire happens, while, in a multiple case of misfire, it is not important to detect the cylinders. Following the OBD-2 regulations, the monitor must be active from the second cycle, in order to avoid the crank condition, and during every situation of positive torque (no cut-off, idle or any other conditions).
In order to detect the misfire, there are different solutions: nowadays, the most diffused technique is based on the analysis of the signal generated by the tone wheel. The tone wheel is a toothed wheel made by 24-2 or 60-2 teeth, that detects the velocity of the engine: if the velocity is reduced in an unexpected way, it is possible that a misfire was present, due to the fact that it generates a decrease of the useful torque. This system does not require a particular hardware and it is well suited for a low cylinders engine, while, the engine with a high number of cylinders, especially with low velocity and high load, could have more difficulties for what concern the detection of the engine speed variation.
Another similar way to detect the presence of misfire, is based on an acoustic analysis where the vibrations generated during the functioning can be analyzed in order to find a misfire in the driving cycle.
Another different way to detect a misfire is based on the ionization current on the electrodes of the spark plug: monitoring this current, it is possible to detect the presence of the misfire. This system has the advantage based on the fact it is not related to the tone wheel's signal, but additional hardware is required.

4.2.2 Requirements for the software

This functionality aims to detect the misfire events and to evaluate their impact on the emissions, in order to satisfy the OBD-2 compliances. The main input used by the monitor is the crankshaft speed sensor signal. In fact, it is used by the ECU in order to detect the flywheel's teeth. As it was introduced before, the engine produces positive work only during the expansion stroke, while on the other 3 strokes, the engine produces negative work. From the physical point of view, the engine speed is increased only during the expansion stroke, while during the other 3 strokes, the engine speed will decrease, so, this can be simplified stating that there are two different types of revolution: the fast revolution and the slow revolution. The important condition in which the monitor can be active is the fact that the engine must be in "torque condition", so the engine should not be in idle, or in cut off or in another condition, otherwise, the misfire can be associated with another factor.
The core of the analysis is the measurement of the time passed between two calibrated teeth in the expansion stroke and in the combustion stroke. The window in which the time can be evaluated is the same, so it is necessary to evaluate the ratio between these two times, and if this is under a certain threshold then, there is the presence of a misfire. The evaluation of the monitor can be done only in specific conditions in order to obtain reliable results.
In this case, it is necessary to consider a cylinder-event ruster, due to the fact that the informations are gathered every revolution of the piston.
(inserire l'analisi della valutazione dei tempi in base ai denti) For this software functionality, a similar approach to the Pre-Catalyst Oxygen sensor is done: the machine state logic is used, so, it is possible to identify four states that describe the state of the monitor ("UNABLE_TO_PERFORM", "PENDING", "ACTIVE" and "COMPLETE", as the previous software functionality). Even in this case, some conditions are necessary to pass from a state to another one.
The enabling conditions in order to pass from the state "UNABLE_TO_PERFORM" (monitor temporarily disabled) to "PENDING" (monitor waiting for the activation conditions) are:

- an enable command shall be present;
- if particular faults are detected (e.g. injector faults, too rich system, ignition fault, O2 sensor malfunction ...), monitor will be disabled for the current trip;
- the coolant temperature at crank must be limited by a minimum and a maximum threshold;
- if the barometric pressure is lower than a calibration, the monitor must be disabled for the current trip;
- if the level of the fuel in the storage is lower than a calibration, then the monitor is disabled for the current trip;
- the monitor will start only if is completed a warm-up cycle; it is defined considering the temperature of the cooling fluid:
  - ECT at crank must be lower than a threshold;
  - the current ECT must be greater than a minimum;
  - the ECT must be risen by a certain calibrated value.
- a flag in order to overpass all the other enabling conditions shall be present (for debug use).

The activation conditions in order to activate the monitor are:

- steady state condition: the engine speed (expressed in rpm) and the Manifold Absolute Pressure (MAP, expressed in mbar) are controlled and it is necessary to compare the raw signal with the filtered signal (1);
- a certain time from the crank has to be passed in order to activate the monitor;
- the engine should work only in during the torque condition;
- a specific range of the temperature of the coolant fluid must be present in order to evaluate the monitor;
- a specific low risk map shall be present in order to detect the correct fault: two different arrays discriminate the minimum and maximum load (MAP), in function of the engine speed, and it is also necessary to define a minimum value and a maximum value of the engine speed to activate the monitor.

From the moment when the monitor is active, a cycle counter is increased, when this counter reaches a minimum calibrated threshold, the state of the monitor will become "COMPLETE". During the monitoring window, a ratio between the time measured during the expansion stroke and the combustion stroke of the same engine cycle is done. If this ratio is below a threshold, then the combustion does not occured for the cycle (3).

$$\frac{TimeTeeth_{intake}}{TimeTeeth_{combustion}} \leq Ratio_{thr}(MAP;speed) \tag{3}$$

During the monitoring window, every time a misfire occurs, a counter increases; if the counter exceeds a maximum threshold, then a DTC is activated.
For every driving cycle, it is necessary to post two outputs: the first is the counter of engine

cycles where a misfire is occured in the actual driving cycle, while the second is the Exponential Weighted Moving Average for the previous driving cycles (4):

$$EWMA = 0,1 * (MisCount_{actual}) + 0,9 * (MisCount_{prev,avg}) \tag{4}$$

4.2.3 Simulink logic

The same machine state logic that was implemented for the lambda sensor, can be found in this software functionality. In order to be more coherent, the adopted nomenclature for the first functionality, is used for the second one.
In figure 21 and figure 22 can be found the representation of the enabling condition and the activation condition for the misfire monitor. As before for the oxygen sensor, the two groups of conditions converge in one "AND" block, so the result of the operating conditions for the ICE can have as output 0 or 1.
For what concern the check and the final result of the test, it is necessary just to analyze the ratio between the so called "fast cycle" and the "slow cycle": if this ratio is below a predetermined threshold, the final result of the test will be "FAIL".

