

Master's Degree Thesis
Master's Degree
in MECHATRONIC ENGINEERING

Development toolchain for Vehicle Electrification

POLITECNICO DI TORINO



Advisor:

Prof. Stefano CARABELLI

Candidates:

Ylber HOXHA
Muaaz TARIQ

Academic year

2019-2020

Summary	
Acknowledgements.....	11
Abstract.....	12
Introduction.....	14
What we are doing.....	15
Why we are doing it?	15
How we are doing?.....	15
Chapter 1, Part 1	16
1. ISO26262	16
ISO26262 series of standards:	16
Functional safety	17
Process definition	20
2. V-cycle.....	22
German V-Modell.....	22
Objectives of V-Modell	23
Basic V-Cycle components	23
Systems engineering.....	23
3. Model Based Design	25
3.1. MAAB Guidelines.....	27
4. Hybrid V-Cycle.....	30
ISO26262 into V-cycle.....	30
Hybrid V-Functional safety concept	30
Steps for the Hybrid V-Functional safety cycle	31
Explanation for the steps of Hybrid V- cycle.....	32
1. Customer requirements	32
2. Technical and production model	32
3. Rapid control prototype.....	33
4. Testing of Production model on dSpace	33
Minor changes.....	33
5. Vehicle Management Unit selection	33
6. Vehicle management unit.....	33
7. Integration and testing	33
Minor changes.....	34
Major changes.....	34
8. Final product requirements.....	34

5. Innovative Modular Architecture.....	35
1. Environment	35
2. Plant.....	35
3. Control.....	36
4. Human machine interface	36
5. Operator	36
6. Interface	36
5.1. Interfaces	36
5.2. Concept of reusability and modularity in Hybrid V-cycle.....	37
Software block.....	37
Hardware block.....	37
Rapid Control Prototype Block	38
Vehicle Management Unit.....	38
5.3. Architecture impact in integration and testing	38
Software in the loop.....	39
Hardware in the loop containing Rapid control prototype	39
Code generation.....	40
Frame	40
Rapid Control Prototype Loop (4 th & 5 th step)	41
Hardware in the loop for vehicle management unit	42
VMU loop (7 th step).....	43
Flexibility in model	43
Different combinations of software in the loop,	43
Proposed Combinations of Hardware in the Loop Containing Rapid Control Prototype	45
Proposed Combinations of Hardware in the Loop for Vehicle Management Unit.....	46
1. Customer requirements	47
2. Technical and production model	47
3. Rapid control prototype.....	48
4. Testing of Production model on dSpace	48
5. Vehicle Management Unit selection	48
6. Vehicle management unit.....	48
7. Integration and testing	48
8. Final product requirements.....	48
5.4. Standards and guidelines check.....	49

6. Documentation.....	50
PART 1, CHAPTER 2	53
1. Customer Requirements	54
Item definition	54
Hazard analysis and risk assessments	58
1.1. Functional safety concept.....	60
Concept model.....	60
Architecture & Design	62
Interfaces	63
2. Technical Model.....	64
Initiation	64
Software architecture and specification of safety requirements.....	65
Model-based Design.....	67
Task 1 - Supervisory control	68
Task 2 - Digital Filter	69
Validation of Task 2	70
Task 3 - Fast Fourier Transform	71
Validation of Task 3	72
Emergency Task.....	73
Scheduling.....	74
Interrupts	76
Validation of Task 1	78
Control frame	78
Human-Machine-Interface	80
Testing.....	83
Software in the Loop	83
Standards check.....	85
PART 2, BAT-MAN PROJECT.....	87
1. Customer Requirements.....	88
Item definition	88
Estimation of State of Charge	89
Estimation of State of Health	90
Lead-Acid battery.....	91
Requirements specification	91
Hazard analysis and risk assessment.....	93

Concept model.....	95
Architecture & Design	98
Environment.....	100
Plant.....	101
Control - Hardware architecture.....	103
Control - Software architecture	105
Application architecture	105
Interfaces	107
Requirement 3-2.....	107
Housing and LIN Connection.....	110
Production development – Software level	111
2. Technical model.....	112
Control frame	115
SIL	116
Production Model.....	120
Standards and guidelines check.....	121
HMI	124
Suppliers.....	127
3. VMU.....	127
CC2640R2F-Q1 MCU characteristics and safety	127
INA226.....	128
4. Integration & Testing.....	131
Conclusion.....	135
Appendix.....	136
Technical_model_Digital_Filter_load_file_22_11_19	136
Concept_model_Digital_Filter_load_file.....	138
Technical_model_BatmanApp – ert_main.c.....	138
Project_zero.c – Initialization in Main function.....	143
Bibliografi.....	146

Table of Figures:

Figure 1. Go to market statistics	14
Figure 2. Functional safety cascade (cadence, 2019)	18
Figure 3. Safety lifecycle for software product development (Iso.org, 2018)	18
Figure 4. Artefacts	20
Figure 5. V-cycle for software	22
Figure 6. Systems engineering	24
Figure 7. V-Model for System Development and types of Simulation (MathWorks, n.d.).....	25
Figure 8. Model based design model flow	26
Figure 9. MAAB guideline for Simulink modelling example (MathWorks, n.d.)	28
Figure 10. Prohibited blocks inside controllers (MathWorks, n.d.).....	28
Figure 11. MAAB for filenames (MathWorks, n.d.)	28
Figure 12. Iso26262 Enforced on V-cycle	30
Figure 13. Hybrid V-functional safety concept	31
Figure 14. Steps for hybrid V-cycle.....	31
Figure 15. Reusability and modularity concept	35
Figure 16. Interface analysis (Ross, 2016).....	36
Figure 17. Software blocks definition.....	37
Figure 18. Hardware blocks definition	37
Figure 19. RCP representation of hardware block.....	38
Figure 20. VMU representation of hardware block	38
Figure 21. Software in the loop.....	39
Figure 22. Hardware in the loop containg rapid control prototype.....	40
Figure 23. Code generationin.....	40
Figure 24. Defining frame.....	41
Figure 25. RCP loop	41
Figure 26. Hardware in the loop for VMU	42
Figure 27. VMU loop.....	43
Figure 28. Combinations of software in the loop	44
Figure 29. Combinations of software in the loop	44
Figure 30. Combinations of software in the loop	45
Figure 31. Proposed combinations of hardware in loop for rcg	46
Figure 32. Proposed combination of hardware in the loop for VMU	47
Figure 34. Meeting Notes snapshot	51
Figure 35. Documentation architecture.....	51
Figure 36. Digital filter and FFT block scheme.....	54
Figure 37. Layout.....	55
Figure 38. ASIL Selection table.....	59
Figure 39. Concept model in Simulink	61
Figure 40. Results of Concept Model	61
Figure 41. Item Architecture.....	62
Figure 42. Modelling and coding guidelines	64
Figure 43. Notations for software architectural design.....	65
Figure 44. Error detection at the sw architectural level	65
Figure 45. Methods for the verification of the software architectural design.....	66

Figure 46. Illustration of HW Interrupt.....	66
Figure 47. Illustration of model hierarchy	67
Figure 48. Notations for software unit design	67
Figure 49. Design principles for sw unit design and implementation	68
Figure 50. Methods for software unit testing.....	68
Figure 51. Stateflow chart.....	69
Figure 52. Task control	69
Figure 53. Digital Filter modelling	70
Figure 54. Results 1 of Digital Filter and FFT.....	70
Figure 55. Results 2 of Digital Filter and FFT.....	71
Figure 56. Task control	72
Figure 57. FFT modelling	72
Figure 58. FFT result	72
Figure 59. FFT Result	73
Figure 60. Interrupt handling	73
Figure 61. Scheduling	74
Figure 62. Hardware interrupt block and its parameters.....	76
Figure 63. Hardware interrupt flowchart	77
Figure 64. Control content with Interrupt simulation	77
Figure 65. Chart mode	78
Figure 66. Quantizer	78
Figure 67. FFT Output with 4 bit quantization	79
Figure 68. PWM Design	79
Figure 69. PWM Results.....	80
Figure 70. HMI Design	81
Figure 71. HMI Feedback	81
Figure 72. Simulink model architecture view	82
Figure 73. SIL Settings	83
Figure 74. Simulink model to run SIL	84
Figure 75. SIL results.....	84
Figure 76. ISO 26262 check results.....	85
Figure 77. MISRA C:2012 check results	85
Figure 78. MAAB guidelines check results	86
Figure 79. Code generation advisor check results	86
Figure 80. Item definition	88
Figure 81. Item definition	89
Figure 82. ASIL Selection	95
Figure 83. Concept model from customer	96
Figure 84. Inputs (left) and Outputs (right)	97
Figure 85. Inputs (left) and Outputs (right)	97
Figure 86. Inputs (left) and Outputs (right)	98
Figure 87. Concept model modified	98
Figure 88. Architecture illustration.....	99
Figure 89. Relation between Cn and Temperature	100
Figure 90. Battery typical operating points (bluebox.co.uk, n.d.)	101
Figure 91. Ageing in different temperatures (bluebox.co.uk, n.d.)	101

Figure 92. Battery location in the trunk of the car	102
Figure 93. Thevenin model	102
Figure 94. Hardware architecture	104
Figure 95. Power supply architecture	104
Figure 96. Software architecture.....	105
Figure 97. Application architecture	105
Figure 98. Kalman filter working principle	106
Figure 99. Error management flowchart.....	106
Figure 100. EMC Coupling.....	108
Figure 101. Housing and LIN Connection.....	110
Figure 102. Temperature in Environment module.....	112
Figure 103. Plant module.....	112
Figure 104. HMI module	113
Figure 105. Parameters of BAT-MAN Application	113
Figure 106. Control Module	113
Figure 107. Architecture in Simulink	114
Figure 108. 16 bit ADC Output of Voltage (left), 16bit ADC Output of Current (right).....	115
Figure 109. SoC with 12 bit quantization (left), SoH with 12 bit quantization (right).....	115
Figure 110. SoC with 16 bit quantization (left), SoH with 16 bit quantization (right).....	115
Figure 111. SIL Model for BATMAN.....	116
Figure 112. Data set 1: Simulation (left) vs SIL (right).....	116
Figure 113. Data set 2: Simulation (left) vs SIL (right).....	117
Figure 114. Data set 3: Simulation (left) vs SIL (right).....	117
Figure 115. Data set 1: Difference between SoH in simulation and from SIL	117
Figure 116. Data set 1: SoC difference between Simulation and SIL	118
Figure 117. Data set 2: SoH difference between Simulation and SIL	118
Figure 118. Data set 2: SoC difference between Simulation and SIL	118
Figure 119. Data set 3: SoH difference between Simulation and SIL	119
Figure 120. Data set 3: SoC difference between Simulation and SIL	119
Figure 121. Profiled sections of code	120
Figure 122. Traceability report	121
Figure 123. Static code metrics.....	121
Figure 124. ISO 26262 check report.....	122
Figure 125. IEC 61508 check report.....	122
Figure 126. MISRA C check report.....	123
Figure 127. MAAB Guidelines check report.....	123
Figure 128. Code generation advisor check summary	124
Figure 129. BAT-MAN Mobile app first page (Fazio & Fazi, 2019).....	125
Figure 130. BATMAN App second page	125
Figure 131. Updated BATMAN App	126
Figure 132. Android studio code modification	126
Figure 133. Hardware architecture with chosen components (Fazio & Fazi, 2019)	127
Figure 134. INA226 Layout example (ti.com, n.d.)	129
Figure 135. BAT-MAN PCB.....	129
Figure 136. Generated code flowchart.....	131
Figure 137. 'ert_main' Source file.....	132

Figure 138. CCS workspace	133
Figure 139. Memory occupation of firmware architecture	133
Figure 140. Memory occupation integrated.....	133

Acknowledgements

Firstly, I would like to thank my advisor Professor Stefano Carabelli for giving me this opportunity to work on such an important topic of today's industry. I would like to thank him for our long discussions, sharing his new ideas and his availability throughout these last six months and his patience with me.

Secondly, I would like to thank my supervisor Engineer Giovanni Guida for his continuous support and motivation given to me every single day. I thank him for his patience since I certainly know that it was not easy for him to break through my stubborn character and teach me the industrial environment and its mentality.

I sincerely thank my family for supporting me in any possible ways and in the same time always letting me a free man, making my own decisions and live by them. Thanks to that I have become the person I am today.

Abstract

The biggest challenge of the automotive industry today is the increasing complexity of it. Today, a high-end car software has approximately 100 million lines of code, that makes it one of the most complex machines. This comes with a drawback, increasing the probability of software defects which can cause system failures, thus increasing the risk of damage to a human. Moreover, this complexity has increased the cost of the production. Both of these mentioned topics created the objective of finding more efficient ways for developing a structural toolchain and reusable software, which indeed are the key words of this thesis. The main reference for this thesis is the international standard for automotive industry ISO26262 titled as “Road Vehicles – Functional Safety” that provides us with the requirements for Electrical and/or Electronic systems.

ISO26262 is strictly defined for requirements that must be fulfilled, not tools or ways to satisfy those requirements, thus the need and necessity to develop toolchains to satisfy these requirements. Our goal was creating a development toolchain integrating or combining ISO26262, V-Cycle and Systems Engineering in one, that is what we called Hybrid-V-Cycle. In our quest to integrate all the methods into one, we started with the V-Cycle which is a standard in the development of any component of vehicle and then enforced this cycle upon the ISO26262 standards. By this way, we developed a Hybrid-V-Cycle with feedback loops for continuous improvement which will be our first chapter. Each step of it is further improved by pointing the safety requirements we must fulfill in that step. Furthermore, we will explain our innovative approach for the Architecture of the system that focuses in two main things – Modularity and Reusability. In the last part of the chapter there will be an example on how we apply our Hybrid development chain with a simple project – Digital Filter. This part is done as a group of to people, by me and my colleague Muaaz Tariq.

In the second part I will jump to a real automotive project – a battery manager for Lead-Acid batteries called BAT-MAN, developed by ‘Brain Technologies s.r.l.’ that is going to be our Customer. This part is done by me. The project is focused on innovative estimation algorithm to estimate State of Charge and State of Health of a battery. On our several interactions with the customer we set up the requirements and they provide us with Concept model. After that we proceed as in the first chapter. It is important to mention that there are several parts of this project that we were not able to share because the BAT-MAN project is in the process of the patent application. The ascending branch of the V-Cycle will be the future of this project, where the software of BAT-MAN must be integrated, tested and after all be certified by ISO26262 and release on the market.

The newly developed Hybrid-V-Cycle caters the needs for automotive component development considering all the safety standards now in place. With the example we showed the effectiveness of this development toolchain and applying it to the real-world BAT-MAN project showed that how it can be helpful in tackling real world complex problems. This Hybrid-V-cycle makes sure that we are compliant with the functional safety standards and makes the work easier to handle, thus increasing the efficiency. Also, adding here the modular architecture developed in this thesis makes it usable for several other fields, especially complex Control Engineering projects.

Introduction

Automotive sector is very competitive and challenging nowadays. There are many companies which are trying hard to increase their share of the market. This competition is forcing companies to use innovative methods to reduce the time of production (time to market) of any product. A research conducted by Jabil [2] shows that in 2017, 68% of the automotive manufacturers told that their time to market is less than 2 years. While it steadily increased to 71% in 2018. Shortening the time to market is a good trend but it raises some problems of functional safety. If we are reducing the time to market, we must cut down our time for the whole process chain. The major time-consuming factors which are delaying process are;



Figure 1. Go to market statistics

[2]

We are considering the major problems which are;

- High research and development costs
- Meeting government and safety regulation
- Long test cycles
- Procurement/supplier selection
- Meeting Government and safety regulations

In order to reduce the time required for getting your product to pass through all the safety regulations, the automotive sector has developed some safety rules which are proving to be less time consuming and since most of them follow the same standards so it is easy for the government to pass the product in less time. The safety standards which are being followed are ISO 26262. While in order to reduce the time to market the automotive manufacturers are starting to use new approaches for the product development which is model based designing. Apart from the need to reduce time to market, companies are also focusing on streamlining the projects. They are trying to develop certain systematic principles to follow by which they can create a new product. A set of basic rules which will be followed in every project and can be

adapted to different kinds of project. The basic purpose is to develop a systematic way to find the solution of the problem.

What we are doing

Developing new Hybrid V-cycle considering, ISO 26262 (safety standard for electric components in vehicles) with model-based engineering, systems engineering and V-cycle to streamline the process for product development while keeping in mind the functional safety concepts of the model.

Why we are doing it?

We are doing it to reduce the time to develop a software or hardware for our vehicles so that we can reduce time to market for both parts.

How we are doing?

We are taking a simple example, whose requirements are provided by the customer and trying to pass it through all the phases of our process so that we can establish the whole procedure for simple process and then we can move forward and apply the same process to real world projects.

Chapter 1, Part 1

1. ISO26262

ISO 26262 is the safety standard which is specific for automotive industry. It applies to safety-related road vehicle electronic and electrical systems, and addresses hazards due to malfunctions. It provides the whole lifecycle of the E/E system (including H/w and S/w components). Important thing about this standard is the documentation. We must produce documents and know which steps to follow to produce these documents. The standard defines everything, and we follow the whole procedure to get results. The description of the standard as given by the official website is as follows.

ISO26262 series of standards:

- Provides a reference for the automotive safety lifecycle and supports the tailoring of the activities to be performed during the lifecycle phases, i.e., development, production, operation, service and decommissioning
- Provides an automotive-specific risk-based approach to determine integrity levels [Automotive Safety Integrity Levels (ASILs)]
- Uses ASILs to specify which of the requirements of ISO 26262 are applicable to avoid unreasonable residual risk
- Provides requirements for functional safety management, design, implementation, verification, validation and confirmation measures and
- Provides requirements for relations between customers and suppliers.

[3]

<i>Severity level</i>	<i>Title</i>	<i>Description</i>
S0	Low	No injury
S1	Moderate	Light or moderate non life-threatening injuries to the driver or passengers or people around the vehicle
S2	Serious	Severe and life-threatening to the driver or passenger or people around the vehicle or in other surrounding vehicles
S3	Severe	Life-threatening injuries life-threatening to the driver or passenger or people around the vehicle or in other surrounding vehicles

Table 1. Severity levels [3]

<i>Exposure level</i>	<i>Title</i>	<i>Description</i>
E0	Incredible	Situations that are extremely unusual
E1	Rare	Very low probability
E2	Sometimes	Low probability
E3	Quite often	Medium probability
E4	Often-Always	High probability

Table 2. Exposure levels [3]

<i>Controllability</i>	<i>Title</i>	<i>Description</i>
C0	Controllable in general	Controllable in general by all drivers
C1	Simply Controllable	Less than 1% of the drivers or other traffic participants are usually unable to control the damage
C2	Normally Controllable	Less than 10% of the drivers or other traffic participants are usually unable to control the damage
C3	Difficult to Control	The average driver or other traffic participant is usually unable, or barely able to control the damage

Table 3. Controllability levels [3]

The Draft International Standard (DIS) of ISO 26262 was published in June 2009. Since the publication of the draft, ISO 26262 has gained traction in the automotive industry. Because a public draft standard is available, lawyers treat ISO 26262 as the technical state of the art. The technical state of the art is the highest level of development of a device or process at a time. According to German law, car producers are generally liable for damage to a person caused by the malfunction of a product. If the malfunction could not have been detected by the technical state of the art, the liability is excluded [German law on product liability (§ 823 Abs. 1 BGB, § 1 ProdHaftG)].

[4]

Functional safety

According to ISO 26262, functional safety is defined as the “absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical/electronic systems”.



Figure 2. Functional safety cascade [5]

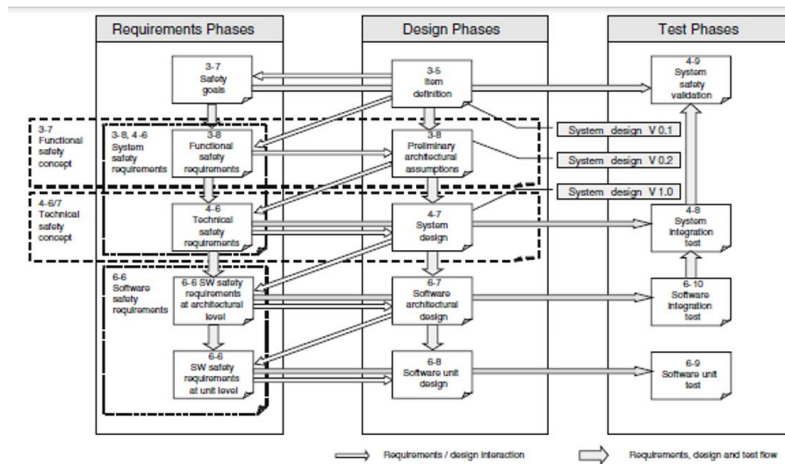


Figure 3. Safety lifecycle for software product development [3]

This standard is relatively new in the automotive industry. It is entirely based on concept of functional safety. It was developed to enforce functional safety measures in a robust manner. With the fast-changing technology, every company wants to reduce the time required for testing the model. But this must be done in a safe way, hence, ISO 26262 enforces safety standards to already existing development models to produce the same safety functions. ISO 26262 is divided in following parts 10 portions;

1. Vocabulary
2. Management of Functional Safety
3. Concept Phase
4. Product Development: System Level
5. Product Development: Hardware Level
6. Product Development: Software Level

7. Production and Operation
8. Supporting Processes
9. ASIL-oriented and Safety-oriented Analyses
10. Guidelines on ISO 26262

Concept phase – This is the first development phase that ISO 26262 defines. It includes:

- Item definition - using layouts, illustrations, definitions to define the project clearly
- Hazard analysis and risk assessment – using FMEA, Situational analysis etc. we define the hazards and analyze their risks
- Functional safety – after Hazard analysis and risk assessment is done, we define the ASIL, a Safe state and the Functional safety concepts

Furthermore, while we are in the first steps of development of V-Cycle we also have to handle:

- Customer requirements – meetings with customer must be arranged and a table of requirements must be created
- Concept model – the Concept model can be given by the Customer, if not, we shall do. The definition of Concept model will be explained in Model based design part.

Here we can see that immediately our development toolchain has to melt V-Cycle, ISO 26262 and requirement engineering into one Hybrid V-Cycle.

Product Development: System Level – Here we define our systems architecture and interfaces. i.e. system level product development. In this thesis we will introduce an innovative architecture where the keywords of it are Modularity and Reusability. This architecture will be very helpful especially in the integration and testing part where the Modules can be very easily handled. Indeed, the integration and testing is defined in ISO 26262 in ‘4-7 System and item integration and testing’. Furthermore, the technical safety aspects will be defined, taken by ISO 26262.

Product Development: Software Level – In this thesis we will be dealing with Model based design, thus the software produced by us will be automatically generated. In this chapter ISO 26262 defines:

- General topics for software development, i.e. with a level of abstraction
- Specification of software safety requirement
- Define safety aspects
- Software architecture design
- Integration and Testing

We need to add also:

- Technical model
- Simulation
- Verification with Concept model
- Production model
- Code generation

Again, we see that we need a development that makes these work altogether.

Process definition

According to ISO 26262, every process must be defined clearly before it starts, i.e. we must define:

- Methodologies
- Tool aspects
- Safety aspects
- Techniques
- Artefact

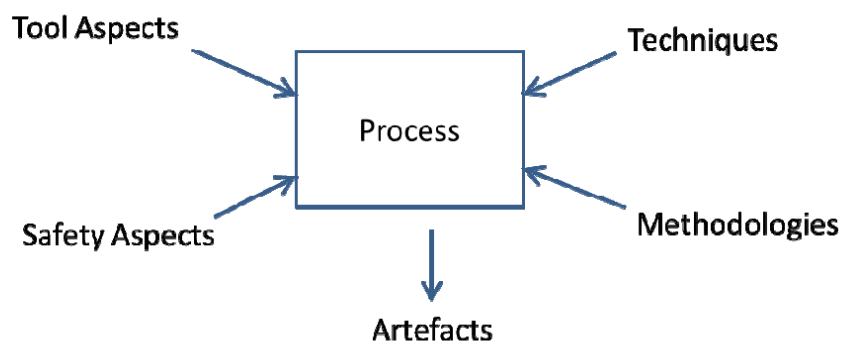


Figure 4. Artefacts [6]

Methodologies – This means that we need to define what kind of methodology we are using in that process. For example, Model-based Design is the Methodology in the process of ‘Technical Model’. Another example for the process of ‘Hazard Analysis and Risk Assessment’ the methodology can be a type of FMEA, Situational Analysis etc.

Tool aspects – What tools are we using for getting the process done. For example, MATLAB is one of the tools used for design, Embedded Coder for code generation etc.

Safety aspects – This must define the aspects of the process that have to do with safety or functional safety.

Techniques – Here we must define which techniques are we using for fulfilling the Functional Safety requirements. They will be chosen from the tables provided by ISO 26262 for the specified ASIL.

Artefact – Artefacts are basically the outputs of the process. Here we must define what will be achieved in the end of the process. For example, in every project, the Concept phase artefacts will be:

- Defined item
- Customer requirements
- Safety goal
- Functional safety concept
- Concept model

2. V-cycle

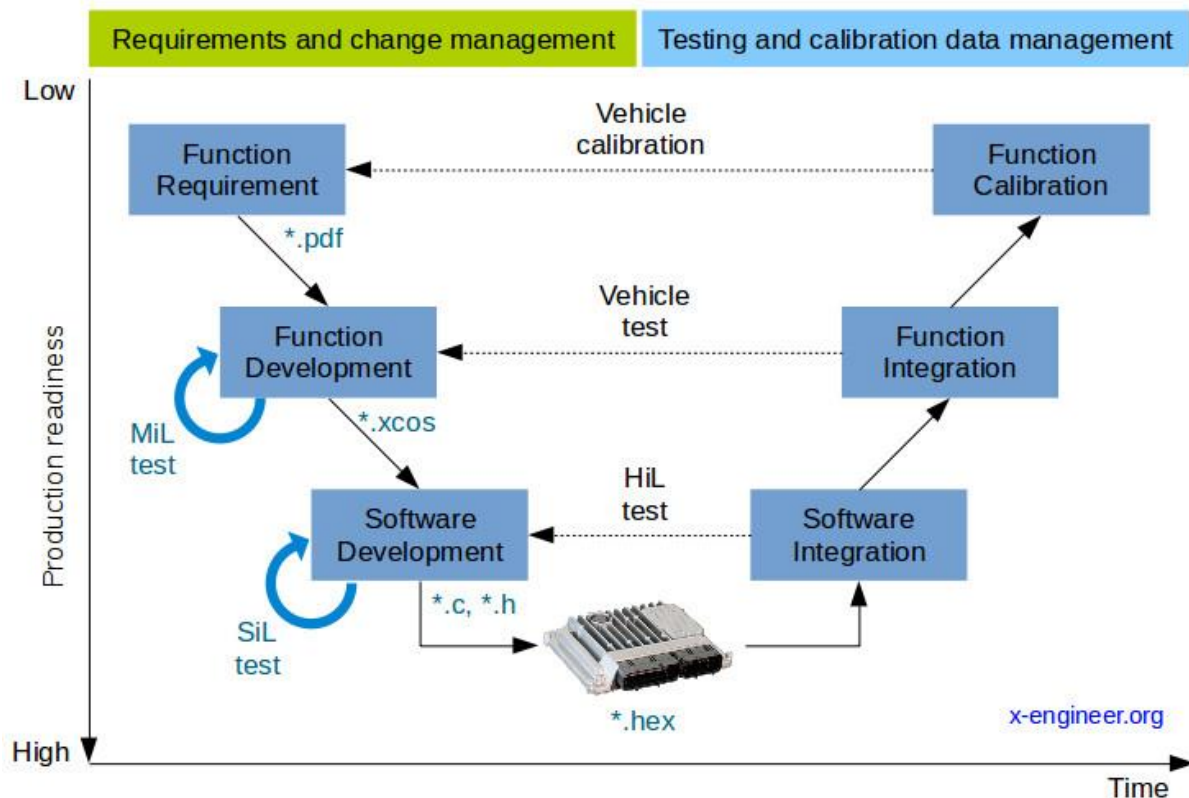


Figure 5. V-cycle for software [6]

[7]

As mentioned in the beginning, every company is trying to develop a systematic procedure to approach a problem. The standard software development process used in the automotive industry is the **V-cycle**. V-cycle is divided in 3 major categories which are;

- German V-Modell
- US government V-cycle
- General testing V-model

In our thesis, we will only discuss German V-cycle and use it to develop our own method.

German V-Modell

The V-Modell is a model for planning and realizing Projects. The V-Modell improves project transparency, project management and the probability of success by specifying concrete approaches with the respective results and responsible roles. It describes “Who” has to do “What” and “When” within the project. The V-modell was first introduced in 1997 [8] for civil and military agencies. Since, then due to the rapid advancement of automation, this model was updated to adapt to the new technological developments. The V-Modell introduced in 1997 was updated in 2004. Following things were incorporated in that model;

- Project-specific and organization-specific adaptability, applicability within the scope of the project, scalability to different project sizes and changeability and growth potential of the V-Modell itself.
- Consideration of the state-of-the-art of technology and adaptation to current regulations and standards
- Extension of the application to the entire system life cycle already during the development
- Introduction of an organization-specific process for improving process models

Objectives of V-Modell

The objectives of V-Modell are described as follows;

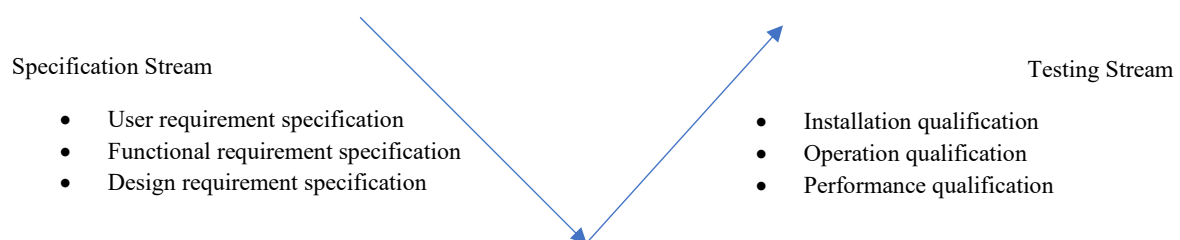
- Minimization of project risks
- Improvement and guarantee of quality
- Reduction of total cost over project and system life cycle
- Improvement of communication between stake holders

Basic V-Cycle components

The V model splits the software development process into two main phases. The left side of the V is the part of requirement analysis, function/software design and change management. The right side of the V concentrates the main verification and validation activities. The left side of the model can also be termed as validation while the right side can be termed as verification.

Validation; The assurance that product, service or system meets the need of the customer and other identified stake holders. It often involves acceptance and suitability with external service.

Verification; Evaluation of whether a product, service or system complies with regulation requirement specification or imposed condition. It is often an internal process.



Systems engineering

This approach can be traced back to 1940. It has many definitions depending on its uses but the classical one is;

“An interdisciplinary approach to translating users' needs into the definition of a system, its architecture and design through an iterative process that results in an effective operational system. Systems engineering applies over the entire life cycle, from concept development to final disposal”.

The definition used in our project can be represented by the following figure;

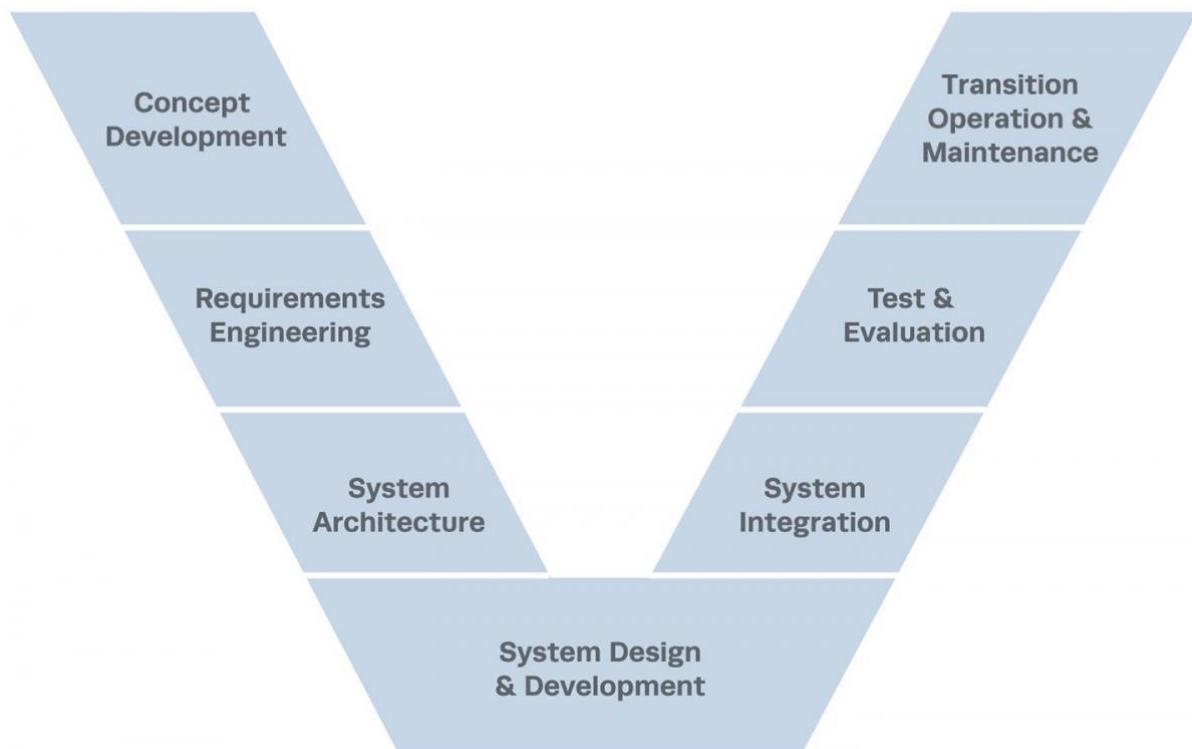


Figure 6. Systems engineering

[9]

This is a common graphical representation of the system engineering life cycle. The left side of the V represents concept development and the decomposition of requirements into functions and physical entities that can be architected, designed, and developed. The right side of the V represents integration of these entities (including appropriate testing to verify that they satisfy the requirements) and their ultimate transition into the field, where they are operated and maintained.

But this systematic approach does not enforce or mentions any safety checks, or it does not incorporate functional safety concept inside the development process. There are tests available which are specific for each V-cycle, but they are not standards.

3. Model Based Design

[10]

Model-Based Software Development is an embedded software initiative where a two-sided model is used to verify control requirements and that the code runs on target electronic hardware. One side is the Control Model, representing the embedded software of the system. The architecture of the embedded software is modeled with blocks containing algorithms, functions and logic components. Compiled software is auto generated from this model. The other side is the Plant Model, representing the physical aspects of the system. Each block contains mathematics that allows it to emulate the behavior of that physical item.

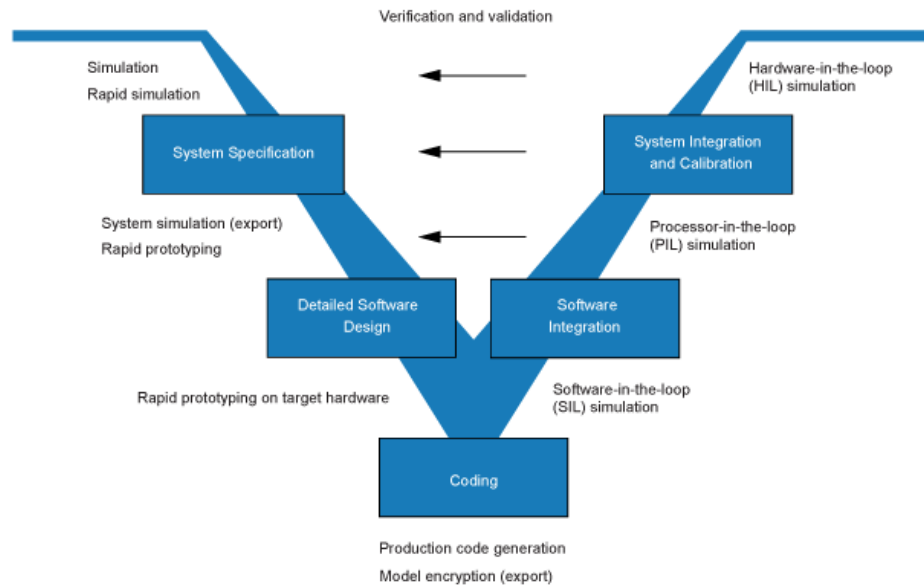


Figure 7. V-Model for System Development and types of Simulation [11]

In the left part of the V-Model we have different types of Simulation and Prototyping that are:

- Simulation
- Rapid Simulation
- Rapid Prototyping
- Rapid Prototyping on Target Hardware

On the other hand, in the right side of the V-Model we have In-the-Loop testing that are:

- Software-in-the-loop
- Processor-in-the-loop
- Hardware-in-the-loop

Depending on what we want to simulate or test, we must choose the right target environment that can be:

- Development computer
- Real-time simulator
- Embedded microprocessor

SIL testing is done to verify the automatically generated source code and runs on development computer and it is not in real-time.

