**Discorso presentazione discussione tesi**

**Slide 1: Introduction**

Good [morning/afternoon], everyone. My name is Mattia Oliva, and I am here to present my thesis on the *definition of a library for an automotive ECU API layer using Model-Based approach* with a focus on *memory management and on-board diagnostics.*

Before starting,I would like to thank my advisor, Prof. Massimo Violante, and my internship tutor, Eng. Emilio Bertrand, for their guidance throughout this journey.

**Slide 2: Collaboration with Metatron**

This thesis was made possible through collaboration with Metatron S.p.A., a company within the Landi Renzo Group that specialises in developing electronic and mechanical components for alternative fuels. The work was conducted at the Metatron Research Centre in Volvera, Turin.

**Slide 3: Current Situation**

Over the years, Metatron has developed a complex and layered code base in response to evolving standards and customer demands. However, this complexity has led to challenges in managing the code effectively.

The idea for this thesis came from the need to overcome the current complexity of SW in order to cut down on errors and time, as well as the desire to provide the customers with easy-to-use interfaces for model-based development.

Our goal was to create a flexible and standardised API layer that would simplify development, enhance portability, and ensure compliance with diagnostic standards across various markets.

**Slide 4: Thesis Goals**

This thesis actually continues the path of standardisation and revision of the interface level of the Metatron platforms and does so while focussing on the On-Board diagnostic.

To address this challenge, the process was split into smaller steps.

The first step was, of course, the analysis of the current ECU API layer/MBSL libraries and the documentation about the main functions of a generic ECU.

The second step was the implementation of a non-volatile memory management strategy.

That was then followed by the definition and implementation of the actual diagnostic manager functions, but only after a careful study of the state-of-the-art diagnostic standards like Euro-VI.

Last but not least, the plan included the development of one or more demo applications to validate the implemented solutions.

**Slide 5: Architecture**

Here you can see the software architecture of the HDS9 board we used as a base, with a well-defined separation of concerns like AUTOSAR’s.

The work of this thesis is limited to the upper and middle layers, with the implementation of APIs to be used at the model-based level.

This, instead, is the architecture of the implemented solution; as you can see, at the foundation of this work there’s a solid and general-purpose memory management strategy.

We needed it to be general purpose, both as a good principle per se and mainly because at this moment we still had no idea of how the on-board diagnostic system would be implemented.

We designed the strategy to ensure flexibility as we navigated the complexities of the OBD system.

We made use of the NVRAM modules of the board in combination with the designed strategy to obtain data redundancy and reliability.

**Slide 6: NVRAM Management - HIL Testing**

To validate our memory management system under continuous working conditions (powering on, reading, writing, and powering off) and to ensure the correct functioning of the strategy, we conducted Hardware-in-the-Loop (HIL) testing.

Two automated tests were implemented using LabVIEW, focussing on the system's resilience under stress and the flexibility of our memorisation strategy.

Here you can see a Data Acquisition system used for the tests.

**Slide 7: NVRAM Management - Results**

The data collected were then compared to the expected results (the values calculated by the board were all defined by mathematical formulas that can be found in the thesis).

The results of our tests were promising. The first test demonstrated correct data retrieval and module selection, while the second ran for over four days, simulating 6,000 ignition cycles and collecting more than 30,000 samples.

We observed the expected behaviour, confirming the integrity of the data stored in memory.

**Slide 8: Diagnostic Requirements**

We'll now move on to the thesis' main goal, which is to restructure the on-board diagnostic system to meet the most recent requirements.

At the same time, it was also meant to decrease the system's and its interfaces' complexity for greater maintainability and ease-of-use by future customers.

It may also serve as a starting point for future implementations.

The work started with the analysis of the top-of-the-line On-Board Diagnostic standards, like Euro-VI and China-VI, defining the system requirements to subsequently develop the architecture you see here.

**Slide 9: Diagnosis Flow**

The different components of the architecture correspond to the various phases of the diagnostic flow, that is, the sequence of actions intercurring between a fault detection by the board’s sensors and its actual validation and eventual insertion in memory.

As it can be seen in this representation, we selected a modular approach to enhance maintainability.

Circled by a dotted line, you can see the parts that can be managed by the users. Any other component works “behind the scenes” and can only be accessed by a user through predefined APIs.

Of all these components, the most important is without a doubt the finite state machine, which, as you can see, pilots the three final components: the recovery actions, the malfunction indicator light, and the error memory.

**Slide 10: FSM**

The automaton, named ADIA from Automatic Diagnostic, plays the crucial role of validating the identified faults and managing the error memory accordingly, all while setting up the MIL and the recovery lines.

Each fault has its own ADIA, which cycles through different states based on the received inputs and the user-defined calibrations.

The original code was simplified, condensing the multiple types of ADIA into a single, standard-compliant automaton, as you can see from these images. Please note that the actual implementation was not through model-based software but with lower-level code, as reported in the thesis.

**Slide 11: Error Memory & Freeze Frame**

As said, the ADIA pilots the Error Memory and Freeze Frame (a snapshot of the engine state at the moment of the fault), both vital components for the OBD, whose behaviours are widely described in the standards.

Their contents are essential for the correct functioning of the system.

For their implementation, we focused on minimising the data stored in non-volatile memory while ensuring that essential information is retained for effective diagnostics.

Here you can see the direct correlation between the structure for the faults in memory and the associated freeze frames.

The fields of the latter were obtained by combining and pruning the requests of both Euro-VI and China-VI.

**Slide 12: Interfaces**

To facilitate model-based design, besides creating APIs that are now available as SIMULINK blocks in the company’s library, we developed user-friendly SIMULINK masks to simplify the setup process for the OBD strategy.

**Slide 13: Video**

Now, I would like to show you a short video of the execution of the demo application made to test the final version of the solution.

On the left, you can see the screen of CANape, an analysis tool used to communicate with the board and read and write data directly to it. Inside of this screen, two graphs show the behaviour of the MIL and the internal counter of the chosen fault's ADIA.

Next to the counter graph, you can see a visual representation of the fault-activated recovery lines.

On the right, instead, the breakout box of the board is used to show the MIL status with the red LED and the created fault with the blue one.

The open load fault is created by detaching a pin.

As you can see, the LED turns off, and the ADIA counter immediately starts to raise as it receives the “open load” fault.

Once the counter reaches the threshold, it stabilises, and the recovery lines are activated. In this case, only the first line was selected.

As a fault is now present in memory, the MIL behaviour changes, showing three peaks rather than one every five seconds.

Once we heal the fault by reconnecting the pin, the ADIA counter begins to fall, and once it reaches zero, the fault is devalidated, the recovery line is turned off, and after a while the MIL behaviour returns to normal.

**Slide 14: Work Recap**

Before concluding, here is a short summary of the work done for this thesis.

First, we analysed pre-existing code and implemented a new strategy, using HIL testing to tune the designed solution.

In the second part, we started with the analysis of the current standards to design and implement each component of a new OBD strategy, performing both integration and system testing, and producing useful interfaces for the users to interact with the system.

**Slide 15: Conclusion & Next Steps**

In conclusion, we have successfully simplified the system architecture, thus increasing maintainability and reducing the error points. We also increased modularity and ensured compliance with OBD standards.

Moving forward, we aim to integrate protocols with the Basic Software Level and identify a pilot project to fully test the system in a real-world environment.

**Slide 14: Thank You**

That’s all, folks. Thank you for your attention. I am happy to answer any questions you may have.