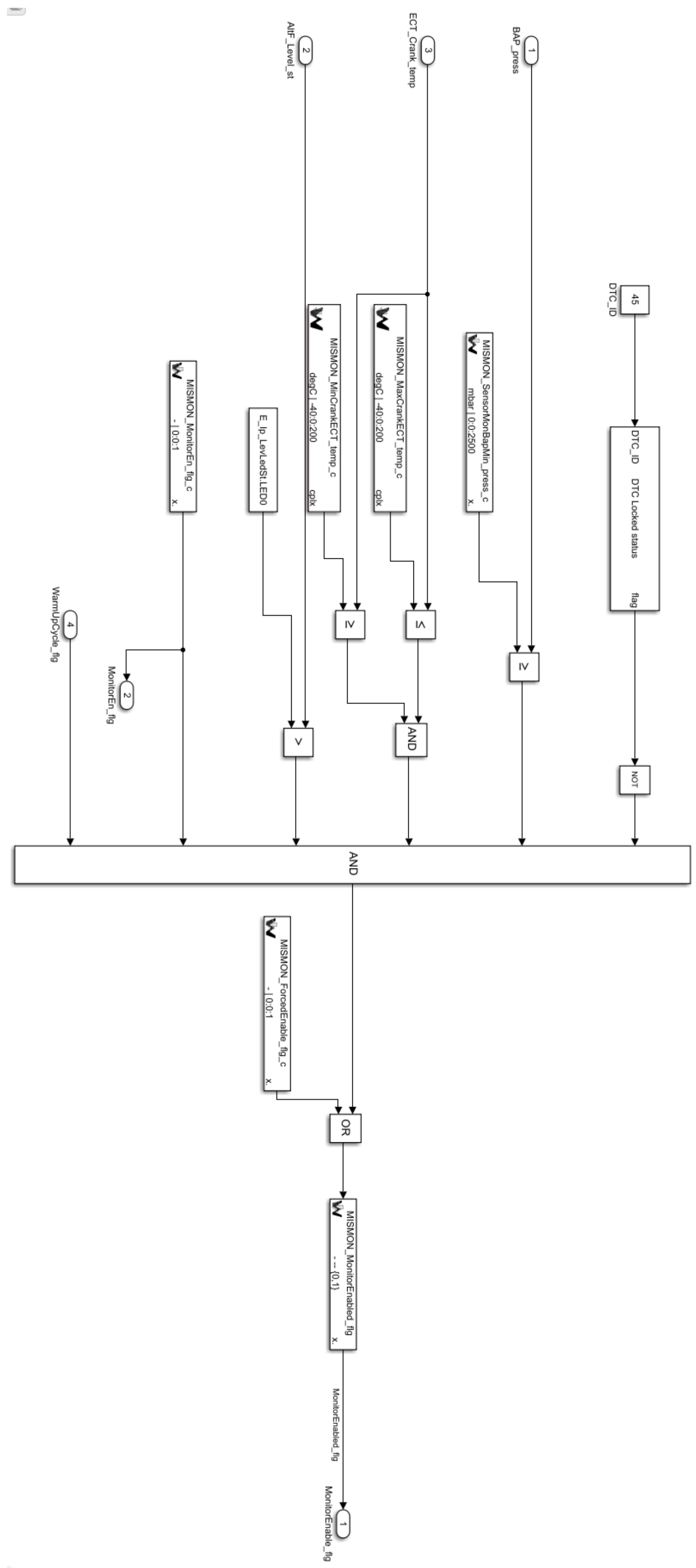


Figure 21: Simulink blocks for enabling conditions for Misfire monitor

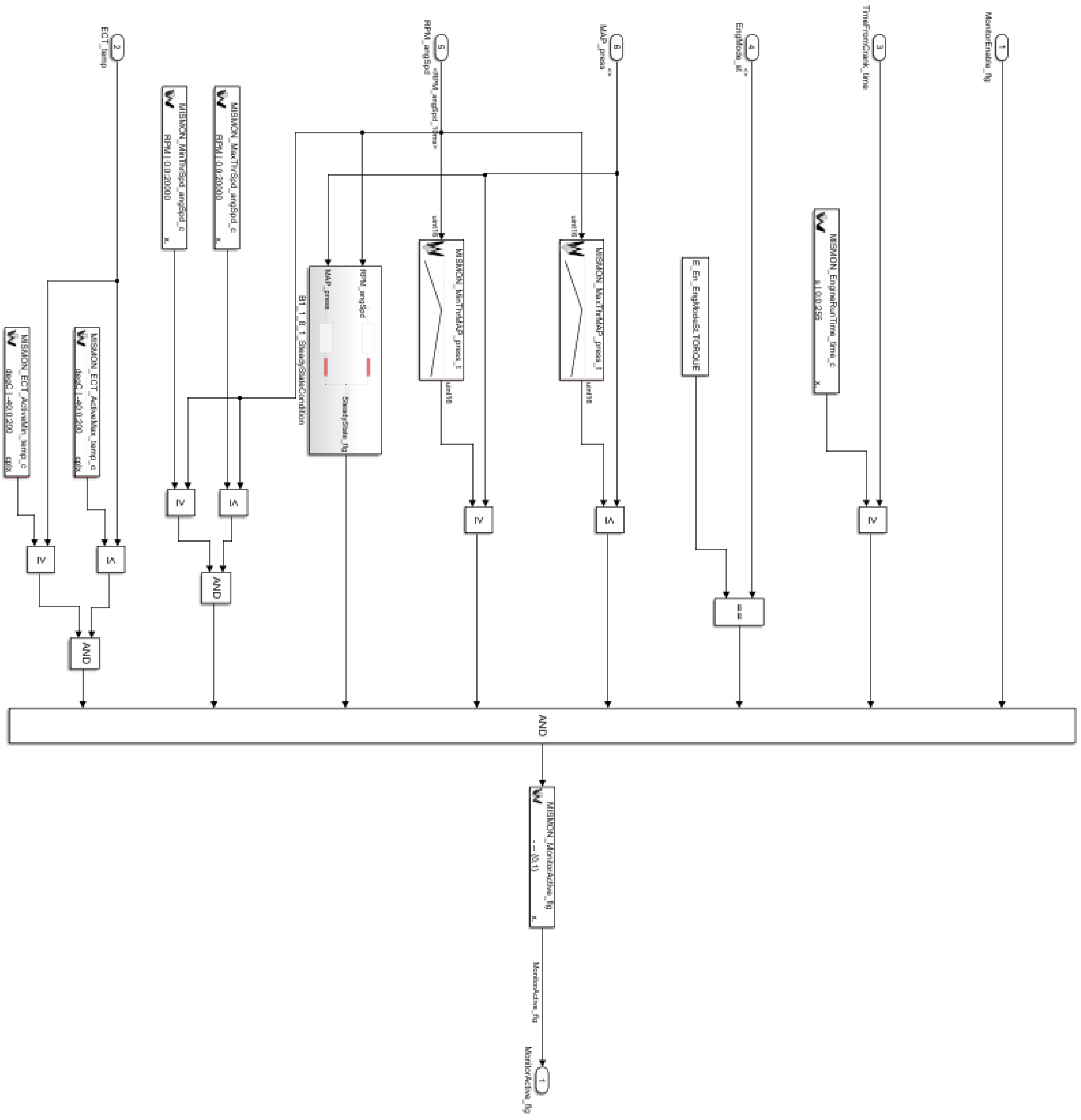


Figure 22: Simulink blocks for activation conditions for Misfire monitor

In figure 23, it is possible to recognize the Simulink blocks used in order to check if a misfire, during the driving cycle was present or not: the logic implies a control based on a percentage of fails during a predetermined number of engine cycles, if the percentage of fails is higher respect to a predetermined value, then the result coming from the monitor will be "FAIL".

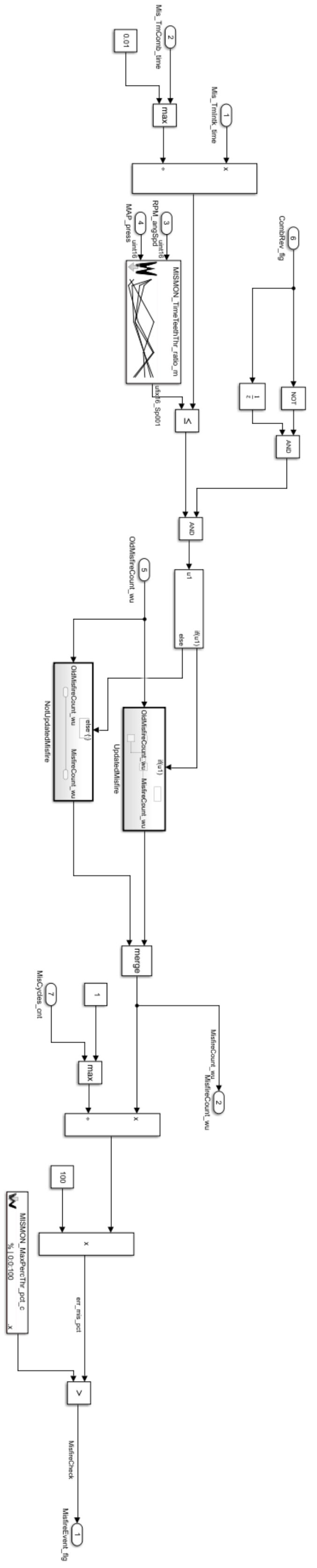


Figure 23: Simulink blocks for Misfire check

In the next chapter, a discussion about the types of automatic that could be done in order to validate a software is made. It will introduce the results and the discussion about the test done during the training activity in Westport.