PIL testing is needed to verify the object code and can be run either on embedded hardware or development computer with Simulink and an IDE. It is also not real-time.

HIL in the other hand is done to verify overall system functionality. It executes on the target hardware and it is real-time.

These all stages will be described further when we are describing how our method works and where we are using it in our system.

The most important reason of using this type of approach is the ability to get it standardized. Since, all the code is autogenerated and the blocks used are Simulink with no self-defined function, hence, it's easy to get it standardized by regulatory bodies.

Concept Model – should grasp and show the behavior of the main tasks either separately or together. The technology is still not defined, except for analog/digital.

Technical Model – should exhibit the main technical aspects, i.e. the sampling rate and the quantization levels, the saturation levels as well as other non-linearities. In addition, it must define the overall logic, i.e. states, transitions between states, tasks associated with states.

Production Model - is an adaptation of the Technical model with the blocksets provided by the VMU and/or RCP manufacturers in order to generate and download code for their hardware.

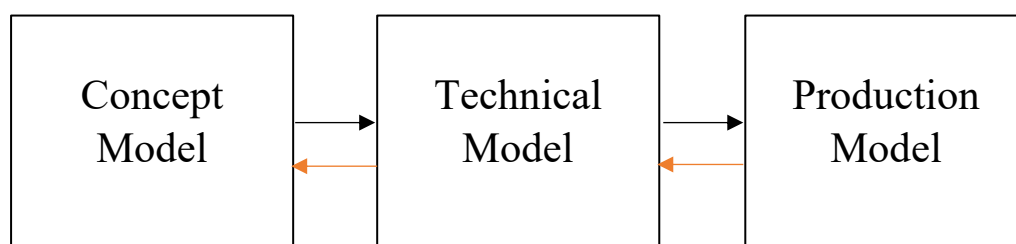


Figure 8. Model based design model flow

Concept Model simulation results allow to proceed to the assembly of the Technical Model whose simulation results (compared to the Concept Model) allows to issue hardware and software requirements to procure a suitable VMU and/or RCP platform. Once procured it is possible to proceed with the “code production” according to the specific platform.

If simulation results of Technical model are wrong, or something done in Concept Model is not possible in Technical Model, we must go back to Concept Model and do the needed modifications. In more serious problems, the Customer requirements might be also modified and Customer must be notified.

If results of Production Model are wrong, or something done in Technical Model is facing problems to be implemented for a certain VMU or RCP, we must go back to Technical Model and make the required changes.

3.1. MAAB Guidelines¹

The Mathworks Automotive Advisory Board (MAAB) developed certain guidelines for using MATLAB, Simulink, Stateflow and Embedded coder to meet the requests from its key automotive industry customers such as Ford, Daimler Benz and Toyota and now involves the major part of automotive industry. MAAB Guidelines can be:

- Global MAAB
- JMAAB (Japan)

Since we are not specifically targeting the Japan automotive industry we will be using Global MAAB version 3.0. The objective of MAAB Guidelines are:

- System integration without problems
- Well-defined interfaces
- Reusable models
- Readable models
- Professional documentation
- Fast software changes
- Easy exchange of models
- Understandable documentation

The guidelines given by MAAB can be rated as three different priorities:

- Mandatory
- Strongly recommended
- Recommended

Mandatory guidelines are those guidelines that all companies agree that are absolutely essential.

Strongly recommended guidelines are those guidelines that are agreed upon to be a good practice. Models should conform to these guidelines to the greatest extent.

Recommended guidelines are those guidelines that are recommended to improve the appearance of the model diagram but are not critical.

Since our tools that we will use on this work are from MATLAB, Simulink, Stateflow and Embedded coder we must strictly apply the *Mandatory* guidelines and as most as possible two other priority rates. In the following, we will show some examples of MAAB Guidelines.

A *Strongly recommended* requirement is the position of the block names. They must be located below the block, as in figure shown below:

¹ Mathworks.com

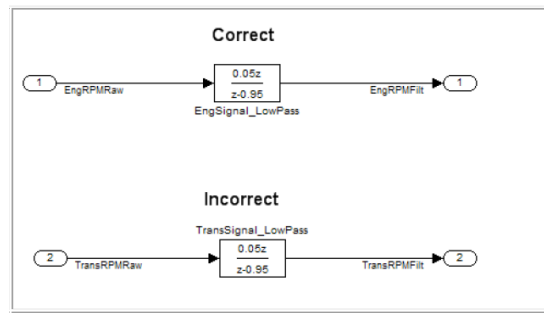


Figure 9. MAAB guideline for Simulink modelling example [11]

A *Mandatory* requirement that we shall apply is the block that are not allowed to be inside controllers as the figure below shows:

ID: Title	jm_0001: Prohibited Simulink standard blocks inside controllers
Priority	mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	<ul style="list-style-type: none"> Control algorithm models must be designed from discrete blocks. The MathWorks "Simulink Block Data Type Support" table provides a list of blocks that support production code generation. <ul style="list-style-type: none"> Use blocks that are listed as "Code Generation Support". Do not use blocks that are listed as "Not recommended for production code" – see footnote 4 in the table. In addition to the blocks defined by the above rule, do not use the following blocks

Figure 10. Prohibited blocks inside controllers [11]

Also, naming the files is very important and *Mandatory* according to MAAB, and they should be names as shown in the figure below:

ID: Title	ar_0001: Filenames								
Priority	Mandatory								
Scope	MAAB								
MATLAB Version	All								
Prerequisites									
Description	<p>A filename conforms to the following constraints:</p> <table border="1"> <tr> <td>FORM</td><td>filename = name.extension name: no leading digits, no blanks extension: no blanks</td></tr> <tr> <td>UNIQUENESS</td><td> <input type="checkbox"/> all filenames within the parent project directory <input type="checkbox"/> cannot conflict with C / C++ or MATLAB keywords </td></tr> <tr> <td>ALLOWED CHARACTERS</td><td> name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ extension: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 </td></tr> <tr> <td>UNDERSCORES</td><td> name: <ul style="list-style-type: none"> can use underscores to separate parts cannot have more than one consecutive underscore cannot start with an underscore cannot end with an underscore extension: <ul style="list-style-type: none"> should not use underscores </td></tr> </table>	FORM	filename = name.extension name: no leading digits, no blanks extension: no blanks	UNIQUENESS	<input type="checkbox"/> all filenames within the parent project directory <input type="checkbox"/> cannot conflict with C / C++ or MATLAB keywords	ALLOWED CHARACTERS	name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ extension: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9	UNDERSCORES	name: <ul style="list-style-type: none"> can use underscores to separate parts cannot have more than one consecutive underscore cannot start with an underscore cannot end with an underscore extension: <ul style="list-style-type: none"> should not use underscores
FORM	filename = name.extension name: no leading digits, no blanks extension: no blanks								
UNIQUENESS	<input type="checkbox"/> all filenames within the parent project directory <input type="checkbox"/> cannot conflict with C / C++ or MATLAB keywords								
ALLOWED CHARACTERS	name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ extension: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9								
UNDERSCORES	name: <ul style="list-style-type: none"> can use underscores to separate parts cannot have more than one consecutive underscore cannot start with an underscore cannot end with an underscore extension: <ul style="list-style-type: none"> should not use underscores 								
Rationale	<input checked="" type="checkbox"/> Readability <input type="checkbox"/> Verification and Validation <input checked="" type="checkbox"/> Workflow <input checked="" type="checkbox"/> Code Generation <input checked="" type="checkbox"/> Simulation								
Last Change	V3.00								

Figure 11. MAAB for filenames [11]

For signal naming with a priority *Strongly recommended*, a signal name [11]:

- should not start with a number
- should not have blank spaces
- should not have any control characters
- should not return carriage returns
- underscores can be used to separate parts
- cannot have more than one consecutive underscore
- cannot start with an underscore
- cannot end with an underscore

4. Hybrid V-Cycle

The purpose of creating a Hybrid V-Cycle comes from the need of integrating ISO 26262, Model based design flow in V-Cycle.

ISO26262 into V-cycle

In our project we are mapping these points on the V-cycle for automotive safety in order to make V-cycle coherent with functional safety rules of ISO 26262. The following picture shows our concept,

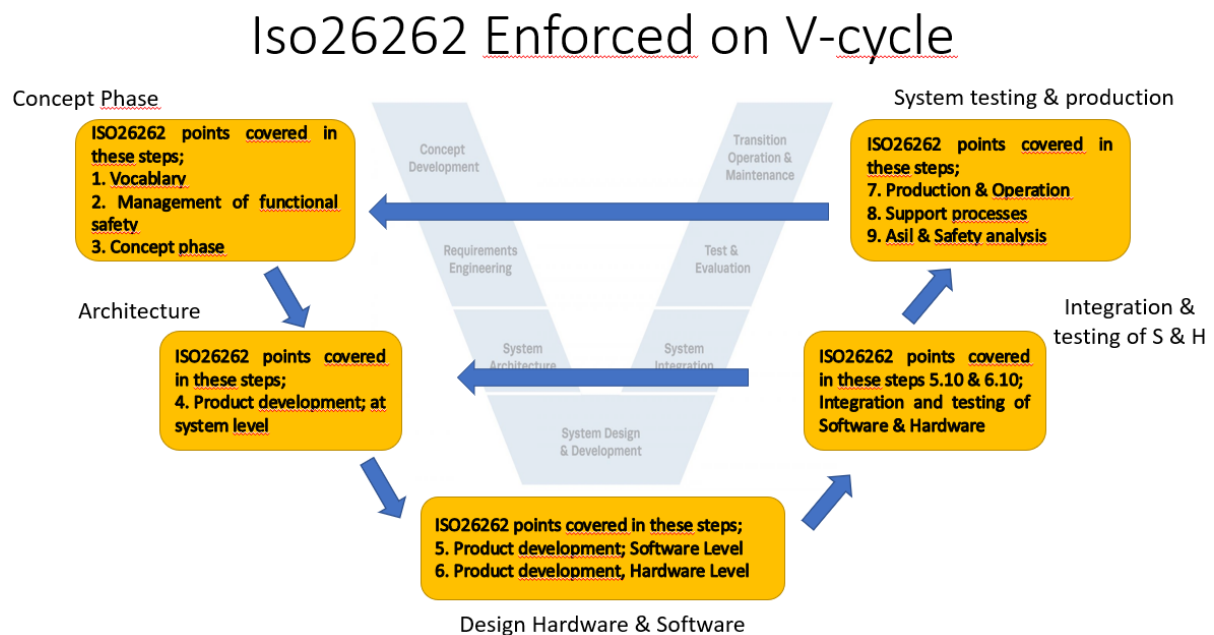


Figure 12. Iso26262 Enforced on V-cycle

Hybrid V-Functional safety concept

As can be seen from the figure 5 that we have mapped iso safety points on the V-Cycle. Now, we will mention our own V-cycle which shows the functional safety concepts already incorporated inside the model-based design.

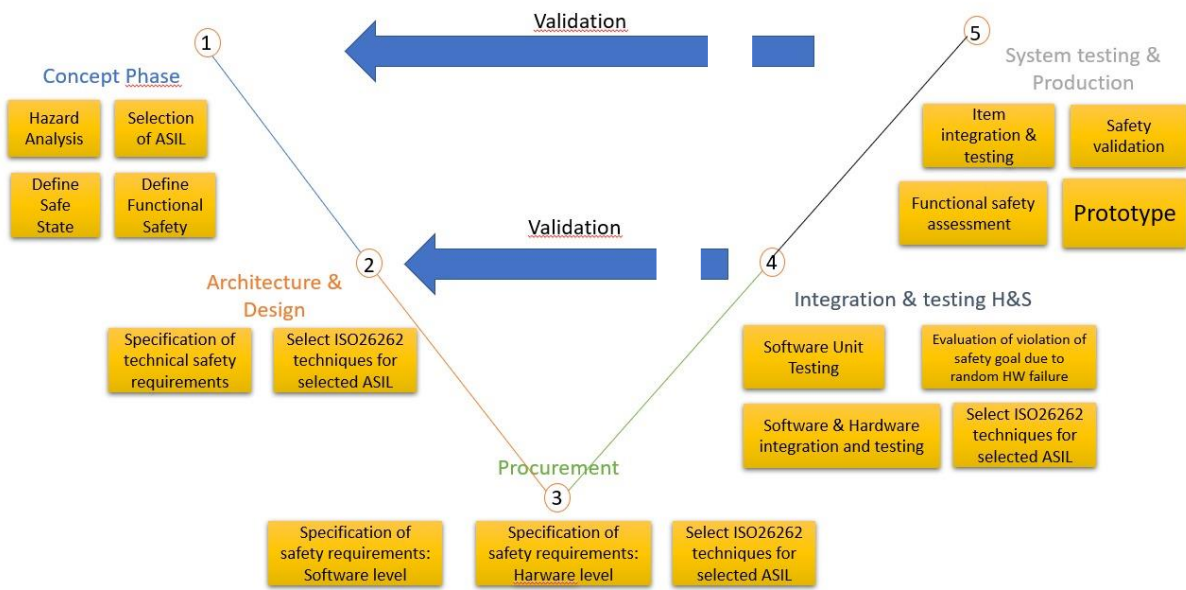


Figure 13. Hybrid V-functional safety concept

Steps for the Hybrid V-Functional safety cycle

The figure 6 can be further defined in order to give us in-depth information about the whole process. We will divide the process mainly in three different categories which includes company, customer and supplier. The following diagram shows the interaction between them,

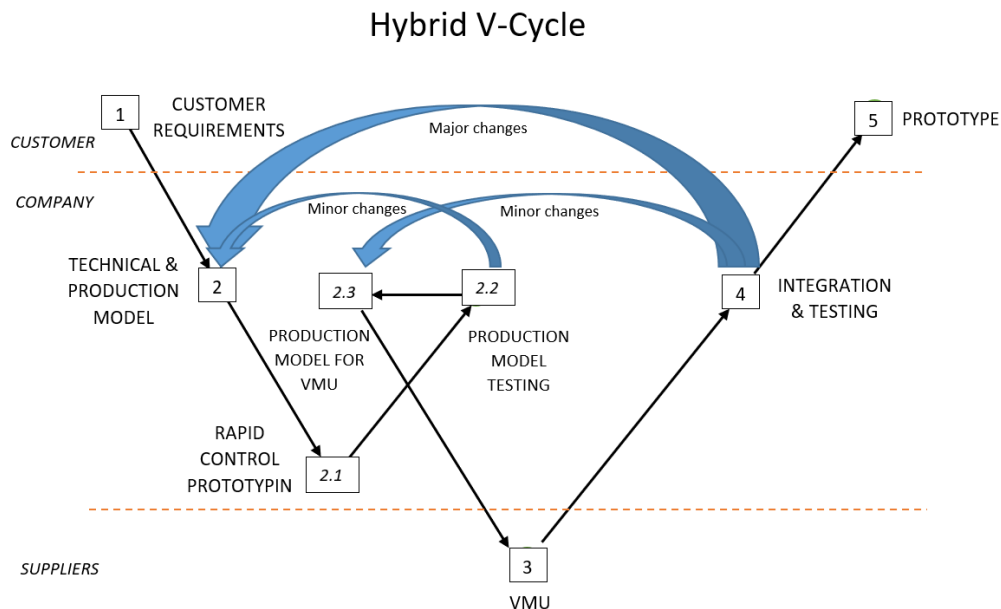


Figure 14. Steps for hybrid V-cycle

All the numbers on the diagram shows the points we follow to reach result. Two lines dividing the customer, company and supplier. The keyword's we want to cover in this concept are following,

- Functional safety
- V-Cycle process
- Model based design
- Modularity
- Reusability

The first three keywords were incorporated when we were talking about the figure 6. We included all the concepts related to first three topics. In order to make code easy and reusable we are using the architecture which is divided by certain defined interfaces which helps us to make our code modular. We know all types of inputs and outputs so we can easily replace the block between the interfaces to get the required function from the model.

Explanation for the steps of Hybrid V- cycle

No. Of Point	Models in the Nodes
1	Customer Requirements
2	Technical & production model
2.1	Rapid Control Prototype (RCP)
2.2	Testing of production model on dspace
2.3	Production model for VMU (vehicle management unit)
3	Vehicle Management Unit
4	Integration and testing
5	Prototype
-	Major changes
-	Minor changes

Table 4. Steps for Hybrid V-cycle

Now, we will explain the steps we mentioned in figure 7.

1. Customer requirements

Customer can provide us with the requirements or some model which we follow when developing technical models inside the company. This step is important because we understand all the requirements set up by the customer. After understanding the requirements, we interpret it and work on them to find out the best possible solution for the problem within the limits set by the customer.

2. Technical and production model

After specifying all the customer requirements, in technical model step, we focus on the making of simplest model as possible according to our understanding of the requirements laid

down by our customer. After making the basic technical model we need a production to set all the parameters in order to run it on our rapid control prototype platform to quickly lay out the specifications for our vehicle management unit.

3. Rapid control prototype

Rapid control prototyping is a very efficient method to develop, optimize, and test new control strategies in a real environment quickly without manual programming [12]. After developing the model based on the requirements put down by the customer, we should run our model on this rapid control prototype in order to fine tune requirements and see how the program is working in this environment.

4. Testing of Production model on dSpace

Production model has a lot of flexibility and room for improvement. We need to optimize the model for code production and see how many bits are required to give us satisfactory results. We need optimization in order to reduce the memory of our code, hence, the cost of our vehicle management unit. So, the production model and rapid control prototype gives us the requirement for our vehicle management unit.

Minor changes

While testing the model on our rapid prototyping platform we are unable to reach a conclusion or if the model is not producing the results desired by the customer, we have to go back again to the second step which is “technical and production model”. We change the model so that we can make it function as desired by the customer.

Moving from the fourth step to second one, costs nothing. Since we haven’t purchased anything and everything up-till now is on the software. So, we can iterate it as many times as we like considering the requirements from the customer. The main advantage of introducing rapid control prototype is to see whether the chosen equipment is suitable for this application or we need further improvements to reduce cost while maintain the same functionality.

5. Vehicle Management Unit selection

After the specifications laid down by Dspace we will order the VMU from our vendors. We will move to this step after finalizing the model. If we need certain changes in the technical and production model, we will move straight to second point.

6. Vehicle management unit

This step will be performed outside the company. We will set up the requirements we need for our VMU. These requirements will be passed on to our vendors and vendor will be chosen accordingly.

7. Integration and testing

After getting the VMU from our supplier we will integrate our code with hardware and test it in different environments. The most important test is of fault injection in which we deliberately inject a fault in the system and see how robust our code is. After testing of our system if everything goes well then, we can move on to the next stage which is laying down the final product requirements. But if we are not able to produce the desired results, we have to go back to fourth step or all the way back to second step.

Minor changes

At the seventh step, we are testing on the real board and we have already bought this from the vendor. So, if we change it then we have to pay some damages but since we haven't mass produced the system we still can go back to testing our model on rapid control prototype to change interfacing between the VMU and sensors to make it more efficient. It's not recommended to change after you have bought the VMU from the vendor but if the system fails under fault injection system and it could be easily replaced by small changes then it is still feasible.

Major changes

If at the seventh step while testing on the real hardware we have problem which is related to the understanding of basic requirements, then we must go back the second step which is technical and production model. This loop costs the same as the minor change after the seventh step but it means that we have not understood the requirements well enough and have to revise those or to come up with new model to satisfy customer needs. The hybrid V-Cycle helps us to standardize the procedure and it makes management of the project easy. Even if we have gone to the second step, to move forward we can not skip any step in between, and we must follow the procedure again.

8. Final product requirements

After integration and testing is successful, we will set out the product details for the customer.

5. Innovative Modular Architecture

The concept of modularity and reusability can be explained by the following diagram. Figure from the notes of the prof:

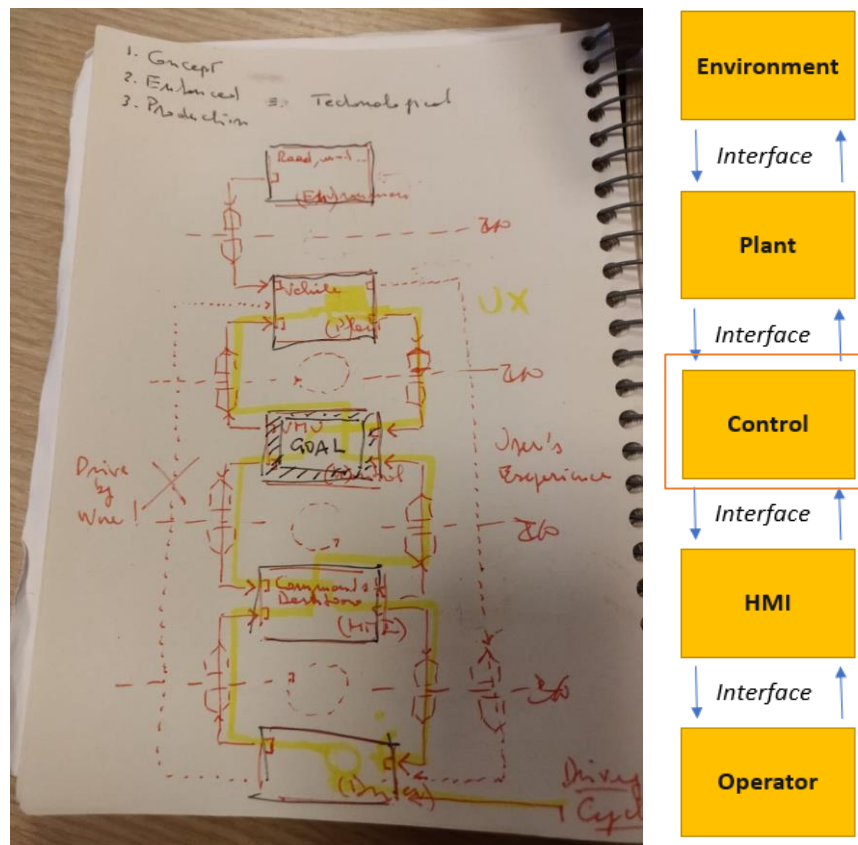


Figure 15. Reusability and modularity concept

Figure 15 shows dotted lines which helps us to divide blocks in different sections. When we are talking about the modularity, we mean that we can replace the block and then test it for some other project. The process remains the same but whatever is in the blocks, by changing it we can change our target.

1. Environment

This block represents the external environment for our model. For example, if we are discussing about electronic circuits then we should consider the electromagnetic interference in our circuits from the external environments. We can simulate all the external influences in software (Simulink). This block is used to simulate the actual environment as close as possible to the real environment but only on the software. Every software has some restrictions, so we try to be as close as to the real environment. For example, if we are generating signal, we add noise in the signal to reproduce the external affects. It affects the plant hence we have drawn signs in interference from this block to plant.

2. Plant

This block represents inputs we provide to control in order to make the decisions. This block is also simulated in the software.

3. Control

It is the main block in our scheme. It takes inputs from the plant and issues output to execute actions based on inputs. It also contains inputs from human machine interface. Our whole algorithm to control system is executed in this control block.

4. Human machine interface

This block in software represents interaction between human and machines. Each software gives us some controls which can reproduce actual human machine interaction. In software, it is represented by buttons and switches which are in the software only and they don't have any physical presence. You can choose the type of button and set some parameters to mimic actual behavior.

5. Operator

The programmer performs function of the operator. In real world, operator will input commands while here since everything is on the computer, hence the person controlling computer will be considered as operator.

6. Interface

These things define the connection between two blocks. In the above method we are connecting software with software, so interfaces are represented by just connections in the software.

5.1. Interfaces

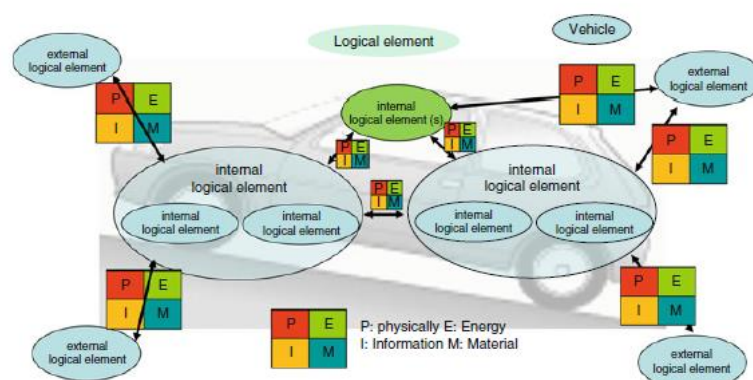


Fig. 3.9 Multi-dimensional boundary and interface analysis (Source Derived from Ford-FMEA-Handbook)

Figure 16. Interface analysis [13]

According to the Ford-FMEA-handbook there are four kinds of interfaces. [13]

- *Physical interface*
- *Energy interfaces*
- *Material transfer (interface)*
- *Information interfaces*

5.2. Concept of reusability and modularity in Hybrid V-cycle

In order to introduce this concept, we introduce the blocks. As shown in figure 8, we have divided the procedure in some blocks. We are going to define each block in order to understand the whole procedure.

Software block

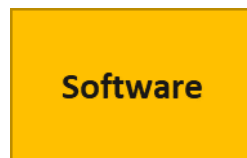


Figure 17. Software blocks definition

This block represents the software portion of our Hybrid V-cycle. software will include anything which is not present physically, but it is designed and tested on computer. There is no interaction between physical parts. We design our systems and satisfy all the requirements virtually on a computer.

Hardware block



Figure 18. Hardware blocks definition

This block represents the hardware portion of our Hybrid V-cycle. In this part, we have a physical equipment. We are no more working on the software which is all inside computer. This hardware block can include VMU, RCP, input from environment and all the sensors. VMU and RCP are included in the hardware block but in order to define the process in a better way we are going to highlight them separately just to identify the steps where we are introducing VMU and RCP.

Rapid Control Prototype Block



Figure 19. RCP representation of hardware block

Rapid control prototype is a part of hardware block. We have represented it in a different way just to clarify the steps where we are using RCP.

Vehicle Management Unit



Figure 20. VMU representation of hardware block

Vehicle management unit is also a part of hardware block. We have represented it in a different way in order to clearly identify the steps where we are using VMU.

5.3. Architecture impact in integration and testing

As mentioned earlier that our procedure contains 5 important points which are as follows:

- Functional safety
- V-Cycle process
- Model based design
- Modularity
- Reusability

Now, we will link the modularity and reusability concepts with other concepts of functional safety, V-cycle process and model-based design. To explain it we have divided it in 3 parts. The first one is

- Software in the loop
- Hardware in the loop containing Rapid control prototype
- Hardware in the loop containing VMU in the loop

Software in the loop

Simulink (2° step)

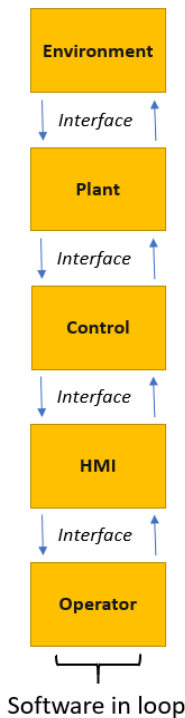


Figure 21. Software in the loop

This step is covered by 2nd step of our Hybrid V-functional safety cycle. In this step, we are going to develop our technical and production model based on the requirements demanded by our customer. This is the general scheme of our methodology in which we are going to divide our model into 5 main blocks. All these blocks are simulated in the software and at this stage no hardware is involved. The blocks are:

- Environment
- Plant
- Control
- Human machine interface
- Operator
- Interface

Hardware in the loop containing Rapid control prototype

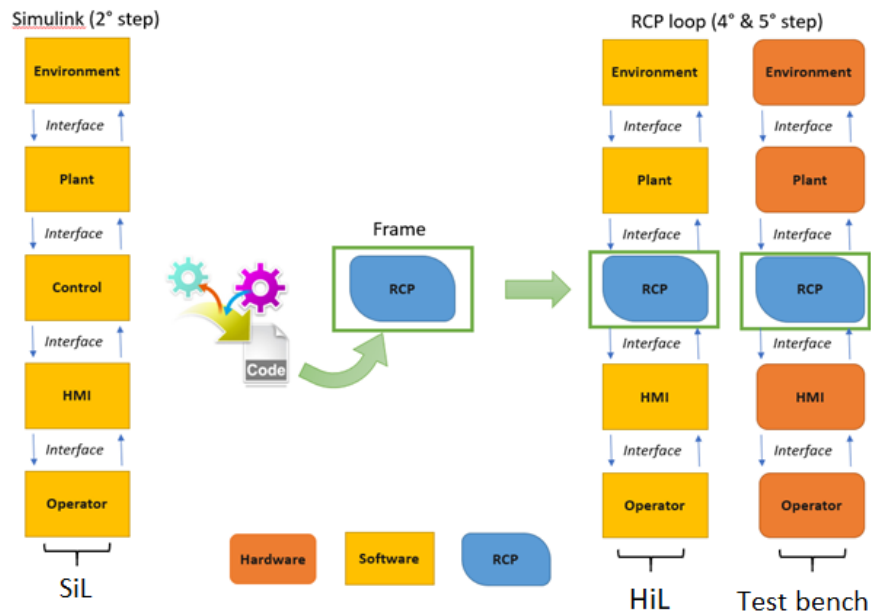


Figure 22. Hardware in the loop containing rapid control prototype

This step is covered by 4th and 5th step of our Hybrid V-functional safety cycle. As shown by the above diagram, after software in the loop we are doing code generation for our control block.

Code generation

We have the control block in software. In order to run it on our rapid control platform we convert it to a code. This process of code generation is handled automatically by software which produces C or C++. This automatic code is not optimized and in the following steps we will first try to run code on our rapid control prototype which can handle a large code size and is only introduced to check our control block performance and robustness. This equipment helps us to fine tune our control block and check for any potential errors.



Figure 23. Code generation

Frame

Every RCP requires some timers and assignment of ports which will help other blocks to communicate with it. We develop the frame in the next step to make sure that interfaces interact smoothly with RCP.

Frame

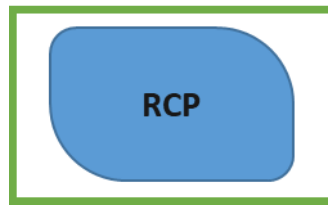


Figure 24. Defining frame

Rapid Control Prototype Loop (4th & 5th step)

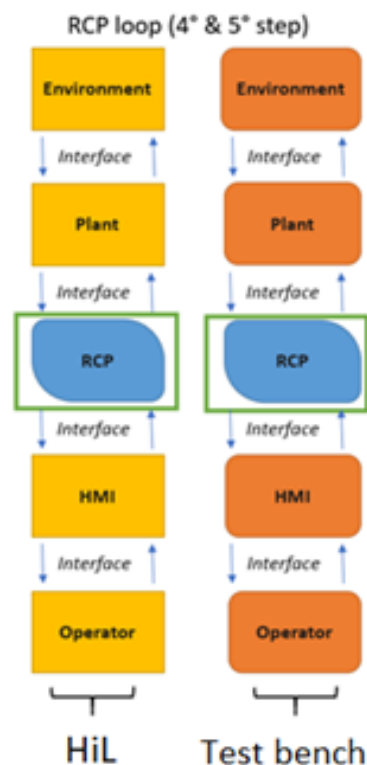


Figure 25. RCP loop

After constructing frame and placing code inside our rapid control prototype we replace the control block (referring to the left column) with rapid control prototype. After testing this configuration, we will observe how our control block is performing. We should make a distinction here, the yellow blocks on the left column with names, environment, plant, HMI and operator are all in the software. They are still controlled by computer which is connected to RCP which is a physical equipment with generated code running inside it.

The right column has different sets of blocks. In this step, we have replaced all the software with hardware blocks. In the previous step, we were controlling everything from the computer. All the blocks except from RCP were not physically present. In this step, which is shown by column on right side of the diagram, we are going to replace all the software blocks with the hardware. All the inputs will be from hardware blocks. The operator will be a real person operating system with the HMI. While plant will be our sensors which will monitor the values

and give it as an input to RCP. The environment will be everything surrounding equipment, which is affecting the system.

Hardware in the loop for vehicle management unit

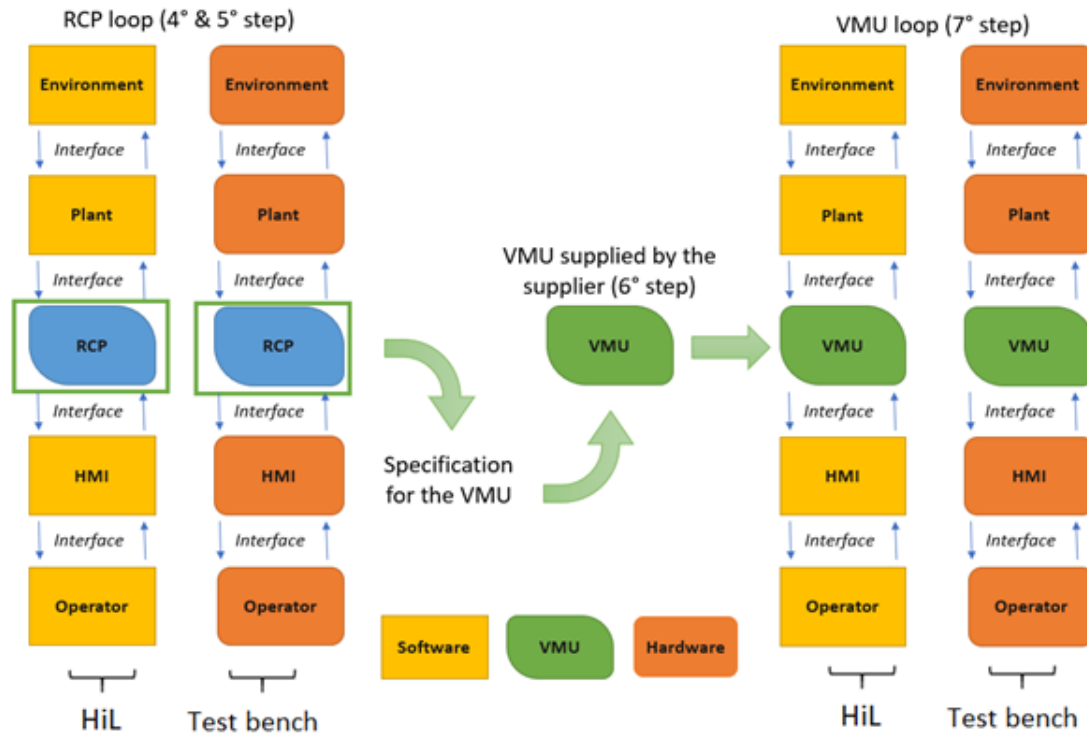


Figure 26. Hardware in the loop for VMU

This step is covered by 7th step of our Hybrid V-functional safety cycle. As shown by the above diagram, the RCP gives us the specification of the VMU. It tells us the specifications of memory and other aspects of VMU. We need these aspects in order to select the vendor which will give us the best product at reasonable rates. The RCP also tells about the performance. So, instead of buying different VMU we will set the requirements set by RCP by running the code at various bit rates, to meet the performance requirements set by the customer, keeping in mind the safety aspects of our operation. If we can reduce the power requirements of the VMU we will be able to reduce the cost of purchase. VMU is supplied by the supplier which is represented by step 6th in our Hybrid V-functional safety cycle.

VMU loop (7th step)

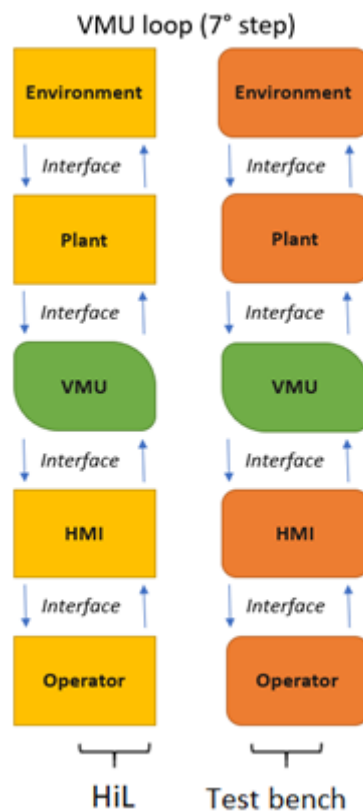


Figure 27. VMU loop

As seen before, we will apply the same procedure as applied before in the RCP loop. We have 2 columns which are included in step 7th of our Hybrid V-Functional safety cycle. Column on the left side shows the integration of software in VMU and then testing it on a test bench. In this test bench, we have all the blocks in software except from the VMU which we got from the supplier. We will run the code first with this configuration to check the performance of the VMU and to make sure everything is in order and after that we will replace all the software blocks with hardware to do the final tests before finalizing the solution provided to the customer.

Flexibility in model

The procedure explained above is one of the many combinations that could be adopted in order to obtain the desired results. Now, we will explain some of the other combinations of the same procedure. For example, if we are considering the software in the loop we can have many different combinations of it.

Different combinations of software in the loop,

We will discuss some of the combinations here in order to show the flexibility of our Hybrid V-Cycle.