5 Automatic tests

This chapter is dedicated to the description of the innovative part of the thesis: automatic test. The activity was done in parallel with Capgemini, where they prepared a particular sequence of tests as an example. The first type of test is the so-called static test, in which a particular stated sequence of actions is made in order to acquire inputs and compare outputs with the planned one. The second type of test is based on the acquisition of a particular signal (signal based test or dynamic test), and then, the output signal is compared to the planned one. The automatic tests are used in order to be more precise than the manual test, in fact, it is possible to build a particular sequence of inputs in a precise way, in order to obtain the wanted result.
The software used for this purpose is Automation Desk: it can use the information from Control Desk (software developed by the same supplier) and information from the calibration file (INCA). The output of the test can be a report that can be customized in order to be more clean and more organized. In order to be customized, python language is used.
The automatic tests can be very useful because they can be re-used for any test, they can save time and they can be very precise in terms of scheduling.

5.1 Sequence based test

In this chapter, a first description of the automatic tests executed with a sequence of actions is done. The test made by Capgemini was a sequence of action that aims to do the so-called static test. A static test is a sequence of operating points that the engine must cover in a rigid way.

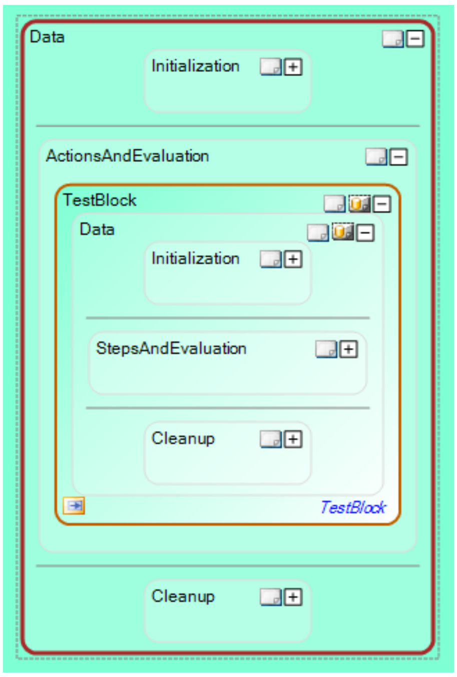


Figure 24: General structure of a static automatic test

In figure 24, it is possible to see the sequence from the graphical point of view. So, the sequence of points for every variable is stated at priori, then, it is used inside the sequence to cover the path. The sequence can be organized in different macro areas: the first can be considered as the initialization, in which all the variables are initialized to a stated value; then, the next step is defined as “steps and evaluation”, where the operating points that were stated before, are covered by the engine, and the results are evaluated comparing them with the expected results; then a cleanup is used in order to restore the system at its original status. For this case, a simple sequence of points is used, but the internal sequence of operations can be changed in function of the needs of the user. In order to validate the O2 monitor, a dedicated sequence is used.

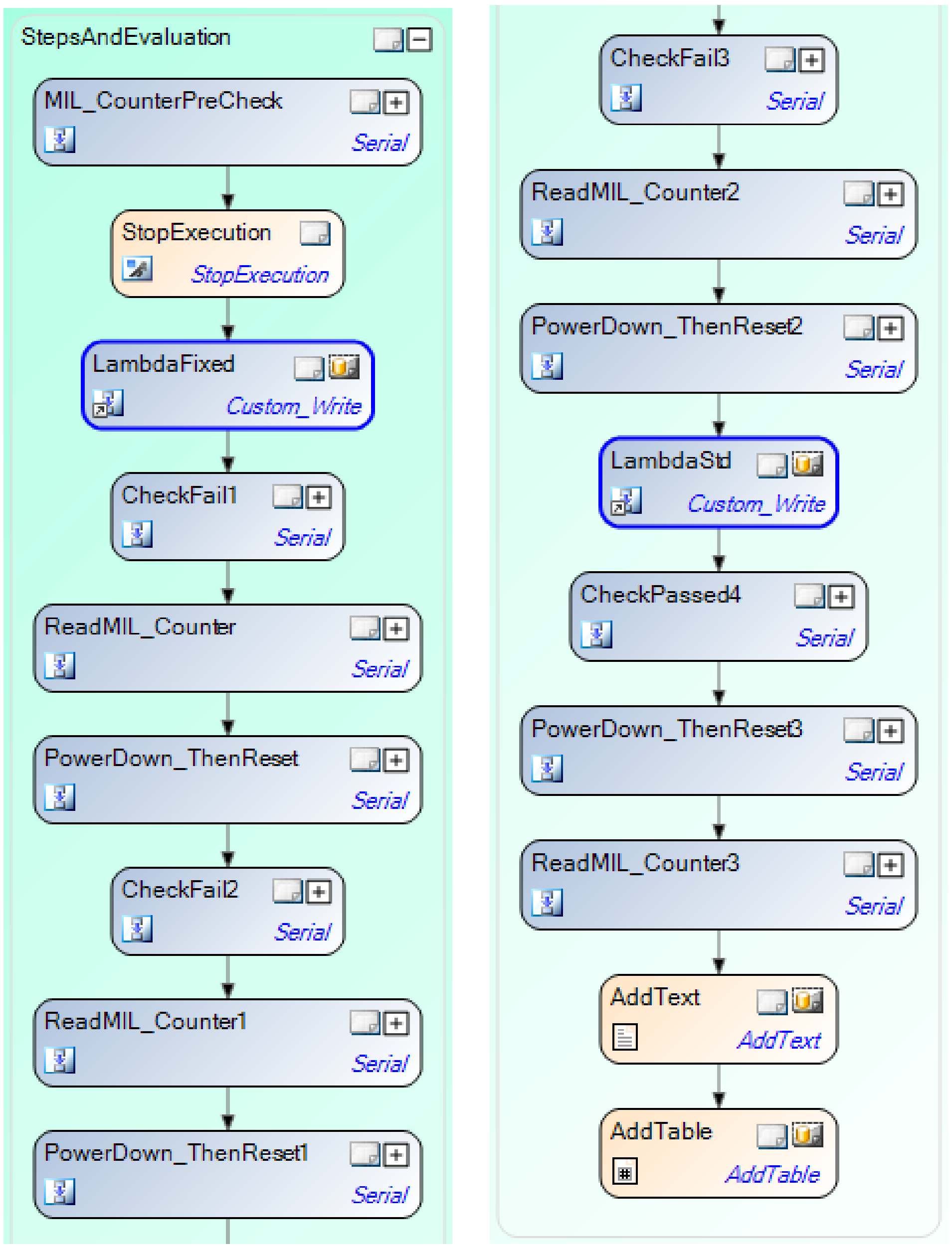


Figure 25: General structure of an ad-hoc automatic test

The aim was to validate the highest number of software requirements that are possible to validate. For this particular case, just one test is presented in order to describe the steps that are covered. As usual, an initialization step is covered in order to define the start of the sequence, then a sequence of actions and evaluation is done, and a cleanup concludes the test. In particular, in the evaluation, referring to chapter 4.2.2 where the software requirements are defined, every state of the monitor is evaluated, so it is necessary to create the operating conditions in order to accomplish the enabling conditions and the activation conditions. The next step is to control the deactivation of the control unit and the state of the DTC (this controls the activation of the MIL). The test is created in order to have a fail or a pass when it is needed, so it is possible to control the state of the DTC in any condition.