Simulink (2° step)

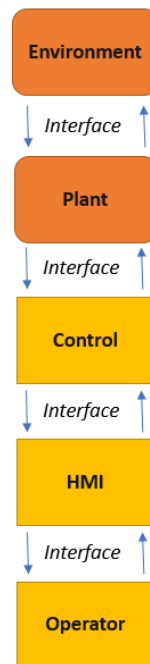


Figure 28. Combinations of software in the loop

Here we can see that two software blocks are replaced by two hardware blocks. The environment is an actual environment while the plant is also considered as a hardware block. In some cases, we are using sensors to take input from the outside and then in order to process and control it we are using software. We can replace any block with hardware except from the control block since it's an early to invest on a controller. Some other combinations of the software in the loop model can be seen below.

Simulink (2° step)

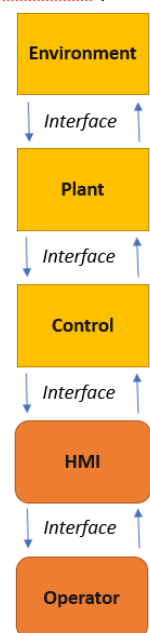


Figure 29. Combinations of software in the loop

In figure 21, we have replaced the human machine interface with actual buttons and a human is controlling that panel to produce the results.

Simulink (2° step)

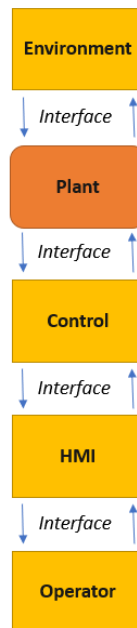


Figure 30. Combinations of software in the loop

In the figure 22, we replaced plant with a hardware block while other blocks are still in software.

Proposed Combinations of Hardware in the Loop Containing Rapid Control Prototype

One of the combinations is explained in figure 17. We can consider other combinations also which are:

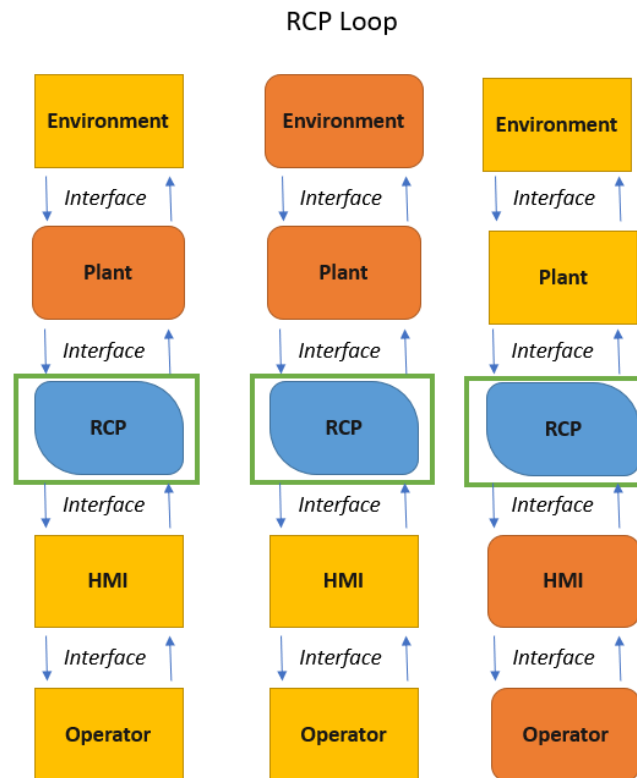


Figure 31. Proposed combinations of hardware in loop for rcp

Comparing figure 17 with 23 shows that even in the 5th step we can have a software block. This strategy makes our Hybrid V-cycle more flexible and it could be easily adapted to different conditions depending upon our requirements.

In figure 23, the RCP is also considered as a hardware block, but we have mentioned it with different color to clearly identify this step. We must clearly define the interfaces when we are moving from software block to hardware block. We should keep in mind what are the requirements of hardware and software block. If the interfaces are not properly defines then it is impossible for the blocks to interact with each other.

Proposed Combinations of Hardware in the Loop for Vehicle Management Unit

As explained in the figure 19, about hardware in the loop for vehicle management unit, we can thing of other combinations also and make a hybrid model to satisfy our requirements. We will see an example to understand how it might work.

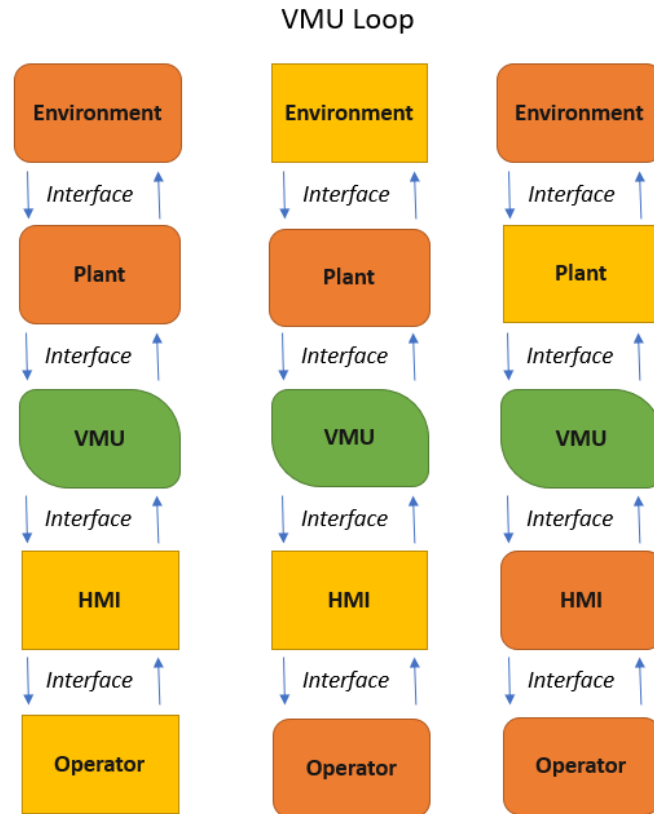


Figure 32. Proposed combination of hardware in the loop for VMU

We will give the brief overview of whole process again and in following chapters we will take few examples for better understanding how the whole method works by considering actual examples, starting from a simple digital filter leading to the more complex problem.

1. Customer requirements

As described in the start, these are the requirements set by customer. It can be in the shape of requirements or a model. We completely understand it before starting to develop the model.

2. Technical and production model

After specifying all the customer requirements, in technical model step, we focus on the making of simplest model as possible according to our understanding of the requirements laid down by our customer. After making the basic technical model we need a production to set all the parameters in order to run it on our rapid control prototype platform to quickly lay out the specifications for our vehicle management unit.

In this step, we develop the model following ‘software in the loop’ and ‘model in the loop’ based on model-based engineering. We have nothing in hardware, every program is in software, for example Simulink. We modify the model according to requirements of customer. We also try to find out innovative solutions for satisfying requirements set out by our customer. There might be some requirements which cannot be incorporated in our model or even if they are incorporated, we pay a higher price for getting slightly better performance. At this point

everything is in the software, hence, it costs nothing to make any changes in the model. We can simply do it by a click of the button.

As described in figure 20-22, we can adopt different combinations of the software and hardware parts for making our system.

3. Rapid control prototype

Rapid control prototyping is a very efficient method to develop, optimize, and test new control strategies in a real environment quickly without manual programming [12]. After developing the model based on the requirements put down by the customer, we should run our model on this rapid control prototype in order to fine tune requirements and see how the program is working in this environment.

4. Testing of Production model on dSpace

Production model has a lot of flexibility and room for improvement. We need to optimize the model for code production and see how many bits are required to give us satisfactory results. We need optimization in order to reduce the memory of our code, hence, the cost of our vehicle management unit. So, the production model and rapid control prototype gives us the requirement for our vehicle management unit.

As mentioned above, if our model is not satisfying the customer requirements then we must go the second step again and follow the procedure again.

At the fourth step, we can have different combinations of software and hardware blocks. We can adopt the suitable combination of both of these.

5. Vehicle Management Unit selection

After the specifications laid down by Dspace we will order the VMU from our vendors.

6. Vehicle management unit

This step will be performed outside the company. We will set up the requirements we need for our VMU. These requirements will be passed on to our vendors and vendor will be chosen accordingly.

7. Integration and testing

After getting the VMU from our supplier we will integrate our code with hardware and test it in different environments. The most important test is of fault injection in which we deliberately inject a fault in the system and see how robust our code is.

As can be seen from the figure 26, if the integrations and testing is not successful then we again move either to step 4 or step 2 depending upon the changes we have to make. If we have to make minor changes we have to move from step 7 to step 4 while if we have to make a major change we have to move straight from 7th step to 2nd one and again we have to follow all the points leading up to 7th step again.

8. Final product requirements

After finalizing the testing and code we will set out the product details for the customer.

5.4. Standards and guidelines check

Our product has certification goals, thus through all the development we have put our effort to comply with them. Simulink gives us the tools to check if our model and generated code complies with the standards and guidelines we set before:

- ISO26262
- MISRA C
- MAAB Guidelines

This can be done in Simulink via Model Advisor:

- Simulink -> Analysis tab -> Model Advisor

For MISRA-C Model Advisor can run the check immediately, but for ISO26262 and MAAB Guidelines we must download the add-in Simulink Check™ that includes:

- ISO 26262
- IEC 61508
- IEC 62304
- DO-178
- MAAB Guidelines

For additional checks, we decided to run also IEC 61508 since it is the parent standard of ISO 26262, allowing us to target different fields in the future.

Also, an important thing to do for MISRA C certification is preparation of a compliance statement that is something we will not do in this thesis.

When using MISRA C:2012 coding guidelines to evaluate the quality of your generated C code, you are required per section 5.3 of the MISRA C:2012 Guidelines for the Use of C Language in Critical Systems document to prepare a compliance statement for the project being evaluated. To assist you in the development of this compliance statement, MathWorks® evaluates the MISRA C:2012 guidelines against C code generated by using Embedded Coder. The results of the evaluation are published as: [11]

- Compliance summary tables
- Deviations

An extra check for more robustness, we will use also ‘Code Generation Advisor’ that helps us to check:

- RAM Efficiency
- Traceability
- Safety precaution
- Debugging
- ROM Efficiency
- Execution efficiency

6. Documentation

Professional and reliable documentation is a must in every process on any field but especially in Automotive industry where the ever-increasing complexity of its processes demands an increase in documentation quality as well. There are several different software and tools for doing it but they come with a very high cost. Thus, we have come up with our very low-cost but efficient and clean approach using Dropbox and Dropbox Paper. The Dropbox account called “VMU Project” is divided on 4 different main sections that are:

- Development
- Documentation
- References
- Meeting Notes (Dropbox paper)

Development contains everything that concerns technical side of the project such as Simulink modelling. It also has 3 different folders:

- Concept model
- Technical model
- Production model

Documentation folder contains documentation of the development such as:

- Requirements
- Methodologies
- Safety lifecycle according to ISO26262
- Presentations to be presented in the meetings with the customer
- Results of every development step

References folder contains every reference that is used in the documentation and development.

Meeting Notes in the other hand plays a key role in keeping track of the work that has to be done, creates a very collaborative environment allowing every participant to comment on notes, sharing ideas, references, alerting everyone that a report, model or documentation is ready and can be found on one of the folders that we already explained. A snapshot from the Meeting Notes below shows it very clearly how we used it:

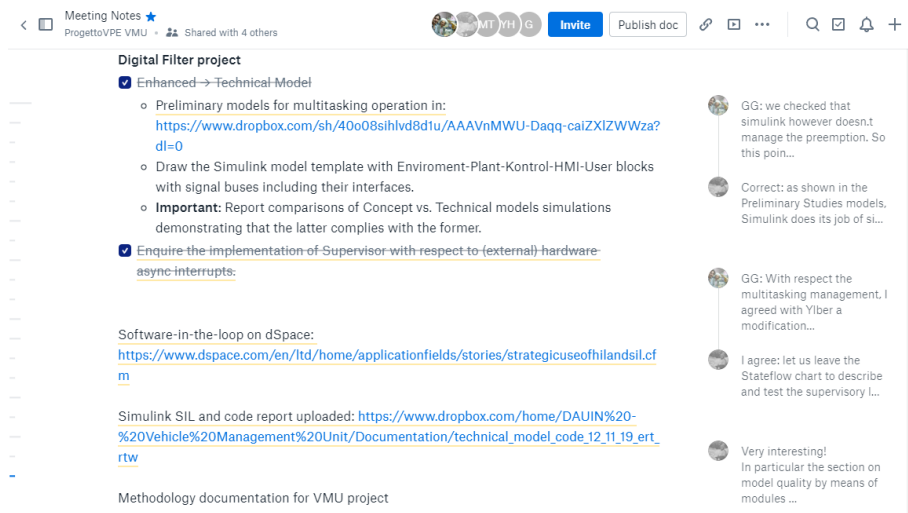


Figure 33. Meeting Notes snapshot

The illustration below sums up the architecture of Dropbox:

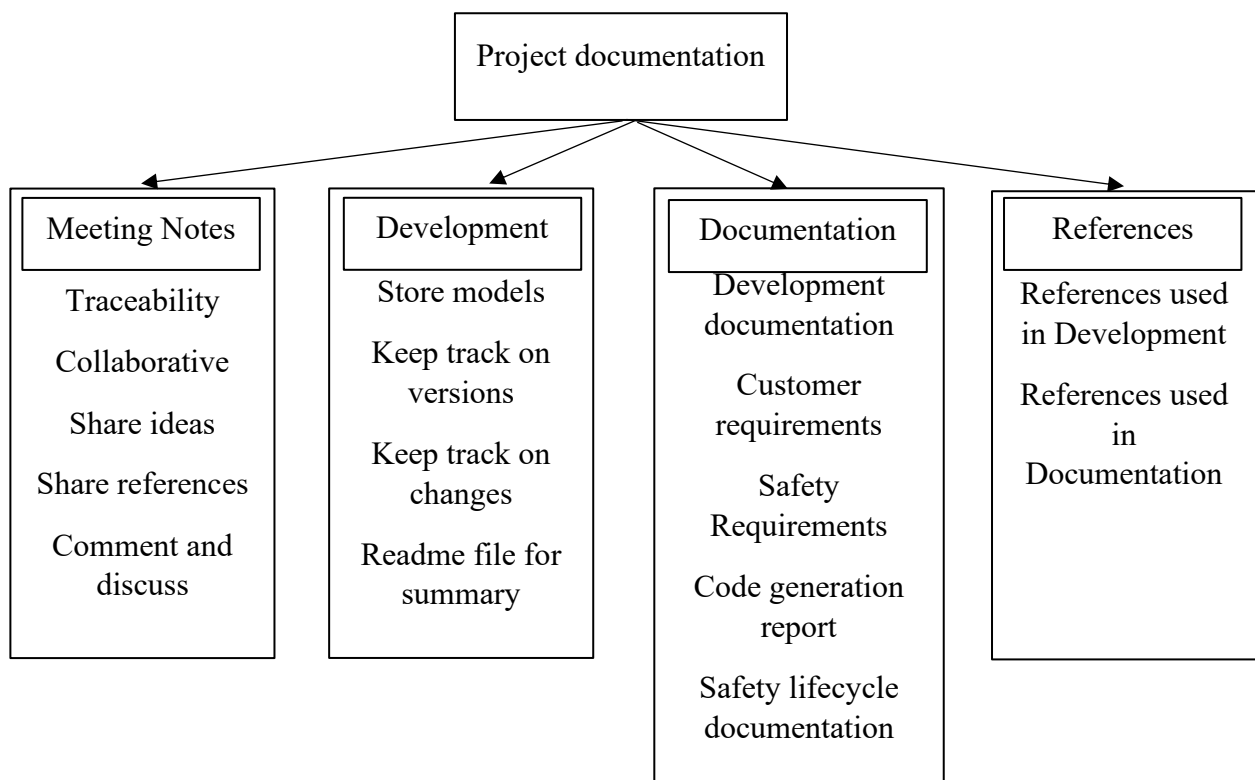


Figure 34. Documentation architecture

PART 1, CHAPTER 2
AN EXAMPLE

1. Customer Requirements

Artefacts of this process are:

- Defined item
- Customer requirements
- Safety goal
- Functional safety concept
- Concept model

Item definition

Objective:

Perform a Fast Fourier transform on the input signal and design a low-pass digital filter with certain requirements given by the customer. The system shall be designed in such a way that it must be easy to test and validate.

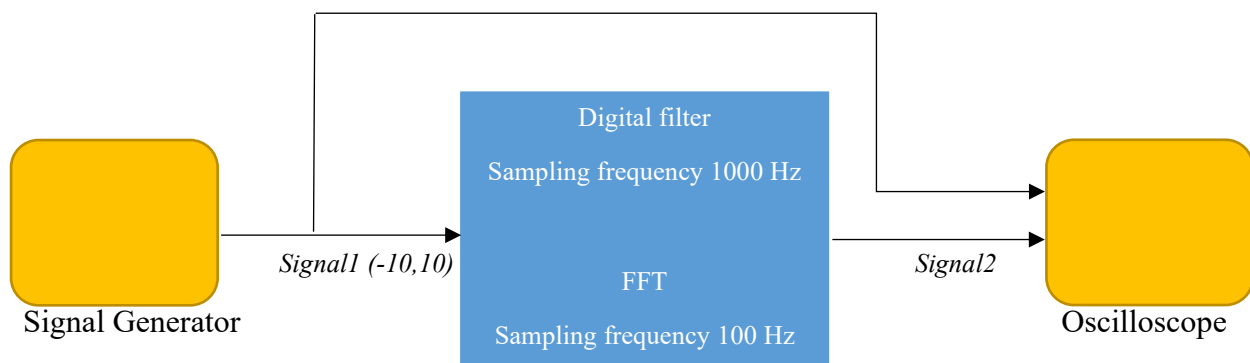


Figure 35. Digital filter and FFT block scheme

What is a Fast Fourier Transform?

When we want to decompose a signal that is composed by different signals with different frequencies into pure frequencies that they are made of, we apply a Fourier transform on it. A Fast Fourier Transform, in the other hand, is an efficient algorithm that makes us implement the Fourier transform in much faster way.

What is a Low-pass Digital Filter?

An analog low pass filter is a filter that passes analog signals with frequency below cut-off frequency. A digital low-pass filter is the same except that it acts on discrete-time signals. It is programmable, doesn't age and provides way higher performance than analog filter. Method, type and implementation of the low-pass digital filter will be discussed on the Analysis and Architecture phase. Layout of the item to be developed:

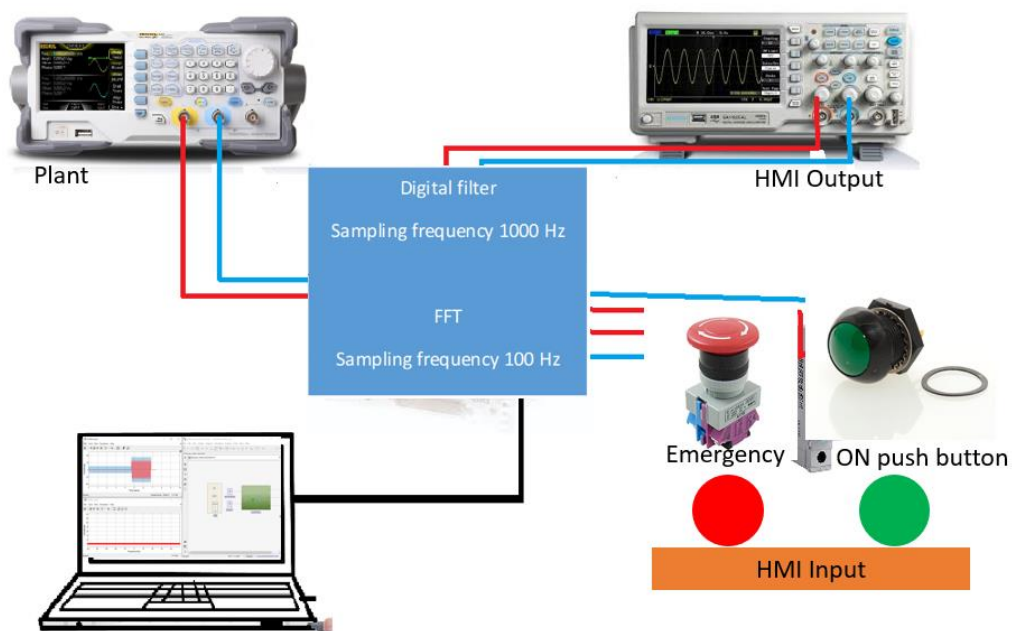


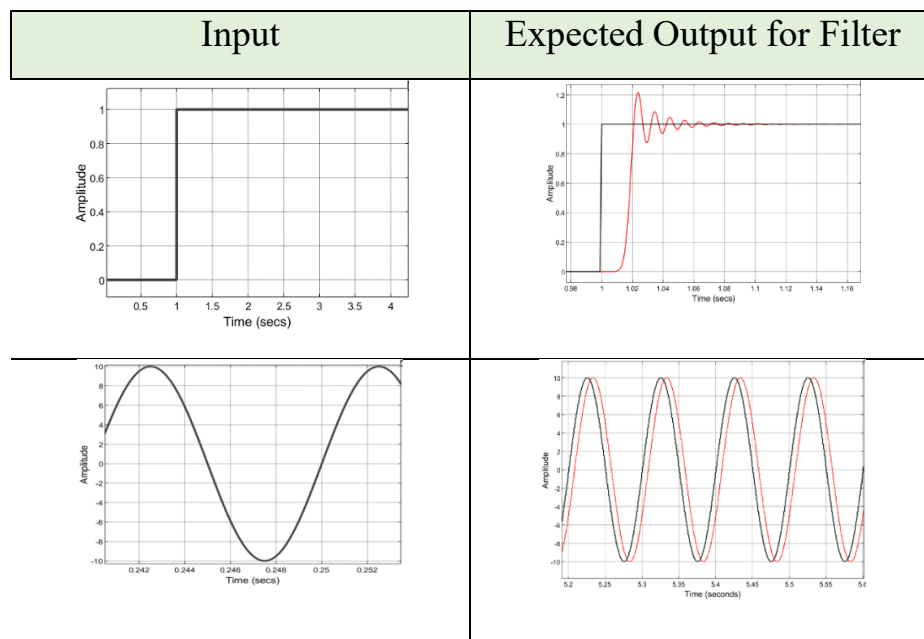
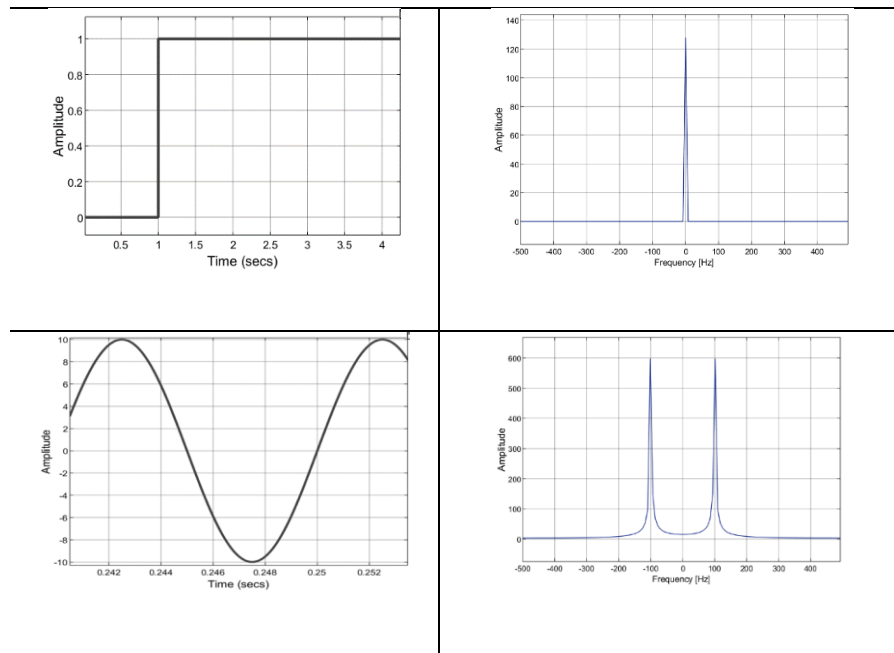
Figure 36. Layout

Signal Name	Symbol	Function	Unit	Value range
switch_on	s _O	Turn on system	V	Binary
emergency_stop	s _E	Emergency stop	V	Binary
Signal1	u	Input signal	V	+ - 10
Fourier analysis	F(u)	Output	V	Undefined
Filtered signal	y	Output	V	+ - 10
On_button	b _O	Input	V	Binary
Emergency_switch	b _E	Input	V	Binary

Table 5. Signals

Expected results:

Input	Expected Output for FFT
-------	-------------------------



Requirements specification:

Function 1	Digital filter
Requirement 1-1	Sampling frequency must be $f_s = 1000$ Hz

<i>Requirement 1-2</i>	Input signal voltage shall be in the range of ± 10 [V]
<i>Requirement 1-3</i>	Output signal voltage shall be in the range of ± 10 [V]
<i>Function 2</i>	<i>Fast Fourier Transform</i>
<i>Requirement 2-1</i>	Sampling frequency must be $f_s = 100$ Hz
<i>Function 3</i>	<i>Supervisory Control</i>
<i>Requirement 3-1</i>	External Start/Stop push button must be added
<i>Requirement 3-2</i>	System is turned by the Start/Stop push button
<i>Requirement 3-3</i>	System is stopped by the Start/Stop push button
<i>Requirement 3-4</i>	An external emergency switch must be added
<i>Requirement 3-5</i>	When emergency switch is turned on, output must go to 0

Table 6. Requirements specification

Hazard analysis and risk assessments

Hazard identification:

Component	Failure Mode	Effect on the item
Signal generator	FM1: Wrong input signal FM2: Not grounded FM3: Signal voltage beyond limits	Hazard 1: Item gives wrong output Hazard 2: Possibility of damage Hazard 3: Possible damage on electrical components
Microprocessor	FM4: Microprocessor fails in executing instructions	Hazard 4: Item does not give any output
On/Off switch	FM5: Switch does not turn on FM6: Switch does not turn off	Hazard 5: Item does not turn on Hazard 6: Item does not turn off
Emergency button	FM7: Emergency button does not send the signal	Hazard 7: High possibility of damage
Oscilloscope	FM8: Oscilloscope does not show any output FM9: Not grounded FM10: Oscilloscope shows wrong output	Hazard 8: Item does not give any output Hazard 9: Possibility of damage Hazard 10: Item gives a wrong output

Table 7. Hazard analysis

Hazard analysis and assessment:

Hazard	Effect	Comment
Hazard 1:	Severity: 1 Exposure: 4 Controllability: 1	Wrong output can lead on wrong conclusions Signal generator is always on when item is on Simply controllable
Hazard 2:	Severity: 1 Exposure: 4 Controllability: 1	Possible damage Signal generator is always on when item is on Simply controllable
Hazard 3:	Severity: 1 Exposure: 4 Controllability: 1	High voltage can cause electrical damage, causing damage to operator Signal generator is always on when item is on Simply controllable
Hazard 4:	Severity: 0	No damage possible when there is no output

	Exposure: 3 Controllability: 1	Microprocessor often is working when item is working Simply controllable
Hazard 5:	Severity: 0 Exposure: 4 Controllability: 1	No voltage when item is not turned on On switch is always needed when item must be turned on Simply controllable
Hazard 6:	Severity: 1 Exposure: 4 Controllability: 1	Low happening possibility of undesired actions On/Off switch is always needed when item must be turned on Simply controllable
Hazard 7:	Severity: 3 Exposure: 1 Controllability: 3	Undesired actions can cause damage on electrical components, causing damage to the operator Emergency button is used rarely Can be difficult to control
Hazard 8:	Severity: 0 Exposure: 4 Controllability: 1	No damage possible when there is no output Oscilloscope is always on when item is on Simply controllable
Hazard 9:	Severity: 0 Exposure: 4 Controllability: 1	No damage can be caused Oscilloscope is always on when item is on Simply controllable
Hazard 10:	Severity: 1 Exposure: 4 Controllability: 1	Possible damage Oscilloscope is always on when item is on Simply controllable

Table 8. Risk assessment

ASIL Selection:

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 37. ASIL Selection table

From the table:

- ASIL A for Hazard 7
- QM for all other Hazards

Safety Goal:

Safety goal 1: Item shall stop immediately when the Emergency button is pushed and enter to the safe state

Safe state: The item shall to the OFF state, where the output goes to zero and the user is informed.

1.1. Functional safety concept

Functional safety:

Because the required ASIL is ASIL A, functional safety action is necessary, but it must be low-cost since the hazardous situation is very unlikely.

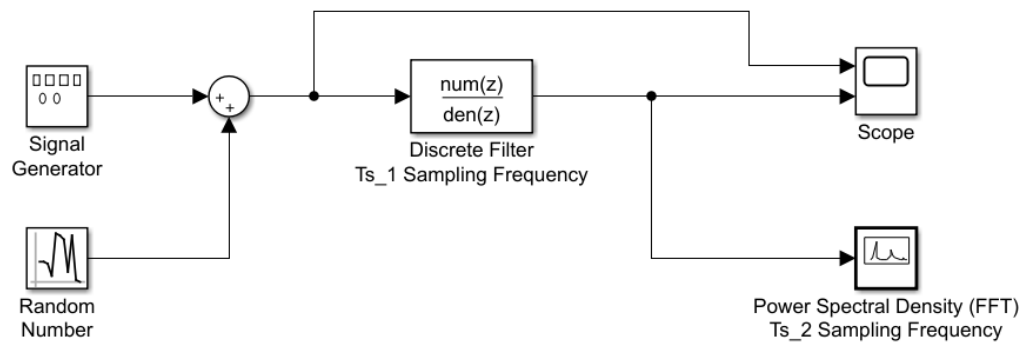
Proposed functional safety action:

Functional Safety 1: Transition to emergency must be highest priority

Concept model

As already explained in the first chapter, concept Model should grasp and show the behavior of the main tasks either separately or together.

Digital Filter Concept Model



- Main specs:
- +/- 10 V input
 - filter sampling frequency 1 kHz
 - fft (spectrum) update 1 Hz
 - ON/OFF & Emergency buttons
 - live filter update

Figure 38. Concept model in Simulink

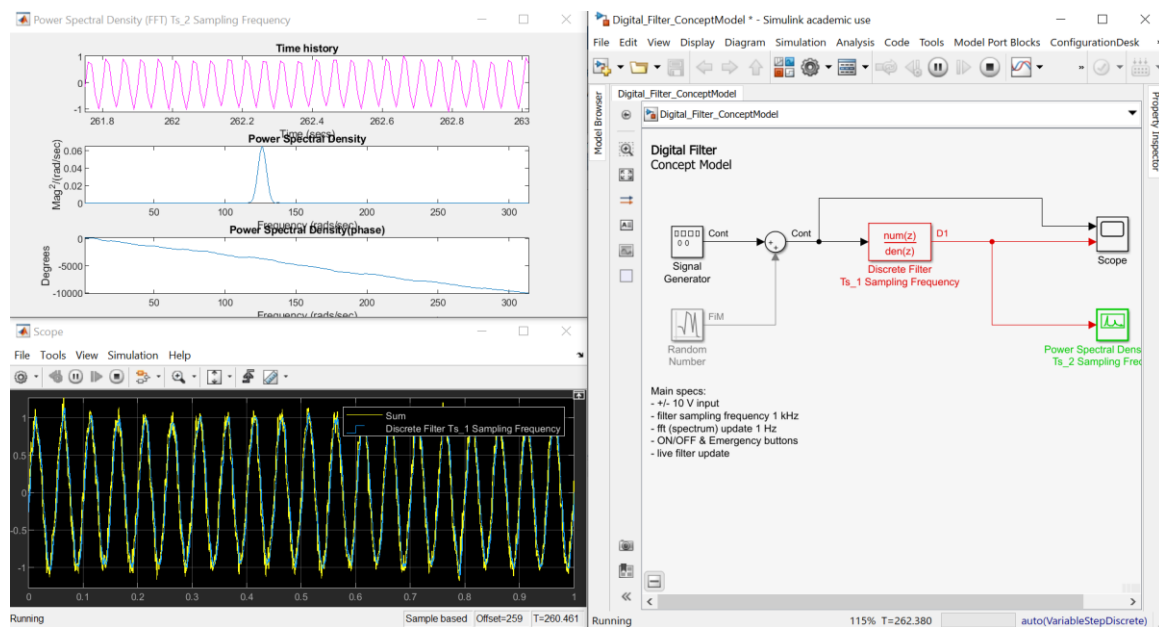


Figure 39. Results of Concept Model

Architecture & Design

Modular architecture and the components with interfaces must be defined:

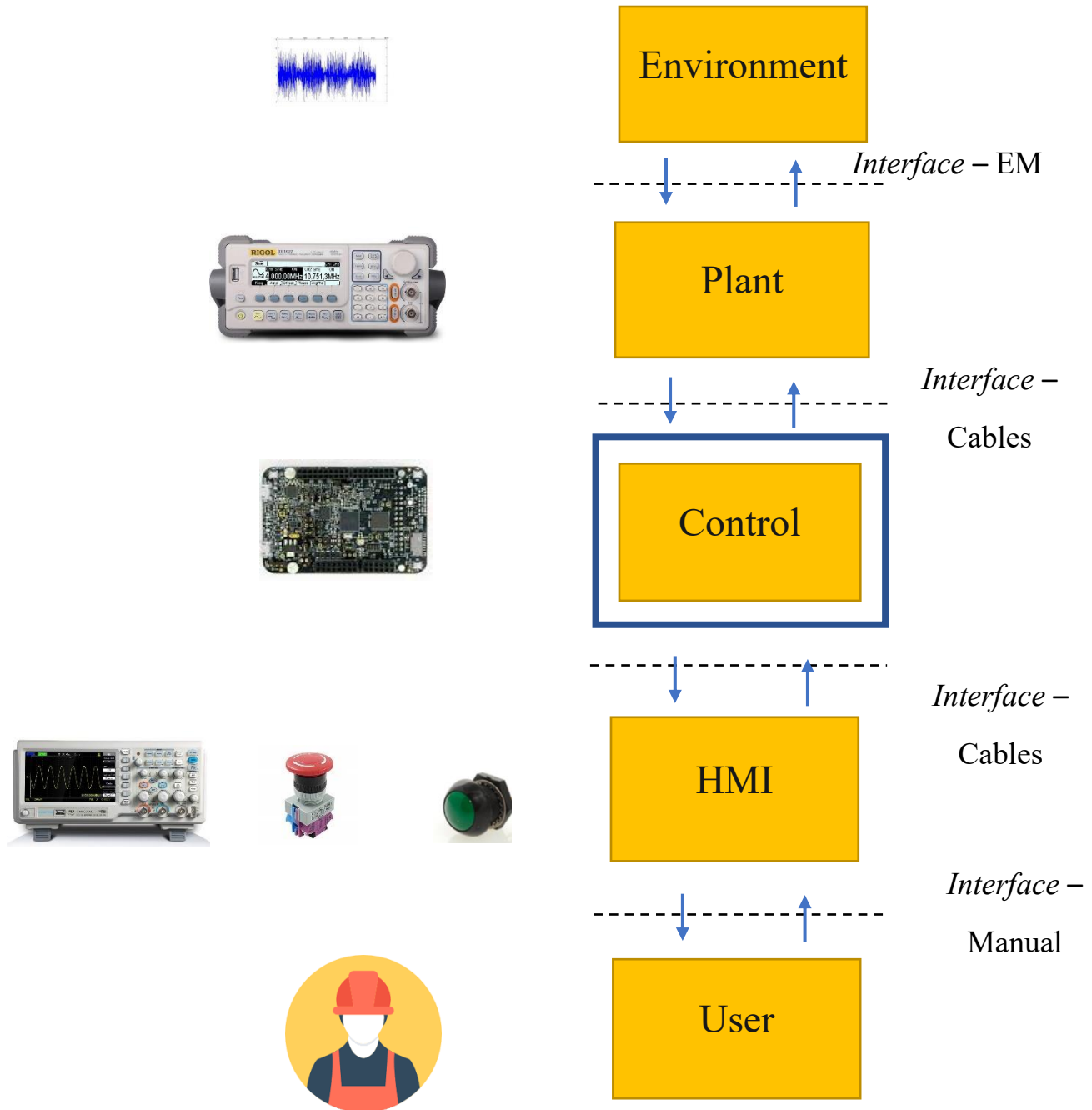


Figure 40. Item Architecture

The following characteristic or features specified should be seen as requirements: [13]

- The environment: The item will be placed in a Lab
- There are no permitted ways of use

- Only one mode of operation specified
- Both our functions, Digital filtering and DFT are function call subsystems
- Input signal is analog with range ± 10 V, with a certain high frequency noise
- Output signal is analog with range ± 10 V

Interfaces

Physical interface

- The item will be placed in a typical Lab desk
- Room temperature
- Signal range 20V peak-to-peak

Energy interfaces

- Electric energy only
- 20V peak-to-peak voltage transfer
- Energy provision via cables

Material transfer (interface)

- No material transfer

Information interfaces

- Signal processing
- Analog input to ADC to RCP Platform to DAC to Oscilloscopes
- Bus or communication systems CAN or Ethernet

2. Technical Model

Technical Model should exhibit the main technical aspects, i.e. the sampling rate and the quantization levels, the saturation levels as well as other non-linearities. In addition, it must define the overall logic, i.e. states, transitions between states, tasks associated with states.

Tool – MATLAB Simulink

Techniques – Will be specified and chosen in each stage

Methodologies – Model-based Design, MAAB Guidelines

Artefacts – Software code

Safety aspect – Techniques recommended by ISO26262 for ASIL

Initiation

Guidelines to perform modelling that are required by ISO26262:

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity ^a	++	++	++	++
1b	Use of language subsets ^b	++	++	++	++
1c	Enforcement of strong typing ^c	++	++	++	++
1d	Use of defensive implementation techniques	0	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++
^a An appropriate compromise of this topic with other methods in ISO 26262-6 may be required.					
^b The objectives of method 1b are					
□ Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.					
□ Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.					
□ Exclusion of language constructs which could result in unhandled run-time errors.					
^c The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.					

Figure 41. Modelling and coding guidelines

Since we have ASIL A, we must choose an appropriate combination of some requirements since they are alternative entries. A good appropriate combination for our item will be:

1. *Enforcement of low complexity*
2. *Use of unambiguous graphical representation*
3. *Use of naming conventions*

Specification of software safety requirement:

Recalling the functional safety defined in the previous phase:

Functional Safety 1: Transition to emergency must be highest priority.

According to ISO26262 we must define the components of the item that are responsible to achieve or maintain the safe state, which are:

1. Emergency switch
2. Microcontroller

Functions related to safety requirement:

1. Supervisory control/Stateflow

Software architecture and specification of safety requirements

Methods for notation that are given by ISO26262, for ASIL A *Informal notations* is highly recommended, so we decide for 1a.

Methods		ASIL			
		A	B	C	D
1a	Informal notations	++	++	+	+
1b	Semi-formal notations	+	++	++	++
1c	Formal notations	+	+	+	+

Figure 42. Notations for software architectural design

For error handling we decide for *Range checks of input and output data* since it is highly recommended and can give us good desired results.

Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check ^a	+	+	+	++
1c	Detection of data errors ^b	+	+	+	+
1d	External monitoring facility ^c	0	+	+	++
1e	Control flow monitoring	0	+	++	++
1f	Diverse software design	0	0	+	++
^a Plausibility checks can include using a reference model of the desired behaviour, assertion checks, or comparing signals from different sources.					
^b Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.					
^c An external monitoring facility can be for example an ASIC or another software element performing a watchdog function.					

Figure 43. Error detection at the sw architectural level

For verification ISO26262 gives us several methods. An appropriate and robust combinations would be:

1. *Walk-through of the design*
2. *Inspection of the design*
3. *Control flow analysis*

Methods		ASIL			
		A	B	C	D
1a	Walk-through of the design ^a	++	+	0	0
1b	Inspection of the design ^a	+	++	++	++
1c	Simulation of dynamic parts of the design ^b	+	+	+	++
1d	Prototype generation	0	0	+	++
1e	Formal verification	0	0	+	+
1f	Control flow analysis ^c	+	+	++	++
1g	Data flow analysis ^c	+	+	++	++
^a In the case of model-based development these methods can be applied to the model.					
^b Method 1c requires the usage of executable models for the dynamic parts of the software architecture.					
^c Control and data flow analysis may be limited to safety-related components and their interfaces.					

Figure 44. Methods for the verification of the software architectural design

The following elements shall be verified:

- Compliance with the software safety requirements
- Compatibility with the target hardware
- Adherence to design guidelines

Unambiguous illustration of architectural design that realizes the software safety requirements:

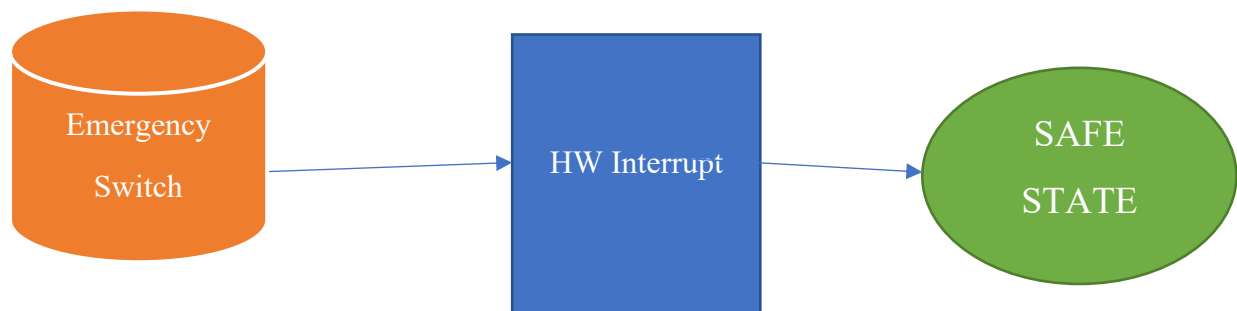


Figure 45. Illustration of HW Interrupt

We have three states:

- OFF State – which is set by default
- ON State – where the actions required are performed and inform the User via HMI
- Emergency State – Where we set the output to zero and inform the User via HMI

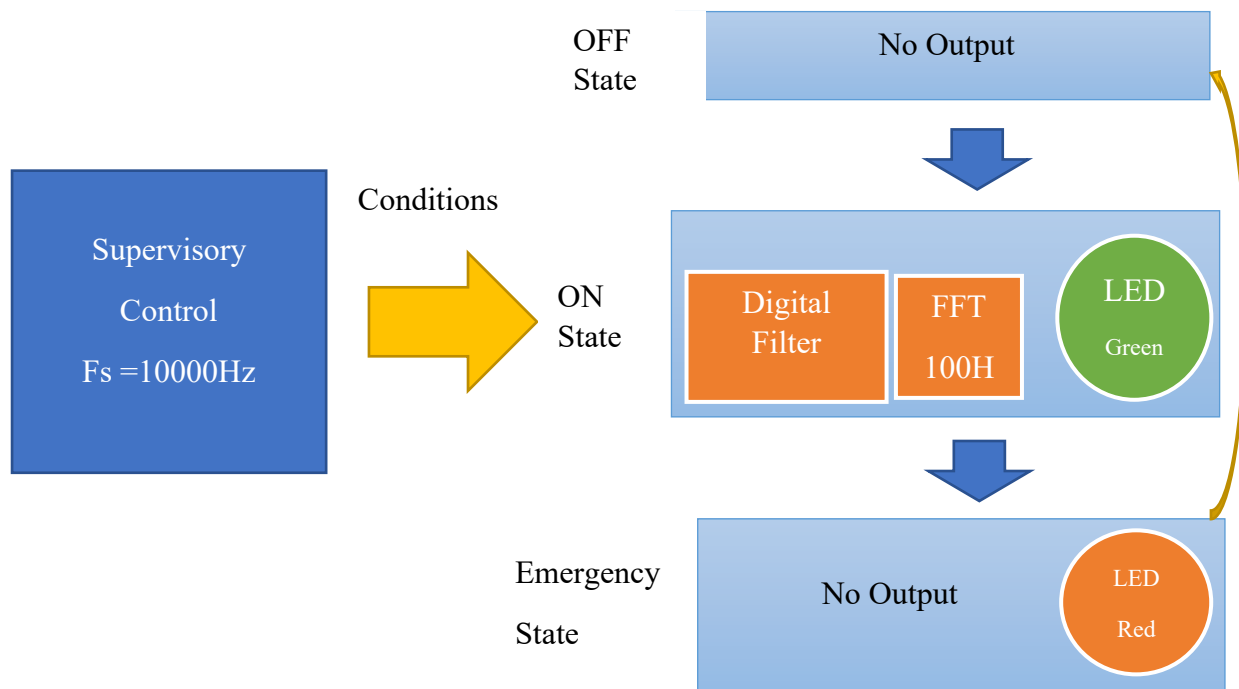


Figure 46. Illustration of model hierarchy

Model-based Design

Notations to be followed for ASIL A:

- *Natural language*
- *Informal notations*

Methods		ASIL			
		A	B	C	D
1a	Natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

Figure 47. Notations for software unit design

Design principles for software unit design and implementation to be followed for each unit:

- *No hidden data flow or control flow*
- *No recursions*
- *Initialization of variables*

Methods		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions ^a	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation ^{a, b}	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names ^a	+	++	++	++
1e	Avoid global variables or else justify their usage ^a	+	+	++	++
1f	Limited use of pointers ^a	0	+	+	++
1g	No implicit type conversions ^{a, b}	+	++	++	++
1h	No hidden data flow or control flow ^c	+	++	++	++
1i	No unconditional jumps ^{a, b, c}	++	++	++	++
1j	No recursions	+	+	++	++

^a Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

^b Methods 1g and 1i are not applicable in assembler programming.

^c Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

Figure 48. Design principles for sw unit design and implementation

Methods for software unit testing:

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test ^a	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test ^b	+	+	+	++
1d	Resource usage test ^c	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable ^d	+	+	++	++

^a The software requirements at the unit level are the basis for this requirements-based test.

^b This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).

^c Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

^d This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

Figure 49. Methods for software unit testing

Requirement-based test is highly recommended and it is enough for our model.

Task 1 - Supervisory control

Concerning the discrete control, we must have 3 states:

- OFF state, which is set by default
- ON state
- Emergency state

Transition conditions from OFF to ON: On pushbutton must be pushed for 2 seconds. Color of the led will be yellow during the transition.

Transition from ON to OFF: On pushbutton must be pushed for 2 seconds. Color of the led will be yellow during the transition.

Transition from ON to Emergency: Emergency switch is turned on. Color of the led will be yellow during the transition.

Transition from ON to Emergency: Emergency switch is turned off.

Transition from Emergency to ON is not possible for safety reasons.

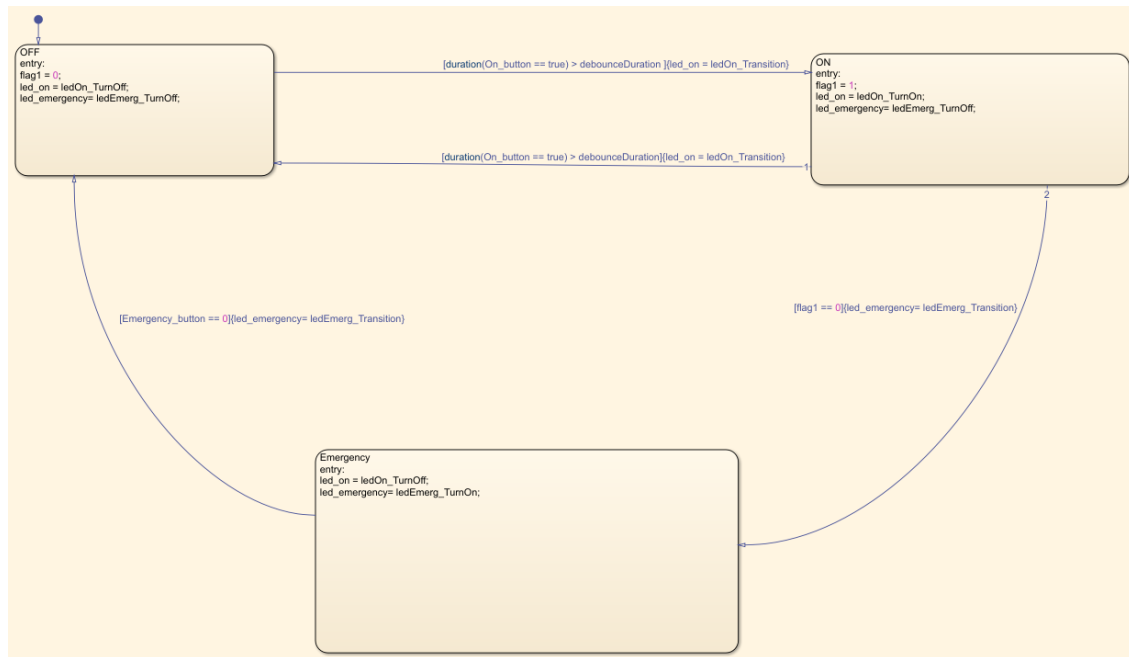


Figure 50. Stateflow chart

Task 2 - Digital Filter

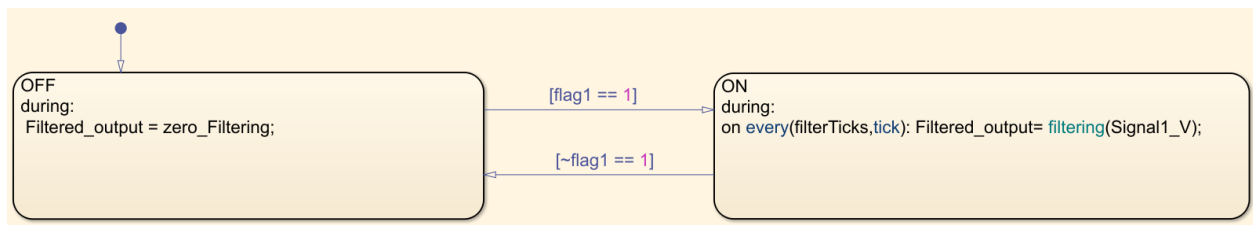


Figure 51. Task control

Design characteristics:

- *Frequency response*: Lowpass
- *Sampling frequency*: $f_s = 1000 \text{ Hz}$

- Cutting frequency: $f_c = 100 \text{ Hz}$
- Simulink block used: FIR Filter
- Filter order: 2

Simulink block used:

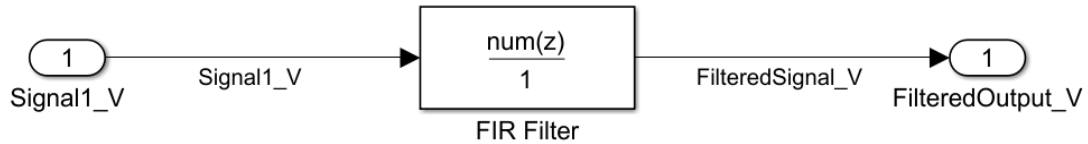


Figure 52. Digital Filter modelling

Validation of Task 2

Results with a given input of 100Hz and amplitude 10:

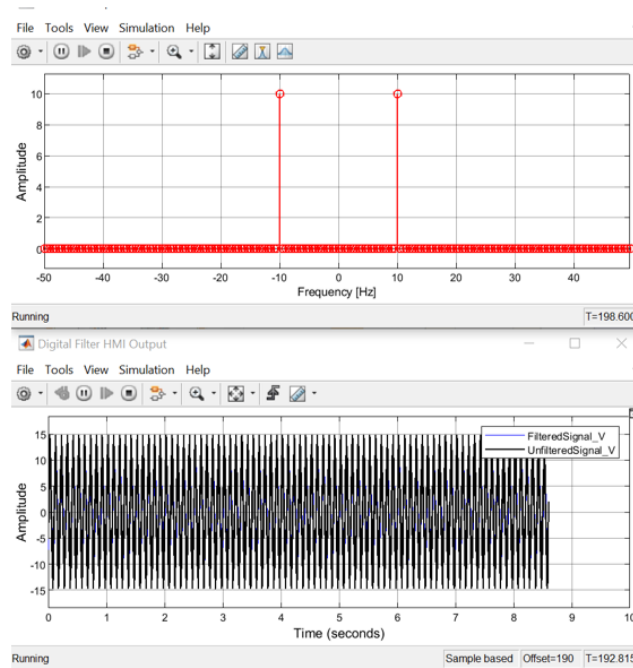


Figure 53. Results 1 of Digital Filter and FFT

Results with a given input composed by two signals with 10Hz amplitude 10V and 40Hz amplitude 5V:

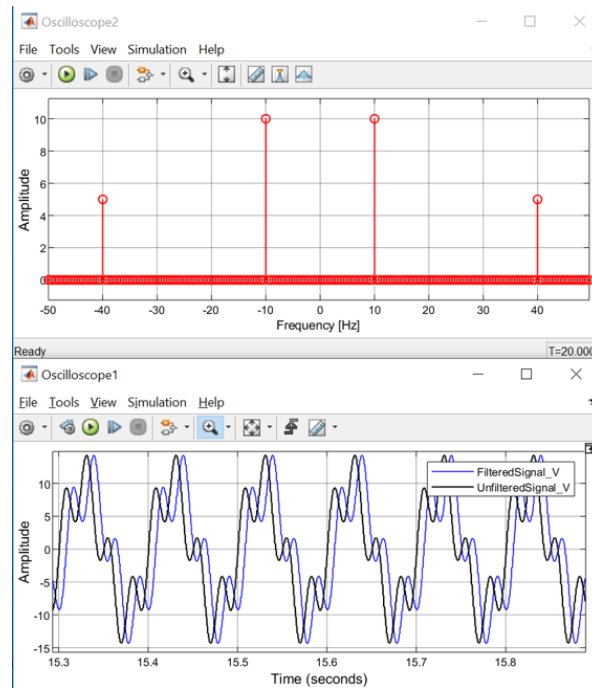


Figure 54. Results 2 of Digital Filter and FFT

From these results we can say that we satisfy the results of Concept Model. Furthermore, we added:

- state machine
- quantization
- sampling
- holding
- HMI design

Now to start the process of transitioning from Technical to Production model shall start we must implement:

- Scheduling of the tasks
- Interrupt handling

Task 3 - Fast Fourier Transform

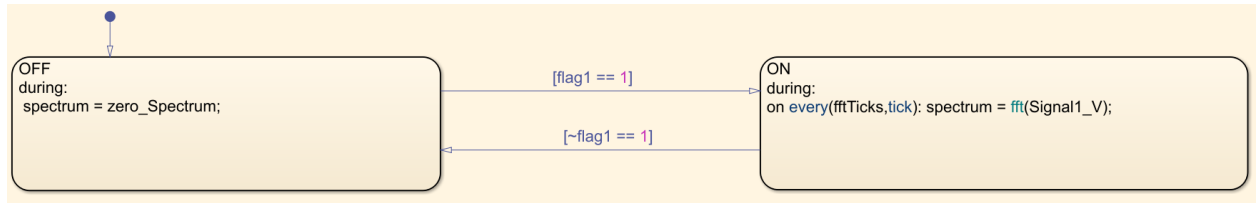


Figure 55. Task control

FFT:

Performs FFT in the incoming signal

Complex to Magnitude-Angle:

Converts the complex values to magnitude. By default it has 2 outputs, magnitude and angle, but in our case only the Magnitude is selected.

FFT Shift Matlab function:

Shifts zero-frequency component to center of spectrum. It is useful for visualizing the Fourier transform with the zero-frequency component in the middle of the spectrum.

FFT in Simulink:



Figure 56. FFT modelling

Validation of Task 3

Result for a given input with 100Hz frequency and amplitude 10:

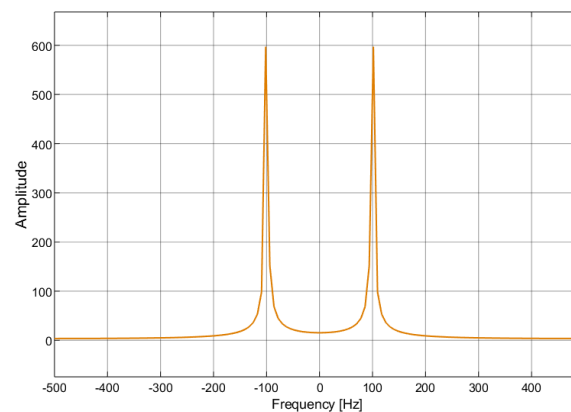


Figure 57. FFT result

Result for a given input with 300Hz frequency and amplitude 10:

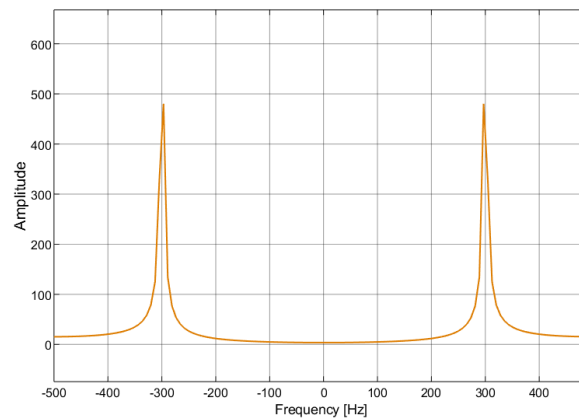


Figure 58. FFT Result

Emergency Task

Emergency Task as specified in Functional Safety requirement must be the highest priority, i.e. this will translated as the task will be triggered by a Hardware Interrupt with highest priority and cannot be pre-empted by any other task. The task will be managed with Stateflow with following flow:

1. Hardware interrupt is ON
2. Emergency task starts running
3. Set an emergency global variable to 'True'
4. Emergency Task finishes

Now, depending on which task is next in queue:

5. Supervisory Control goes to Emergency State
6. Task controls go to OFF

All of the tasks are controlled by the emergency global variable because we do not know which task was pre-empted by the Hardware Interrupt, thus which task will run after the Emergency Task. Doing this, we make sure that everything goes to OFF after Emergency task runs.

Emergency function-call task Stateflow:

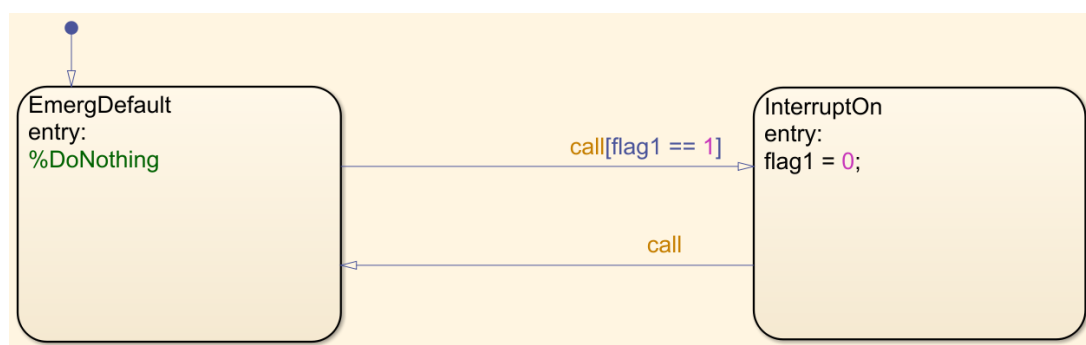


Figure 59. Interrupt handling

How can we manage Scheduling and Interrupts together with Validation of the task will be analyzed in the following.

Scheduling

Scheduling of the tasks is very important in Real-time applications such as ours. In this subchapter we will analyze how can we use Simulink to do so and implement it in our example. Simulink offers two types of scheduling:

- Time-Based Scheduling
- Event-Based Scheduling

Firstly, as seen in Customer Requirements table, in our application we have 3 Synchronous tasks that have to be executed as Time-Based Scheduling:

- Supervisory Task
- Digital Filter Task
- FFT Task

Secondly, we have another task that must be executed Asynchronously, i.e. only when the Emergency switch is turned on, thus this task must be scheduled as Event-Based Scheduling. The illustration below shows the scheduling principle without taking to account the execution times of the tasks:

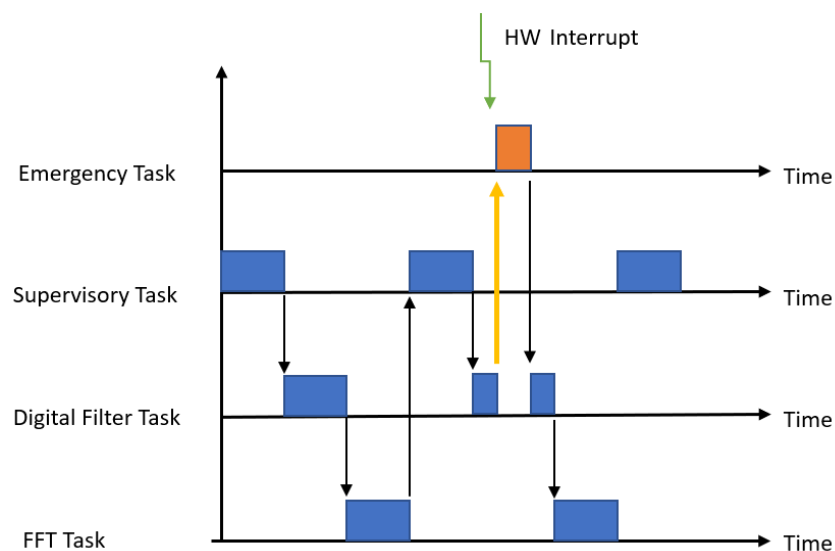


Figure 60. Scheduling

Simulink offers several ways to handle the scheduling of tasks(subsystems). Here we will use ‘Temporal logic scheduler’ that is implemented via Stateflow. This technique allows us two different ways to use it:

- Event-based Temporal logic
- Absolute-time Temporal logic

For Absolute-time Temporal logic the operators that can be used are:

- `after(x,time)`
- `before(x,time)`
- `every(x,time)`
- `temporalCount(time)`
- `elapsed(time)`

Time can be set as seconds(sec), milliseconds(msec), microseconds(usec), and ‘x’ is the time value.

But, for RTOS applications using Absolute-time is not recommended from Simulink. Thus we will use Event-based Temporal logic to execute Synchronous tasks. The operators are the same as Absolute-time, but they are used in a different way. The syntax is as follows:

- `every(n,tick)`

The example is given for the operator ‘every’ but it is the same for every operator. The important thing is that the variable ‘tick’ has to be linked in a Timer, Clock or to the base rate. Here, we decide to not put anything more and complicate the model and use the base rate. In this case, the base-rate and sub-rate tasks will be managed by the OS itself and not by timer interrupts. The logic goes like this:

- Execute Supervisory Control Task in a base rate that is 10Khz
- Execute Digital Filter Task every 10 base rates, thus frequency is 1KHz
- Execute FFT Task every 100 base rates, thus frequency 100Hz.

Interrupts

A more complex process to be managed in Simulink is handling Interrupt Service Routine(ISR). The block that creates an ISR and it also is supported from Embedded Coder, i.e. its code can be automatically generated is the block called '*Hardware Interrupt*' block. This block can be used only in subsystems that are set as a 'Function-call subsystems' and it is different for every type of hardware. In this case we will analyze the most common one:

- ARM Cortex-M processors

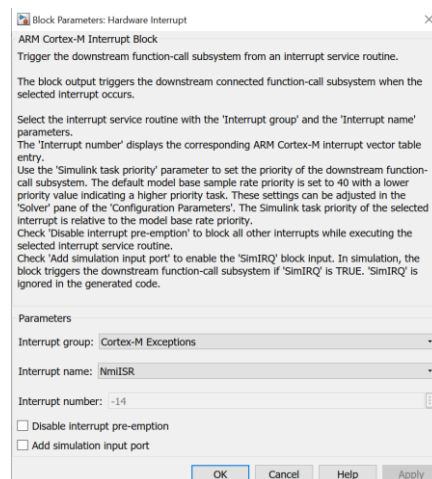
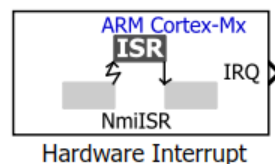


Figure 61. Hardware interrupt block and its parameters

(Another example is '*External Interrupt*' block for Arduino Hardware.)

To use the '*Hardware Interrupt*' block we must set its parameters:

- Set interrupt group, in this case Cortex-M
- Set interrupt name, it will correspond to the specific entry of the processors interrupt vector table. A good option is to leave it as it is and then check if that is available in the processors vector table
- Interrupt number, corresponds to the position of interrupt in the processor vector table
- Check the 'Disable interrupt pre-emption' because we do not want other interrupts to preempt the 'Emergency Task'

Of course, when the code will be generated and integrated with the firmware, the GPIO input of the board that the hardware interrupt is connected (in our case the Emergency Switch), must be linked to the ISR via hand-written code.

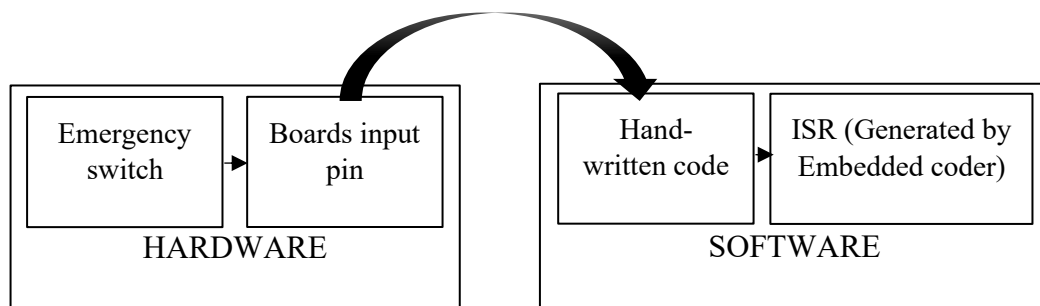


Figure 62. Hardware interrupt flowchart

All this procedure must be done when we set which hardware we will going to use. For now, we will use the Interrupt simulation block. Note that This block cannot be used for code generation.

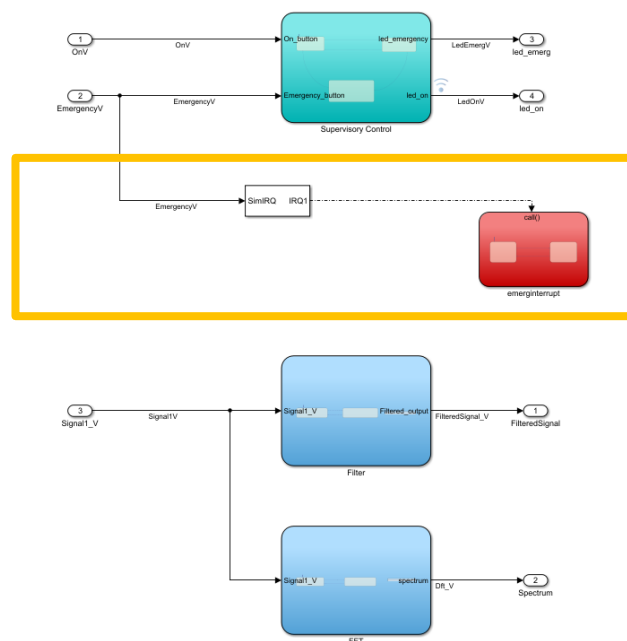


Figure 63. Control content with Interrupt simulation

Validation of Task 1

Monitoring the Supervisory Control via Chart Mode we see that every input of us given through HMI Module is working, as can be seen in the following picture:

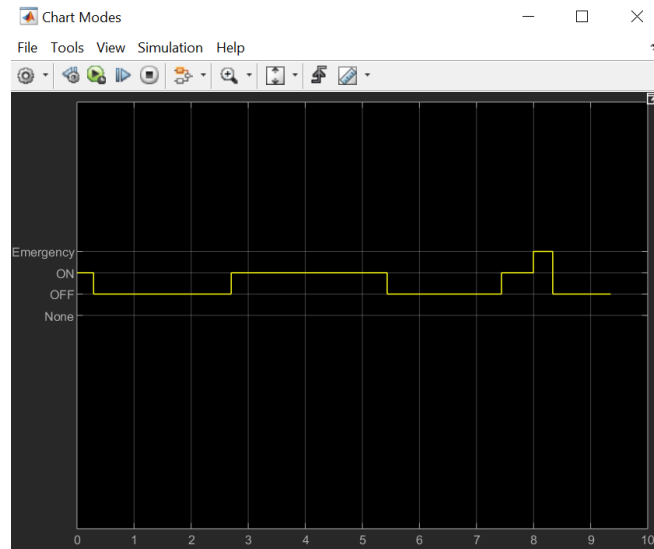


Figure 64. Chart mode

Control frame

Control frame input must take the Analog input signal and give a digital output to the Control content. Here, we have done it via Quantizer block in Simulink.

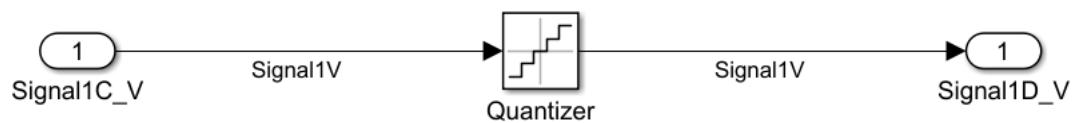


Figure 65. Quantizer

Several tests have been done to see the number of bits needed to have a good result, of course keeping in mind that we are in a simulation environment. We have concluded that until 6 bits, the results with our range of frequency and amplitude is enough. The results with 4 bits give us a wrong result about signal spectrum, since the algorithm does not have enough information to give us the good result. The output with 4 bit quantization is shown below:

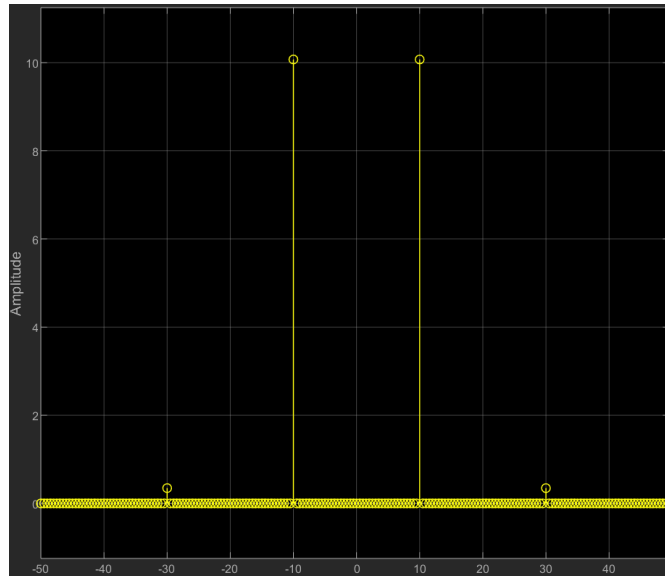


Figure 66. FFT Output with 4 bit quantization

The control frame output must take the calculated digital output from the Control content and transform it to Analog. Here we have done it via PWM and a Lowpass filter. The parameters to be set are:

- Saturation levels
- Period of Repeating sequence
- Value of Repeating sequence
- Relay switch on and off values
- Output when off and on
- LPF characteristics

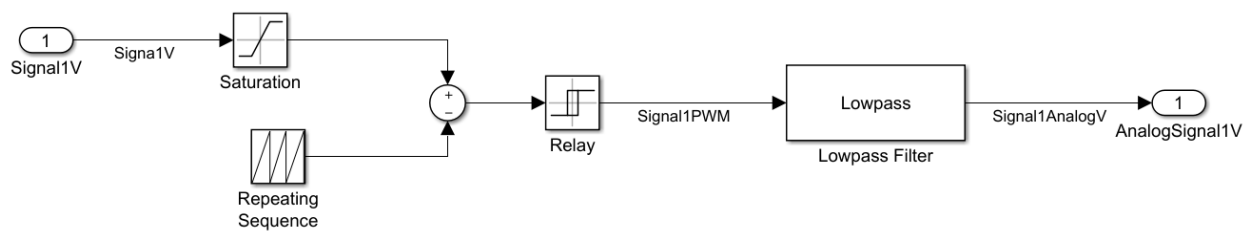


Figure 67. PWM Design

Results after reconstruction of the Signal1:

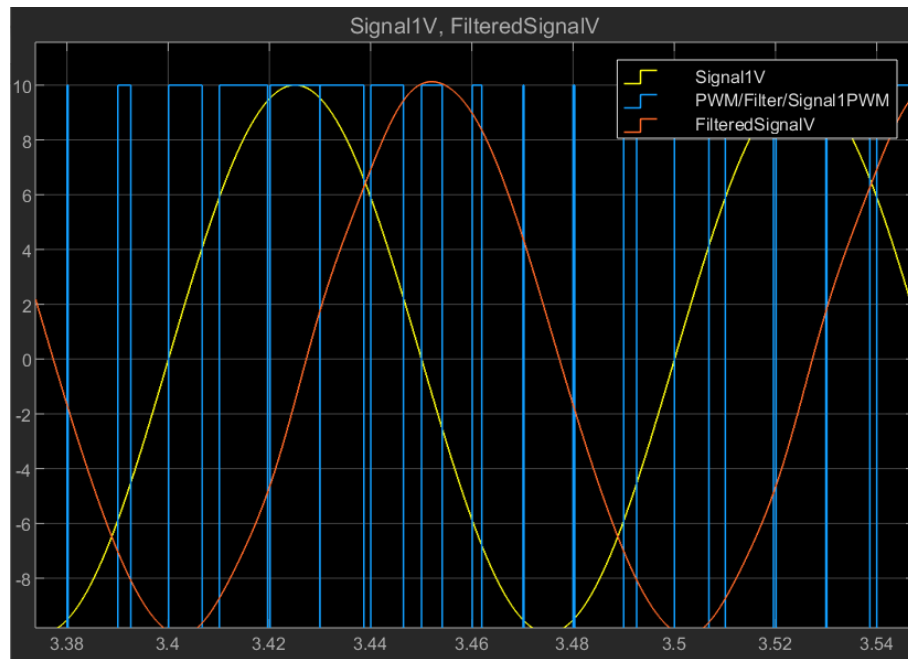


Figure 68. PWM Results

Human-Machine-Interface

Components:

- Start/Stop push button
- Emergency switch
- Oscilloscope
- On LED
- Emergency LED

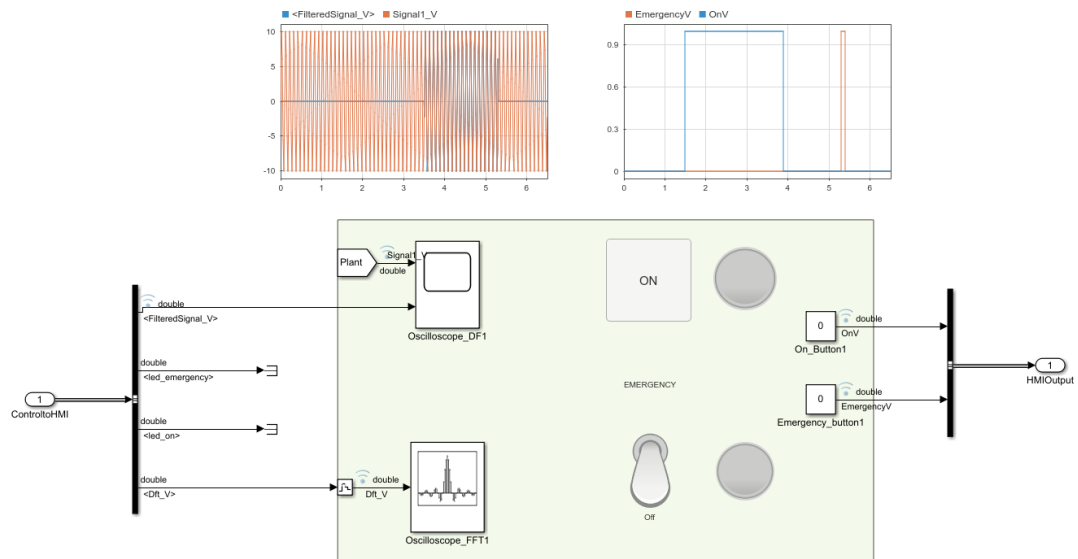


Figure 69. HMI Design

HMI feedback:

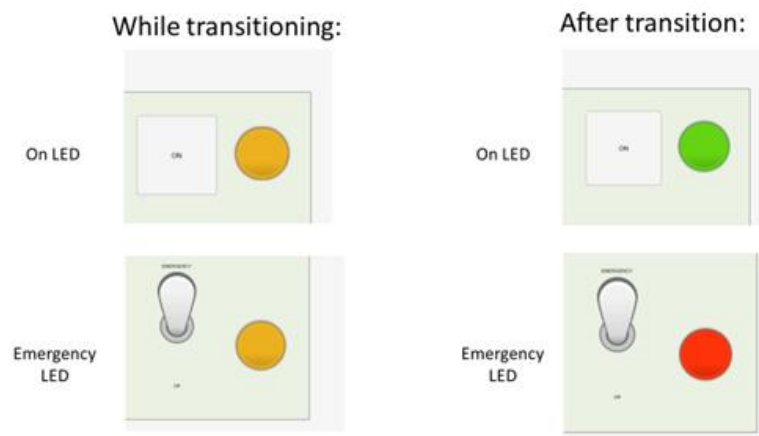
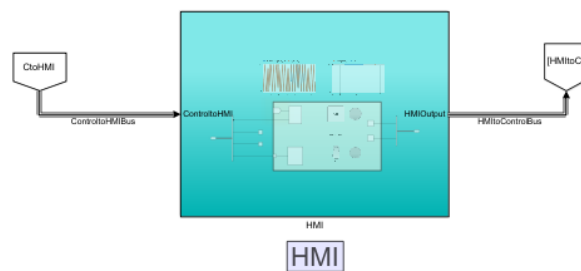
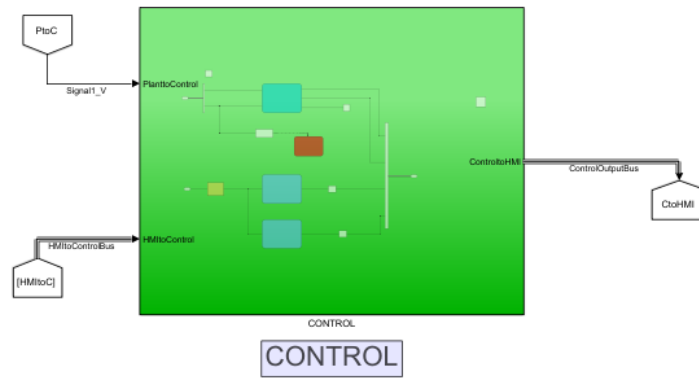
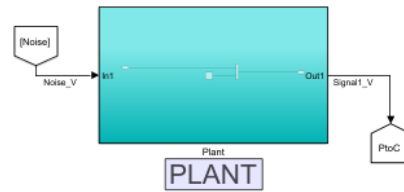
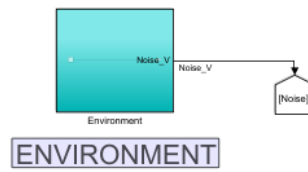


Figure 70. HMI Feedback



USER

Figure 71. Simulink model architecture view

Testing

For this example we only had a chance to perform SIL and not PIL or HIL since we did not go further for VMU selection. We focused on Model Based design and code generation thus SIL was necessary.