5.2 Signal based test

The aim of the signal based test is common with the sequence based test: to validate a software. The inputs provided to the system are prevently prepared using an acquisition from INCA software (.mf4 file). This type of approach can be better for what concerns the reality of the inputs because there is real-time management of the signals with a precision of milliseconds, while the other kind of tests have not the certainty of the timing, because of the host communication. The signal based test has some disadvantages in terms of interaction with the system: it is possible just to change the variable around the control unit, so the internal variable of the ECU can not be changed.
This type of the test is based on a sequence where the test can be prepared (pre-actions where the inputs are set to the initial value) and a part where the test begins and the outputs are controlled. The types of signals are: capture, stimulate and trigger. The trigger is a signal that defines the start and the end of the test. The "stimulate signals" are just signals that are associated with the input from ControlDesk. The "capture signals" are the signals that are controlled and compared with a reference value. So based on these, it is possible to decide which are the inputs and which are the outputs that must be checked.

In the following all the signals of the Dynamic Test Case:

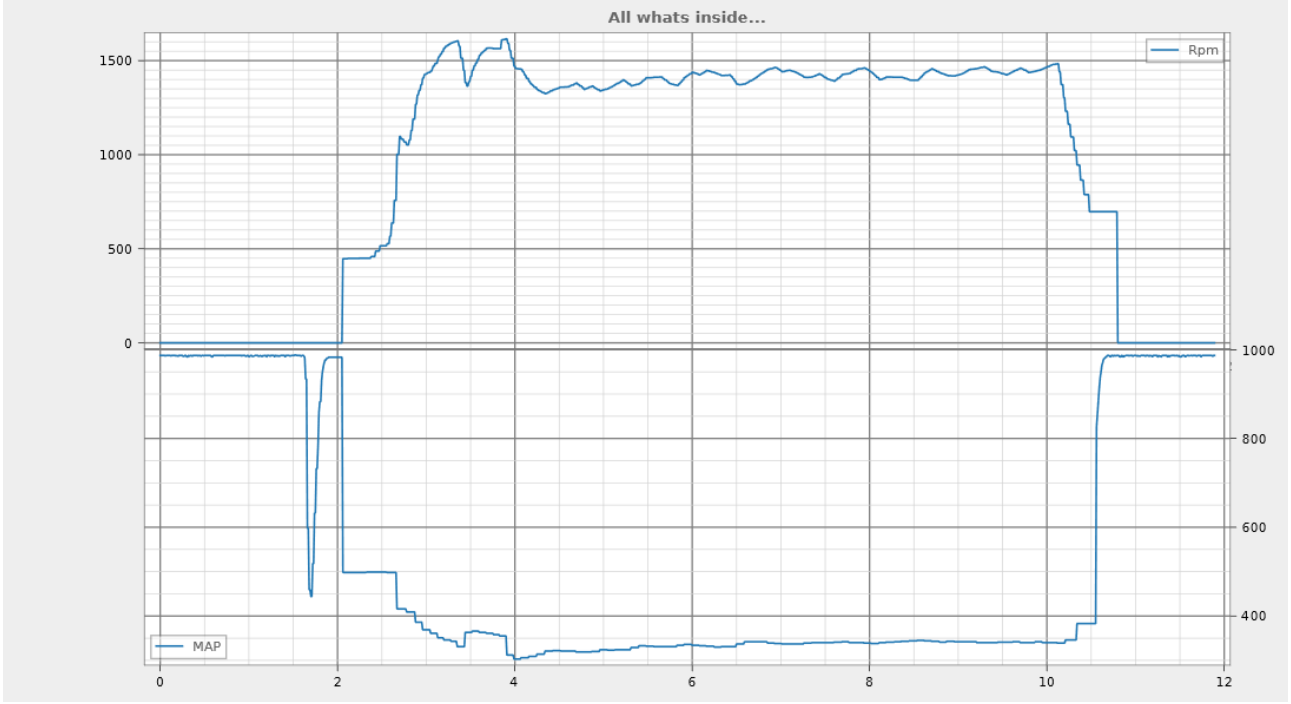


Figure 26: Inputs for the dynamic test

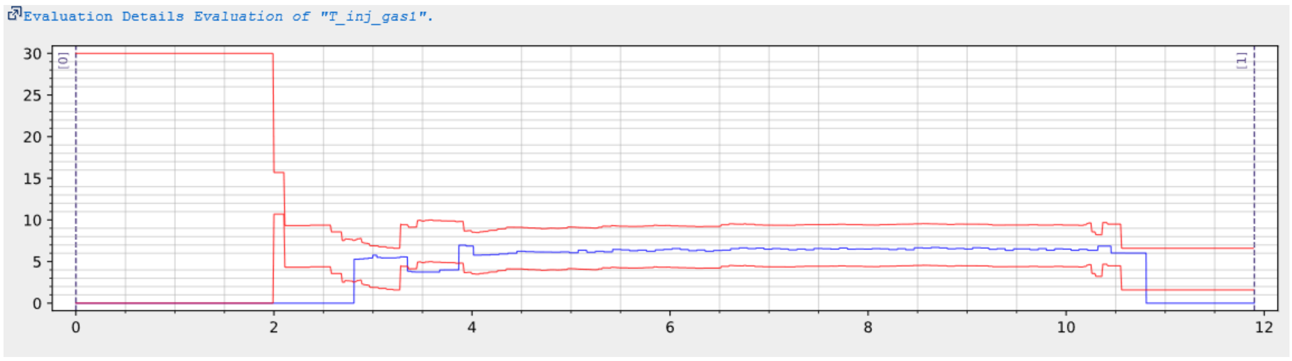


Figure 27: Result for the dynamic test in HiL

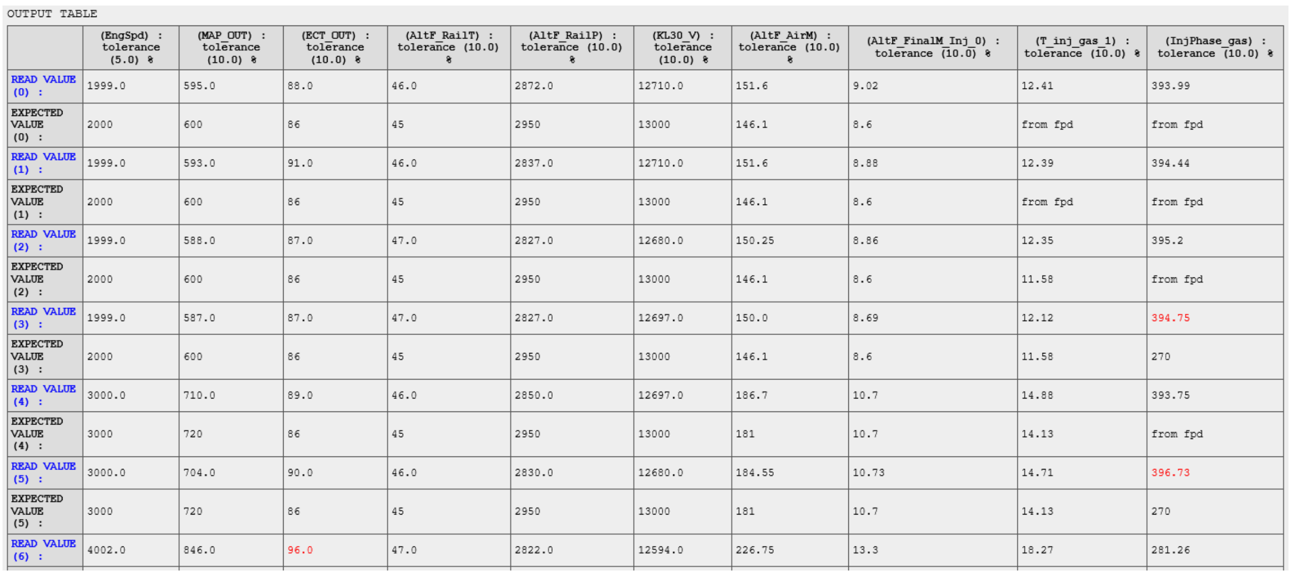
In figure 26, it is possible to see an example of the inputs provided to the ECU: the inputs must be provided from an acquisition taken from INCA (this type of approach can be useful in order to reproduce a complex manoeuvre made on vehicle, like a driving cycle). In this case, the test was conducted in order to check the correct analysis of the injection phase and of the quantity of fuel injected inside the combustion chamber (for simplicity, just the injected quantity of fuel in ms was checked and presented inside the report).
As it was introduced previously, the report of of the result can be visible in figure 27, in this case, a continuous check on the produced result is present, so it is possible to customize the tolerance on the result in order to provide reliable result on it. The example reported in figure 27 is just an example of a single test, so this can not be considered as the official conducted test for the quantity of fuel injected.

6 Activity results

This chapter is dedicated to the discussion of the conclusion. Manual test results and automatic test results will be compared in order to find discrepancies and to discuss the advantages and disadvantages of the two approaches. Then, a discussion of the limits of these approaches is presented.

6.1 Results and discussion

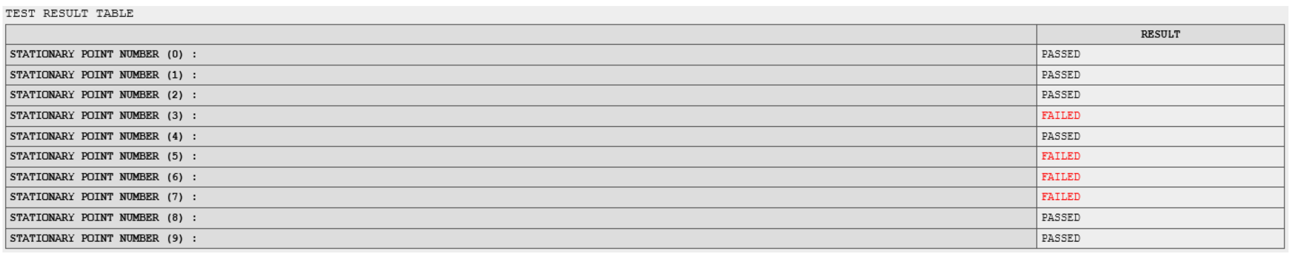
In this sub-chapter, a general description of the tests conducted is done. First, it is possible to see from figure 28 and figure 29 an example of the results and the inputs for a static test: the test was simplified in order to get more visible results, in fact, just ten points were presented in the report. The test was conducted considering comparing expected results and calculated outputs, so for each point a check was done. The first conducted test (treated as example) was done in order to check the right quantity of injected fuel inside the combustion chamber and the injection phase. As it is possible to check, there are some points that are not correct, this means that the software could have problems (the fails were wanted in order to prove the consistency of the application).



OUTPUT TABLE

	(EngSpd) : tolerance (5.0) %	(MAP_OUT) : tolerance (10.0) %	(ECT_OUT) : tolerance (10.0) %	(AltF_RailT) : tolerance (10.0) %	(AltF_RailP) : tolerance (10.0) %	(KL30_V) : tolerance (10.0) %	(AltF_AirM) : tolerance (10.0) %	(AltF_FinalM_Inj_0) : tolerance (10.0) %	(T_inj_gas_1) : tolerance (10.0) %	(InjPhase_gas) : tolerance (10.0) %
READ VALUE (0) :	1999.0	595.0	88.0	46.0	2872.0	12710.0	151.6	9.02	12.41	393.99
EXPECTED VALUE (0) :	2000	600	86	45	2950	13000	146.1	8.6	from fpd	from fpd
READ VALUE (1) :	1999.0	593.0	91.0	46.0	2837.0	12710.0	151.6	8.88	12.39	394.44
EXPECTED VALUE (1) :	2000	600	86	45	2950	13000	146.1	8.6	from fpd	from fpd
READ VALUE (2) :	1999.0	588.0	87.0	47.0	2827.0	12680.0	150.25	8.86	12.35	395.2
EXPECTED VALUE (2) :	2000	600	86	45	2950	13000	146.1	8.6	11.58	from fpd
READ VALUE (3) :	1999.0	587.0	87.0	47.0	2827.0	12697.0	150.0	8.69	12.12	394.75
EXPECTED VALUE (3) :	2000	600	86	45	2950	13000	146.1	8.6	11.58	270
READ VALUE (4) :	3000.0	710.0	89.0	46.0	2850.0	12697.0	186.7	10.7	14.88	393.75
EXPECTED VALUE (4) :	3000	720	86	45	2950	13000	181	10.7	14.13	from fpd
READ VALUE (5) :	3000.0	704.0	90.0	46.0	2830.0	12680.0	184.55	10.73	14.71	396.73
EXPECTED VALUE (5) :	3000	720	86	45	2950	13000	181	10.7	14.13	270
READ VALUE (6) :	4002.0	846.0	96.0	47.0	2822.0	12594.0	226.75	13.3	18.27	281.26

Figure 28: Example of results for the static test



TEST RESULT TABLE

	RESULT
STATIONARY POINT NUMBER (0) :	PASSED
STATIONARY POINT NUMBER (1) :	PASSED
STATIONARY POINT NUMBER (2) :	PASSED
STATIONARY POINT NUMBER (3) :	FAILED
STATIONARY POINT NUMBER (4) :	PASSED
STATIONARY POINT NUMBER (5) :	FAILED
STATIONARY POINT NUMBER (6) :	FAILED
STATIONARY POINT NUMBER (7) :	FAILED
STATIONARY POINT NUMBER (8) :	PASSED
STATIONARY POINT NUMBER (9) :	PASSED

Figure 29: Typical results of a static test

The advantage to use this type of test is the possibility to select each operating point with a particular precision. The checks are made considering a particular tolerance: this data can vary based on the type of output/input that must be wanted to check. So, if the tolerance is too

strict, there is the possibility that some checks could failed, so further considerations on the test must be taken (opposite result will arise if the system takes in consideration a too wide tolerance). The results obtained from the report can not be considered as they are, but other considerations because some fluctuations of the results are possible, so the test is not 100% reliable if an user check is not performed.
In figure 30, it is possible to see a test made on SiL: inputs can be imposed as the user wants, even in an illogical way, in fact, this type of test must be applied only for a first control of the logic of the software.