Besides this, here we also perform the checks for the standards and guidelines via Simulink to see if our work satisfied ISO 26262, MISRA C, MAAB Guidelines.

Tools – Simulink, Simulink Check™

Safety Aspect – SIL Testing, Standards check

Techniques – SIL

Artefacts – SIL results, passed or failed Standard check

Software in the Loop

“Software-in-the-loop simulation mode denotes simulations in which the software of the real control system is embedded in the simulation loop. The simulation contains part of the real system, i.e. the control software, together with simulated parts, i.e. the device hardware and the environment. The executable code of the real control system is directly embedded in the simulation. In SIL, the software of the real control system is deployed on simulated devices that reside within a simulated environment with simulated sources of dynamism. SIL simulation is typically used during the late stages of application development. SIL simulation enables experimenting with the control system on simulated devices before deployment.” [14]

In our case we will carry this task in Simulink. Certainly, SIL model contains only CONTROL module and nothing else.

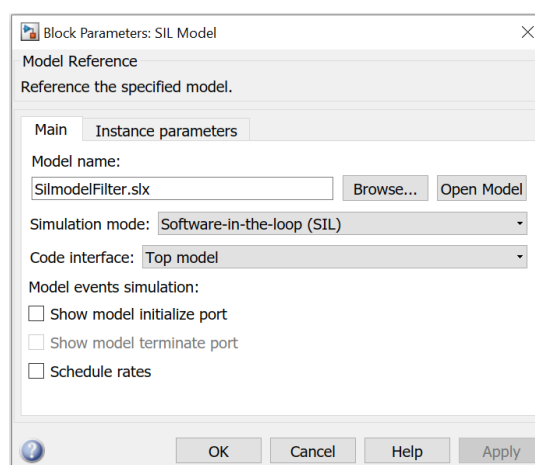


Figure 72. SIL Settings

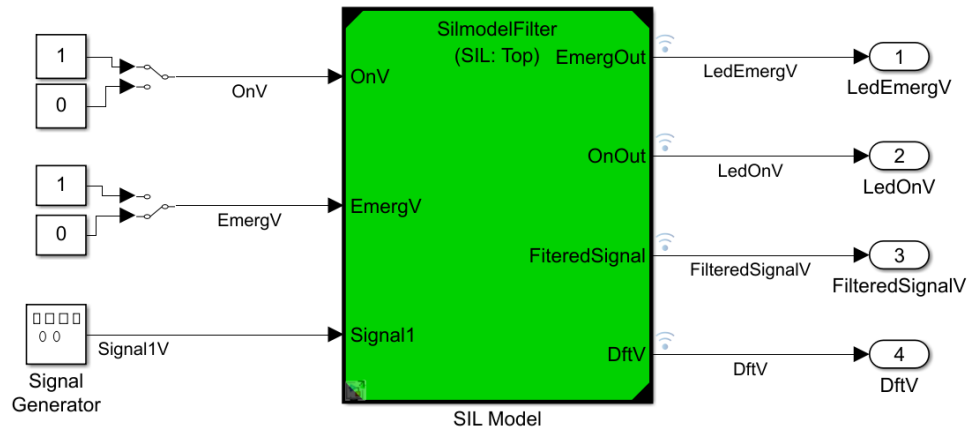


Figure 73. Simulink model to run SIL

Because this example is computationally simple with very few variables, we see that the results are the same as Model-in-the-Loop. In the contrary, we will see that differences will be rather bigger when we deal with a real application.

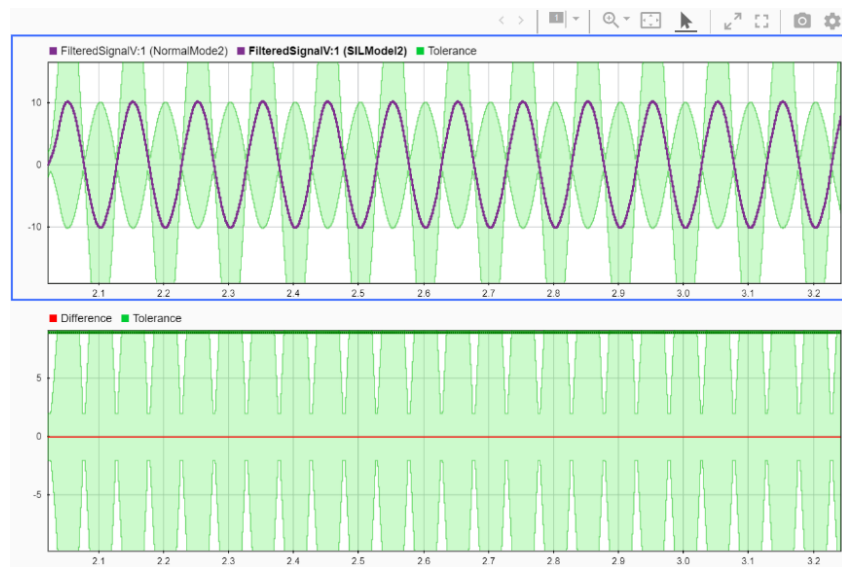


Figure 74. SIL results

Standards check

The results of modelling were satisfactory and failed none of the standards or guidelines. But, there is room to improve the *Warnings* given. All the reports of each check are generated and stored in documentation. In the following figures we can see the summary of results for each of the ran checks:

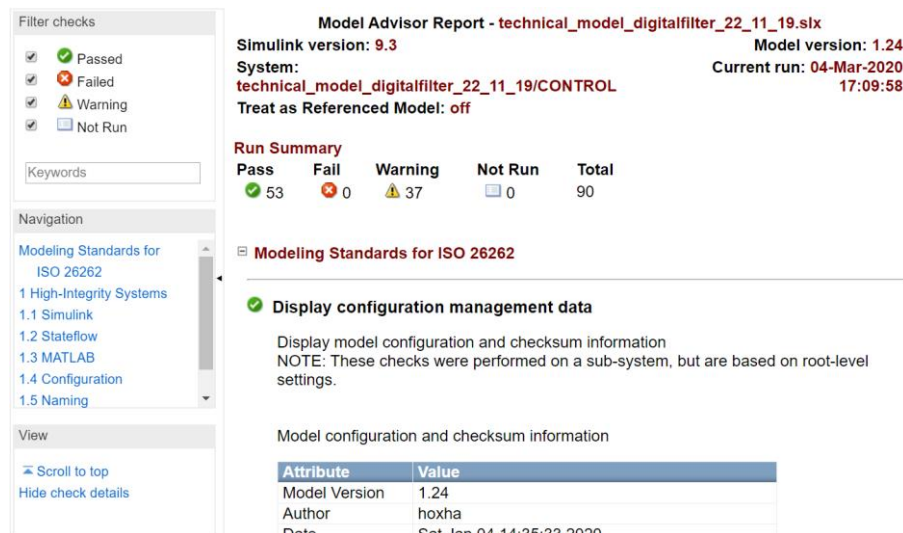


Figure 75. ISO 26262 check results

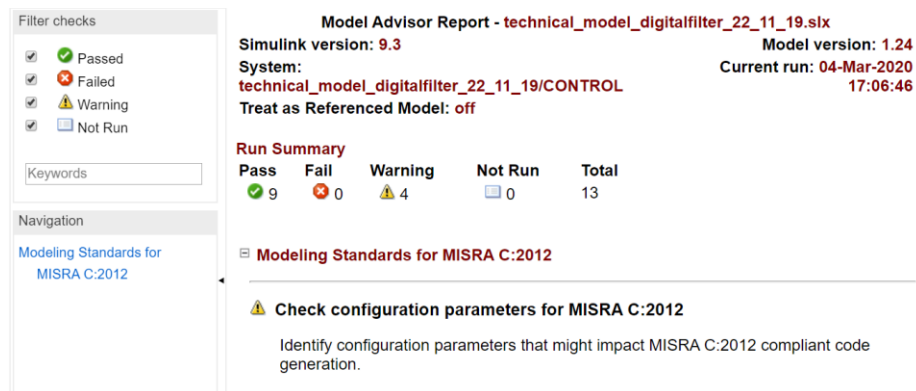


Figure 76. MISRA C:2012 check results

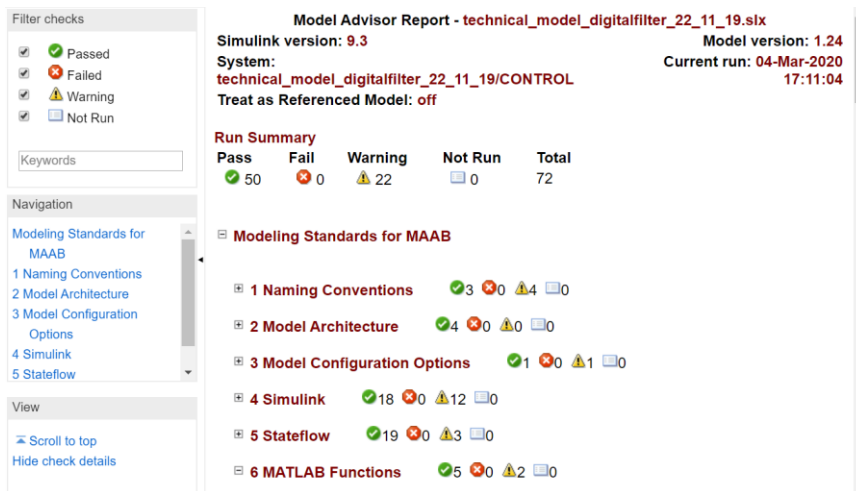


Figure 77. MAAB guidelines check results

In addition of the standards, we run also Code Generation Advisor with no failures:

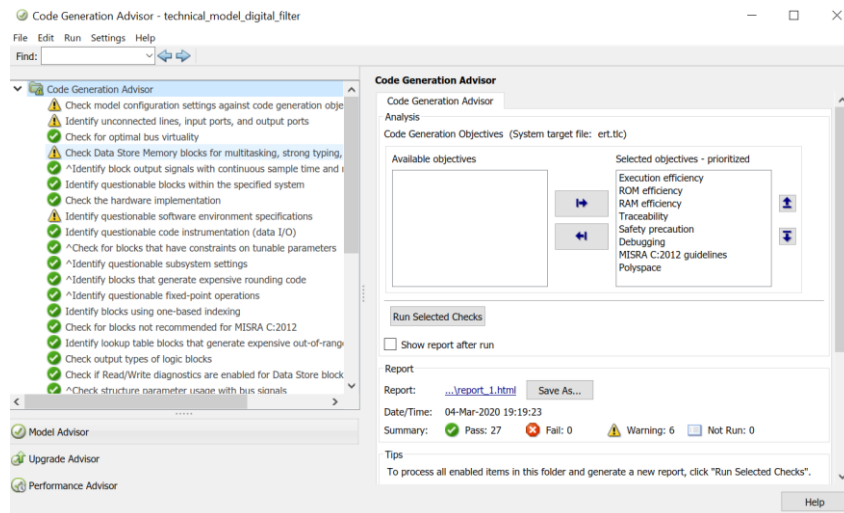


Figure 78. Code generation advisor check results

PART 2, BAT-MAN PROJECT

1. Customer Requirements

Item definition

“The proposal of BAT-MAN is to make significant technologic innovations (especially relative to the techniques of estimation and diagnostics), realizing at the same time a product idea (realizing a prototype) that, on the one hand it can offer immediate and large-scale feedback on the solutions developed, and on the other can act as a forerunner to a series of applications based on the same technologies, either in areas closely related to accumulation systems, or in areas where advanced diagnostic and estimation techniques can bring a significant added value”. [1]

In other words, BAT-MAN project is an innovative approach to estimate the State of Charge (SoC) and State of Health (SoH) of a Lead-Acid battery. This project comes to life to fulfill the great demand of the Automotive Industry which is going in the way of full vehicle electrification, where BAT-MAN becomes a necessity.

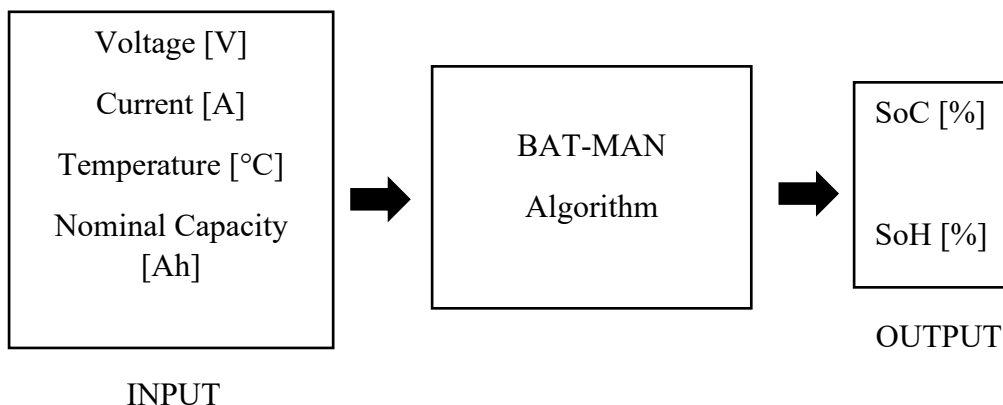
The BAT-MAN estimation algorithm has four inputs and two outputs. The inputs are:

- Terminal Voltage (measured)
- Current (measured)
- Temperature (measured)
- Nominal Capacity (Characteristic data of the battery)

The outputs in the other hand as we already mentioned are:

- SoC (State of Charge of the battery)
- SoH (State of Health of the battery)

Figure 79. Item definition



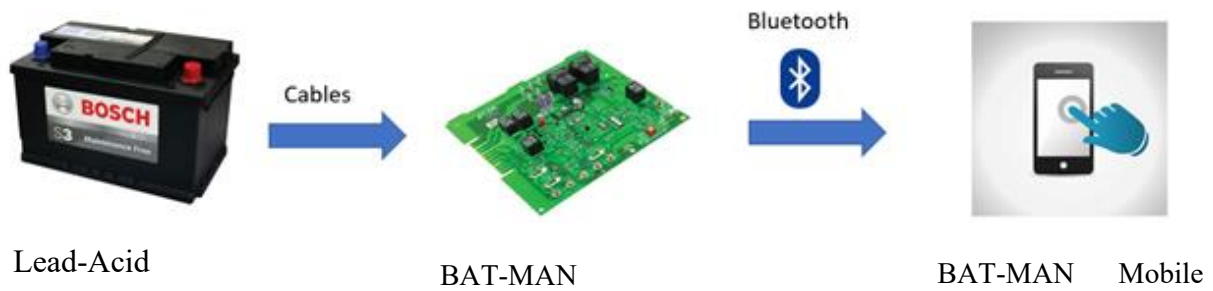


Figure 80. Item definition

What is State of Charge?

The amount of charge in the battery that can be extracted from it, in percentage. The formula:

$$SoC = (SoC_0 + \frac{\int_0^t I(\tau) d\tau}{C_n})$$

Nominal Capacity C_n , is the energy storage capacity of a battery in theory, that is given by the producer of the battery. The formula?

$$C_n = C_r + C_{lost}$$

where C_{lost} is the amount of energy that is no more available or extractable from the battery due to its health.

What is State of Health of a battery?

The amount of charge extractable from the battery in its fully charged state, relative to the amount when the battery was new. i.e. its Nominal Capacity given by the producer.

$$SoH = \frac{C_r}{C_n} \cdot 100$$

where C_r is the Real Capacity, i.e. the amount of charge that is extractable from the battery. The formula:

$$C_r = C_{released} + C_{relesable}$$

Estimation of State of Charge

There are several methods to estimate the batteries state of charge. From most of the literature they are categorized in :

1. Direct measurement, uses physical battery characteristic and it can be categorized in:
 - a. Open circuit voltage method
 - b. Impedence method

- c. Terminal voltage method
- 2. Book-keeping estimation, uses discharging current integration over time, and it can be categorized as:
 - a. Coulomb counting method
 - b. Modified coulomb counting method
- 3. Adaptive systems, estimation algorithms that adapt to different conditions, such as:
 - a. Kalman Filters
 - b. Deep learning
 - c. Machine learning
- 4. Hybrid methods, use a combination of methods:
 - a. Coulomb counting and Kalman filter
 - b. Coulomb counting and EMF combination
 - c. Per-unit system and Extended Kalman Filter combination

Estimation of State of Health

SoH estimation methods can be categorized as:

- 1. Capacity estimation techniques
- 2. Internal Resistance estimation techniques

While for the first method we already have talked about, the Internal resistance estimation technique formula is:

$$SoH = \frac{R_{eol} - R_{current}}{R_{eol} - R_{nominal}} \cdot 100$$

where R_{eol} is the internal resistance of the battery in its end of life, and in the other hand $R_{current}$ is the current internal resistance of the battery and $R_{nominal}$ is the nominal internal resistance.

To further continue defining our items properties we must understand what are the factors that SoH depends on. We must define them in this phase, but will furthermore study the effect in the Architecture section, where we modularize the whole system and indeed understand the interactions between them. Factors can be categorized as follows:

- a. Temperature
- b. Overcharging, can damage the electrolytes of the lead-acid battery. It can decrease the capacity of the battery and increase its internal resistance
- c. Cycles of Charge/Discharge
- d. Amplitude of the current extracted, if it goes beyond the factory limits
- e. Ageing, chemical components age due time which decreases the capacitance of the battery

Lead-Acid battery

Today, Lead-Acid battery is the main energy source for starting the automotive engines. Of course, we are not responsible for the lead-acid battery safety itself because that was certainly taken care of by the producer of the battery. Instead, what we are interested in is the range of the output it can deliver, which will be our boards input.

Equipment	Current Sank
Ignition	2-9A
Radio	0.5-5A
Windshield Wipers	7.5A
Headlamps (Low Beam, Dim)	17-18A
Headlamps (High Beam, Bright)	19-20A
Parking lights	4-10A
Brake lights	6-11A
Interior lights	2-4A
Bonnet Light	0.5-1A
Horn	4A
Power Window (One window)	5A
ABS Brakes (Max)	14A
Boot Light	1A
Blower (Heater, Air Conditioner)	14A
Heated Rear Window Defogger	13-28A
Heated Seat	5A
Power Seat Motor	10-13A
Summer Starting (Petrol)	150-200A
Summer Starting (Diesel)	450-550A
Winter Starting (Petrol)	250-350A
Winter Starting (Diesel)	700-800A

Table 9. Typical current loads of passenger cars [15]

The above table, taken by the customer shows us that the current can go until 800 Amps, while the voltage is always 13 Volts. The current depends on the temperature, lower temperatures indicates higher current, thus temperature sensor will become very handy for further analysis of the input data.

Requirements specification

After several meetings with the company, we have ended up with this table of requirements:

Function 1	BAT-MAN Application
<i>Requirement 1-1</i>	Sampling frequency must be $f_s = 10 \text{ Hz}$
<i>Requirement 1-2</i>	Input signal voltage shall be in the range of $\pm 13 \text{ [V]}$
<i>Requirement 1-3</i>	Must calculate SoC, highest priority
<i>Requirement 1-4</i>	Must calculate SoH, highest priority
<i>Requirement 1-5</i>	Must calculate Reliability of the result, lower priority
Function 2	HMI
<i>Requirement 2-1</i>	A mobile app must be created as HMI
<i>Requirement 2-2</i>	Communication between HW and App must be Bluetooth
<i>Requirement 2-3</i>	Must consist two windows, one for inputs of the user (to insert the data of the car, battery of the car, Nomical capacity of the battery). The second is for Outputs.
<i>Requirement 2-4</i>	Start push button must be added (Mobile App) to start communication
<i>Requirement 2-5</i>	Graphs to monitor the output must be added
<i>Requirement 2-6</i>	Graphs to monitor the inputs must be added
<i>Requirement 2-7</i>	Data must be saved on certain files for further use
Function 3	Bluetooth transmission
<i>Requirement 3-1</i>	Outputs must be transmitted every time we have a result
<i>Requirement 3-2</i>	Housing must be designed for EMC standards

Table 10. Requirements

We can see that these requirements demanded by the company are very similar with our example in previous chapter. This shows us that our example was very much on point, thus giving us more confidence to keep going further.

Hazard analysis and risk assessment

“In Part 3 of ISO/DIS 26262 [ISO/DIS 26262-3 2009] the process of hazard and risk assessment is described. Potential hazards are identified following an analysis of the operational situations of the system. The system may be a vehicle, a vehicle system, or a vehicle function. For purposes of our analysis, we will assume the H&RA process is being applied to a vehicle. The identified potential hazards are then categorized based on the following factors: severity, probability of exposure and controllability. The categorization results in the determination of an ASIL to the potential hazard. The ASIL is also assigned to the safety goal(s) formulated to prevent or mitigate the potential hazard, in order to avoid unreasonable risk. Risk reduction (safety) requirements are then derived from these safety goals and inherit their ASIL” [16].

Through brainstorming and consulting with several groups of people, we have ended up with the following hazards. It is important to say that this is only for Lead-Acid battery. Because the company aspires to adapt this project for Lithium-Ion batteries in the future, that would be significantly different and possibly would have ended with different ASIL determination.

Component	Failure Mode	Effect on the item
Microcontroller	FM1: Sensors are damaged	Hazard 1: Electronic board gets wrong or no input
	FM2: Not grounded	Hazard 2: Possibility of damage
	FM3: Bluetooth transmitter is damaged	Hazard 3: Electronic board does not send an output
	FM4: Microprocessor fails in executing instructions	Hazard 4: No results available
	FM5: Input signal beyond limits	Hazard 5: Possible damage on electrical components
Interfaces/Cables	FM6: Cables not well-isolated	Hazard 6: Noisy input
	FM7 : Bluetooth connection failure	Hazard 7: User cannot be informed

HMI/Mobile App	FM8: Connection failure/error in the application FM9: App fails to store or show the data	Hazard 8: User cannot be informed Hazard 9: No serious damage
----------------	--	--

Table 11. Hazard analysis

Hazard	Effect	Comment
Hazard 1:	Severity: 0 Exposure: 4 Controllability: 1	Wrong information possibility without any serious damage Sensors are always on when the item is on Simply controllable
Hazard 2:	Severity: 1 Exposure: 4 Controllability: 1	Possible damage Electronic board is always on when the item is on Simply controllable
Hazard 3:	Severity: 0 Exposure: 2 Controllability: 1	No possible damage Bluetooth is not always working, only time to time Simply controllable
Hazard 4:	Severity: 0 Exposure: 3 Controllability: 1	No damage possible when there is no output Microprocessor often is working when item is working Simply controllable
Hazard 5:	Severity: 1 Exposure: 3 Controllability: 1	Possible damage on electronic board Electronic board often is taking input from sensors Simply controllable
Hazard 6:	Severity: 0 Exposure: 3 Controllability: 1	No possible damage Cables are often transmitting electrical energy Simply controllable
Hazard 7:	Severity: 0 Exposure: 2 Controllability: 2	No possible damage Bluetooth is not always working, only time to time Can be difficult to control

Hazard 8:	Severity: 0	No possible damage
	Exposure: 3	Item often is working
	Controllability: 2	Can be difficult to control
Hazard 9:	Severity: 0	No possible damage
	Exposure: 2	App is not always connected
	Controllability: 2	Can be difficult to control

Table 12. Risk assessment

In addition, the situational analysis, i.e. analyzing different situations would not be needed because in this application clearly the main and by far most important and dangerous situation is the cranking phase, i.e. engine start. In our case we see that the highest level of risk is given by Hazard 2: $Controllability(1) + Severity(1) + Exposure(4) = 6$. From the table we can determine that this projects ASIL is QM(Quality Management).

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 81. ASIL Selection

Quality Management leaves us with no requirements from ISO26262. This conclusion will make a significant difference from the previous chapter where we determined the example as ASIL A. Now, we will have way more tolerance on doing things but keeping in mind that if something is possible which might give us more robustness in safety point of view, we will set that as something to be done. Also durability, quality and reliability must be taken into account.

Concept model

Brain Technologies has provided us with their – what we call ‘Concept model’. In this subchapter we will explain the model, i.e. the work that has been already done, in which way it is done and then modify the model as it should be done with respect to MAAB guidelines. After that we set our objectives and see the results to be compared with the ‘Technical Model’.

The algorithm is divided in two modules:

1. Series of extended Kalman Filters
2. Error management and selection of the best estimate

Inputs of the first module, thus the inputs of the whole algorithm are:

- Voltage
- Current
- Temperature
- Sampling time
- Initial SoC in first time algorithm runs, after that the input is the previous SoC
- Initial Voltage
- Nominal capacity

while the outputs of it, thus the inputs of the second module:

- Estimation of SoCs from different Kalman filters
- Estimated error absolute value
- *Current*

And the outputs of the second module, thus the whole algorithms:

1. SoH
2. SoC
3. Reliability, which is the value in percentage of how much we trust the results given to the user.

It is important to see the following figure, which is exactly what was given to us. It is very clear that firstly we must apply the MAAB design guidelines and then proceed further. Also, a lot of the work has been done on Matlab, which is not preferable but still we can fix it when we are in the code generation point, converting that code into MISRA standard code.

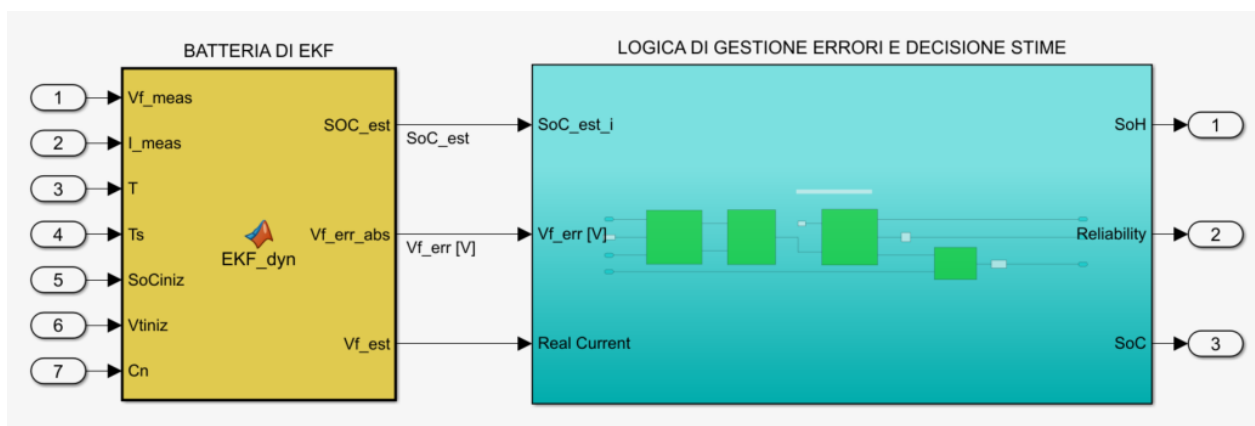


Figure 82. Concept model from customer

The inputs of the BAT-MAN algorithm in this case are given from the real data taken by the real batteries. Certainly, the team before has also run the algorithm with the battery model. The illustration below shows the architecture of the algorithm:

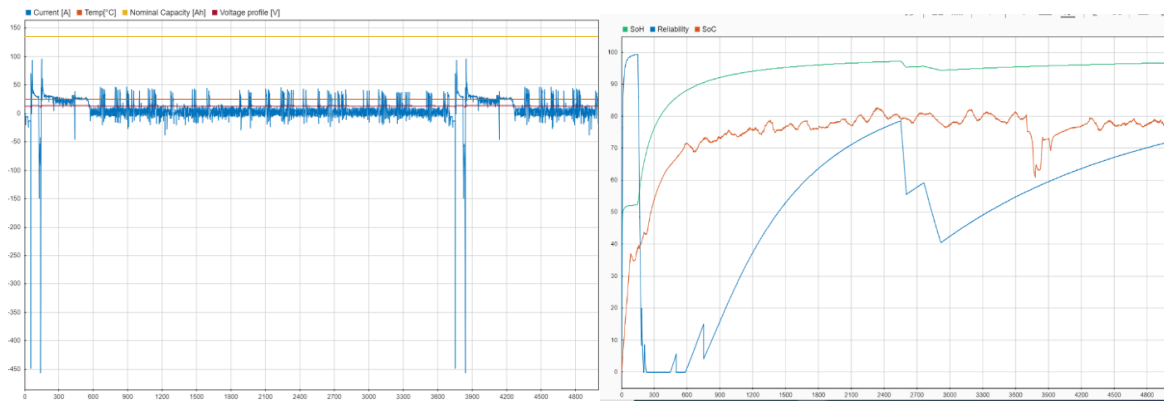


Figure 83. Inputs (left) and Outputs (right)

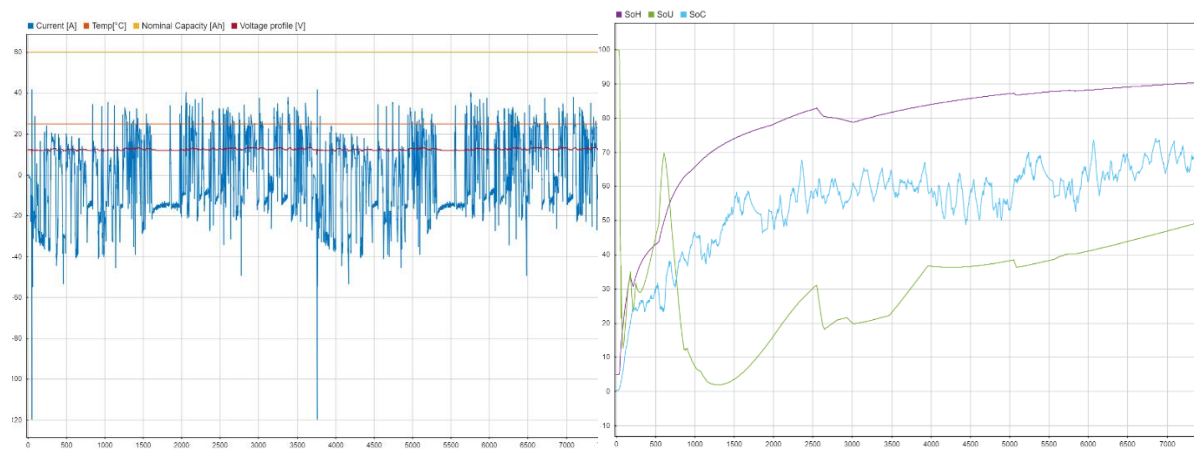


Figure 84. Inputs (left) and Outputs (right)

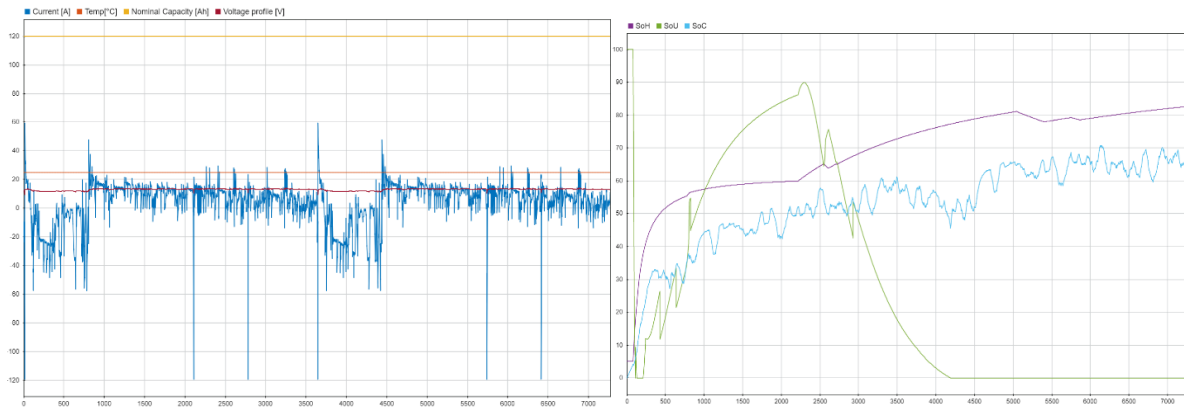


Figure 85. Inputs (left) and Outputs (right)

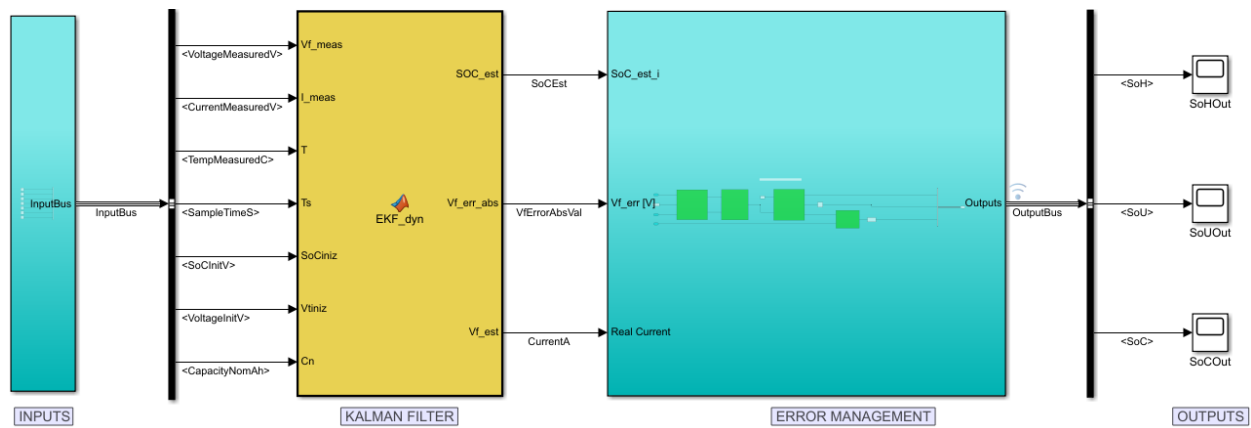


Figure 86. Concept model modified

Architecture & Design

Techniques – System is defined as QM

Methodologies – Modular and reusable architecture

Artefacts – Architecture components definition and interfaces

Safety aspect –

- Some techniques recommended by ISO26262 for ASIL A, even though the system is QM

As we understood in the example tutorial, there are features that must be specified. For BAT-MAN they are as follows:

- The environment: The item will be placed in a vehicle
- There are no permitted ways of use
- Only one mode of operation specified
- Neither of the functions are function-call
- Input voltage is in the range of +15 V.
- Input current is in the range of ± 900 A
- Input temperature is not specified. It is considered as environment effect.