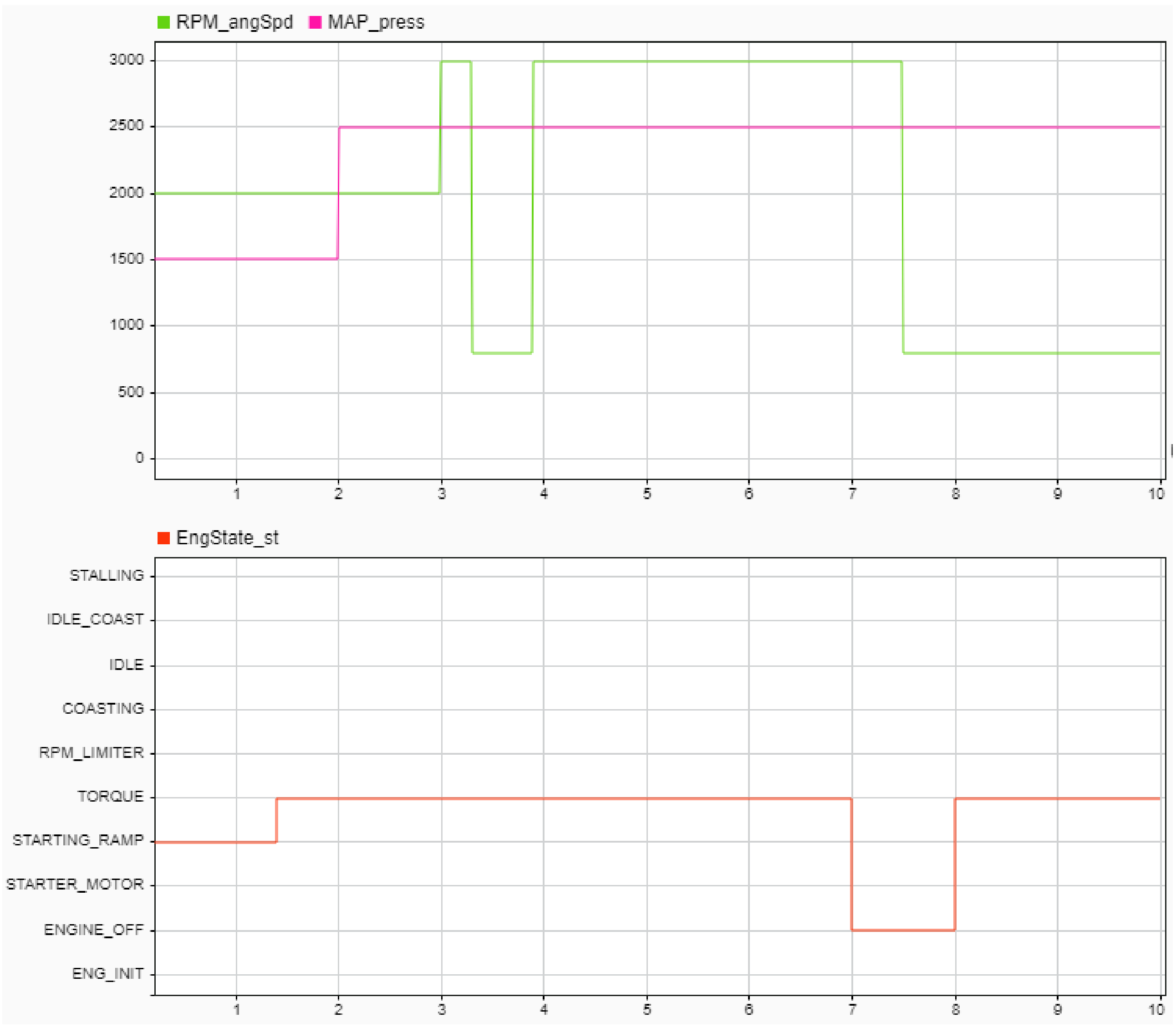


Figure 30: Points of operation for the test in SiL

The status of the lambda sensor monitor can be used as reference in order to take under control the functionality of the system: as it is possible to be assumed, as the load value decreases below a certain threshold, the status of the logic will degrade to "PENDING" (see the chapter 4.1.2 in order to find the software requirement that respects this function). Then, as the system returns to its correct operating condition, the test ends. The results of this test can be seen in figure 31 and figure 32: the test in fact, is passed, so it means that the lambda sensor is new and the functioning is correct. It is necessary to specify that the voltage of the lambda sensor is managed by the user, and, in this case, in order to pass the frequency check, the output voltage has a value of frequency that is reasonable with the thresholds during the check.