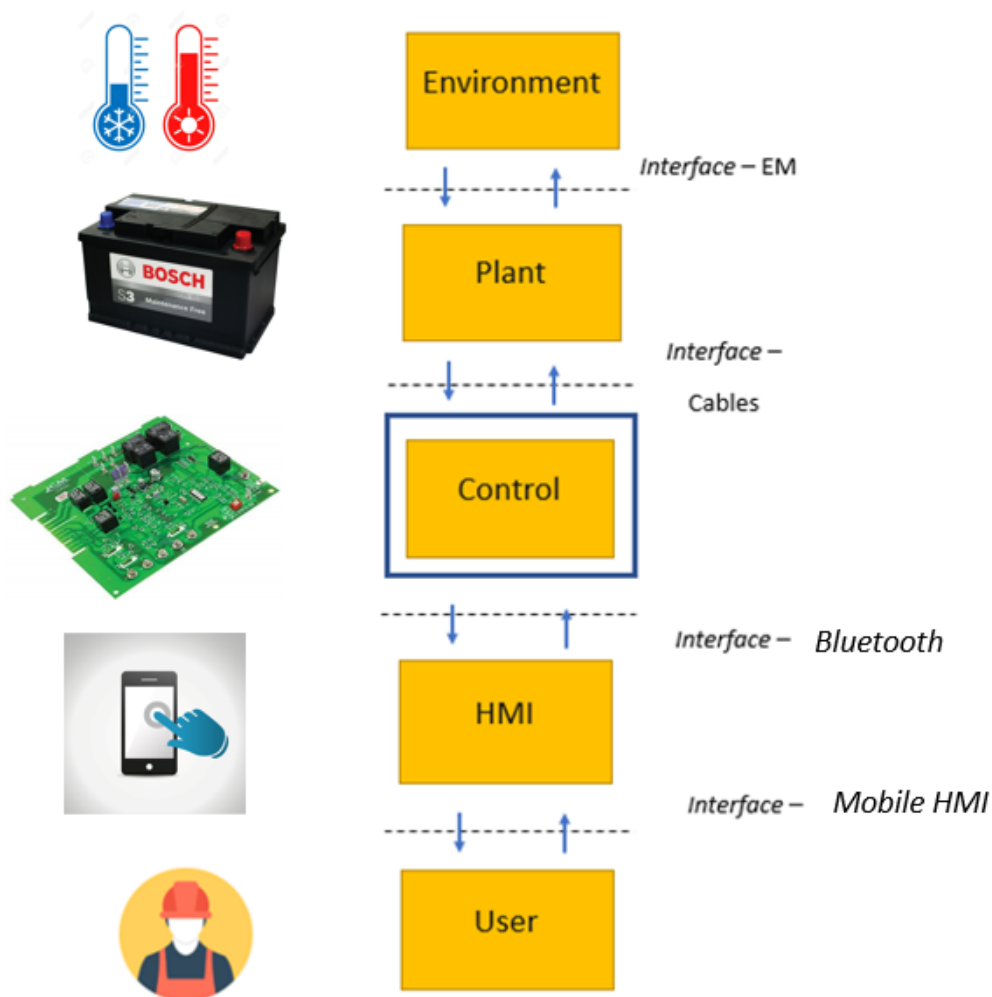


Figure 87. Architecture illustration

BAT-MAN has two operation modes:

- Device is connected to the Mobile app
- Device is not connected to the Mobile App

Environment

Lead-Acid batteries are deeply affected by the environment temperature, thus it is necessary to design our model including temperature input as disturbance in the Environment module of our architecture. Due to a lot of electro-chemical interaction that happen in this kind of battery the temperature will affect the aging of it. This effect can be described as the Arrhenius equation.

“Svante Arrhenius was a Swedish scientist who discovered the life of lead-acid batteries is affected by variations in temperature. He established that for every 10°C increase in temperature the battery life would be halved. Therefore, as an example, it follows that if the life is 30 years at 15°C then at 25°C the life will be 15 years. The equation also suggests that at 5°C the life will be 60 years but unfortunately other things come into play when batteries are very old, typically over 30 years, and the Arrhenius equation is only really valid between about 15°C and 40°C for operational batteries”. [17]

The data sheet of ‘Hawker Cyclon’ battery shows this following graph:

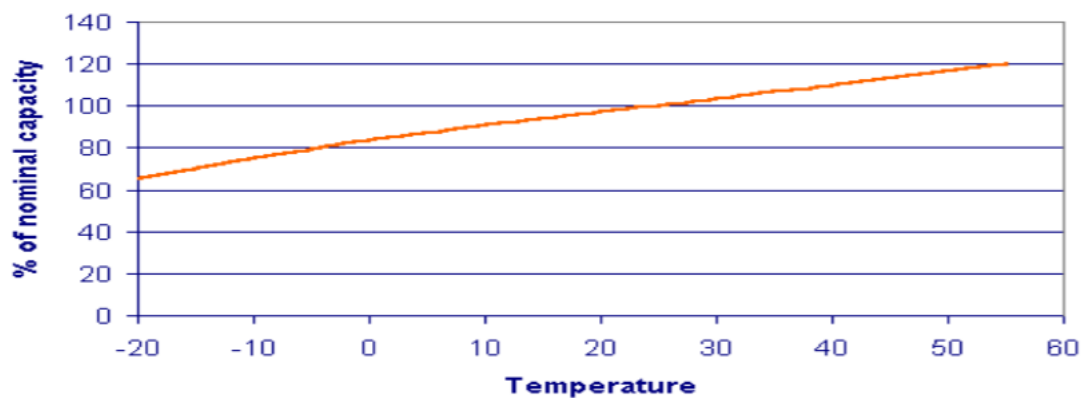


Figure 88. Relation between Cn and Temperature [17]

Nominal operating temperature is 20°C. Thus, we must know that when we talk about different characteristics of the battery, it is given that the operating temperature is 20°C.

To understand better we might take an example. Let us suppose that a battery operates in different temperatures in different months. Then, we compute the aging of that battery in one year. The following figure shows a typical operating temperature of a battery designed according to IEC 60896 that has a life of 12 years:

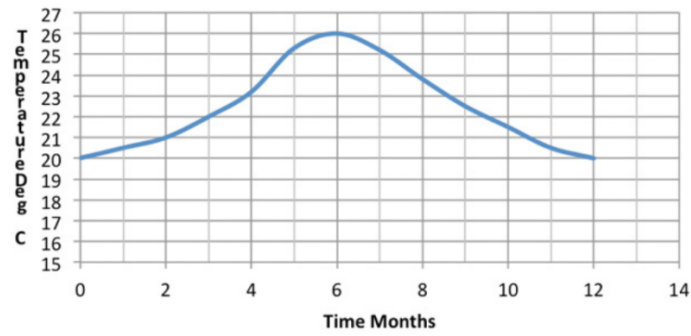


Figure 89. Battery typical operating points [17]

Taking to account the Arrhenius equation the following table was formed:

Month	Average Temperature	% of Nominal Life	Calculation (100% / Temp %)	Aging Effect	Aged days in the Month
1	20.5	95	100/95	1.05	31.58
2	21	92	100/92	1.09	32.61
3	22	87	100/87	1.15	34.48
4	23.2	80	100/80	1.25	37.50
5	25.3	70	100/70	1.43	42.86
6	26	65	100/65	1.54	46.15
7	25.2	70	100/70	1.43	42.86
8	23.8	77	100/77	1.30	38.96
9	22.5	85	100/85	1.18	35.29
10	21.5	90	100/90	1.11	33.33
11	20.5	95	100/95	1.05	31.58
12	20	100	100/100	1.00	30.00
TOTAL AGED DAYS PER CALENDAR YEAR					437.21

Figure 90. Ageing in different temperatures [17]

We see that the battery does not age the same in different temperatures. With the typical operating temperatures, we can conclude that in 365 days the battery ages about 437 days. Thus, if the life expectancy in theory is 12 years as this considered battery, in the reality it would be approximately 10 years. This simple example makes it very clear to us the effect of temperature cannot be neglected and certainly must be taken care of.

Plant

In this phase we shall define the plant (Lead-Acid battery) in more specific and technical detail. The location of the plant in the vehicle can vary from the producer of the vehicle. Usually, in most of the cars Lead-Acid battery is located in the cars hood. But nevertheless, there are cars that have the battery located inside cars trunk. Thus, both cases have to be considered if needed in the future of the development.



Figure 91. Battery location in the trunk of the car

Firstly, the customer that has already done the concept model mostly has been using the real data that were collected from different kind of batteries. But surely that is not enough and a model must be defined. Certainly, there are a lot of different ways to do it, such as:

- 1) Electrochemical modelling
 - a) Shepherd model
 - b) Nernst model
 - c) Unnewehr universal model
- 2) Equivalent circuit model
 - a) Rint model
 - b) Thevenin model
 - c) DP model

Electrochemical modelling needs a lot of memory, time consuming, computationally expensive, thus it is out of question immediately. The customer, for now has chosen to use Thevenin model. A simple Thevenin model is shown in the figure below:

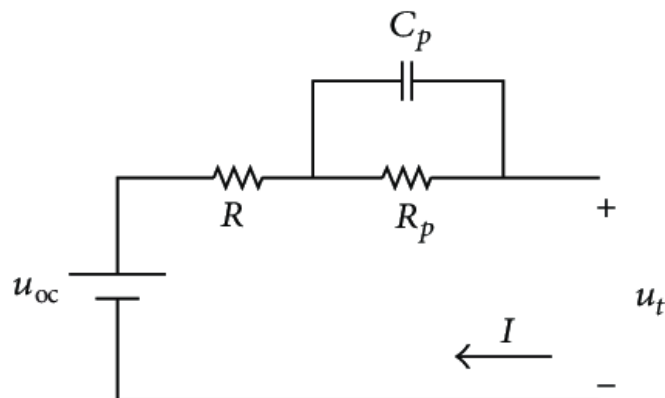


Figure 92. Thevenin model

Data flow from the Plant to Control:

- Voltage
- Current
- Temperature

Data flow from Control to Plant:

- None

Control - Hardware architecture

“Architecture is often seen as the spine of each product. ISO 26262, part 1, chapter 1.3 describes architecture as the representation of a vehicle system, of functions, systems or elements, which are identifiable through components, their distinctions, intersections and allocation to electric hardware and software. The functional concept (ISO 26262, part 1, chapter 1.50) is mentioned as the basis for the definition of vehicle systems. According to the glossary the functional concept is compiled from specifications of intended functions and their interactions in order to achieve the desired behavior. Therefore, it is evident that architecture needs to fulfill two requirements. It provides the product structure and its intersections as well as the foundation for the description of the technical behavior. Each component or element and their intersections ask for certain requirements. The intended behavior as well as the behavior in case of a failure has to be specified. This forces us to plan and define all levels of abstraction, perspectives, intersections as well as their desired technical behavior in advance.” [13]

As we already know, our system takes as input the current, voltage and temperature thus the need to have a way to acquire those data becomes necessary. As a first hardware component we must think of components that are able to:

- Measure Voltage in the terminals of the plant
- Current flowing
- Temperature

Secondly, as a requirement from the customer we know that the output data must be sent by Bluetooth to HMI. Thus, concerning the microcontroller we shall choose a:

- Wireless microcontroller

Thirdly, again from the customer we know that the output data must be saved even if there is no connection between User HMI and control. Thus, as a component we must add:

- Flash Memory

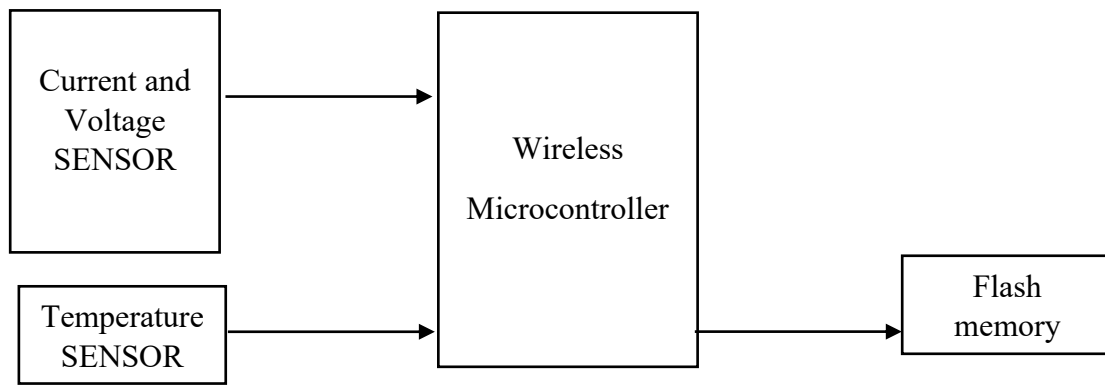


Figure 93. Hardware architecture

In addition of all this, for safety reasons as mentioned, another component that must be added is:

- Protection circuit

Concerning the power supply of the hardware, a clever choice is to use the power of the Plant, not only acquiring the data of it.. This of course would require another component, a Converter.

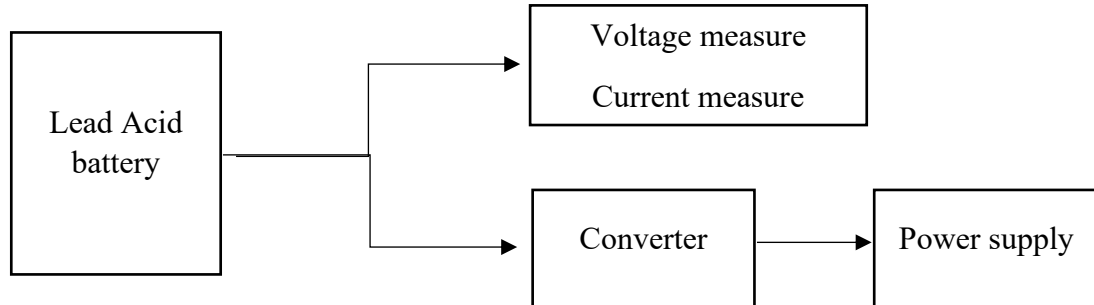


Figure 94. Power supply architecture

“Each electric component creates heat, which has to be directed outside. The thermos conductivity of the housing plays an important role here. Overheating is a major cause for fire in control units. This is explicitly mentioned in ISO 26262 since this could be a failure function of the electronic.” [13]

Starting from the information that above mentioned phenomena is explicitly mentioned in ISO 26262:

- A compatible housing must be designed to avoid thermos conductivity

Control - Software architecture

The software architecture for BAT-MAN project can be divided into 3 main modules:

1. Acquire the data
2. Process the data
3. Transmit the data through Bluetooth

Acquire data module and transmit the data module will be hand-written code. Process the data, i.e. BAT-MAN algorithm will be obtained through automatic code-generation via Simulink. All of the modules will be part of one block of code without function calls or interrupts to execute the tasks.

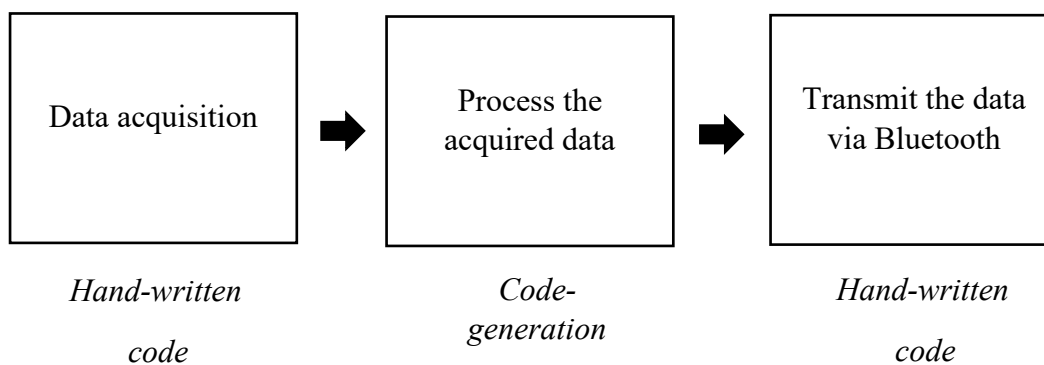
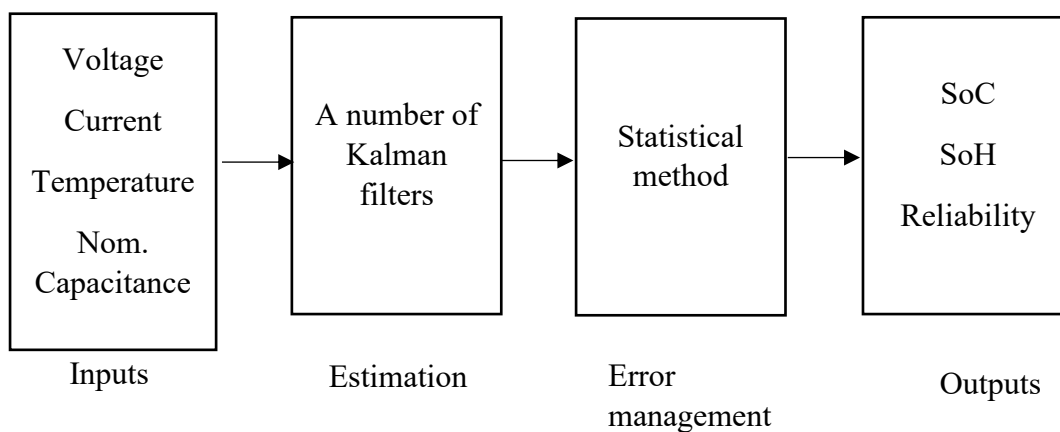


Figure 95. Software architecture

Concerning the module of transmitting the data via Bluetooth, there are several choices given by different companies such as Texas Instruments with CC26xx wireless MCUs or Renesas Synergy with SK-S7G2 Synergy MCUs or the PK-S5D9 Synergy MCUs. Thus, the quality and safety of the code are guaranteed by these companies that have already done the tests and fulfilled their requirements.

Application architecture

Figure 96. Application architecture



Kalman filtering estimation flowchart:

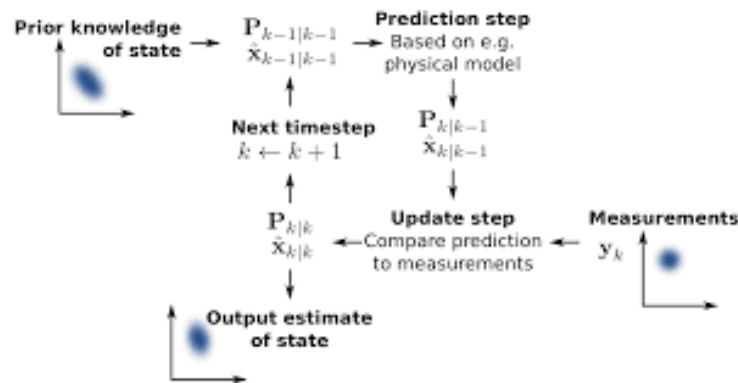


Figure 97. Kalman filter working principle

Error management flowchart:

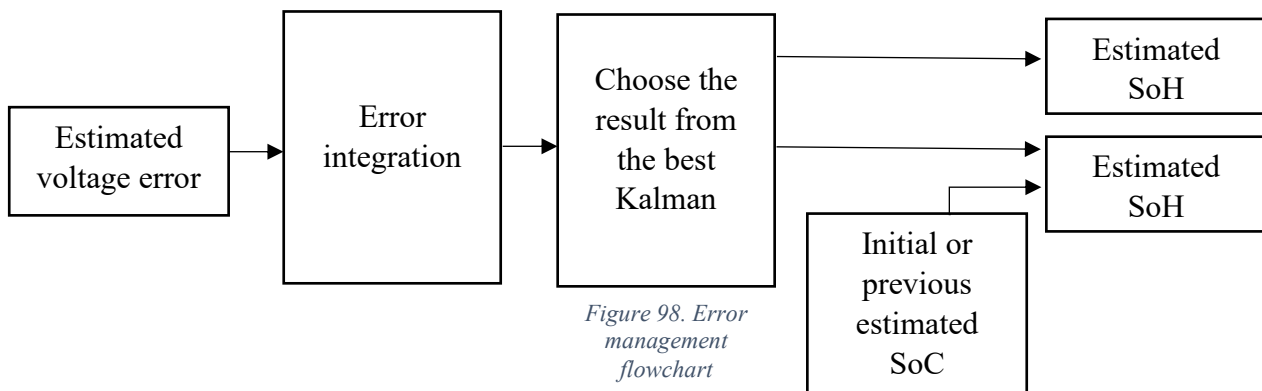


Figure 98. Error management flowchart

Even though our system is defined as QM, we can, as a safety culture, set the following requirements for the software development:

- Compiled code size must be shown
- Memory consumption, i.e. will the allocated memory in the hardware is enough, robust or not enough must be checked and tested
- Interrupts, Function calls, Periodic, overall Scheduling
- Operating system will be preemptive or non-preemptive
- Application flow, number of tasks, number of variables
- No recursive loops
- Code in MISRA-C, Optimized

Interfaces

Again, we must define all the interfaces of the system according to the 4 different types of them.

Physical interface

- The item will be placed in a vehicle
- Environment temperature
- Voltage range from 13V
- Current range +- 900A

Energy interfaces

- Electric energy
- Thermal energy
- Voltage measurement
- Current measurement
- Energy provision via cables

Material transfer (interface)

- No material transfer

Information interfaces

- Signal processing
- Analog input to ADC to Bluetooth antenna to Mobile phone
- Bus or communication systems CAN or Ethernet

Environment - Plant interface:

- Energy interface, i.e. thermal energy/temperature

Plant – Control interface:

- Energy interface, i.e. Electric energy via cables
- Information interfaces, i.e. Signal processing/Measurement, Analog to AD Converter

Control – HMI:

Information interfaces, i.e. Bluetooth communication

Requirement 3-2

“The housing has to be constructed in a way that it fits in the vehicle, provides protection against humidity and dirt ensures that cables can be fixed, fulfills the EMC requirements and allows the arising heat dissipates.” [13]

Electromagnetic Compatibility known as EMC is a test for electronic devices to make sure that they do not emit radiated and conducted emissions above a certain level set by regulators. EMC takes care of 3 different categories of issues:

- *Emissions*
- *Susceptibility*
- *Coupling*

Emissions include electromagnetic energy generation by certain sources. EMC takes care of the unwanted emissions and how we must tackle them. Susceptibility, studies the “victim” electronic component, i.e. the vulnerability of the component to be affected by certain unwanted electromagnetic emissions. And lastly, coupling is the phenomena where the unwanted emissions release by a source and reaches the sink or the “victim”.

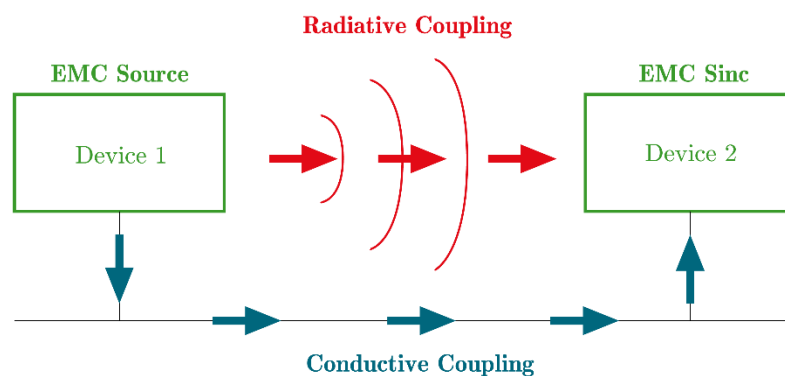


Figure 99. EMC Coupling

To be able to tackle these problems, first we must understand different kinds of interferences and sources of them, and then see which on of those we have in our board. There are two types of interferences, continuous and transient interferences. Following table has been created to manage it easier:

Continuous interference	Sources	BAT-MAN PCB
Audio frequency	Power supply units	<i>PCB is supplied directly by the plant</i>

	Audio processing equipment	Does not contain
	FM radio transmission type	Does not contain
Radio frequency	Radio frequency transmissions	Does not contain
	TV or radio receivers	Does not contain
	Microcontrollers	<i>Contains</i>
Broadband noise	Solar activity	Does not contain
	Arc welders	Does not contain
	Spread-spectrum mobile telephony	Does not contain

Table 13. EMC Analysis I

Transient interference	Sources	BAT-MAN PCB
Electromagnetic pulse	Switching the circuit, or other components	<i>Contains</i>
	Power line pulses	Does not contain
	Electrostatic discharge	Possible
	Lightning electromagnetic pulse	Does not contain
	Nuclear electromagnetic pulse	Does not contain
Repetitive Electromagnetic pulse	Electrical motors	Does not contain
	Ignition systems	Does not contain
	Continual switching actions of digital circuit	<i>Contains</i>

Table 14. EMC Analysis II

The countermeasures that we can take for reducing the emissions are:

- Grounding
- Quality cables

- Housing
- Decoupling in critical points
- Avoid as much as possible the unnecessary switching of the circuit or other components
- Design to operate at lower levels of signals

Furthermore, we can increase the measures of the second category, susceptibility of components. Such a measures can be:

- Protection circuit
- Fuses

Housing and LIN Connection

Housing, as required from EMC requirements is built in a 3D Printer. The following figure shows the case implemented in the hardware and LIN connection:



Figure 100. Housing and LIN Connection

Modern cars use different type of networks such as:

- CAN (Controller Area Network)
- MOST (Media Oriented Systems Transport)
- FlexRay
- Ethernet
- LIN (Local Interconnect Network)

Why LIN?

LIN connection we are using in BAT-MAN is the most common network used in Automotive industry because of its simplicity and low cost. It is a 1-wire bus and it is mostly used in the applications where high speed is not needed, such as: [www.electronicdesign.com REFERENCE]

- Power door locks
- Power windows
- Power seats
- Power mirrors
- Heating and air conditioning control
- Interior lights
- Seat heaters

That is why LIN connection is very suitable for our application. Considering safety, LIN is standardized by:

- ISO17897

This International Standard specifies the requirements for setting up the interchange of digital information between onboard Electronic Control Units (ECUs) of road vehicles and suitable diagnostic testers. This communication is established in order to facilitate inspection, test diagnosis and adjustment of vehicles, systems and ECUs. [3]

- ISO9141

ISO 17987-1:2016 gives an overview of the structure and the partitioning of ISO 17987 (all parts). In addition, it outlines the use case where the ISO 17987 (all parts) will be used. The terminology defined in ISO 17987-1:2016 is common for all LIN communication systems and is used throughout ISO 17987 (all parts). It has been established in order to define the use cases for LIN. [3]

Production development – Software level

According to ISO26262 we shall define the tools, Techniques, Methodologies, Artefacts and Safety aspect:

Tool – MATLAB Simulink, Simulink embedded coder

Techniques – System is defined as QM

Methodologies – Model-based Design, MAAB Guidelines

Artefacts –

- Application code
- SIL testing

- Compare MIL to SIL

Safety aspect –

- Some techniques recommended by ISO26262 for ASIL A, even though the system is QM
- Generated code will be compliant with MISRA-C

2. Technical model

We shall start from the first module that is Environment which contains the temperature input that we already defined in the Architecture. In Simulink, we define the block as a Constant block that will get the value from the Matlab data file. What concerns scheduling of the model, we do not have any interrupts. The application is run as a whole and its timing will be done in the firmware that will be given by the supplier together with the hardware.

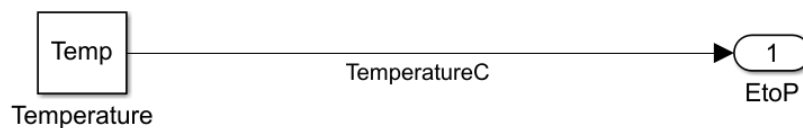


Figure 101. Temperature in Environment module

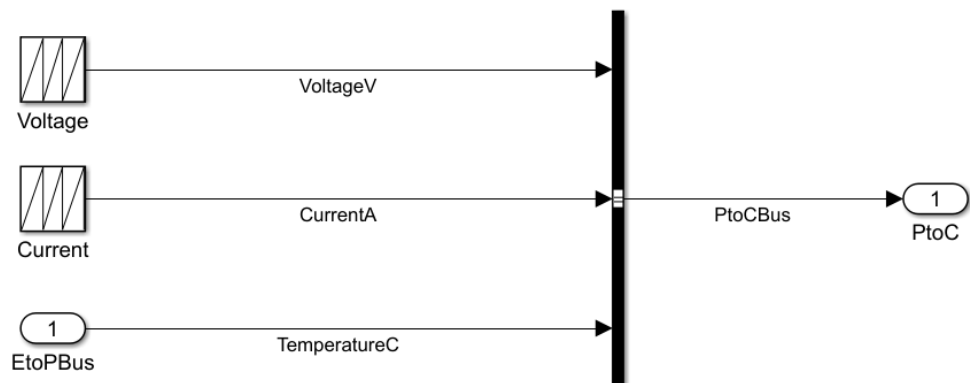


Figure 102. Plant module

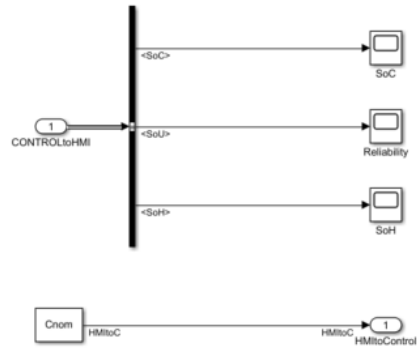


Figure 103. HMI module

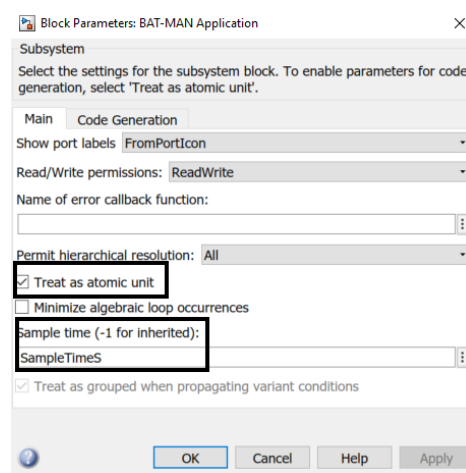


Figure 104. Parameters of BAT-MAN Application

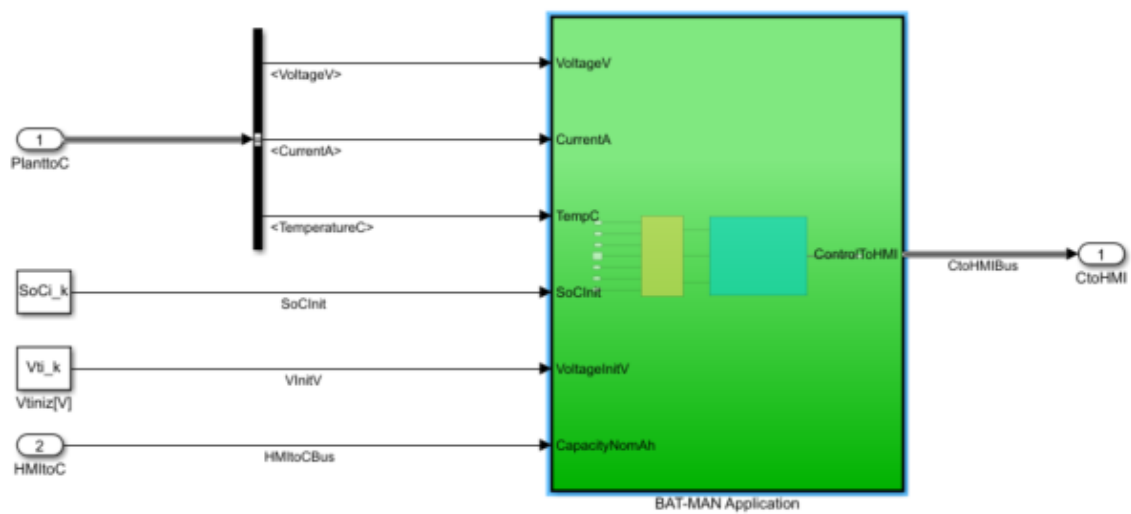


Figure 105. Control Module

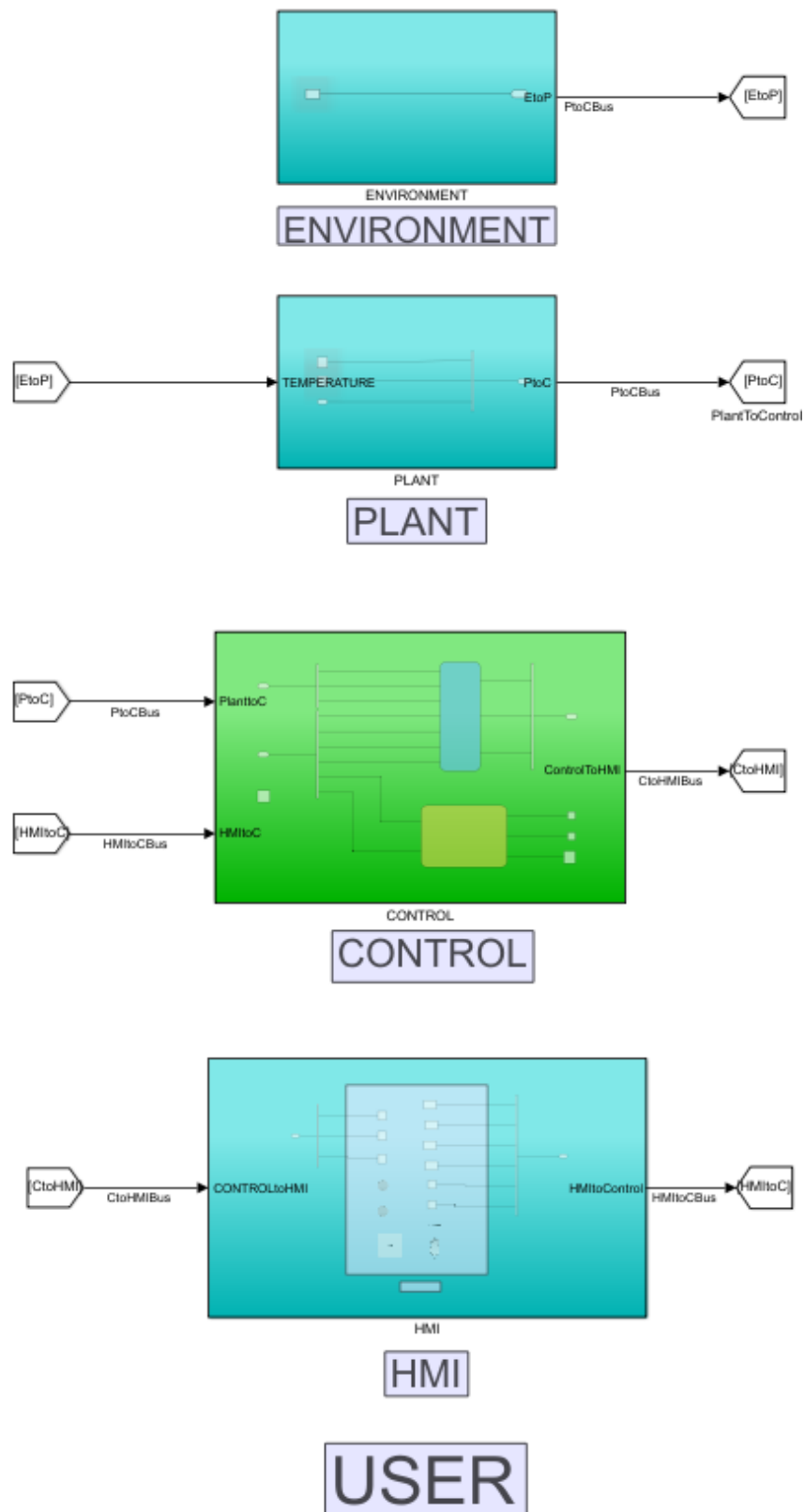


Figure 106. Architecture in Simulink

Control frame

As we already know, the Control frame contains the A/D Conversion. In Simulink model we must test different number of bits to see the results and thus setting how many bits we need. The following figures show the results with 12 and 16 bits. Obviously, there was no need to go lower then 12 bits since the results are not satisfying.