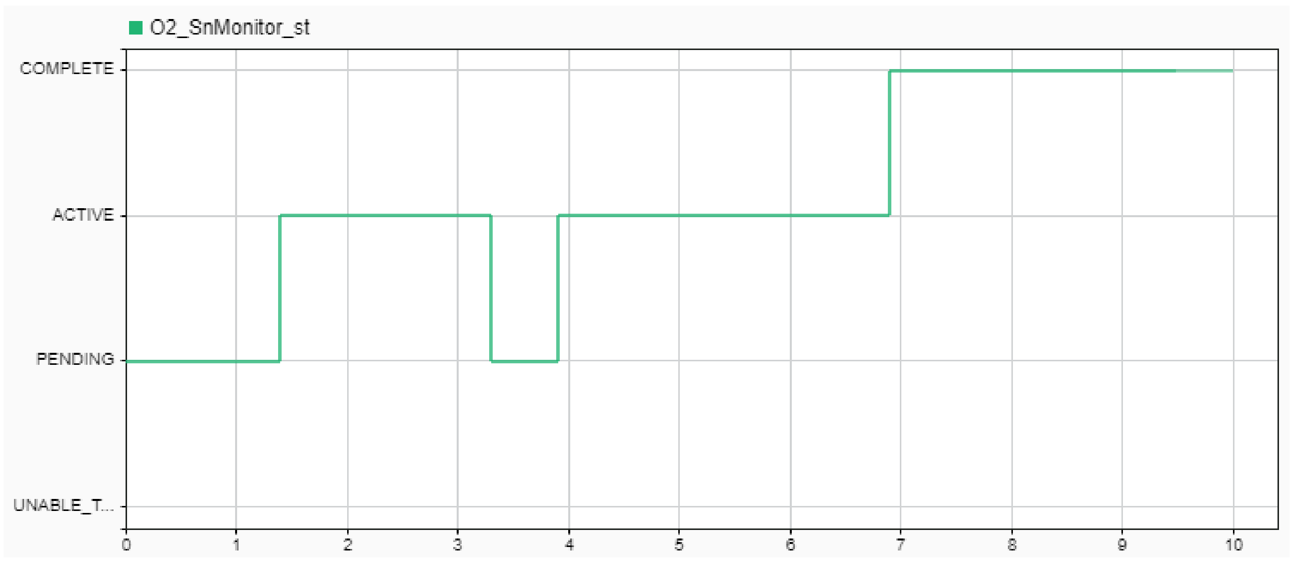


Figure 31: Status of test in SiL for the Oxygen Sensor Monitor

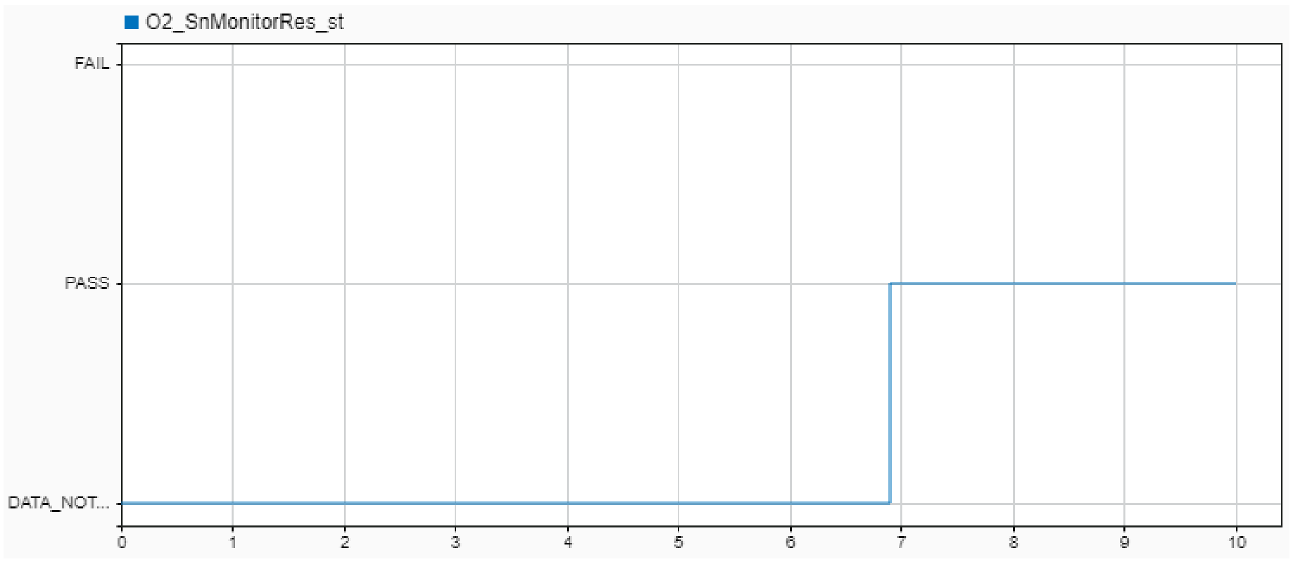


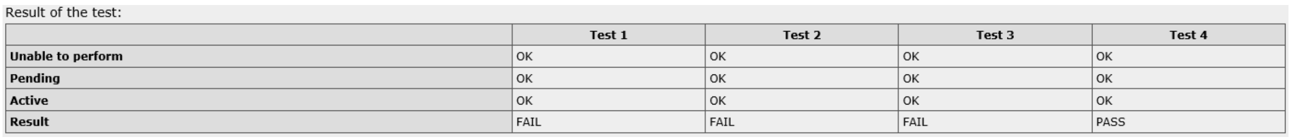
Figure 32: Result of test in SiL for the Oxygen Sensor Monitor

In figure 33, the result coming from an ad-hoc test is present: the test is conducted following the sequence presented in figure 25. The test was made in order to create 3 failed checks and 1 passed check in order to test the functionality of the logic made for the DTC state. In order to create a correct test, only one factor was changed in order to create a "PASS" or a "FAIL". The factor that change the result of the test is the value of output voltage from the lambda sensor, in fact, this could be changed with an appropriate action from AutomationDesk using the following blocks:

- LambdaFixed: the characteristic of the output voltage is flat, so the value of the frequency is equal to zero and this is the cause of the fail of the test (this condition is valid until the characteristic is changed by a manual action from the interface or from another block in the same environment);

- LambdaStd: the characteristic of the output voltage is restored and it follows the standard value, so the value of frequency has a reasonable value and the test is passed.

The result of the test are presented in the report in figure 33, where other conditions except the result of the test are checked: restore of the status of the monitor ("UNABLE_TO_PERFORM"), enabling conditions and activation conditions verified ("PENDING" and "ACTIVE").



Result of the test:

	Test 1	Test 2	Test 3	Test 4
Unable to perform	OK	OK	OK	OK
Pending	OK	OK	OK	OK
Active	OK	OK	OK	OK
Result	FAIL	FAIL	FAIL	PASS

Figure 33: Typical results of an ad-hoc test

The results coming from the report could be considered good and the test can be repeated multiple times in order to check possible changes in the software functionality, and, if the results are the same, the software can be considered good.
In figure 34, it is possible to see the points of operation of a manual test conducted in HiL: the angular speed of the engine and the associated load can vary in a very steep way and the rate of change of the data can be changed by the user.
This can be assumed as an example of inputs for an HiL test and, as it was descripted in the previous chapter, this provides a better simulation respect to the SiL test because of the implementation of a load plates that could provide a higher accuracy on the simulation of real situations.

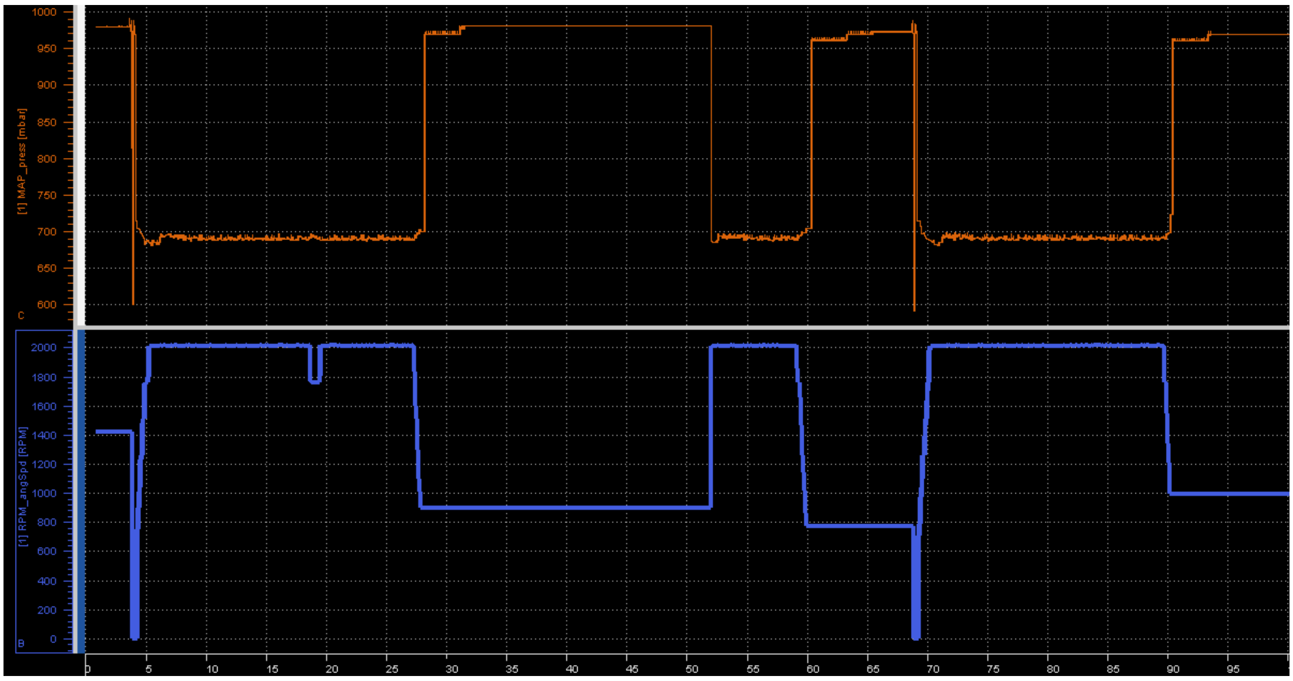


Figure 34: Points of operation for the test in HiL

The figure 35 is referencing to a test done on the HiL bench test: as it is possible to recognize, just the representation of angular speed and load is present, then, the status and the result of the monitor are presented. Following the software requirements that were descripted in the previous chapter, the result of this test is good, so the frequency, offset and drift checks are resulted good. In this case, two checks are present, this is due to the fact that a test on the