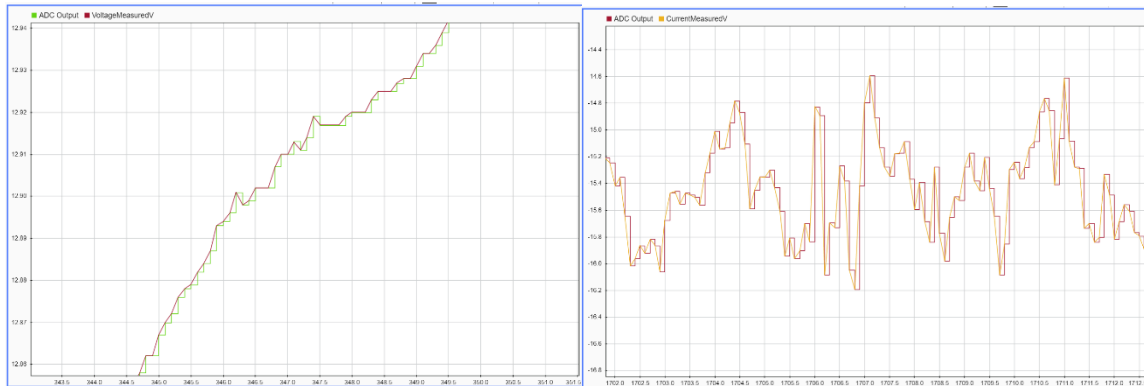


Figure 107. 16 bit ADC Output of Voltage (left), 16bit ADC Output of Current (right)

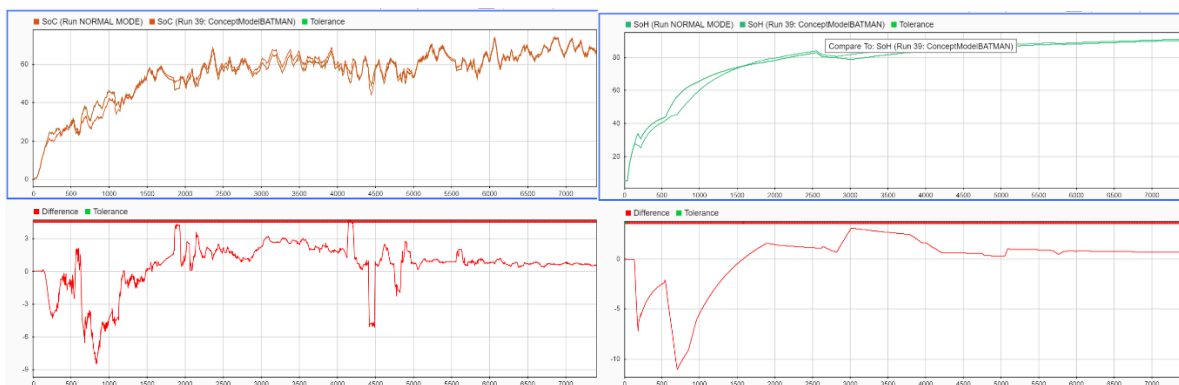


Figure 108. SoC with 12 bit quantization (left), SoH with 12 bit quantization (right)

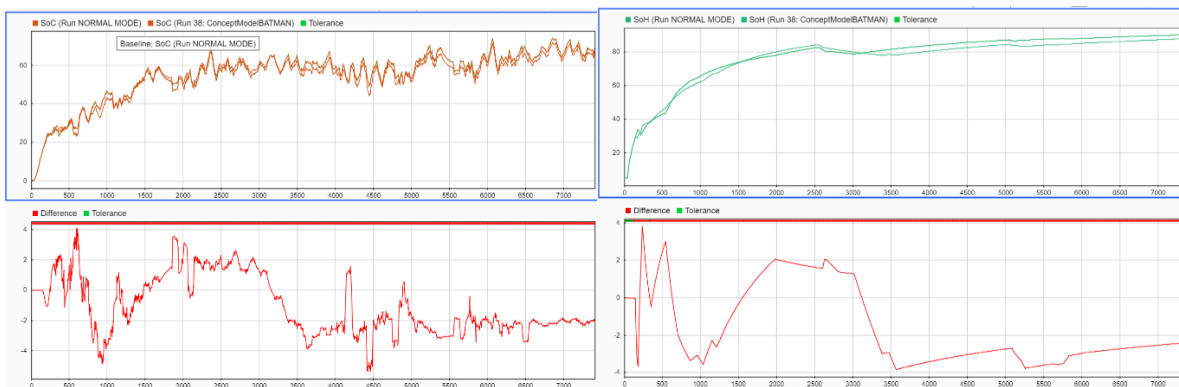


Figure 109. SoC with 16 bit quantization (left), SoH with 16 bit quantization (right)

As we can see from the figures, 12 bits are not enough, thus 16 bits ADC must be implemented.

SIL

SIL Model has been done in a same way as in the example. SIL model for BATMAN Application is shown below:

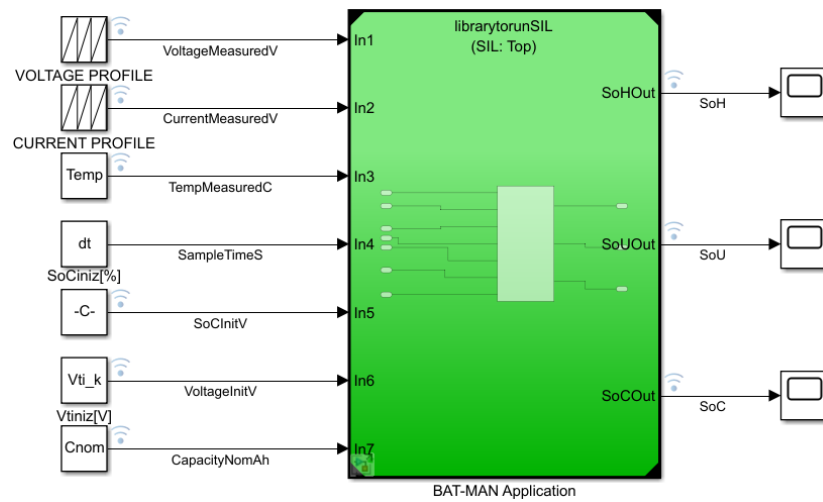


Figure 110. SIL Model for BATMAN

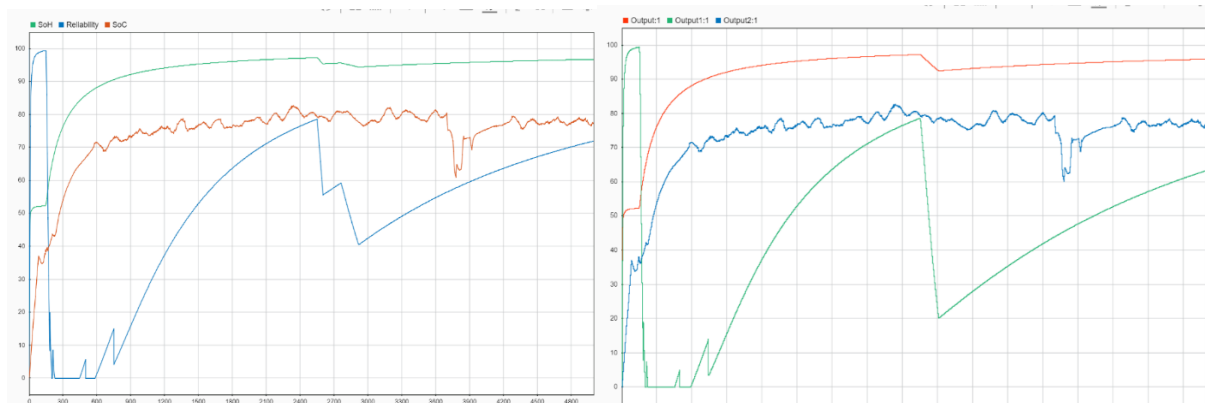


Figure 111. Data set 1: Simulation (left) vs SIL (right)

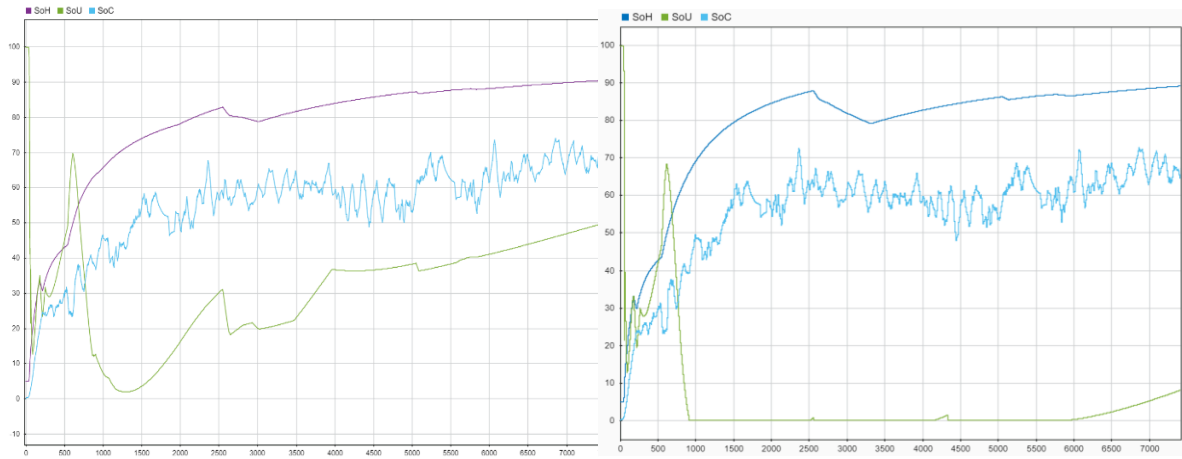


Figure 112. Data set 2: Simulation (left) vs SIL (right)

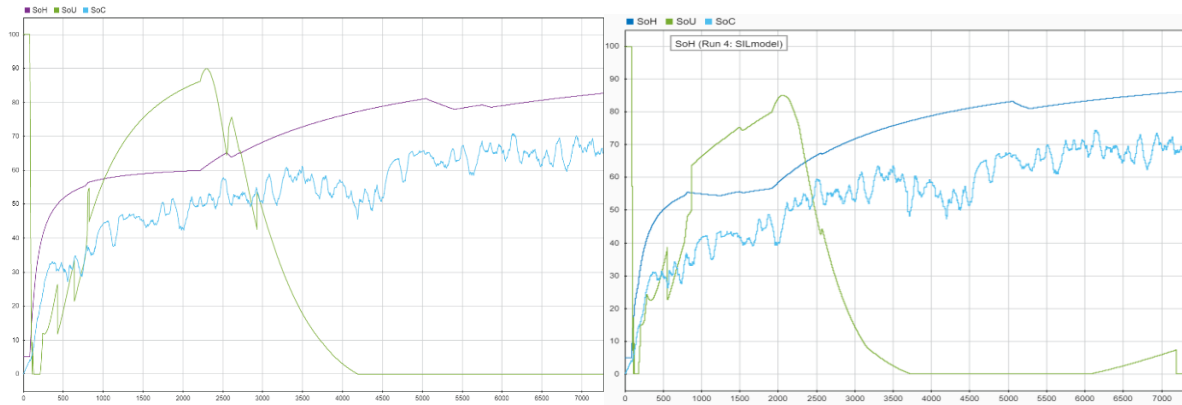


Figure 113. Data set 3: Simulation (left) vs SIL (right)

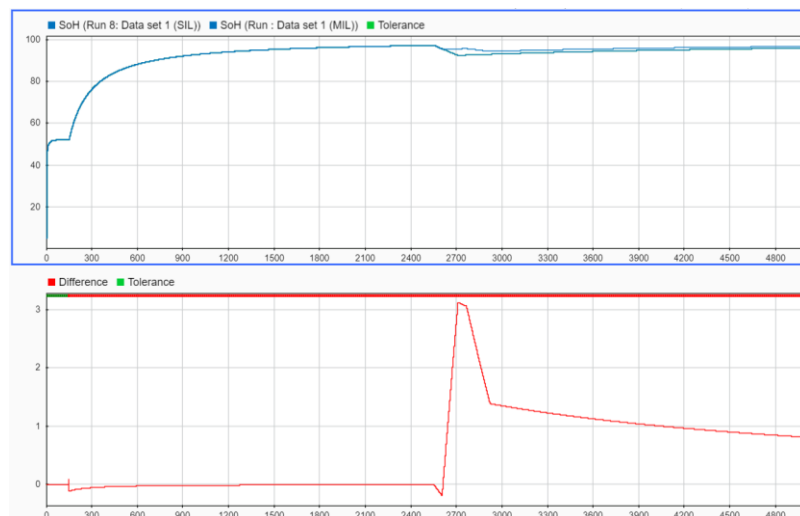


Figure 114. Data set 1: Difference between SoH in simulation and from SIL

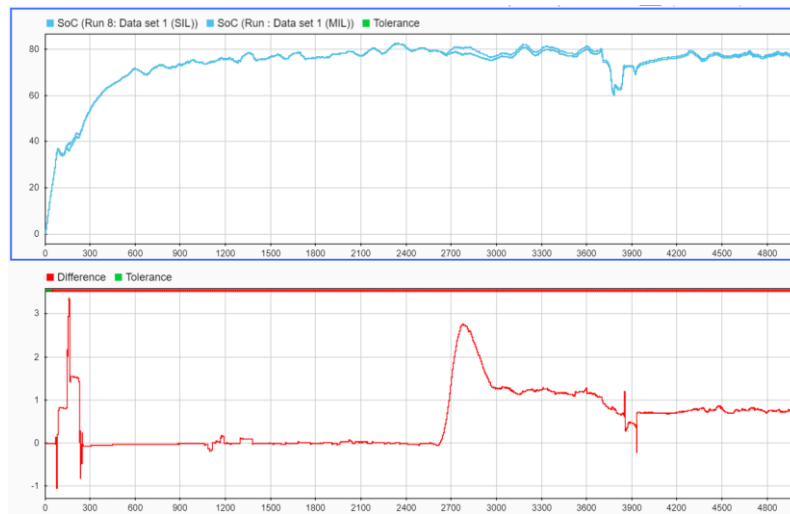


Figure 115. Data set 1: SoC difference between Simulation and SIL

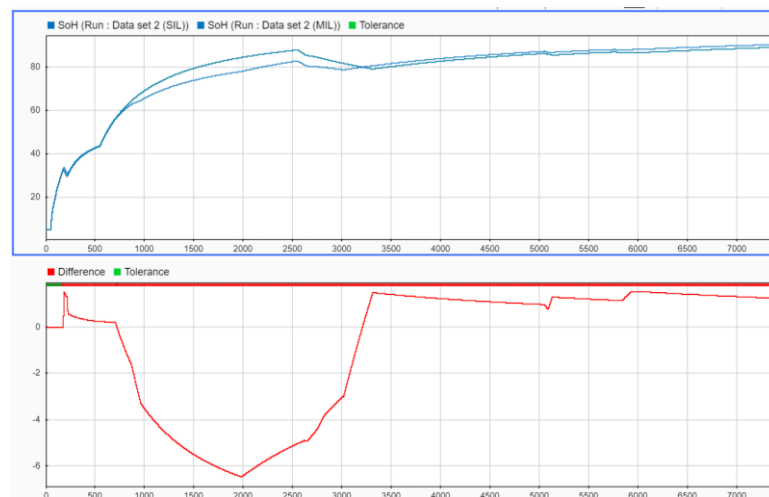


Figure 116. Data set 2: SoH difference between Simulation and SIL

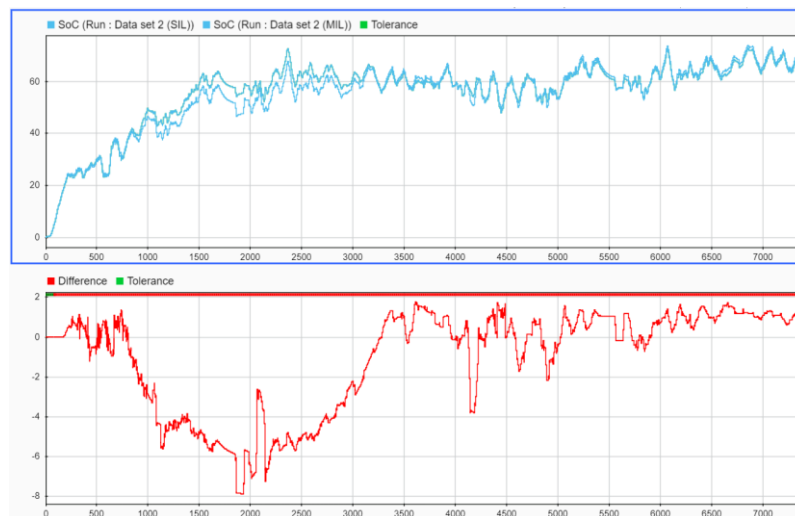


Figure 117. Data set 2: SoC difference between Simulation and SIL

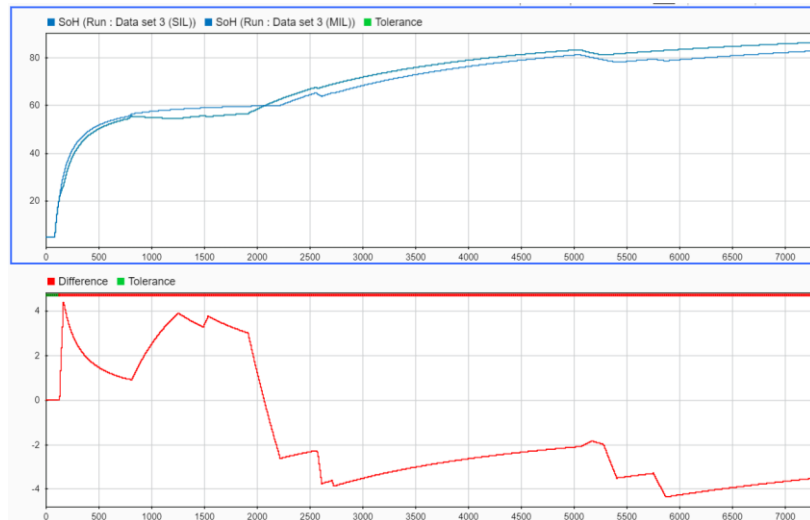


Figure 118. Data set 3: SoH difference between Simulation and SIL

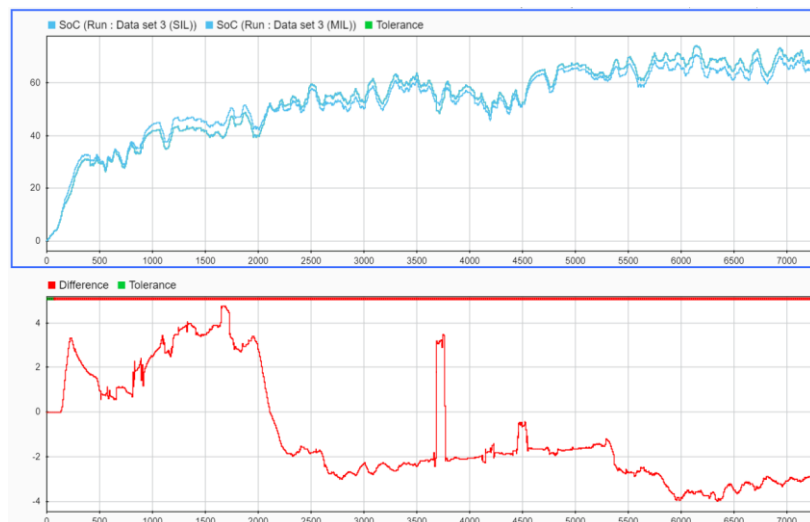


Figure 119. Data set 3: SoC difference between Simulation and SIL

Use code execution profiling to: [11]

Determine whether the generated code meets execution time requirements for real-time deployment on your target hardware.

Identify code sections that require execution speed improvements.

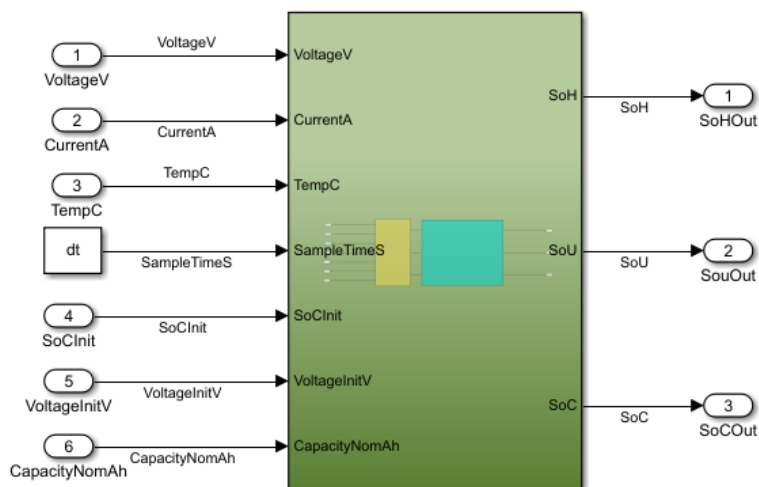
2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls
initialize	18320	18320	18320	18320	1
step [0.1 0]	1867719	18836	1867719	18836	72711
terminate	371	371	371	371	1

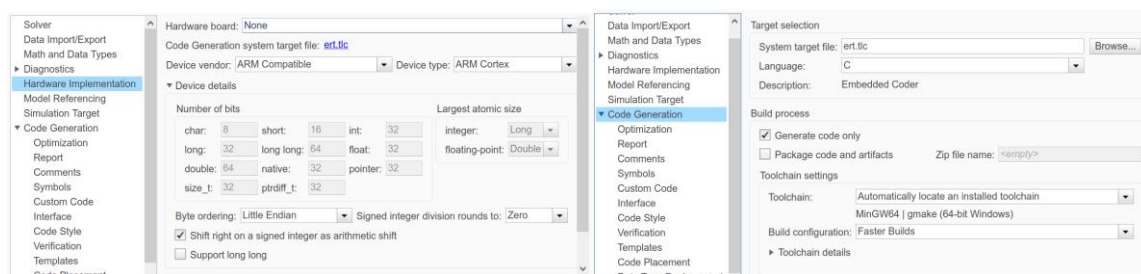
Figure 120. Profiled sections of code

Production Model

The purpose of the Production Model is adapting our Control Module to the target hardware and then initiate the code generation specific to it. The signal names are very important, as mentioned a lot of times, because we will be dealing with them in Code Composer Studio where we have to integrate this generated code to the firmware architecture. Firstly, I have taken the Control Module from the Technical model and export it into a blank model that will be our Production Model. The screenshot of the model is shown below:



Our target is ARM Cortex M MCU, thus we must go on according to this characteristic. In ‘Configuration Parameters’ in Simulink we set the device to ARM Cortex, and the tool chosen is Embedded Coder, as shown below:

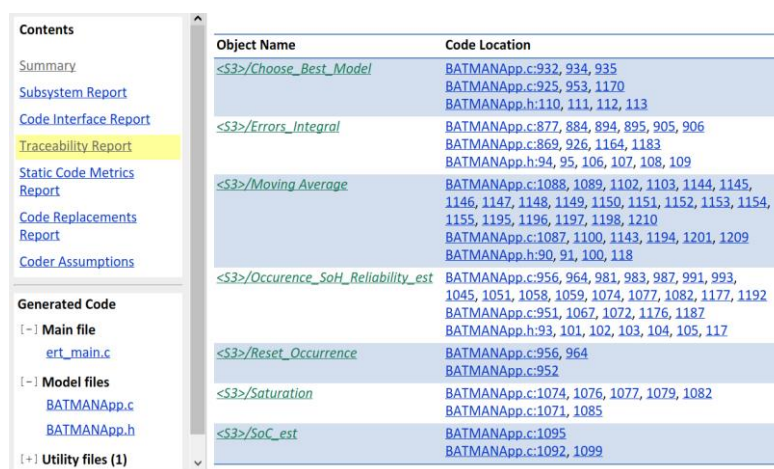


All the other parameters are set as default except:

- Code interface packaging: Reusable function
- File packaging format: Compact
- Optimization level: Maximum, and as Priority: Minimize RAM
- Static code metrics ON

Furthermore, concerning ISO26262 the code traceability and Static code metrics are required. Both of these are provided by Simulink Embedded Coder.

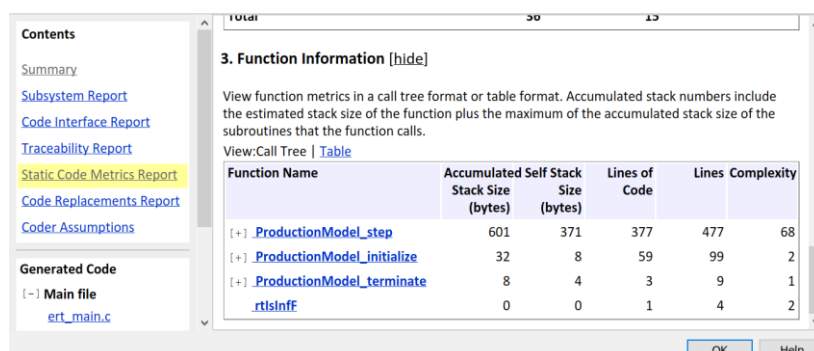
Traceability Report:



Object Name	Code Location
<S3>/Choose_Best_Model	BATMANApp.c:932, 934, 935 BATMANApp.c:925, 953, 1170 BATMANApp.h:110, 111, 112, 113
<S3>/Errors_Integral	BATMANApp.c:877, 884, 894, 895, 905, 906 BATMANApp.c:869, 926, 1164, 1183 BATMANApp.h:94, 95, 106, 107, 108, 109
<S3>/Moving_Average	BATMANApp.c:1088, 1089, 1102, 1103, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1195, 1196, 1197, 1198, 1210 BATMANApp.c:1087, 1100, 1143, 1194, 1201, 1209 BATMANApp.h:90, 91, 100, 118
<S3>/Occurrence_SoH_Reliability_est	BATMANApp.c:956, 964, 981, 983, 987, 991, 993, 1045, 1051, 1058, 1059, 1074, 1077, 1082, 1177, 1192 BATMANApp.c:951, 1067, 1072, 1176, 1187 BATMANApp.h:93, 101, 102, 103, 104, 105, 117
<S3>/Reset_Occurrence	BATMANApp.c:956, 964 BATMANApp.c:952
<S3>/Saturation	BATMANApp.c:1074, 1076, 1077, 1079, 1082 BATMANApp.c:1071, 1085
<S3>/SoC_est	BATMANApp.c:1095 BATMANApp.c:1092, 1099

Figure 121. Traceability report

Static code metrics:

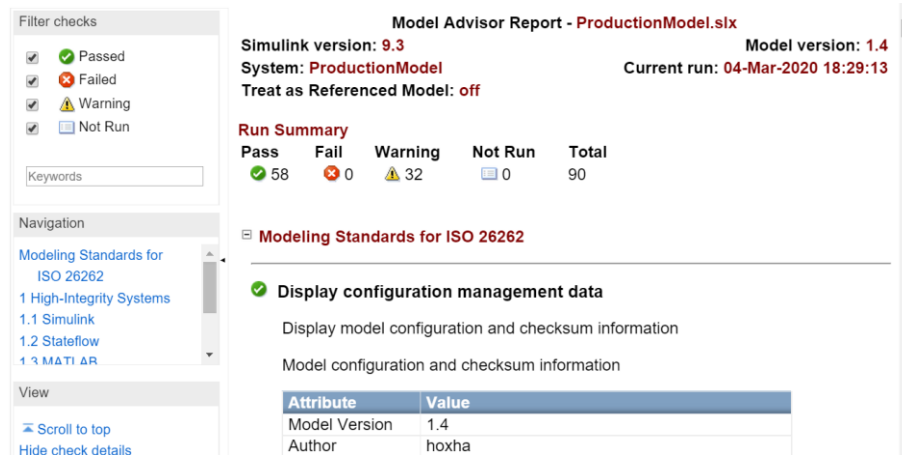


Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
[+] ProductionModel_step	601	371	377	477	68
[+] ProductionModel_initialize	32	8	59	99	2
[+] ProductionModel_terminate	8	4	3	9	1
rtIsInf	0	0	1	4	2

Figure 122. Static code metrics

Standards and guidelines check

As conclusion, results were very good and satisfactory, passing all the checks. Furthermore, the reports have been generated for each check as documentation of the process. The results figures are shown below:



Model Advisor Report - ProductionModel.slx

Simulink version: 9.3 Model version: 1.4
System: ProductionModel Current run: 04-Mar-2020 18:29:13
Treat as Referenced Model: off

Run Summary

Pass	Fail	Warning	Not Run	Total
58	0	32	0	90

Modeling Standards for ISO 26262

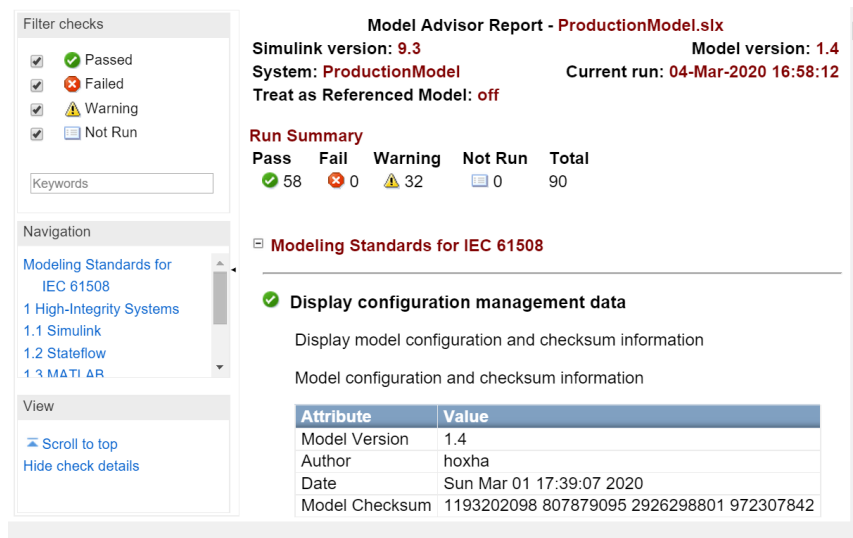
✓ **Display configuration management data**

Display model configuration and checksum information

Model configuration and checksum information

Attribute	Value
Model Version	1.4
Author	hoxha

Figure 123. ISO 26262 check report



Model Advisor Report - ProductionModel.slx

Simulink version: 9.3 Model version: 1.4
System: ProductionModel Current run: 04-Mar-2020 16:58:12
Treat as Referenced Model: off

Run Summary

Pass	Fail	Warning	Not Run	Total
58	0	32	0	90

Modeling Standards for IEC 61508

✓ **Display configuration management data**

Display model configuration and checksum information

Model configuration and checksum information

Attribute	Value
Model Version	1.4
Author	hoxha
Date	Sun Mar 01 17:39:07 2020
Model Checksum	1193202098 807879095 2926298801 972307842

Figure 124. IEC 61508 check report

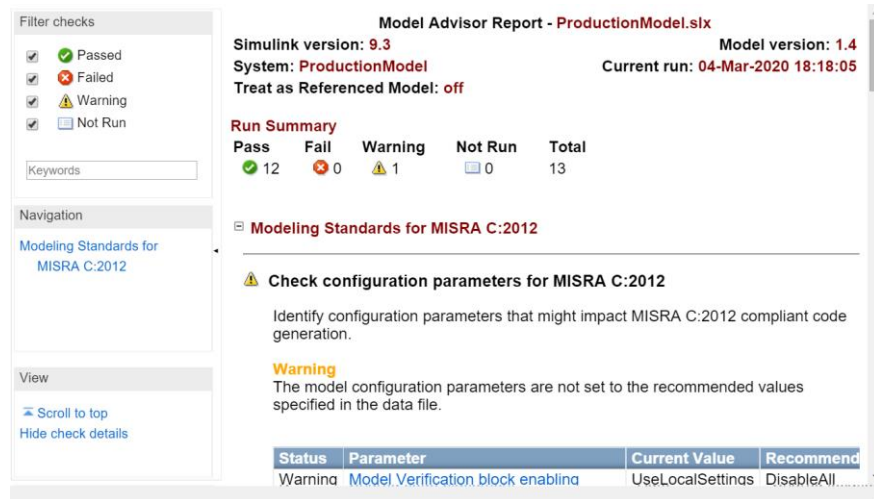


Figure 125. MISRA C check report

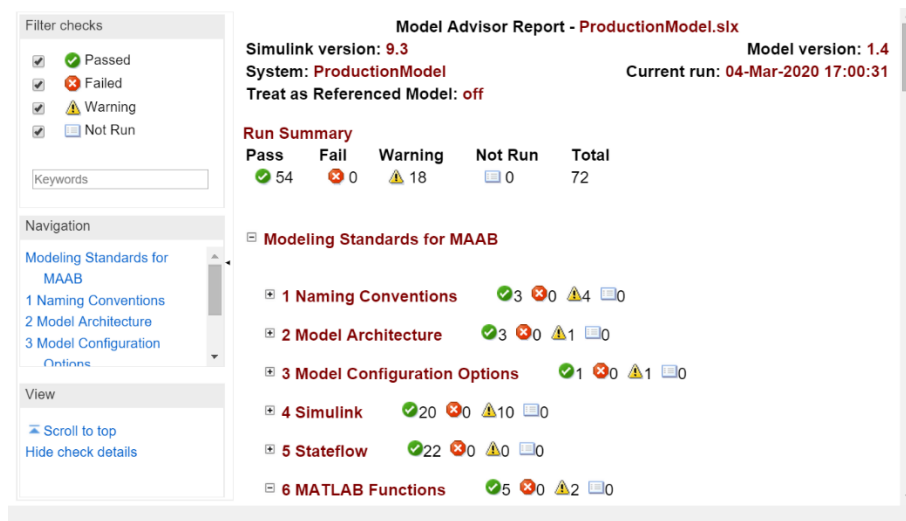


Figure 126. MAAB Guidelines check report

In addition of the standards, we run also Code Generation Advisor for:

- RAM Efficiency
- Traceability
- Safety precaution
- Debugging
- ROM Efficiency
- Execution efficiency
- MISRA C: 2012 Guidelines

The result is shown in the figure below:

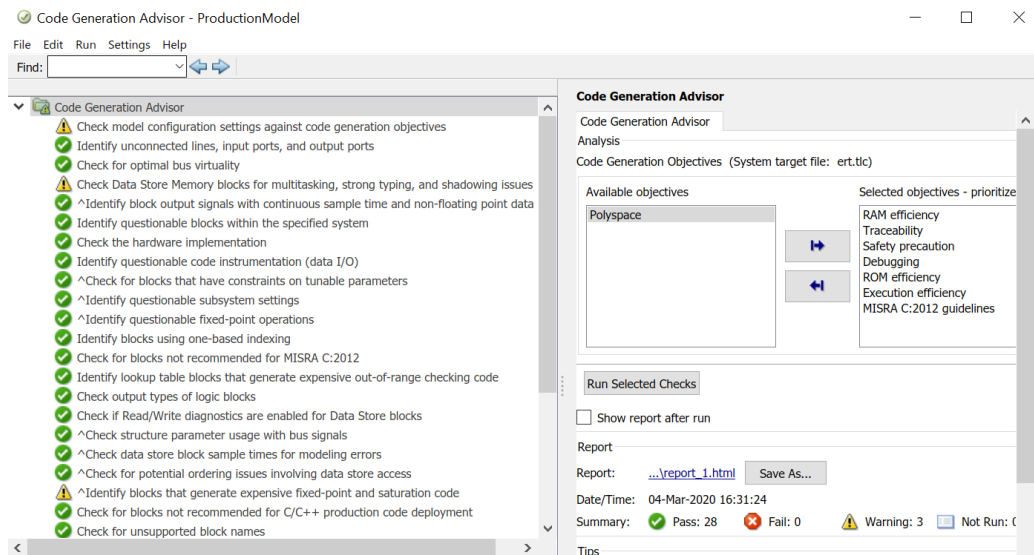


Figure 127. Code generation advisor check summary

HMI

In BAT-MAN project there is an Android app done previously by the customer ‘Brain technologies s.r.l’. It takes the data via Bluetooth from the microcontroller of BAT-MAN. The inputs that the User can give to the app are:

Car Data:

- Brand of the car (optional)
- Model of the car (optional)
- Year it was produced (optional)
- Engine type (optional):

Battery Data:

- Brand of the battery (optional)
- SN (optional)
- Age (optional)
- Capacity in Ah (necessary)

Device Found!