DTC state logic is wanted too, so it was necessary to shut down the ECU, in order to repeat another test and to pass in another state for the DTC.

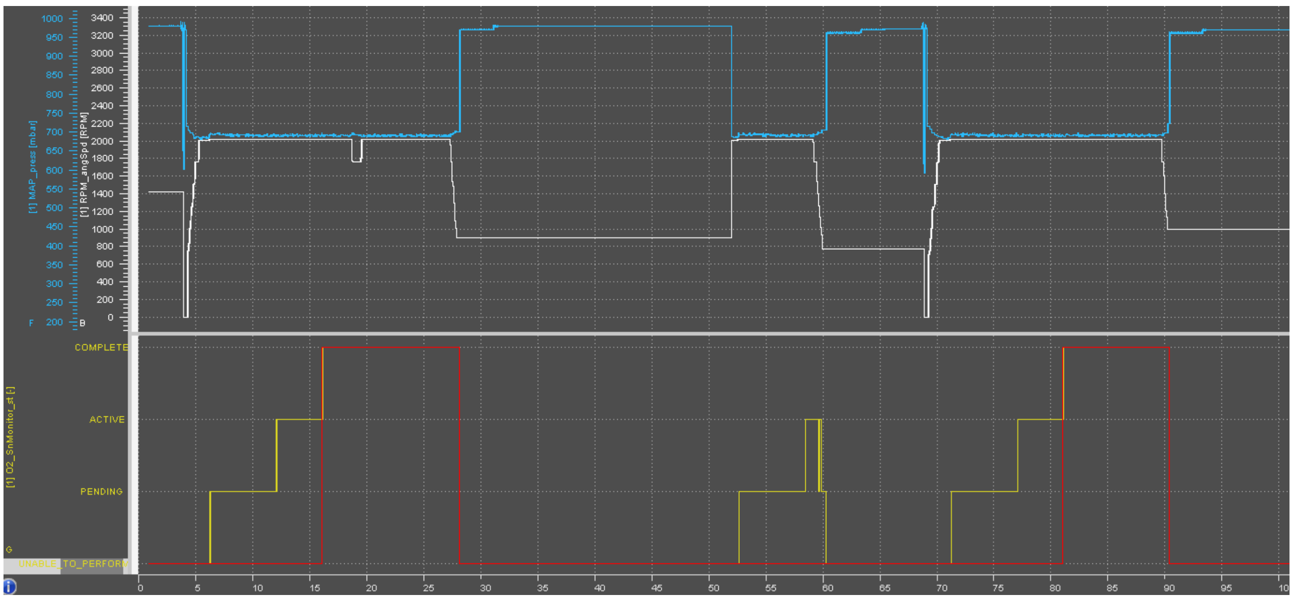


Figure 35: Output for HiL test

As it was possible to see, only two types of automatic tests were used, this is due to the fact that the validation of a software depends by the logic that is necessary to test. In this particular case, the software does not need a specific manoeuvre or a specific driving cycle, because of the fact that the lambda sensor is checked from the functional point of view. The lambda sensor could be affected mainly by ageing, so a specific driving cycle is not required.
The test that were conducted for software validation must be very robust in order to be repeated in the future: it is probable that some changes on the software could be applied, so the results of the test must be the same as before the changes.

6.2 Limits of the research

In this part of the thesis the limits of the research and possible future development of it will be discussed. The automation test is a very useful tool in terms of software validation because of the fact that it could reduce so much the time to validate and to check the possible errors that are present inside a software. The most important limit of the application is the unsuitability of a test realized for one single software functionality for all the software functionalities, this is due to the fact that every software functionality considers a different logic, so a different type of approach to test it. This problem could lead to create for every function a dedicated test that could be used to test future development of the same logic, but not the logic of different software functionalities. In terms of cost and time could be expensive, but it could be considered even as an investment for the future in which a new version of the software with the same logic is released, and it is necessary to test it.
Another problem relative to the software used in order to create the tests is the communication with other softwares when a dynamic automatic test is designed: this type of approach could be very useful in order to copy different driving cycles that could be very complex, but it is useless for what concern the control of the variable inside the CPU.

7 Conclusions

This last chapter is dedicated to a final discussion of the activity covered in the stage period in Westport Fuel System Italy. The thesis focused on a preliminary part where the software funcyionalities were described in their entirety: as first, a description of the software requirements was done in which their functionality was rendered, then a description of the logic and of the conducted test were done.
The aim of this thesis was to focus the attention on the automation of the tests: respect to the manual test, in which the manoeuvres have a bad accuracy, with an automatic test is possible to be very precise, in particular, for the logic of the software functionality that take in consideration timers or specific manoeuvres that have to be done in a specific order, the automatic test is very reliable and accurate. In fact, the errors coming from the human are reduced because every action is preselected with a specific timing decided by the user in advance.
Another advantage emerged from the thesis was the possible re-usability of the test: for example, if the software changes in some way, it is possible to re-test it in order to prove that the produced result is the same. The reliability of this tool is very high because the requirements of the software can always be tested in every moment without increasing the probability of failure of a test, so the important advantage of this methodology is its repeatability.
In conclusion, the company considers to implement this methodology in its process development, therefore, the produced result could be used for commercial purposes of the company. The thesis, in fact, aimed to point out a method of validation and not to create concrete results about a software functionality, but just to underline the need of a company to develop its software validation process.

References

[1] Ahmed Toema, «Physics-based Characterization of Lambda Sensor Output to Control Emissions from Natural Gas Fueled Engines,» Manhattan, 2010

[2] Bovee, «Development of Controller for the Post-Catalyst Oxygen Sensor on EcoCAR Vehicle,» 2010.

[3] Della Fornace, Analisi dell'emissione sonora di un motore a combustione interna: diagnosi misfire e identificazione delle principali caratteristiche della combustione, Master's Degree, University of Bologne, 2009.

[4] Ghiani, Sistema su dispositivo mobile per il monitoraggio dell'autovettura tramite autodiagnosi OBD, Master's Degree, University of Pisa, 2006.

[5] Nibert, Herniter and Chambers, «Model-Based System Design for MIL, SIL, and HIL,» Los Angeles, 2012

[6] Zoffoli, Sviluppo di Modelli Zero-Dimensionali per la Simulazione di Componenti di Motori e Veicoli su Piattaforme HiL, Bologna, 2018.