BATMAN 1.0

A4:DA:32:42:67:DD

Car Data

Brand _____

Model _____

Year _____

Engine _____

OK

Battery Data

Brand _____

SN _____

Age _____

Capacity _____ Ah

OK

Save as:

PATH

20191128_183749

START!

Figure 128. BAT-MAN Mobile app first page [15]

The output for now are only the sensors readings:

- Temperature of the board and the hood
- Humidity of the board and the hood
- Battery Voltage
- Current
- Shunt Voltage

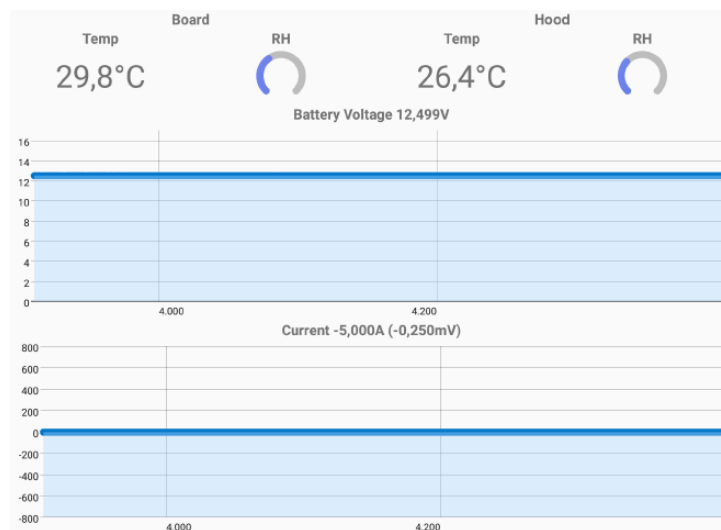


Figure 129. BATMAN App second page

The customer requirement was to replace the Temperature places with SoC and SoH, while the Temperature data must be shown in the Gauges of Humidity. Thus, the Humidity does not have

to be shown in the app. For this reason, we had to jump to Android Studio, given the source files by the customer make the needed changes.

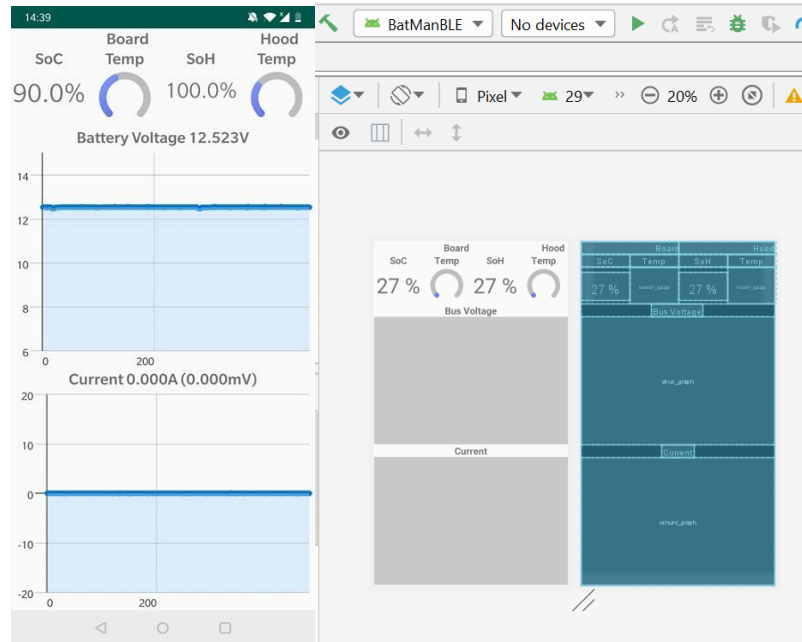


Figure 130. Updated BATMAN App

```
private void updateTH() {
    double SoH = vHoodT.value;
    double temp = (vHoodH.value*165)/Math.pow(2,16) -40; //- 40 added by me

    HoodHGauge.setValue((int) temp);
    HoodT_tw.setText(String.format("%.1f", SoH)+"%");

    SoC = vBoardT.value;
    temp = (vBoardT.value*165)/Math.pow(2,16) + 40;
    BoardHGauge.setValue((int) temp);
    BoardT_tw.setText(String.format("%.1f", SoC)+"%");
}
```

Figure 131. Android studio code modification

Suppliers

3. VMU

Supplier:

- Politecnico di Torino

The engineering of BAT-MAN PCB has been done by two students as given in our references section in collaboration with 'Brain technologies srl'. All the documentation and testing of it can be found in that thesis [15].

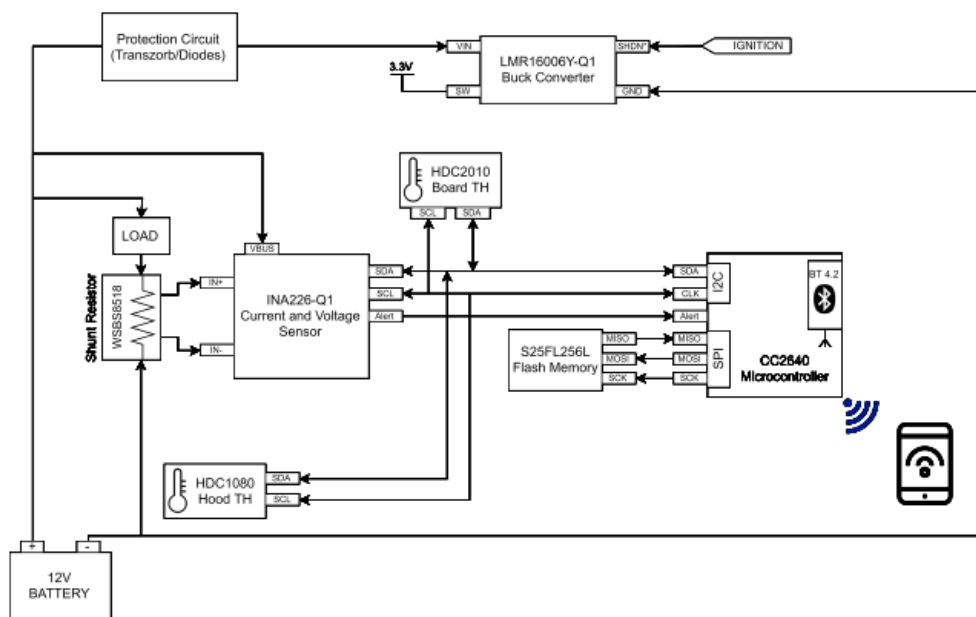


Figure 132. Hardware architecture with chosen components [15]

The previous figure shows the hardware architecture in more details. As we already said, the microcontroller as we can see is an wireless microcontroller produced by Texas Instruments. Components already chosen by the customer are:

- CC2640R2F-Q1 Microcontroller
- INA226-Q1: Amplify the input to be transmitted to CC2640R2F MCU
- LMR16006Y-Q1: Buck Converter to regulate the voltage to 3.3V to supply the hardware
- WSBM8518 Shunt resistor: Since the customer has decided to measure the current via Shunt resistor method
- Protection circuit

CC2640R2F-Q1 MCU characteristics and safety

The CC2640R2F device is a 2.4 GHz wireless microcontroller supporting Bluetooth® 5.1 Low Energy and Proprietary 2.4 GHz applications. The device is optimized for low-power wireless communication and advanced sensing in building security systems, HVAC, asset tracking, and medical markets, and applications where industrial performance is required. [18]

Some of the important features of the MCU that we must mention are:

- Arm Cortex-M3
- Up to 48-MHz clock speed
- 275KB of nonvolatile memory
- Up to 28KB system SRAM
- 8KB SRAM for cache
- JTAG Debugging
- 12bit ADC

Concerning safety this MCU has all GPIOs compliant with *RoHS* (Restriction of Hazardous Substances Directive), those packages are:

- 2.7-mm x 2.7-mm YFV DSBGA34 (14 GPIOs)
- 4-mm x 4-mm YFV DSBGA34 (10 GPIOs)
- 5-mm x 5-mm YFV DSBGA34 (15 GPIOs)
- 7-mm x 7-mm YFV DSBGA34 (31 GPIOs)

Furthermore, considering EMC, this MCU is compliant with RF (Radio frequency) regulations such as:

- ETSI EN 300 328 (Europe)
- EN 300 440 Class 2 (Europe)
- FCC CFR47 Part 15 (US)
- ARIB STD-T66 (Japan)

INA226

The INA226 is a digital current sense amplifier with an I²C- and SMBus-compatible interface. It provides digital current, voltage, and power readings necessary for accurate decision-making in precisely-controlled systems. Programmable registers allow flexible configuration for measurement resolution as well as continuous-versus triggered operation. Detailed register information appears at the end of this data sheet, beginning with Table 4. See the Functional Block Diagram section for a block diagram of the INA226 device. [18]

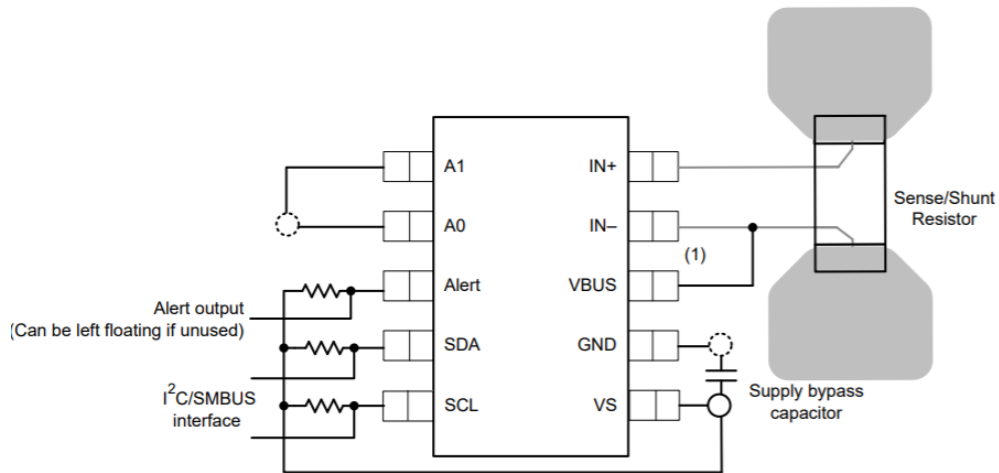


Figure 133. INA226 Layout example [18]

It is important to say that INA226-Q1 is not compliant to ISO26262 standards. But since our item is defined as a QM and does not dictate any safety requirement we can continue.

Some features of the device:

- Device is qualified by AEC-Q100
- Temperature Grade 1: -40 °C to 125 °C
- Sensing voltages from 0 to 36 V
- High accuracy, 0.1% max Gain error, 10 μ V Offset
- I2C communication
- Typical for HEV/EV Battery management application

For further information about the components chosen by the customer can be found in the documentation that they have done in the reference [15].

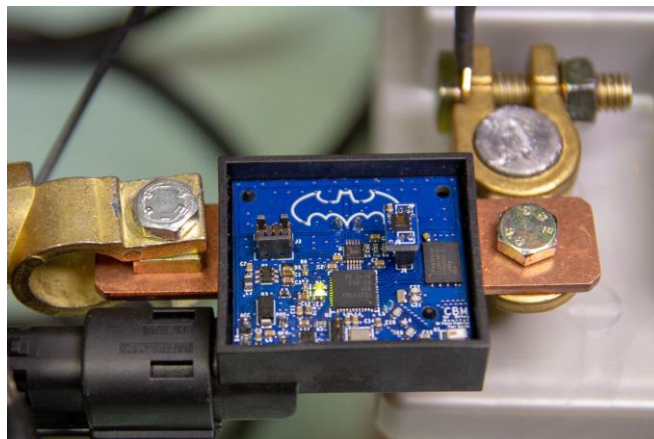


Figure 134. BAT-MAN PCB

Together with the hardware the supplier gives us also the firmware architecture done in CSS and Sensor Controller Studio. We shall integrate our application into that firmware. The firmware is based on the Texas instruments open source project called 'project_zero'. The main file has been chosen by the supplier to be 'project_zero.c'. Thus, now the job is to integrate BAT-MAN inside 'project_zero.c'.

4. Integration & Testing

Tools – Code Composer Studio

Techniques – Automatic code generation

Artefacts – SW/HW Integration

Generated code results and architecture:

3 files must be uploaded/integrated to the firmware architecture:

- ProductionModel.c
- ProductionModel.h
- rtwtypes.h

where ‘ModelName.c’, that in our case is ‘ProductionModel.c’ is the source code of the model. ‘ProductionModel.h’ is the header file that represents model-specific data types. And ‘rtwtypes.h’ is the file with hardware specific data types. There are 3 functions that we must integrate: Function for initialization, Step and function for Terminating. Function for initialization must be integrated in the first part of the main where every other function is initialized. The step function must be called in the main loop, and terminate function in the end if it is needed (usually never goes there).

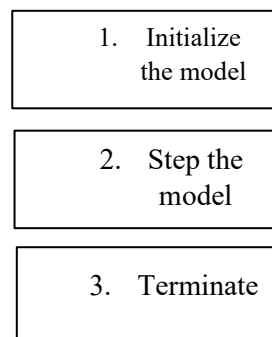


Figure 135. Generated code flowchart

Firstly, we shall upload the files ‘ProductionModel.c’, ‘ProductionModel.h’, ‘rtwtypes.h’ into the folder named ‘Applications’ which is part of the project file ‘BATMAN’. In this folder we can find all the other firmware applications such as Bluetooth module.

Secondly, in generated code we also have a file called ‘ert_main’. This file is an example main file automatically generated by Embedded coder which shows us how the main file must look like.

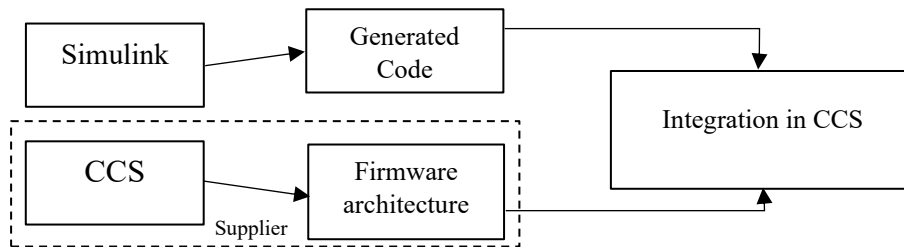


Figure 136. Illustration of integration process

```

ert_main.c
98  /* Indicate task complete */
99  OverrunFlag = false;
100
101  /* Disable interrupts here */
102  /* Restore FPU context here (if necessary) */
103  /* Enable interrupts here */
104  ~}
105
106  /*
107  * The example "main" function illustrates what is required by your
108  * application code to initialize, execute, and terminate the generated code.
109  * Attaching rt_OneStep to a real-time clock is target specific. This example
110  * illustrates how you do this relative to initializing the model.
111  */
112  int_T main(int_T argc, const char *argv[])
113  {
114      RT_MODEL_BATMANApp_T *const BATMANApp_M = BATMANApp_MPtr;
115
116      /* Unused arguments */
117      (void) argc;
118      (void) argv;
119
120      /* Pack model data into RTM */
121      BATMANApp_M->blockIO = &BATMANApp_B;
122      BATMANApp_M->dwork = &BATMANApp_DW;
123
124      /* Initialize model */
125      BATMANApp_initialize(BATMANApp_M, &BATMANApp_U_VoltageV, &BATMANApp_U_CurrentA,
126                          &BATMANApp_U_TempC, &BATMANApp_U_SampleTimeS,
127                          &BATMANApp_U_SoCInit, &BATMANApp_U_VoltageInitV,
128                          &BATMANApp_U_CapacityNomAh, &BATMANApp_Y_SoH,
129                          &BATMANApp_Y_SoU, &BATMANApp_Y_SoC);
130
131      /* Attach rt_OneStep to a timer or interrupt service routine with
132       * period 0.1 seconds (the model's base sample time) here. The
133       * call syntax for rt_OneStep is
134       *
135       * rt_OneStep(BATMANApp_M);
136       */
137      printf("Warning: The simulation will run forever. "
138            "Generated ERT main won't simulate model step behavior. "
139            "To change this behavior select the 'MAT-file logging' option.\n");
140      fflush(NULL);
141      while (rtmGetErrorStatus(BATMANApp_M) == (NULL)) {
142          /* Perform other application tasks here */
143      }
144
145      /* Disable rt_OneStep() here */
146
147      /* Terminate model */
148      BATMANApp_terminate(BATMANApp_M);
149      return 0;
150  }
151
152  /*
153  * File trailer for generated code.
154  *
155  * [EOF]
156  */
157
C source file

```

Figure 137. 'ert_main' Source file

Code integration in CCS:


```

1 //project_zero.c
2 //*****BATMANAPP Initialization*****
3
4 // RT_MODEL_BATMANAPP_T *const BATMANAPP_M = BATMANAPP_MPtr;
5
6 // Unused arguments
7 // (void)(argc);
8 // (void)(argv);
9
10 // Pack model data into RTM
11
12 BATMANAPP_M->blockIO = &BATMANAPP_B;
13 BATMANAPP_M->dwork = &BATMANAPP_DW;
14
15 // Initialize model
16
17 BATMANAPP_initialize(BATMANAPP_M, &BATMANAPP_U_VoltageV, &BATMANAPP_U_CurrentA,
18                     &BATMANAPP_U_TempC, &BATMANAPP_U_SampleTimeT,
19                     &BATMANAPP_U_SoCinit, &BATMANAPP_U_VoltageInitV,
20                     &BATMANAPP_U_NomCapacityAh, &BATMANAPP_Y_SoHOut,
21                     &BATMANAPP_Y_ReliabilityOut, &BATMANAPP_Y_SoCOut);
22
23

```

Figure 138. CCS workspace

After building, we see that it is impossible to run this software in the existing firmware and target hardware because memory occupation exceeds the resources, as the picture below shows the memory allocation before and after the integration of the code:

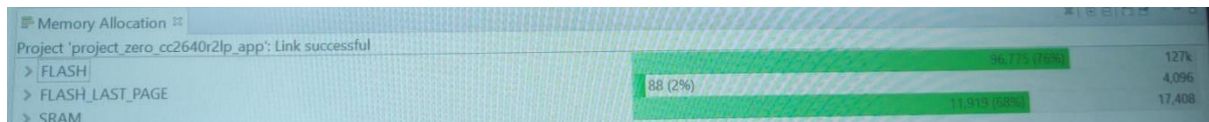
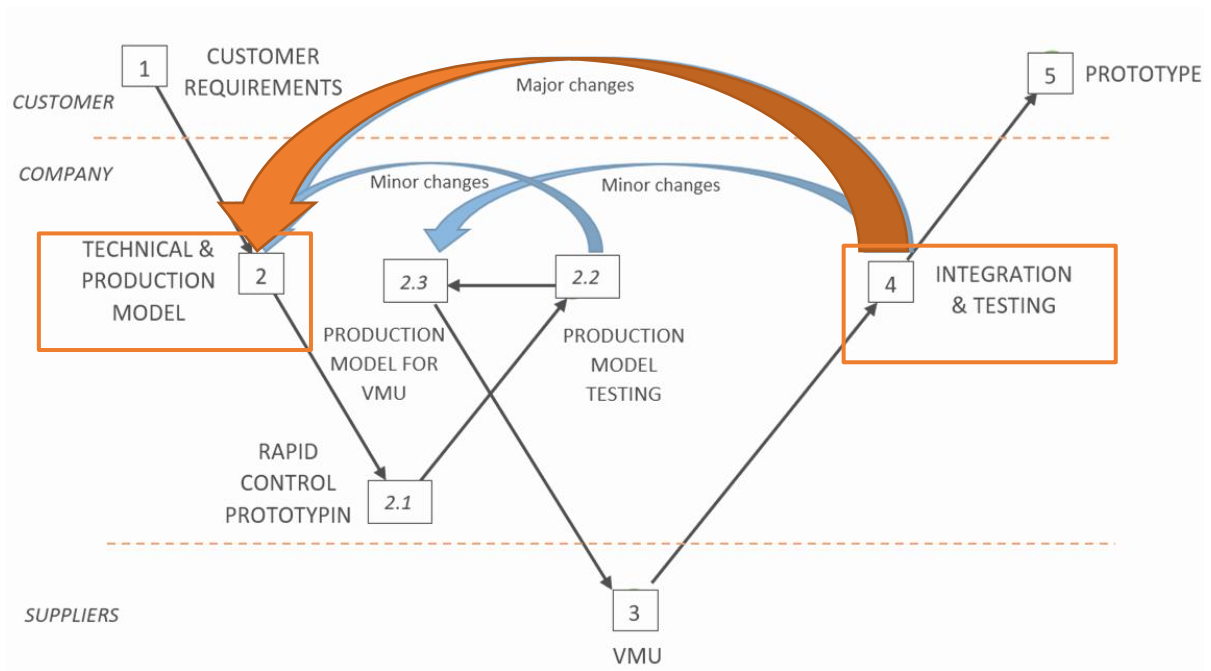


Figure 139. Memory occupation of firmware architecture



Figure 140. Memory occupation integrated

From above results we can say that there are 31kB of data that cannot be allocated in SRAM, i.e. the generated code needs more RAM than it is available. The project has failed in step 4 Integration & Testing, thus the next step is to go back to step 2 and re-design Technical Model to fit the Memory Size of the Hardware.



Conclusion

In the first part of this thesis, from doing a lot of research and taking inputs and references from industries, International Standard Organization and Academia, we have achieved to develop a toolchain, a path or a workflow that is so much needed in today's world and especially in Automotive industry. This work can be also seen as literature for students that are interested in project development and take Model-based design classes. It is not possible to find such a work as open source and for free, it is kept inside the companies and most of the advanced tools that we tried to substitute such as Documentation are too expensive even for middle sized companies.

In the second part of this work, I have applied the work done in the first part into a real project that have given us very good results such SIL results and a green light to continue the project and made sure that there is place for this project in embedded Automotive environment. Here, I dig deeper into the development toolchain since the project was real and the problems indeed were real. As good results were achieved such as passing the tests of MAAB, ISO 26262, IEC 61504 and MISRA-C were achieved, also results that made us go back into the development flow such as memory allocation insufficiency cannot be seen as bad results since this always happens in the real projects, and that is exactly why the development toolchain of the first part is crucial and absolutely necessary.

As a conclusion, we can say that this thesis has built the groundwork for 'Brain Technologies srl' to further continue its work in Automotive field with BAT-MAN project and others in the future. We have achieved solid results and opened the gates for the company to standardize the BAT-MAN project. Furthermore, this work can be used as a reference for students who are dealing with V-Cycle or Functional Safety or project in Model-based design as a reference that cannot be found otherwise in open source communities.

Appendix

Technical_model_Digital_Filter_load_file_22_11_19

%Project Name: Digital Filter

%----- HMI -----

%To start and stop running push the ON button

%

%For emergency stop switch the EMERGENCY switch

%----- Environment -----

%Noise signal is defined as amplitude 5 with frequency 400Hz
%To try different environment noise, modify the Noise Generator
block

%----- Plant -----

%Input signal (Signal1) is defined as amplitude 10 with frequency
10Hz
%To try different input signal, modify the Signal Generator block

%----- Control -----

%Digital filter is designed with sampling frequency 1000Hz
%
%Cutting frequency is 150Hz
%It is defined as function call subsystem inside Stateflow
%Designed with Simulink filter wizard with simple
blocks(gains,delays..)

%DFT is applied on the input signal (Plant) Signal1
%It is defined as function call subsystem inside Stateflow

%Supervisory Control parameters

debounceDuration = 2;

ledOn_TurnOff = 0;

ledOn_TurnOn = 1;

ledOn_Transition = 2;

```

ledEmerg_TurnOff = 0;
ledEmerg_TurnOn = 1;
ledEmerg_Transition = 2;

%Sampling time
samplingTime_supervisoryControl = 1/10000;
filteringTicks = 10;
fftTicks = 1;

%FFT Parameters
buffer_Size = 200;
buffer_Overlap = 199;
fft_Gain = 0.01;

%Digital Filter block parameters
filterCoefficients = [0.5 0.5]
initial_conditions = 0;
filterTicks = 1;

%Output in OFF and Emergency state
zero_Spectrum = zeros(buffer_Size,1);
zero_Filtering = 0;

%Control frame input

voltageRange = 25; %Voltage range
numBits = 12;      %Number of bits

%Control frame output design
%PWM
upperLimitSat = 10;
lowerLimitSat = -10;
carrierPeriod = [0 0.01];
carrierVal = [-10 10];
onPoint = 0.1;
offPoint = -0.01;
onOutput= 10;
offOutput = -10;

%LPF
passbandFreq = 15;
stopbandFreq = 80;
maxPassRipple = 0.1;
minStopAtten = 80;
%----- Simulation -----
----

%
%To see the characteristics of signals above, open Simulation data
Inspector
%
%Run simulink file here
sim('technical_model_digitalfilter_22_11_19');

```

Concept_model_Digital_Filter_load_file

```
% Digital Filter
% Concept Model: IIR & DFT

% Simulation parameters

% "Slow" sampling period
Ts_2 = 0.01; % s

% "Fast" sampling period
Ts_1 = 0.001; % s
Ws_1 = 2*pi/Ts_1 % rad/s

% Low-pass IIR
p1 = -2*pi*100 % rad/s
K = abs(p1) % unitary DC gain

H = zpke([], [p1], K)
figure, bode(H), grid

Hz = c2d(H, Ts_1, 'tustin')
zpke(Hz)

figure, bode(H, Hz), grid

[Hz_Num, Hz_Den] = tfdata(Hz, 'v')
```

Technical_model_BatmanApp – ert_main.c

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only. Not for
 * government, commercial, or other organizational use.
 *
 * File: ert_main.c
 *
 * Code generated for Simulink model 'BATMANApp'.
 *
 * Model version      : 1.3
```

```

* Simulink Coder version      : 9.1 (R2019a) 23-Nov-2018
* C/C++ source code generated on : Sun Mar 1 17:30:28 2020
*
* Target selection: ert.tlc
* Embedded hardware selection: ARM Compatible->ARM Cortex
* Code generation objectives: Unspecified
* Validation result: Not run
*/

```

```

#include <stddef.h>

#include <stdio.h>          /* This ert_main.c example uses printf/fflush */

#include "BATMANApp.h"      /* Model's header file */

#include "rtwtypes.h"

```

```

static RT_MODEL_BATMANApp_T BATMANApp_M_;

static RT_MODEL_BATMANApp_T *const BATMANApp_MPtr = &BATMANApp_M_/* Real-
time model */

static B_BATMANApp_T BATMANApp_B;    /* Observable signals */

static DW_BATMANApp_T BATMANApp_DW;  /* Observable states */

```

```

/* '<Root>/VoltageV' */

```

```

static real_T BATMANApp_U_VoltageV;

```

```

/* '<Root>/CurrentA' */

```

```

static real_T BATMANApp_U_CurrentA;

```

```

/* '<Root>/TempC' */

```

```

static real_T BATMANApp_U_TempC;

```

```

/* '<Root>/SampleTimeS' */

```

```

static real_T BATMANApp_U_SampleTimeS;

```

```

/* '<Root>/SoCInit' */

```

```

static real_T BATMANApp_U_SoCInit;

/* '<Root>/VoltageInitV' */
static real_T BATMANApp_U_VoltageInitV;

/* '<Root>/CapacityNomAh' */
static real_T BATMANApp_U_CapacityNomAh;

/* '<Root>/SoH' */
static real_T BATMANApp_Y_SoH;

/* '<Root>/SoU' */
static real_T BATMANApp_Y_SoU;

/* '<Root>/SoC' */
static real_T BATMANApp_Y_SoC;

/*
 * Associating rt_OneStep with a real-time clock or interrupt service routine
 * is what makes the generated code "real-time". The function rt_OneStep is
 * always associated with the base rate of the model. Subrates are managed
 * by the base rate from inside the generated code. Enabling/disabling
 * interrupts and floating point context switches are target specific. This
 * example code indicates where these should take place relative to executing
 * the generated code step function. Overrun behavior should be tailored to
 * your application needs. This example simply sets an error status in the
 * real-time model and returns from rt_OneStep.
 */
void rt_OneStep(RT_MODEL_BATMANApp_T *const BATMANApp_M);
void rt_OneStep(RT_MODEL_BATMANApp_T *const BATMANApp_M)
{
    static boolean_T OverrunFlag = false;

```



```
/* Disable interrupts here */
```

```
/* Check for overrun */
```

```
if (OverrunFlag) {  
    rtmSetErrorStatus(BATMANApp_M, "Overrun");  
    return;  
}
```

```
OverrunFlag = true;
```

```
/* Save FPU context here (if necessary) */
```

```
/* Re-enable timer or interrupt here */
```

```
/* Set model inputs here */
```

```
/* Step the model */
```

```
BATMANApp_step(BATMANApp_M, BATMANApp_U_VoltageV, BATMANApp_U_CurrentA,  
    BATMANApp_U_TempC, BATMANApp_U_SampleTimeS, BATMANApp_U_SoCInit,  
    BATMANApp_U_VoltageInitV, BATMANApp_U_CapacityNomAh,  
    &BATMANApp_Y_SoH, &BATMANApp_Y_SoU, &BATMANApp_Y_SoC);
```

```
/* Get model outputs here */
```

```
/* Indicate task complete */
```

```
OverrunFlag = false;
```

```
/* Disable interrupts here */
```

```
/* Restore FPU context here (if necessary) */
```

```
/* Enable interrupts here */
```

```
}
```

```
/*
```

- * The example "main" function illustrates what is required by your
- * application code to initialize, execute, and terminate the generated code.
- * Attaching rt_OneStep to a real-time clock is target specific. This example
- * illustrates how you do this relative to initializing the model.

*/

```
int_T main(int_T argc, const char *argv[])
```

```
{
```

```
    RT_MODEL_BATMANApp_T *const BATMANApp_M = BATMANApp_MPtr;
```

```
    /* Unused arguments */
```

```
    (void)(argc);
```

```
    (void)(argv);
```

```
    /* Pack model data into RTM */
```

```
    BATMANApp_M->blockIO = &BATMANApp_B;
```

```
    BATMANApp_M->dwork = &BATMANApp_DW;
```

```
    /* Initialize model */
```

```
    BATMANApp_initialize(BATMANApp_M,                                &BATMANApp_U_VoltageV,
    &BATMANApp_U_CurrentA,
```

```
        &BATMANApp_U_TempC, &BATMANApp_U_SampleTimeS,
```

```
        &BATMANApp_U_SoCInit, &BATMANApp_U_VoltageInitV,
```

```
        &BATMANApp_U_CapacityNomAh, &BATMANApp_Y_SoH,
```

```
        &BATMANApp_Y_SoU, &BATMANApp_Y_SoC);
```

```
    /* Attach rt_OneStep to a timer or interrupt service routine with
```

```
    * period 0.1 seconds (the model's base sample time) here. The
```

```
    * call syntax for rt_OneStep is
```

```
    *
```

```
    * rt_OneStep(BATMANApp_M);
```

```
    */
```

```
    printf("Warning: The simulation will run forever. "
```

```
        "Generated ERT main won't simulate model step behavior. "
```

```

        "To change this behavior select the 'MAT-file logging' option.\n");
fflush((NULL));
while (rtmGetErrorStatus(BATMANApp_M) == (NULL)) {
    /* Perform other application tasks here */
}

/* Disable rt_OneStep() here */

/* Terminate model */
BATMANApp_terminate(BATMANApp_M);
return 0;
}

/*
 * File trailer for generated code.
 *
 * [EOF]
 */

```

Project_zero.c – Initialization in Main function

```

static void ProjectZero_taskFxn( UArg a0, UArg a1 )//
{
    // Initialize application
    ProjectZero_init();
    /*    HDC2010 SENSOR CONTROLLER    */

    // Initialize the Sensor Controller
    scifOsalInit();
    scifOsalRegisterCtrlReadyCallback(scCtrlReadyCallback);
    scifOsalRegisterTaskAlertCallback(scTaskAlertCallback);
    scifInit(&scifDriverSetup);

    //*****BATMANAPP
    INITIALIZATION*****

    RT_MODEL_BATMANAPP_T *const BATMANAPP_M = BATMANAPP_MPptr;

```

```

// Unused arguments
(void)(argc);
(void)(argv);

// Pack model data into RTM

BATMANAPP_M->blockIO = &BATMANAPP_B;
BATMANAPP_M->dwork = &BATMANAPP_DW;

// Initialize model

BATMANAPP_initialize(BATMANAPP_M, &BATMANAPP_U_VoltageV,
&BATMANAPP_U_CurrentA,
&BATMANAPP_U_TempC, &BATMANAPP_U_SampleTimeT,
&BATMANAPP_U_SoCinit, &BATMANAPP_U_VoltageInitV,
&BATMANAPP_U_NomCapacityAh, &BATMANAPP_Y_SoHOut,
&BATMANAPP_Y_ReliabilityOut, &BATMANAPP_Y_SoCOut);

// Trial Values

BATMANAPP_U_SampleTimeT = 0.1;
BATMANAPP_U_SoCinit = 40;
BATMANAPP_U_VoltageInitV = 30;
BATMANAPP_U_NomCapacityAh = 50;

//*****END OF BATMANAPP INITIALIZATION*****

```


Bibliografi

- [1] Brain Technologies, «Preliminary analysis of BATMAN project,» Torino.
- [2] "Jabil," 2017. [Online]. Available: <https://www.jabil.com/insights/blog-main/automotive-industry-trends-point-to-shorter-product-development-cycles.html>.
- [3] Iso.org, "ISO online browsing platform," 2018. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en>.
- [4] N. Instruments, «white paper,» 5 3 2019. [Online]. Available: <https://www.ni.com/it-it/innovations/white-papers/11/what-is-the-iso-26262-functional-safety-standard-.html>.
- [5] cadence, «cadence,» 08 2019. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/automotive-functional-safety-wp.pdf.
- [6] Feabhas ltd, «A quick guide to ISO 26262,» Hungerford.
- [7] x-engineer.org, 2019. [Online]. Available: <https://x-engineer.org/graduate-engineering/modeling-simulation/model-based-design/essential-aspects-of-the-v-cycle-software-development-process/>.
- [8] 2 Gennaio 2020. [Online]. Available: <http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.1-eng/Dokumentation/pdf/V-Modell-XT-eng-Teil1.pdf>.
- [9] mitre.org, «mitre,» may 2014. [Online]. Available: <https://www.mitre.org/publications/systems-engineering-guide/systems-engineering-guide/the-evolution-of-systems>.
- [10] C. Jackson, «lifecycle insights,» 2019. [Online]. Available: <https://www.lifecycleinsights.com/tech-guide/model-based-development/>.
- [11] «MathWorks,» [Online]. Available: <https://www.mathworks.com/help/ecoder/gs/v-model-for-system-development.html#brufb98-7>.
- [12] dspace, 2019. [Online]. Available: https://www.dspace.com/en/ltd/home/applicationfields/our_solutions_for/bussimulation/bussimulation_usecases/rapid_control_prototyping.cfm.
- [13] H.-L. Ross, Functional safety for road vehicles, Springer International Publishing, 2016.
- [14] D. W. Adeline M. Uhrmacher, Multi-Agent systems: Simulation and Applications, 2009.
- [15] F. D. Fazio e D. Fazi, *Current and Voltage Sensing Circuit for Automotive Batteries SoH and SoC Determination*, 2019.

- [16] Rami Debouk, Jeff Joyce, «ISO26262 Hazard and Risk Assessment Methodology,» 2010.
- [17] [Online]. Available: <https://www.blueboxbatteries.co.uk/blog/how-does-temperature-affect-lead-acid-batteries-30#.XlFeKGhKg2w>].
- [18] [Online]. Available: www.ti.com.
- [19] M. Violante, «A quick guide to ISO26262».
- [20] K. S. David Smith, Functional Safety, 2004.
- [21] M. Colabella, *Identification and Predictive Analysis of Storage Systems*, 2019.
- [22] G. L. M. R. M. S. D.-I. S. R. T. Jinhao Meng.
- [23] J. Meng, L. Guangzhao , Ricco, Swierczynski, Stroe e Teodorescu, «Overview of Lithium-Ion Battery Modeling Methods for State-of-Charge Estimation in Electrical Vehicles,» *MDPI*.
- [24] Mathworks, «Real-Time Workshop Embedded Coder User's Guide».